

GOLDILOCKS 3.2 User Manual (ko)



07228 서울특별시 영등포구 영신로 220 KnK디지털타워 1109호

전화: 02-322-6288 팩스: 02-322-6788

Webpage: www.sunjesoft.com

Support: technet@sunjesoft.com

GOLDILOCKS 3.2 User Manual (ko)

SUNJESOFT Inc.

차례

차례	v
파트 I. Getting Started	1
1. 시작하기	5
1.1 서문	6
1.2 개요	7
1.3 GOLDDILOCKS Cluster의 특징	10
2. 튜토리얼	15
2.1 GOLDDILOCKS Instance 관리	16
2.2 GOLDDILOCKS 설치, 삭제와 Database 생성	22
2.3 Database 저장구조 관리	39
2.4 Schema Object 관리	46
2.5 사용자 관리	49
2.6 GOLDDILOCKS 프로퍼티	53
2.7 GOLDDILOCKS Utility	55
3. Cluster 튜토리얼	59
3.1 GOLDDILOCKS Cluster System 관리	60
3.2 GOLDDILOCKS 설치와 Database 생성	68
3.3 Schema Object 관리	72
3.4 GOLDDILOCKS 프로퍼티	82
4. What's New	85
4.1 Feature Matrix	86
4.2 What's New in GOLDDILOCKS 3.2	147
4.3 Patch Notes	156
파트 II. Administration Manual	189
5. GOLDDILOCKS 데이터베이스 관리 기본	211

5.1	GOLDILOCKS 데이터베이스 생성과 구성	212
5.2	GOLDILOCKS Instance의 시작과 종료	215
5.3	프로세스 관리	222
5.4	메모리 관리	228
5.5	모니터링	230
6.	GOLDILOCKS 데이터베이스의 구조 및 저장 구조	243
6.1	제어 파일 관리	244
6.2	Redo Log File 관리	248
6.3	Archive Redo Log File 관리	252
6.4	테이블스페이스 관리	254
6.5	데이터 파일 관리	260
7.	GOLDILOCKS 데이터베이스의 백업과 복구	263
7.1	ARCHIVELOG 모드	264
7.2	백업과 복구	266
8.	GOLDILOCKS 데이터베이스 이중화	285
8.1	이중화 소개	286
8.2	운영 방법	287
8.3	Trace Log	294
9.	Database Information	297
9.1	DICTIONARY_SCHEMA	298
9.2	INFORMATION_SCHEMA	495
9.3	PERFORMANCE_VIEW_SCHEMA	530
10.	Server Property	593
10.1	Server Property 정보	594
10.2	AGING_INTERVAL	595
10.3	AGING_PLAN_INTERVAL	596
10.4	ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10	597
10.5	ARCHIVELOG_FILE	598
10.6	ARCHIVELOG_MODE	599
10.7	BACKUP_DIR_1 ~ BACKUP_DIR_10	600
10.8	BLOCK_READ_COUNT	601
10.9	BULK_IO_PAGE_COUNT	602
10.10	CDISPATCHER_HOT_POLICY_INTERVAL	603
10.11	CDISPATCHER_SOCKET_BUFFER_SIZE	604
10.12	CDISPATCHER_SYNC_THREADS	605
10.13	CDISPATCHER_THREADS	606
10.14	CHARACTER_SET	607
10.15	CHAR_LENGTH_UNITS	608

10.16	CHECK_DEDICATE_CONNECTION_INTERVAL	610
10.17	CLIENT_MAX_COUNT	611
10.18	CLIENT_NUMA_POLICY	612
10.19	CLOSE_PSM_CHILD_STMTS	613
10.20	CLUSTER_ASYNC_COMMIT	614
10.21	CLUSTER_ASYNC_REPLICATION	615
10.22	CLUSTER_CM_BUFFER_COUNT	616
10.23	CLUSTER_CM_BUFFER_SIZE	617
10.24	CLUSTER_CM_READ_BUFFER_SIZE	618
10.25	CLUSTER_COMMIT_SLAVES	619
10.26	CLUSTER_COMMIT_STREAM_ISOLATION	620
10.27	CLUSTER_CONNECTION	621
10.28	CLUSTER_CONNECTION_TIMEOUT_SEC	622
10.29	CLUSTER_DATA_SYNC_SERVERS	623
10.30	CLUSTER_DISPATCHER_IN_QUEUE_SIZE	624
10.31	CLUSTER_DISPATCHER_NUMA_STREAM_MAP	625
10.32	CLUSTER_DISPATCHER_OUT_QUEUE_SIZE	626
10.33	CLUSTER_HEARTBEAT_INTERVAL	627
10.34	CLUSTER_HEARTBEAT_RETRY_COUNT	628
10.35	CLUSTER_IGNORE_INACTIVE_MEMBER	629
10.36	CLUSTER_MAX_PACKET_SIZE	630
10.37	CLUSTER_MAX_PAYLOAD_SIZE	631
10.38	CLUSTER_PACKET_ALLOCATION_TIMEOUT	632
10.39	CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT	633
10.40	CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT	634
10.41	CLUSTER_SERVER_RESPONSE_QUEUE_SIZE	635
10.42	CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY	636
10.43	CLUSTER_SPLIT_BRAIN_RETRY_COUNT	637
10.44	COMMITTER_HOT_POLICY_INTERVAL	638
10.45	CONTROL_FILE_0 ~ CONTROL_FILE_7	639
10.46	CONTROL_FILE_COUNT	640
10.47	CONTROL_FILE_TEMP_NAME	641
10.48	COORDINATOR_COMMIT_WRITE_MODE	642
10.49	CSERVERS	643
10.50	DATABASE_ACCESS_MODE	644
10.51	DATABASE_INSTANCE_NAME	645
10.52	DATA_STORE_MODE	646
10.53	DA_CLIENT_NUMA_NODE	647
10.54	DDL_AUTOCOMMIT	648

10.55	DDL_LOCK_TIMEOUT	649
10.56	DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION	650
10.57	DEFAULT_INDEX_LOGGING	651
10.58	DEFAULT_INDEX_PCTFREE	652
10.59	DEFAULT_INITRANS	653
10.60	DEFAULT_MAXTRANS	654
10.61	DEFAULT_PCTFREE	655
10.62	DEFAULT_PCTUSED	656
10.63	DEFAULT_REMOVAL_BACKUP_FILE	657
10.64	DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST	658
10.65	DEFAULT_SHARDING	659
10.66	DISABLE_DDL_CDC_GIVEUP	662
10.67	DISABLE_UPDATE_PK_CDC_GIVEUP	663
10.68	DISALLOWED_PROTOCOL_TARGETTYPE	664
10.69	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL	665
10.70	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME	666
10.71	DISPATCHER_CM_BUFFER_SIZE	667
10.72	DISPATCHER_CM_UNIT_SIZE	668
10.73	DISPATCHER_CONNECTIONS	669
10.74	DISPATCHER_HOT_POLICY_INTERVAL	670
10.75	DISPATCHER_LOAD_BALANCING	671
10.76	DISPATCHER_NUMA_STREAM_MAP	672
10.77	DISPATCHER_QUEUE_SIZE	673
10.78	DISPATCHER_REQUEST_MINI_QUEUE_COUNT	674
10.79	DISPATCHER_RESPONSE_MINI_QUEUE_COUNT	675
10.80	DISPATCHERS	676
10.81	FETCH_FAILOVER	677
10.82	GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY	678
10.83	GLOBAL_JOURNAL_BUFFER_SIZE	679
10.84	GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE	680
10.85	GLOBAL_PROPERTY_LOCK_TIMEOUT	681
10.86	GLOBAL_TRANSACTION_COMMIT_WRITE_MODE	682
10.87	GLOBAL_TRANSACTION_ISOLATION_SCOPE	683
10.88	GLOBAL_TRANSACTION_LOG_DIR	684
10.89	GLOBAL_TRANSACTION_LOG_FILE_SIZE	685
10.90	GMASTER_NUMA_NODE	686
10.91	GMON_AUTOSTART	687
10.92	HINT_ERROR	688
10.93	IDLE_TIMEOUT	689

10.94	INDEX_BUILD_PARALLEL_FACTOR	690
10.95	INDEX_TREE_MERGE_PARALLEL_FACTOR	691
10.96	INST_ALLOCATOR_COUNT	692
10.97	INST_TABLE_BLOCK_SIZE	693
10.98	IN_DOUBT_DECISION	694
10.99	JOURNAL_TEMP_DIR	695
10.100	KEEPALIVE_IDLE_TIME	696
10.101	LOCAL_CLUSTER_MEMBER	697
10.102	LOCAL_CLUSTER_MEMBER_HOST	698
10.103	LOCAL_CLUSTER_MEMBER_PORT	699
10.104	LOCAL_JOURNAL_BUFFER_SIZE	700
10.105	LOCATION_FILE	701
10.106	LOCATOR_QUERY_TIMEOUT	702
10.107	LOCK_HASH_TABLE_SIZE	703
10.108	LOG_BLOCK_SIZE	704
10.109	LOG_BUFFER_SIZE	705
10.110	LOG_DIR	706
10.111	LOG_FILE_SIZE	707
10.112	LOG_GROUP_COUNT	708
10.113	LOG_MIRROR_MODE	709
10.114	LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	710
10.115	LOG_MIRROR_TIMEOUT	711
10.116	LOG_SYNC_INTERVAL	712
10.117	LOG_SYNC_INTERVAL_MSEC	713
10.118	MAX_GROUP_COUNT	714
10.119	MAX_JOURNAL_FILE_SIZE	715
10.120	MAX_NODE_COUNT	716
10.121	MAXIMUM_CONCURRENT_ACTIVITIES	717
10.122	MAXIMUM_FLANGE_COUNT	718
10.123	MAXIMUM_FLUSH_LOG_BLOCK_COUNT	719
10.124	MAXIMUM_FLUSH_PAGE_COUNT	720
10.125	MAXIMUM_JOURNAL_REPLAY_COUNT	721
10.126	MAXIMUM_NAMED_CURSOR_COUNT	722
10.127	MAXIMUM_SESSION_CM_BUFFER_SIZE	724
10.128	MEASURE_CLUSTER_LATENCY	725
10.129	MEMORY_MERGE_RUN_COUNT	726
10.130	MEMORY_SORT_RUN_SIZE	727
10.131	MINIMUM_UNDO_PAGE_COUNT	728
10.132	MIN_SAMPLE_ROW_COUNT	729

10.133	NET_BUFFER_SIZE	730
10.134	NLS_DATE_FORMAT	731
10.135	NLS_TIME_FORMAT	732
10.136	NLS_TIME_WITH_TIME_ZONE_FORMAT	733
10.137	NLS_TIMESTAMP_FORMAT	734
10.138	NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT	735
10.139	NUMA	736
10.140	NUMA_MAP	737
10.141	OFFLINE_MEMBER_AFTER_FAILOVER	738
10.142	ONLINE_JOURNAL_REPLAY_THRESHOLD	739
10.143	OS_GROUP_ACCESS	740
10.144	PACKET_COMPRESSION_THRESHOLD	741
10.145	PAGE_CHECKSUM_TYPE	742
10.146	PARALLEL_IO_FACTOR	743
10.147	PARALLEL_IO_GROUP_1 ~ PARALLEL_IO_GROUP_16	744
10.148	PARALLEL_LOAD_FACTOR	745
10.149	PENDING_LOG_BUFFER_COUNT	746
10.150	PLAN_CACHE	747
10.151	PLAN_CACHE_SIZE	748
10.152	PRIVATE_STATIC_AREA_SIZE	749
10.153	PROCESS_MAX_COUNT	750
10.154	QUERY_TIMEOUT	751
10.155	READABLE_ARCHIVELOG_DIR_COUNT	752
10.156	READABLE_BACKUP_DIR_COUNT	753
10.157	REBALANCE_BLOCK_READ_COUNT	754
10.158	RECOMPILE_CHECK_MINIMUM_PAGE_COUNT	755
10.159	RECOMPILE_PAGE_PERCENT	756
10.160	RECOVERY_LOG_BUFFER_SIZE	757
10.161	REDO_LOG_COMPRESSION_THRESHOLD	758
10.162	REFINE_RELATION	759
10.163	SESSION_FATAL_BEHAVIOR	760
10.164	SESSION_MEMORY_INIT_SIZE	761
10.165	SESSION_MEMORY_SHRINK_THRESHOLD	762
10.166	SHARED_MEMORY_ADDRESS	763
10.167	SHARED_MEMORY_STATIC_KEY	764
10.168	SHARED_MEMORY_STATIC_NAME	765
10.169	SHARED_MEMORY_STATIC_SIZE	766
10.170	SHARED_REQUEST_QUEUE_COUNT	767
10.171	SHARED_SERVERS	768

10.172	SHARED_SESSION	769
10.173	SNAPSHOT_STATEMENT_TIMEOUT	770
10.174	SQL_HISTORY_SIZE	771
10.175	SQL_HISTORY_TYPE	772
10.176	SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	773
10.177	SYSTEM_LOGGER_DIR	774
10.178	SYSTEM_MEMORY_AUX_TABLESPACE_SIZE	775
10.179	SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	776
10.180	SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	777
10.181	SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	778
10.182	SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	779
10.183	SYSTEM_TABLESPACE_DIR	780
10.184	SYSTEM_UDS_DIR	781
10.185	TCP_CLIENT_NUMA_NODE	782
10.186	TCP_NODELAY	783
10.187	TEMP_SEGMENT_CACHE_SIZE	784
10.188	TEMP_UNDO_ENABLED	785
10.189	TIMED_STATISTICS	786
10.190	TIMEZONE	787
10.191	TRACE_ALTER_SYSTEM	788
10.192	TRACE_DDL	789
10.193	TRACE_LOG_ID	790
10.194	TRACE_LOG_MSGBUF_SIZE	791
10.195	TRACE_LOG_TIME_DETAIL	792
10.196	TRACE_LOGGER	793
10.197	TRACE_LOGGER_REMOTE_HOST	794
10.198	TRACE_LOGGER_REMOTE_PORT	795
10.199	TRACE_LOGIN	796
10.200	TRACE_LONG_RUN_CURSOR	797
10.201	TRACE_LONG_RUN_SQL	799
10.202	TRACE_XA	800
10.203	TRANSACTION_ALLOCATION_TIMEOUT	801
10.204	TRANSACTION_COMMIT_WRITE_MODE	802
10.205	TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT	803
10.206	TRANSACTION_TABLE_SIZE	804
10.207	TRANSACTION_TIMEOUT	806
10.208	UNDO_RELATION_ALLOCATION_TIMEOUT	807
10.209	UNDO_RELATION_COUNT	808
10.210	UNDO_SHRINK_THRESHOLD	810

10.211 USE_LARGE_PAGES	811

파트 III. SQL Manual 813

11. SQL Elements	855
11.1 Syntax Elements	856
11.2 Data Type	877
11.3 Format 문자열	903
11.4 Expressions	916
11.5 Pseudo Columns	923
11.6 Operators	927
11.7 Functions	930
11.8 Conditions	938
11.9 Built-in Data Type References	950
11.10 Built-in Function References	976
12. SQL Languages	1,189
12.1 Data Definition Language	1,190
12.2 Data Manipulation Language	1,199
12.3 Data Query Language	1,207
12.4 Control Language	1,220
12.5 Cluster의 SQL 처리	1,225
13. SQL Objects	1,273
13.1 Database	1,274
13.2 Profile	1,280
13.3 Audit Policy	1,286
13.4 Authorization	1,313
13.5 Schema	1,323
13.6 Tablespace	1,333
13.7 Table	1,336
13.8 Index	1,341
13.9 View	1,345
13.10 Sequence	1,348
13.11 Synonym	1,352
13.12 Stored Procedure	1,355
13.13 Stored Function	1,357
14. Cluster Objects	1,359
14.1 Cluster System	1,360
14.2 Cluster Group	1,367
14.3 Cluster Member	1,369

14.4	Cluster Location	1,371
14.5	Cluster Table과 Shard	1,372
14.6	Global Secondary Index	1,388
15.	SQL Tuning	1,391
15.1	SQL Tuning 개요	1,392
15.2	SQL 처리 과정	1,393
15.3	Query Optimizer	1,399
15.4	SQL 실행 계획	1,432
15.5	SQL Trace Log	1,470
16.	SQL References	1,477
16.1	ALTER AUDIT POLICY	1,478
16.2	ALTER CLUSTER GROUP name ADD MEMBER	1,481
16.3	ALTER CLUSTER GROUP name OFFLINE MEMBER	1,484
16.4	ALTER CLUSTER LOCATION	1,487
16.5	ALTER DATABASE ADD LOGFILE	1,489
16.6	ALTER DATABASE ARCHIVELOG	1,492
16.7	ALTER DATABASE BACKUP	1,494
16.8	ALTER DATABASE CLEAR AUDIT TRAIL	1,498
16.9	ALTER DATABASE CLEAR PASSWORD HISTORY	1,500
16.10	ALTER DATABASE DELETE BACKUP	1,502
16.11	ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	1,505
16.12	ALTER DATABASE DROP LOGFILE	1,507
16.13	ALTER DATABASE MOVE SHARD	1,510
16.14	ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS	1,513
16.15	ALTER DATABASE REBALANCE	1,515
16.16	ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP	1,518
16.17	ALTER DATABASE RENAME LOGFILE	1,521
16.18	ALTER DATABASE RECOVER	1,523
16.19	ALTER DATABASE REGISTER	1,528
16.20	ALTER DATABASE RESET LOCAL CLUSTER MEMBER	1,530
16.21	ALTER DATABASE RESTORE	1,532
16.22	ALTER INDEX	1,535
16.23	ALTER INDEX name AGING	1,537
16.24	ALTER INDEX name STORAGE	1,540
16.25	ALTER INDEX name RENAME TO	1,544
16.26	ALTER PROFILE	1,546
16.27	ALTER SEQUENCE	1,550
16.28	ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;	1,555
16.29	ALTER SESSION SET property_name	1,557

16.30	ALTER SYSTEM CHECKPOINT	1,559
16.31	ALTER SYSTEM CLEANUP PLAN	1,561
16.32	ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER	1,563
16.33	ALTER SYSTEM JOIN DATABASE	1,565
16.34	ALTER SYSTEM {MOUNT OPEN} DATABASE	1,568
16.35	ALTER SYSTEM [KILL DISCONNECT] SESSION	1,571
16.36	ALTER SYSTEM RECONNECT GLOBAL CONNECTION	1,574
16.37	ALTER SYSTEM RESET property_name	1,576
16.38	ALTER SYSTEM SET property_name	1,579
16.39	ALTER SYSTEM SWITCH LOGFILE	1,583
16.40	ALTER TABLE	1,585
16.41	ALTER TABLE name ADD COLUMN	1,589
16.42	ALTER TABLE name SET UNUSED COLUMN	1,593
16.43	ALTER TABLE name ALTER COLUMN	1,596
16.44	ALTER TABLE name RENAME COLUMN	1,608
16.45	ALTER TABLE name ADD CONSTRAINT	1,610
16.46	ALTER TABLE name DROP CONSTRAINT	1,613
16.47	ALTER TABLE name ALTER CONSTRAINT	1,616
16.48	ALTER TABLE name RENAME CONSTRAINT	1,619
16.49	ALTER TABLE name ADD GLOBAL SECONDARY INDEX	1,622
16.50	ALTER TABLE name DROP GLOBAL SECONDARY INDEX	1,628
16.51	ALTER TABLE name ALTER GLOBAL SECONDARY INDEX	1,630
16.52	ALTER TABLE name MOVE SHARD	1,634
16.53	ALTER TABLE name REBALANCE	1,637
16.54	ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list	1,639
16.55	ALTER TABLE name SPLIT SHARD	1,641
16.56	ALTER TABLE name RENAME SHARD	1,645
16.57	ALTER TABLE name READ { ONLY WRITE }	1,648
16.58	ALTER TABLE name RENAME TO	1,651
16.59	ALTER TABLE name STORAGE	1,653
16.60	ALTER TABLE name ADD SUPPLEMENTAL LOG	1,656
16.61	ALTER TABLE name DROP SUPPLEMENTAL LOG	1,658
16.62	ALTER TABLESPACE	1,660
16.63	ALTER TABLESPACE name RENAME TO	1,663
16.64	ALTER TABLESPACE name BACKUP	1,665
16.65	ALTER TABLESPACE name [ONLINE OFFLINE]	1,668
16.66	ALTER TABLESPACE name ADD [DATAFILE MEMORY]	1,671
16.67	ALTER TABLESPACE name DROP [DATAFILE MEMORY]	1,674
16.68	ALTER TABLESPACE name RENAME DATAFILE	1,677

16.69	ALTER USER	1,680
16.70	ALTER VIEW	1,686
16.71	ANALYZE SYSTEM	1,689
16.72	ANALYZE TABLE	1,691
16.73	AUDIT POLICY	1,698
16.74	CLOSE cursor_name	1,704
16.75	COMMENT ON name IS	1,706
16.76	COMMIT	1,711
16.77	CREATE AUDIT POLICY	1,714
16.78	CREATE CLUSTER GROUP	1,721
16.79	CREATE CLUSTER LOCATION	1,724
16.80	CREATE INDEX	1,726
16.81	CREATE PROFILE	1,734
16.82	CREATE SCHEMA	1,745
16.83	CREATE SEQUENCE	1,750
16.84	CREATE SYNONYM	1,756
16.85	CREATE TABLE	1,760
16.86	CREATE TABLE AS SELECT	1,795
16.87	CREATE GLOBAL TEMPORARY TABLE	1,801
16.88	CREATE TABLESPACE	1,806
16.89	CREATE MEMORY DATA TABLESPACE	1,808
16.90	CREATE MEMORY TEMPORARY TABLESPACE	1,812
16.91	CREATE USER	1,816
16.92	CREATE VIEW	1,822
16.93	DECLARE cursor_name	1,827
16.94	DELETE FROM	1,840
16.95	DELETE FROM name RETURNING	1,844
16.96	DELETE FROM name RETURNING .. INTO	1,848
16.97	DELETE FROM name WHERE CURRENT OF cursor_name	1,852
16.98	DROP AUDIT POLICY	1,856
16.99	DROP CLUSTER GROUP	1,858
16.100	DROP CLUSTER LOCATION	1,860
16.101	DROP INDEX	1,862
16.102	DROP PROFILE	1,864
16.103	DROP SCHEMA	1,866
16.104	DROP SEQUENCE	1,869
16.105	DROP SYNONYM	1,871
16.106	DROP TABLE	1,873
16.107	DROP TABLESPACE	1,877

16.108	DROP USER	1,880
16.109	DROP VIEW	1,884
16.110	EXECUTE statement_name	1,886
16.111	EXECUTE IMMEDIATE 'sql_string'	1,891
16.112	FETCH cursor_name	1,894
16.113	GRANT privileges TO	1,900
16.114	INSERT INTO	1,912
16.115	INSERT INTO name RETURNING	1,917
16.116	INSERT INTO name RETURNING .. INTO	1,921
16.117	LOCK TABLE	1,925
16.118	NOAUDIT POLICY	1,929
16.119	OPEN cursor_name	1,934
16.120	PREPARE statement_name	1,938
16.121	RELEASE SAVEPOINT savepoint_specifier	1,943
16.122	REVOKE privileges FROM	1,945
16.123	ROLLBACK	1,949
16.124	SAVEPOINT savepoint_specifier	1,953
16.125	SELECT	1,956
16.126	SELECT .. FOR UPDATE	2,037
16.127	SELECT .. INTO	2,042
16.128	SELECT .. INTO .. FOR UPDATE	2,045
16.129	SET CONSTRAINTS	2,051
16.130	SET SESSION AUTHORIZATION user_identifier	2,060
16.131	SET SESSION CHARACTERISTICS AS transaction_mode	2,062
16.132	SET TIME ZONE	2,064
16.133	SET TRANSACTION transaction_mode	2,066
16.134	TRUNCATE TABLE	2,068
16.135	UPDATE	2,071
16.136	UPDATE name RETURNING	2,076
16.137	UPDATE name RETURNING .. INTO	2,080
16.138	UPDATE name WHERE CURRENT OF cursor_name	2,084

파트 IV. PSM Manual **2,089**

17.	Overview of PSM	2,105
17.1	PSM의 특징점	2,106
17.2	Language Elements	2,108
17.3	Processing Transaction in PSM	2,109
18.	PSM DataTypes	2,111

18.1	Built-in Data Types	2,112
18.2	Attribute Data Types	2,115
18.3	User-defined Record Type	2,117
18.4	User-defined Collection Type	2,119
18.5	SYS_REFCURSOR	2,127
19.	PSM Control Statements	2,129
19.1	Assignment	2,130
19.2	PL Block	2,134
19.3	NULL Statement	2,137
19.4	Testing Conditions	2,138
19.5	Iterative Control	2,141
19.6	Sequential Control	2,144
19.7	Error Handling	2,150
20.	PSM Cursor Statements	2,165
20.1	Declaration	2,166
20.2	OPEN	2,170
20.3	FETCH	2,171
20.4	CLOSE	2,173
20.5	EXPLICIT CURSOR ATTRIBUTES	2,174
20.6	IMPLICIT_CURSOR_ATTRIBUTES	2,176
20.7	CURSOR VARIABLES	2,178
21.	Using PSM Subprograms	2,185
21.1	Anonymous PL Block	2,186
21.2	Nested Procedure	2,187
21.3	Nested Function	2,189
21.4	Schema-level Procedure	2,190
21.5	Schema-level Function	2,194
21.6	Built-in Procedures	2,197
22.	Using SQLs in PSM	2,199
22.1	Static SQLs	2,200
22.2	Dynamic SQL	2,217
23.	PSM Language Element References	2,221
23.1	Assignment Statement	2,222
23.2	Basic LOOP Statement	2,226
23.3	Block (BEGIN .. END)	2,228
23.4	CASE Statement	2,231
23.5	CLOSE Statement	2,234
23.6	Collection Method Invocation	2,236

23.7	COLLECTION Variable Declaration	2,240
23.8	CONTINUE Statement	2,244
23.9	Cursor FOR LOOP Statement	2,247
23.10	Cursor Variable Declaration	2,251
23.11	DELETE Statement Extension	2,253
23.12	EXCEPTION_INIT Pragma	2,256
23.13	Exception Declaration	2,258
23.14	Exception Handler	2,260
23.15	EXECUTE IMMEDIATE Statement	2,263
23.16	EXIT Statement	2,267
23.17	Explicit Cursor Attribute	2,269
23.18	Explicit Cursor Declaration and Definition	2,272
23.19	FETCH Statement	2,276
23.20	FOR LOOP Statement	2,279
23.21	Function Declaration and Definition	2,282
23.22	GOTO Statement	2,285
23.23	IF Statement	2,287
23.24	Implicit Cursor Attribute	2,289
23.25	INSERT Statement Extension	2,291
23.26	NULL Statement	2,294
23.27	OPEN Statement	2,296
23.28	OPEN FOR Statement	2,299
23.29	Procedure Call	2,302
23.30	Procedure Declaration and Definition	2,304
23.31	RAISE Statement	2,306
23.32	Record Variable Declaration	2,309
23.33	RETURN Statement	2,311
23.34	RETURNING INTO clause	2,313
23.35	%ROWTYPE Attribute	2,315
23.36	Scalar Variable Declaration	2,317
23.37	SELECT INTO Statement	2,320
23.38	SQLCODE Function	2,322
23.39	SQLERRM Function	2,324
23.40	%TYPE Attribute	2,326
23.41	UPDATE Statement Extension	2,328
23.42	WHILE LOOP Statement	2,331
24.	PSM SQL References	2,333
24.1	ALTER FUNCTION	2,334
24.2	ALTER PROCEDURE	2,336

24.3	CALL Statement	2,339
24.4	CREATE FUNCTION	2,342
24.5	CREATE PROCEDURE	2,347
24.6	DROP FUNCTION	2,352
24.7	DROP PROCEDURE	2,355

파트 V. Developer Manual 2,359

25.	ODBC	2,367
25.1	GOLDILOCKS ODBC Driver 개요	2,368
25.2	데이터 원본 구성	2,371
25.3	GLOBAL CONNECTION	2,380
25.4	카탈로그 함수	2,385
25.5	ODBC API References	2,390
25.6	XA API References	2,719
26.	JDBC	2,737
26.1	GOLDILOCKS JDBC 개요	2,738
26.2	기능 명세	2,743
26.3	JDBC API References	2,765
27.	Embedded SQL	2,953
27.1	Precompiler	2,954
27.2	Embedded SQL	2,964
27.3	Advanced Topic	3,077
27.4	Embedded SQL Reference	3,117
28.	PDO	3,143
28.1	PDO 개요	3,144
28.2	설치/구성	3,144
28.3	사용	3,146
28.4	사용 예	3,147
29.	PyDBC	3,157
29.1	GOLDILOCKS PyDBC	3,158
29.2	API Reference	3,161
29.3	Exception	3,176
29.4	Data Type	3,177
30.	Hibernate	3,181
30.1	개요	3,182
30.2	연동	3,182
30.3	사용 예	3,183

파트 VI. Utility Manual	3,191
31. gcreatedb	3,199
31.1 gcreatedb 소개	3,200
31.2 Command Option	3,202
32. glsnr	3,207
32.1 glsnr 소개	3,208
32.2 Command Option	3,209
32.3 Listener Configuration	3,212
33. gsql/gsqlnet (Interactive SQL Tool)	3,217
33.1 gsql 개요	3,218
33.2 gsql 실행	3,222
33.3 Interactive Command 사용	3,228
33.4 Command Option References	3,265
33.5 Interactive Command References	3,273
34. gloder/glodernet (Upload/download Tool)	3,351
34.1 gloder 개요	3,352
34.2 gloder 사용	3,356
34.3 Control File 구문	3,379
34.4 gloder Argument References	3,386
35. gdump	3,399
35.1 gdump 개요	3,400
35.2 사용 예	3,403
36. tablediff	3,417
36.1 tablediff 개요	3,418
36.2 사용법	3,420
37. gsyncher	3,427
37.1 gsyncher 소개	3,428
37.2 사용 예	3,429
38. gmon	3,433
38.1 gmon 소개	3,434
38.2 사용 예	3,435
39. gtrclogger	3,437
39.1 gtrclogger 소개	3,438
39.2 사용 예	3,440
40. glocator	3,441
40.1 glocator 소개	3,442
40.2 glocator 사용	3,447

40.3	기능	3,450
40.4	glocator Configuration	3,452
41.	gagent	3,459
41.1	gagent 소개	3,460
41.2	gagent Configuration	3,464
42.	gloctl	3,469
42.1	gloctl 소개	3,470
42.2	Interactive Command Reference	3,474
42.3	Location File	3,481
42.4	Configuration	3,483
파트 VII.	Replication	3,485
43.	개요	3,489
43.1	GOLDILOCKS Replication 소개	3,490
43.2	특징	3,491
44.	CYCLONE	3,493
44.1	CYCLONE	3,494
44.2	준비사항	3,499
44.3	환경설정	3,504
44.4	운영하기	3,522
44.5	CYCLONE 운영 예	3,527
44.6	Cluster 환경에서 CYCLONE 운영	3,534
44.7	모니터링 (CYMON)	3,539
45.	LOGMIRROR	3,545
45.1	LOGMIRROR	3,546
45.2	준비사항	3,547
45.3	환경설정	3,549
45.4	운영	3,553
45.5	CYCLONE과 연동 예	3,556
부록 A.	Error Codes	3,563
A.1	OS 관련 에러	3,564
A.2	타입과 연산 관련 에러	3,568
A.3	자원 관리 관련 에러	3,574
A.4	저장 관리 관련 에러	3,576
A.5	딕셔너리 캐시 관련 에러	3,580
A.6	SQL 구문 처리 관련 에러	3,581
A.7	PSM 관련 에러	3,595

A.8	세션 관련 에러	3,598
A.9	ODBC 관련 에러	3,599
A.10	JDBC 관련 에러	3,601
A.11	Embedded SQL 관련 에러	3,603
A.12	Communication 관련 에러	3,605
A.13	ServerLibrary 관련 에러	3,606
A.14	gsq/ gsqlnet 관련 에러	3,607
A.15	gloader/ gloadernet 관련 에러	3,609
A.16	gmaster 관련 에러	3,610
A.17	glsnr 관련 에러	3,611
A.18	cyclone 관련 에러	3,612
A.19	LogMirror 관련 에러	3,614
A.20	cymon 관련 에러	3,615
A.21	gdispatcher 관련 에러	3,616
A.22	gbalancer 관련 에러	3,617
A.23	gsyncher 관련 에러	3,618
A.24	Cluster 관련 에러	3,619
A.25	cserver 관련 에러	3,620
A.26	cdispatcher 관련 에러	3,621
A.27	gtrclogger 관련 에러	3,622
A.28	glocator 관련 에러	3,623
A.29	gloctl 관련 에러	3,624
A.30	gmon 관련 에러	3,625
A.31	gagent 관련 에러	3,626
부록 B.	Wait Event	3,627
B.1	Wait Event	3,628
B.2	Wait Event의 Class	3,628
B.3	Wait Event 항목	3,629
부록 C.	Open Source License	3,647

파트 I.

Getting Started

1. 시작하기	5
1.1 서문	6
대상	6
1.2 개요	7
GOLDILOCKS Database Management System	7
GOLDILOCKS Architecture	7
GOLDILOCKS Cluster System Architecture	9
1.3 GOLDILOCKS Cluster의 특징	10
Cluster 특징점	10
Constraint of Cluster	11
2. 튜토리얼	15
2.1 GOLDILOCKS Instance 관리	16
Overview	16
프로퍼티 설정	16
Background 프로세스	17
Client 프로세스	18
Instance의 메모리 구조	19
Instance 시작과 종료	19
Listener 시작과 종료	21
2.2 GOLDILOCKS 설치, 삭제와 Database 생성	22
Overview	22
릴리즈 플랫폼	22
시스템 요구사항	23
GOLDILOCKS Package 구성	23
GOLDILOCKS Software 설치	29
Database 생성	34
Dictionary Schema 정보 구축	37
2.3 Database 저장구조 관리	39
Database 저장구조	39
Database 저장 구조 정보 확인	41
일반적인 데이터 저장소 작업	42
Store Mode	45
2.4 Schema Object 관리	46
Schema Object	46
Schema Object 관리 권한	46
테이블 관리	46
인덱스 관리	47
시퀀스	48
2.5 사용자 관리	49

사용자 생성	49
사용자 제거	50
사용자 변경	50
2.6 GOLDILOCKS 프로퍼티	53
데이터베이스 생성 시 적용되는 프로퍼티	53
데이터베이스 기동 시 적용되는 프로퍼티	53
2.7 GOLDILOCKS Utility	55
gcreatedb	55
gsql (GOLDILOCKS interactive sql tool)	56
gloader (GOLDILOCKS data upload/ download tool)	57
3. Cluster 튜토리얼	59
3.1 GOLDILOCKS Cluster System 관리	60
Overview	60
프로퍼티 설정	60
Background 프로세스	61
Client 프로세스	62
Instance의 메모리 구조	62
Cluster System 시작과 종료	63
3.2 GOLDILOCKS 설치와 Database 생성	68
GOLDILOCKS Package 구성	68
GOLDILOCKS Software 설치	68
Database 생성	69
Dictionary Schema 정보 구축	70
3.3 Schema Object 관리	72
테이블 관리	72
인덱스 관리	78
Global 시퀀스	79
3.4 GOLDILOCKS 프로퍼티	82
4. What's New	85
4.1 Feature Matrix	86
Architecture	86
SQL	104
API	119
Utility	129
Replication	143
4.2 What's New in GOLDILOCKS 3.2	147
Architecture	147
SQL	149
API	151

Utility	152
Replication	154
4.3 Patch Notes	156
3.2.14 Patch Note	156
3.2.13 Patch Note	156
3.2.12 Patch Note	156
3.2.11 Patch Note	157
3.2.10 Patch Note	162
3.2.9 Patch Note	170
3.2.8 Patch Note	170
3.2.7 Patch Note	171
3.2.6 Patch Note	172
3.2.5 Patch Note	174
3.2.4 Patch Note	178
3.2.3 Patch Note	181
3.2.2 Patch Note	182
3.2.1 Patch Note	188

1.

시작하기

1.1 서문

본 문서는 GOLDILOCKS를 구성, 관리, 사용하는 운영자를 위하여 제작된 가이드이다. 이 문서의 목적은 GOLDILOCKS를 설치하고 관리할 때 알아두어야 할 기본적인 개념을 전달하는 것이다. 또한 GOLDILOCKS 시스템을 사용할 때 주의하여야 할 기본적인 관리 포인트들을 알려준다.

- 본 문서에서 제시되는 설명은 절대적이지 않으며 설치되는 환경 및 사용 방법에 따라 변경될 수 있다.
- 본 문서는 GOLDILOCKS 3.2 버전을 기준으로 작성되었다.
- 본 문서는 RedHat 계열 Linux 플랫폼을 기준으로 작성되었다.

대상

본 문서의 대상은 다음과 같다.

- GOLDILOCKS database에서 프로그램을 개발하면서 기본적인 관리 방법에 관한 지식이 필요한 자
- GOLDILOCKS database의 운용자 및 성능 관리자
- GOLDILOCKS cluster system의 운용자 및 성능 관리자

1.2 개요

본 장에서는 GOLDDILOCKS를 처음 접하는 사용자들을 위해서 기본적인 GOLDDILOCKS의 구조 및 특징에 대해 설명한다. GOLDDILOCKS는 standalone 또는 cluster system architecture 중 하나를 선택하여 사용할 수 있는데, 각각의 architecture 및 사용법에 대한 차이점을 이해할 수 있도록 설명하였다.

GOLDDILOCKS Database Management System

GOLDDILOCKS는 사용자의 데이터를 모두 in-memory에 있는 테이블에 저장하고, 이에 대한 검색 및 갱신 연산을 효과적으로 수행하는 In-memory relational DBMS 이다.

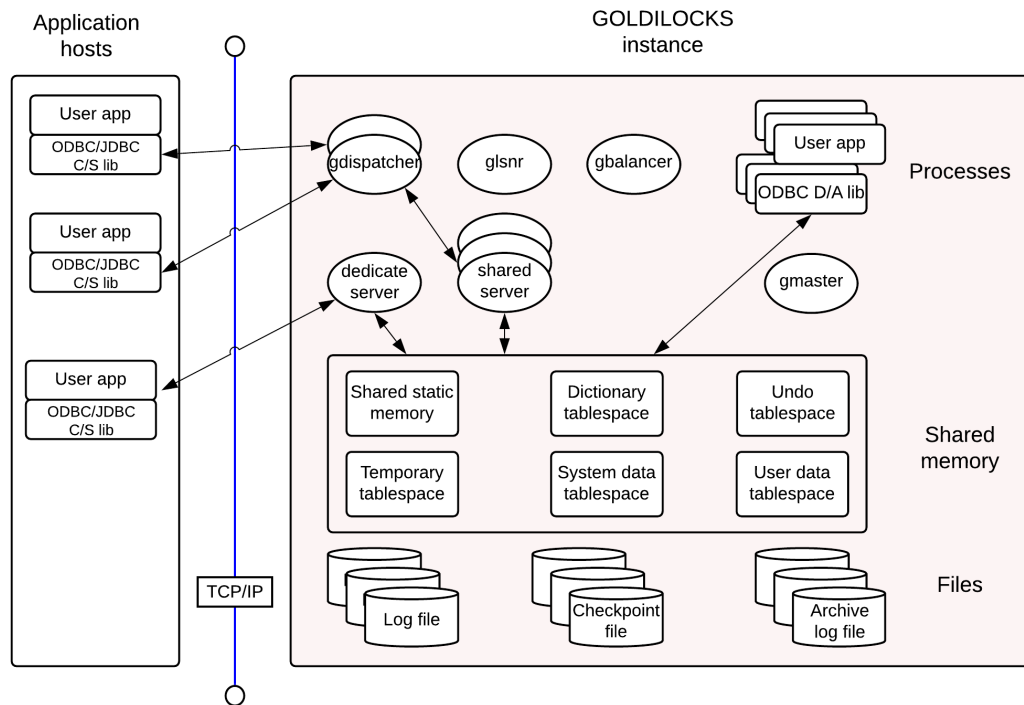
GOLDDILOCKS database 시스템은 다음과 같이 구성된다.

- 사용자가 install한 GOLDDILOCKS 소프트웨어 바이너리들
- In-memory 상에 한 개 이상의 shared memory들로 구현된 tablespace의 집합인 database
- Database의 영속성을 지원하기 위한 각종 파일들
 - 각 shared memory당 한 개씩 shared memory와 동일한 크기로 생성되는 data file들
 - 장애가 발생했을 때 database의 회복을 지원하는 redo log file들
 - Database 설정에 사용되는 각종 configuration을 위한 파일들
 - 그 외 database 운영 중에 발생하는 event 등의 내용이 기록되는 trace log 파일들
- In-memory 상의 database를 관리하는 gmaster 프로세스와 그 내부의 여러 system thread들

GOLDDILOCKS Architecture

GOLDDILOCKS database는 특정 응용 프로그램에서 발생한 오류가 전체 database 시스템으로 확산되지 않도록 하기 위해 multi-thread 구조가 아닌 shared memory 기반의 multi-process 구조로 되어 있다. GOLDDILOCKS database의 전체적인 구조는 [그림-1]과 같다. Shared memory 상에 데이터들이 적재되며 관리 데몬인 gmaster 프로세스는 boot-up, log flush, aging과 같은 전반적인 데이터베이스 관리 작업을 한다. 또한 디스크 파일에 redo log file 및 data file들을 저장하여 데이터들의 영속성을 보장한다. GOLDDILOCKS database를 사용하는 응용 프로그램은 다음 두 가지 접근 모델 중 하나를 사용한다.

그림 1 GOLDILOCKS architecture



- Direct access (D/A) model

- 사용자 응용 프로그램이 GOLDILOCKS database와 동일한 장비에서 운영될 때 사용될 수 있다.
- 사용자 응용 프로그램은 D/A 용 GOLDILOCKS ODBC/ JDBC 라이브러리를 link하여 사용해야 한다.
- D/A용 GOLDILOCKS ODBC/ JDBC 라이브러리는 내부에 질의 처리 및 저장관리 모듈을 포함하고 있어 해당 데이터베이스를 구성하는 shared memory를 직접 attach하여 사용자의 요청을 처리한다.
- 응용 프로그램 프로세스와 database 처리모듈 간에 통신 부하를 제거할 수 있어 소수의 low-latency를 필요로 하는 업무 구현에 적합하다.

- Client/ Server (C/S) model

- 사용자 응용 프로그램이 GOLDILOCKS database와 동일한 장비 또는 다른 장비에서 운영될 때 사용된다.
- 사용자 응용 프로그램은 C/S용 GOLDILOCKS ODBC/ JDBC 라이브러리를 link하여 사용해야 한다.
- C/S용 GOLDILOCKS 개발 라이브러리는 connect string에 명시된 데이터베이스를 서비스하는 server(gserver)와의 TCP 통신을 통해 사용자의 요청을 처리한다.
- 단일 응용 프로그램의 응답 속도는 D/A 모델에 비해 떨어지지만 응용 프로그램 위치에 의존적이지 않고 응용 프로그램에 장애가 발생하더라도 비교적 안정적으로 운영할 수 있다.
- C/S model에는 dedicated 모드와 shared 모드라는 두 가지 모드가 있다. Dedicated 모드는 하나의 client에 대해 하나의 server (gserver) process가 실행되는 구조이고 shared 모드는 dispatcher (gdispatcher)와 shared-server (gserver)가 항상 실행되어 있어 다수의 client에 대응하는 방식이다.
- Dedicated 모드는 data 양이 많은 업무에 적합하며 shared 모드는 client 수가 많고 data 양이 많지 않은 업무에 적합하다.
- Dedicated, shared 모드 설정은 **odbc.ini 파일, Listener Configuration**을 참조한다.

노트

D/A 모델에서는 응용 프로그램이 직접 database에 접근하여 조작하기 때문에 초기 개발단계에서 발생하는 많은 문제들로 인해 database instance가 불안정해 질 수 있으므로, 초기 개발단계에서는 C/S 모델로 개발한 후에 마무리 단계에서 D/A 모델로 전환하는 것이 전체적으로 효율적일 수 있다.

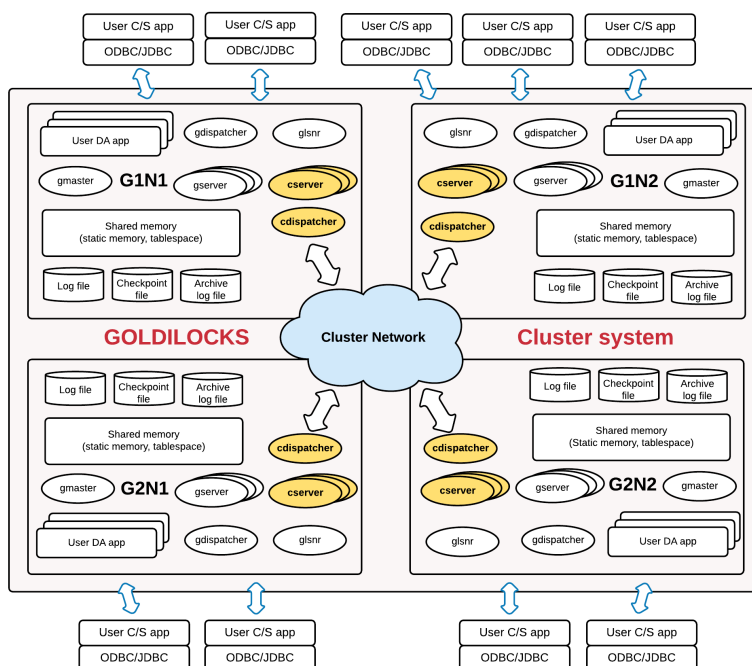
GOLDILOCKS Cluster System Architecture

GOLDILOCKS는 단독 (standalone) database로 구성하여 사용할 수 있을 뿐만 아니라 다수의 database를 클러스터 (cluster)라는 하나의 관리 단위로 묶어서 사용할 수도 있다. Cluster system으로 구성하여 사용할 경우, 사용자가 원하는 sharding 정책에 따라 테이블 data를 다수의 노드에 분배하여 저장할 수 있기 때문에 서비스 가용성이 높아지고 병렬 처리에 따른 많은 장점을 누릴 수 있다.

GOLDILOCKS cluster system은 cluster wide하게 수행되는 트랜잭션의 ACID를 보장하여 cluster system에 소속된 어떤 노드에 접속하여 트랜잭션을 수행하더라도 standalone 서버에서 수행하는 트랜잭션과 같은 데이터 신뢰성을 제공할 수 있다.

GOLDILOCKS cluster system에 포함되는 각각의 database는 multi-process 구조 및 데이터 적재 방식 등에 있어서 대부분 standalone database와 동일한 architecture를 가진다. 다만 cluster 내의 member 노드 간 효율적인 통신을 위한 프로세스 (cdispatcher)와 각 cluster member 노드 내에서 데이터의 저장 및 관리를 위한 cluster server (cserver) 프로세스가 추가되었고, 공유 메모리에는 cluster system의 트랜잭션 관리를 위한 테이블스페이스 및 관리 영역들이 추가되었다.

그림 2 GOLDILOCKS cluster system architecture



1.3 GOLDDILOCKS Cluster의 특징

Cluster 특징점

GOLDDILOCKS cluster는 기존 standalone 시스템이 갖는 트랜잭션 성능의 한계 및 저장 공간의 한계를 극복할 수 있는 shared nothing 구조의 cluster system이다.

- 고성능 (High throughput)
 - GOLDDILOCKS cluster는 제한없이 group을 추가할 수 있으며 group 추가에 따른 선형적 성능 향상이 가능하다.
 - 기존 메모리 기반 standalone 시스템이 갖는 저장 공간의 한계를 극복할 수 있다.
- 고가용성 (High availability)
 - Group이 다수의 member로 구성되었을 때 group 내에서 적어도 한 개의 member가 운용되고 있다면 가용성에는 영향을 미치지 않는다.
 - Group 내의 모든 member가 가용상태가 아닌 경우에도 그 group을 제외한 나머지 group은 정상적인 서비스를 제공한다.
- 온라인 확장 및 온라인 복구
 - 서비스가 진행 중인 상태에서도 group이나 member를 추가할 수 있는데 이는 진행 중인 서비스에 영향을 미치지 않는다.
 - 장애로 인해 서비스가 중단된 member도 온라인 상태에서 cluster에 참여할 수 있다.
- 완벽한 트랜잭션 제공
 - 트랜잭션이 준수해야 하는 다음 속성을 완벽하게 제공한다.
 - 원자성 (Atomicity)
 - 일관성 (Consistency)
 - 독립성 (Isolation)
 - 지속성 (Durability)
- 완벽한 MVCC (multi-version concurrency control) 제공
 - GOLDDILOCKS cluster는 standalone과 마찬가지로 global statement level consistency를 제공한다.
 - 특정 시점에 시작한 SQL이 어떠한 노드에 접근하든지 다수 개의 버전 중 원하는 시점의 버전에 접근할 수 있다.
- 표준 SQL 및 표준 DBC 제공
 - SQL 92에 준하는 SQL을 제공한다.
 - JDBC/ ODBC등 각종 표준 DBC를 제공한다.
- 응용 프로그램 호환성
 - 기존 standalone에서 개발된 응용 프로그램 소스나 SQL을 수정하지 않고 그대로 GOLDDILOCKS cluster에 사용할 수 있다.

Constraint of Cluster

GOLDILOCKS cluster는 다음 제약 조건을 제외하고 모든 SQL 구문을 standalone과 동일하게 사용할 수 있다.

노트

Sharded table에 대한 PRIMARY KEY, UNIQUE 제약 조건, UNIQUE INDEX는 sharding key를 포함해야 한다.

다음은 UNIQUE(name) 제약 조건이 sharding key인 id column을 포함하지 않아 실패한 예이다.

```
gSQL>
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name VARCHAR(128),
    UNIQUE (name)
)
SHARDING BY HASH(id);
ERR-HYC00(16380): UNIQUE or PRIMARY KEY must include all sharding key columns for cluster
system
```

다음과 같이 UNIQUE(id, name)나 UNIQUE(name, id)와 같은 sharding key를 포함하여 제약 조건을 생성해야 한다.

```
gSQL>
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name VARCHAR(128),
    UNIQUE( id, name )
)
SHARDING BY HASH (id);
Table created.
```

노트

Non-deterministic 구문들은 cluster member 간 동일한 row를 식별하기 위해 global secondary index를 가지고 있어야 한다.

다음은 global secondary index를 생략하고 강제로 table을 생성한 경우에 발생하는 에러의 예이다.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER ) WITHOUT GLOBAL SECONDARY INDEX;
Table created.
gSQL> INSERT INTO t1 VALUES (1), (2), (3), (4), (5);
5 rows created.
gSQL> COMMIT;
Commit complete.
```

다음은 세 건을 삭제하는 예제 구문으로써 cluster member들이 동일한 row들을 삭제한다고 보장할 수 없다.

```
gSQL> DELETE FROM t1 FETCH 3;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

다음은 RANDOM(1, 100)을 사용하여 cluster member들이 동일한 row를 동일한 값으로 변경한다고 보장할 수 없음을 보여주는 예이다.

```
gSQL> UPDATE t1 SET c1 = RANDOM(1, 100);
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

다음은 updatable cursor를 이용해 현재 위치의 row를 변경하는 예로써 cluster member 간에 동일한 row를 식별하기 위해서는 global secondary index가 필요하다.

```
gSQL> \var v1 INTEGER
gSQL> DECLARE cur1 CURSOR FOR SELECT c1 FROM t1 FOR UPDATE;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> FETCH cur1 INTO :v1;
V1
--
1
1 row fetched.
```

```
gSQL> UPDATE t1 SET c1 = 1 WHERE CURRENT OF cur1;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

노트

지연 가능한 제약 조건을 지원하지 않는다.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id) DEFERRABLE;
ERR-HYC00(16388): does not support deferrable constraints in the cluster system :
ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id) DEFERRABLE
*
ERROR at line 1:
```

노트

서로 다른 server에서 동일한 sequence를 사용할 경우 동일한 순서대로 사용될지 여부를 보장하지 않는다.

- g1n1 서버에서 수행된다.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-----
1
1 row selected.
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-----
2
1 row selected.
```

- g2n1 서버에서 수행된다.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-----
21
```

1 row selected.

- 다시 g1n1 서버에서 수행된다.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
```

```
NEXTVAL
```

```
-----
```

```
3
```

1 row selected.

2.

튜토리얼

2.1 GOLDDILOCKS Instance 관리

본 장에서는 GOLDDILOCKS instance를 관리하기 위한 기반 지식에 대해 설명한다.

Overview

GOLDDILOCKS database system은 database와 instance로 구성된다. Database는 메모리상에 존재하는 dictionary data들과 사용자 data, 그리고 이에 대한 각각의 data file, 온라인 redo file 등 database를 구동하기 위한 각종 file들의 집합이다.

Database instance는 GOLDDILOCKS database를 운용하기 위해 사용되는 run-time 정보들을 담고 있는 메모리 부분과, 이를 운용하고 관리하는 background process를 의미한다. 각 database instance들은 shared memory를 구성할 때 사용되는 shared memory key 값과 "GOLDDILOCKS_DATA" 환경 변수 값으로 구분된다.

프로퍼티 설정

\$GOLDDILOCKS_DATA/conf/golddilocks.property.conf 파일에는 GOLDDILOCKS의 프로퍼티가 기술되어 있다. 기본 프로퍼티로도 GOLDDILOCKS를 설치할 수 있는데 본 문서에서는 TBS (tablespace), LOG, CONTROL FILE 등 주요 프로퍼티에 대해서만 설명한다.

다음은 GOLDDILOCKS 설치 시 주요 프로퍼티 항목과 그에 대한 설명이다.

표 2-1 주요 프로퍼티

프로퍼티	설명	기본값
SYSTEM_TABLESPACE_DIR	시스템 TBS들이 저장되는 디렉토리 경로로써 해당 경로에는 다음과 같은 TBS들이 설치된다. • DICTIONARY_TBS • MEM_DATA_TBS • MEM_UNDO_TBS • MEM_TEMP_TBS • MEM_TRANS_TBS	'<GOLDDILOCKS_DATA>/db'
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	딕셔너리 테이블스페이스 사이즈이다.	128M
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	데이터 테이블스페이스 사이즈이다.	200M
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	Undo 테이블스페이스 사이즈이다.	32M
LOG_DIR	기본 로그 디렉토리 경로이다.	'<GOLDDILOCKS_DATA>/wal'

프로퍼티	설명	기본값
SYSTEM_LOGGER_DIR	시스템 로그 디렉토리 경로이다.	'<GOLDILOCKS_DATA>/trc'
CONTROL_FILE_COUNT	컨트롤 파일의 개수이다.	2
CONTROL_FILE_0	첫 번째 컨트롤 파일의 경로이다.	'<GOLDILOCKS_DATA>/wal/control_0.ctl'
CONTROL_FILE_1	두 번째 컨트롤 파일의 경로이다.	'<GOLDILOCKS_DATA>/wal/control_1.ctl'

노트

- DICTIONARY_TBS: GOLDILOCKS의 dictionary table들을 저장하고 있는 tablespace 이다.
- MEM_DATA_TBS: 사용자 table/ index가 생성되는 tablespace 이다.
- MEM_UNDO_TBS: Transaction의 undo (rollback) 정보를 담고 있는 tablespace 이다.
- MEM_TRANS_TBS (Cluster only): Cluster system에서 global transaction의 복구를 위해 사용되는 tablespace로써 사이즈는 transaction table 크기와 cluster node의 수에 근거하여 자동으로 산정된다.

Database나 instance의 설정을 변경하려면 text 프로퍼티 파일 (\$GOLDILOCKS_DATA/conf/goldilocks.properties.conf)을 변경하거나 환경 변수에 GOLDILOCKS_<property_name> 형태의 변수를 새로 정의하면 된다. 프로퍼티 파일에 설정된 값이 환경 변수에 설정된 값보다 우선한다.

Background 프로세스

GOLDILOCKS는 instance를 관리하기 위한 background 프로세스 (gmaster)를 가지고 있다. gmaster는 내부적으로 여러 개의 system thread로 구성되어 있으며 그 내용은 다음과 같다.

표 2-2 System threads

Thread	설명
Main thread	gmaster 프로세스를 구동/ 종료한다.
Log archiving thread	온라인 redo log file을 switching 할 때 이전 redo log file을 지정된 위치로 복사하여 보관한다.
Ager thread	Drop 된 schema object들이 사용하고 있던 자원들을 정리한다.
Page flusher thread	Checkpoint가 발생할 때 in-memory 상에서 dirty 된 페이지들을 disk로 저장하기 위하여 IO slave들에 작업을 배분하고 이를 제어한다.
Log flusher thread	Run-time에 redo log buffer에 쌓인 log record들을 온라인 redo log file에 주기적으로 쌓는다.
Checkpoint thread	Redo log file을 switching 할 때 in-memory 상의 변경 사항을 disk 상에 존재하는 data file과 온라인 redo log file로 내린다.
Cleanup thread	비정상적으로 종료된 client들이 사용하던 자원을 정리하고 해당 트랜잭션들을 rollback 한다.

Thread	설명
IO slave thread	Checkpoint나 datafile loading 등과 같은 data file과 관련된 모든 disk IO를 수행한다.
Process monitor thread	Balancer (gbalancer), dispatcher (gdispatcher), shared-server (gserver) 등과 같은 process들을 실행하고 monitoring하여 비정상적으로 종료된 경우 재실행한다.
Cluster recover thread (Cluster only)	Cluster system에서 global transaction을 복구한다.
Failover thread (Cluster only)	Cluster system에서 특정 노드나 네트워크에 장애가 발생하면 장애 member에 대한 offline 및 coordinator 재선정과 같은 failover 작업을 처리한다.

Client 프로세스

Client/ Server 모델

Client/ Server (이하 C/S) 모델을 사용하는 응용 프로그램은 접속 요청을 기다리고 있는 listener (glsnr)에 접속하여 새로운 데이터베이스 서비스 프로세스 (gserver)를 생성하고 (dedicated 모드) 이 프로세스와의 TCP 통신으로 사용자의 요청을 처리한다. 프로세스간 통신을 통해 작업을 수행하므로 응용 프로그램 단에서 발생하는 모든 signal은 데이터베이스의 상태에 영향을 미치지 않으며, cleanup threads는 비정상적으로 종료된 응용 프로그램이 사용하던 모든 자원을 주기적으로 체크하여 반납한다.

Direct Access 모델

GOLDILOCKS는 Client/ Server (C/S) 모델과 더불어 Direct Access (D/A) 모델을 지원한다. 모든 응용 프로그램들은 GOLDILOCKS가 제공하는 서버 library를 링크하여 직접 database와 instance에 접근한다. 따라서 별도의 서비스 프로세스 없이 응용 프로그램 프로세스들만 존재한다.

D/A 모델로 작성된 응용 프로그램 프로세스가 운영 중에 발생하는 signal에 의해 비정상적으로 중단된 경우, 해당 프로세스가 사용하던 자원은 라이브러리가 connection 할 때 설정한 signal handler 함수에 의해 다음 두 단계로 정리된다.

1. 해당 프로세스의 signal handler는 자신이 사용하는 session 객체에 비정상 종료함을 표시하고 종료된다.
2. C/S 모델과 동일하게 gmaster의 cleanup thread에 의해 일정 시간 후에 database로 반환된다.

D/A 모델에서는 응용 프로그램 프로세스가 직접 데이터베이스 영역에 접근하므로 다음 사항들에 유의하여 사용해야 한다.

- D/A 모델로 운영 중이던 응용 프로그램 프로세스가 SEGV 등과 같은 치명적인 signal을 받아 바로 종료되면 library가 connection 할 때 등록한 signal handler가 사용 중이던 공유자원을 해제해야 하므로 응용 프로그램 내에 특정 시그널의 handler를 설치해야 할 경우, database에 connect 하기 전에 선언해야 한다.
- 응용 프로그램 프로세스를 강제 종료시킬 때는 프로세스가 signal 발생을 탐지할 수 없는 SIGKILL (kill -9)을 사용하지 말고 SIGTERM, SIGQUIT 이나 SIGUSR2 등을 사용해야 한다.

Instance의 메모리 구조

Database instance가 사용하는 메모리의 크기는 프로퍼티 파일에 지정된 관련 프로퍼티들에 의해 결정된다. Instance가 사용하는 shared memory는 크게 static 영역과 테이블스페이스 영역으로 나뉜다. Static 영역은 database를 운용하기 위해 여러 프로세스 사이에 공용으로 사용되는 instance의 기본 정보들과 각 session, statement 및 transaction 정보들, redo log buffer, dictionary cache 정보, 그리고 그 외 여러 가지 운용 정보들을 담고 있다. 테이블스페이스 영역은 각 테이블스페이스의 내용을 포함하는 page frame들과 이를 제어하기 위한 page control header (PCH)들로 구성되어 있다.

응용 프로그램 프로세스의 메모리는 connect 할 때 attach한 instance 메모리 외에 프로세스 단위로 공유하는 ODBC environment와 여러 ODBC handle들, 그리고 bind 정보와 같은 기타 정보들을 포함하고 있는 heap 메모리 영역을 추가적으로 포함하고 있다.

Instance 시작과 종료

GOLDILOCKS instance를 시작하려면 SHARED_MEMORY_STATIC_KEY 프로퍼티가 다른 instance들과 다르게 설정되어 있어야 한다. 이후 gsql을 이용하여 시작할 수 있는데 다음과 같이 sysdba role을 가지도록 실행시켜야 한다.

C/S 모델의 dedicated 모드에서 GOLDILOCKS instance를 시작하거나 종료하려면 listener가 실행되어 있어야 한다.

C/S 모델의 shared 모드에서는 GOLDILOCKS instance를 시작하거나 종료할 수 없다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

GOLDILOCKS instance는 다음과 같은 여러 단계의 startup phase를 갖는다.

- NOMOUNT
 - GOLDILOCKS instance를 관리하는 데몬인 gmaster 프로세스를 띄운다
- MOUNT
 - \$GOLDILOCKS_DATA 환경 변수를 이용하여 프로퍼티들과 복구를 위한 control file을 읽어들인다.
- OPEN
 - 테이블스페이스의 내용을 data file들로부터 로딩한 후 redo log file들을 이용하여 복구하고 no-logging index들을 새로 build 하여 dictionary cache를 생성한 후에 사용자의 서비스 접속을 기다린다.

다음 gsql 명령으로 GOLDILOCKS instance를 startup 한다.

```
gSQL> \startup nomount
Startup success
gSQL> alter system mount database;
System altered.
gSQL> alter system open database;
System altered.
```

만일 바로 OPEN 단계로 진입하려면 다음과 같이 수행한다.

```
gSQL> \startup open
Startup success
```

GOLDILOCKS instance가 종료되면 관리 데몬 프로세스인 gmaster가 종료되어 더 이상의 접속 및 기타 database 작업을 수행할 수 없다.

GOLDILOCKS instance는 다음과 같은 네 가지 방법으로 종료할 수 있다.

- NORMAL
 - 새로운 session의 접속을 차단하고 현재 접속된 모든 session이 종료될 때까지 기다린 후, checkpoint를 수행하고 instance를 내린다.
- TRANSACTIONAL
 - 새로운 transaction의 시작을 차단하고 현재 수행 중인 transaction들이 종료될 때까지 기다린 후, checkpoint를 수행하고 instance를 내린다.
- IMMEDIATE
 - 새로운 단위 연산 (FETCH, EXECUTE 등과 같은 GOLDILOCKS database와의 통신 단위)의 수행을 차단하고 현재 수행 중인 모든 단위 연산이 종료될 때까지 기다린 후, 모든 transaction들을 rollback하고 checkpoint를 수행한 후 instance를 내린다.
- ABORT
 - 접속 중인 session들의 상태와 관계없이 바로 gmaster를 종료시켜 instance를 내린다.

Instance를 종료할 때도 sysdba role로 gsql을 수행하고 다음과 같이 \shutdown 명령을 수행한다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown normal
Shutdown success
gSQL>
```

Listener 시작과 종료

Client/ server 환경에서 서비스하려면 listener를 구동시켜야 한다.

다음과 같이 listener를 구동한다.

```
% glsnr --start  
  
Listener is started successfully.  
%
```

다음과 같이 listener를 종료한다.

```
% glsnr --stop  
  
Listener is stopped.  
%
```

Listener 제어에 대한 자세한 설명은 **glsnr**을 참조한다.

Cluster system의 시작과 종료 방법은 **Cluster System 시작과 종료**를 참조한다.

2.2 GOLDDILOCKS 설치, 삭제와 Database 생성

본 절에서는 GOLDDILOCKS software를 install하고 database를 생성하는 방법에 대해 설명한다.

Overview

GOLDDILOCKS software는 *goldilocks- $\langle version\_no \rangle$ - $\langle os\_type \rangle$ - $\langle cpu\_type \rangle$.tar.gz*라는 이름의 압축파일로 제공되는데 압축을 풀면 해당 위치에 필요한 software 바이너리들과 각종 sample 및 기본적인 database directory 구조가 생성된 후 install이 완료된다. Install이 끝나면 해당 package 이름의 디렉토리가 생성되며 그 아래에 *goldilocks_home*과 *goldilocks_data* 디렉토리가 생성된다.

- *goldilocks_home* 디렉토리
 - GOLDDILOCKS product의 바이너리 홈이다.
 - GOLDDILOCKS_HOME 환경 변수로 정의된다.
 - 실행 바이너리, 클라이언트용 라이브러리, 헤더 파일, 라이선스 등이 위치한다.
- *goldilocks_data* 디렉토리
 - GOLDDILOCKS가 생성할 사용자 데이터베이스 (instance)의 홈이다.
 - GOLDDILOCKS_DATA 환경 변수로 정의된다.
 - 데이터 파일, 로그파일, 프로퍼티 파일 등이 존재하는 디폴트 위치이다.

이후 *\$GOLDDILOCKS_DATA* 위치에 database를 생성하는 작업은 *\$GOLDDILOCKS_HOME/bin* 디렉토리에 존재하는 *gcreatedb*라는 유틸리티를 사용한다.

릴리즈 플랫폼

GOLDDILOCKS는 다음과 같은 릴리즈 플랫폼에서 사용할 수 있다.

표 2-3 릴리즈 플랫폼

플랫폼	플랫폼 이름	OS	CPU	비고
Server platform	linux-x86_64	linux	x86_64	>= linux kernel 2.6 >= glibc 2.1 >= gcc 4.1.2 >= java 1.4 <= java 1.6
	hpux11.31-itanium-64	HP-UX 11.31	itanium	>= java 1.4 <= java 1.6
	aix6-powerpc-64	AIX 6.1	powerpc	>= java 1.4

플랫폼	플랫폼 이름	OS	CPU	비고
				<= java 1.6
Client platform	linux-x86_64	linux	x86_64	>= linux kernel 2.6 >= glibc 2.1 >= gcc 4.1.2 >= java 1.4 <= java 1.6
	linux-x86_32	linux	x86_32	>= Linux kernel 2.6 >= glibc 2.1 >= gcc 4.1.2 >= java 1.4 <= java 1.6
	hpux11.31-itanium-64	HP-UX 11.31	itanium	>= java 1.4 <= java 1.6
	hpux11.31-itanium-32	HP-UX 11.31	itanium	>= java 1.4 <= java 1.6
	aix6-powerpc-64	AIX 6.1	powerpc	>= java 1.4 <= java 1.6
	windows-x86-64	Windows	PENTINUM x86	java 1.6
	windows-x86-32	Windows	PENTINUM x86	java 1.6

시스템 요구사항

GOLDILOCKS를 install 하기 전에 다음과 같은 시스템 요구사항을 확인해야 한다.

- Physical memory와 디스크의 공간이 최소 2 G 이상 확보되어 있어야 한다.
- 충분한 양의 paging (swap) 영역이 존재해야 한다.
- Install 할 platform에 적합한 패키지 버전을 가지고 있어야 한다.

GOLDILOCKS Package 구성

GOLDILOCKS 설치 시 디렉토리의 구성 내용에 대해 설명한다.

Package 디렉토리 구성

표 2-4 상위 디렉토리 구성

디렉토리 이름	Server	Client	설명
GOLDILOCKS_H			바이너리와 라이브러리가 설치되며 업데이트 할 때 overwrite 할 수 있는

디렉토리 이름	Server	Client	설명
OME	O	O	그룹이다.
GOLDILOCKS_DATA	O	X	실제 데이터가 저장되는 경로이며 overwrite 할 수 없는 그룹이다.

표 2-5 Package 디렉토리 구성

구 분	이름	설명
GOLDILOCKS_HOME	admin	데이터베이스를 생성할 때 필요한 스키마 스크립트
	bin	실행 파일
	lib	Library 파일
	include	ODBC, XA, embedded SQL 등과 같은 header 파일
	license	라이선스 파일
	sample	샘플 파일
	msg	에러 메시지 파일
	script	편의성 스크립트 파일 (추후 지원 예정)
	app_dev	Application development
GOLDILOCKS_DATA	conf	환경 설정 파일
	db	데이터베이스 파일
	wal	로그 파일, 컨트롤 파일
	archive_log	Archive log file
	backup	백업 파일
	trc	Trace 로그 파일, 경고 메시지 파일
	journal	Cluster rebalance 할 때 사용되는 저널 파일

Package 파일 목록

각 디렉토리에 포함되어 있는 파일 및 server package 또는 client package에 포함되는지 여부를 설명한다.

표 2-6 admin/ standalone 디렉토리

파일 이름	Server	Client	설명
README	O	X	read me
DictionarySchema.sql	O	X	Dictionary schema 생성 스크립트
InformationSchema.sql	O	X	Information schema 생성 스크립트
PerformanceViewSchema.sql	O	X	PerformancView schema 생성 스크립트

표 2-7 admin/ cluster 디렉토리

파일 이름	Server	Client	설명
README	O	X	read me
DictionarySchema.sql	O	X	Dictionary schema 생성 스크립트
InformationSchema.sql	O	X	Information schema 생성 스크립트

파일 이름	Server	Client	설명
PerformanceViewSchema.sql	O	X	PerformancView schema 생성 스크립트

GOLDILOCKS를 standalone으로 사용할 때는 admin/ standalone 디렉토리 내의 생성 스크립트를 사용하고, cluster system을 구성하여 사용할 때는 admin/ cluster 디렉토리 내의 생성 스크립트를 사용한다.

표 2-8 bin 디렉토리 (Unix)

파일 이름	Server	Client	설명
README	O	O	read me
gmaster	O	X	GOLDILOCKS master
gcreatedb	O	X	Database creation tool
glsnr	O	X	Listener control tool
gbalancer	O	X	Load balancer for C/S shared
gdispatcher	O	X	Manage multiple connections for C/S shared
gserver	O	X	Instance manager for C/S
gsql	O	X	Interactive SQL tool
gsqlnet	O	O	Interactive SQL tool for C/S
gpec	O	O	Embedded SQL precompiler
logmirror	O	X	Redo-log replication tool
cyclone	O	X	CDC replication tool
gloader	O	X	Import/ export tool
gloadernet	O	O	Import/ export tool for C/S
cymon	O	O	CDC monitoring tool
gdump	O	X	Control/ log/ data/ binary property viewer
gsyncher	O	X	Synchronization utility for shared memory log and disk log files
tablediff.jar	O	X	Table comparison tool
cdispatcher	O	X	Manage cluster connections and distribute protocols in cluster system
cserver	O	X	Instance manager for cluster system
gtrclogger	O	X	Trace log manager for cluster system
gmon	O	X	Process monitoring tool
galocator	O	X	Location management tool
gagent	O	X	Location provider tool
gloctl	O	O	Interactive location editing tool

표 2-9 bin 디렉토리 (Windows client)

파일 이름	Server	Client	설명
README	X	O	read me
gloadernet.exe	X	O	Import/ export tool for C/S
gpec.exe	X	O	Embedded SQL precompiler
gsqlnet.exe	X	O	Interactive SQL tool for C/S

파일 이름	Server	Client	설명
gloctl.exe	X	O	-

표 2-10 lib 디렉토리 (Unix)

파일 이름	Server	Client (64 bit)	Client (32 bit)	설명
README	O	O	O	read me
libstib.so	O	X	X	Infiniband 용 shared library
libgoldilocks.a	O	X	X	D/A와 C/S가 포함된 static library for ODBC
libgoldilocks.a	O	X	X	D/A 전용 static library for ODBC
libgoldilocksas.so	O	X	X	D/A 전용 shared library for ODBC
libgoldilocksc.a	O	O	O	C/S 전용 static library for ODBC
libgoldilockscs-ul32.so	O	O	X	64 bit-C/S 전용 shared library for ODBC (SQLLEN = 4 byte)
libgoldilockscs-ul64.so	O	O	X	64 bit-C/S 전용 shared library for ODBC (SQLLEN = 8 byte)
libgoldilockscs.so	X	X	O	32 bit-C/S 전용 shared library for ODBC
libgoldilockscvtGB18030_32.so	X	X	O	32 bit GB18030 character set 변환 library
libgoldilockscvtGB18030_64.so	O	O	X	64 bit GB18030 character set 변환 library
libgoldilockscvtUHC_32.so	X	X	O	32 bit UHC character set 변환 library
libgoldilockscvtUHC_64.so	O	O	X	64 bit UHC character set 변환 library
libgoldilocksesql.a	O	O	O	Embedded SQL 용 static library
libgoldilocksesqls.so	O	O	O	Embedded SQL 용 shared library
libgoldilockss.so	O	X	X	D/A와 C/S가 포함된 shared library for ODBC
goldilocks4.jar	O	O	O	C/S 전용 JDBC library (java 1.4용)
goldilocks5.jar	O	O	O	C/S 전용 JDBC library (java 1.5용)
goldilocks6.jar	O	O	O	C/S 전용 JDBC library (java 1.6용)
libgoldilocksjni.so	O	X	X	D/A 전용 JDBC shared library

표 2-11 lib 디렉토리 (Windows client)

파일 이름	Server	Client (64 bit)	Client (32 bit)	설명
goldilocksc.lib	X	O	O	C/S 전용 static library for ODBC
goldilockscs.dll	X	X	O	32 bit-C/S 전용 shared library for ODBC
goldilockscs-ul64.dll	X	O	X	64 bit-C/S 전용 shared library for ODBC (SQLLEN = 8 byte)
goldilockssetup32.dll	X	X	O	32 bit setup library for ODBC
goldilockssetup64.dll	X	O	X	64 bit setup library for ODBC
goldilocksesql.lib	X	O	O	static library for embedded SQL
goldilocksesqls.dll	X	O	O	shared library for embedded SQL
goldilockscvtGB18030_				

파일 이름	Server	Client (64 bit)	Client (32 bit)	설명
32.dll	X	X	O	32 bit GB18030 character set 변환 library
goldilockscvtGB18030_64.dll	X	O	X	64 bit GB18030 character set 변환 library
goldilockscvtUHC_32.dll	X	X	O	32 bit UHC character set 변환 library
goldilockscvtUHC_64.dll	X	O	X	64 bit UHC18030 character set 변환 library
goldilocks6.jar	X	O	O	C/S 전용 JDBC library (java 1.6용)
gdlc.lib	X	O	O	C/S 전용 static library for ODBC
gdlds.lib	X	O	O	C/S 전용 shared library for ODBC
gdlds.dll	X	O	O	C/S 전용 shared library for ODBC

표 2-12 include 디렉토리 (Unix)

파일 이름	Server	Client	설명
README	O	O	read me
sql.h	O	O	ODBC 헤더 파일
sqlca.h	O	O	ODBC 헤더 파일
sqlext.h	O	O	ODBC 헤더 파일
sqltypes.h	O	O	ODBC 헤더 파일
sqlucode.h	O	O	ODBC 헤더 파일
goldilocks.h	O	O	GOLDILOCKS ODBC 응용 프로그램 개발용 헤더 파일
goldilockstypes.h	O	O	GOLDILOCKS ODBC 데이터타입 명세 파일
xa.h	O	O	표준 XA 헤더 파일
goldilocksxa.h	O	O	GOLDILOCKS XA 헤더 파일
goldilocksesql.h	O	O	Embedded SQL 헤더파일

표 2-13 include 디렉토리 (Windows client)

파일 이름	Server	Client	설명
README	X	O	read me
goldilocks.h	X	O	GOLDILOCKS ODBC 응용 프로그램 개발용 헤더 파일
goldilockstypes.h	X	O	GOLDILOCKS ODBC 데이터타입 명세 파일
goldilocksxa.h	X	O	GOLDILOCKS XA 헤더 파일
goldilocksesql.h	X	O	Embedded SQL 헤더파일
sqlca.h	X	O	ODBC 헤더 파일

표 2-14 license 디렉토리

파일 이름	Server	Client	설명
README	O	X	read me

표 2-15 msg 디렉토리

파일 이름	Server	Client	설명
README	O	O	read me
goldilocks_error.msg	O	O	에러 메시지 파일

표 2-16 conf 디렉토리

파일 이름	설명
README	read me
goldilocks.property.conf	database 운영 프로퍼티 텍스트 파일
goldilocks.listener.conf	Listener 프로퍼티 파일
goldilocks.invited.conf	database 접속 가능 client 관리 파일
goldilocks.excluded.conf	database 접속 거부 client 관리 파일
goldilocks.gagent.conf	gagent 전용 설정 파일
tablediff.conf	tablediff 설정 파일
cyclone.master.conf	Cyclone master 전용 기본 파일
cyclone.slave.conf	Cyclone slave 전용 기본 파일
logmirror.master.conf	LogMirror master 전용 기본 파일
logmirror.slave.conf	LogMirror slave 전용 기본 파일
odbc.ini	template for odbc configuration
gsq.ini	template for gsql configuration
glogin.sql	gsql 기동시 실행 명령 목록

표 2-17 db 디렉토리

파일 이름	설명
README	read me

표 2-18 wal 디렉토리

파일 이름	설명
README	read me

표 2-19 archive_log 디렉토리

파일 이름	설명
README	read me

표 2-20 backup 디렉토리

파일 이름	설명
README	read me

표 2-21 trc 디렉토리

파일 이름	설명
README	read me

GOLDILOCKS Software 설치

GOLDILOCKS를 설치하기 전에 운영체제 및 환경 설정에 대해 설명한다.

커널 파라미터

공유 메모리 (shared memory)

공유 메모리란 Inter Process Communication (IPC)의 일종으로 복수의 프로그램에서 데이터를 공유하기 위해 사용되는 메모리를 가리키는 용어이다. GOLDILOCKS에서는 공유 메모리와 함께 gsql, gloder 및 Client/ Server (C/S) 환경의 ODBC를 이용하는 사용자 프로그램을 사용한다. 또한 운영에 필요한 tablespace들이 모두 공유 메모리에 생성되기 때문에 다른 프로그램에서 권장되는 값보다 정확하게 설정해야 한다.

다음은 GOLDILOCKS 설치에 사용되는 공유 메모리에 필요한 항목과 권장값이다.

표 2-22 Shared memory 관련 kernel property

파라미터 이름	설명	권장값	비고
shmmax	하나의 shared memory 세그먼트가 가지는 최대 크기	가장 큰 datafile의 크기보다 큰 값	구상하고 있는 테이블스페이스에 속한 가장 큰 datafile의 크기보다 큰 값으로 설정한다.
shmmni	시스템이 사용 가능한 shared memory 세그먼트의 최대 개수	전체 datafile의 개수 + 1 보다 큰 값	전체 datafile의 개수 + 1 (SSA를 위한 shared memory 세그먼트)보다 큰 값으로 설정한다.
shmall	전체 shared memory 세그먼트의 총합 (Page 수)	Tablespace 구성의 총합보다 큰 값	시스템이 사용 가능한 shared memory의 총 페이지 수를 의미한다. 일반적으로 8 GB 이상의 큰 공유 메모리를 사용할 경우에 이 파라미터가 사용된다. GOLDILOCKS의 테이블스페이스의 총합이 32 G 라면 shmall은 그 이상의 값으로 설정되어야 한다.

다음은 테이블스페이스의 총합이 32 G 였을 때 shmall 값의 예이다.

```
kernel.shmmax = 34359738368
kernel.shmmni=4096
kernel.shmall = 8388609
```

- shmmax 값은 32 G이며, PAGE_SIZE 4096 byte라고 가정하였다.

8388609 = (34359738368 / 4096) + 1

- 이 경우 shmall은 8388609 이상의 값을 가져야 한다.

세마포어 (semaphore)

세마포어 (semaphore)는 공유 메모리 (shared memory)와 같은 IPC의 일종으로 운영체제에서 자원을 사용하는 복수의 프로세스의 행동을 제어하는 기술이다. 세마포어는 설정에 따라 복수의 프로세스가 해당 자원을 동시에 참조하게 할 수 있고 다른 프로세스가 사용 중인 자원에 대해 해당 프로세스가 사용을 마칠 때까지 대기하도록 할 수도 있다.

GOLDILOCKS에서도 앞에서 설명한 공유 메모리에 대한 접근순서를 제어하기 위해 세마포어가 사용된다. 예를 들어, 다수의 GOLDILOCKS 클라이언트 프로그램에서 동일한 데이터 변경을 요청하는 경우 이를 적절하게 제어할 수 있어야 하기 때문이다. 세마포어의 파라미터 값은 GOLDILOCKS를 운영할 때 발생하는 세마포어 오퍼레이션에 따라 적절한 값으로 설정되어야 하며 본 문서에서는 리눅스의 일반값을 권장한다.

다음은 GOLDILOCKS 설치에 사용되는 세마포어의 권장값이다.

표 2-23 세마포어 관련 커널 파라미터 권장값

커널 파라미터	설명	권장값
semmsl	한 개의 세마포어 셋에 설정되는 세마포어의 개수	250
semmni	세마포어 셋의 개수	128
semmns	세마포어 셋의 총 개수 (semmni * semmsl)	32000
semopm	시스템 콜 별로 수행할 수 있는 세마포어의 최대 개수	100

Redhat, Ubuntu와 같은 리눅스 기반 시스템의 systemd가 관리하는 세션 목록에서 IPC 자원 생성 사용자가 로그아웃하면 해당 IPC 자원이 자동 삭제되는 옵션이 생김에 따라 시스템을 다음과 같이 설정해 주어야 세마포어 삭제를 방지할 수 있다. (커널 3.0.0 이상)

```
# cp -i /etc/systemd/logind.conf /etc/systemd/logind.conf_prev
# cat /etc/systemd/logind.conf
[Login]
#NAutoVTs=6
#ReserveVT=6
...
RemoveIPC=no
```

- 수정 후 수행한다.

```
# systemctl restart systemd-logind
```


네트워크 (network)

Backlog는 TCP socket listen 할 때 accept 되길 기다리는 socket들의 queue 길이이다. GOLDILOCKS glsnr의 backlog는 glsnr의 config 파일의 BACKLOG로 설정한다. System에서 backlog의 최대값이 somaxconn 보다 큰 경우에는 somaxconn로 설정되므로 이 때는 somaxconn를 늘려주어야 한다.

GOLDILOCKS에서 C/S의 shared 모드로 동작할 때 사용되는 Unix Domain Socket (UDS)의 queue 길이는 max_dgram_qlen으로 설정된다. Client들이 동시에 접속할 때 이 값이 작으면 glsnr, gbalancer, gdispatcher 간의 통신에 병목이 발생한다.

다음은 GOLDILOCKS 설치에 사용되는 네트워크의 권장값이다.

표 2-24 네트워크 관련 커널 파라미터 권장값

커널 파라미터	설명	권장값
somaxconn	listen backlog의 최대값	1024
max_dgram_qlen	Unix Domain Socket Queue 크기	256

파라미터 적용

일회성으로 적용할 때는 다음과 같이 적용한다. (시스템을 재기동할 때는 다시 적용해야 한다.)

```
[SHELL]> echo 34359738368 > /proc/sys/kernel/shmmax
[SHELL]> echo 8388608 > /proc/sys/kernel/shmall
[SHELL]> echo 4096 > /proc/sys/kernel/shmmni
[SHELL]> echo 250 32000 100 128 /proc/sys/kernel/sem
[SHELL]> echo 1024 > /proc/sys/net/core/somaxconn
[SHELL]> echo 256 > /proc/sys/net/unix/max_dgram_qlen
```

시스템이 다시 시작될 때도 적용되게 하려면 /etc/sysctl.conf에 다음과 같이 기술한다.

```
# shared memory
kernel.shmmax = 34359738368
kernel.shmall = 8388608
kernel.shmmni = 4096
# semaphore
kernel.sem = 250 32000 100 128
# network
net.core.somaxconn = 1024
net.unix.max_dgram_qlen = 256
```

기술된 내용이 반영되도록 하려면 다음 명령어를 이용한다.

```
[SHELL]> sysctl -p
```

파라미터 확인

기술된 파라미터는 다음 명령어로 확인할 수 있다.

```
[SHELL]> ipcs -l
----- Shared Memory Limits -----
max number of segments = 4096
max seg size (kbytes) = 33554432
max total shared memory (kbytes) = 33554432
min seg size (bytes) = 1
----- Semaphore Limits -----
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 100
semaphore max value = 32767
```

압축해제

GOLDILOCKS의 패키지는 압축 형태로 제공되므로 압축을 해제하면 기본 설치가 끝난다.

다음은 GOLDILOCKS 패키지를 설치하는 간단한 예이다.

```
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
[SHELL]> gzip -d goldilocks-mercury.2.1.0-linux-x86_64.tar.gz
[SHELL]> tar -xvf goldilocks-server-mercury.2.1.0-linux-x86_64.tar
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlext.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/goldilocks.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlca.h
...
```

압축 해제가 완료되면 <package_file_name>으로 생성된 디렉토리의 이름을 원하는 대로 변경할 수 있다. 이에 따라 \$GOLDILOCKS_HOME과 \$GOLDILOCKS_DATA 환경 변수도 변경해야 한다.

압축 해제 후 생성되는 디렉토리에 대한 설명은 **GOLDILOCKS Package 구성**을 참조한다.

환경 변수 설정

GOLDILOCKS package의 압축을 해제한 후에는 향후 GOLDILOCKS software를 실행하고 응용 프로그램을 개발하기 위하여 \$GOLDILOCKS_HOME 디렉토리 아래에 생성된 bin과 lib를 각각 다음과 같이 PATH와 LD_LIBRARY_PATH에 추가한다. (GOLDILOCKS client 응용 프로그램을 개발할 때는 항상 컴파일 옵션의 include file directory로

\$GOLDILOCKS_HOME/include를 넣어주어야 한다)

```
export PATH=$GOLDILOCKS_HOME/bin:$PATH
export LD_LIBRARY_PATH=$GOLDILOCKS_HOME/lib:$LD_LIBRARY_PATH
```

GOLDILOCKS를 사용하는데 필요한 환경 변수가 있다. 설치할 때 참조하는 변수도 있기 때문에 반드시 설치하기 전에 설정해야 한다.

표 2-25 GOLDILOCKS 설치 시 참조되는 OS 환경 변수

환경 변수	설명	비고
GOLDILOCKS_HOME	GOLDILOCKS 바이너리가 설치될 디렉토리 경로	GOLDILOCKS를 운영할 때 참조하는 변수로써 GOLDILOCKS가 설치될 디렉토리를 환경 변수로 미리 설정해야 한다.
GOLDILOCKS_DATA	GOLDILOCKS database instance가 생성될 위치	GOLDILOCKS의 데이터베이스를 생성하거나 운영할 때 참조하는 변수이다.
PATH	GOLDILOCKS의 실행파일들이 위치한 디렉토리 경로	GOLDILOCKS의 각종 바이너리를 절대 경로 없이 실행할 때 설정해야 한다.
LANG	터미널의 character set	- GOLDILOCKS의 DATABASE를 생성할 때 기술한 character set과 다를 경우 영문, 숫자, 특수 문자를 제외한 문자들이 올바르게 출력되지 않거나 string 관련 함수들이 올바르게 동작하지 않을 수 있다. - GB18030, SQL_ASCII, UHC, UTF8에 대응하는 locale을 설정해야 한다. - 예: export LANG=ko_KR.utf8

```
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
[HELL]> gzip -d goldilocks-mercury.2.1.0-linux-x86_64.tar.gz
[HELL]> tar -xvf goldilocks-server-mercury.2.1.0-linux-x86_64.tar
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlxext.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/goldilocks.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlca.h
...
```

DATABASE 삭제

GOLDILOCKS DATABASE를 다시 생성하기 위해 기존 DATABASE를 삭제하려면 데이터 파일, 컨트롤 파일, redo log file, (archive log 사용시) archive log file을 삭제하면 된다.

다음은 GOLDILOCKS DATABASE를 삭제하는 간단한 예이다.

```
## $GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
## $GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
goldilocks_data/
```

```
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
    goldilocks_home/
[HELL]> rm -rf $GOLDILOCKS_DATA/db/*.dbf
[HELL]> rm -rf $GOLDILOCKS_DATA/wal/*.ctl
[HELL]> rm -rf $GOLDILOCKS_DATA/wal/*.log
[HELL]> rm -rf $GOLDILOCKS_DATA/archive_log/*.log
```

데이터 파일이나 archive log file 등의 위치는 사용자 설정에 따라 다를 수 있다.

삭제

GOLDILOCKS 패키지는 압축 파일 형태로 제공되므로 특별한 삭제 원칙이 필요하진 않으며 DATABASE를 종료한 후에 설치된 디렉토리를 삭제하면 된다.

다음은 GOLDILOCKS 패키지를 삭제하는 간단한 예이다.

```
## $GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
## $GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
    goldilocks_data/
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
    goldilocks_home/
[HELL]> rm -rf $GOLDILOCKS_DATA
[HELL]> rm -rf $GOLDILOCKS_HOME
[HELL]> rm -rf $GOLDILOCKS_BASE
## ~/.bash_profile
$GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/ ❶ 삭제
$GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
    goldilocks_data/ ❷ 삭제
$GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
    goldilocks_home/ ❸ 삭제
```

Database 생성

프로퍼티 생성이 끝난 후 데이터베이스를 생성한다. 데이터베이스는 \$GOLDILOCKS_HOME/bin/gcreatedb 명령어를 이용하여 생성할 수 있다. 다음은 gcreatedb 명령어를 사용하는 방법이다.

```
[HELL]> gcreatedb --help
Usage
    gcreatedb [options]
Options:
```

```

--cluster      cluster system (if not specified, stand-alone system)
--db_name      database name
--db_comment   database comment
--timezone     timezone ( {+/-}{TZH:TZM} )
--character_set character set
                SQL_ASCII
                UTF8
                UHC
                GB18030
--char_length_units char length units
                OCTETS
                CHARACTERS
--home         home directory
--member       local member name
--host         host address
--port         host port
--silent       suppresses the display of the result message
--help         print help message

```

examples:

```

gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00" --
character_set="UTF8" --char_length_units="OCTETS" --silent

```

데이터베이스를 생성할 때는 \$GOLDILOCKS_HOME/conf/goldilocks.properties.conf를 참조하여 각각의 ***_TABLESPACE_SIZE 값으로 goldilocks.properties.conf의 SYSTEM_TABLESPACE_DIR 경로에 tablespace file들이 생성된다.

Cluster system에 참여시킬 database를 생성할 경우에는 반드시 --cluster 옵션을 명시하여 실행하여야 한다.

다음은 gcreatedb 명령어의 실행 인자에 대한 설명이다.

표 2-26 gcreatedb 인자

실행 인자	설명
--cluster	Cluster system에서 사용할 database 이다. 생략할 경우, standalone database를 생성한다.
--db_name	데이터베이스 이름이다. 생략할 경우, goldilocks로 설정된다.
--db_comment	데이터베이스에 대한 설명이다. 생략할 경우, goldilocks database로 설정된다.
--timezone	타임 존이다. 생략할 경우, TIMEZONE 프로퍼티로 설정된다.
	데이터베이스 character set 이다. GOLDILOCKS는 다음과 같은 네 가지 character set을 지원한다. GB18030: 중국어 간체


```
--timezone="+09:00"          \
--character_set="UHC"         \
--char_length_units="OCTETS"
```

Database created

```
[SHELL]> ls $GOLDILOCKS_DATA/db
```

```
system_data.dbf  system_dict.dbf  system_undo.dbf
```

다음은 cluster system에서 사용될 데이터베이스를 생성하는 예이다.

```
[SHELL]> gcreatedb --cluster
```

Database created

```
[SHELL]> gcreatedb --cluster --member=G1N1
```

Database created

```
[SHELL]> gcreatedb --cluster --member=G1N1 --host=127.0.0.1 --port 10101
```

Database created

```
[SHELL]> gcreatedb --cluster          \
--db_name="TEST_DB"                  \
--home=$GOLDILOCKS_DATA              \
--host=127.0.0.1                     \
--port=10101                          \
--db_comment="g1n1 db comment"       \
--timezone="+09:00"                  \
--character_set="UHC"                 \
--char_length_units="OCTETS"
```

Database created

```
[SHELL]> ls $GOLDILOCKS_DATA/db
```

```
system_data.dbf  system_dict.dbf  system_trans.dbf  system_undo.dbf
```

Dictionary Schema 정보 구축

시스템 및 객체 정보를 조회하기 위해 다음과 같은 스키마를 생성한다.

주의

Database를 생성한 후에는 다음과 같은 스키마들을 반드시 구축해야 한다. 구축하지 않을 경우 객체의 구조 정보를 획득하는 ODBC, JDBC의 catalog API (예: SQLTables() 함수 등)들이 오동작하여 third party tool 들과 연동하지 않을 수 있다.

- `DICTIONARY_SCHEMA`: `DBA_*`, `ALL_*`, `USER_*` 등의 객체 정보를 조회하는 view 및 table로 구성된다.
- `INFORMATION_SCHEMA`: SQL 표준의 `INFORMATION_SCHEMA`에 포함되는 view 및 table로 구성된다.
- `PERFORMANCE_VIEW_SCHEMA`: Fixed table들의 정보를 조합하여 시스템 정보를 조회하는 view로 구성된다.

해당 스키마들에 포함된 view와 table들은 시스템의 정보를 편리하게 조회할 수 있도록 한다.

`GOLDILOCKS` instance를 `OPEN` 상태로 구동한 후에 다음과 같이 sql 파일들을 수행한다. 데이터베이스를 생성한 후에 최초 한 번은 수행해야 한다.

노트

Dictionary schema 정보를 구축하기 위한 스크립트는 standalone용과 cluster system용으로 구분되어 있으므로 반드시 용도에 맞는 스크립트를 사용하여 구축해야 한다.

다음은 standalone database용 스크립트를 사용하여 dictionary schema 정보를 구축하는 방법이다.

```
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/DictionarySchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/InformationSchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/PerformanceViewSchema.sql
```

다음은 cluster system용 스크립트를 사용하여 dictionary schema 정보를 구축하는 방법이다.

```
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.sql
```


2.3 Database 저장구조 관리

본 절에서는 GOLDDLOCKS instance를 구성하는 데이터베이스의 구성 요소들에 대해 설명한다.

Database 저장구조

GOLDDLOCKS database는 크게 memory 영역과 disk 영역으로 나뉜다. Memory 영역은 하나 이상의 shared memory 집합으로 구성된 테이블스페이스들의 집합체이며 in-memory database이기 때문에 replace 작업에 의해 disk로 내려가는 경우가 발생하지 않는다.

Disk 영역은 각 테이블스페이스의 shared memory 한 개당 하나씩 존재하는 data file들과 instance의 구성 정보를 저장하고 있는 control file, instance의 환경설정이 저장된 프로퍼티 파일, 그리고 database 복구를 위해 기록되는 온라인 redo log file들로 구성되어 있다.

Control File

Control file은 database의 디스크상에 물리적인 저장 정보를 기록하고 있는 파일이며 instance를 startup 할 때 가장 먼저 읽어들이며 database의 상태를 파악하는 파일이다. Control file에는 다음과 같은 항목들이 기록되어 있다

- 이전에 checkpoint 할 때의 instance 상태
- 이전에 startup 할 때의 Transaction Durability Mode (CDS/ TDS)
- 온라인 redo log file 정보
- 각 테이블스페이스 별 data file 정보

온라인 Redo Log File

온라인 redo log file은 database가 비정상적으로 종료되었을 때 미처 data file에 기록되지 못한 변경 정보들을 instance를 재기동할 때 복구하기 위해 해당 instance에서 transaction들이 database 변경 연산을 수행한 모든 정보를 남긴 파일이다. 기본적으로 database가 생성될 때 LOG_FILE_SIZE 프로퍼티의 값에 해당하는 크기로 네 개가 생성되며 필요한 경우 추가할 수도 있다. 이 redo log file들은 circular 방식으로 순환하며 재사용된다.

온라인 redo log file이 다음 파일로 switching 되는 시점에는 checkpoint가 발생하여 일부 dirty 된 페이지들을 해당 data file로 내리는데 만일 checkpoint 연산이 지연되어 다음으로 사용될 온라인 redo log file에서 변경된 페이지가 아직 내려가지 못한 경우에는 (ACTIVE 상태인 경우) checkpoint가 완료될 때까지 모든 transaction들이 정지된다. 따라서 응용 프로그램의 성격에 따라 적당한 크기의 온라인 redo file을 생성하는 것이 database 시스템 성능 향상에 도움이 된다.

Undo Segment

Undo segment들은 부분 또는 전체 rollback을 할 때 사용하기 위해 transaction들이 변경 연산을 수행하기 이전의 이미지들을 미리 기록해 둔 segment이며 변경 연산을 수행하는 transaction당 한 개씩 할당되어 사용된다. Undo segment들은 MEM_UNDO_TBS 테이블스페이스에 저장되는 동시에 다수의 변경 transaction이 발생하거나 대량의 변경 연산이 하나의 transaction으로 수행되는 경우 (bulk delete 등)에 대비하여 충분한 undo 테이블스페이스를 확보해 두는 것이 좋다.

Data File

Data file은 테이블스페이스에 저장된 테이블/ 인덱스의 내용이 그대로 저장된 파일이다.

Data file은 다음 요소들로 구성되어 있다.

- Page
 - Database I/O의 최소 단위로서 현재 8 Kbyte로 정의되어 있다.
- Extent
 - 일정 개수의 연속된 page의 집합이다. Segment가 테이블스페이스로부터 공간을 할당해가는 최소 단위이다.
 - 각 테이블스페이스별로 extent의 크기는 다를 수 있다.
- Segment
 - 특정 타입의 데이터들을 모아 둔 객체이다. Extent들의 집합으로 구성된다.

예외적으로 MEM_TEMP_TBS 테이블스페이스와 같은 임시 테이블스페이스는 redo logging 하지 않으며 data file도 생성하지 않는다.

테이블스페이스

Database는 테이블과 인덱스 등을 담고 있는 논리적 구조인 테이블스페이스로 나뉘어진다. 테이블스페이스는 하나 이상의 shared memory의 집합으로 구현되며 데이터를 적재할 공간을 관리하는 도구이다. V\$TABLESPACE 테이블을 조회하면 현재 database에 존재하는 테이블스페이스들에 대한 정보가 출력된다.

GOLDILOCKS는 기본적으로 다음과 같은 테이블스페이스들을 지원한다.

표 2-27 GOLDILOCKS의 테이블스페이스 종류

소유자	이름	설명
SYSTEM	DICTIONARY_TBS	Database를 운용하기 위한 기본적인 dictionary 테이블들이 저장된다.
	MEM_UNDO_TBS	Undo segment들과 transaction 정보들이 저장된다.
	MEM_DATA_TBS	사용자가 schema 객체를 생성할 때 별도로 지정하지 않을 경우, 기본적으로 이 테이블스페이스에 저장된다.
	MEM_TEMP_TBS	Query 중에 생성되는 임시 테이블이나 테이블스페이스를 지정하지 않은 index들이 여기에 생성된다. Logging이 일어나지 않기 때문에 instance를 재기동할 때 index가 다시 build된다.

소유자	이름	설명
	MEM_TRANS_TBS	Cluster system에서 global transaction을 복구하기 위해 사용된다. Cluster system으로 구성할 경우에만 생성된다.
USER	사용자 정의	사용자가 특정 용도의 테이블들을 테이블스페이스로 모아 관리하고 싶을 때 생성하여 사용한다.

테이블스페이스 타입

다음과 같이 다섯 가지 테이블스페이스 타입이 있다.

- DICT
 - Dictionary 테이블과 인덱스를 저장하기 위한 테이블스페이스 타입이다.
- DATA
 - 테이블, 인덱스와 같이 일반적인 schema 객체들을 저장하기 위한 테이블스페이스 타입이다.
 - Redo logging과 페이지 flush의 대상이 된다
- UNDO
 - Undo segment들을 저장하기 위한 테이블스페이스 타입이다.
 - Redo logging과 페이지 flush의 대상이 된다
- TEMPORARY
 - SELECT query 도중에 생성되는 임시 테이블들과 no-logging index들을 저장하기 위한 테이블스페이스 타입이다.
 - Redo logging과 페이지 flush가 일어나지 않는다.
- TRANSACTION (Cluster only)
 - Cluster system에서 global transaction을 복구하기 위해 사용되는 테이블스페이스 타입이다.

Database 저장 구조 정보 확인

본 절에서는 앞서 기술한 여러 database 저장 구조에 관한 정보를 확인할 수 있는 방법을 설명한다.

Control File 정보

Control file은 바이너리 형식으로 저장되어 있기 때문에 그 내용을 확인하려면 다음과 같이 gdump라는 유틸리티를 사용해야 한다.

```
[SHELL]> gdump CONTROL control_0.ctl
```

온라인 Redo Log File 정보

gdump tool을 사용하여 control file을 조회하면 각 온라인 redo file들의 이름과 현재 상태가 표시된다.

Data File 정보

gdump tool을 사용하여 control file을 조회하면 각 테이블스페이스에 속한 데이터 파일들과 각각의 상태가 표시된다. 또한 V\$DATAFILE 테이블을 조회하면 현재 상태가 표시된다.

테이블스페이스 정보

gdump tool을 사용하여 control file을 조회하면 현재 database에 존재하는 각 테이블스페이스의 이름과 상태가 표시된다. 또한 V\$TABLESPACE 테이블을 SQL로 조회할 수도 있다.

프로퍼티 정보

기본적으로는 텍스트 파일인 \$GOLDILOCKS_Data/conf/goldilocks.properties.conf 파일을 열어서 확인할 수 있는데 온라인 상태에서는 V\$PROPERTY 테이블을 조회하면 현재 적용 중인 프로퍼티 값들을 알 수 있다.

일반적인 데이터 저장소 작업

데이터 저장소인 테이블스페이스에서는 다음과 같은 작업을 할 수 있다.

테이블스페이스 생성

USER DATA 테이블스페이스는 다음과 같은 명령으로 생성할 수 있다.

```
gSQL> CREATE TABLESPACE TEST_TBS DATAFILE 'TEST_TBS.dbf' SIZE 10M;
Tablespace created.
```

TEMPORARY 테이블스페이스는 data file을 가지지 않으므로 다음과 같은 구문으로 생성한다.

```
gSQL> CREATE TEMPORARY TABLESPACE TEST_TEMP_TBS MEMORY 'TEST_TEMP_TBS' SIZE 10M;
Tablespace created.
```

테이블스페이스 공간 사용 상황 조회

테이블스페이스의 공간은 한 개 이상의 연속된 page로 구성된 extent 단위로 할당되고 해제된다. 특정 테이블스페이스의 extent 한 개의 크기 (BYTE)는 다음과 같이 조회할 수 있다.

```
gSQL> SELECT EXTENT_SIZE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
EXTENT_SIZE
-----
262144
```

```
1 row selected.
```

테이블스페이스의 모든 extent 상태는 D\$TABLESPACE_EXTENT 테이블을 사용하여 조회할 수 있다. 사용 중인 extent는 STATE column이 'U'로 표시되고 free 상태의 extent는 'F'로 표시된다. 따라서 현재 테이블스페이스의 공간 중 남은 공간의 크기 (extent의 개수)는 다음과 같이 확인할 수 있다

```
gSQL> SELECT COUNT(*) FROM D$TABLESPACE_EXTENT('TEST_TBS') WHERE STATE = 'F';
COUNT(*)
-----
        38
1 row selected.
```

위의 결과를 보면 현재 TEST_TBS의 빈 공간은 $38 * 262144 = 9437184$ Byte인 것을 알 수 있다.

테이블스페이스 변경

테이블스페이스를 변경하는 방법에는 add/ remove data file (temporary 테이블스페이스의 경우는 memory)과 online/ offline이 있다.

Add/ Drop Data File (or Memory)

데이터베이스를 운영하는 중에 테이블스페이스의 여유 공간이 부족할 경우 다음과 같은 구문으로 DATA 테이블스페이스에 추가 공간을 할당할 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD DATAFILE 'TEST_TBS2.dbf' SIZE 10M;
Tablespace altered.
```

TEMPORARY 테이블스페이스에는 다음과 같이 공간을 추가한다. DATA 테이블스페이스와 달리 메모리의 이름을 부여하는데 이는 database 내에서 고유한 이름이어야 한다.

```
gSQL> ALTER TABLESPACE TEST_TEMP_TBS ADD MEMORY 'TEST_TEMP_TBS2' SIZE 10M;
Tablespace altered.
```

DATA 테이블스페이스의 공간은 다음과 같은 구문으로 삭제한다. 단, 해당 공간을 일부라도 사용한 경우에는 삭제할 수 없다.

```
gSQL> ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf';
Tablespace altered.
```

마찬가지로 TEMPORARY 테이블스페이스의 공간은 다음과 같이 회수한다.

```
gSQL> ALTER TABLESPACE TEST_TEMP_TBS DROP MEMORY 'TEST_TEMP_TBS2';
Tablespace altered.
```

테이블스페이스 Offline

테이블스페이스에 속한 data file의 위치를 변경하는 등의 작업을 수행할 때는 해당 테이블스페이스를 offline 상태로 변경하여야 한다. 다음 구문을 이용하여 offline 상태로 변경할 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;
Tablespace altered.
```

다음과 같이 다시 online 상태로 변경할 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

Rename

테이블스페이스의 이름을 변경하려면 다음 구문을 수행한다.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME TO TEST_TBS2;
Tablespace altered.
```

만일 테이블스페이스 내의 data file 위치를 바꾸려면 해당 테이블스페이스를 우선 offline 상태로 변경해야 하며 이후에 OS 명령어로 해당 data file을 직접 이동시킨 후 ALTER TABLESPACE 구문으로 테이블스페이스의 data file 이름을 변경한 후 테이블스페이스를 online 상태로 다시 변경해야 한다.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf';
ERR-42000(16164): file does not exist :
ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf'
*
ERROR at line 1:
```

- OS 명령어로 파일 이름을 변경한다.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf';
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

테이블스페이스 제거

필요 없어진 테이블스페이스는 다음과 같이 제거할 수 있다. INCLUDING 이하 구문은 옵션이며 이 옵션을 사용하면 해당 테이블스페이스에 존재하는 모든 내용 (schema object)들을 제거할 뿐만 아니라 data file들도 제거한다.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

Store Mode

GOLDILOCKS는 특정 환경에서 최대의 성능을 발휘할 수 있도록 하기 위해 instance 단위로 store mode를 설정하여 사용할 수 있도록 한다. Store mode란 연산 성능을 향상시키기 위해 transaction들이 ACID 프로퍼티 중에서 어떤 부분을 포기할 것인지를 정해 놓은 것이다.

GOLDILOCKS는 현재 다음과 같은 두 가지 store mode를 지원한다.

- Transactional Data Store (TDS) Mode
 - 이 모드가 GOLDILOCKS database의 default store mode이다.
 - 해당 instance에서 수행되는 모든 transaction들이 redo 및 undo 로그들을 모두 남기는 일반적인 DBMS의 store mode이다. 따라서 transaction들을 rollback 할 수 있고 주기적으로 checkpoint가 이루어지기 때문에 비정상적으로 종료되더라도 data file과 온라인 redo log file을 이용하여 복구할 수 있다.
- Concurrent Data Store (CDS) Mode
 - 해당 instance에서 수행되는 모든 transaction들이 undo 로그는 남기지만 redo 로그는 남기지 않는다. 따라서 transaction들은 모든 run-time 에러들에 대처할 수 있고 rollback도 가능하다. 단, instance가 비정상 종료되거나 shutdown abort 명령으로 종료되면 checkpoint가 발생하지 않아 수행되었던 모든 변경 정보들을 잃어버린다. (복구 기능을 제공하지 않는다) 이 모드는 transaction들 간의 동시성을 제어하기 때문에 동일한 객체에 서로 다른 transaction들이 동시에 접근해도 정상적인 동작을 보장한다.
 - 이 모드는 cache 서버와 같이 매우 빈번한 조회와 갱신이 이루어지지만 durability를 보장할 필요는 없는 run-time 정보 위주의 데이터베이스에서 주로 사용된다.

Store mode는 instance를 startup 할 때 프로퍼티 설정을 통해 작동시키는데 transaction들이 서로 다른 store mode로 동작할 수는 없다. 또한 instance가 online인 상태에서는 변경할 수도 없기 때문에 startup 할 때 해당 mode를 신중하게 결정해야 한다.

2.4 Schema Object 관리

Schema Object

Schema object는 사용자가 생성한 논리적 구조물들이다. GOLDILOCKS는 table, index, synonym, view, sequence, constraint, stored procedure와 같은 schema object들을 지원한다.

Schema Object 관리 권한

현재 GOLDILOCKS는 사용자와 권한을 지원한다. 따라서 생성되는 모든 객체는 모든 사용자에게 공유될 수 없고 공유하기 위해서는 권한을 부여해야 한다.

테이블 관리

본 절에서는 테이블에 대한 개요와 테이블과 관련된 정보를 확인하는 방법, 그리고 테이블의 생성/변경/데이터 로딩/삭제 방법에 대해 설명한다.

테이블

테이블은 사용자의 데이터를 저장하는 가장 기본적인 단위이다. 테이블은 column과 row로 구성된다.

테이블 타입

현재 GOLDILOCKS는 clustered table, partitioned table을 지원하지 않기 때문에 (데이터의 저장 순서가 특정 column의 sorting order와 상관없는) 일반적인 heap table만 지원한다.

- 기본적인 heap table만 지원한다.
- Primary key/ unique/ not null constraint를 지원한다.
- 지원하는 DataTypes
 - BOOLEAN
 - SMALLINT, INTEGER, BIGINT, REAL, DOUBLE, NUMERIC, FLOAT
 - CHAR(MAX 2000), VARCHAR(MAX 4000), BINARY, VARBINARY, LONG VARCHAR, LONG VAR BINARY
 - DATE, TIME, TIMESTAMP, INTERVAL (TIME/ TIMESTAMP는 WITH/ WITHOUT TIMEZONE 지원)
 - BLOB 타입은 지원하지 않는다.
- Column 수/ 인덱스 수/ constraint 수에는 제한이 없다.

- `SELECT * FROM TABLES;` 질의를 통해 전체 테이블 정보를 확인할 수 있다.

특정 테이블에 저장된 row가 많아져서 그 크기가 너무 커진 경우, bulk delete를 해도 빈 공간이 다시 테이블스페이스에 반납되지는 않는다. 반면 TRUNCATE 연산을 하면 기존의 공간이 모두 테이블스페이스로 반납된다.

GOLDILOCKS를 cluster system으로 구성하여 사용할 경우에는 사용자가 원하는 분배 정책에 따라 테이블의 data를 다수의 노드에 분산하여 저장하고 조회할 수 있다. 자세한 내용은 **Cluster System 테이블 관리**를 참조한다.

인덱스 관리

본 절에서는 인덱스 개요, 생성 및 삭제에 대해 설명한다.

인덱스 개요

인덱스는 테이블에 연계된 부가적 schema object이다. 인덱스를 사용하면 특정 조건에 해당하는 row의 위치를 찾는 성능이 향상되고 만일 얻으려는 column이 해당 인덱스의 key column일 경우 그 row의 column 값도 알아낼 수 있다.

GOLDILOCKS에서 인덱스는 필요한 만큼 테이블에 생성할 수 있다. 그러나 너무 많은 인덱스는 해당 테이블에 대한 삽입/ 변경/ 삭제 연산에 부하를 주어 성능 저하를 유발할 수도 있다.

Primary key나 unique constraint는 해당 column에 자동으로 인덱스를 생성한다.

인덱스 속성

- B-link tree 형태의 index를 지원한다.
 - GOLDILOCKS는 기본적으로 B-link tree 방식으로 구현된 index를 제공한다.
- Index node 한 개의 크기는 8 Kbyte이다.
- 최대 key column 개수는 32 개, 최대 key 길이는 2000 byte이다.
- Unique index를 지원한다.
- Ascending/ descending, NULLS FIRST, NULLS LAST를 지원한다.
 - 인덱스의 key column들을 작은 순서대로 sorting할지 (ASC), 큰 순서대로 sorting할지 (DESC) 정의할 수 있다.
 - 인덱스의 key column들에서 NULL 값들을 가장 앞에 나열할지 (NULLS FIRST) 가장 나중에 나열할지 (NULLS LAST) 정의할 수 있다.
- `SELECT * FROM INDEXES;` 질의를 통해 전체 인덱스 정보를 확인할 수 있다.
- 인덱스도 테이블과 동일하게 segment로 구현되므로 테이블과 유사한 방법으로 index 크기를 계산할 수 있다.

시퀀스

시퀀스는 사용자가 고유한 수를 생성하게 하는 schema 객체이다.

다음과 같이 생성한다.

```
gSQL> CREATE SEQUENCE customers_seq START WITH 1000 INCREMENT BY 1 NOCACHE NOCYCLE;  
Sequence created.
```

NEXTVAL을 이용하여 시퀀스를 사용할 수 있다.

```
gSQL> SELECT customers_seq.NEXTVAL FROM dual;
```

다음과 같이 시퀀스를 제거한다.

```
gSQL> DROP SEQUENCE customers_seq;
```

GOLDILOCKS를 cluster system으로 구성하여 사용할 경우에는 사용자가 시퀀스를 생성할 때 global 시퀀스 객체가 자동으로 생성된다. 이 객체는 cluster system에 속한 모든 member 노드가 공유하며 사용할 수 있는 시퀀스 값들의 global pool을 관리한다. 각 member 노드는 NEXTVAL을 호출할 때 global 객체로부터 정해진 개수만큼의 시퀀스 값들을 할당받아 사용한다. 자세한 내용은 **Global 시퀀스**를 참조한다.

2.5 사용자 관리

사용자 생성

GOLDILOCKS 데이터베이스의 사용자는 SYS 사용자와 CREATE USER ON DATABASE 권한을 가진 사용자만 생성할 수 있다. 또한 생성한 사용자에게 접속하기 위해서는 CREATE SESSION ON DATABASE 권한이 필요하다.

구문은 다음과 같다.

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE tablespace_name ]
```

구문 규칙과 파라미터는 다음과 같다.

- user_identifier
 - 생성할 user의 이름이다.
 - 동일한 사용자 이름이나 역할 이름이 존재하지 않아야 한다.
 - user_identifier의 길이는 128 byte 보다 작아야 한다.
- password
 - 생성할 user의 password로 암호화되어 저장된다.
 - Password의 길이는 128 byte보다 작아야 한다.
 - Password는 대소문자를 구분한다.
- DEFAULT TABLESPACE tablespace_name
 - 생성할 user가 생성하는 테이블, 인덱스 (LOGGING) 등의 객체가 저장될 기본 tablespace를 지정한다.
 - DEFAULT TABLESPACE 절을 생략할 경우, DATABASE를 생성할 때 (<database definition>) 정의한 default data tablespace (MEM_DATA_TBS)로 지정된다.
- TEMPORARY TABLESPACE tablespace_name
 - 생성할 user가 생성하는 임시 테이블, 인덱스 (NO LOGGING), 질의 처리 과정에서 생성하는 중간 결과들을 저장할 TABLESPACE를 지정한다.
 - TEMPORARY TABLESPACE 절을 생략할 경우, DATABASE를 생성할 때 (<database definition>) 정의한 default temporary tablespace (MEM_TEMP_TBS)로 지정된다.
- INDEX TABLESPACE tablespace_name
 - 생성할 user가 생성하는 인덱스 객체가 저장될 기본 tablespace를 지정한다.
 - INDEX TABLESPACE 절을 생략할 경우, 기본값은 INDEX TABLESPACE NULL이다.

사용자 제거

GOLDILOCKS 데이터베이스에서 생성한 사용자를 제거한다. 사용자 범위와 접근 권한은 사용자 생성과 동일하다.

구문은 다음과 같다.

```
<drop user statement> ::=
    DROP USER [ IF EXISTS ] user_identifier [ <drop behavior> ]
    ;
<drop behavior> ::=
    RESTRICT
    | CASCADE
```

구문 규칙과 파라미터는 다음과 같다.

- IF EXISTS
 - 사용자가 존재하지 않더라도 에러가 발생하지 않는다.
- user_identifier
 - 제거할 데이터베이스 사용자의 이름이다.
 - Database를 생성할 때 자동으로 생성되는 SYS와 같은 사용자는 제거할 수 없다.
 - Tablespace 객체 등과 같이 user_identifier가 생성했지만 소유자가 아닌 객체는 제거하지 않는다.
- drop behavior
 - 생략할 경우, 기본값은 RESTRICT 이다

사용자 변경

GOLDILOCKS 데이터베이스에서 생성한 사용자의 정의를 변경한다. 사용자에게 ALTER USER의 권한이 필요하며 user_identifier와 동일한 사용자일 경우 권한없이 수행할 수 있다.

구문은 다음과 같다.

```
<alter user statement> ::=
    ALTER USER user_identifier <alter user action>
    | ALTER USER PUBLIC <alter schema path>
    ;
<alter user action> ::=
    <alter password>
    | <alter profile>
    | <alter default tablespace>
    | <alter temporary tablespace>
```

```

| <alter index tablespace>
| <alter schema path>
<alter password> ::=
    IDENTIFIED BY new_password [ REPLACE old_password ]
<alter profile> ::=
    PROFILE { profile_name | DEFAULT | NULL }
<password expire> ::=
    PASSWORD EXPIRE
<account lock> ::=
    ACCOUNT { LOCK | UNLOCK }
<alter default tablespace> ::=
    DEFAULT TABLESPACE tablespace_name
<alter temporary tablespace> ::=
    TEMPORARY TABLESPACE tablespace_name
<alter index tablespace> ::=
    INDEX TABLESPACE { tablespace_name | NULL }
<alter schema path> ::=
    SCHEMA PATH ( { schema_name | CURRENT PATH } [, ...] )

```

구문 규칙과 파라미터는 다음과 같다.

- user_identifier
 - 변경할 사용자 이름이다.
- <alter password>
 - 사용자의 password를 변경한다.
 - IDENTIFIED BY new_password
 - 새로운 password로 암호화되어 저장된다.
 - Password의 길이는 128 byte보다 작아야 한다.
 - Password는 대소문자를 구별한다.
 - REPLACE old_password
 - ALTER USER ON DATABASE 권한이 있을 경우에는 생략할 수 있다.
 - ALTER USER ON DATABASE 권한이 없을 경우에는 생략할 수 없다.
 - 사용자와 user_identifier가 동일해야 한다.
- <alter profile>
 - 비밀번호 관리 정책을 위한 profile을 변경한다.
 - PROFILE profile_name
 - 사용자가 생성한 profile_name을 할당한다.
 - PROFILE DEFAULT
 - 기본 profile "DEFAULT"를 할당한다.
 - PROFILE NULL
 - Profile을 할당하지 않는다.

- <password expire>
 - 사용자의 비밀번호를 만료시킨다.
- <account lock>
 - ACCOUNT LOCK
 - 사용자 계정을 잠근다.
 - ACCOUNT UNLOCK
 - 계정 잠금을 해제한다.
- <alter default tablespace>
 - 사용자의 기본 tablespace를 변경한다.
 - tablespace_name은 data tablespace이어야 한다.
- <alter temporary tablespace>
 - 사용자의 temporary tablespace를 변경한다.
 - tablespace_name은 temporary tablespace이어야 한다.
- <alter index tablespace>
 - 사용자의 index tablespace를 변경한다.
 - INDEX TABLESPACE tablespace_name을 지정한다.
 - Data tablespace을 지정할 경우, LOGGING 인덱스이어야 한다.
 - Temporary tablespace를 지정할 경우, NOLOGGING 인덱스이어야 한다.
 - INDEX TABLESPACE NULL
 - Index tablespace를 지정하지 않는다.

2.6 GOLDILOCKS 프로퍼티

GOLDILOCKS 프로퍼티는 데이터베이스를 생성할 때 적용되는 프로퍼티와 online/ offline 중에 변경할 수 있는 프로퍼티로 나뉜다. 테이블스페이스와 redo log file이 저장되는 경로는 startup 단계들 중에 MOUNT 단계에서만 변경할 수 있다.

데이터베이스 생성 시 적용되는 프로퍼티

표 2-28 데이터베이스 생성 시 적용되는 프로퍼티

이름	설명
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	딕셔너리 테이블스페이스의 초기 사이즈
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	시스템 데이터 테이블스페이스의 초기 사이즈
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	시스템 undo 테이블스페이스의 초기 사이즈
SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	시스템 임시 테이블스페이스의 초기 사이즈
LOG_BLOCK_SIZE	Redo log file의 블록 사이즈
LOG_FILE_SIZE	Redo log file의 초기 사이즈
LOG_GROUP_COUNT	Redo log file의 개수
CHARACTER_SET	Character set
TIMEZONE	타임 존
CHAR_LENGTH_UNITS	캐릭터 길이 단위

데이터베이스 기동 시 적용되는 프로퍼티

GOLDILOCKS 데이터베이스에는 100개 이상의 프로퍼티들이 존재한다. 그 중에서 자주 사용되는 프로퍼티들은 다음과 같다.

표 2-29 데이터베이스 기동 시 적용되는 프로퍼티

이름	설명
SHARED_MEMORY_STATIC_KEY	공유 메모리가 생성되는 키값
SHARED_MEMORY_STATIC_SIZE	생성되는 공유 메모리의 사이즈
DATA_STORE_MODE	GOLDILOCKS 인스턴스의 저장모드
LOG_BUFFER_SIZE	로그 버퍼 사이즈
LOG_DIR	Redo log 디렉토리 경로
PRIVATE_STATIC_AREA_SIZE	세션별 정적 영역의 크기
CLIENT_MAX_COUNT	접속할 수 있는 최대 세션 개수
PROCESS_MAX_COUNT	최대 프로세스 개수

이름	설명
NET_BUFFER_SIZE	세션별 통신 버퍼 크기

2.7 GOLDILOCKS Utility

gcreatedb

gcreatedb 유틸리티는 GOLDILOCKS database를 초기화하고 서비스 할 준비 작업을 한다. gcreatedb는 data file, log file 등을 주어진 property에 따라 GOLDILOCKS_DATA 환경 변수의 위치에 생성한다. 구문은 다음과 같다.

```
[SHELL]> gcreatedb --help
```

Usage

```
gcreatedb [options]
```

Options:

```
--cluster      cluster system (if not specified, stand-alone system)
--db_name       database name
--db_comment    database comment
--timezone      timezone ( {+/-}{TZH:TZM} )
--character_set character set
                  SQL_ASCII
                  UTF8
                  UHC
                  GB18030
--char_length_units char length units
                  OCTETS
                  CHARACTERS
--home          home directory
--member        local member name
--host          host address
--port          host port
--silent        suppresses the display of the result message
--help          print help message
```

examples:

```
gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00" --
character_set="UTF8" --char_length_units="OCTETS" --silent
```

데이터베이스를 생성할 때 \$GOLDILOCKS_HOME/conf/goldilocks.properties.conf를 참조하여 각각의 ***_TABLESPACE_SIZE 값으로 goldilocks.properties.conf의 SYSTEM_TABLESPACE_DIR 경로에 tablespace file들을 생성한다.

다음은 gcreatedb 명령어의 실행 인자에 대한 설명이다.

표 2-30 gcreatedb 인자

실행 인자	설명
--cluster	Cluster system에서 사용될 database이다. 생략할 경우, standalone database를 생성한다.
--db_name	데이터베이스 이름이다. 생략할 경우, goldilocks로 설정된다.
--db_comment	데이터베이스에 대한 설명이다. 생략할 경우, goldilocks database로 설정된다.
--timezone	타임 존이다. 생략할 경우, TIMEZONE 프로퍼티로 설정된다.
--character_set	데이터베이스의 character set이다. GOLDILOCKS는 다음과 같은 네 가지 character set을 지원한다. • GB18030: 중국어 간체 • SQL_ASCII: ASCII 문자만 지원하는 character set • UHC: 통합형 한글 코드 (Unified Hangul Code) • UTF8: Unicode Transformation Format - 8 생략할 경우, CHARACTER_SET 프로퍼티로 설정된다.
--char_length_units	캐릭터 길이의 단위이다. • OCTETS: 1 바이트를 한 글자로 인식한다. • CHARACTERS: 한 글자 (n 바이트)를 한 글자로 인식한다. 생략할 경우, CHAR_LENGTH_UNITS 프로퍼티로 설정된다.
--home	Database home 디렉토리이다. 프로퍼티 파일을 찾아서 각종 database 파일들의 생성 및 저장 위치로 참조한다. 생략할 경우, GOLDILOCKS_DATA 환경 변수에 설정된 값을 사용한다.
--member	Cluster system 내에서 사용할 local database의 member 이름이다. 생략할 경우, LOCAL_CLUSTER_MEMBER 프로퍼티에 설정된 값을 사용한다.
--host	Cluster system member 간의 통신을 위해 사용할 local member의 IP address이다. 생략할 경우, LOCAL_CLUSTER_MEMBER_HOST 프로퍼티에 설정된 값을 사용한다.
--port	Cluster system member 간의 통신을 위해 사용할 local member의 TCP listen port이다. 생략할 경우, LOCAL_CLUSTER_MEMBER_PORT 프로퍼티에 설정된 값을 사용한다.
--silent	출력 메시지를 보이지 않게 한다.
--help	도움말을 출력한다.

gsq (GOLDILOCKS interactive sql tool)

gsq는 GOLDILOCKS를 관리하기 위해 임의의 SQL을 실행시키는 interactive command line 유틸리티이다. DBA는 gsql을 이용해 초기 table schema를 생성하거나 현재 상태를 점검하는 등의 작업을 수행할 수 있다.

gsq 실행 방법은 다음과 같다.

```
[SHELL]> gsql <userid> <passwd>
gSQL> CREATE TABLE T1 ( COL1 INTEGER );
create success
gSQL> \q
[SHELL]>
```

gloader (GOLDILOCKS data upload/ download tool)

gloader는 기존의 database에 존재하는 데이터들을 text 형식으로 파일로 내리거나 기존에 text 파일로 저장된 데이터들을 새 database로 loading하는 유틸리티이다. gloader가 사용하는 text data file 형식은 Comma-Separated Value (CSV) 이다.

gloader 사용법은 다음과 같다.

```
gloader userid passwd [export|import] control='control_file_name' \
data='data_file_name' log='log_file_name' bad='bad_file_name'
```

표 2-31 gloader 인자

실행 인자	설명
export import	gloader를 통해 기존 테이블의 내용을 data_file_name으로 내려받을 것인지, data_file_name에 현재 존재하는 데이터들을 control_file_name에 명시된 테이블로 올릴 것인지를 선언한다.
userid	사용할 userid를 명시한다.
passwd	userid 사용자의 비밀번호를 명시한다.
control	export나 import 작업에 대한 상세 설정을 기록한 파일 경로를 명시한다.
data	export 될 target data file이나 import 할 data file을 명시한다.
log	import나 export의 진행 상황 및 경과 시간 등을 기록하는 log file 경로를 명시한다.
bad	import 할 때 여러가지 오류로 인해 insert에 실패한 data record들을 모아서 기록하는 bad file 경로를 명시한다.

다음은 control file 내용의 예이다.

```
TABLE TEST_TBL
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY ''''
```


3.

Cluster 튜토리얼

3.1 GOLDDILOCKS Cluster System 관리

본 장에서는 다수의 GOLDDILOCKS database를 이용하여 cluster system을 구성하고 관리하기 위한 기반 지식에 대해 설명한다. 본 장의 내용은 standalone database와 비교하여 cluster system에 추가된 부분이나 cluster system 만 가지는 특징 위주로 설명하였으므로 이를 이해하기 위해서는 standalone database에 대한 튜토리얼을 사전에 숙지해야 한다.

Overview

GOLDDILOCKS는 앞서 살펴본 바와 같이 단독 (standalone) database로 구성하여 사용할 수 있을 뿐만 아니라 다수의 database를 하나의 클러스터 (cluster)로 묶은 다음 서비스에 적합한 data 분배 정책을 사용자가 직접 선택하여 사용할 수도 있다. 즉, GOLDDILOCKS cluster system을 사용하면 사용자가 대용량 데이터를 다수의 서버에 원하는 방식으로 분산하여 처리할 수 있으므로 서비스 가용성이 높아지고 병렬처리를 통해 처리량이 개선된다.

GOLDDILOCKS cluster system은 하나 이상의 cluster group으로 구성되며 하나의 cluster group은 하나 이상의 cluster member로 구성된다. 별도의 응용 프로그램 server나 meta server를 필요로 하지 않으며 응용 프로그램들은 data server에 해당하는 cluster member에 접속하여 동작한다. 동일한 cluster group에 소속된 cluster member들은 동일한 data 복제본 (replica)을 유지한다.

GOLDDILOCKS database를 standalone으로 사용할지 또는 cluster system으로 사용할지 여부는 각 노드의 database를 생성할 때 미리 결정해야 한다. 만약 cluster system으로 사용하려면 사용자가 각 노드에서 database를 생성할 때 cluster 관련 옵션을 추가해야 한다.

프로퍼티 설정

Cluster system을 구축하기 위한 각종 프로퍼티들은 standalone database로 사용할 때와 마찬가지로 각 서버의 \$ GOLDDILOCKS_DATA/conf/golddilocks.property.conf 파일에 기술되어 있다. Cluster system을 구성하여 사용하더라도 각각의 database를 위한 TBS (tablespace), LOG, CONTROL FILE 등에 관한 주요 프로퍼티는 standalone과 동일한 방식으로 설정하면 된다. 단, 동일한 서버에서 다수의 database를 생성하여 cluster system으로 구성할 때는 각 파일들의 PATH와 port 등이 cluster member 간에 중복되지 않게 유의해야 한다.

다음은 cluster system을 구성할 때 고려해야 하는 주요 프로퍼티 항목과 설명이다.

표 3-1 주요 프로퍼티

프로퍼티	설명	기본값
	시스템 TBS들이 저장되는 디렉토리 경로이다. 해당 경로에는 다음과 같은 TBS들이 설치된다. • DICTIONARY_TBS	

프로퍼티	설명	기본값
SYSTEM_TABLESPACE_DIR	- MEM_DATA_TBS - MEM_UNDO_TBS - MEM_TEMP_TBS - MEM_TRANS_TBS	'<GOLDILOCKS_DATA>/db'
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	딕셔너리 테이블스페이스의 크기이다.	256M
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	데이터 테이블스페이스의 크기이다.	200M
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	Undo 테이블스페이스의 크기이다.	32M
LOG_DIR	기본 로그 디렉토리의 경로이다.	'<GOLDILOCKS_DATA>/wal'
SYSTEM_LOGGER_DIR	시스템 로그 디렉토리의 경로이다.	'<GOLDILOCKS_DATA>/trc'
CONTROL_FILE_COUNT	컨트롤 파일의 개수이다.	2
CONTROL_FILE_0	첫 번째 컨트롤 파일의 경로이다.	'<GOLDILOCKS_DATA>/wal/control_0.ctl'
CONTROL_FILE_1	두 번째 컨트롤 파일의 경로이다.	'<GOLDILOCKS_DATA>/wal/control_1.ctl'
LOCAL_CLUSTER_MEMBER	Local 서버가 cluster system 내에서 사용할 cluster member의 이름이다.	'G1N1'
LOCAL_CLUSTER_MEMBER_HOST	Local 서버의 host name이다.	'127.0.0.1'
LOCAL_CLUSTER_MEMBER_PORT	Local 서버가 cluster system 내에서 통신하기 위해 사용할 TCP listen port이다.	10101

Cluster member 이름과 host-port 조합은 cluster system 내에서 유일해야 한다. 위의 프로퍼티 설정을 생략하려면 각 노드에서 gcreatedb를 이용하여 database를 생성할 때 --member, --host, --port 옵션을 이용하여 해당 정보를 부여한다. 프로퍼티 또는 gcreatedb 옵션을 통해 부여된 member 정보는 내부적으로 \$GOLDILOCKS_DATA/wal/location.ctl 파일에 저장되어 관리된다.

프로퍼티를 변경하려면 text 프로퍼티 파일 (\$GOLDILOCKS_DATA/conf/goldilocks.properties.conf)을 변경하거나 환경 변수에 GOLDILOCKS_<property_name> 형태의 변수를 새로 정의하면 된다. 프로퍼티 파일에 설정된 값이 환경 변수에 설정된 값보다 우선한다.

Background 프로세스

GOLDILOCKS cluster system은 각 member 노드에 instance를 관리하기 위한 background 프로세스(gmaster)를 가지고 있다. 각 노드의 gmaster는 내부적으로 여러 개의 system thread로 구성되어 있고 대부분의 thread는 standalone과 동일하지만 다음과 같은 system thread들이 cluster system 관리를 위해 추가되었다.

다음의 thread들은 cluster 모드로 생성된 database를 시작할 경우에만 기동된다.

- Cluster recover thread: Cluster system에서 global transaction을 복구한다.
- Failover thread: Cluster system에서 특정 노드나 네트워크에 장애가 발생했을 때 장애 member에 대한 offline 및 coordinator 재선정과 같은 failover를 처리한다.

다음은 GOLDDILOCKS를 cluster system으로 구성하여 사용할 경우에 추가적으로 기동되는 프로세스들이다.

- cdispatcher: Cluster 패킷의 송수신 및 세션 관리 등을 담당한다.
- cserver: 자신이 속한 member 노드의 database 변경 및 조회 연산을 수행한다.

GOLDDILOCKS cluster system에서는 각 member 노드 간에 복잡한 cluster protocol 통신이 필요한데 cluster dispatcher (cdispatcher)는 이를 위해 네트워크 통신 컨텍스트 관리 및 패킷 분배 메커니즘을 효율적으로 수행하는 프로세스이다. 또한, heartbeat 등을 통해 cluster 세션의 유효성을 지속적으로 모니터링하는 역할을 수행하기도 한다.

Cluster system에서는 특정 노드 (driver 노드)에서 수행하는 SQL이 대상 테이블의 sharding 정책을 참고하여 원격 member 노드에 data를 저장하거나 해당 노드에 저장된 data를 조회해야 할 때가 있다. 이 때 각 member 노드에서는 원격으로부터의 이런 요청을 수행하여 결과를 반환하는 프로세스가 필요한데 이러한 역할을 수행하는 프로세스가 cserver이다.

Client 프로세스

Standalone과 마찬가지로 사용자는 cluster system에서도 Client/ Server (C/S) 모델과 Direct Access (D/A) 모델을 모두 사용할 수 있다. 다만 cluster system의 각 member 노드는 자신만의 listener를 가지고 있으므로 client 프로그램은 (C/S 모드로 사용 시) 접속하고자 하는 cluster member의 listen port를 사전에 인지하고 있어야 한다. Cluster system의 어떤 노드에 접속하든 사용자는 standalone database와 동일하게 다양한 형태의 transaction을 처리할 수 있다.

그 밖에 signal handling이나 connection에 대한 cleanup, 각종 공유 자원 해제 등은 standalone과 cluster system에서 동일한 방식으로 처리된다.

Instance의 메모리 구조

GOLDDILOCKS cluster system은 다수의 shared-nothing database를 cluster system이라는 하나의 관리 단위로 묶어서 사용하기 위한 것이다. 따라서 각 cluster member 노드의 메모리 사용 방식은 standalone과 매우 유사하다. 각 member 노드가 사용하는 메모리 크기는 자신의 프로퍼티 파일에 지정된 관련 프로퍼티들에 의해 결정된다. Static 영역은 standalone과 동일하게 database를 운용하기 위한 instance 기본 정보들과 각 session, statement 및 transaction 정보들, redo log buffer, dictionary cache 정보, 그리고 그 외 여러가지 운용 정보들을 담고 있다. 이런 기본 운용 정보 외에 location 정보와 cluster session 정보 등과 같은 cluster system 관리를 위한 추가 정보들이 저장된다.

테이블스페이스 영역은 각 테이블스페이스의 내용을 담은 page frame들과 이를 제어하기 위한 Page Control Hea

der (PCH)들로 구성되어 있다.

응용 프로그램 프로세스의 메모리는 connect 할 때 attach한 instance 메모리 이외에 프로세스 단위로 공유하는 ODBC environment와 여러 ODBC handle들, 그리고 bind 정보 등과 같은 기타 정보들을 포함하고 있는 heap 메모리 영역을 추가적으로 포함하고 있다.

Cluster System 시작과 종료

GOLDILOCKS system을 시작하려면 사전에 각 member 노드에 instance를 생성 (gcreatedb)한 후에 cluster group과 cluster member를 등록해야 한다. 이후 gsql과 gsqlnet을 통해 sysdba role을 이용하여 system을 시작하고 종료할 수 있다.

C/S 모델의 dedicated 모드에서 GOLDILOCKS cluster system을 시작하거나 종료하려고 할 경우, 즉 gsqlnet 을 이용하고자 할 경우에는 **listener**가 실행되어 있어야 한다.

C/S 모델의 shared 모드에서는 GOLDILOCKS cluster system을 시작하거나 종료할 수 없다.

```
% gsql --as sysdba
Enter user-name: sys
Enter password:
Connected to an idle instance.
gSQL>
```

GOLDILOCKS cluster system은 다음과 같은 startup phase를 갖는다. Standalone과 비교할 때 OPEN 상태가 LOCAL OPEN과 GLOBAL OPEN 상태로 세분화되었다.

- NOMOUNT
 - GOLDILOCKS instance를 관리하는 데몬인 gmaster 프로세스를 띄운다
- MOUNT
 - \$GOLDILOCKS_DATA 환경 변수를 이용하여 프로퍼티들과 복구를 위한 control file을 읽어들인다.
- LOCAL OPEN
 - Data file들로부터 테이블스페이스의 내용을 로딩한 후에 redo log file들을 이용하여 복구하고 in-doubt global transaction을 복구하며 no-logging index들을 새로 build하고 dictionary cache를 생성한다.
- OPEN (GLOBAL OPEN)
 - 각 cluster member 간의 cluster session을 연결하고 shard map 등을 정리하며 글로벌 관리자 (global coordinator) 및 그룹 관리자 (group coordinator)를 선정한 후, 최종적으로 사용자의 서비스 접속을 기다린다.

GOLDILOCKS cluster system을 시작하거나 종료하기 위해서는 기본적으로 다음과 같은 사전 작업을 통해 cluster system 환경을 구성해 두어야 한다. gcreatedb를 이용하여 각 member database를 생성할 때 member 이름, host 주소, port 번호를 고유하게 부여할 경우, 각 노드의 property 설정 과정을 생략할 수 있다.

- Property 설정: 각 member 노드에서 사용할 property file을 수정한다.
- Database 생성: gcreatedb를 통해 각 member 노드에 database를 생성한다.
- Cluster group 생성 및 member 추가: Cluster system을 구성할 group과 member를 추가한다.
- Listener 기동: gsqlnet을 이용하기 위해서는 각 노드에서 listener를 미리 기동해 두어야 한다.

다음 구문을 사용하여 cluster group과 member를 생성한다.

- **ALTER CLUSTER GROUP name ADD MEMBER**
- **CREATE CLUSTER GROUP**
- Database 생성: 각 노드에서 수행한다.

```
% gcreatedb --cluster --db_name='goldilocks' --member='g1n1' \
    --host='192.168.0.11' --port 10110
% gcreatedb --cluster --db_name='goldilocks' --member='g1n2' \
    --host='192.168.0.12' --port 10120
```

- Cluster group과 member 생성: 한 노드에서 수행한다.

```
gSQL> create cluster group g1 cluster member g1n1
    host '192.168.0.11' port 10110;
gSQL> alter cluster group g1 add cluster member g1n2
    host '192.168.0.12' port 10120;
```

위와 같이 GOLDILOCKS cluster system 구성을 완료하면 다음 두 가지 방법을 이용하여 cluster system 전체를 시작하거나 종료할 수 있다.

- 각 member 노드에 접속하여 하나씩 시작하고 종료하는 방법
 - \startup 및 \shutdown 명령을 이용한다.
- 하나의 member 노드에서 전체 member 노드를 동시에 시작하고 종료하는 방법
 - gsqlnet을 통해 접속한 후에 \cstartup 및 \cshutdown 명령을 이용한다.

다음은 위의 첫 번째 방법대로 각 member 노드에 접속하여 cluster system을 기동하는 예이다. \startup 명령은 중간 과정없이 곧바로 LOCAL OPEN 단계까지 진입할 때 사용하며, 이는 \startup nomount, alter system mount database, alter system open local database의 세 단계로 분리하여 수행할 수도 있다.

\startup을 통해 각 노드에서 LOCAL OPEN 단계까지 기동한 후, 최종적으로 하나의 노드에 접속하여 GLOBAL OPEN 단계까지 기동하면 된다.

- LOCAL OPEN까지 startup: 각 member 노드에서 수행한다.

```
% gsql sys gliese --as sysdba
gSQL> \startup
Startup success.
```

- OPEN까지 startup: 한 노드에서 수행한다.

```
% gsql sys gliese --as sysdba
gSQL> alter system open global database;
System altered.
```

Cluster system에 포함된 노드의 개수가 많을 경우에 위의 첫 번째 방법으로 startup을 진행하면 운영자 작업에 부담이 될 수 있다. Cluster system이 종료된 후 재기동할 경우, 모든 member 노드에서 startup 할 때마다 LOCAL OPEN까지의 작업 과정을 수행해야 하기 때문이다. 운영상의 편의를 위해 다음과 같이 간단한 방법으로 모든 member를 한 번에 startup 할 수 있다.

- GLOBAL OPEN까지 startup: 한 노드에서 수행한다.

```
% gsqlnet sys gliese --as sysdba
gSQL> \cstartup
Startup success.
```

노트

- \cstartup 및 \cshutdown 명령어는 gsqlnet에서만 수행 가능하며 gsql에서는 지원하지 않는다. 그리고 기동하려는 모든 member 노드에 listener가 사전에 기동되어 있어야 한다.
- 하나의 물리적 노드에서 다수의 member가 포함된 GOLDILOCKS cluster system을 구축하고자 할 경우에는 각 database를 생성할 때 home directory 및 member 이름, cluster port 정보가 중복되지 않도록 주의해야 한다.

GOLDILOCKS system이 종료되면 각 member 노드에서 관리 데몬 프로세스인 gmaster가 종료되어 더 이상의 접속이나 기타 database 작업을 수행할 수 없다.

GOLDILOCKS cluster system의 종료 모드에는 다음과 같은 두 가지 종류가 있다.

- NORMAL: 새로운 session의 접속을 차단하고 현재 접속된 모든 session이 종료될 때까지 기다린 후에 checkpoint를 수행하고 instance를 내린다.
- ABORT: 접속 중인 session들의 상태와 관계없이 gmaster를 즉시 종료시켜 instance를 내린다.

다음과 같이 gsql을 이용하여 \shutdown 명령을 수행하면 접속한 member의 노드만 종료된다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown normal
Shutdown success
gSQL>
```

Cluster system에 속한 모든 member를 한 번에 종료시키고자 할 경우에는 다음과 같이 gsqlnet의 \cshutdown 명령을 이용한다. normal 옵션은 생략할 수 있다.

```
% gsqlnet sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \cshutdown normal
Shutdown success
gSQL>
```

주의

\cshutdown을 수행할 때 abort 옵션을 사용할 경우, 모든 member 노드들에 대해 gmaster를 강제로 종료 시키므로 \cstartup을 이용하여 다시 기동할 때는 종료 시점의 노드 상태에 따라 일부 member 노드가 cluster system 참여 (join)에 실패할 수도 있다. 물론 다음에 살펴볼 join 명령어와 rebalance 과정을 통해 다시 참여시킬 수는 있지만 특별한 경우를 제외하고는 되도록 \cshutdown normal을 이용하여 종료하는 것이 안전하다.

Cluster system에 속한 하나의 member 노드 또는 일부의 member 노드를 종료하거나 시작하고자 할 경우에는 다음 과정을 따른다. 다음은 cluster member 노드 중에 G1N2라는 이름을 가진 member 노드만 재시작한 후에 cluster system에 다시 참여시키는 과정이다.

```
% gsql sys gliese --as sysdba --dsn=g1n2
Connected to GOLDILOCKS Database.
gSQL> \shutdown normal
Shutdown success
gSQL> \startup
Startup success
gSQL> alter system join database;
System altered.
```

Transaction이 발생하는 상황에서 member 노드를 종료할 경우, 해당 노드를 재시작한 후에 cluster system에 다시 참여 (join) 시키려면 변경이 발생한 테이블들에 대한 리밸런싱 (rebalancing) 작업이 필요할 수 있다. 만약 리밸런싱 작업을 수행하지 않으면 driver 노드에서 수행하는 transaction이 리밸런싱되지 않은 테이블에 대한 변경을 수행할

때 실패할 수 있다.

리밸런싱 작업이란 테이블의 데이터 분배 정책 및 속성 정보를 member 노드간에 동기화하고, 테이블 데이터를 데이터 분배 정책에 따라 각 member 노드에 다시 분할 저장하는 일련의 과정이다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown normal
Shutdown success
gSQL> \startup
Startup success
gSQL> alter system join database;
ERR-42000(16405): some tables in the database need to be rebalanced
System altered.
gSQL> alter database rebalance;
Database altered.
```

3.2 GOLDDILOCKS 설치와 Database 생성

GOLDDILOCKS cluster system에 포함될 member database 설치 및 생성 방법은 standalone과 대부분 동일하다. 다음 database 설치 및 생성 과정에서는 standalone과 비교하여 cluster system만 가지는 특징과 방법상의 차이를 설명한다.

GOLDDILOCKS Package 구성

GOLDDILOCKS cluster system을 구성하기 위해 사용하는 package는 standalone database를 구성하기 위한 것과 동일하다. 다만 dictionary와 performance view 등을 생성하기 위한 스크립트는 standalone과 cluster 용이 다음과 같이 분리되어 있으므로 database를 생성한 후에는 반드시 각각의 용도에 맞는 스크립트를 이용하여 생성해야 한다.

Standalone database를 위한 스크립트는 \$GOLDDILOCKS_HOME/admin/standalone 디렉토리 아래에 위치하고 cluster system을 위한 스크립트는 \$GOLDDILOCKS_HOME/admin/cluster 디렉토리 아래에 위치한다.

표 3-2 admin/ standalone 디렉토리

파일 이름	설명
README	read me
DictionarySchema.sql	Dictionary schema 생성 스크립트
InformationSchema.sql	Information schema 생성 스크립트
PerformanceViewSchema.sql	PerformanceView schema 생성 스크립트

표 3-3 admin/ cluster 디렉토리

파일 이름	설명
README	read me
DictionarySchema.sql	Dictionary schema 생성 스크립트
InformationSchema.sql	Information schema 생성 스크립트
PerformanceViewSchema.sql	PerformanceView schema 생성 스크립트

GOLDDILOCKS Software 설치

Cluster system에 포함되는 모든 member 노드에 GOLDDILOCKS software를 설치해야 하며 설치 방법은 standalone과 동일하다. 커널 파라미터의 설정 및 확인 방법, 환경 변수 설정 방법은 standalone과 동일하므로 해당 튜토리얼을 참고한다.

이 때, cluster member 노드 각각에 대한 property 설정이나 database 이름, database 버전, character set, time

zone 등의 설정이 동일해야만 cluster system을 정상적으로 구성할 수 있다.

Database 생성

Cluster system을 구성하는 각 member 노드에서 database를 생성하려면 standalone과 마찬가지로 gcreatedb utility를 사용한다.

다음은 cluster database를 생성할 때만 사용되는 gcreatedb 옵션이다. 이외의 옵션은 standalone과 동일하다.

표 3-4 gcreatedb 인자

실행 인자	설명
--cluster	Cluster database라는 것을 나타낸다. 생략할 경우, standalone database를 생성한다.
--member	Cluster system 내에서 사용할 local database의 member 이름이다. 생략할 경우, LOCAL_CLUSTER_MEMBER 프로퍼티에 설정된 값을 사용한다.
--host	Cluster system member 간에 통신하기 위해 사용할 local member의 IP address이다. 생략할 경우, LOCAL_CLUSTER_MEMBER_HOST 프로퍼티에 설정된 값을 사용한다.
--port	Cluster system member 간에 통신하기 위해 사용할 local member의 TCP listen port이다. 생략할 경우, LOCAL_CLUSTER_MEMBER_PORT 프로퍼티에 설정된 값을 사용한다.

다음은 cluster system에서 사용될 데이터베이스를 생성하는 예이다.

```
[SHELL]> gcreatedb --cluster
Database created
[SHELL]> gcreatedb --cluster --member=G1N1
Database created
[SHELL]> gcreatedb --cluster --member=G1N1 --host=127.0.0.1 --port 10101
Database created
[SHELL]> gcreatedb --cluster \
--db_name="TEST_DB" \
--home=$GOLDILOCKS_DATA \
--host=127.0.0.1 \
--port=10101 \
--db_comment="g1n1 db comment" \
--timezone="+09:00" \
--character_set="UHC" \
--char_length_units="OCTETS"
Database created
[SHELL]> ls $GOLDILOCKS_DATA/db
system_data.dbf system_dict.dbf system_trans.dbf system_undo.dbf
```

Dictionary Schema 정보 구축

Standalone과 마찬가지로 cluster system을 정상적으로 사용하기 위해서는 dictionary 스키마 정보를 구축해야 한다. 아래 스키마들을 구축하지 않을 경우 객체의 구조 정보를 획득하는 ODBC, JDBC의 catalog API (예: SQLTables () 함수 등)들이 오작동하여 third party tool들과 연동하지 않을 수 있으므로 database를 생성한 후에 반드시 구축해 주어야 한다.

Cluster system에서 스키마를 구축할 때는 되도록 cluster group과 member를 모두 생성한 후에 작업하는 것이 좋다. 이처럼 cluster system 구성을 완료한 후에는 하나의 member 노드에 접속하여 생성 스크립트를 수행할 때, GOLDILOCKS에서 자동으로 모든 member 노드에 스키마를 생성하기 때문이다.

- **DICTIONARY_SCHEMA**: DBA_*, ALL_*, USER_* 등의 객체 정보를 조회하는 view와 table로 구성된다.
- **INFORMATION_SCHEMA**: SQL 표준의 INFORMATION_SCHEMA에 포함되는 view와 table로 구성된다.
- **PERFORMANCE_VIEW_SCHEMA**: Fixed table들의 정보를 조합하여 시스템 정보를 조회하는 view로 구성된다.

다음은 cluster system 용 스크립트를 이용하여 스키마들을 구축하는 방법을 보여준다. 위에서 언급한대로 GLOBAL OPEN 상태에서 하나의 member 노드에 접속하여 수행하면 된다.

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.sql
```

위에서 구축한 스키마 정보를 이용하여 cluster system의 다양한 구조 정보들을 조회할 수 있는데, 조회 할 때 객체 뒤에 group 및 member 이름을 부여하여 사용자가 조회하고 싶은 노드의 정보만 쉽게 추출할 수 있는 것이 standalone과의 차이이다.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL> select origin_member_name, stat_name, stat_value
2   from gv$system_mem_stat
3   where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
ORIGIN_MEMBER_NAME  STAT_NAME              STAT_VALUE
-----
G1N1                 PLAN_CACHE_TOTAL_SIZE  17844952
G2N2                 PLAN_CACHE_TOTAL_SIZE  16796288
G2N1                 PLAN_CACHE_TOTAL_SIZE  16796288
G1N2                 PLAN_CACHE_TOTAL_SIZE  16796288
G3N1                 PLAN_CACHE_TOTAL_SIZE  16796288
G3N2                 PLAN_CACHE_TOTAL_SIZE  16796288
6 rows selected.
```



```
gSQL> select origin_member_name, stat_name, stat_value
  2   from gv$system_mem_stat@g1n2
  3   where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
```

ORIGIN_MEMBER_NAME	STAT_NAME	STAT_VALUE

G1N2	PLAN_CACHE_TOTAL_SIZE	16796288

1 row selected.

```
gSQL> select origin_member_name, stat_name, stat_value
  2   from gv$system_mem_stat@g1
  3   where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
```

ORIGIN_MEMBER_NAME	STAT_NAME	STAT_VALUE

G1N1	PLAN_CACHE_TOTAL_SIZE	17844952

G1N2	PLAN_CACHE_TOTAL_SIZE	16796288

2 row selected.

3.3 Schema Object 관리

Schema object는 사용자가 생성한 논리적 구조물들이다. GOLDBLOCKS cluster system은 table, index, global sequence와 같은 객체를 지원한다. 본 장에서는 standalone과 비교하여 cluster system이 가지는 각 schema object들의 특징을 살펴 보고, 이를 효율적으로 관리하기 위한 방법을 설명한다.

테이블 관리

GOLDBLOCKS cluster system에서 테이블을 생성할 경우, 사용자는 다음 네 가지 sharding 정책 중 하나를 옵션으로 지정할 수 있다. Sharding 정책은 cluster system에서 테이블의 data를 각 cluster group에 어떤 방식으로 분배하여 저장할지를 결정한다. 이는 cluster system에서만 지정할 수 있는 옵션이며 database를 standalone으로 생성할 경우에는 사용할 수 없다.

- Cloned strategy
 - 테이블의 모든 data를 동일하게 복제한다.
- Hash sharding strategy
 - Sharding key의 hash 값을 기준으로 테이블 data를 분배한다.
- Range sharding strategy
 - Sharding key의 범위 (range)값을 기준으로 테이블 data를 분배한다.
- List sharding strategy
 - Sharding key의 나열 (list)값을 기준으로 테이블 data를 분배한다.

테이블 생성에 대한 자세한 내용은 **CREATE TABLE** 구문과 **Cluster Table**과 **Shard** 개념을 참조한다.

Sharding 정책 옵션을 생략할 경우 **DEFAULT_SHARDING** 프로퍼티에 의해 결정된다. **DEFAULT_SHARDING**의 기본값은 0으로써 cloned table을 생성한다.

```
CREATE TABLE t1
(
  id    INTEGER,
  name  VARCHAR(128)
);
```

위와 같이 사용자가 sharding 정책을 부여하지 않은 채 테이블을 생성할 경우, GOLDBLOCKS cluster system은 내부적으로 다음과 같은 구문으로 테이블을 생성한다.

```
CREATE TABLE t1
(
  id    INTEGER,
```

```

    name VARCHAR(128)
)
    CLONED
    AT CLUSTER WIDE
;

```

노트

Hash/ range/ list sharded table의 경우 제약 조건을 생성할 때 PRIMARY KEY, UNIQUE 제약 조건이 sharding key를 포함하도록 해야 한다.

Cloned Strategy

테이블 data를 특정한 조건에 의해 분할하는 것이 아니라 모든 data를 복제하는 방식이다. Data 복제 대상은 cluster system 내의 모든 노드일 수도 있고 특정 cluster group만 지정할 수도 있다. 즉, 지정한 cluster group의 cluster member 들에 복제본 (clone)을 배치하는 정책이다.

- AT CLUSTER WIDE
 - Cluster system의 모든 cluster group의 모든 cluster member에 clone을 배치한다.
 - Cluster group과 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 통해 clone을 재배포할 수 있다.
- AT CLUSTER GROUP group_list
 - 지정한 cluster group들의 모든 cluster member에 clone을 배치한다.
 - 지정된 cluster group에 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 통해 clone을 재배포할 수 있다.
 - Cluster group 추가는 clone 재배포에 영향을 주지 않는다.

옵션을 생략할 경우, 기본값은 AT CLUSTER WIDE로 자동 지정된다.

```

CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(128)
)
    CLONED
    AT CLUSTER WIDE;

```

```

CREATE TABLE t1
(
    id    INTEGER,

```

```

        name VARCHAR(128)
    )
    CLONED
    AT CLUSTER GROUP G1, G2;

```

Hash Sharding Strategy

Sharding key로 지정된 column의 hash 값에 따라 테이블 data를 분배하는 정책이다.

Hash sharding 정책을 사용하려면 다음과 같은 조건에 맞게 sharding key를 지정해야 한다.

- Column을 32 개까지 나열할 수 있다.
- 동일한 column은 사용할 수 없다.
- LONG VARCHAR, LONG VARBINARY 타입의 column은 사용할 수 없다.

```

CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
    SHARDING BY HASH(id);

```

위와 같이 hash sharding과 관련된 옵션들을 생략할 경우, GOLDDILOCKS system은 다음과 같은 구문으로 해석한다.

```

CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
    SHARDING BY HASH(id)
    SHARD COUNT 24
    AT CLUSTER WIDE;

```

테이블 row들은 id column의 해시값을 기준으로 24 개의 shard 중 하나에 분배되며, 24 개의 shard는 cluster system 전체에 골고루 배치된다. 자세한 내용은 **Cluster Table**과 **Shard**을 참조한다.

Range Sharding Strategy

테이블 data를 sharding key로 지정된 column의 범위 값에 따라 분배하는 정책이다. 각 범위 값에 따라 구분된 shard들을 특정 group에 지정하여 배치할 수도 있고 CLUSTER WIDE하게 배치할 수도 있다.

다음은 sharding key column 값의 범위에 따라 여섯 개의 range shard를 정의한 후, 이를 CLUSTER WIDE 하게 분배하기 위한 테이블 생성 구문이다. 만일 테이블을 생성한 후에 cluster group이 추가될 경우, REBALANCE 기능을 이용하여 추가된 group을 포함한 shard의 재배포 작업을 수행할 수 있다.

- Range sharded table을 생성한다.
- 기존 cluster group인 (g1, g2, g3)에 여섯 개의 shard들을 배치한다.

```

CREATE TABLE t1
(
    id    INTEGER,

```

```

    name VARCHAR(32)
)
SHARDING BY RANGE (id)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 200000 ),
    SHARD s2 VALUES LESS THAN ( 400000 ),
    SHARD s3 VALUES LESS THAN ( 500000 ),
    SHARD s4 VALUES LESS THAN ( 600000 ),
    SHARD s5 VALUES LESS THAN ( 800000 ),
    SHARD s6 VALUES LESS THAN ( MAXVALUE )
;

```

- Cluster group을 추가한다.

```

CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;

```

- Range shard를 재배치한다.
- 생성된 g4를 포함하여 (g1, g2, g3, g4) cluster group에 여섯 개의 shard들을 재배치한다.

```

ALTER TABLE t1 REBALANCE;

```

다음은 sharding key column 값의 범위에 따라 정의되는 range shard들을 특정 cluster group에만 배치하도록 지정하는 구문의 예이다. 즉, 200,000 보다 작은 범위 값을 가지는 shard s1은 cluster group g1에 배치되고, s2는 cluster group g2에 배치되며, shard s3는 g3에 배치된다. 이처럼 range shard 별로 cluster group을 지정하여 생성한 테이블의 경우에는 이후 cluster group이 추가되더라도 REBALANCE 기능을 통해 shard를 재배치할 수 없다.

- Range sharded table을 생성한다.
- 각 range shard들을 지정한 cluster group에 배치한다.

```

CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY RANGE (id)
    SHARD s1 VALUES LESS THAN ( 200000 )    AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( 400000 )    AT CLUSTER GROUP g2,
    SHARD s3 VALUES LESS THAN ( 500000 )    AT CLUSTER GROUP g3,
    SHARD s4 VALUES LESS THAN ( 600000 )    AT CLUSTER GROUP g2,
    SHARD s5 VALUES LESS THAN ( 800000 )    AT CLUSTER GROUP g3,

```

```
SHARD s6 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g1
;
```

- Cluster group을 추가한다.

```
CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Range shard를 재배치한다.
- 새로 생성한 cluster group g4에는 shard가 배치되지 않는다.

```
ALTER TABLE t1 REBALANCE;
```

Range sharding key를 지정하기 위해 고려해야 하는 조건은 다음과 같다.

- Column은 32 개까지 나열할 수 있다.
- 동일한 column은 사용할 수 없다.
- LONG VARCHAR, LONG VARBINARY 타입의 column은 사용할 수 없다.

List Sharding Strategy

테이블 data를 sharding key로 지정된 column의 나열 (list) 값에 따라 분배하는 정책이다. Range sharding 정책과 마찬가지로 각 shard들을 특정 group에 지정하여 분배할 수도 있고 CLUSTER WIDE하게 분배되도록 할 수도 있다.

List sharding 정책에서 sharding key를 지정하기 위한 조건은 다음과 같다.

- 하나의 column만 사용할 수 있다.
- LONG VARCHAR, LONG VARBINARY 타입의 column은 사용할 수 없다.

다음은 생성되는 list shard들을 CLUSTER WIDE하게 배치되도록 테이블을 생성하는 구문의 예이다. 테이블을 생성한 후에 추가되는 cluster group을 포함하여 shard를 재배치하기 위해 REBALANCE 기능을 사용할 수 있다.

- List sharded table을 생성한다.
- 기존 cluster group인 (g1, g2, g3)에 다섯 개의 shard들을 배치한다.

```
CREATE TABLE city
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY LIST (name)
    AT CLUSTER WIDE
```

```

SHARD s1 VALUES IN ( 'SEOUL' ),
SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ),
SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' ),
SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' ),
SHARD s5 VALUES IN ( DEFAULT )
;

```

- Cluster group을 추가한다.

```

CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;

```

- List shard를 재배치한다.
- 추가된 g4를 포함하여 (g1, g2, g3, g4) cluster group에 다섯 개의 shard들을 재배치한다.

```
ALTER TABLE city REBALANCE;
```

다음은 sharding key column 값의 나열 값에 따라 정의되는 list shard들을 특정 cluster group에만 배치되도록 지정하는 구문의 예이다. 이후 cluster group이 추가되더라도 REBALANCE 기능을 통해 shard를 재배치할 수 없다.

- List sharded table을 생성한다.
- 각 list shard를 지정한 cluster group에 배치한다.

```

CREATE TABLE city
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY LIST (name)
    SHARD s1 VALUES IN ( 'SEOUL' )                AT CLUSTER GROUP g1,
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ) AT CLUSTER GROUP g2,
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' )    AT CLUSTER GROUP g3,
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' )       AT CLUSTER GROUP g2,
    SHARD s5 VALUES IN ( DEFAULT )                AT CLUSTER GROUP g1
;

```

- Cluster group을 추가한다.

```

CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;

```

- List shard를 재배치한다.
- 생성된 cluster group g4에 shard가 배치되지 않는다.

```
ALTER TABLE city REBALANCE;
```

인덱스 관리

Global Secondary Index

Cluster system에는 다수의 member 노드가 존재하고 테이블의 레코드들은 sharding 정책에 따라 각 member 노드에 분할 또는 복제 저장된다. Standalone 시스템의 경우 레코드가 저장될 때 해당 database 에 고유한 값 (Row Identifier: RID)을 함께 저장하여 레코드에 대한 유일성을 보장하는데, 이 경우 cluster system 환경에서 각 노드들의 값이 중복될 가능성이 있기 때문에 유일성을 보장할 수 없다.

따라서 cluster system 전체에서 레코드의 유일성을 보장하기 위해 global RID (GRID)가 추가되었다. 레코드에 대한 GRID 값은 레코드가 갱신되더라도 변경되지 않고 shard key가 변경되어 다른 shard로 이동되더라도 변경되지 않기 때문에 cluster system 환경에서 특정 레코드의 유일성을 항상 보장한다.

Global secondary index는 cluster system 환경에서 레코드들의 GRID 값을 빠르게 검색할 수 있도록 key로 구성된 B-tree 인덱스이다.

사용자는 테이블을 생성할 때 global secondary index의 생성 여부를 **DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION**을 통해 선택할 수 있고 테이블을 생성한 후에 생성된 global secondary index를 삭제하거나 다시 생성할 수도 있다. global secondary index는 테이블당 한 개만 생성할 수 있다.

테이블에 대한 non-deterministic 질의를 수행하기 위해서는 global secondary index가 반드시 필요하고, 만약 테이블에 global secondary index가 없을 경우 non-deterministic 질의는 다음과 같이 실패한다.

```
gSQL> DELETE FROM T1 LIMIT 1;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

테이블의 global secondary index 생성 여부를 확인하고 싶을 경우에는 다음과 같이 **USER_GSI_PLACE** 디렉터리를 조회하거나, **ALL_GSI_PLACE** 및 **DBA_GSI_PLACE** 디렉터리를 조회한다.

```
gSQL> CREATE TABLE T1( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT *
2 FROM USER_GSI_PLACE@LOCAL
3 WHERE TABLE_NAME = 'T1';
```



```

TABLE_SCHEMA TABLE_NAME GROUP_ID GROUP_NAME MEMBER_ID MEMBER_NAME MEMBER_OFFLINE
-----
BLOCKS
-----
PUBLIC      T1          1 G1          1 G1N1        FALSE
64
PUBLIC      T1          1 G1          2 G1N2        FALSE
null
2 rows selected.
gSQL> DROP TABLE T1;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT *
2 FROM USER_GSI_PLACE@LOCAL
3 WHERE TABLE_NAME = 'T1';
no rows selected.

```

Global 시퀀스

GOLDILOCKS cluster system은 사용자가 지정한 조건에 부합하는 시퀀스 값들의 집합을 다수의 member 노드가 공유하여 사용할 수 있도록 기존의 시퀀스 객체를 확장한 global 시퀀스 객체를 제공한다. 즉, cluster system에서 사용자가 시퀀스를 생성하면 내부적으로 global 시퀀스 객체가 자동으로 생성되며 이후 각 member 노드에서 NEXTVAL을 호출할 때 해당 global 객체로부터 특정한 범위의 시퀀스 값들을 할당받아 사용한다. 다음과 같이 기본적인 생성 및 사용 방법은 standalone의 시퀀스와 같다.

```

gSQL> CREATE SEQUENCE global_user_seq START WITH 1000 INCREMENT BY 1 NOCACHE NOCYCLE;
Sequence created.
gSQL> SELECT global_user_seq.NEXTVAL FROM dual;
NEXTVAL
-----
1
1 row selected.
gSQL> DROP SEQUENCE global_user_seq;
Sequence dropped.

```

Standalone에서 사용되는 시퀀스와 마찬가지로 global 시퀀스 객체에서도 cache size를 생성 옵션으로 지정할 수 있는데, 이는 global 시퀀스 객체로부터 몇 개의 시퀀스 값들을 획득하여 local cache에 적재해둘 것인지를 의미한다. 다음은 cache size를 5로 지정하여 global 시퀀스 객체를 생성하였을 때 각 member 노드의 local cache 상황과 NEXTVAL 호출 시의 반환값을 정리한 것이다.

```
gSQL> CREATE SEQUENCE seq START WITH 1 CACHE 5;
```

Sequence created.

Member 이름	NEXTVAL 결과	Local cache 잔여 시퀀스 개수		설명
		G1N1	G1N2	
G1N1	1	4	0	From global object (Alloc 1 ~ 5)
G1N1	2	3	0	From local cache
G1N1	3	2	0	From local cache
G1N2	6	3	4	From global object (Alloc 6 ~ 10)
G1N2	7	3	3	From local cache
G1N2	8	3	2	From local cache
G1N2	9	3	1	From local cache
G1N2	10	3	0	From local cache
G1N2	11	3	4	From global object (Alloc 11 ~ 15)
G1N2	12	3	3	From local cache
G1N1	4	1	3	From local cache
G1N1	5	0	3	From local cache
G1N1	16	4	3	From global object (Alloc 16 ~ 20)

위의 표에서 G1N1과 G1N2가 global 시퀀스 객체로부터 각 2 회씩, 총 4 회에 걸쳐 시퀀스 값들을 할당받았다. 시퀀스를 생성할 때 CACHE 옵션값을 5로 지정하였으므로 global 객체로부터 할당받을 때는 시퀀스 값들을 다섯 개씩 할당받았다. 이렇게 member 노드가 할당받은 다섯 개의 시퀀스 값들을 자신의 local cache에 보관해둔 후 NEXTVAL을 호출할 때마다 하나씩 반환한다.

- G1N1
 - 첫 번째 NEXTVAL을 호출하면 global 시퀀스 객체로부터 다섯 개 (1~5)의 시퀀스 값들을 할당받는다.
 - 한 개는 NEXTVAL의 결과로 제공하고 나머지 네 개의 값들은 자신의 local cache에 쌓아둔다.
 - 이 후 호출되는 NEXTVAL은 local cache에 적재해둔 시퀀스 값들을 차례로 반환한다.
- G1N2
 - 첫 번째 NEXTVAL을 호출하면 G1N1에서 이미 할당받은 범위 직후의 시퀀스 값들 (6~10)을 할당받는다.
 - 6은 NEXTVAL의 결과로 제공하고 나머지 네 개의 값들 (7~10)은 자신의 local cache에 쌓아둔다.
 - 이후 호출되는 NEXTVAL은 local cache에 적재해 둔 시퀀스 값들을 반환한다.
 - Local cache가 소진되어 다시 global 시퀀스 객체로부터 다섯 개 (11~15)의 시퀀스 값들을 할당받는다.
- G1N1
 - Local cache가 소진되어 G1N2에서 최종으로 할당받은 시퀀스 값 직후의 다섯 개 (16~20) 시퀀스 값들을 할당받는다.

모든 member 노드는 자신의 노드에서 NEXTVAL을 호출한 적이 없거나, 이미 할당받은 시퀀스 값들을 모두 소진할 경우 반드시 global 시퀀스 객체로부터 CACHE 크기 만큼의 시퀀스를 할당받는다. 따라서 시퀀스 값을 빠른 속도로 획득해야 하는 시스템의 경우, 시퀀스를 생성할 때 CACHE 크기를 적절한 크기로 설정하여 지나치게 빈번한 할당작업이 발생하지 않도록 해야 한다. 왜냐하면 global 시퀀스 객체로부터 시퀀스를 할당받을 때는 standalone database에서와 달리 네트워크 통신 비용이 발생하기 때문이다.

Local cache에 쌓아 둔 시퀀스 값들은 시스템 장애 및 운영자 작업 등의 이유로 database가 재기동되면 다시 사용할 수 없다. 재기동 후에는 첫 번째 NEXTVAL을 호출할 때 global 시퀀스 객체로부터 다시 시퀀스 값들을 할당받는다. 따라서 시퀀스를 생성할 때 부여하는 CACHE 크기는 장애가 발생했을 때 유실될 수 있는 시퀀스 값들의 범위를 의미하기도 하므로 응답 성능 뿐만 아니라 유실 범위를 고려하여 적절한 크기로 설정해야 한다.

Cluster system에서는 특정 노드를 기준으로 NEXTVAL 호출의 결과값이 연속적이지 않을 수 있다. 위의 예에서 G1 N1 노드가 NEXTVAL을 호출할 때 반환되는 시퀀스 값이 연속적이지 않은 것을 볼 수 있다. 첫 번째로 할당받은 시퀀스 (1~5) 값들을 소진한 후에 두 번째로 할당받은 값의 범위가 16 ~ 20 이므로 사용자는 5의 다음 시퀀스 값으로 16을 반환받는다. 이는 하나의 global 시퀀스 pool을 다수의 노드가 경쟁적으로 할당받아 사용하는 방식이기 때문이다.

기존의 standalone database 용 시퀀스와 비교하여 global 시퀀스 객체의 특징과 제약 사항은 다음과 같다.

- ALTER SEQUENCE 구문에서 INCREMENT BY 옵션을 사용하여 부호를 변경할 수 없다. (크기는 변경할 수 있다.)
- CYCLE 옵션을 사용할 경우에는 전체 시퀀스 pool의 크기에 따라 member 노드 간에 중복된 값이 반환될 수 있다. 따라서 CYCLE 옵션이 필요한 경우에는 INCREMENT BY 및 CACHE SIZE, cluster member 노드 개수를 고려하여 시퀀스 pool을 충분히 크게 생성해두도록 한다.
- 장애 상황이 아닌 경우라도 특정 member 노드를 기준으로 반환되는 시퀀스 값이 연속적이지 않을 수 있다. 물론, 하나의 member 노드만 NEXTVAL을 호출하는 경우에는 연속적인 시퀀스 값을 얻을 수 있다.
- NOCACHE일 경우에는 CACHE SIZE가 1이므로 local cache에 추가적인 시퀀스 값들을 쌓아두지 않는다. 이 경우, NEXTVAL을 호출할 때마다 global 시퀀스 객체로부터 매번 시퀀스를 한 개씩 할당받게 되어 네트워크 비용이 증가하므로 성능이 상당히 하락할 수 있다.
- 시퀀스 생성 및 변경, 삭제는 기본적으로 AUTO COMMIT으로 동작한다.
- ALTER 구문에 의해 CACHE 및 INCREMENT 크기가 변경될 경우, 모든 노드의 local cache에 쌓인 시퀀스 값들이 모두 리셋된다. 즉, 이후 NEXTVAL을 호출할 때 다시 global 시퀀스 객체로부터 새로운 시퀀스 집합을 할당받아야 한다.

3.4 GOLDDLOCKS 프로퍼티

GOLDDLOCKS cluster system에서 사용할 수 있는 주요 프로퍼티는 다음과 같다.

이름	설명
LOCAL_CLUSTER_MEMBER	Member 이름
LOCAL_CLUSTER_MEMBER_HOST	Cluster session 접속용 host address
LOCAL_CLUSTER_MEMBER_PORT	Cluster session 접속용 port
CDISPATCHER_THREADS	Cluster dispatcher thread 개수
CSERVERS	Cluster server 프로세스 개수
CLUSTER_DATA_SYNC_SERVERS	Replica data 동기화를 위한 cluster server 프로세스 개수

GOLDDLOCKS cluster system에서 각 프로퍼티는 다음과 같이 두 가지의 변경 가능한 범위 (scope)를 가진다.

- LOCAL: 전체 member의 프로퍼티 값을 변경하거나 특정 member에 한정하여 프로퍼티 값을 변경할 수 있다.
- GLOBAL: 특정 member에 한정하여 프로퍼티 값을 변경할 수 없고 반드시 cluster system 내의 모든 member에 적용해야 한다.

예를 들어 PRIVATE_STATIC_AREA_SIZE 프로퍼티의 경우 다음과 같이 LOCAL 범위까지 변경이 가능하다는 것을 알 수 있으므로 (IS_GLOBAL column = FALSE), alter system set 구문을 통해 전체 member 노드 값을 변경하거나 특정 member를 지정하여 그 값을 변경할 수 있다. alter system set 구문 뒤에 AT 절을 추가하지 않을 경우 cluster system의 모든 member에 적용한다는 의미이다.

```
% gsql test test
Connected to GOLDDLOCKS Database.
gSQL> select origin_member_name, property_name, property_value, is_global
2   from gv$property
3   where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME  PROPERTY_NAME          PROPERTY_VALUE  IS_GLOBAL
-----
G1N1                PRIVATE_STATIC_AREA_SIZE  104857600      FALSE
G1N2                PRIVATE_STATIC_AREA_SIZE  104857600      FALSE
2 rows selected.
gSQL> alter system set private_static_area_size = 2000000000 at g1n2;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
2   from gv$property
3   where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME  PROPERTY_NAME          PROPERTY_VALUE  IS_GLOBAL
-----
```

```

G1N1          PRIVATE_STATIC_AREA_SIZE 104857600      FALSE
G1N2          PRIVATE_STATIC_AREA_SIZE 2000000000      FALSE
2 rows selected.
gSQL> alter system set private_static_area_size = 3000000000;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
       2   from gv$property
       3  where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME PROPERTY_NAME          PROPERTY_VALUE IS_GLOBAL
-----
G1N1          PRIVATE_STATIC_AREA_SIZE 3000000000      FALSE
G1N2          PRIVATE_STATIC_AREA_SIZE 3000000000      FALSE
2 rows selected.

```

반면에 DDL_AUTOCOMMIT의 경우에는 다음과 같이 변경 가능한 범위가 GLOBAL이므로 alter system set 구문에 AT 절을 이용하여 특정 member를 지정할 경우 에러가 발생한다.

```

% gsql test test
Connected to GOLDILOCKS Database.
gSQL> select origin_member_name, property_name, property_value, is_global
       2   from gv$property
       3  where property_name = 'DDL_AUTOCOMMIT';
ORIGIN_MEMBER_NAME PROPERTY_NAME  PROPERTY_VALUE IS_GLOBAL
-----
G1N1          DDL_AUTOCOMMIT NO          TRUE
G1N2          DDL_AUTOCOMMIT NO          TRUE
2 rows selected.
gSQL> alter system set ddl_autocommit = false at g1n2;
ERR-42000(16398): the domain of property does not match with domain 'G1N2' :
alter system set ddl_autocommit = false at g1n2
                                *
ERROR at line 1:
gSQL> alter system set ddl_autocommit = false;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
       2   from gv$property
       3  where property_name = 'DDL_AUTOCOMMIT';
ORIGIN_MEMBER_NAME PROPERTY_NAME  PROPERTY_VALUE IS_GLOBAL
-----
G1N1          DDL_AUTOCOMMIT NO          TRUE
G1N2          DDL_AUTOCOMMIT NO          TRUE

```

2 rows selected.

4.

What's New

4.1 Feature Matrix

본 장에서는 각 major version 별로 추가된 주요 기능들에 대해 간략히 설명한다.

Architecture

System Architecture

System architecture에 대한 feature matrix는 다음과 같다.

표 4-1 System architecture의 feature matrix

Feature	1.x	2.x	3.1	3.2
Shared Nothing Cluster	X	X	O	O
DA (Direct Attach)	O	O	O	O
JDBC DA (Direct Attach)	X	X	O	O
C/S (Client/Server) Dedicated	X	O	O	O
C/S (Client/Server) Shared	X	O	O	O
multi-process applications	O	O	O	O
multi-threaded applications	O	O	O	O
Linux platform	O	O	O	O
HP platform	X	O	O	O
AIX platform	X	O	O	O
Windows Client Platform	X	O	O	O
CDC(Change Data Capture) replication	X	O	O	O
CDC replication with log mirror	X	O	O	O
multi-level start up	X	O	O	O
parallel database loading	O	O	O	O
parallel index build	X	O	O	O
SQL plan cache	X	O	O	O

Storage Internal

Storage internal에 대한 feature matrix는 다음과 같다.

표 4-2 Storage internal의 feature matrix

Feature	1.x	2.x	3.1	3.2
memory dictionary tablespace	O	O	O	O
memory data tablespace	O	O	O	O
memory undo tablespace	O	O	O	O
memory temporary tablespace	X	O	O	O
memory bitmap data segment	O	O	O	O
memory bitmap undo segment	O	O	O	O
memory bitmap instant segment	X	O	O	O
memory heap table	O	O	O	O
memory instant table	X	O	O	O
memory B-tree index	O	O	O	O
memory instant B-tree	X	O	O	O
memory instant hash	X	O	O	O
global secondary index	X	X	O	O

Transaction Control

Transaction control에 대한 feature matrix는 다음과 같다.

표 4-3 Transaction control의 feature matrix

Feature	1.x	2.x	3.1	3.2
CDS(Concurrency Data Store) database mode	O	O	O	O
TDS(Transactional Data Store) database mode	O	O	O	O
read-only database	X	O	O	O
read/write database	O	O	O	O
flat transaction	O	O	O	O
distributed transaction	X	O	O	O
read-only transaction	X	O	O	O
read/write transaction	O	O	O	O
READ COMMITTED isolation level	O	O	O	O
SERIALIZABLE isolation level with SELECT FOR UPDATE	O	O	O	O
MVCC(Multi Version Concurrency Control)	O	O	O	O
multi-version read consistency	O	O	O	O
multi-statement consistent read	O	O	O	O
implicit lock for DML	O	O	O	O
writer don't blocks readers	O	O	O	O
row-level locking	O	O	O	O
deadlock detection	O	O	O	O
deadlock resolution	O	O	O	O
lock granularity	O	O	O	O
read lock	O	O	O	O
write lock	O	O	O	O
intention lock	O	O	O	O
WAL(Write Ahead Logging)	O	O	O	O
repeat history	O	O	O	O
restart recovery	O	O	O	O
circular logging	O	O	O	O
buffered logging	O	O	O	O
logging group	X	O	O	O
supplemental logging	X	O	O	O
mirrored logging	X	O	O	O
synchronous commit	O	O	O	O
asynchronous commit	O	O	O	O
grouped commit	O	O	O	O
total rollback	O	O	O	O
implicit statement rollback	O	O	O	O

Feature	1.x	2.x	3.1	3.2
savepoint management	X	O	O	O

Backup & Recovery

Backup & recovery에 대한 feature matrix는 다음과 같다.

표 4-4 Backup & recovery의 feature matrix

Feature	1.x	2.x	3.1	3.2
off-line backup	O	O	O	O
on-line backup	X	O	O	O
full backup	X	O	O	O
incremental backup	X	O	O	O
complete recovery	O	O	O	O
incomplete recovery	X	O	O	O
auto instance recovery	O	O	O	O
tablespace recovery	X	O	O	O
file recovery	X	O	O	O

Database Information

DICTIONARY_SCHEMA 스키마

DICTIONARY_SCHEMA 스키마에 대한 feature matrix는 다음과 같다.

표 4-5 DICTIONARY_SCHEMA schema의 feature matrix

계열	Feature	1.x	2.x	3.1	3.2
ALL_ 계열 view	ALL_ALL_TABLES	X	O	O	O
	ALL_ARGUMENTS	X	X	O	O
	ALL_CATALOG	X	O	O	O
	ALL_CLUSTER_TABLES	X	X	O	O
	ALL_COL_COMMENTS	X	O	O	O
	ALL_COL_PLACE	X	X	O	X
	ALL_COL_PRIVS	X	O	O	O
	ALL_COL_PRIVS_MADE	X	O	O	O
	ALL_COL_PRIVS_RECD	X	O	O	O
	ALL_CONSTRAINTS	X	O	O	O
	ALL_CONS_COLUMNS	X	O	O	O
	ALL_DB_PRIVS	X	O	O	O
	ALL_DB_PRIVS_MADE	X	O	O	O
	ALL_DB_PRIVS_RECD	X	O	O	O
	ALL_DEPENDENCIES	X	X	O	O
	ALL_GLOBAL_SECONDARY_INDEXES	X	X	O	O
	ALL_GSI_PLACE	X	X	O	O
	ALL_INDEXES	X	O	O	O
	ALL_IND_COLUMNS	X	O	O	O
	ALL_IND_PLACE	X	X	O	O
	ALL_NONSCHEMA_COMMENTS	X	O	O	O
	ALL_OBJECTS	X	O	O	O
	ALL_PROCEDURES	X	X	O	O
	ALL_PROC_PRIVS	X	X	O	O
	ALL_PROC_PRIV_MADE	X	X	O	O
	ALL_PROC_PRIV_RECD	X	X	O	O
	ALL_SCHEMAS	X	O	O	O
	ALL_SCHEMA_PATH	X	O	O	O
	ALL_SCHEMA_PRIVS	X	O	O	O
	ALL_SCHEMA_PRIVS_MADE	X	O	O	O
	ALL_SCHEMA_PRIVS_RECD	X	O	O	O
	ALL_SEQUENCES	X	O	O	O
	ALL_SEQ_PRIVS	X	O	O	O

계열	Feature	1.x	2.x	3.1	3.2
	ALL_SEQ_PRIVS_MADE	X	O	O	O
	ALL_SEQ_PRIVS_RECD	X	O	O	O
	ALL_SHARD_KEY_COLUMNS	X	X	O	O
	ALL_SOURCE	X	X	O	O
	ALL_SYNONYMS	X	O	O	O
	ALL_TABLES	X	O	O	O
	ALL_TAB_COLS	X	O	O	O
	ALL_TAB_COLUMNS	X	O	O	O
	ALL_TAB_COMMENTS	X	O	O	O
	ALL_TAB_IDENTITY_COLS	X	O	O	O
	ALL_TAB_PLACE	X	X	X	O
	ALL_TAB_SHARDS	X	X	O	O
	ALL_TAB_PRIVS	X	O	O	O
	ALL_TAB_PRIVS_MADE	X	O	O	O
	ALL_TAB_PRIVS_RECD	X	O	O	O
	ALL_TBS_PRIVS	X	O	O	O
	ALL_TBS_PRIVS_MADE	X	O	O	O
	ALL_TBS_PRIVS_RECD	X	O	O	O
	ALL_USERS	X	O	O	O
	ALL_VIEWS	X	O	O	O
	DBA_ALL_TABLES	X	O	O	O
	DBA_ARGUMENTS	X	X	O	O
	DBA_CATALOG	X	O	O	O
	DBA_CLUSTER	X	X	O	O
	DBA_CLUSTER_COMMENTS	X	X	O	O
	DBA_CLUSTER_TABLES	X	X	O	O
	DBA_COL_COMMENTS	X	O	O	O
	DBA_COL_PLACE	X	X	O	X
	DBA_COL_PRIVS	X	O	O	O
	DBA_CONSTRAINTS	X	O	O	O
	DBA_CONS_COLUMNS	X	O	O	O
	DBA_DB_PRIVS	X	O	O	O
	DBA_DEPENDENCIES	X	X	O	O
	DBA_EXTENTS	X	O	O	O
	DBA_GLOBAL_SECONDARY_INDEXES	X	X	O	O
	DBA_GSI_PLACE	X	X	O	O
	DBA_INDEXES	X	O	O	O
	DBA_IND_COLUMNS	X	O	O	O
	DBA_IND_PLACE	X	X	O	O
	DBA_NONSCHEMA_COMMENTS	X	O	O	O

계열	Feature	1.x	2.x	3.1	3.2
DBA_ 계열 view	DBA_OBJECTS	X	O	O	O
	DBA_PROCEEDURES	X	X	O	O
	DBA_PROC_PRIVS	X	X	O	O
	DBA_PROFILES	X	O	O	O
	DBA_SCHEMAS	X	O	O	O
	DBA_SCHEMA_PATH	X	O	O	O
	DBA_SCHEMA_PRIVS	X	O	O	O
	DBA_SEQUENCES	X	O	O	O
	DBA_SEQ_PRIVS	X	O	O	O
	DBA_SHARD_KEY_COLUMNS	X	X	O	O
	DBA_SOURCE	X	X	O	O
	DBA_STAT_SYSTEM	X	X	O	O
	DBA_SYNONYMS	X	O	O	O
	DBA_SYS_PRIVS	X	O	O	O
	DBA_TABLES	X	O	O	O
	DBA_TABLESPACES	X	O	O	O
	DBA_TAB_COLS	X	O	O	O
	DBA_TAB_COLUMNS	X	O	O	O
	DBA_TAB_COMMENTS	X	O	O	O
	DBA_TAB_IDENTITY_COLS	X	O	O	O
	DBA_TAB_PLACE	X	X	O	O
	DBA_TAB_PRIVS	X	O	O	O
	DBA_TAB_SHARDS	X	X	O	O
	DBA_TBS_PRIVS	X	O	O	O
	DBA_USERS	X	O	O	O
	DBA_VIEWS	X	O	O	O
	USER_ALL_TABLES	X	O	O	O
	USER_ARGUMENTS	X	X	O	O
	USER_CATALOG	X	O	O	O
	USER_CLUSTER_TABLES	X	X	O	O
	USER_COL_COMMENTS	X	O	O	O
	USER_COL_PLACE	X	X	O	X
	USER_COL_PRIVS	X	O	O	O
	USER_COL_PRIVS_MADE	X	O	O	O
	USER_COL_PRIVS_RECD	X	O	O	O
	USER_CONSTRAINTS	X	O	O	O
	USER_CONS_COLUMNS	X	O	O	O
	USER_DEPENDENCIES	X	X	O	O
	USER_EXTENTS	X	O	O	O
	USER_GLOBAL_SECONDARY_INDEXES	X	X	O	O

계열	Feature	1.x	2.x	3.1	3.2
USER_ 계열 view	USER_GSI_PLACE	X	X	O	O
	USER_INDEXES	X	O	O	O
	USER_IND_COLUMNS	X	O	O	O
	USER_IND_PLACE	X	X	O	O
	USER_OBJECTS	X	O	O	O
	USER_PROCEDURES	X	X	O	O
	USER_PROC_PRIVS	X	X	O	O
	USER_PROC_PRIVS_MADE	X	X	O	O
	USER_PROC_PRIVS_RECD	X	X	O	O
	USER_SCHEMAS	X	O	O	O
	USER_SCHEMA_PATH	X	O	O	O
	USER_SCHEMA_PRIVS	X	O	O	O
	USER_SCHEMA_PRIVS_MADE	X	O	O	O
	USER_SCHEMA_PRIVS_RECD	X	O	O	O
	USER_SEQUENCES	X	O	O	O
	USER_SEQ_PRIVS	X	O	O	O
	USER_SEQ_PRIVS_MADE	X	O	O	O
	USER_SEQ_PRIVS_RECD	X	O	O	O
	USER_SHARD_KEY_COLUMNS	X	X	O	O
	USER_SOURCE	X	X	O	O
	USER_SYNONYMS	X	O	O	O
	USER_SYS_PRIVS	X	O	O	O
	USER_TABLES	X	O	O	O
	USER_TABLESPACES	X	O	O	O
	USER_TAB_COLS	X	O	O	O
	USER_TAB_COLUMNS	X	O	O	O
	USER_TAB_COMMENTS	X	O	O	O
	USER_TAB_IDENTITY_COLS	X	O	O	O
	USER_TAB_PLACE	X	X	O	O
	USER_TAB_PRIVS	X	O	O	O
	USER_TAB_PRIVS_MADE	X	O	O	O
	USER_TAB_PRIVS_RECD	X	O	O	O
	USER_TAB_SHARDS	X	X	O	O
	USER_USERS	X	O	O	O
	USER_VIEWS	X	O	O	O
	AUDIT_POLICIES	X	X	X	O
	AUDIT_POLICY_ENABLED	X	X	X	O
	AUDIT_POLICY_OPTIONS	X	X	X	O
	AUDIT_TRAIL	X	X	X	O
	DATABASE_PROPERTIES	X	O	O	O

계열	Feature	1.x	2.x	3.1	3.2
기타 view	DBC_TABLE_TYPE_INFO	X	O	O	O
	DICTIONARY	X	O	O	O
	DICT_COLUMNS	X	O	O	O
	IMPLEMENTATION_INFO	X	O	O	O
	IMPLEMENTATION_INFO_BASE	X	O	O	O
	JDBC_CLIENT_PROPS	X	O	O	O
	PRODUCT	X	O	O	O
	SESSION_PRIVS	X	O	O	O
	SUPPLEMENTAL_LOG_TABLE_INFO	X	O	O	O
Aliased synonym	COLS	X	O	O	O
	DICT	X	O	O	O
	IND	X	O	O	O
	OBJ	X	O	O	O
	SEQ	X	O	O	O
	TABS	X	O	O	O

INFORMATION_SCHEMA 스키마

INFORMATION_SCHEMA 스키마에 대한 feature matrix는 다음과 같다.

표 4-6 INFORMATION_SCHEMA schema의 feature matrix

Feature	1.x	2.x	3.1	3.2
COLUMNS	X	O	O	O
COLUMN_PRIVILEGES	X	O	O	O
CONSTRAINT_COLUMN_USAGE	X	O	O	O
CONSTRAINT_TABLE_USAGE	X	O	O	O
INFORMATION_SCHEMA_CATALOG_NAME	X	O	O	O
KEY_COLUMN_USAGE	X	O	O	O
PARAMETERS	X	X	O	O
REFERENTIAL_CONSTRAINTS	X	O	O	O
ROUTINES	X	X	O	O
ROUTINE_PRIVILEGES	X	X	O	O
ROUTINE_ROUTINE_USAGE	X	X	O	O
ROUTINE_SEQUENCE_USAGE	X	X	O	O
ROUTINE_TABLE_USAGE	X	X	O	O
SCHEMATA	X	O	O	O
SEQUENCES	X	O	O	O
SQL_FEATURES	X	O	O	O
SQL_IMPLEMENTATION_INFO	X	O	O	O
SQL_PACKAGES	X	O	O	O

Feature	1.x	2.x	3.1	3.2
SQL_PARTS	X	O	O	O
SQL_SIZING	X	O	O	O
STATISTICS	X	O	O	O
TABLES	X	O	O	O
TABLE_CONSTRAINTS	X	O	O	O
TABLE_PRIVILEGES	X	O	O	O
USAGE_PRIVILEGES	X	O	O	O
VIEWS	X	O	O	O
VIEW_ROUTINE_USAGE	X	X	O	O
VIEW_TABLE_USAGE	X	O	O	O

PERFORMANCE_VIEW_SCHEMA 스키마

PERFORMANCE_VIEW_SCHEMA 스키마에 대한 feature matrix는 다음과 같다.

표 4-7 PERFORMANCE_VIEW_SCHEMA schema의 feature matrix

Feature	1.x	2.x	3.1	3.2
GV\$_____	X	X	O	O
V\$AGABLE_INFO	X	X	O	O
V\$ARCHIVELOG	X	O	O	O
V\$AUDITABLE_DB_PRIVILEGES	X	X	X	O
V\$AUDITABLE_SYSTEM_ACTIONS	X	X	X	O
V\$BACKUP	X	O	O	O
V\$BALANCER	X	O	O	O
V\$CLUSTER_DISPATCHER	X	X	O	O
V\$CLUSTER_LOCATION	X	X	O	O
V\$CLUSTER_MEMBER	X	X	O	O
V\$COLUMNS	X	O	O	O
V\$CONTROLFILE	X	O	O	O
V\$DATAFILE	X	O	O	O
V\$DB_FILE	X	O	O	O
V\$DISPATCHER	X	O	O	O
V\$ERROR_CODE	X	O	O	O
V\$GLOBAL_TRANSACTION	X	O	O	O
V\$JOURNALING	X	X	O	O
V\$INCREMENTAL_BACKUP	X	O	O	O
V\$INSTANCE	X	O	O	O
V\$KEYWORDS	X	O	O	O
V\$LATCH	X	O	O	O
V\$LOCK_WAIT	X	O	O	O

Feature	1.x	2.x	3.1	3.2
V\$LOGFILE	X	O	O	O
V\$PROCESS_MEM_STAT	X	O	O	O
V\$PROCESS_SQL_STAT	X	O	O	O
V\$PROCESS_STAT	X	O	O	O
V\$PROPERTY	X	O	O	O
V\$PSM_RESERVED_WORDS	X	X	O	O
V\$QUEUE	X	O	O	O
V\$RESERVED_WORDS	X	O	O	O
V\$SESSION	X	O	O	O
V\$SESSION_AUDIT	X	X	X	O
V\$SESSION_CONNECT_INFO	X	O	O	O
V\$SESSION_EVENT	X	X	O	O
V\$SESSION_MEM_STAT	X	O	O	O
V\$SESSION_SQL_STAT	X	O	O	O
V\$SESSION_STAT	X	O	O	O
V\$SESSION_WAIT	X	X	O	O
V\$SHARED_MODE	X	O	O	O
V\$SHARED_SERVER	X	O	O	O
V\$SHM_SEGMENT	X	O	O	O
V\$SPROPERTY	X	O	O	O
V\$SQLFN_METADATA	X	O	O	O
V\$SQL_CACHE	X	O	O	O
V\$SQL_COMMAND	X	X	O	O
V\$SQL_HISTORY	X	X	O	O
V\$STATEMENT	X	O	O	O
V\$SYSTEM_EVENT	X	X	O	O
V\$SYSTEM_MEM_STAT	X	O	O	O
V\$SYSTEM_SQL_STAT	X	O	O	O
V\$SYSTEM_STAT	X	O	O	O
V\$TABLES	X	O	O	O
V\$TABLESPACE	X	O	O	O
V\$TABLESPACE_STAT	X	X	O	O
V\$TRANSACTION	X	O	O	O
V\$WAIT_EVENT_CLASS_NAME	X	X	O	O
V\$WAIT_EVENT_NAME	X	X	O	O
V\$XA_TRANSATION	X	X	O	O

Server Property

Server Property에 대한 feature matrix는 다음과 같다.

표 4-8 Server property의 feature matrix

Feature	1.x	2.x	3.1	3.2
AGING_INTERVAL	O	O	O	O
AGING_PLAN_INTERVAL	X	O	O	O
ARCHIVELOG_DIR	O	O	X	X
ARCHIVELOG_DIR_1 ~ DIR_10	X	O	O	O
ARCHIVELOG_FILE	X	O	O	O
ARCHIVELOG_MODE	X	O	O	O
BACKUP_DIR_1 ~ DIR_10	X	O	O	O
BLOCK_READ_COUNT	O	O	O	O
BULK_IO_PAGE_COUNT	X	O	O	O
CDISPATCHER_HOT_POLICY_INTERVAL	X	X	O	O
CDISPATCHER_SOCKET_BUFFER_SIZE	X	X	O	O
CDISPATCHER_THREADS	X	X	O	O
CHAR_LENGTH_UNITS	X	O	O	O
CHARACTER_SET	X	O	O	O
CHECK_DEDICATE_CONNECTION_INTERVAL	X	X	O	O
CHECK_DEDICATE_SOCKET	X	X	O	X
CLIENT_MAX_COUNT	O	O	O	O
CLIENT_NUMA_POLICY	X	X	O	O
CLOSE_PSM_CHILD_STMTS	X	X	O	O
CLUSTER_ASYNC_COMMIT	X	X	O	O
CLUSTER_ASYNC_REPLICATION	X	X	O	O
CLUSTER_CM_BUFFER_COUNT	X	X	O	O
CLUSTER_CM_BUFFER_SIZE	X	X	O	O
CLUSTER_CM_READ_BUFFER_SIZE	X	X	O	O
CLUSTER_COMMIT_SLAVES	X	X	O	O
CLUSTER_COMMIT_STREAM_ISOLATION	X	X	O	O
CLUSTER_CONNECTION	X	X	O	O
CLUSTER_CONNECTION_TIMEOUT_SEC	X	X	O	O
CLUSTER_DATA_SYNC_SERVERS	X	X	O	O
CLUSTER_DISPATCHER_IN_QUEUE_SIZE	X	X	O	O
CLUSTER_DISPATCHER_NUMA_STREAM_MAP	X	X	O	O
CLUSTER_DISPATCHER_OUT_QUEUE_SIZE	X	X	O	O
CLUSTER_HEARTBEAT_INTERVAL	X	X	O	O
CLUSTER_HEARTBEAT_RETRY_COUNT	X	X	O	O
CLUSTER_IGNORE_INACTIVE_MEMBER	X	X	O	O

Feature	1.x	2.x	3.1	3.2
CLUSTER_MAX_PACKET_SIZE	X	X	O	O
CLUSTER_MAX_PAYLOAD_SIZE	X	X	O	O
CLUSTER_PACKET_ALLOCATION_TIMEOUT	X	X	O	O
CLUSTER_SERVER_RESPONSE_QUEUE_SIZE	X	X	O	O
CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY	X	X	O	O
CLUSTER_SPLIT_BRAIN_RETRY_COUNT	X	X	O	O
COMMITTER_HOT_POLICY_INTERVAL	X	X	O	O
CONTROL_FILE_0 ~ FILE_7	X	O	O	O
CONTROL_FILE_COUNT	X	O	O	O
CONTROL_FILE_TEMP_NAME	X	O	O	O
COORDINATOR_COMMIT_WRITE_MODE	X	X	O	O
CSERVERS	X	X	O	O
DA_CLIENT_NUMA_MODE	X	X	O	O
DATA_STORE_MODE	O	O	O	O
DATABASE_ACCESS_MODE	X	O	O	O
DATABASE_INSTANCE_NAME	X	X	O	O
DDL_AUTOCOMMIT	X	O	O	O
DDL_LOCK_TIMEOUT	O	O	O	O
DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION	X	X	O	O
DEFAULT_INDEX_LOGGING	X	O	O	O
DEFAULT_INDEX_PCTFREE	X	X	O	O
DEFAULT_INITRANS	O	O	O	O
DEFAULT_MAXTRANS	O	O	O	O
DEFAULT_PCTFREE	O	O	O	O
DEFAULT_PCTUSED	O	O	O	O
DEFAULT_REMOVAL_BACKUP_FILE	X	O	O	O
DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST	X	O	O	O
DEFAULT_SHARDING	X	X	O	O
DISABLE_DDL_CDC_GIVEUP	X	O	O	O
DISABLE_UPDATE_PK_CDC_GIVEUP	X	O	O	O
DISALLOWED_PROTOCOL_TARGETTYPE	X	X	O	O
DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL	X	X	X	O
DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME	X	X	O	O
DISPATCHERS	X	O	O	O
DISPATCHER_CM_BUFFER_SIZE	X	O	O	O
DISPATCHER_CM_UNIT_SIZE	X	O	O	O
DISPATCHER_CONNECTIONS	X	O	O	O
DISPATCHER_HOT_POLICY_INTERVAL	X	X	O	O
DISPATCHER_LOAD_BALANCING	X	X	O	O
DISPATCHER_NUMA_STREAM_MAP	X	X	O	O

Feature	1.x	2.x	3.1	3.2
DISPATCHER_QUEUE_SIZE	X	O	O	O
DISPATCHER_REQUEST_MINI_QUEUE_COUNT	X	X	O	O
DISPATCHER_RESPONSE_MINI_QUEUE_COUNT	X	X	O	O
FETCH_FAILOVER	X	X	O	O
GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY	X	X	X	O
GLOBAL_JOURNAL_BUFFER_SIZE	X	X	O	O
GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE	X	X	O	O
GLOBAL_PROPERTY_LOCK_TIMEOUT	X	X	O	O
GLOBAL_TRANSACTION_COMMIT_WRITE_MODE	X	X	O	O
GLOBAL_TRANSACTION_ISOLATION_SCOPE	X	X	O	O
GLOBAL_TRANSACTION_LOG_DIR	X	X	O	O
GLOBAL_TRANSACTION_LOG_FILE_SIZE	X	X	O	O
GMASTER_NUMA_NODE	X	X	O	O
GMON_AUTOSTART	X	X	O	O
HINT_ERROR	X	O	O	O
IDLE_TIMEOUT	O	O	O	O
IN_DOUBT_DECISION	X	O	O	O
INDEX_BUILD_PARALLEL_FACTOR	X	O	O	O
INDEX_TREE_MERGE_PARALLEL_FACTOR	X	X	O	O
INST_ALLOCATOR_COUNT	X	X	O	O
INST_TABLE_BLOCK_SIZE	X	X	O	O
JOURNAL_TEMP_DIR	X	X	O	O
KEEPALIVE_IDLE_TIME	X	O	O	O
LOCAL_CLUSTER_MEMBER	X	X	O	O
LOCAL_CLUSTER_MEMBER_HOST	X	X	O	O
LOCAL_CLUSTER_MEMBER_PORT	X	X	O	O
LOCAL_JOURNAL_BUFFER_SIZE	X	X	O	O
LOCATION_FILE	X	X	O	O
LOCATOR_QUERY_TIMEOUT	X	X	O	O
LOCK_HASH_TABLE_SIZE	X	O	O	O
LOG_BLOCK_SIZE	O	O	O	O
LOG_BUFFER_SIZE	O	O	O	O
LOG_DIR	O	O	O	O
LOG_FILE_SIZE	O	O	O	O
LOG_GROUP_COUNT	O	O	O	O
LOG_MIRROR_MODE	X	O	O	O
LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	X	O	O	O
LOG_MIRROR_TIMEOUT	X	O	O	O
LOG_SYNC_INTERVAL	X	O	O	O
LOG_SYNC_INTERVAL_MSEC	X	X	O	O

Feature	1.x	2.x	3.1	3.2
MAX_GROUP_COUNT	X	X	O	O
MAX_JOURNAL_FILE_SIZE	X	X	O	O
MAX_NODE_COUNT	X	X	O	O
MAXIMUM_CONCURRENT_ACTIVITIES	X	O	O	O
MAXIMUM_FLANGE_COUNT	X	X	O	O
MAXIMUM_FLUSH_LOG_BLOCK_COUNT	O	O	O	O
MAXIMUM_FLUSH_PAGE_COUNT	O	O	O	O
MAXIMUM_JOURNAL_REPLAY_COUNT	X	X	X	O
MAXIMUM_NAMED_CURSOR_COUNT	X	O	O	O
MAXIMUM_SESSION_CM_BUFFER_SIZE	X	O	O	O
MEASURE_CLUSTER_LATENCY	X	X	O	O
MEDIA_RECOVERY_LOG_BUFFER_SIZE	X	O	X	X
MEMORY_MERGE_RUN_COUNT	X	O	O	O
MEMORY_SORT_RUN_SIZE	X	O	O	O
MIN_SAMPLE_ROW_COUNT	X	X	O	O
MINIMUM_UNDO_PAGE_COUNT	O	O	O	O
NET_BUFFER_SIZE	X	O	O	O
NLS_DATE_FORMAT	X	O	O	O
NLS_TIME_FORMAT	X	O	O	O
NLS_TIME_WITH_TIME_ZONE_FORMAT	X	O	O	O
NLS_TIMESTAMP_FORMAT	X	O	O	O
NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT	X	O	O	O
NUMA	X	X	O	O
NUMA_MAP	X	X	O	O
OFFLINE_MEMBER_AFTER_FAILOVER	X	X	O	O
ONLINE_JOURNAL_REPLAY_THRESHOLD	X	X	X	O
OS_GROUP_ACCESS	X	X	O	O
PACKET_COMPRESSION_THRESHOLD	X	X	X	O
PAGE_CHECKSUM_TYPE	X	O	O	O
PARALLEL_IO_FACTOR	O	O	O	O
PARALLEL_IO_GROUP_1 ~ GROUP_16	O	O	O	O
PARALLEL_LOAD_FACTOR	O	O	O	O
PENDING_LOG_BUFFER_COUNT	O	O	O	O
PLAN_CACHE	X	O	O	O
PLAN_CACHE_SIZE	X	O	O	O
PRIVATE_STATIC_AREA_SIZE	X	O	O	O
PROCESS_MAX_COUNT	O	O	O	O
QUERY_TIMEOUT	O	O	O	O
READABLE_ARCHIVELOG_DIR_COUNT	X	O	O	O
READABLE_BACKUP_DIR_COUNT	X	O	O	O

Feature	1.x	2.x	3.1	3.2
REBALANCE_BLOCK_READ_COUNT	X	X	O	O
RECOMPILE_CHECK_MINIMUM_PAGE_COUNT	X	O	O	X
RECOMPILE_PAGE_PERCENT	X	O	O	X
RECOVERY_LOG_BUFFER_SIZE	X	X	O	O
REDO_LOG_COMPRESSION_THRESHOLD	X	X	X	O
REFINE_RELATION	X	O	O	O
SESSION_FATAL_BEHAVIOR	X	O	O	O
SESSION_MEMORY_INIT_SIZE	X	X	X	O
SESSION_MEMORY_SHRINK_THRESHOLD	X	X	X	O
SHARED_MEMORY_ADDRESS	O	O	O	O
SHARED_MEMORY_STATIC_KEY	O	O	O	O
SHARED_MEMORY_STATIC_NAME	O	O	O	O
SHARED_MEMORY_STATIC_SIZE	O	O	O	O
SHARED_REQUEST_QUEUE_COUNT	X	O	O	O
SHARED_SERVERS	X	O	O	O
SHARED_SESSION	X	O	O	O
SNAPSHOT_STATEMENT_TIMEOUT	X	O	O	O
SQL_HISTORY_SIZE	X	X	O	O
SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	X	O	O	O
SYSTEM_LOGGER_DIR	X	O	O	O
SYSTEM_MEMORY_AUX_TABLESPACE_SIZE	X	X	X	O
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	O	O	O	O
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	O	O	O	O
SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	O	O	O	O
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	O	O	O	O
SYSTEM_TABLESPACE_DIR	O	O	O	O
SYSTEM_UDS_DIR	X	X	O	O
TCP_NODELAY	X	X	O	O
TEMP_SEGMENT_CACHE_SIZE	X	X	X	O
TEMP_UNDO_ENABLED	X	X	X	O
TIMED_STATISTICS	X	X	O	O
TIMER_INTERVAL	O	O	X	X
TIMEZONE	X	O	O	O
TRACE_ALTER_SYSTEM	X	O	O	O
TRACE_DDL	O	O	O	O
TRACE_LOG_ID	X	O	O	O
TRACE_LOG_MSGBUG_SIZE	X	X	O	O
TRACE_LOG_TIME_DETAIL	X	O	O	O
TRACE_LOGGER	X	X	O	O
TRACE_LOGGER_REMOTE_HOST	X	X	O	O

Feature	1.x	2.x	3.1	3.2
TRACE_LOGGER_REMOTE_PORT	X	X	O	O
TRACE_LOGIN	X	O	O	O
TRACE_LONG_RUN_CURSOR	X	O	O	O
TRACE_LONG_RUN_SQL	X	O	O	O
TRACE_XA	X	O	O	O
TRANSACTION_ALLOCATION_TIMEOUT	X	X	X	O
TRANSACTION_COMMIT_WRITE_MODE	O	O	O	O
TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT	X	O	O	O
TRANSACTION_TABLE_SIZE	O	O	O	O
TRANSACTION_TIMEOUT	X	X	O	O
UNDO_RELATION_ALLOCATION_TIMEOUT	X	X	X	O
UNDO_RELATION_COUNT	O	O	O	O
UNDO_SHRINK_THRESHOLD	X	O	O	O
USE_LARGE_PAGES	X	X	X	O

SQL

SQL Element

Data Type

데이터 타입에 대한 feature matrix는 다음과 같다.

표 4-9 Data type의 feature matrix

Type	Feature	1.x	2.x	3.1	3.2
문자 스트링 타입	CHAR	O	O	O	O
	VARCHAR	O	O	O	O
	LONG VARCHAR	X	O	O	O
이진 스트링 타입	BINARY	O	O	O	O
	VARBINARY	O	O	O	O
	LONG VARBINARY	X	O	O	O
십진 숫자 타입	SMALLINT	X	O	O	O
	INTEGER	X	O	O	O
	BIGINT	X	O	O	O
	NUMERIC	O	O	O	O
	DECIMAL	X	X	O	O
	NUMBER	X	O	O	O
	REAL	X	O	O	O
	DOUBLE PRECISION	X	O	O	O
	FLOAT	X	O	O	O
이진 숫자 타입	NATIVE_SMALLINT	X	O	O	O
	NATIVE_INTEGER	X	O	O	O
	NATIVE_BIGINT	X	O	O	O
	NATIVE_REAL	X	O	O	O
	NATIVE_DOUBLE	X	O	O	O
BOOLEAN 타입	BOOLEAN	O	O	O	O
날짜/시간 타입	DATE	O	O	O	O
	TIME	O	O	O	O
	TIME WITH TIME ZONE	O	O	O	O
	TIMESTAMP	O	O	O	O
	TIMESTAMP WITH TIME ZONE	O	O	O	O
	INTERVAL YEAR TO MONTH	O	O	O	O
	INTERVAL YEAR	O	O	O	O
	INTERVAL MONTH	O	O	O	O
	INTERVAL DAY TO SECOND	O	O	O	O

Type	Feature	1.x	2.x	3.1	3.2
INTERVAL 타입	INTERVAL DAY	O	O	O	O
	INTERVAL HOUR	O	O	O	O
	INTERVAL MINUTE	O	O	O	O
	INTERVAL SECOND	O	O	O	O
	INTERVAL DAY TO HOUR	O	O	O	O
	INTERVAL DAY TO MINUTE	O	O	O	O
	INTERVAL HOUR TO MINUTE	O	O	O	O
	INTERVAL HOUR TO SECOND	O	O	O	O
	INTERVAL MINUTE TO SECOND	O	O	O	O
ROWID 타입	ROWID	X	O	O	O

Function

함수 및 연산자에 대한 feature matrix는 다음과 같다.

표 4-10 Function의 feature matrix

Feature	1.x	2.x	3.1	3.2
expr1 * expr2	O	O	O	O
expr1 + expr2	O	O	O	O
datetime + interval	X	O	O	O
+ expr	O	O	O	O
expr1 - expr2	O	O	O	O
datetime - interval	X	O	O	O
- expr	O	O	O	O
expr1 / expr2	O	O	O	O
str1 str2	O	O	O	O
expr <comp> expr	O	O	O	O
expr <comp> (subquery)	X	O	O	O
(subquery) <comp> expr	X	O	O	O
(subquery) <comp> (subquery)	X	O	O	O
(expr, ...) <comp> (expr, ...)	X	O	O	O
(expr, ...) <comp> (subquery)	X	O	O	O
(subquery) <comp> (expr, ...)	X	O	O	O
expr <comp> {ALL ANY SOME} (expr, ...)	X	O	O	O
expr <comp> {ALL ANY SOME} (subquery)	X	O	O	O
(subquery) <comp> {ALL ANY SOME} (expr, ...)	X	O	O	O
(subquery) <comp> {ALL ANY SOME} (subquery)	X	O	O	O
(expr, ...) <comp> {ALL ANY SOME} (expr_list, ...)	X	O	O	O
(expr, ...) <comp> {ALL ANY SOME} (subquery)	X	O	O	O
(subquery) <comp> {ALL ANY SOME} (expr_list, ...)	X	O	O	O

Feature	1.x	2.x	3.1	3.2
ABS(num)	O	O	O	O
ACOS(num)	O	O	O	O
ADDDATE(date, interval)	X	O	O	O
ADDDATE(expr, days)	X	O	O	O
ADDTIME(expr1, expr2)	X	O	O	O
ADD_MONTHS(date, number)	X	O	O	O
AND	O	O	O	O
ASCII(char)	X	X	O	O
ASIN(num)	O	O	O	O
ATAN(num)	O	O	O	O
ATAN2(num1, num2)	O	O	O	O
AVG(num)	X	O	O	O
expr1 [NOT] BETWEEN [ASYMMETRIC SYMMETRIC] expr2 AND expr3	X	O	O	O
BITAND(num1, num2)	O	O	O	O
BITNOT(num)	O	O	O	O
BITOR(num1, num2)	O	O	O	O
BITXOR(num1, num2)	O	O	O	O
BIT_LENGTH(str)	O	O	O	O
BYTE_LENGTH(str)	O	O	O	O
CASE .. WHEN .. THEN .. ELSE .. END	X	O	O	O
CASE2(condition, result, ...)	X	O	O	O
CAST(expr AS datatype)	O	O	O	O
CBRT(num)	O	O	O	O
CEIL(num)	O	O	O	O
CEILING(num)	O	O	O	O
CHAR_LENGTH(str)	O	O	O	O
CHARACTER_LENGTH(str)	O	O	O	O
CHR(num)	X	X	O	O
CLOCK_DATE()	O	O	O	O
CLOCK_LOCALTIME()	O	O	O	O
CLOCK_LOCALTIMESTAMP()	O	O	O	O
CLOCK_TIME()	O	O	O	O
CLOCK_TIMESTAMP()	O	O	O	O
COALESCE(expr1, ..., exprN)	X	O	O	O
CONCAT(str1, str2)	O	O	O	O
CONCATENATE(str1, str2)	O	O	O	O
COS(num)	O	O	O	O
COT(num)	O	O	O	O
COUNT(expr)	X	O	O	O

Feature	1.x	2.x	3.1	3.2
COUNT(*)	X	O	O	O
CURRENT_CATALOG	O	O	O	O
CURRENT_DATE	O	O	O	O
CURRENT_SCHEMA	O	O	O	O
CURRENT_TIME	O	O	O	O
CURRENT_TIMESTAMP	O	O	O	O
CURRENT_USER	O	O	O	O
seq.CURRVAL	O	O	O	O
CURRVAL(seq)	O	O	O	O
DATEADD(datepart, number, date)	O	O	O	O
DATEDIFF(datepart, startdate, enddate)	X	O	O	O
DATE_ADD(date, interval)	X	O	O	O
DATE_PART(field, datetime)	X	O	O	O
DECODE(expr, comparison, result, ...)	X	O	O	O
DEGREES(radians)	O	O	O	O
DIGEST (data, type)	X	X	O	O
DUMP(expr)	X	O	O	O
EXISTS(subquery)	X	O	O	O
EXP(num)	O	O	O	O
EXTRACT(field FROM datetime)	X	O	O	O
FACTORIAL(num)	O	O	O	O
FLOOR(num)	O	O	O	O
FROM_BASE64(str)	X	X	O	O
GREATEST(expr, ...)	X	O	O	O
HEX(str)	X	X	O	O
expr1 [NOT] IN (expr, ...)	X	O	O	O
expr1 [NOT] IN (subquery)	X	O	O	O
subquery [NOT] IN (<expr_list>)	X	O	O	O
subquery [NOT] IN (subquery)	X	O	O	O
<expr_list> [NOT] IN (<expr_list>, ...)	X	O	O	O
<expr_list> [NOT] IN (subquery)	X	O	O	O
subquery [NOT] IN (<expr_list>, ...)	X	O	O	O
INITCAP(str)	X	O	O	O
INSTR(str, substr, ...)	X	O	O	O
IS NOT NULL	O	O	O	O
IS NULL	O	O	O	O
LAST_DAY(date)	X	O	O	O
LAST_IDENTITY_VALUE()	X	X	O	O
LEAST(expr, ...)	X	O	O	O
LENGTH(str)	O	O	O	O

Feature	1.x	2.x	3.1	3.2
LENGTHB(str)	O	O	O	O
string [NOT] LIKE pattern ESCAPE escape_char	X	O	O	O
LN(num)	O	O	O	O
LOCALTIME	O	O	O	O
LOCALTIMESTAMP	O	O	O	O
LOCAL_GROUP_ID()	X	X	O	O
LOCAL_GROUP_NAME()	X	X	O	O
LOCAL_MEMBER_ID()	X	X	O	O
LOCAL_MEMBER_NAME()	X	X	O	O
LOG(num2)	O	O	O	O
LOG(num1, num2)	O	O	O	O
LOGON_USER()	X	O	O	O
LOWER(str)	O	O	O	O
LPAD(str, length, fill)	X	O	O	O
LTRIM(str, [str])	X	O	O	O
MAX(expr)	X	O	O	O
MIN(expr)	X	O	O	O
MOD(num1, num2)	O	O	O	O
MONTHS_BETWEEN(date1, date2)	X	X	X	O
NEXT_DAY(date, day)	X	X	O	O
seq.NEXTVAL	O	O	O	O
NEXTVAL(seq)	O	O	O	O
NEXT VALUE FOR seq	O	O	O	O
NOT	X	O	O	O
NULLIF(expr1, expr2)	X	O	O	O
NVL(expr1, expr2)	X	O	O	O
NVL2(expr1, expr2, expr3)	X	O	O	O
OCTET_LENGTH(str)	O	O	O	O
OVERLAY(str1 PLACING str2 FROM start FOR length)	X	O	O	O
OR	X	O	O	O
PI()	O	O	O	O
POSITION(str1 IN str2)	O	O	O	O
POWER(num1, num2)	O	O	O	O
RADIANS(degrees)	O	O	O	O
RANDOM(min, max)	O	O	O	O
REPEAT(str, num)	X	O	O	O
REPLACE(str, from, to)	X	O	O	O
REVERSE(str)	X	X	X	O
ROUND(num)	X	O	O	O
ROUND(date, fmt)	X	O	O	O

Feature	1.x	2.x	3.1	3.2
ROWID_GRID_BLOCK_ID(rowid)	X	X	O	O
ROWID_GRID_BLOCK_SEQ(rowid)	X	X	O	O
ROWID_MEMBER_ID(rowid)	X	X	O	O
ROWID_OBJECT_ID(rowid)	X	O	O	O
ROWID_PAGE_ID(rowid)	X	O	O	O
ROWID_ROW_NUMBER(rowid)	X	O	O	O
ROWID_SHARD_ID(rowid)	X	X	O	O
ROWID_TABLESPACE_ID(rowid)	X	O	O	O
ROWNUM	X	X	O	O
RPAD(str, length, fill)	X	O	O	O
RTRIM(str, [str])	X	O	O	O
SESSION_ID()	X	O	O	O
SESSION_SERIAL()	X	O	O	O
SESSION_USER	X	O	O	O
SHARD_GROUP_ID(table, expr)	X	X	O	O
SHARD_GROUP_NAME(table_name, shard_key_value [, ...])	X	X	O	O
SHARD_ID(table, expr)	X	X	O	O
SHARD_NAME(table_name, shard_key_value [, ...])	X	X	O	O
SHIFT_LEFT(num, cnt)	O	O	O	O
SHIFT_RIGHT(num, cnt)	O	O	O	O
SIGN(num)	O	O	O	O
SIN(num)	O	O	O	O
SPLIT_PART(str, delimiter, field)	X	O	O	O
SQRT(num)	O	O	O	O
STATEMENT_DATE()	O	O	O	O
STATEMENT_LOCALTIME()	O	O	O	O
STATEMENT_LOCALTIMESTAMP()	O	O	O	O
STATEMENT_TIME()	O	O	O	O
STATEMENT_TIMESTAMP()	O	O	O	O
STATEMENT_VIEW_SCN()	X	O	O	O
STATEMENT_VIEW_SCN_DCN()	X	X	O	O
STATEMENT_VIEW_SCN_GCN()	X	X	O	O
STATEMENT_VIEW_SCN_LCN()	X	X	O	O
STDDEV([ALL DISTINCT] expr)	X	X	X	O
STDDEV_POP(expr)	X	X	X	O
STDDEV_SAMP(expr)	X	X	X	O
SUBSTR(str FROM start FOR length)	X	O	O	O
SUBSTR(str, start, length)	X	O	O	O
SUBSTRB(str, start, length)	X	O	O	O
SUBSTRING(str FROM start FOR length)	X	O	O	O

Feature	1.x	2.x	3.1	3.2
SUBSTRING(str, start, length)	X	O	O	O
SUM(expr)	X	O	O	O
SYSDATE	O	O	O	O
SYS_EXTRACT_UTC(datetime_with_timezone)	X	X	O	O
SYSTIME	O	O	O	O
SYSTIMESTAMP	O	O	O	O
TAN(num)	O	O	O	O
TO_CHAR(datetime, fmt)	X	O	O	O
TO_CHAR(number, fmt)	X	O	O	O
TO_BASE64(str)	X	X	O	O
TO_DATE(str, fmt)	X	O	O	O
TO_NATIVE_DOUBLE(str, fmt)	X	O	O	O
TO_NATIVE_REAL(str, fmt)	X	O	O	O
TO_NUMBER(num, fmt)	X	O	O	O
TO_TIME(str, fmt)	X	O	O	O
TO_TIME_TZ(str, fmt)	X	O	O	O
TO_TIME_WITH_TIME_ZONE(str, fmt)	X	O	O	O
TO_TIMESTAMP(str, fmt)	X	O	O	O
TO_TIMESTAMP_TZ(str, fmt)	X	O	O	O
TO_TIMESTAMP_WITH_TIME_ZONE(str, fmt)	X	O	O	O
TRANSACTION_DATE()	O	O	O	O
TRANSACTION_LOCALTIME()	O	O	O	O
TRANSACTION_LOCALTIMESTAMP()	O	O	O	O
TRANSACTION_TIME()	O	O	O	O
TRANSACTION_TIMESTAMP()	O	O	O	O
TRANSLATE(str, from, to)	X	O	O	O
TRIM(LEADING TRAILING BOTH trim_char FROM source)	O	O	O	O
TRUNC(num, scale)	X	O	O	O
TRUNC(date, fmt)	X	O	O	O
UPPER(str)	O	O	O	O
UNHEX(str)	X	X	O	O
UNHEX_TO_CHARSTR(str)	X	X	O	O
USER_ID()	O	O	O	O
UUID()	X	X	O	O
VAR_POP(expr)	X	X	X	O
VAR_SAMP(expr)	X	X	X	O
VARIANCE([ALL DISTINCT] expr)	X	X	X	O
VERSION()	O	O	O	O
WIDTH_BUCKET(num, min, max, cnt)	O	O	O	O

Object

SQL Object

SQL 객체를 생성/ 제거/ 변경하는 DDL에 대한 feature matrix는 다음과 같다.

표 4-11 SQL 객체 DDL의 feature matrix

객체	Feature	1.x	2.x	3.1	3.2
Database 객체	ALTER DATABASE ARCHIVELOG	X	O	O	O
	ALTER DATABASE ADD LOGFILE	X	O	O	O
	ALTER DATABASE DROP LOGFILE	X	O	O	O
	ALTER DATABASE RENAME LOGFILE	X	O	O	O
	ALTER DATABASE BEGIN/END BACKUP	X	O	O	O
	ALTER DATABASE RECOVER	X	O	O	O
	ALTER DATABASE RECOVER TABLESPACE	X	O	O	O
	ALTER DATABASE REGISTER	X	O	O	O
	ALTER DATABASE RESTORE	X	O	O	O
	ANALYZE SYSTEM	X	X	O	O
	COMMENT ON object IS ..	X	O	O	O
Profile 객체	CREATE PROFILE	X	O	O	O
	DROP PROFILE	X	O	O	O
	ALTER PROFILE	X	O	O	O
Audit policy 객체	CREATE AUDIT POLICY	X	X	X	O
	DROP AUDIT POLICY	X	X	X	O
	ALTER AUDIT POLICY	X	X	X	O
	AUDIT POLICY	X	X	X	O
	NOAUDIT POLICY	X	X	X	O
Authorization 객체	CREATE USER	X	O	O	O
	DROP USER	X	O	O	O
	ALTER USER	X	O	O	O
	GRANT privileges TO	X	O	O	O
	REVOKE privileges FROM	X	O	O	O
Schema 객체	CREATE SCHEMA	X	O	O	O
	DROP SCHEMA	X	O	O	O
Tablespace 객체	CREATE MEMORY DATA TABLESPACE	X	O	O	O
	CREATE MEMORY TEMPORARY TABLESPACE	X	O	O	O
	DROP TABLESPACE	X	O	O	O
	ALTER TABLESPACE .. RENAME TO	X	O	O	O
	ALTER TABLESPACE .. BEGIN/END BACKUP	X	O	O	O
	ALTER TABLESPACE .. ADD [DATAFILE MEMORY]	O	O	O	O
	ALTER TABLESPACE .. DROP [DATAFILE MEMORY]	X	O	O	O

객체	Feature	1.x	2.x	3.1	3.2
	ALTER TABLESPACE .. RENAME DATAFILE	X	O	O	O
	ALTER TABLESPACE .. { ONLINE OFFLINE }	X	O	O	O
Table 객체	CREATE TABLE	O	O	O	O
	CREATE TABLE AS SELECT	X	O	O	O
	CREATE GLOBAL TEMPORARY TABLE	X	X	X	O
	CREATE GLOBAL TEMPORARY TABLE AS SELECT	X	X	X	O
	DROP TABLE	O	O	O	O
	TRUNCATE TABLE	X	O	O	O
	ALTER TABLE .. STORAGE	X	O	O	O
	ALTER TABLE .. RENAME TO	X	O	O	O
	ALTER TABLE .. ADD COLUMN	X	O	O	O
	ALTER TABLE .. SET UNUSED COLUMN	X	O	O	O
	ALTER TABLE .. ALTER COLUMN	X	O	O	O
	ALTER TABLE .. RENAME COLUMN	X	O	O	O
	ALTER TABLE .. RENAME CONSTRAINT	X	X	X	O
	ALTER TABLE .. ADD CONSTRAINT	X	O	O	O
	ALTER TABLE .. DROP CONSTRAINT	X	O	O	O
	ALTER TABLE .. ALTER CONSTRAINT	X	O	O	O
	ALTER TABLE .. ADD SUPPLEMENTAL LOG	X	O	O	O
	ALTER TABLE .. DROP SUPPLEMENTAL LOG	X	O	O	O
	ALTER TABLE .. READ { ONLY WRITE }	X	X	X	O
	ANALYZE TABLE	X	X	O	O
View 객체	CREATE VIEW	X	O	O	O
	DROP VIEW	X	O	O	O
	ALTER VIEW	X	O	O	O
Index 객체	CREATE INDEX	O	O	O	O
	DROP INDEX	O	O	O	O
	ALTER INDEX .. AGING	X	X	O	O
	ALTER INDEX .. STORAGE	X	O	O	O
	ALTER INDEX .. RENAME	X	X	X	O
Sequence 객체	CREATE SEQUENCE	O	O	O	O
	DROP SEQUENCE	O	O	O	O
	ALTER SEQUENCE	X	O	O	O
Synonym 객체	CREATE SYNONYM	X	O	O	O
	DROP SYNONYM	X	O	O	O
	CREATE PUBLIC SYNONYM	X	O	O	O
	DROP PUBLIC SYNONYM	X	O	O	O
Stored procedure 객체	CREATE PROCEDURE	X	X	O	O
	DROP PROCEDURE	X	X	O	O
	ALTER PROCEDURE	X	X	O	O

객체	Feature	1.x	2.x	3.1	3.2
Stored function 객체	CREATE FUNCTION	X	X	O	O
	DROP FUNCTION	X	X	O	O
	ALTER FUNCTION	X	X	O	O

Cluster Object

Cluster 객체를 생성/ 제거/ 변경하는 DDL에 대한 feature matrix는 다음과 같다.

표 4-12 Cluster 객체 DDL의 feature matrix

객체	Feature	1.x	2.x	3.1	3.2
Cluster system 객체	ALTER DATABASE REBALANCE	X	X	O	O
	ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	X	X	O	O
Cluster group 객체	CREATE CLUSTER GROUP	X	X	O	O
	DROP CLUSTER GROUP	X	X	O	O
Cluster member 객체	ALTER CLUSTER GROUP name ADD MEMBER	X	X	O	O
	ALTER CLUSTER GROUP name OFFLINE MEMBER	X	X	O	O
	ALTER DATABASE RESET LOCAL CLUSTER MEMBER	X	X	O	O
	ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER	X	X	X	O
	ALTER SYSTEM JOIN DATABASE	X	X	O	O
Cluster location 객체	CREATE CLUSTER LOCATION	X	X	O	O
	DROP CLUSTER LOCATION	X	X	O	O
	ALTER CLUSTER LOCATION	X	X	O	O
Cluster table과 shard 객체	ALTER TABLE name REBALANCE	X	X	O	O
	ALTER TABLE name MOVE SHARD	X	X	O	O
	ALTER TABLE name SPLIT SHARD	X	X	O	O
	ALTER TABLE name RENAME SHARD	X	X	X	O
Global secondary index 객체	ALTER TABLE name ADD GLOBAL SECONDARY INDEX	X	X	O	O
	ALTER TABLE name DROP GLOBAL SECONDARY INDEX	X	X	O	O
	ALTER TABLE name ALTER GLOBAL SECONDARY INDEX	X	X	O	O

SQL Language

DML

데이터를 조작하는 DML 구문의 feature matrix는 다음과 같다.

표 4-13 DML의 feature matrix

Feature	1.x	2.x	3.1	3.2
INSERT INTO ..	O	O	O	O
INSERT INTO .. RETURNING query	X	O	O	O
INSERT INTO .. RETURNING .. INTO ..	X	O	O	O
DELETE FROM ..	O	O	O	O
DELETE FROM .. RETURNING query	X	O	O	O
DELETE FROM .. RETURNING .. INTO ..	X	O	O	O
DELETE FROM .. WHERE CURRENT OF cursor	X	O	O	O
UPDATE ..	O	O	O	O
UPDATE .. RETURNING query	X	O	O	O
UPDATE .. RETURNING .. INTO ..	X	O	O	O
UPDATE .. WHERE CURRENT OF cursor	X	O	O	O
CALL proc_name	X	X	O	O

Query

데이터를 조회하는 SELECT 구문의 feature matrix는 다음과 같다.

표 4-14 SELECT의 feature matrix

Feature	1.x	2.x	3.1	3.2
<query expression>	O	O	O	O
<query specification>	O	O	O	O
<select list>	O	O	O	O
<from clause>	O	O	O	O
<joined table>	X	O	O	O
<where clause>	O	O	O	O
<group by clause>	X	O	O	O
<order by clause>	X	O	O	O
<offset limit clause>	O	O	O	O
<set operator>	X	O	O	O
<subquery>	X	O	O	O
<hint clause>	X	O	O	O

Control Language

제어 구문의 feature matrix는 다음과 같다.

표 4-15 제어 구문의 feature matrix

구분	Feature	1.x	2.x	3.1	3.2
Transaction	COMMIT	O	O	O	O
	ROLLBACK	O	O	O	O
	SAVEPOINT	X	O	O	O
	RELEASE SAVEPOINT	X	O	O	O
	LOCK TABLE	X	O	O	O
	SET CONSTRAINTS	X	O	O	O
	SET TRANSACTION	X	O	O	O
Session	SET SESSION CHARACTERISTICS AS	X	O	O	O
	SET SESSION AUTHORIZATION	X	O	O	O
	SET TIME ZONE	X	O	O	O
	ALTER SESSION SET property	O	O	O	O
System	ALTER SYSTEM {OPEN MOUNT} DATABASE	X	O	O	O
	ALTER SYSTEM CHECKPOINT	O	O	O	O
	ALTER SYSTEM KILL SESSION	X	O	O	O
	ALTER SYSTEM RECONNECT GLOBAL CONNECTION	X	X	X	O
	ALTER SYSTEM SWITCH LOGFILE	X	O	O	O
	ALTER SYSTEM SET property	X	O	O	O
	ALTER SYSTEM RESET property	X	O	O	O

PSM Language

Persistent Stored Module (PSM) language element의 feature matrix는 다음과 같다.

표 4-16 Persistent Stored Module (PSM) language element의 feature matrix

Feature	1.x	2.x	3.1	3.2
Assignment Statement	X	X	O	O
Basic LOOP Statement	X	X	O	O
Block (BEGIN .. END)	X	X	O	O
CASE Statement	X	X	O	O
CLOSE Statement	X	X	O	O
Collection Method Invocation	X	X	O	O
Collection Variable Declaration	X	X	O	O
CONTINUE Statement	X	X	O	O
Cursor FOR LOOP Statement	X	X	O	O
Cursor Variable Declaration	X	X	O	O
DELETE Statement Extension	X	X	O	O
EXCEPTION_INIT Pragma	X	X	O	O
Exception Declaration	X	X	O	O
Exception Handler	X	X	O	O
EXECUTE IMMEDIATE Statement	X	X	O	O
EXIT Statement	X	X	O	O
Explicit Cursor Declaration and Definition	X	X	O	O
FETCH Statement	X	X	O	O
FOR LOOP Statement	X	X	O	O
GOTO Statement	X	X	O	O
IF Statement	X	X	O	O
Implicit Cursor Attribute	X	X	O	O
INSERT Statement Extension	X	X	O	O
Named Cursor Attribute	X	X	O	O
NULL Statement	X	X	O	O
OPEN Statement	X	X	O	O
OPEN FOR Statement	X	X	O	O
Procedure Call	X	X	O	O
Procedure Declaration and Definition	X	X	O	O
RAISE Statement	X	X	O	O
Record Variable Declaration	X	X	O	O
RETURN Statement	X	X	O	O
RETURNING INTO clause	X	X	O	O
%ROWTYPE Attribute	X	X	O	O
Scalar Variable Declaration	X	X	O	O

Feature	1.x	2.x	3.1	3.2
SELECT INTO Statement	X	X	O	O
SQLCODE Function	X	X	O	O
SQLERRM Function	X	X	O	O
%TYPE Attribute	X	X	O	O
UPDATE Statement Extension	X	X	O	O
WHILE LOOP Statement	X	X	O	O

API

ODBC

ODBC 표준 API에 대한 feature matrix는 다음과 같다.

표 4-17 ODBC 표준 feature matrix

Feature	1.x	2.x	3.1	3.2
SQLAllocHandle()	O	O	O	O
SQLBindCol()	O	O	O	O
SQLBindParameter()	O	O	O	O
SQLCloseCursor()	O	O	O	O
SQLColAttribute()	X	O	O	O
SQLColumnPrivileges()	X	O	O	O
SQLColumns()	X	O	O	O
SQLConnect()	O	O	O	O
SQLDescribeCol()	O	O	O	O
SQLDescribeParam()	O	O	O	O
SQLDisconnect()	O	O	O	O
SQLDriverConnect()	X	O	O	O
SQLEndTran()	O	O	O	O
SQLExecDirect()	O	O	O	O
SQLExecute()	O	O	O	O
SQLExtendedFetch()	X	O	O	O
SQLFetch()	O	O	O	O
SQLFetchScroll()	X	O	O	O
SQLForeignKeys()	X	O	O	O
SQLFreeHandle()	O	O	O	O
SQLFreeStmt()	O	O	O	O
SQLGetConnectAttr()	O	O	O	O
SQLGetCursorName()	X	O	O	O
SQLGetData()	X	O	O	O
SQLGetDescField()	O	O	O	O
SQLGetDescRec()	O	O	O	O
SQLGetDiagField()	O	O	O	O
SQLGetDiagRec()	O	O	O	O
SQLGetEnvAttr()	O	O	O	O
SQLGetFunctions()	O	O	O	O
SQLGetInfo()	X	O	O	O
SQLGetStmtAttr()	O	O	O	O

Feature	1.x	2.x	3.1	3.2
SQLGetTypeInfo()	X	O	O	O
SQLMoreResults()	X	O	O	O
SQLNumParams()	O	O	O	O
SQLNumResultCols()	O	O	O	O
SQLParamData()	X	O	O	O
SQLPrepare()	O	O	O	O
SQLPrimaryKeys()	X	O	O	O
SQLProcedureColumns()	X	O	O	O
SQLProcedures()	X	O	O	O
SQLPutData()	X	O	O	O
SQLRowCount()	O	O	O	O
SQLSetConnectAttr()	O	O	O	O
SQLSetCursorName()	X	O	O	O
SQLSetDescField()	O	O	O	O
SQLSetDescRec()	O	O	O	O
SQLSetEnvAttr()	O	O	O	O
SQLSetPos()	X	O	O	O
SQLSetStmtAttr()	O	O	O	O
SQLSpecialColumns()	X	O	O	O
SQLStatistics()	X	O	O	O
SQLTablePrivileges()	X	O	O	O
SQLTables()	X	O	O	O

ODBC 표준 이외의 부가적으로 지원하는 API에 대한 feature matrix는 다음과 같다.

표 4-18 ODBC 표준 외 함수의 feature matrix

Feature	1.x	2.x	3.1	3.2
xa_open	X	O	O	O
xa_close	X	O	O	O
xa_start	X	O	O	O
xa_end	X	O	O	O
xa_rollback	X	O	O	O
xa_prepare	X	O	O	O
xa_commit	X	O	O	O
xa_recover	X	O	O	O
xa_forget	X	O	O	O
SQLGetXaSwitch	X	O	O	O
SQLGetXaConnectionHandle	X	O	O	O
SQLGetGroupCount	X	X	X	O
SQLGetGroupIDs	X	X	X	O

Feature	1.x	2.x	3.1	3.2
SQLGetGroupName	X	X	X	O
SQLGetSuitableGroupID	X	X	X	O

JDBC

JDBC에 대한 class feature matrix는 다음과 같다.

표 4-19 JDBC class의 feature matrix

Feature	1.x	2.x	3.1	3.2
CallableStatement	X	X	O	O
CommonDataSource	X	O	O	O
Connection	X	O	O	O
ConnectionPoolDataSource	X	O	O	O
DatabaseMetaData	X	O	O	O
DataSource	X	O	O	O
Driver	X	O	O	O
ParameterMetaData	X	O	O	O
PooledConnection	X	O	O	O
PreparedStatement	X	O	O	O
ResultSet	X	O	O	O
ResultSetMetaData	X	O	O	O
RowId	X	O	O	O
Savepoint	X	O	O	O
Statement	X	O	O	O
XAConnection	X	O	O	O
XADataSource	X	O	O	O
XAResource	X	O	O	O
GoldilocksInterval	X	O	O	O
GoldilocksTypes	X	O	O	O

Embedded SQL

Precompiler Option

Precompiler의 option에 대한 feature matrix는 다음과 같다.

표 4-20 Precompiler option의 feature matrix

Feature	1.x	2.x	3.1	3.2
--help	X	O	O	O
--include-path	X	O	O	O
--no-prompt	X	O	O	O
--output	X	O	O	O
--unsafe-null	X	O	O	O
--version	X	O	O	O

Embedded SQL 전용 구문

Embedded SQL에서만 사용할 수 있는 SQL 구문에 대한 feature matrix는 다음과 같다.

표 4-21 Embedded SQL 전용 구문의 feature matrix

Feature	1.x	2.x	3.1	3.2
EXEC SQL AT	X	O	O	O
EXEC SQL ATOMIC INSERT	X	O	O	O
EXEC SQL AUTOCOMMIT	X	O	O	O
EXEC SQL BEGIN DECLARE SECTION	X	O	O	O
EXEC SQL COMMIT RELEASE	X	O	O	O
EXEC SQL CONNECT	X	O	O	O
EXEC SQL CONTEXT ALLOCATE	X	O	O	O
EXEC SQL CONTEXT FREE	X	O	O	O
EXEC SQL CONTEXT USE	X	O	O	O
EXEC SQL DISCONNECT	X	O	O	O
EXEC SQL END DECLARE SECTION	X	O	O	O
EXEC SQL FOR	X	O	O	O
EXEC SQL GET GROUPID	X	X	X	O
EXEC SQL INCLUDE	X	O	O	O
EXEC SQL INCLUDE SQLCA	X	O	O	O
EXEC SQL OPTION	X	O	O	O
EXEC SQL ROLLBACK RELEASE	X	O	O	O
EXEC SQL WHENEVER	X	O	O	O

Host Variable Data Type

Host 변수에 사용할 수 있는 embbeded SQL data type의 feature matrix는 다음과 같다.

표 4-22 Host data type의 feature matrix

Feature	1.x	2.x	3.1	3.2
C native type	X	O	O	O
struct, union	X	O	O	O
typedef	X	O	O	O
VARCHAR	X	O	O	O
LONG VARCHAR	X	O	O	O
BINARY	X	O	O	O
LONG VARBINARY	X	O	O	O
BOOLEAN	X	O	O	O
NUMBER	X	O	O	O
DATE	X	O	O	O
TIME	X	O	O	O
TIME WITH TIMEZONE	X	O	O	O
TIMESTAMP	X	O	O	O
TIMESTAMP WITH TIMEZONE	X	O	O	O
INTERVAL YEAR	X	O	O	O
INTERVAL MONTH	X	O	O	O
INTERVAL DAY	X	O	O	O
INTERVAL HOUR	X	O	O	O
INTERVAL MINUTE	X	O	O	O
INTERVAL SECOND	X	O	O	O
INTERVAL YEAR TO MONTH	X	O	O	O
INTERVAL DAY TO HOUR	X	O	O	O
INTERVAL DAY TO MINUTE	X	O	O	O
INTERVAL DAY TO SECOND	X	O	O	O
INTERVAL HOUR TO MINUTE	X	O	O	O
INTERVAL HOUR TO SECOND	X	O	O	O
INTERVAL MINUTE TO SECOND	X	O	O	O

Dynamic SQL

Dynamic SQL에 대한 feature matrix는 다음과 같다.

표 4-23 Dynamic SQL의 feature matrix

Feature	1.x	2.x	3.1	3.2
SELECT .. INTO	X	O	O	O
EXECUTE IMMEDIATE sql	X	O	O	O

Feature	1.x	2.x	3.1	3.2
PREPARE stmt	X	O	O	O
EXECUTE stmt	X	O	O	O
DECLARE cursor FOR sql	X	O	O	O
DECLARE cursor FOR stmt	X	O	O	O
OPEN cursor	X	O	O	O
OPEN cursor USING	X	O	O	O
FETCH cursor INTO	X	O	O	O
CLOSE cursor	X	O	O	O
DELETE .. WHERE CURRENT OF cursor	X	O	O	O
UPDATE .. WHERE CURRENT OF cursor	X	O	O	O

PyDBC

Module

PyDBC가 제공하는 pygoldilocks의 method feature matrix는 다음과 같다.

표 4-24 pygoldilock method의 feature matrix

Feature	1.x	2.x	3.1	3.2
connect	X	X	X	O
Date	X	X	X	O
Time	X	X	X	O
Timestamp	X	X	X	O
DateFromTicks	X	X	X	O
TimeFromTicks	X	X	X	O
TimestampFromTicks	X	X	X	O
Binary	X	X	X	O
STRING	X	X	X	O
BINARY	X	X	X	O
NUMBER	X	X	X	O
DATETIME	X	X	X	O
ROWID	X	X	X	O
getDecimalSeparator	X	X	X	O
setDecimalSeparator	X	X	X	O

pygoldilocks module의 attribute feature matrix는 다음과 같다.

표 4-25 pygoldilock attribute의 feature matrix

Feature	1.x	2.x	3.1	3.2
apilevel	X	X	X	O
threadsafety	X	X	X	O
paramstyle	X	X	X	O
version	X	X	X	O
lowercase	X	X	X	O

Connection

Connection 객체의 method feature matrix는 다음과 같다.

표 4-26 Connection method의 feature matrix

Feature	1.x	2.x	3.1	3.2
cursor	X	X	X	O
commit	X	X	X	O

Feature	1.x	2.x	3.1	3.2
rollback	X	X	X	O
close	X	X	X	O
getinfo	X	X	X	O
execute	X	X	X	O
set_attr	X	X	X	O

Connection 객체의 attribute feature matrix는 다음과 같다.

표 4-27 Connection attribute의 feature matrix

Feature	1.x	2.x	3.1	3.2
autocommit	X	X	X	O
searchescape	X	X	X	O
timeout	X	X	X	O

Cursor

Cursor 객체의 method feature matrix는 다음과 같다.

표 4-28 Cursor method의 feature matrix

Feature	1.x	2.x	3.1	3.2
excute	X	X	X	O
executemany	X	X	X	O
fetchone	X	X	X	O
fetchall	X	X	X	O
fetchmany	X	X	X	O
commit	X	X	X	O
rollback	X	X	X	O
skip	X	X	X	O
nextset	X	X	X	O
close	X	X	X	O
setinputsizes	X	X	X	O
setoutputsize	X	X	X	O
callproc	X	X	X	O
callfunc	X	X	X	O
tables	X	X	X	O
columns	X	X	X	O
statistics	X	X	X	O
rowIdColumns	X	X	X	O
rowVerColumns	X	X	X	O
primaryKeys	X	X	X	O

Feature	1.x	2.x	3.1	3.2
foreignKeys	X	X	X	O
procedures	X	X	X	O
getTypeInfo	X	X	X	O

Cursor 객체의 attribute feature matrix는 다음과 같다.

표 4-29 Cursor attribute의 feature matrix

Feature	1.x	2.x	3.1	3.2
Description	X	X	X	O
rowcount	X	X	X	O
arraysize	X	X	X	O
connection	X	X	X	O
fast_executemany	X	X	X	O

Row

Row 객체의 attribute feature matrix는 다음과 같다.

표 4-30 Row attribute의 feature matrix

Feature	1.x	2.x	3.1	3.2
cursor_description	X	X	X	O

Utility

gcreatedb

Command Usage

gcreatedb의 command usage에 대한 feature matrix는 다음과 같다.

표 4-31 gcreatedb command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--character_set	O	O	O	O
--char_length_units	X	O	O	O
--cluster	X	X	O	O
--db_comment	O	O	O	O
--help	O	O	O	O
--host	X	X	O	O
--member	X	X	O	O
--port	X	X	O	O
--silent	O	O	O	O
--timezone	X	O	O	O

glsnr

Command Usage

glsnr의 command usage에 대한 feature matrix는 다음과 같다.

표 4-32 glsnr command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--help	X	O	O	O
--home	X	X	O	O
--silent	X	O	O	O
--start	X	O	O	O
--status	X	O	O	O
--stop	X	O	O	O

Configuration File

glsnr의 configuration에 대한 feature matrix는 다음과 같다.

표 4-33 glsnr configuration file syntax의 feature matrix

Feature	1.x	2.x	3.1	3.2
BACKLOG	X	O	O	O
DEFAULT_CS_MODE	X	O	O	O
LISTENER_LOG_DIR	X	X	O	O
LISTEN_PORT	X	O	O	O
TCP_EXCLUDED	X	O	O	O
TCP_INVITED	X	O	O	O
TCP_HOST	X	O	O	O
TCP_VALIDNODE_CHECKING	X	O	O	O
TIMEOUT	X	O	O	O
USR_DIR	X	X	O	O

gsql/ gsqlnet

Command Usage

gsql의 command usage에 대한 feature matrix는 다음과 같다.

표 4-34 gsql command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
username password	O	O	O	O
--as {SYSDBA ADMIN}	X	O	O	O
--conn-string	X	O	O	O
--dsn	X	O	O	O
--enable-color	O	O	O	O
--help	O	O	O	O
--import	O	O	O	O
--no-prompt	O	O	O	O
--prompt	O	O	O	O
--silent	O	O	O	O
--version	O	O	O	O

Interactive gsql Command

gsql 프롬프트 상태에서 사용하는 interactive gsql command에 대한 feature matrix는 다음과 같다.

표 4-35 Interactive gsql command의 feature matrix

Feature	1.x	2.x	3.1	3.2
\\	O	O	O	O
\connect userid password [as sysdba]	X	O	O	O
\cshutdown	X	X	O	O
\cstartup	X	X	O	O
\ddl_cluster	X	X	O	O
\ddl_db	X	O	O	O
\ddl_tablespace	X	O	O	O
\ddl_profile	X	O	O	O
\ddl_audit_policy	X	X	X	O
\ddl_auth	X	O	O	O
\ddl_schema	X	O	O	O
\ddl_publicsynonym	X	O	O	O
\ddl_table	X	O	O	O
\ddl_constraint	X	O	O	O
\ddl_index	X	O	O	O
\ddl_view	X	O	O	O

Feature	1.x	2.x	3.1	3.2
\ddl_sequence	X	O	O	O
\ddl_synonym	X	O	O	O
\ddl_procedure	X	X	O	O
\desc	O	O	O	O
\dynamic sql :var	X	O	O	O
\exec	O	O	O	O
\exec :var := :value	O	O	O	O
\exec sql	O	O	O	O
\explain plan [on only]	O	O	O	O
\help	O	O	O	O
\history	O	O	O	O
\host {os_command}	X	X	O	O
\import	O	O	O	O
\idesc	O	O	O	O
\{n}	O	O	O	O
\prepare sql	O	O	O	O
\print	O	O	O	O
\quit	O	O	O	O
\set autocommit	O	O	O	O
\set color	O	O	O	O
\set colsize	X	O	O	O
\set ddlsize	X	O	O	O
\set error	O	O	O	O
\set history	O	O	O	O
\set linesize	O	O	O	O
\set numsize	X	O	O	O
\set pagesize	O	O	O	O
\set timing	O	O	O	O
\set vertical	O	O	O	O
\shutdown {abort immediate transactional normal}	X	O	O	O
\startup {nomount mount open}	X	O	O	O
\var	O	O	O	O

gloader/ gloadernet

Command Usage

gloader의 command usage에 대한 feature matrix는 다음과 같다.

표 4-36 gloader command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
username password	O	O	O	O
--array	O	O	O	O
--atomic	O	O	O	O
--bad	O	O	O	O
--buffered	X	O	O	O
--commit	O	O	O	O
--control	O	O	O	O
--data	O	O	O	O
--dsn	X	O	O	O
--errors	X	O	O	O
--export	O	O	O	O
--fieldterm	X	X	O	O
--filesize	X	O	O	O
--format	X	O	O	O
--help	O	O	O	O
--import	O	O	O	O
--lineterm	X	X	O	O
--log	O	O	O	O
--no-prompt	O	O	O	O
--parallel	O	O	O	O
--propagation	X	O	O	O
--qualifier	X	X	O	O
--silent	O	O	O	O
--AsTIMESTAMP	X	O	O	O
--where	X	X	X	O
--group-id	X	X	X	O
--directio-size	X	X	X	O

Control File Syntax

gloader의 control file syntax에 대한 feature matrix는 다음과 같다.

표 4-37 gloder control file syntax의 feature matrix

Feature	1.x	2.x	3.1	3.2
CHARACTERSET	X	O	O	O
FIELDS TERMINATED BY	O	O	O	O
OPTIONALLY ENCLOSED BY	O	O	O	O
TABLE table_name	O	O	O	O
TABLE schema_name.table_name	X	O	O	O
LTRIM	X	X	O	O
RTRIM	X	X	O	O
LINES TERMINATED BY	X	X	O	O
WHERE	X	X	X	O

gdump

Command Usage

gdump의 command usage에 대한 feature matrix는 다음과 같다.

표 4-38 gdump command usage의 feature matrix

Feature		1.x	2.x	3.1	3.2
Common arguments	--silent	X	O	O	O
File type	BACKUP	X	O	O	O
	COMMIT_LOG	X	X	O	O
	CONTROL	X	O	O	O
	DATA	X	O	O	O
	LOG	X	O	O	O
	LOG_BUFFER	X	X	O	O
	PEND_BUFFER	X	X	O	O
	PROPERTY	X	O	O	O
BACKUP file arguments	--body	X	O	O	O
	--tbs	X	O	O	O
	--number	X	O	O	O
	--fetch	X	O	O	O
CONTROL file arguments	--section	X	O	O	O
DATA file arguments	--header	X	O	O	O
	--number	X	O	O	O
	--fetch	X	O	O	O
LOG file arguments	--all	X	X	O	O
	--fetch	X	O	O	O
	--header	X	X	O	O
	--number	X	O	O	O
	--offset	X	O	O	O

tablediff

Configuration File

tablediff의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-39 tablediff configuration file의 feature matrix

Feature		1.x	2.x	3.1	3.2
Source table	SOURCE_PASSWORD	X	O	O	O
	SOURCE_SCHEMA	X	O	O	O
	SOURCE_TABLE	X	O	O	O
	SOURCE_URL	X	O	O	O
	SOURCE_USER	X	O	O	O
Target table	TARGET_PASSWORD	X	O	O	O
	TARGET_SCHEMA	X	O	O	O
	TARGET_TABLE	X	O	O	O
	TARGET_URL	X	O	O	O
	TARGET_USER	X	O	O	O
Sync operation	TARGET_INSERT	X	O	O	O
	TARGET_UPDATE	X	O	O	O
	TARGET_DELETE	X	O	O	O
	SOURCE_INSERT	X	O	O	O
Operation options	DIFF_BIN_FILE	X	O	O	O
	DIFF_OUT_FILE	X	O	O	O
	DISPLAY_CALL_STACK	X	O	O	O
	DISPLAY_ROW_UNIT	X	O	O	O
	EXCLUDE_COLUMNS	X	O	O	O
	LOGGING_ON_DIFF	X	O	O	O
	LOGGING_ON_SUCCESS	X	O	O	O
	JOB_QUEUE_SIZE	X	O	O	O
	JOB_THREAD	X	O	O	O
	JOB_UNIT_SIZE	X	O	O	O
	PARTITION_RANGE	X	O	O	O
	SYNC_OUT_FILE	X	O	O	O
	WHERE_CLAUSE	X	O	O	O

gsyncher

Command Usage

gsyncher의 command usage에 대한 feature matrix는 다음과 같다.

표 4-40 gsyncher command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--log	X	O	O	O
--silent	X	O	O	O
--home	X	X	O	O
--copy-right	X	O	O	O
--backup-path	X	O	O	O
--help	X	O	O	O

gmon

Command Usage

gmon의 command usage에 대한 feature matrix는 다음과 같다.

표 4-41 gmon command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--start	X	X	O	O
--stop	X	X	O	O
--status	X	X	O	O
--home	X	X	O	O
--silent	X	X	O	O
--no-copyright	X	X	O	O
--help	X	X	O	O

gtrclogger

Command Usage

gtrclogger의 command usage에 대한 feature matrix는 다음과 같다.

표 4-42 gtrclogger command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--dir	X	X	O	O
--help	X	X	O	O
--port	X	X	O	O
--start	X	X	O	O
--stop	X	X	O	O

glocator

Command Usage

glocator의 command usage에 대한 feature matrix는 다음과 같다.

표 4-43 glocator command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--create	X	X	O	O
--start	X	X	O	O
--stop	X	X	O	O
--conf	X	X	O	O
--status	X	X	O	O
--sync	X	X	X	O
--silent	X	X	O	O
--no-copyright	X	X	O	O
--help	X	X	O	O

Configuration File

glocator의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-44 glocator command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
PORT	X	X	O	O
WORKER_COUNT	X	X	O	O
SESSION_QUEUE_SIZE	X	X	O	O
SESSION_ALLOCATOR_SIZE	X	X	O	O
PACKET_ALLOCATOR_SIZE	X	X	O	O
SYSTEM_LOGGER_DIR	X	X	O	O
SYSTEM_UDS_DIR	X	X	O	O
LOCATION_FILE_DIR	X	X	O	O
LOCATION_FILE_SIZE	X	X	O	O
LOCATION_FILE_MAX_SIZE	X	X	O	O
SESSION_TIMEOUT	X	X	O	O
FAILOVER_TIMEOUT	X	X	O	O
ALTERNATE_LOCATORS	X	X	X	O
SYNC_RETRY_COUNT	X	X	X	O
SYNC_RESPONSE_TIMEOUT	X	X	X	O

gagent

Command Usage

gagent의 command usage에 대한 feature matrix는 다음과 같다.

표 4-45 gagent command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--start	X	X	O	O
--stop	X	X	O	O
--conf	X	X	O	O
--status	X	X	O	O
--home	X	X	O	O
--silent	X	X	O	O
--no-copyright	X	X	O	O
--help	X	X	O	O

Configuration File

gagent의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-46 gagent configuration file의 feature matrix

Feature	1.x	2.x	3.1	3.2
PORT	X	X	O	O
LOCATOR_HOST	X	X	O	O
LOCATOR_PORT	X	X	O	O
COMMAND_QUEUE_SIZE	X	X	O	O
COMMAND_ALLOCATOR_SIZE	X	X	O	O
PACKET_ALLOCATOR_SIZE	X	X	O	O
SYSTEM_LOGGER_DIR	X	X	O	O
SESSION_TIMEOUT	X	X	O	O
UPDATE_LOCATION_TIME	X	X	O	O
ALTERNATE_LOCATORS	X	X	X	O

gloctl

Command Usage

gloctl의 command usage에 대한 feature matrix는 다음과 같다.

표 4-47 gloctl command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--dsn	X	X	O	X
--conf	X	X	X	O
--ip	X	X	O	O
--port	X	X	O	O
--import	X	X	O	O
--silent	X	X	O	O
--no-copyright	X	X	O	O
--help	X	X	O	O

Configuration File

gloctl의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-48 gloctl configuration file의 feature matrix

Feature	1.x	2.x	3.1	3.2
PORT	X	X	X	O
LOCATOR_HOST	X	X	X	O
LOCATOR_PORT	X	X	X	O

Replication

cyclone

Command Usage

cyclone의 command usage에 대한 feature matrix는 다음과 같다.

표 4-49 cyclone command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--conf	X	O	O	O
--encrypt	X	X	O	O
--group	X	O	O	O
--help	X	O	O	O
--key	X	X	O	O
--master	X	O	O	O
--reset	X	O	O	O
--silent	X	O	O	O
--slave	X	O	O	O
--start	X	O	O	O
--status	X	O	O	O
--stop	X	O	O	O
--sync	X	O	O	O
--stand-alone	X	X	X	O
--recovery	X	X	X	O
--local	X	X	X	O

Configuration File

cyclone의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-50 cyclone configuration file의 feature matrix

Configuration	Feature	1.x	2.x	3.1	3.2
Common configuration	COMM_CHUNK_COUNT	X	O	O	O
	DSN	X	O	O	O
	USER_ENCRYPT_PW	X	X	O	O
	GROUP_NAME	X	O	O	O
	HOST_IP	X	O	O	O
	HOST_EXTERNAL_IP	X	X	O	O
	HOST_PORT	X	O	O	O
	PORT	X	O	O	O

Configuration	Feature	1.x	2.x	3.1	3.2
	PROTOCOL	X	X	O	O
	USER_ID	X	O	O	O
	USER_PW	X	O	O	O
MASTER configuration	CAPTURE_TABLE	X	O	O	O
	LOG_PATH	X	O	O	O
	READ_LOG_BLOCK_COUNT	X	O	O	O
	TRANS_SORT_AREA_SIZE	X	O	O	O
	TRANS_FILE_PATH	X	O	O	O
	SYNCHER_COUNT	X	O	O	O
	SYNC_ARRAY_SIZE	X	O	O	O
	GIVEUP_INTERVAL	X	O	O	O
	LOG_CAPTURE_INTERVAL_1	X	X	O	O
	LOG_CAPTURE_INTERVAL_2	X	X	O	O
SLAVE configuration	APPLIER_COUNT	X	O	O	O
	APPLY_ARRAY_SIZE	X	O	X	X
	APPLY_COMMIT_SIZE	X	O	O	O
	APPLY_TABLE	X	O	O	O
	MASTER_IP	X	O	O	O
	PROPAGATE_MODE	X	O	O	O
	CLUSTER	X	X	X	O
	ORACLE_DRIVER	X	X	X	O

logmirror

Command Usage

logmirror의 command usage에 대한 feature matrix는 다음과 같다.

표 4-51 logmirror command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--conf	X	O	O	O
--help	X	O	O	O
--infiniband	X	O	O	O
--master	X	O	O	O
--silent	X	O	O	O
--slave	X	O	O	O
--start	X	O	O	O
--stop	X	O	O	O

Configuration File

logmirror의 configuration file에 대한 feature matrix는 다음과 같다.

표 4-52 logmirror configuration file의 feature matrix

Configuration	Feature	1.x	2.x	3.1	3.2
Common configuration	PORT	X	O	O	O
MASTER configuration	DSN	X	O	O	O
	HOST_IP	X	O	O	O
	HOST_PORT	X	O	O	O
	PROTOCOL	X	X	O	O
	USER_ID	X	O	O	O
	USER_PW	X	O	O	O
SLAVE configuration	LOG_PATH	X	O	O	O
	MASTER_IP	X	O	O	O

cymon

Command Usage

cymon의 command usage에 대한 feature matrix는 다음과 같다.

표 4-53 cymon command usage의 feature matrix

Feature	1.x	2.x	3.1	3.2
--conf	X	O	O	O
--help	X	O	O	O
--cycle	X	O	O	O
--key	X	X	O	O
--start	X	O	O	O
--stop	X	O	O	O
--status	X	O	O	O

4.2 What's New in GOLDBLOCKS 3.2

본 장에서는 GOLDBLOCKS 3.2에 새로 추가된 기능들에 대해 간략히 설명한다.

Architecture

System Architecture

변동 사항 없음

Storage Internal

변동 사항 없음

Transaction Control

변동 사항 없음

Backup & Recovery

변동 사항 없음

Database Information

DICTIONARY_SCHEMA

Audit policy 객체 정보를 조회하기 위한 다음 view가 추가되었다.

- AUDIT_POLICIES
- AUDIT_POLICY_OPTIONS
- AUDIT_POLICY_ENABLED

Audit record를 조회하기 위해 **AUDIT_TRAIL**이 추가되었다.

다음 view가 제거되었다.

- ALL_COL_PLACE
- DBA_COL_PLACE
- USER_COL_PLACE

INFORMATION_SCHEMA

변동 사항 없음

PERFORMANCE_VIEW_SCHEMA

Audit policy의 옵션을 정의할 때 system action, privilege action에 나열할 수 있는 정보를 조회하기 위한 view가 추가되었다.

- V\$AUDITABLE_DB_PRIVILEGES
- V\$AUDITABLE_SYSTEM_ACTIONS

Server Property

Global Temporary Table을 위한 프로퍼티 추가

Global temporary table에 대해 undo logging tablespace를 지정하기 위해 TEMP_UNDO_ENABLED 프로퍼티가 추가되었다.

Global temporary table이나 global temporary index의 segment cache 크기를 지정하기 위해 TEMP_SEGMENT_CACHE_SIZE 프로퍼티가 추가되었다.

Page 개수 변경에 따른 Recompile 기능 제거

3.1 까지 지원하던 page 개수 변경에 따른 recompile 기능이 제거되었다.
따라서 다음과 같은 프로퍼티를 지원하지 않는다.

- RECOMPILE_CHECK_MINIMUM_PAGE_COUNT
- RECOMPILE_PAGE_PERCENT

Auxiliary Tablespace를 위한 프로퍼티 추가

Auxiliary tablespace의 크기를 결정하기 위한 SYSTEM_MEMORY_AUX_TABLESPACE_SIZE 프로퍼티가 추가되었다.

통신 데이터 압축을 위한 프로퍼티 추가

통신 데이터 압축 여부를 결정하는 PACKET_COMPRESSION_THRESHOLD 프로퍼티가 추가되었다.

Redo Log 압축을 위한 프로퍼티 추가

Redo log 압축 여부를 결정하는 REDO_LOG_COMPRESSION_THRESHOLD 프로퍼티가 추가되었다.

USE_LARGE_PAGES 프로퍼티 추가

HugePage를 사용하기 위해 USE_LARGE_PAGES 프로퍼티가 추가되었다.

SQL

SQL Element

Data Type

변동 사항 없음

Function

다음과 같이 분산 관련 aggregation 함수가 추가되었다.

- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- VARIANCE
- VAR_POP
- VAR_SAMP

문자열 함수 REVERSE가 추가되었다.

날짜형 함수 MONTHS_BETWEEN이 추가되었다.

Object

Audit Policy

SQL 수행을 감사할 수 있는 Audit Policy 객체가 추가되었다.

Global Temporary Table

세션에 종속적인 임시 테이블인 Global Temporary Table 객체가 추가되었다.

SQL Language

ANALYZE TABLE 구문의 병렬처리

ANALYZE TABLE 구문에 병렬처리 옵션이 추가되었다.

Audit Policy DDL

Audit Policy 객체를 제어할 수 있는 다음 DDL이 추가되었다.

- Audit policy 생성
 - CREATE AUDIT POLICY

- Audit policy 제거
 - **DROP AUDIT POLICY**
- Audit policy 변경
 - **ALTER AUDIT POLICY**
- Audit policy 활성화
 - **AUDIT POLICY**
- Audit policy 비활성화
 - **NOAUDIT POLICY**
- Audit trail 삭제
 - **ALTER DATABASE CLEAR AUDIT TRAIL**

User DDL

사용자의 default index tablespace가 추가되었다.

- User 생성
 - **CREATE USER**
- User 변경
 - **ALTER USER**

Table DDL

테이블 객체를 변경하는 다음 DDL이 추가되었다.

- 테이블 제약 조건의 이름 변경
 - **ALTER TABLE name RENAME CONSTRAINT**
- 테이블의 속성 변경
 - **ALTER TABLE name READ { ONLY | WRITE }**
- Cluster 환경에서 테이블의 특정 shard 이름 변경
 - **ALTER TABLE name RENAME SHARD**

Global temporary table을 생성하는 DDL이 추가되었다.

Index DDL

인덱스 객체를 변경하는 다음과 같은 DDL이 추가되었다.

- 인덱스 이름 변경
 - **ALTER INDEX name RENAME TO**

Cluster System DDL

Cluster system 객체를 변경하는 다음과 같은 DDL이 추가되었다.

- 회복 불가능한 cluster member 지정

- **ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER**

System DCL

System 객체를 제어하는 다음과 같은 DCL이 추가되었다.

- GLOBAL CONNECTION을 사용하는 세션의 재접속 설정
 - **ALTER SYSTEM RECONNECT GLOBAL CONNECTION**

API

ODBC

odbc.ini

odbc.ini 파일에 data source name 키워드로 LOCATOR_SERVICE와 PACKET_COMPRESSION_THRESHOLD가 추가되었다.

Location 키워드로 ALTERNATE_LOCATORS와 CONNECTION_TIMEOUT이 추가되었다.

GLOBAL CONNECTION

GLOBAL CONNECTION을 지원한다.

Statement Attributes

Statement 속성값으로 SQL_ATTR_FETCH_FAILOVER가 추가되었다.

JDBC

연결 프로퍼티

연결 프로퍼티에 packet_compression_threshold가 추가되었다.

Embedded SQL

EXEC SQL GET GROUPID INTO 명령이 추가되었다.

PDO

Venus 3.2 버전부터 PDO에서 GOLDBLOCKS에 접근할 수 있는 **PDO** 드라이버를 제공한다.

PyDBC

Venus 3.2 버전부터 python 언어를 위한 API인 **PyDBC**를 제공한다.

Ruby

Venus 3.2 버전부터 ruby 언어를 위한 API인 **Ruby** 드라이버를 제공한다.

Hibernate

Venus 3.2 버전부터 Java ORM 프레임워크인 **Hibernate**와 연동할 수 있는 소스를 제공한다.

Utility

gcreatedb

변동 사항 없음

glsnr

변동 사항 없음

gsq/gsqnet

Audit Policy 객체의 DDL 출력

Audit 객체에 대한 DDL을 출력하기 위해 interactive command `\ddl_audit_policy`가 추가되었다.

SET HEADING {ON | OFF}

질의 결과에 헤더를 출력할지 여부를 설정하기 위해 `\set heading`가 추가되었다.

gloader/gloadernet

WHERE 절

export (데이터 다운로드)에 다음과 같은 조건문을 설정할 수 있도록 하였다.

- WHERE
- --where

--group-id

gloader 명령 인자로 --group-id가 추가되었다.

--directio-size

gloader 명령 인자로 --directio-size가 추가되었다.

gdump

변동 사항 없음

tablediff

변동 사항 없음

gsyncher

변동 사항 없음

gmon

변동 사항 없음

gtrclogger

변동 사항 없음

glocator

Configuration

다중화 관련 configuration 키워드로 **ALTERNATE_LOCATORS**가 추가되었다.

Argument

Argument에 **--sync** 옵션이 추가되었다.

gagent

Configuration

Configuration 키워드로 **ALTERNATE_LOCATORS**가 추가되었다.

gloctl

Configuration

gloctl의 구동 환경을 설정하는 **Configuration** 파일이 추가되었다.

Configuration 파일을 지정하는 **conf** option이 추가되었다.

--dsn

gloctl을 구동할 때 dsn 옵션이 제거되었다.

Replication

cyclone

Recovery 기능이 추가되었다.

Cluster 환경에서 **CYCLONE** 운영 기능이 추가되었다.

Slave를 지원하는 database가 GOLDDLOCKS 이외에 **ORACLE_DRIVER**를 지원한다.

Standalone의 **실행 옵션**이 추가되었다.

Local **실행 옵션**이 추가되었다.

logmirror

변동 사항 없음

cymon

변동 사항 없음

4.3 Patch Notes

3.2.14 Patch Note

ISSUE-6253 Large page 를 사용하는 공유 메모리 할당에 실패할 경우 normal page를 사용할 수 있는 property를 지원한다.

개요

공유 메모리를 할당할 때 normal page와 large page를 사용할 수 있도록 하고, large page를 이용한 공유 메모리 할당에 실패하는 경우 normal page를 이용하여 공유 메모리를 할당할 수 있는 `USE_LARGE_PAGES` property를 지원한다.

수정 전 대처

없음

3.2.13 Patch Note

ISSUE-5862 JDBC, Multi-thread 프로그램에서 Connection 객체를 공유할 때 deadlock 현상이 발생한다.

개요

Multi-thread 프로그램에서 Connection 객체를 공유하여 사용하면 deadlock이 발생한다.

수정 전 대처

Thread 별로 Connection 객체를 별도로 생성해서 사용한다.

3.2.12 Patch Note

ISSUE-4362 Cluster에서 cserver가 free 된 메모리를 참조하여 비정상적으로 종료되었다.

개요

Cluster에서 원격 멤버가 dml을 수행할 때 cserver 세션에서 사용한 메모리를 free 한 후에 다른 dml을 수행할 경우 free 된 메모리를 이용하는 버그로 인해 비정상적으로 종료되는 문제가 있어 이를 수정하였다.

3.2.11 Patch Note

ISSUE-3869 ODBC에서 array fetch 할 때 row status의 결과가 잘못되었다.

개요

ODBC에서 array fetch 할 때 SQLFetch 함수를 호출한 후에 row의 status 값을 확인할 수 있다. 만일 SQLFetch의 반환값이 SQL_SUCCESS가 아닐 경우에는 row status나 diagnostic을 확인해야 한다.

하지만 SQLFetch의 반환값이 SQL_SUCCESS_WITH_INFO인 상황에서 diagnostic 메시지는 조회할 수 있지만 row의 status는 모두 SQL_ROW_SUCCESS라고 잘못 나온다.

현상 및 증상

다음은 숫자로 변환하지 못하는 문자열 데이터이다.

```
CREATE TABLE T1 ( I1 VARCHAR(10) );
INSERT INTO T1 VALUES ( '1' );
INSERT INTO T1 VALUES ( '2A' );
INSERT INTO T1 VALUES ( '3' );
INSERT INTO T1 VALUES ( 'AB' );
COMMIT;
```

다음은 위 데이터를 숫자 타입으로 변환해서 array fetch하는 예제의 일부이다.

```
sRet = SQLPrepare( sStmt,
                  (SQLCHAR*)"SELECT I1 FROM T1 ORDER BY I1",
                  SQL_NTS );
sRet = SQLBindCol( sStmt,
                  1,
                  SQL_C_LONG,
                  sI1,
                  sizeof(SQLINTEGER),
                  sI1Ind );
```

```

sRet = SQLSetStmtAttr( sStmt,
                      SQL_ATTR_ROW_BIND_TYPE,
                      (SQLPOINTER)SQL_BIND_BY_COLUMN,
                      0 );
sRet = SQLSetStmtAttr( sStmt,
                      SQL_ATTR_ROW_STATUS_PTR,
                      sRowStatus,
                      0 );
sRet = SQLExecute( sStmt );
sRet = SQLFetch( sStmt );
switch( sRet )
{
    case SQL_SUCCESS_WITH_INFO:
        for( i = 0; i < sFetched; i++ )
            printf( "row status: %d\n", sRowStatus[i]);
        break;
    default
        break;
}

```

숫자로 변환하지 못하는 데이터가 포함되어 있지만 row status가 모두 SQL_ROW_SUCCESS라고 나온다.

```

row status: 0
row status: 0
row status: 0
row status: 0

```

수정 전 대처

없음

ISSUE-3534 Cluster failover를 처리하는 과정에서 gagent가 서버를 shutdown하지 않고 서버가 스스로 종료되도록 변경하였다.

개요

Cluster failover를 처리하는 과정에서 gagent가 non-viability 라는 결과를 수신할 경우, 기존에는 gagent가 SHUT DOWN ABORT를 실행하여 서버를 종료하였다. 그러나 gagent가 서버를 shutdown하지 않고 서버가 스스로 종료되도록 변경하였다.

ISSUE-3534 Cluster failover를 처리하는 과정에서 glocator가 질의를 요청한 gagent로만 결과를 전송하도록 변경하였다.

개요

Cluster failover를 처리하는 과정에서 glocator는 질의를 요청한 gagent 뿐만 아니라 failover 대상인 gagent에도 failover 결과를 보낸다. 그러나 이 경우, failover 대상인 gagent도 glocator에게 질의를 보내어 cluster failover를 처리하게 된다. 따라서 glocator가 질의를 요청한 gagent에게만 결과를 보내도록 변경하였다.

ISSUE-3314 JDBC, CallableStatement의 메소드인 registerOutParameter(), set..()이 동일한 파라미터에 사용되면 정상적인 값을 얻을 수 없다.

개요

CallableStatement를 이용하여 bind type이 명확하지 않은 procedure를 호출하여 out parameter의 값을 얻으려고 할 때, registerOutParameter() 메소드와 set...() 메소드를 이용하여 bind type을 INPUT OUTPUT으로 설정한다. 그리고 out parameter의 결과값을 얻기 위해 get...() 메소드를 호출하면 정상적인 값을 반환하지 않는다.

현상 및 증상

다음과 같이 table과 procedure를 생성한다.

```
CREATE TABLE PROC_TABLE ( I1 INTEGER );
INSERT INTO PROC_TABLE VALUES ( 1 );
INSERT INTO PROC_TABLE VALUES ( 2 );
INSERT INTO PROC_TABLE VALUES ( 3 );
INSERT INTO PROC_TABLE VALUES ( 4 );
INSERT INTO PROC_TABLE VALUES ( 5 );
INSERT INTO PROC_TABLE VALUES ( 6 );
COMMIT;
CREATE OR REPLACE PROCEDURE PROC_TEST_1( A1 INTEGER, A2 OUT INTEGER )
IS
BEGIN
    SELECT COUNT(*)
        INTO A2
        FROM PROC_TABLE
        WHERE I1 >= A1;
END;
/
```

다음은 procedure PROC_TEST_1을 gsql에서 호출한 결과이다. 결과값은 두 번째 parameter의 out value에 저장

된다.

```
gSQL> var v1 integer
gSQL> var v2 integer
gSQL> exec :v1 := 1
gSQL> call proc_test_1 (:v1,:v2);
Procedure Call complete.
gSQL> print
NAME                VALUE
-----
VAR_ELAPSED_TIME__  null
V1                  1
V2                  6
gSQL>
```

다음은 procedure PROC_TEST_1을 호출하는 프로그램 코드의 일부이다.

```
CallableStatement sCstmt = aCon.prepareCall( "CALL PROC_TEST_1( ?, ? )" );
sCstmt.setInt( 1, 1 );
sCstmt.setInt( 2, 1 );
sCstmt.registerOutParameter( 1, Types.INTEGER );
sCstmt.registerOutParameter( 2, Types.INTEGER );
sCstmt.execute();
System.out.println("OUTPUT: " + sCstmt.getInt(1) + ", " + sCstmt.getInt(2) );
sCstmt.close();
```

해당 프로그램을 실행하면 다음과 같이 out parameter 값이 두 번째 parameter가 아닌 첫 번째 parameter에 저장된다.

```
OUTPUT: 6, 0
```

수정 전 대처

다음과 같이 input, output 타입을 정확하게 확인하여 위와 같은 set 메소드와 registerOutParameter 메소드를 모두 사용하지 말아야 정상적인 값이 출력된다.

```
CallableStatement sCstmt = aCon.prepareCall( "CALL PROC_TEST_1( ?, ? )" );
sCstmt.setInt( 1, 1 );
sCstmt.registerOutParameter( 2, Types.INTEGER );
sCstmt.execute();
System.out.println("OUTPUT: " + sCstmt.getInt(1) + ", " + sCstmt.getInt(2) );
sCstmt.close();
```

OUTPUT: 1, 6

ISSUE-3302 JDBC, CallableStatement의 메소드인 getBytes() 메소드를 실행하면 에러가 발생한다.

개요

CallableStatement에서 out parameter의 타입이 BINARY, VARBINARY 또는 LONG VARBINARY인 경우 getBytes() 메소드를 사용하면 에러가 발생한다.

현상 및 증상

다음은 CallableStatement에 Types.BINARY로 out parameter를 등록하는 예이다.

```
CallableStatement sCStmt = aCon.prepareCall( "BEGIN ? := x'aaff'; END; " );
sCStmt.registerOutParameter(1, java.sql.Types.BINARY);
sCStmt.executeUpdate();
byte[] sValue = sCStmt.getBytes(1);
```

위의 코드를 포함한 프로그램을 실행하면 다음과 같은 에러가 발생한다.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
    at indep.jdbc.dt.RowCache.readBytes(RowCache.java:637)
    at indep.jdbc.dt.Column.getBytes(Column.java:589)
    at indep.jdbc.core.JdbcCallableStatement.getBytes(JdbcCallableStatement.java:316)
```

수정 전 대처

없음

gloader의 명령 인자로 --group-id를 추가하였다.

개요

--group-id 인자는 클러스터 환경에서 sharded 테이블에 그룹별로 데이터를 업로드 할 수 있도록 한다.

gloader의 명령 인자로 --directio-size를 추가하였다.

개요

--directio-size 인자를 이용하여 direct IO 크기를 조절할 수 있다.

3.2.10 Patch Note

ISSUE-3253 서비스 도중에 오프라인 테이블스페이스를 복구하면 로그버퍼에 기록된 로그를 복구하지 않는 문제가 있다.

개요

GOLDILOCKS의 오프라인 테이블스페이스를 온라인으로 변경할 때 복구가 필요한 경우가 있고 그렇지 않은 경우도 있다. 복구가 필요한 경우는 오프라인 구문에서 IMMEDIATE 옵션을 사용하였거나, 운영 중에 데이터 파일에 문제가 발생하여 해당 테이블스페이스가 오프라인 된 경우이다. 이 경우, 해당 테이블스페이스에 대한 로그가 버퍼에 남아있을 수 있는데 로그 파일에 기록된 로그만으로 복구를 수행하여 시스템이 비정상 종료하거나 데이터베이스 일관성이 어긋나는 문제가 발생하였다.

현상 및 증상

사용자가 생성한 테이블스페이스에 테이블 T1을 생성한 후 트랜잭션 TX1이 T1을 갱신하는 도중에 데이터 파일을 삭제하고 체크포인트를 발생시킨다. 체크포인트를 수행할 때 데이터 파일이 존재하지 않으면 해당 테이블스페이스를 오프라인으로 변경하고 이후 트랜잭션 TX1을 rollback한다. 그리고 로그버퍼의 로그가 디스크로 씌여지지 않은 상태에서 오프라인 된 테이블스페이스를 복구하고 온라인으로 변경하면 비정상적으로 종료된다.

수정 전 대처

없음

ISSUE-3222 Cluster 환경에서 GOLDILOCKS 시스템 프로세스가 hangup 상태일 때 시스템 전체가 멈춘다.

개요

Cluster 환경에서 원격 cluster member에 protocol을 전송한 후 응답을 기다릴 때, 원격 cluster member의 GOLDILOCKS 시스템 프로세스가 hangup 상태여서 응답을 보내주지 못하는 경우 응답을 기다리는 세션 뿐만 아니라 전체 시스템이 멈추는 현상이 발생한다. 이는 GOLDILOCKS 내부적으로 응답을 반드시 받아야만 하는 protocol에 대해 query timeout 이나 session 상태를 체크하지 않기 때문에 나타나는 현상이다. 이에, 특정 시간동안 응답이 없는 경우 세션을 종료시키거나 응답이 없는 원격 cluster member를 failover 시킨 후에 서비스를 진행하도록 수정하였다.

세션을 종료시키는 정책을 사용할 경우, CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT에 설정된 시간(초) 동안 대기한 후에 세션을 종료시킨다. 반면에 failover 하는 정책을 사용할 경우, CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT에 설정된 시간(초) 만큼 대기한 후 응답이 없는 원격 cluster member를 failover 시킨다.

현상 및 증상

G1N1, G1N2로 구성된 1 by 2 cluster에서 G1N2 member의 commit 서버를 hangup 상태로 만든 후 G1N1에 접속한 세션에서 commit을 실행하면 세션이 응답을 받지 못하여 멈추게 된다.

G1N2 member의 gmaster를 hangup 상태로 만든 후 G1N1에 접속한 세션에서 *ALTER SYSTEM SWITCH LOGFILE*을 수행하면 세션이 응답을 받지 못하여 멈추게 된다.

수정 전 대처

없음

ISSUE-3175 gpec이 #define 구문에서 주석을 처리하지 못한다.

개요

#define 구문에 주석이 있는 경우 gpec이 이를 처리하지 못한다.

현상 및 증상

다음 gc 파일의 매크로 AA는 1로 동일한 내용이어야 한다. 하지만 gpec이 매크로에 있는 주석을 처리하지 못하여 다른 내용으로 처리하고 있다. gpec이 다음 gc 파일을 처리하면 경고 메시지를 출력한다.

```
#define AA 1 /* comment */
#define AA 1
```

```
ERR-42000(41028): 'AA' macro is already defined at line 3, in file test.gc
```

수정 전 대처

없음

ISSUE-3175 gpec이 #if, #else에서 define 구문을 정상적으로 처리하지 못한다.

개요

#if와 #endif 또는 #else와 #endif 사이에 #define을 사용할 경우, gpec이 이를 정상적으로 처리하지 못한다.

현상 및 증상

#if와 #endif 또는 #else와 #endif 사이에 false 조건에 속한 #define이 있을 경우 gpec이 이를 처리하지 않아야 함에도 불구하고 처리하고 있다.

다음과 같은 gc 파일이 있을 때 gpec은 false 조건에 해당하는 `#define AA 2`를 처리하지 않아야 한다. 그러나 실제로는 이를 무시하지 않고 처리하여 에러를 발생시킨다.

```
#if 1
#define AA 1
#else
#define AA 2
#endif
```

```
ERR-42000(41028): 'AA' macro is already defined at line 4, in file test.gc
```

수정 전 대처

없음

ISSUE-3175 gpec이 처리하는 #define 구문의 개수가 고정되어 있다.

개요

gpec이 관리하는 #define 구문이 256 개로 고정되어 있으며 256 개를 초과하면 에러가 발생한다.

현상 및 증상

gc 파일에 서로 다른 #define 구문이 256 개를 넘으면 다음과 같은 에러가 발생한다.

```
ERR-42000(41000): syntax error at line 258, in file test.gc
ERR-42000(41027): too many 'define' macro (256)
```

수정 전 대처

없음

ISSUE-3175 gpec이 비어있는 브라켓 주석을 정상적으로 처리하지 못한다.

개요

`/**/`와 같이 내용 없이 비어있는 브라켓 주석 다음에 임의의 내용이 있는 브라켓 주석이 존재할 경우 gpec이 정상적으로 파싱하지 못한다.

현상 및 증상

다음은 빈 브라켓 주석과 일반적인 브라켓 주석이 함께 사용된 gc 파일이다.

```

/**/
EXEC SQL BEGIN DECLARE SECTION;
int value;
EXEC SQL END DECLARE SECTION;
/* comment */
EXEC SQL SELECT 1 INTO :value FROM DUAL;

```

이 gc 파일을 gpec이 파싱하면 도중에 다음과 같은 에러가 발생한다.

```

ERR-42000(41000): syntax error at line 8, in file a.gc:
SELECT 1 INTO :value FROM DUAL;
                ^  ^

Error at line 1
ERR-42000(41002): Host variable "value" not declared
ERR-42000(41006): Fatal error while doing embedded SQL precompiling

```

수정 전 대처

빈 브라켓 주석을 사용하지 않는다.

ISSUE-3243 glsnr이 잘못된 프로토콜을 받으면 종료된다.

개요

glsnr이 잘못된 프로토콜을 받으면 종료되어 버리는 문제를 수정하였다. 수정 후 다음과 같은 로그 메시지를 출력하지만 glsnr이 종료되지는 않는다.

```

2020-01-15 17:57:51.835036 THREAD(27742,139777341413120)]
[LISTENER] Invalid communication protocol : 192.168.0.123

```

현상 및 증상

glsnr이 잘못된 프로토콜을 받을 경우 다음과 같은 로그 메시지를 남기며 종료된다.

```

[2020-01-15 11:14:28.738666 THREAD(5706,140285308184384)]
[LISTENER] abnormally terminated
ERR-08S01(24001): Invalid communication protocol

```

수정 전 대처

없음

ISSUE-3174 ODBC 속성에 LOCALITY_GROUP_POLICY, LOCALITY_GROUP_PATH, LOCALITY_MEMBER_POLICY, LOCALITY_MEMBER_PATH 항목을 추가하였다.

개요

ODBC 속성에 다음 항목들을 추가하였다.

- LOCALITY_GROUP_POLICY
- LOCALITY_GROUP_PATH
- LOCALITY_MEMBER_POLICY
- LOCALITY_MEMBER_PATH

ISSUE-3220 세 개 이상의 join에 대해 다수의 subquery 조건이 존재하면 일부 subquery 조건이 누락되었다.

개요

세 개 이상의 table을 join 할 때 두 개 이상의 subquery 조건이 존재할 경우 subquery 조건을 처리할 위치를 결정 (push-down subquery filter) 한다.

이 때, 첫 번째 subquery 조건을 가장 하위의 table에 배치하고, 두 번째 subquery 조건을 그 상위 join에 배치할 경우, 첫 번째 subquery 조건이 누락된다.

현상 및 증상

다음과 같이 table과 data를 생성한다.

```
CREATE TABLE r ( r_c1 INTEGER,
                  r_c2 INTEGER );

COMMIT;

CREATE TABLE s ( s_c1 INTEGER,
                  s_c2 INTEGER );

COMMIT;

CREATE TABLE t ( t_c1 INTEGER,
                  t_c2 INTEGER );

COMMIT;

CREATE TABLE u ( u_c1 INTEGER,
                  u_c2 INTEGER );

COMMIT;

CREATE TABLE v ( v_c1 INTEGER,
                  v_c2 INTEGER );
```



```

COMMIT;
INSERT INTO r VALUES ( 1, 1 );
INSERT INTO s VALUES ( 1, 1 );
INSERT INTO t VALUES ( 1, 1 );
INSERT INTO u VALUES ( 1, 1 );
INSERT INTO v VALUES ( 1, 1 );
COMMIT;

```

다음 질의와 같이 EXISTS 조건이 존재할 경우, 조건을 만족하는 결과가 존재하지 않는다.

```

SELECT
    *
FROM r,
    s,
    t
WHERE r_c1 = s_c1
    AND s_c1 = t_c1
    AND EXISTS (
        SELECT u_c1
        FROM u
        GROUP BY u_c1
        HAVING u_c1 < 0
    )
;
no rows selected.

```

그러나 위의 질의에 다음과 같이 AND NOT EXISTS subquery 조건을 추가할 경우, 잘못된 질의 결과를 생성한다.

```

--# wrong result
SELECT
    *
FROM r,
    s,
    t
WHERE r_c1 = s_c1
    AND s_c1 = t_c1
    AND EXISTS (
        SELECT u_c1
        FROM u
        GROUP BY u_c1
        HAVING u_c1 < 0
    )

```

```

        AND NOT EXISTS ( SELECT *
                           FROM v
                           WHERE v_c1 = r_c1 + s_c1 )
;
R_C1 R_C2 S_C1 S_C2 T_C1 T_C2
-----
      1      1      1      1      1      1
1 row selected.

```

수정 전 대처

다음과 같이 NOT EXISTS subquery에 NO_PUSH_SUBQ 힌트를 추가하면 올바른 결과를 얻을 수 있다.

```

SELECT
    *
FROM r,
    s,
    t
WHERE r_c1 = s_c1
    AND s_c1 = t_c1
    AND EXISTS (
        SELECT u_c1
        FROM u
        GROUP BY u_c1
        HAVING u_c1 < 0
    )
    AND NOT EXISTS ( SELECT /*+ NO_PUSH_SUBQ */ *
                      FROM v
                      WHERE v_c1 = r_c1 + s_c1 )
;
no rows selected.

```

ISSUE-3199 EmbeddedSQL에서 array로 GroupId를 얻을 때 정상적으로 처리하지 못한다.

개요

Array로 GroupId를 얻고 캐시된 SQL 구문을 다시 실행하면 프로그램이 비정상적으로 종료된다.

현상 및 증상

```
EXEC SQL BEGIN DECLARE SECTION;
int sGroupId[5];
int sValue[5];
EXEC SQL END DECLARE SECTION;
int i;
for( i = 0; i < 5; i++ ) {
    sValue[i] = i;
}
EXEC SQL GET GROUPID INTO: sGroupId
        INSERT INTO TEST_T1 VALUES( :sValue );
EXEC SQL
        INSERT INTO TEST_T1 VALUES( :sValue );
```

캐시된 INSERT INTO TEST_T1 VALUES(:sValue) 구문을 다시 수행하면 프로그램이 비정상적으로 종료된다.

수정 전 대처

Array를 사용하지 않거나 캐시된 SQL 구문을 사용하지 않도록 호스트 변수를 변경한다.

ISSUE-3197 gpec이 < ... > 문자열을 정상적으로 처리하지 못한다.

개요

gpec이 파싱할 때 < > 가 같은 줄에 있으면 정상적으로 처리되지 않는다.

현상 및 증상

```
for( i = 0; i < 5; i++ ) { // > COMMENT
    sValue[i] = i;
}
```

< 5; i++) { // > 문자열을 정상적으로 처리하지 못하여 gpec을 수행할 때 에러가 발생한다.

수정 전 대처

다음과 같이 브라켓이나 주석의 위치를 변경하여 gc 파일을 작성한다.

```
for( i = 0; i < 5; i++ ) // > COMMENT
{
    sValue[i] = i;
}
```

```
for( i = 0; i < 5; i++ ) {
// > COMMENT
sValue[i] = i;
}
```

3.2.9 Patch Note

ISSUE-3188 gsql에 set heading을 추가하였다.

개요

set heading {on|off}를 이용하여 질의 결과에 헤더를 출력할지 여부를 설정할 수 있다.

3.2.8 Patch Note

ISSUE-3175 gpec이 non-ascii 문자를 처리하지 못한다.

개요

gpec이 처리하는 preprocessor #if, #ifdef, #elif, #else가 false 그룹일 경우, 해당 그룹 안의 내용은 공백으로 바뀐다. 그러나 false 그룹 안의 non-ascii 문자는 공백으로 처리되지 못한다.

현상 및 증상

```
#if 0
EXEC SQL INSERT INTO TEST_T1 VALUES( :sC1, :sC2 ); -- 주석
#endif
```

위 예제 코드를 gpec으로 처리하면 모두 공백으로 바뀌어야 하지만 non-ascii 문자는 남아 있다.

주석

수정 전 대처

non-ascii 문자를 c 주석 형식으로 처리한다.

ISSUE-2958 Embedded SQL에서 SQL statement의 group ID를 얻을 수 있다.

개요

Global connection을 사용하는 cluster 환경에서 shard key가 설정된 table을 대상으로 delete/ insert/ select/ update 구문의 group ID를 얻을 수 있다. 자세한 내용은 embedded SQL의 EXEC SQL GET GROUPID INTO를 참조한다.

ISSUE-3186 노드 두 개가 동시에 비정상 종료될 경우, failover 과정에서 hang이 발생할 수 있다.

개요

Domain coordinator 노드와 global coordinator 노드가 동시에 비정상 종료될 경우, failover 과정에서 hang이 발생할 수 있어 이를 수정하였다.

현상 및 증상

failover 과정에서 hang이 발생함으로써 비정상 종료된 노드들이 속한 그룹들의 온라인 트랜잭션 서비스가 중지될 수 있다.

수정 전 대처

없음

3.2.7 Patch Note

ISSUE-3093 gpec이 preprocessor #define을 파싱하는 중에 에러가 발생한다.

개요

Preprocessor #define의 대체 문자열에 C의 예약어가 올 경우, 에러가 발생한다.

현상 및 증상

```
#define SQLCA_STORAGE_CLASS extern
```

gpec을 수행할 때 파싱 에러가 발생한다.

```
$ gpec test.gc
FileName: test.gc
Pre-compile test.gc -> test.c
ERR-42000(41000): syntax error at line 1, in file test.gc:
#define SQLCA_STORAGE_CLASS extern
.....^
Error at line 1, in file test.gc
```

수정 전 대처

gpec을 수행하지 않는 일반 헤더 파일에 키워드를 정의하여 사용한다.

3.2.6 Patch Note

ISSUE-3149 SELECT INTO 구문에 구조체 배열을 사용하는 파일을 gpec이 정상적으로 파싱하지 못한다.

개요

SELECT INTO 구문에 호스트 변수로 구조체 배열을 사용하는 gc파일을 gpec이 정상적으로 처리하지 못한다.

현상 및 증상

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct AA
{
    char c1[10+1];
    char c2[10+1];
    char c3[10+1];
    char c4[10+1];
} AA;
AA sArr[10];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT c1,c2,c3, c4
        INTO :sArr FROM EMP;
STL_TRY(sqlca.sqlcode == 0);
```

위 예제의 SELECT INTO 구문이 다음과 같이 올바르지 못한 구문으로 변경되었다.

```
sqlargs.sqlstmt = (char *)"SELECT c1,c2,c3,c4\n"
"      INTO :sArr ?, ?FROM EMP\n"
```

수정 전 대처

없음

ISSUE-3145 세션에서 사용 가능한 메모리가 있는데도 새로운 메모리를 할당하여 SSA가 증가한다.

개요

세션이 시작된 이후에 사용한 메모리는 세션에서 관리하며 재사용될 수 있고, 메모리 단편화를 고려하여 효율적으로 메모리를 할당하기 위해 메모리 크기에 따라 여러 개의 레벨로 관리된다. 사용 후에 해제된 메모리를 재할당하는 과정에서 단편화를 최소화하기 위해 재할당 할 수 있는 메모리가 있음에도 불구하고 새로운 메모리 청크를 할당함으로써 SSA가 지속적으로 증가하는 문제가 있어서 수정하였다.

현상 및 증상

LONG VARBINARY 타입을 포함한 테이블을 조회할 때 V\$SYSTEM_MEM_STAT을 살펴보면 VARIABLE_STATIC_ALLOC_SIZE가 지속적으로 증가하는 것을 확인할 수 있다.

```
gSQL> SELECT STAT_NAME, ROUND(STAT_VALUE/1024/1024) FROM V$SYSTEM_MEM_STAT WHERE STAT_NAME = '
VARIABLE_STATIC_ALLOC_SIZE';
STAT_NAME                                ROUND(STAT_VALUE/1024/1024)
-----
VARIABLE_STATIC_ALLOC_SIZE                1800.25
1 row selected.
gSQL> SELECT STAT_NAME, ROUND(STAT_VALUE/1024/1024) FROM V$SYSTEM_MEM_STAT WHERE STAT_NAME = '
VARIABLE_STATIC_ALLOC_SIZE';
STAT_NAME                                ROUND(STAT_VALUE/1024/1024)
-----
VARIABLE_STATIC_ALLOC_SIZE                3663.41
1 row selected.
```

수정 전 대처

없음

ISSUE-3144 필드 구분자와 라인 구분자의 첫 문자를 동일하게 사용하면 gloader가 데이터를 정상적으로 import하지 못한다.

개요

필드 구분자와 라인 구분자의 첫 문자를 동일하게 사용하면 데이터가 누락되거나 gloader 프로세스가 비정상적으로 종료된다.

현상 및 증상

```
>$ cat test.dat
1234^C2^C3456^S
>$ gloader test test -i --tablename TEST --fieldterm ^C --lineterm ^R\n --data test.dat
gSQL> SELECT * FROM TEST
I1    I2 I3
-----
1234  C2 3456S
1 row selected.
```

위 예제에서 필드 구분자와 라인 구분자의 첫 문자가 ^로 동일하다. gloader가 import를 수행한 후에 column I3의 정상적인 결과값은 3456^S 이 되어야 하지만 실제 결과는 ^ 문자가 누락된 비정상 데이터이다.

수정 전 대처

필드 구분자와 라인 구분자의 첫 문자를 서로 다르게 사용한다.

3.2.5 Patch Note

ISSUE-3093 #define과 #undef를 declare section 내에 선언해야 gpec이 이를 처리함 또한 #ifdef 와 같은 if group의 false 값에 대한 문장을 공백으로 변경하지 않는다.

개요

gpec이 declare section 내에 선언된 #define과 #undef를 처리하는 바람에 #if, #ifdef와 같은 if group에서 매크로를 정상적으로 처리하지 못하고 있다. 또한 false 값에 해당하는 if group의 c code를 공백으로 변경하지 않아 c code 또는 SQL statement 중간에 preprocessor를 사용할 수 없다.

현상 및 증상

Preprocessor #define과 #undef를 declare section 내에 선언하지 않으면 gpec이 처리하지 못하여 해당 매크로를 인식하지 못한다. SQL 중간에 if group에 해당하는 preprocessor를 사용하지 못하여 사용자가 같은 내용의 코드를 반복하여 작성해야 한다.

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char
#ifdef _DEV_           ❶
sTrue[10];             ❷
#else
sFalse[10];
#endif
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT        ❸
#ifdef _DEV_
    "true" INTO :sTrue
#else
    "false" INTO :sFalse
#endif
FROM DUAL;
```

노트

- ❶ _DEV_ 가 declare section 밖에 선언되어 false가 된다.
- ❷ Preprocessor가 공백 처리되지 않아 gpec이 파싱 에러 처리한다.
- ❸ SQL 구문 파싱 중에 에러가 발생한다.

수정 전 대처

Preprocessor #define과 #undef를 declare section 내에 선언하여 사용하고, c code와 SQL 구문 중간에 preprocessor를 사용하지 않는다.

ISSUE-3075 JDBC의 `PreparedStatement.setCharacterStream(int, Reader, int)` method와 `PreparedStatement.addBatch()` method를 반복 수행할 때 data type이 변경되어 에러가 발생한다.

개요

JDBC의 `PreparedStatement` 클래스 method인 `setAsciiStream()`, `setBinaryStream()`, `setCharacterStream()` method를 수행할 때 기존에는 매개 변수 `length`로 data type을 결정하였던 것을 long data type만 사용하도록 변경하였다.

현상 및 증상

`setCharacterStream()` method로 `VARCHAR` 타입이 허용하는 길이의 데이터를 설정한 후에 `addBatch()` method를 호출하고, `setCharacterStream()` method로 `VARCHAR` 타입이 허용하는 길이를 초과하는 데이터를 설정하려고 시도하면 data type이 `VARCHAR`에서 `LONG VARCHAR`로 변경되어 에러가 발생한다.

```
gSQL> create table t1 ( i1 long varchar );
Table created.
```

```
sPstmt.setCharacterStream( 1, new StringReader( DATA ), DATA.length() );
sPstmt.addBatch();
sPstmt.setCharacterStream( 2, new StringReader(BIG_DATA), BIG_DATA.length() );
sPstmt.addBatch();
```

```
Caused by: java.sql.SQLException: Parameter type[LONG VARCHAR] is mismatch with previous type[
VARCHAR] during batch
```

수정 전 대처

`PreparedStatement` 클래스의 `set Ascii/ Binary/ Character Stream()` method에는 매개 변수 `length`가 주어진 method와 `length`가 없는 method 두 종류가 있다. 이 중에 매개 변수 `length`가 없는 method를 이용한다.

ISSUE-3056 JDBC의 `ResultSet.getBoolean()`을 수행했을 때 `TRUE/ FALSE`를 반환하는 문자열이 확장되었다.

개요

JDBC의 `ResultSet.getBoolean()`을 수행할 때 기존에는 "true", "false" 문자열만 boolean 타입으로 변환할 수 있었지만 "t", "f", "y", "n", "yes", "no", "on", "off", "1", "0" 문자열도 변환할 수 있도록 변경하였다.

현상 및 증상

문자열 "0", "1" 을 ResultSet.getBoolean()으로 읽을 경우, 에러가 발생한다.

```
gSQL> create table t1 ( i1 varchar(10) );
Table created.
gSQL> insert into t1 values ( '1' );
1 row created.
gSQL> commit;
Commit complete.
```

```
ResultSet rs = stmt.executeQuery("select * from t1");

while(rs.next())
{
    System.out.println( rs.getBoolean(1) );
}
```

Exception in thread "main" java.sql.SQLException: The value[1] is out of range of [boolean] type

수정 전 대처

없음

ISSUE-3055 서버와 클라이언트의 character set이 다른 상태에서 연결할 때 invalid 문자를 전송한다.

개요

서버와 클라이언트의 character set이 다른 상태에서 연결할 경우 invalid 문자가 전송되는 문제를 수정하였다.

현상 및 증상

리눅스 서버에서 "가" 사용자가 생성된다.

```
gSQL> create user "가" identified by test;
User created.
gSQL> grant create session to "가";
Grant succeeded.
gSQL> commit;
Commit complete.
```

윈도우 클라이언트에서 리눅스 서버로 연결할 때 에러가 발생한다.

```
D:\goldilocks_home\bin>gsqlnet.exe "가" test
ERR-28000(16004): invalid username/password; logon denied
```

리눅스 클라이언트에서 리눅스 서버로 연결할 때는 정상적으로 동작한다.

```
% gsqlnet "가" test
gSQL>
```

수정 전 대처

서버와 클라이언트의 character set을 동일하게 설정하거나 연결할 때 사용되는 문자열에 ASCII만 포함시키도록 한다.

ISSUE-2359 부모 세션이 없는 cluster peer가 존재한다.

개요

원격 노드에 부모 세션이 없는 cluster peer가 존재하는 문제가 발생하여 이를 수정하였다.

현상 및 증상

부모 세션이 로그인을 시도하다가 패스워드 변경 과정에서 에러가 발생한 경우, 원격 노드에 부모 세션이 없는 cluster peer가 남아 있을 수 있다.

패스워드를 변경하는 과정에서 원격 노드에 cluster peer 세션이 만들어질 수 있고 패스워드 변경에 실패하면 cluster peer 세션을 종료하지 않은 상태에서 부모 세션을 종료한다.

수정 전 대처

없음

3.2.4 Patch Note

ISSUE-3026 gsql에서 \ddl_tablespace 수행 시 AT 구문 위치를 수정하였다.

개요

gsql에서 \ddl_tablespace를 수행할 때 AT 구문 위치가 잘못되는 문제를 수정하였다.

현상 및 증상

\ddl_tablespace로 생성된 SQL 구문을 실행할 때 오류가 발생한다.

```
gSQL> create tablespace test_tbs datafile 'test.dbf' size 10m;
Tablespace created.
gSQL> \ddl_tablespace test_tbs
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE
        '/home/sunje/goldilocks_data/db/test.dbf'
        AT "G2N1"
        SIZE 10485760 REUSE
    ,
        '/home/sunje/goldilocks_data/db/test.dbf'
        AT "G2N2"
        SIZE 10485760 REUSE
    ONLINE
    LOGGING
    EXTSIZE 262144
;
COMMIT;
```

```
gSQL> CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE
        '/home/sunje/goldilocks_data/db/test.dbf'
        AT "G2N1"
        SIZE 10485760 REUSE
    ,
        '/home/sunje/goldilocks_data/db/test.dbf'
        AT "G2N2"
        SIZE 10485760 REUSE
    ONLINE
    LOGGING
    EXTSIZE 262144;
ERR-42000(40000): syntax error:
    AT "G2N1"
    ^^
Error at line 4
```

수정 전 대처

\ddl_tablespace로 생성된 SQL에서 AT 구문 위치를 수정한다.

```
gSQL> CREATE MEMORY DATA TABLESPACE "TEST_TBS"
      DATAFILE
        '/home/sunje/goldilocks_data/db/test.dbf'
      SIZE 10485760 REUSE
      AT "G2N1"
    ,
      '/home/sunje/goldilocks_data/db/test.dbf'
      SIZE 10485760 REUSE
      AT "G2N2"
    ONLINE      LOGGING
    EXTSIZE 262144;
Tablespace created.
```

ISSUE-3023 BEGIN BACKUP AT DOMAIN을 사용할 때 오류가 있다.

개요

특정 그룹이나 멤버에서만 백업을 수행하기 위해 AT DOMAIN 절을 사용할 경우, 정상적으로 동작하지 않는 문제가 있어서 이를 수정하였다.

현상 및 증상

다음과 같이 G1N2 멤버가 ARCHIVELOG로 동작하고 있는데도 BEGIN BACKUP이 실패하는 문제가 있다.

```
gSQL> SELECT ARCHIVELOG_MODE FROM V$ARCHIVELOG;
ARCHIVELOG_MODE
-----
ARCHIVELOG
1 row selected.
gSQL> ALTER DATABASE BEGIN BACKUP AT G1N2;
ERR-HY000(16247): MEMBER(G1N1): cannot BACKUP; noarchive log mode
```

수정 전 대처

AT DOMAIN을 사용하지 않고 BACKUP BEGIN/ END를 수행한다.

3.2.3 Patch Note

ISSUE-3006 EXPLAIN PLAN ONLY를 수행할 때 트랜잭션이 생성된다.

개요

EXPLAIN PLAN ONLY를 수행할 때 트랜잭션이 생성되는 문제를 수정하였다.

현상 및 증상

```
gSQL> select * from x$transaction;
no rows selected.
gSQL> \explain plan only update t2 set c2 = 1 where c1 = 10;
>>> start print plan
< Execution Plan >
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
|    0  |  UPDATE STATEMENT                                |
|
|    1  |  UPDATE ("T2")                                    |
0 |
|    2  |  INDEX ACCESS ("T2", "T2_PRIMARY_KEY_INDEX") [CLONED] |
0 |
=====
====
0 - SQL : UPDATE "PUBLIC"."T2"@LOCAL "_A1" SET("C2")=(:_V0) WHERE "_A1"."C1" = :_V1
2 - READ INDEX COLUMNS : C1
    MIN RANGE : C1 = 10
    MAX RANGE : C1 = 10
<<< end print plan
gSQL> select * from x$transaction;
PHYSICAL_TRANS_ID LOGICAL_TRANS_ID DRIVER_MEMBER_POS DRIVER_MEMBER_ID DRIVER_TRANS_ID SLOT_ID
STATE IS_XA INDOUBT_TRANS_BEHAVIOR ATTRIBUTE ISOLATION_LEVEL VIEW_SCN COMMIT_SCN PREV_COMMIT_
SCN TCN BEGIN_LSN USED_UNDO_PAGE_COUNT UNDO_SEGMENT_ID SEQ BEGIN_TIME
PROPAGATE_LOG PREPARABLE GRID_SEQ WEIGHT
-----
```

```

-----
-----
-65480      458808      0      1      458808      56
ACTIVE FALSE      0 READ_ONLY READ COMMITTED 502.1.77 -1.-1.-1 502.0.77
1      -1      0      4294901760 7 2019-07-02 17:40:13.659857 TRUE
TRUE      7 HIGH
1 row selected.
gSQL> commit;
Commit complete.
gSQL> select * from x$transaction;
no rows selected.

```

수정 전 대처

없음

3.2.2 Patch Note

ISSUE-2947 Cluster 환경에서 동일한 세션에서 수행한 원격 그룹의 이전 버전 데이터를 조회한다.

개요

하나의 세션에서 특정 원격 그룹에서만 변경되는 domain 트랜잭션을 실행한 후 select 할 때 동일한 데이터의 이전 버전을 조회하는 문제를 수정하였다.

현상 및 증상

다음과 같이 G1, G2 클러스터 그룹에서 sharded table을 생성한 후 G1에 위치한 shard에 레코드를 추가한다.

```

gSQL> CREATE TABLE T1 ( C1 NUMBER ) SHARDING BY RANGE ( C1 )
SHARD S1 VALUES LESS THAN ( 1000 ) AT CLUSTER GROUP G1,
SHARD S2 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP G2;
Table created.
gSQL> ALTER TABLE T1 ADD PRIMARY KEY ( C1 );
Table altered.
gSQL> INSERT INTO T1 VALUES ( 1 );
1 row created.
gSQL> COMMIT;
Commit complete.

```


G2 그룹의 member에 접속한 세션 (session 1)에서 G1 그룹의 레코드를 삭제 (트랜잭션 T1)한 후, 다른 세션 (session 2)에서 global 트랜잭션 (트랜잭션 T2)을 수행하고 COMMIT 한다. 이 때 G1 그룹에서 T2가 완료되고, G2 그룹에서는 완료되지 않은 상태에서 T1을 COMMIT 한 다음 다시 G1의 레코드를 조회하면 삭제된 레코드가 조회되는 문제가 있다.

```
gSQL> DELETE FROM T1 WHERE C1 = 1;    // T1 -- session 1
1 row deleted.
gSQL> CREATE TABLE T2 ( I1 INTEGER ); // T2 -- session 2
Table created.
gSQL> COMMIT;                          // -- session 2
Commit complete.
gSQL> COMMIT;                          // -- session 1
Commit complete.
gSQL> SELECT * FROM T1 WHERE C1 = 1;    // -- session 1
C1
--
1
1 row selected.
```

수정 전 대처

동일한 세션에서 domain 트랜잭션을 수행한 후에 select를 수행할 경우, SELECT 문을 SELECT FOR UPDATE로 대체해서 수행한다.

ISSUE-2965 JDBC에서 BigDecimal 타입을 parameter로 사용하면 데이터가 손실된다.

개요

Double 타입의 정밀도를 벗어나는 값을 BigDecimal 타입으로 parameter로 사용하면 데이터가 손실되는 문제를 수정하였다.

현상 및 증상

사용자의 의도와 다르게 데이터 손실이 발생한다.

```
gSQL> CREATE TABLE T1 ( C1 NUMBER );
Table created.
```

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES (?)");
pstmt.setBigDecimal(1, new BigDecimal("12345678901234567890.123456789"));
pstmt.executeUpdate();
```

```
gSQL> select * from t1;
               C1
-----
12345678901234600000
1 row selected.
```

수정 전 대처

BigDecimal 타입 대신 문자열로 처리한다.

```
pstmt.setString(1, "12345678901234567890.123456789");
```

```
gSQL> \set numsize 40
gSQL> select * from t1;
               C1
-----
12345678901234567890.123456789
1 row selected.
```

ISSUE-2955 Cluster 환경에서 gsqlnet이 cstartup 또는 cshutdown을 연속적으로 실행할 수 없다.

개요

gsqlnet은 ODBC를 통해 location 파일과 glocator의 접속 정보를 구축한다.

이 정보는 cstartup이나 cshutdown 명령을 처음으로 수행할 때 구축된다.

cstartup이나 cshutdown 명령을 두 번째로 수행하면 정보 구축 과정을 무시하는데 이 때 flag 설정이 잘못되어 에러가 발생하였다.

현상 및 증상

gsqlnet 프로세스에서 cstartup이나 cshutdown을 수행 후 다시 cshutdown 또는 cstartup을 수행하면 에러가 발생한다.

수정 전 대처

gsqlnet 세션을 재시작하여 cstartup 또는 cshutdown 명령을 해야 한다.

ISSUE-2943 Cluster 환경에서 query timeout 발생 시 agable scn이 증가하지 않는다.

개요

클러스터 환경에서 DML이 async로 처리될 때 원격 멤버들의 view scn 정보를 세션에 설정한다. 이 때 원격 멤버들 중 단 하나에게도 DML을 전송하지 못한 상태에서 query timeout과 같은 예외 상황이 발생하면 세션에 설정된 원격 멤버들의 view scn 정보가 초기화 되지 않는 버그가 있었다. 이로 인해 시스템에 수행 중인 statement가 없는데도 해당 세션이 연결되어 있는 동안 시스템의 agable scn이 증가하지 않는 문제가 있었다.

Async 처리에 예외 상황이 발생하면 원격 멤버들의 view scn 정보를 초기화하여 문제가 발생하지 않도록 수정하였다.

현상 및 증상

클러스터 환경에서 query timeout이 발생한 후에 agable scn이 멈추게 되어 undo, data 공간이 부족해진다.

수정 전 대처

Query timeout이 발생한 세션을 종료한다.

ISSUE-2927 트랜잭션 슬롯 부족으로 인해 deadlock이 발생한다.

개요

트랜잭션 슬롯이 부족한 상황에서 모든 서버 프로세스들이 트랜잭션 슬롯을 할당하는 경우 기존 트랜잭션들이 슬롯을 해제하지 않으면 hang이 발생할 수 있다. 이 경우 정해진 시간이 경과하면 해당 패치에 에러가 발생하도록 수정하였다.

현상 및 증상

동시에 여러 세션에서 transaction이 발생할 때 transaction slot이 부족하여 hang이 발생한다.

수정 전 대처

없음

ISSUE-2922 ODBC의 statement 속성에 SQL_ATTR_FETCH_FAILOVER를 추가하였다.

개요

ODBC의 statement 속성에 SQL_ATTR_FETCH_FAILOVER를 추가하였으며 다음과 같은 값을 설정할 수 있다.

- SQL_FETCH_FAILOVER_OFF
- SQL_FETCH_FAILOVER_ON

```
SQLSetStmtAttr( stmt,
                 SQL_ATTR_FETCH_FAILOVER,
                 (SQLPOINTER)SQL_FETCH_FAILOVER_ON,
                 0 )
```

현상 및 증상

없음

수정 전 대처

없음

ISSUE-2513 EXEC SQL AT :sConn DISCONNECT에서 VARCHAR 타입을 인식하지 못한다.

개요

gpec이 EXEC SQL AT 절에서 VARCHAR 타입을 char 타입으로 인식한다.

현상 및 증상

VARCHAR 타입 변수를 EXEC SQL AT 절에 사용했을 때 gpec이 VARCHAR 타입이 아닌 char 타입으로 처리한다.

- Example gc file

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR sAT[20];
EXEC SQL END DECLARE SECTION;
EXEC SQL AT :sAT DISCONNECT;
```

- Example c file generated by gpec

```
sqlargs.conn = (char *)sAT;
sqlargs.sql_ca = &sqlca;
sqlargs.sql_state = SQLSTATE;
sqlargs.sqltype = 35;
```

```
sqlargs.sqlfn = (char *)__FILE__;
sqlargs.sqlln = __LINE__;
sqlargs.sqlstmt = NULL;
sqlargs.atomic = 0;
sqlargs.unsafenull = 0;
sqlargs.iters = 0;
DBESQL_Disconnect(NULL, &sqlargs, NULL, 0);
```

VARCHAR 타입을 char * 타입으로 처리하여 gpec이 생성한 c 파일을 컴파일하는 과정에서 에러가 발생한다.

수정 전 대처

VARCHAR 타입 대신 char 타입을 사용한다.

ISSUE-2349 gpec이 전처리를 처리하는 경우 __LINE__ 매크로가 잘못된 라인을 가리킨다.

개요

gpec이 #if와 같은 전처리를 처리할 때 __LINE__ 매크로가 잘못된 라인을 가리킨다.

현상 및 증상

gpec이 #if, #ifdef, #ifndef, #else, #elif 전처리를 처리할 때 주석을 추가하면서 원래 의도하였던 __LINE__ 매크로 값이 아닌 잘못된 값이 도출된다.

- Example gc file

```
#if 0
    printf("[%s:%d] if\n", __FILE__, __LINE__);
#else
    printf("[%s:%d] else\n", __FILE__, __LINE__);
#endif
printf("[%s:%d] endif\n", __FILE__, __LINE__);
```

gpec이 해당 코드를 처리한 결과는 다음과 같다.

- Example c file generated by gpec

```
#if 0
/*Macro condition FALSE*/
    printf("[%s:%d] if\n", __FILE__, __LINE__);
#else
```

```
/*Macro condition TRUE*/
    printf("[%s:%d] else\n", __FILE__, __LINE__);
#endif
    printf("[%s:%d] endif\n", __FILE__, __LINE__);
```

gpec이 생성한 c 파일을 실행하면 의도하지 않은 라인이 출력된다.

```
$ ./pp_bug
[pp_bug.gc:9] else
[pp_bug.gc:11] endif
```

정상적인 __LINE__ 매크로 값은 다음과 같다.

```
$ ./pp_bug
[pp_bug.gc:7] else
[pp_bug.gc:9] endif
```

수정 전 대처

없음

3.2.1 Patch Note

ISSUE-2902 Cluster 환경에서 global sequence 값을 참조할 때 deadlock이 발생한다.

개요

Cluster 환경에서 global sequence 값 참조 시 local member에 캐싱된 값이 고갈되어 다음 값을 구하기 위해 모든 cluster member 들에 global sequence latch를 잡아야 하는데, 이 과정에서 deadlock 이 발생하여 수정하였다.

현상 및 증상

동시에 여러 세션에서 global sequence 값을 참조하는 statement를 반복해서 수행할 때 deadlock으로 인해 hang 이 발생한다.

수정 전 대처

없음

파트 II.

Administration Manual

5. GOLDDILOCKS 데이터베이스 관리 기본	211
5.1 GOLDDILOCKS 데이터베이스 생성과 구성	212
데이터베이스 생성	212
초기 Property 설정	212
GOLDDILOCKS 설정파일을 이용한 초기 Property 관리	213
5.2 GOLDDILOCKS Instance의 시작과 종료	215
Instance 시작	215
Instance 종료	220
5.3 프로세스 관리	222
마스터 프로세스	222
리스너 프로세스	227
5.4 메모리 관리	228
GOLDDILOCKS 메모리 구조	228
SSA 관리	228
PSA 관리	229
5.5 모니터링	230
TRACE 파일을 이용한 모니터링	230
View를 이용한 성능 모니터링	241
6. GOLDDILOCKS 데이터베이스의 구조 및 저장 구조	243
6.1 제어 파일 관리	244
제어 파일 내용	244
제어 파일 다중화	246
제어 파일 훼손 시 대처 방안	246
제어 파일 정보	246
6.2 Redo Log File 관리	248
Redo Log File 구조	248
Redo Log 그룹과 멤버	249
Redo Log File 훼손 시 대처 방안	250
Redo Log File 정보	251
6.3 Archive Redo Log File 관리	252
Archive Redo Log File 생성	252
Archive Redo Log File 보존과 삭제	252
Archive Redo Log File 디렉토리 다중화	253
6.4 테이블스페이스 관리	254
테이블스페이스 종류	254
테이블스페이스 및 데이터 파일 관리	254
6.5 데이터 파일 관리	260
데이터 파일 정합성	260
데이터 파일 정보	261

7. GOLDILOCKS 데이터베이스의 백업과 복구	263
7.1 ARCHIVELOG 모드	264
ARCHIVELOG 모드	264
NOARCHIVELOG 모드	264
7.2 백업과 복구	266
백업	266
복구	270
8. GOLDILOCKS 데이터베이스 이중화	285
8.1 이중화 소개	286
8.2 운영 방법	287
CYCLONE	287
LOGMIRROR	292
8.3 Trace Log	294
CYCLONE 에러 메시지와 해결 방법	294
LOGMIRROR 에러 메시지와 해결 방법	295
9. Database Information	297
9.1 DICTIONARY_SCHEMA	298
ALL 계열 View	299
DBA 계열 View	362
USER 계열 View	419
기타 View	479
Aliased Synonym	494
9.2 INFORMATION_SCHEMA	495
COLUMNS	496
COLUMN_PRIVILEGES	499
CONSTRAINT_COLUMN_USAGE	500
CONSTRAINT_TABLE_USAGE	501
INFORMATION_SCHEMA_CATALOG_NAME	502
KEY_COLUMN_USAGE	503
PARAMETERS	504
REFERENTIAL_CONSTRAINTS	506
ROUTINES	507
ROUTINE_PRIVILEGES	511
ROUTINE_ROUTINE_USAGE	512
ROUTINE_SEQUENCE_USAGE	513
ROUTINE_TABLE_USAGE	514
SCHEMATA	515
SEQUENCES	516

SQL_FEATURES	517
SQL_IMPLEMENTATION_INFO	518
SQL_PACKAGES	519
SQL_PARTS	520
SQL_SIZING	521
STATISTICS	522
TABLES	523
TABLE_CONSTRAINTS	524
TABLE_PRIVILEGES	525
USAGE_PRIVILEGES	526
VIEWS	527
VIEW_ROUTINE_USAGE	528
VIEW_TABLE_USAGE	529
9.3 PERFORMANCE_VIEW_SCHEMA	530
GV\$ Global View	532
V\$AGABLE_INFO	534
V\$ARCHIVELOG	535
V\$AUDITABLE_DB_PRIVILEGES	536
V\$AUDITABLE_SYSTEM_ACTIONS	537
V\$BACKUP	538
V\$BALANCER	539
V\$CLUSTER_DISPATCHER	540
V\$CLUSTER_LOCATION	541
V\$CLUSTER_MEMBER	542
V\$COLUMNS	543
V\$CONTROLFILE	544
V\$DATAFILE	545
V\$DB_FILE	546
V\$DISPATCHER	547
V\$ERROR_CODE	548
V\$GLOBAL_TRANSACTION	549
V\$INCREMENTAL_BACKUP	550
V\$INSTANCE	551
V\$JOURNALING	552
V\$KEYWORDS	553
V\$LATCH	554
V\$LOGFILE	555
V\$LOCK_WAIT	556
V\$PROCESS_STAT	557

V\$PROCESS_MEM_STAT	558
V\$PROCESS_SQL_STAT	559
V\$PROPERTY	560
V\$PSM_RESERVED_WORDS	561
V\$QUEUE	562
V\$RESERVED_WORDS	563
V\$SESSION	564
V\$SESSION_AUDIT	565
V\$SESSION_CONNECT_INFO	566
V\$SESSION_EVENT	567
V\$SESSION_STAT	568
V\$SESSION_MEM_STAT	569
V\$SESSION_SQL_STAT	570
V\$SESSION_WAIT	571
V\$SHARED_MODE	572
V\$SHARED_SERVER	573
V\$SHM_SEGMENT	574
V\$SPROPERTY	575
V\$SQLFN_METADATA	576
V\$SQL_CACHE	577
V\$SQL_COMMAND	578
V\$SQL_HISTORY	579
V\$STATEMENT	580
V\$SYSTEM_EVENT	581
V\$SYSTEM_STAT	582
V\$SYSTEM_MEM_STAT	583
V\$SYSTEM_SQL_STAT	584
V\$TABLES	585
V\$TABLESPACE	586
V\$TABLESPACE_STAT	587
V\$TRANSACTION	588
V\$WAIT_EVENT_CLASS_NAME	589
V\$WAIT_EVENT_NAME	590
V\$XA_TRANSACTION	591
10. Server Property	593
10.1 Server Property 정보	594
10.2 AGING_INTERVAL	595
기본 정보	595
설명	595

10.3	AGING_PLAN_INTERVAL	596
	기본 정보	596
	설명	596
10.4	ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10	597
	기본 정보	597
	설명	597
10.5	ARCHIVELOG_FILE	598
	기본 정보	598
	설명	598
10.6	ARCHIVELOG_MODE	599
	기본 정보	599
	설명	599
10.7	BACKUP_DIR_1 ~ BACKUP_DIR_10	600
	기본 정보	600
	설명	600
10.8	BLOCK_READ_COUNT	601
	기본 정보	601
	설명	601
10.9	BULK_IO_PAGE_COUNT	602
	기본 정보	602
	설명	602
10.10	CDISPATCHER_HOT_POLICY_INTERVAL	603
	기본 정보	603
	설명	603
10.11	CDISPATCHER_SOCKET_BUFFER_SIZE	604
	기본 정보	604
	설명	604
10.12	CDISPATCHER_SYNC_THREADS	605
	기본 정보	605
	설명	605
10.13	CDISPATCHER_THREADS	606
	기본 정보	606
	설명	606
10.14	CHARACTER_SET	607
	기본 정보	607
	설명	607
10.15	CHAR_LENGTH_UNITS	608
	기본 정보	608
	설명	608

10.16	CHECK_DEDICATE_CONNECTION_INTERVAL	610
	기본 정보	610
	설명	610
10.17	CLIENT_MAX_COUNT	611
	기본 정보	611
	설명	611
10.18	CLIENT_NUMA_POLICY	612
	기본 정보	612
	설명	612
10.19	CLOSE_PSM_CHILD_STMTS	613
	기본 정보	613
	설명	613
10.20	CLUSTER_ASYNC_COMMIT	614
	기본 정보	614
	설명	614
10.21	CLUSTER_ASYNC_REPLICATION	615
	기본 정보	615
	설명	615
10.22	CLUSTER_CM_BUFFER_COUNT	616
	기본 정보	616
	설명	616
10.23	CLUSTER_CM_BUFFER_SIZE	617
	기본 정보	617
	설명	617
10.24	CLUSTER_CM_READ_BUFFER_SIZE	618
	기본 정보	618
	설명	618
10.25	CLUSTER_COMMIT_SLAVES	619
	기본 정보	619
	설명	619
10.26	CLUSTER_COMMIT_STREAM_ISOLATION	620
	기본 정보	620
	설명	620
10.27	CLUSTER_CONNECTION	621
	기본 정보	621
	설명	621
10.28	CLUSTER_CONNECTION_TIMEOUT_SEC	622
	기본 정보	622
	설명	622

10.29	CLUSTER_DATA_SYNC_SERVERS	623
	기본 정보	623
	설명	623
10.30	CLUSTER_DISPATCHER_IN_QUEUE_SIZE	624
	기본 정보	624
	설명	624
10.31	CLUSTER_DISPATCHER_NUMA_STREAM_MAP	625
	기본 정보	625
	설명	625
10.32	CLUSTER_DISPATCHER_OUT_QUEUE_SIZE	626
	기본 정보	626
	설명	626
10.33	CLUSTER_HEARTBEAT_INTERVAL	627
	기본 정보	627
	설명	627
10.34	CLUSTER_HEARTBEAT_RETRY_COUNT	628
	기본 정보	628
	설명	628
10.35	CLUSTER_IGNORE_INACTIVE_MEMBER	629
	기본 정보	629
	설명	629
10.36	CLUSTER_MAX_PACKET_SIZE	630
	기본 정보	630
	설명	630
10.37	CLUSTER_MAX_PAYLOAD_SIZE	631
	기본 정보	631
	설명	631
10.38	CLUSTER_PACKET_ALLOCATION_TIMEOUT	632
	기본 정보	632
	설명	632
10.39	CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT	633
	기본 정보	633
	설명	633
10.40	CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT	634
	기본 정보	634
	설명	634
10.41	CLUSTER_SERVER_RESPONSE_QUEUE_SIZE	635
	기본 정보	635
	설명	635

10.42	CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY	636
	기본 정보	636
	설명	636
10.43	CLUSTER_SPLIT_BRAIN_RETRY_COUNT	637
	기본 정보	637
	설명	637
10.44	COMMITTER_HOT_POLICY_INTERVAL	638
	기본 정보	638
	설명	638
10.45	CONTROL_FILE_0 ~ CONTROL_FILE_7	639
	기본 정보	639
	설명	639
10.46	CONTROL_FILE_COUNT	640
	기본 정보	640
	설명	640
10.47	CONTROL_FILE_TEMP_NAME	641
	기본 정보	641
	설명	641
10.48	COORDINATOR_COMMIT_WRITE_MODE	642
	기본 정보	642
	설명	642
10.49	CSERVERS	643
	기본 정보	643
	설명	643
10.50	DATABASE_ACCESS_MODE	644
	기본 정보	644
	설명	644
10.51	DATABASE_INSTANCE_NAME	645
	기본 정보	645
	설명	645
10.52	DATA_STORE_MODE	646
	기본 정보	646
	설명	646
10.53	DA_CLIENT_NUMA_NODE	647
	기본 정보	647
	설명	647
10.54	DDL_AUTOCOMMIT	648
	기본 정보	648
	설명	648

10.55	DDL_LOCK_TIMEOUT	649
	기본 정보	649
	설명	649
10.56	DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION	650
	기본 정보	650
	설명	650
10.57	DEFAULT_INDEX_LOGGING	651
	기본 정보	651
	설명	651
10.58	DEFAULT_INDEX_PCTFREE	652
	기본 정보	652
	설명	652
10.59	DEFAULT_INITRANS	653
	기본 정보	653
	설명	653
10.60	DEFAULT_MAXTRANS	654
	기본 정보	654
	설명	654
10.61	DEFAULT_PCTFREE	655
	기본 정보	655
	설명	655
10.62	DEFAULT_PCTUSED	656
	기본 정보	656
	설명	656
10.63	DEFAULT_REMOVAL_BACKUP_FILE	657
	기본 정보	657
	설명	657
10.64	DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST	658
	기본 정보	658
	설명	658
10.65	DEFAULT_SHARDING	659
	기본 정보	659
	설명	659
10.66	DISABLE_DDL_CDC_GIVEUP	662
	기본 정보	662
	설명	662
10.67	DISABLE_UPDATE_PK_CDC_GIVEUP	663
	기본 정보	663
	설명	663

10.68	DISALLOWED_PROTOCOL_TARGETTYPE	664
	기본 정보	664
	설명	664
10.69	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL	665
	기본 정보	665
	설명	665
10.70	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME	666
	기본 정보	666
	설명	666
10.71	DISPATCHER_CM_BUFFER_SIZE	667
	기본 정보	667
	설명	667
10.72	DISPATCHER_CM_UNIT_SIZE	668
	기본 정보	668
	설명	668
10.73	DISPATCHER_CONNECTIONS	669
	기본 정보	669
	설명	669
10.74	DISPATCHER_HOT_POLICY_INTERVAL	670
	기본 정보	670
	설명	670
10.75	DISPATCHER_LOAD_BALANCING	671
	기본 정보	671
	설명	671
10.76	DISPATCHER_NUMA_STREAM_MAP	672
	기본 정보	672
	설명	672
10.77	DISPATCHER_QUEUE_SIZE	673
	기본 정보	673
	설명	673
10.78	DISPATCHER_REQUEST_MINI_QUEUE_COUNT	674
	기본 정보	674
	설명	674
10.79	DISPATCHER_RESPONSE_MINI_QUEUE_COUNT	675
	기본 정보	675
	설명	675
10.80	DISPATCHERS	676
	기본 정보	676
	설명	676

10.81	FETCH_FAILOVER	677
	기본 정보	677
	설명	677
10.82	GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY	678
	기본 정보	678
	설명	678
10.83	GLOBAL_JOURNAL_BUFFER_SIZE	679
	기본 정보	679
	설명	679
10.84	GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE	680
	기본 정보	680
	설명	680
10.85	GLOBAL_PROPERTY_LOCK_TIMEOUT	681
	기본 정보	681
	설명	681
10.86	GLOBAL_TRANSACTION_COMMIT_WRITE_MODE	682
	기본 정보	682
	설명	682
10.87	GLOBAL_TRANSACTION_ISOLATION_SCOPE	683
	기본 정보	683
	설명	683
10.88	GLOBAL_TRANSACTION_LOG_DIR	684
	기본 정보	684
	설명	684
10.89	GLOBAL_TRANSACTION_LOG_FILE_SIZE	685
	기본 정보	685
	설명	685
10.90	GMASTER_NUMA_NODE	686
	기본 정보	686
	설명	686
10.91	GMON_AUTOSTART	687
	기본 정보	687
	설명	687
10.92	HINT_ERROR	688
	기본 정보	688
	설명	688
10.93	IDLE_TIMEOUT	689
	기본 정보	689
	설명	689

10.94	INDEX_BUILD_PARALLEL_FACTOR	690
	기본 정보	690
	설명	690
10.95	INDEX_TREE_MERGE_PARALLEL_FACTOR	691
	기본 정보	691
	설명	691
10.96	INST_ALLOCATOR_COUNT	692
	기본 정보	692
	설명	692
10.97	INST_TABLE_BLOCK_SIZE	693
	기본 정보	693
	설명	693
10.98	IN_DOUBT_DECISION	694
	기본 정보	694
	설명	694
10.99	JOURNAL_TEMP_DIR	695
	기본 정보	695
	설명	695
10.100	KEEPALIVE_IDLE_TIME	696
	기본 정보	696
	설명	696
10.101	LOCAL_CLUSTER_MEMBER	697
	기본 정보	697
	설명	697
10.102	LOCAL_CLUSTER_MEMBER_HOST	698
	기본 정보	698
	설명	698
10.103	LOCAL_CLUSTER_MEMBER_PORT	699
	기본 정보	699
	설명	699
10.104	LOCAL_JOURNAL_BUFFER_SIZE	700
	기본 정보	700
	설명	700
10.105	LOCATION_FILE	701
	기본 정보	701
	설명	701
10.106	LOCATOR_QUERY_TIMEOUT	702
	기본 정보	702
	설명	702

10.107	LOCK_HASH_TABLE_SIZE	703
	기본 정보	703
	설명	703
10.108	LOG_BLOCK_SIZE	704
	기본 정보	704
	설명	704
10.109	LOG_BUFFER_SIZE	705
	기본 정보	705
	설명	705
10.110	LOG_DIR	706
	기본 정보	706
	설명	706
10.111	LOG_FILE_SIZE	707
	기본 정보	707
	설명	707
10.112	LOG_GROUP_COUNT	708
	기본 정보	708
	설명	708
10.113	LOG_MIRROR_MODE	709
	기본 정보	709
	설명	709
10.114	LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	710
	기본 정보	710
	설명	710
10.115	LOG_MIRROR_TIMEOUT	711
	기본 정보	711
	설명	711
10.116	LOG_SYNC_INTERVAL	712
	기본 정보	712
	설명	712
10.117	LOG_SYNC_INTERVAL_MSEC	713
	기본 정보	713
	설명	713
10.118	MAX_GROUP_COUNT	714
	기본 정보	714
	설명	714
10.119	MAX_JOURNAL_FILE_SIZE	715
	기본 정보	715
	설명	715

10.120	MAX_NODE_COUNT	716
	기본 정보	716
	설명	716
10.121	MAXIMUM_CONCURRENT_ACTIVITIES	717
	기본 정보	717
	설명	717
10.122	MAXIMUM_FLANGE_COUNT	718
	기본 정보	718
	설명	718
10.123	MAXIMUM_FLUSH_LOG_BLOCK_COUNT	719
	기본 정보	719
	설명	719
10.124	MAXIMUM_FLUSH_PAGE_COUNT	720
	기본 정보	720
	설명	720
10.125	MAXIMUM_JOURNAL_REPLAY_COUNT	721
	기본 정보	721
	설명	721
10.126	MAXIMUM_NAMED_CURSOR_COUNT	722
	기본 정보	722
	설명	722
10.127	MAXIMUM_SESSION_CM_BUFFER_SIZE	724
	기본 정보	724
	설명	724
10.128	MEASURE_CLUSTER_LATENCY	725
	기본 정보	725
	설명	725
10.129	MEMORY_MERGE_RUN_COUNT	726
	기본 정보	726
	설명	726
10.130	MEMORY_SORT_RUN_SIZE	727
	기본 정보	727
	설명	727
10.131	MINIMUM_UNDO_PAGE_COUNT	728
	기본 정보	728
	설명	728
10.132	MIN_SAMPLE_ROW_COUNT	729
	기본 정보	729
	설명	729

10.133	NET_BUFFER_SIZE	730
	기본 정보	730
	설명	730
10.134	NLS_DATE_FORMAT	731
	기본 정보	731
	설명	731
10.135	NLS_TIME_FORMAT	732
	기본 정보	732
	설명	732
10.136	NLS_TIME_WITH_TIME_ZONE_FORMAT	733
	기본 정보	733
	설명	733
10.137	NLS_TIMESTAMP_FORMAT	734
	기본 정보	734
	설명	734
10.138	NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT	735
	기본 정보	735
	설명	735
10.139	NUMA	736
	기본 정보	736
	설명	736
10.140	NUMA_MAP	737
	기본 정보	737
	설명	737
10.141	OFFLINE_MEMBER_AFTER_FAILOVER	738
	기본 정보	738
	설명	738
10.142	ONLINE_JOURNAL_REPLAY_THRESHOLD	739
	기본 정보	739
	설명	739
10.143	OS_GROUP_ACCESS	740
	기본 정보	740
	설명	740
10.144	PACKET_COMPRESSION_THRESHOLD	741
	기본 정보	741
	설명	741
10.145	PAGE_CHECKSUM_TYPE	742
	기본 정보	742
	설명	742

10.146	PARALLEL_IO_FACTOR	743
	기본 정보	743
	설명	743
10.147	PARALLEL_IO_GROUP_1 ~ PARALLEL_IO_GROUP_16	744
	기본 정보	744
	설명	744
10.148	PARALLEL_LOAD_FACTOR	745
	기본 정보	745
	설명	745
10.149	PENDING_LOG_BUFFER_COUNT	746
	기본 정보	746
	설명	746
10.150	PLAN_CACHE	747
	기본 정보	747
	설명	747
10.151	PLAN_CACHE_SIZE	748
	기본 정보	748
	설명	748
10.152	PRIVATE_STATIC_AREA_SIZE	749
	기본 정보	749
	설명	749
10.153	PROCESS_MAX_COUNT	750
	기본 정보	750
	설명	750
10.154	QUERY_TIMEOUT	751
	기본 정보	751
	설명	751
10.155	READABLE_ARCHIVELOG_DIR_COUNT	752
	기본 정보	752
	설명	752
10.156	READABLE_BACKUP_DIR_COUNT	753
	기본 정보	753
	설명	753
10.157	REBALANCE_BLOCK_READ_COUNT	754
	기본 정보	754
	설명	754
10.158	RECOMPILE_CHECK_MINIMUM_PAGE_COUNT	755
	기본 정보	755
	설명	755

10.159	RECOMPILE_PAGE_PERCENT	756
	기본 정보	756
	설명	756
10.160	RECOVERY_LOG_BUFFER_SIZE	757
	기본 정보	757
	설명	757
10.161	REDO_LOG_COMPRESSION_THRESHOLD	758
	기본 정보	758
	설명	758
10.162	REFINE_RELATION	759
	기본 정보	759
	설명	759
10.163	SESSION_FATAL_BEHAVIOR	760
	기본 정보	760
	설명	760
10.164	SESSION_MEMORY_INIT_SIZE	761
	기본 정보	761
	설명	761
10.165	SESSION_MEMORY_SHRINK_THRESHOLD	762
	기본 정보	762
	설명	762
10.166	SHARED_MEMORY_ADDRESS	763
	기본 정보	763
	설명	763
10.167	SHARED_MEMORY_STATIC_KEY	764
	기본 정보	764
	설명	764
10.168	SHARED_MEMORY_STATIC_NAME	765
	기본 정보	765
	설명	765
10.169	SHARED_MEMORY_STATIC_SIZE	766
	기본 정보	766
	설명	766
10.170	SHARED_REQUEST_QUEUE_COUNT	767
	기본 정보	767
	설명	767
10.171	SHARED_SERVERS	768
	기본 정보	768
	설명	768

10.172	SHARED_SESSION	769
	기본 정보	769
	설명	769
10.173	SNAPSHOT_STATEMENT_TIMEOUT	770
	기본 정보	770
	설명	770
10.174	SQL_HISTORY_SIZE	771
	기본 정보	771
	설명	771
10.175	SQL_HISTORY_TYPE	772
	기본 정보	772
	설명	772
10.176	SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	773
	기본 정보	773
	설명	773
10.177	SYSTEM_LOGGER_DIR	774
	기본 정보	774
	설명	774
10.178	SYSTEM_MEMORY_AUX_TABLESPACE_SIZE	775
	기본 정보	775
	설명	775
10.179	SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	776
	기본 정보	776
	설명	776
10.180	SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	777
	기본 정보	777
	설명	777
10.181	SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	778
	기본 정보	778
	설명	778
10.182	SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	779
	기본 정보	779
	설명	779
10.183	SYSTEM_TABLESPACE_DIR	780
	기본 정보	780
	설명	780
10.184	SYSTEM_UDS_DIR	781
	기본 정보	781
	설명	781

10.185	TCP_CLIENT_NUMA_NODE	782
	기본 정보	782
	설명	782
10.186	TCP_NODELAY	783
	기본 정보	783
	설명	783
10.187	TEMP_SEGMENT_CACHE_SIZE	784
	기본 정보	784
	설명	784
10.188	TEMP_UNDO_ENABLED	785
	기본 정보	785
	설명	785
10.189	TIMED_STATISTICS	786
	기본 정보	786
	설명	786
10.190	TIMEZONE	787
	기본 정보	787
	설명	787
10.191	TRACE_ALTER_SYSTEM	788
	기본 정보	788
	설명	788
10.192	TRACE_DDL	789
	기본 정보	789
	설명	789
10.193	TRACE_LOG_ID	790
	기본 정보	790
	설명	790
10.194	TRACE_LOG_MSGBUF_SIZE	791
	기본 정보	791
	설명	791
10.195	TRACE_LOG_TIME_DETAIL	792
	기본 정보	792
	설명	792
10.196	TRACE_LOGGER	793
	기본 정보	793
	설명	793
10.197	TRACE_LOGGER_REMOTE_HOST	794
	기본 정보	794
	설명	794

10.198	TRACE_LOGGER_REMOTE_PORT	795
	기본 정보	795
	설명	795
10.199	TRACE_LOGIN	796
	기본 정보	796
	설명	796
10.200	TRACE_LONG_RUN_CURSOR	797
	기본 정보	797
	설명	797
10.201	TRACE_LONG_RUN_SQL	799
	기본 정보	799
	설명	799
10.202	TRACE_XA	800
	기본 정보	800
	설명	800
10.203	TRANSACTION_ALLOCATION_TIMEOUT	801
	기본 정보	801
	설명	801
10.204	TRANSACTION_COMMIT_WRITE_MODE	802
	기본 정보	802
	설명	802
10.205	TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT	803
	기본 정보	803
	설명	803
10.206	TRANSACTION_TABLE_SIZE	804
	기본 정보	804
	설명	804
10.207	TRANSACTION_TIMEOUT	806
	기본 정보	806
	설명	806
10.208	UNDO_RELATION_ALLOCATION_TIMEOUT	807
	기본 정보	807
	설명	807
10.209	UNDO_RELATION_COUNT	808
	기본 정보	808
	설명	808
10.210	UNDO_SHRINK_THRESHOLD	810
	기본 정보	810
	설명	810

10.211	USE_LARGE_PAGES	811
	기본 정보	811
	설명	811

5.

GOLDILOCKS 데이터베이스 관리 기본

5.1 GOLDILOCKS 데이터베이스 생성과 구성

데이터베이스 생성

GOLDILOCKS package에 포함된 gcreatedb를 이용하여 데이터베이스를 생성한다. 데이터베이스 생성에 앞서 다음과 같은 몇가지 사항을 고려해야 한다.

고려 사항	참고
데이터베이스에서 사용할 테이블, 인덱스가 사용할 공간의 크기를 고려해야 한다.	GOLDILOCKS 데이터베이스의 구조 및 저장 구조
데이터베이스의 파일들이 생성될 위치를 고려해야 한다. 파일들을 적절히 분배하여 disk IO를 분산함으로써 데이터베이스 성능을 향상시킬 수 있기 때문이다. 예를 들어, redo log file을 별도의 디스크에 배치하거나 스트라이핑하고 데이터 파일을 여러 개의 디스크로 분산시켜 disk IO를 분산시키고 병렬 disk IO를 수행할 수 있다.	Redo Log File 관리
서버 프로퍼티 파일에 설정된 각 프로퍼티의 개념과 동작을 숙지하고 지속적으로 관리해야 한다.	- 초기 Property 설정 - GOLDILOCKS 설정파일을 이용한 초기 Property 관리 10. Server Property

위의 고려 사항외에도 파트 I. **Getting Started** 매뉴얼의 **Database 생성** 부분을 참조하여 데이터베이스를 생성한다.

GOLDILOCKS를 사용하기 위해 환경 변수 \$GOLDILOCKS_HOME과 \$GOLDILOCKS_DATA를 설정해야 한다. GOLDILOCKS 데이터베이스를 생성하고 운용하기 위한 프로퍼티 파일 (goldilocks.properties.conf)은 \$GOLDILOCKS_DATA/conf에 존재한다.

- GOLDILOCKS_HOME: GOLDILOCKS package에 포함된 binary가 설치되는 위치이며 업데이트 할 때 overwrite 할 수 있다. (License는 백업해두어야 한다.)
- GOLDILOCKS_DATA: Log file 및 data file, control file 등 GOLDILOCKS 데이터베이스가 사용하는 파일들이 기본적으로 생성되는 경로이며 overwrite 할 수 없다.

초기 Property 설정

GOLDILOCKS에서는 property를 사용하여 운영, 관리에 필요한 정보를 제어할 수 있다.

초기 Property

데이터베이스를 생성하거나 시작할 때 다음과 같이 property를 설정할 수 있다.

1. System environment variable (시스템 환경 변수)
 - 가. GOLDILOCKS가 설치된 후 데이터베이스를 생성하거나 구동하는 명령어창에 입력하여 변경한다.
 - 나. 프로퍼티 이름 앞에 반드시 prefix를 붙여야 한다. Prefix는 GOLDILOCKS_이다.
- 다음은 SHARED_MEMORY_STATIC_SIZE를 100 M로 변경하는 예이다.

```
export GOLDILOCKS_SHARED_MEMORY_STATIC_SIZE=100M
```

2. Property file (프로퍼티 파일): 프로퍼티 파일에서 직접 변경할 수 있으며 prefix는 필요하지 않다.
- 다음은 SHARED_MEMORY_STATIC_SIZE를 200 M로 변경하는 예이다.
- Shared memory static size (100 M ~ 32 G)

```
SHARED_MEMORY_STATIC_SIZE = 200M
```

노트

만약 동일한 프로퍼티를 시스템 환경 변수와 프로퍼티 파일에 설정할 경우 프로퍼티 파일 값이 적용된다. 즉, 시스템 환경 변수로 SHARED_MEMORY_STATIC_SIZE를 100 M로 설정하고, 프로퍼티 파일에 200 M로 설정하였다면 데이터베이스를 구동할 때 SHARED_MEMORY_STATIC_SIZE는 200 M로 적용된다.

GOLDILOCKS 설정파일을 이용한 초기 Property 관리

Property file은 \$GOLDILOCKS_DATA/conf에 위치하며 파일 형식에 따라 두 가지로 구분된다.

- Text property file (텍스트 프로퍼티 파일)
 - 파일이름: goldilocks.properties.conf
 - "프로퍼티 이름 = 값"의 형식으로 구성된 파일이며 사용자가 직접 편집할 수 있다.
 - 만약 binary property file이 존재할 경우에는 text property file을 읽지 않는다.
- Binary property file (바이너리 프로퍼티 파일)
 - 시스템에 의해 생성되는 파일이며 사용자가 직접 편집할 수 없다.
 - 파일이름: goldilocks.properties.binary
 - 해당 바이너리 파일의 내용을 조회하기 위해서는 gdump 툴을 이용해야 한다.

노트

텍스트 프로퍼티 파일과 바이너리 프로퍼티 파일이 함께 존재할 경우에는 바이너리 프로퍼티 파일만 읽는다. 즉, 텍스트 프로퍼티 파일은 아무런 처리를 하지 않는다. 이런 바이너리 프로퍼티 파일은 사용자의 SQL (ALTER SYSTEM SET)에 의해서 관리되며, 편집 또한 SQL을 통해서만 가능하다.

자세한 내용은 **ALTER SYSTEM SET property_name, ALTER SYSTEM RESET property_name**을 참조한다.

- Text property file 편집
 - PROPERTY_NAME = VALUE 형식으로 사용해야 한다.
 - 주석은 '#'을 사용한다.
 - Property는 다음과 같은 세 가지 데이터 타입 중 하나를 갖는다.
 - 문자형: Single quote (')를 사용하여 설정해야 한다. 문자형의 값이 \$GOLDDILOCKS_DATA의 경로를 포함할 경우 <GOLDDILOCKS_DATA>로 사용해야 한다.
 - 숫자형: 계산식 (예: LOG_BUFFER_SIZE=1024 * 1024 (X))은 사용할 수 없다. 경우에 따라 K (kilobyte), M (megabyte), G (gigabyte), T (terabyte), P (petabyte) 와 같은 사이즈를 선택할 수 있다.
 - 논리형: ON/OFF, ENABLE/DISABLE, 1/0, TRUE/FALSE, YES/NO를 사용할 수 있다.

5.2 GOLDILOCKS Instance의 시작과 종료

Instance의 시작과 종료에 대해 설명한다.

Instance 시작

GOLDILOCKS instance는 SYSDBA 권한을 가진 사용자만 구동할 수 있다.

Direct Attach (D/A) 방식과 Client/ Server (C/S)의 dedicated 방식으로 구동할 수 있고 C/S의 shared 방식으로 구동할 수 없다.

다단계 시작

GOLDILOCKS instance는 여러 단계를 거쳐 구동된다. 다단계 구동 기능은 각 단계별로 관리자가 개입하여 데이터베이스의 상태를 변경할 수 있도록 하기 위한 기능이다.

각 단계는 idle, nomount, mount, open으로 구분되며 단계별 특징은 다음과 같다.

Idle 단계

Instance가 구동되어 있지 않은 상태이다.

다음과 같이 instance가 구동되어 있지 않은 상태에서 gsql로 접속하면 idle instance에 접속되고 해당 단계에서는 \startup을 제외한 어떠한 서버 명령도 수행할 수 없다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> select * from dual;
ERR-08003(40044): connection does not exist
gSQL>
```

- Idle에서 관리자가 할 수 있는 작업
 - Instance 구동에 필요한 프로퍼티 조정
 - gsql에서 \startup을 이용한 nomount 전이
- Nomount 단계로 전이될 때 instance 내에서 수행되는 작업
 - GOLDILOCKS instance를 관리하는 데몬인 gmaster를 구동한다.
 - gmaster 내의 timer와 cleanup thread가 구동된다.
 - Shared memory Static Area (SSA)를 할당하고 초기화한다.

표 5-1 Nomount 단계로 전이될 때 적용되는 프로퍼티

이름	설명
CLIENT_MAX_COUNT	최대로 접속할 수 있는 세션 개수
CONTROL_FILE_0 ~ 7	제어 파일 경로
CONTROL_FILE_COUNT	제어 파일의 경로들 중 유효한 경로의 개수
DATA_STORE_MODE	GOLDDILOCKS 인스턴스의 저장모드
PLAN_CACHE_SIZE	플랜 캐시를 위한 공유 메모리의 최대 크기
PROCESS_MAX_COUNT	최대로 접속할 수 있는 프로세스 개수
SHARED_MEMORY_ADDRESS	공유 메모리 주소
SHARED_MEMORY_STATIC_NAME	공유 메모리 이름
SHARED_MEMORY_STATIC_KEY	공유 메모리가 생성되는 key 값
SHARED_MEMORY_STATIC_SIZE	생성되는 공유 메모리의 사이즈
SYSTEM_LOGGER_DIR	시스템 로거의 경로

Idle 상태에서는 프로퍼티를 변경할 수 없다. 관리자가 해당 프로퍼티들을 변경하려면 다음과 같은 방법을 사용한다.

환경 변수를 이용하는 방법으로 GOLDDILOCKS_[property_name]을 원하는 값으로 설정하고 nomount로 전이시키면 해당 프로퍼티가 적용된다.

```
% export GOLDDILOCKS_CLIENT_MAX_COUNT=1000
```

'SCOPE = FILE'을 이용하여 변경하고자 하는 프로퍼티 값을 파일에 기록한다. 기록된 프로퍼티는 nomount로 전이될 때 적용된다.

```
gSQL> alter system set client_max_count = 1000 scope = file;
System altered.
gSQL> \shutdown
Shutdown success
gSQL> \startup
Startup success
gSQL>
```

Nomount 단계

데이터베이스에 아직 mount 되지 않았고 GOLDDILOCKS instance를 관리하는 데몬인 gmaster 프로세스만 구동된 상태이다.

Idle 단계에서 nomount 단계로 전이시키는 방법은 다음과 같다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup nomount
Startup success
```

gSQL>

- Nomount에서 관리자가 할 수 있는 작업
 - Nomount 프로퍼티 조정
 - 자세한 내용은 **ALTER SYSTEM MOUNT DATABASE, ALTER DATABASE RESTORE CONTROLFILE FROM 'file_name'**을 참조한다.
- Mount 단계로 전이될 때 instance 내에서 수행되는 작업
 - 제어 파일 (control file)을 데이터베이스로 로딩한다.
 - 데이터베이스 복구를 위한 준비를 한다.
 - gmaster 내의 checkpoint, log flusher, page flusher, IO slave, archive log thread가 구동된다.

표 5-2 Nomount에서 변경할 수 있는 프로퍼티

이름	설명
DATABASE_ACCESS_MODE	데이터베이스 접근 모드 (READ ONLY, READ WRITE)
LOG_BUFFER_SIZE	Redo log buffer 사이즈
PARALLEL_LOAD_FACTOR	데이터베이스 로딩 후 병렬 작업을 위한 thread 개수
PARALLEL_IO_FACTOR	데이터베이스 로딩을 위한 병렬 thread 개수
PARALLEL_IO_GROUP_1 ~ 16	병렬 로딩할 때의 데이터 파일 그룹
PENDING_LOG_BUFFER_COUNT	지연 로그 버퍼의 개수
TRANSACTION_TABLE_SIZE	트랜잭션 테이블의 크기
UNDO_RELATION_COUNT	Undo 릴레이션의 개수

Mount 단계

데이터베이스에 mount 된 상태이고 control file을 데이터베이스에서 인식하고 있는 상태이다. 해당 단계에서는 control file 내의 모든 section들을 제어할 수 있다.

Nomount 단계에서 mount 단계로 전이시키는 방법은 다음과 같다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup nomount
Startup success
gSQL> alter system mount database;
System altered.
gSQL>
```

- Mount에서 관리자가 할 수 있는 작업
 - Mount 프로퍼티 조정
 - **ALTER SYSTEM {MOUNT | OPEN} DATABASE**
 - **ALTER DATABASE ADD LOGFILE**
 - **ALTER DATABASE DROP LOGFILE**

- ALTER DATABASE RENAME LOGFILE
 - ALTER DATABASE { ARCHIVELOG | NOARCHIVELOG }
 - ALTER DATABASE DELETE BACKUP
 - ALTER DATABASE REGISTER
 - ALTER DATABASE RECOVER
 - ALTER DATABASE RESTORE
 - ALTER SESSION SET property_name
 - ALTER SYSTEM RESET property_name
 - ALTER SYSTEM SWITCH LOGFILE
 - ALTER SYSTEM { KILL | DISCONNECT } SESSION
 - ALTER TABLESPACE name ADD [DATAFILE|MEMORY]
 - ALTER TABLESPACE name RENAME DATAFILE
 - ALTER TABLESPACE name [ONLINE|OFFLINE]
- Open 단계로 전이될 때 instance 내에서 수행되는 작업
 - Instance에서 사용하는 모든 데이터 파일을 공유 메모리로 로딩한다.
 - Instance 복구를 수행한다.
 - NOLOGGING 인덱스를 구축한다.
 - Ager thread가 삭제하지 못한 객체나 파일을 정리한다.
 - 딕셔너리 (dictionary) 객체를 위한 캐시를 구축한다.
 - "SHARED_SESSION" property가 YES로 설정된 경우 gmaster내의 process monitor thread가 구동된다.
 - Process monitor thread는 balancer process, dispatcher process, shared-server process를 실행한다.

표 5-3 Mount에서 변경할 수 있는 프로퍼티

이름	설명
ARCHIVELOG_FILE	Archive 파일의 prefix 이름
IN_DOUBT_DECISION	In-doubt 트랜잭션에 대한 의사결정
LOCK_HASH_TABLE_SIZE	Lock 관리자의 해시 테이블 크기
LOG_MIRROR_MODE	로그 미러링 모드
LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	로그 미러링을 위한 공유 메모리 크기
SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	데이터베이스 수준의 supplemental 로깅 여부

Open 단계

데이터베이스 내의 모든 데이터 파일이 메모리로 로딩되며 사용자에게 서비스를 제공할 수 있는 상태이다. 해당 단계에서는 모든 작업이 허용된다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup mount
Startup success
gSQL> alter system open database;
```

```
System altered.
gSQL>
```

Open 단계로 전이될 때 데이터베이스의 접근 모드를 선택할 수 있다. 데이터베이스 접근 모드는 READ_ONLY와 READ_WRITE 두 가지이며, READ_ONLY는 데이터베이스의 모든 데이터에 대한 읽기만 가능하고, READ_WRITE는 읽기와 갱신이 모두 가능하다.

이전에 데이터베이스가 정상적으로 종료된 경우에만 데이터베이스를 READ_ONLY 모드로 OPEN 할 수 있다. 데이터베이스를 OPEN 할 때 접근 모드가 생략될 경우, 프로퍼티 (DATABASE_ACCESS_MODE)를 참조한다.

```
gSQL> \shutdown abort
Shutdown success
gSQL> \startup mount
Startup success
gSQL> alter system open database read only;
ERR-42000(14038): unable to recover database in READ ONLY mode
gSQL> alter system open database read write;
System altered.
...
gSQL> \shutdown normal
Shutdown success
gSQL> \startup mount
Startup success
gSQL> alter system open database read only;
System altered.
```

진단

Instance를 구동할 때 하나의 \startup 명령어로 여러 단계를 한꺼번에 올릴 수 있다. 만약 특정 단계에서 구동에 실패한 경우, 관리자는 어떤 단계에서 실패했는지 알아야 한다. 현재 instance가 어느 단계까지 올라왔는지는 V\$INSTANCE를 이용해서 확인할 수 있고 관리자는 그 이후 단계부터 instance 구동을 계속 진행할 수 있다.

다음은 \startup에 실패했을 경우 instance를 open 단계까지 구동하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-42000(14051): media recovery required - 'TEST_TBS'
gSQL> select INSTANCE_STATUS from v$instance;
INSTANCE_STATUS
-----
MOUNTED
```

```
1 row selected.
...
gSQL> alter system open database;
System altered.
```

Instance 종료

GOLDBLOCKS instance는 SYSDBA 권한을 가진 사용자만 종료할 수 있고 종료하는 중에는 새로운 세션을 허용하지 않는다.

Direct Attach (D/A) 방식과 Client/ Server (C/S)의 dedicated 방식으로 종료할 수 있고 C/S의 shared 방식으로는 종료할 수 없다.

```
% gsql sys gliese --as sysdba
ERR-RD000(13034): Service is not available
```

Instance 종료에는 normal, immediate, transactional, abort라는 네 가지 모드가 있다.

Shutdown Normal

Shutdown normal은 shutdown 할 때 특정한 모드를 지정하지 않은 경우에 기본 모드로 동작하며 instance가 정상적으로 종료되기를 원할 경우에 사용한다.

```
gSQL> \shutdown normal
Shutdown success
gSQL>
```

Shutdown normal은 다음과 같은 특징을 갖는다.

- 새로운 세션을 허용하지 않는다.
- 이미 접속되어 있는 세션에서는 새로운 트랜잭션이나 statement들을 허용한다.
- Instance에 접속되어 있는 모든 세션이 종료되기를 기다린다.
- 이후 instance를 구동할 때 instance 복구를 수행하지 않는다.

Shutdown Transactional

Shutdown transactional은 진행 중인 세션을 강제 종료하더라도 이미 진행하고 있는 트랜잭션들을 정상적으로 종료하고 싶은 경우에 사용한다.

```
gSQL> \shutdown transactional
Shutdown success
gSQL>
```

Shutdown transactional은 다음과 같은 특징을 갖는다.

- 새로운 세션과 트랜잭션을 허용하지 않는다.
- 이미 진행된 트랜잭션에서는 새로운 statement들을 허용한다.
- 현재 진행 중인 트랜잭션들이 종료되기를 기다린다.
- 현재 진행 중인 트랜잭션이 종료된 이후에 세션이 자동 종료된다.
- 이후 instance를 구동할 때 instance 복구를 수행하지 않는다.

Shutdown Immediate

Shutdown Immediate는 발생한 트랜잭션들을 종료할 수 없는 상황에서 사용자가 instance를 종료하고자 할 때 사용한다.

```
gSQL> \shutdown immediate
Shutdown success
gSQL>
```

Shutdown immediate는 다음과 같은 특징을 갖는다.

- 새로운 세션과 트랜잭션을 허용하지 않는다.
- 현재 진행 중인 세션과 트랜잭션들은 강제로 종료시킨다.
- 시스템의 백그라운드 thread들이 수행 중이던 것을 완료할 때까지 기다린다.
- 이후 instance를 구동할 때 instance 복구를 수행하지 않는다.

Shutdown Abort

Shutdown abort는 instance가 비정상적인 상태라고 판단되었을 때 사용한다.

```
gSQL> \shutdown abort
Shutdown success
gSQL>
```

Shutdown abort는 다음과 같은 특징을 갖는다.

- 새로운 세션과 트랜잭션을 허용하지 않는다.
- 현재 진행 중인 세션과 트랜잭션들은 강제 종료된다.
- 시스템의 백그라운드 thread들을 즉시 종료시킨다.
- 이후 instance를 구동할 때 instance 복구를 수행한다.

5.3 프로세스 관리

본 장에서는 GOLDDILOCKS instance와 관련된 백그라운드 (background) 프로세스들에 대해 설명한다.

마스터 프로세스

마스터 프로세스는 데이터베이스 성능과 모니터링을 위한 비동기적 작업을 수행하며 다수의 thread들로 구성되어 있다.

마스터 프로세스의 실행파일 이름은 gmaster이다.

Checkpoint Thread

체크포인트 thread는 로그 플러싱 thread에서 발생하는 비동기적 체크포인트 이벤트를 수행한다. 체크포인트 이벤트는 redo log file이 스위칭될 때마다 발생한다.

체크포인트 이벤트는 사용자와 무관하게 비동기적으로 수행되며 관련 로그는 system.trc에 다음과 같이 기록된다.

```
[2014-09-11 14:04:34.704465 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-11 14:04:34.743933 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-11 14:04:34.759521 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] end
```

Log Flushing Thread

사용자 트랜잭션들이 기록한 redo log는 로그 버퍼에 기록되며 로그 플러싱 thread에 의해 주기적으로 로그 파일에 기록된다.

Redo log가 로그 파일 끝까지 기록된 경우 로그 스위칭이 발생하고 다음 redo log file에 기록된다. 로그 스위칭이 발생할 때 체크포인트 이벤트가 체크포인트 thread로 전달된다.

만약 로그 스위칭할 때 재사용 가능한 로그 파일이 없는 경우에는 읽기 전용 질의를 제외한 모든 질의가 재사용 가능한 로그 파일이 만들어질 때까지 대기한다.

다음은 로깅이 블로킹될 때 system.trc에 남은 메시지이다.


```
...
[2014-09-11 14:31:44.315871 THREAD(19102,139674683647744)] [INFORMATION]
[LOG FLUSHER] disable logging - blocked lfsn(1)
...
```

Log Archiving Thread

로그 아카이빙 thread는 비동기적으로 redo log file을 아카이빙한다. 해당 thread는 데이터베이스가 ARCHIVELOG 모드로 운영되는 경우에만 동작한다.

로그 아카이빙은 체크포인트 과정의 일부이며 체크포인트 thread가 발생한 로그 아카이빙 이벤트에 의해 수행된다.

다음은 redo_0_0.log가 archive_0.log로 아카이빙될 때 system.trc에 남은 메시지이다.

```
[2014-09-11 14:13:32.515996 THREAD(16913,140631135463168)] [INFORMATION]
[ARCHIVELOG BEGIN] LOG(/home/test/work/product/Gliese/home/wal/redo_0_0.log(0)) => ARCHIVE(/
home/test/work/product/Gliese/home/archive_log/archive_0.log)
[2014-09-11 14:13:33.145850 THREAD(16913,140631135463168)] [INFORMATION]
[ARCHIVELOG END] (/home/test/work/product/Gliese/home/archive_log/archive_0.log) : SUCCESS
...
```

Ager Thread

에이저 thread는 논리적으로 삭제된 데이터베이스 객체를 물리적으로 삭제한다.

GOLDILOCKS는 DROP TABLE 할 때 statement level consistency를 유지하기 위해 논리적 삭제만 수행한다. 즉, DROP TABLE이 수행되었다고 해도 DROP TABLE 이전에 수행되고 있던 statement는 삭제된 테이블에 대한 레코드를 탐색할 수 있다.

다음은 테이블과 테이블스페이스가 물리적으로 삭제될 때 system.trc에 남은 메시지이다.

```
[2014-09-11 14:13:37.966788 THREAD(16925,139892990408448)] [INFORMATION]
[AGER] aging table - object scn(4561), object view scn(4562), type(0), physical id
(25043954302976)
...
[2014-09-11 14:13:37.966917 THREAD(16925,139892990408448)] [INFORMATION]
[AGER] aging tablespace - object scn(4561), object view scn(4564), tablespace id(61)
```

Timer Thread

타이머 thread는 사용자 트랜잭션들의 시간 측정 비용을 줄이기 위해 비동기적으로 시스템에 시간을 설정하며 사용자 트랜잭션들은 시스템에 설정된 시간을 읽는다.

설정된 시간의 정밀도는 10 ms이다.

타이머 thread에서 설정된 시간을 사용하는 경우는 다음과 같고 10 ms의 오차를 갖는다.

- 타임아웃: QUERY_TIMEOUT, IDLE_TIMEOUT, DDL_LOCK_TIMEOUT
- 추적 로그에 남은 메시지 기록 시간
- 로그인 statement나 트랜잭션의 시작 시간
- 트랜잭션 완료 redo log에 기록되는 시간

Page Flusher & IO Slave Thread

체크포인트가 발생하면 변경된 데이터 페이지들을 디스크에 반영한다. 변경 사항은 테이블스페이스에 있는 다수의 데이터 파일에 저장될 수 있다. 이를 위해 page flusher thread는 테이블스페이스와 데이터 파일별로 IO 슬레이브 thread들에 작업을 분배하여 관리하고 IO 슬레이브 thread들은 변경된 페이지를 데이터 파일에 병렬로 기록한다.

변경된 페이지를 저장할 때는 한꺼번에 가능한 많이 저장하는 것이 성능 향상에 도움이 된다. 한 번에 기록되는 페이지 개수는 **MAXIMUM_FLUSH_PAGE_COUNT** 프로퍼티에 따른다.

변경된 페이지들을 데이터 파일에 기록한 후 system.trc에 다음과 같은 메시지를 남긴다.

```
[2014-09-11 14:13:38.329162 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
[2014-09-11 14:13:38.552161 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 1, datafile : 0 )
[2014-09-11 14:13:38.587510 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 2, datafile : 0 )
[2014-09-11 14:13:38.587831 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 62, datafile : 0 )
[2014-09-11 14:13:38.620239 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 63, datafile : 0 )
```

Cleanup Thread

Cleanup thread는 시스템의 자원을 비동기적으로 정리하며 다음과 같은 작업을 수행한다.

- 정상적으로 종료된 세션을 정리한다. GOLDBLOCKS는 사용자의 세션 종료를 논리적으로 처리하며 물리적 세션 종료는 cleanup thread가 수행한다.
- 비정상적으로 종료된 세션을 종료하며 만약 해당 세션이 트랜잭션을 사용하고 있다면 이를 rollback한다.
- Snapshot statement들의 타임아웃을 검사하고 만약 타임아웃을 초과한 세션이 있을 경우 이를 강제로 종료시킨다.

다음은 비정상적으로 종료된 세션을 정리할 때 system.trc에 남은 메시지이다.

```
[2014-09-12 10:34:38.387349 THREAD(23003,140722556352256)] [WARNING]
[CLEANUP] cleaning session - env(19), session(20), transaction(FFFFFFFFFFFFFFFF), program(gsql
), pid(23209), thread(140080441665280)
[2014-09-12 10:34:38.387515 THREAD(23003,140722556352256)] [WARNING]
[CLEANUP] cleaning up 1 sessions
```

다음은 snapshot statement가 타임아웃을 초과한 경우 system.trc에 남은 메시지이다.

```
[2014-09-12 10:49:21.842179 THREAD(3972,139706316711680)] [WARNING]
[CLEANUP] long statement timeout - pid(8029), thread(140053960505088), program(gsql),
statement start time(2014-09-12 10:48:49.963471)
```

비정상적으로 종료된 세션이 배타적 (exclusive) 래치를 획득한 상태에서 공유 메모리를 변경하는 도중에 'kill -9'와 같은 시그널에 의해 종료될 경우, 데이터베이스를 더 이상 운영할 수 없다. 이 경우, 다음과 같은 메시지가 system.trc에 남으며 'SHUTDOWN ABORT'로 instance를 종료시켜야 한다.

```
[2014-09-12 11:12:58.809249 THREAD(15313,140671386121984)] [WARNING]
[CLEANUP] failed to cleaning session - server restart required
..... dead session in critical section - env(3), session(4), transaction(47001E0004), pid
(15296), thread(140178075756288)
```

Process Monitor Thread

Process monitor thread는 process들을 실행하고 모니터링한다.

- "SHARED_SESSION" property가 YES로 설정되었을 때만 실행된다.
- Load-balancer (gbalancer), dispatcher (gdispatcher), shared-server (gserver)를 실행하고 비정상적으로 종료될 경우 재실행한다.
- Listener (glsnr) process는 모니터링 대상이 아니다.

Cluster Recover Thread

Cluster system 환경에서 노드를 시작할 때 노드가 MOUNT 단계로 phase up되면 cluster recover thread가 생성된다. In-doubt 트랜잭션이 있을 경우, cluster recover thread가 원격 노드의 cluster recover thread와 통신하여 in-doubt 트랜잭션을 복구한다.

In-doubt 트랜잭션이 있을 때 cluster recover thread는 원격 노드의 cluster recover thread와 통신하여 복구가 필요한 in-doubt 트랜잭션의 상태를 파악한다. 만약 원격 노드가 재시작되어 복구되기 전이라면 복구를 먼저 수행하도록 메시지를 전송하고 복구가 완료된 후에 in-doubt 트랜잭션의 상태를 구한다.

원격 노드를 통해서 파악할 수 있는 in-doubt 트랜잭션의 상태는 NONE, PREPARE, COMMIT, ROLLBACK 이다. 최소 한 개의 원격 노드에서 COMMIT이나 ROLLBACK 응답을 받을 경우 COMMIT이나 ROLLBACK을 수행한다. 모든 원격 노드에서 NONE 또는 PREPARE 응답을 받은 경우, 모든 cluster 노드에서 COMMIT이나 ROLLBACK을 수

행한 적이 없기 때문에 ROLLBACK을 수행한다.

Cluster recover thread가 in-doubt 트랜잭션을 복구했을 경우 system.trc에 다음과 같은 메시지를 남긴다.

```
[2018-11-22 16:52:00.466805 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] begin recovery
[2018-11-22 16:52:00.467221 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] commit in-doubt transaction - commit scn(999.0.439), global transaction id
(1.29294650), local transaction id(4)
[2018-11-22 16:52:00.468253 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] rollback in-doubt transaction - commit scn(1000.439), global transaction id
(4.34406459), local transaction id(59)
[2018-11-22 16:52:00.469198 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] commit in-doubt transaction - commit scn(1001.0.439), global transaction id
(5.35127356), local transaction id(60)
```

Failover Thread

Cluster system 환경에서 노드를 시작할 때 노드가 LOCAL OPEN 단계로 phase up되면 cluster failover thread가 생성된다. Cluster failover thread는 cluster system에서 특정 노드나 네트워크에 장애가 발생했을 때 장애 노드에 대해 offline이나 coordinator 재선정과 같은 failover 처리를 수행한다

Failover 상황이 발생하면 정상 노드들 중에 failover lock을 획득한 하나의 노드가 다른 노드들의 failover thread와 통신하여 failover를 수행한다.

Failover thread가 failover를 수행한 경우, system.trc에 다음과 같은 메시지를 남긴다.

```
[2018-11-22 15:27:34.619208 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] begin - failover member(5)
[2018-11-22 15:27:34.619418 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] acquire failover lock - driver(0), target(5), driver seq(1)
[2018-11-22 15:27:34.619692 INSTANCE(G1N1) THREAD(20183,140317097449216)] [INFORMATION]
[CDISPATCHER-S2] disconnect member - target member(5)
[2018-11-22 15:27:34.619893 INSTANCE(G1N1) THREAD(20183,140317097449216)] [INFORMATION]
[CDISPATCHER-S2] finalize sender socket - member(5)
[2018-11-22 15:27:34.621860 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] acquire failover lock
...
[2018-11-22 15:27:38.726436 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION][
FAILOVER] member(5) has failovered
[2018-11-22 15:27:38.728624 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] release failover lock - driver(-1), target(5), driver seq(1)
[2018-11-22 15:27:38.728786 INSTANCE(G1N1) THREAD(20140,140219957368576)] [WARNING]
```

```
reset remote session map - member(5)
[2018-11-22 15:27:38.729679 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] finished
```

리스너 프로세스

리스너 프로세스는 Client/ Server 환경에서 네트워크를 통한 원격 접속을 가능하게 한다. 리스너 프로세스는 **LISTEN_PORT**로 클라이언트 접속을 기다리는데 클라이언트가 접속할 때 dedicated 모드일 경우, 새로운 gserver를 구동하여 클라이언트와 연결하고 shared 모드일 경우, load-balancer (gbalancer)를 통해 부하가 적은 dispatcher (gdispatcher)를 선택하여 클라이언트와 연결한다.

gserver는 일종의 작업 서버로서 클라이언트에서 요청한 명령을 수행한다.

만약 LISTEN_PORT가 이미 사용 중일 경우, 다음과 같은 에러가 발생한다.

```
% glsnr --start
ERR-HY000(11077): given address is already in use
```

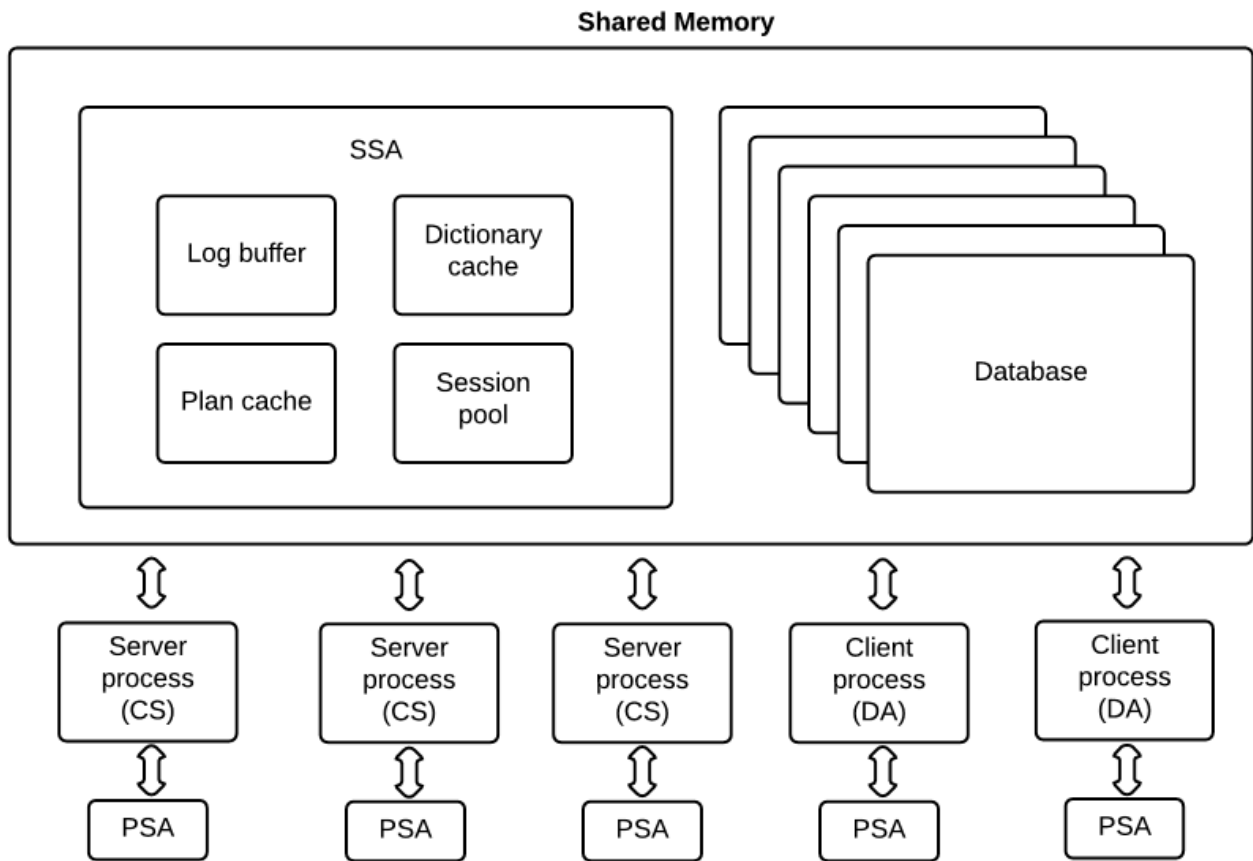
리스너 프로세스는 instance와 무관하게 동작한다. 즉, 리스너 프로세스는 instance의 구동 여부와 관계없이 언제든지 구동하고 종료할 수 있다.

5.4 메모리 관리

GOLDBLOCKS 메모리 구조

GOLDBLOCKS는 시스템의 모든 세션에서 공유하기 위한 메모리 (SSA)와 데이터베이스 페이지를 위한 공유 메모리, 각 세션마다 독립적으로 사용하는 heap 전용 메모리 (PSA)들을 사용한다.

그림 1 Shared Memory



SSA 관리

Shared Static Area (SSA)는 시스템의 모든 세션에서 공유해야 하는 정보들이 저장되어 있는 메모리 영역이다.

SSA에서 참조하는 모든 정보의 위치는 물리적 주소를 사용하기 때문에 새로운 프로세스가 SSA를 사용하려면 같은 물리적 주소를 사용해야 한다.

SSA의 물리적 시작 주소는 `SHARED_MEMORY_STATIC_KEY`와 `SHARED_MEMORY_ADDRESS`에 의해 결정된다. 만약 다른 프로그램에서 같은 `SHARED_MEMORY_STATIC_KEY`에 지정된 공유 메모리 key와 `SHARED_MEM`

ORY_ADDRESS에 지정된 메모리를 이미 사용하고 있다면 다음과 같은 에러가 발생한다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-HY000(11029): shared memory segment exists
gSQL>
```

SSA에는 log buffer, dictionary cache, plan cache, session pool, lock pool, transaction pool과 같은 주요 정보들이 저장된다.

SSA 크기는 **SHARED_MEMORY_STATIC_SIZE**에 의해 결정된다. Session/ lock/ transaction pool 및 dictionary cache에서 사용되는 메모리들은 시스템에서 자동으로 관리하며 사용자가 임의로 사용량을 제어할 수 없는 반면에 log buffer와 plan cache는 사용자가 임의로 제어할 수 있다.

Log buffer와 plan cache의 기본값을 늘리는 경우에는 증가된 양만큼 **SHARED_MEMORY_STATIC_SIZE**를 늘려야 한다. 그렇지 않으면 다음과 같은 에러가 발생할 수 있다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-HY000(13010): Insufficient static area
gSQL>
```

PSA 관리

Private Static Area (PSA)는 각 세션마다 독립적으로 사용하는 heap 메모리 영역이다. 최대 크기는 **PRIVATE_STATIC_AREA_SIZE**에 의해 결정된다.

세션이 만들어질 때는 PSA의 초기값만 할당하는데 세션에서 추가적인 메모리가 필요할 경우, 최대 크기만큼 할당할 수 있다. 다음은 최대 크기를 초과한 경우에 발생하는 에러이다.

```
ERR-HY000(13011): Unable to extend memory: [MAX: 104857600, TOTAL: 102764408, ALLOC: 2097240]
DESC: private static area
```

5.5 모니터링

데이터베이스 모니터링은 향후 이슈가 될 수 있는 문제점을 사전에 인지하고 대비할 뿐만 아니라 데이터베이스 운영을 위한 개선점을 찾기 위해 필요하다. GOLDDILOCKS 데이터베이스는 모니터링을 위해 텍스트 파일 형식의 trace 로그와 다양한 성능 view를 제공한다.

TRACE 파일을 이용한 모니터링

GOLDDILOCKS 데이터베이스는 instance의 시작부터 종료까지 시스템 전반적인 오류, 경고, 정보를 기록하는 시스템 로그와 XA 트랜잭션 로그, DDL 로그와 같은 trace 로그를 제공한다. 또한 SQL trace log를 제공하는데 자세한 내용은

은 **15.5** SQL Trace Log를 참조한다.

TRACE 파일 관리

GOLDDILOCKS 데이터베이스의 trace 로그 파일은 시스템 로그와 DDL 로그를 기록하는 'system.trc'와 XA 트랜잭션 로그를 기록하는 'xa.trc' 두 개의 파일로 구성된다. Trace 로그 파일은 SYSTEM_LOGGER_DIR 프로퍼티에 설정된 디렉토리에 생성되며 기본적으로 'GOLDDILOCKS_DATA' 환경 변수에 설정된 디렉토리 하위의 'trc' 디렉토리에 생성된다. Trace 로그 파일의 크기는 10 Mbyte인데 쓸 공간이 부족할 경우 고유한 파일 확장자가 추가된 파일로 보존하고 새로운 trace 로그 파일을 생성하여 기록한다.

Listener의 trace 로그 파일은 'GOLDDILOCKS_DATA' 환경 변수에 설정된 디렉토리 하위의 'trc' 디렉토리에 'listener.trc'라는 이름으로 생성된다. 로그 파일의 크기는 10 Mbyte인데 쓸 공간이 부족할 경우, 고유한 파일 확장자가 추가된 파일로 보존하고 새로운 trace 로그 파일을 생성하여 기록한다.

시스템 로그 외에 XA 트랜잭션 로그나 DDL 로그는 모니터링 ON/ OFF를 선택할 수 있다. TRACE_DDL 프로퍼티 값을 '0'으로 설정하여 DDL 로그를 OFF하거나 '1'로 설정하여 ON시킨다. XA 로그는 TRACE_XA 프로퍼티를 이용하여 동일한 방법으로 설정한다.

시스템 로그

마스터 프로세스가 시작된 후 종료될 때까지 데이터베이스 instance에 발생하는 오류, 경고, 정보가 시스템 로그에 기록된다.

시스템 로그 포맷

시스템 로그는 다음과 같은 포맷으로 기록된다.


```
[ '로그 기록 날짜 및 시간' THREAD('프로세스 Id', 'thread handle') ] [ '로그 레벨' ]
[ '로그 프리픽스' ] '로그 바디'
```

- '로그 기록 날짜 및 시간'은 로그를 남긴 날짜 및 시간이다.
- THREAD('프로세스 Id', 'thread handle')는 로그를 남긴 프로세스의 ID와 thread handle 정보이다.
- '로그 프리픽스'는 로그를 만든 주체나 기능을 의미하고 '로그 바디'는 세부 내용이다.
- '로그 레벨'은 시스템 로그에 기록되는 로그의 레벨로써 FATAL, ABORT, WARNING, INFO가 있는데 다음과 같은 속성을 가진다.

표 5-4 Log level properties

로그 레벨	설명	처리
FATAL	마스터 프로세스 또는 클라이언트가 비정상적으로 종료된 상태이다.	클라이언트 프로세스 FATAL인 경우 클라이언트 재접속해야 하고 시스템 FATAL인 경우 데이터베이스 instance를 종료한 후에 다시 시작해야 한다. 데이터 파일, 제어 파일, redo log file, 시스템 로그 파일을 백업하여 데이터베이스 제조사에 문의해야 한다.
ABORT	Rollback을 수행한 후 서비스를 계속할 수 있는 상태이다.	시스템 운영을 위한 정상적인 과정이며 rollback 원인을 제거한 후에 다시 수행해야 한다.
WARNING	운영상의 경고이다.	데이터베이스 instance에 비정상적인 상황이 발생하였다. 운영상 문제는 없으나 원인을 분석할 필요가 있다.
INFO	운영상의 정보이다.	-

예를 들어 다음과 같은 시스템 로그는 2014년 9월 11일 17시 30분 55초 경에 남겨진 운영상의 정보로써 프로세스 ID가 21395인 프로세스 (thread handle이 139982731163392)에 의한 시스템 로그이다. 로그 prefix는 'STARTUP-SM'인데 GOLDILOCKS 데이터베이스의 마스터 프로세스가 시작될 때 저장 관리자(storage manager)를 기동하는 중이며, 다단계 시작 중에 NOMOUNT 단계로 전이하였다는 것을 의미한다.

```
[ 2014-09-11 17:30:55.758164 THREAD(21395,139982731163392) ] [ INFORMATION ]
[ STARTUP-SM ] NO-MOUNT PHASE
```

GOLDILOCKS 데이터베이스 운영 정보

데이터베이스 instance의 생성, 다단계 시작 및 종료, 데이터 파일 로드, 재시작 복구, 미디어 복구 수행 등에 대한 로그로써 마스터 프로세스가 시작된 후 종료될 때까지 운영을 위해 필요한 정보를 남긴다.

- GOLDILOCKS instance 생성 로그

데이터베이스 instance를 생성할 때 다음과 같은 시스템 로그가 기록된다. 데이터베이스를 생성하기 위해 NOMOUNT 단계로 전이한 후 제어 파일을 생성한다.

```
=====
Startup GOLDILOCKS
TIME      : 2014-09-03 14:43:17.321020
```

```
=====
```

```
[2014-09-03 14:43:17.321134 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] NO-MOUNT PHASE
[2014-09-03 14:43:17.321658 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] DATA_STORE_MODE(2)
[2014-09-03 14:43:17.335809 THREAD(14979,140542517491456)] [INFORMATION]
.... copy control file from '/goldilocks_data/wal/control_0.ctl' to '/goldilocks_data/wal/
control_1.ctl'
```

그리고 OPEN 단계로 전이하여 시스템 테이블스페이스를 생성한다.

```
[2014-09-03 14:43:17.401356 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] MOUNT PHASE
[2014-09-03 14:43:19.319769 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] PRE-OPEN PHASE
[2014-09-03 14:43:19.320494 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] RECOVER TABLESPACE AND DATAFILE STATE
[2014-09-03 14:43:19.326269 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] OPEN PHASE
[2014-09-03 14:43:21.005536 THREAD(14979,140542517491456)] [INFORMATION]
[TABLESPACE] Create Tablespace(0)
[2014-09-03 14:43:21.005593 THREAD(14979,140542517491456)] [INFORMATION]
[TABLESPACE] Create Tablespace(1)
...
```

시스템 테이블스페이스를 생성한 후에 체크포인트를 수행하고 데이터베이스 instance를 종료한다.

```
[2014-09-03 14:43:21.788129 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] begin - checkpoint lid(0,10128,13), checkpoint lsn(10512), oldest lsn(10512)
[2014-09-03 14:43:21.788188 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] body - checkpoint lid(-1,0,0), checkpoint lsn(-1), active transaction count(0)
[2014-09-03 14:43:21.788203 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] end - checkpoint lid(0,10128,77), checkpoint lsn(10513)
[2014-09-03 14:43:21.788214 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] flush redo log
[2014-09-03 14:43:21.949589 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:43:21.957563 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] CLOSE
[2014-09-03 14:43:21.957595 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] POST CLOSE
[2014-09-03 14:43:21.992521 THREAD(14979,140542517491456)] [INFORMATION]
```

```
[SHUTDOWN-SM] DISMOUNT
[2014-09-03 14:43:21.992557 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

- GOLDILOCKS instance 시작 로그

마스터 프로세스가 데이터베이스 instance의 다단계 시작, 데이터 파일 로드, 재시작 복구, 미디어 복구 수행 등과 같은 로그를 기록한다. 먼저 MOUNT 단계로 전이한 후에 데이터 파일을 로드한다.

```
=====
Startup GOLDILOCKS
TIME      : 2014-09-03 14:43:22.162601
=====
[2014-09-03 14:43:22.162765 THREAD(14982,140025756808960)] [INFORMATION]
[STARTUP-SM] NO-MOUNT PHASE
[2014-09-03 14:43:22.163389 THREAD(14982,140025756808960)] [INFORMATION]
[STARTUP-SM] DATA_STORE_MODE(2)
[2014-09-03 14:43:22.429311 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] MOUNT PHASE
[2014-09-03 14:43:22.559395 THREAD(14983,140025756808960)] [INFORMATION]
[EVENT] system startup : SUCCESS
[2014-09-03 14:43:22.568526 THREAD(14981,139649517561600)] [INFORMATION]
[STARTUP] MOUNT PHASE
[2014-09-03 14:43:22.571200 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] LOAD DATAFILES
[2014-09-03 14:43:22.571241 THREAD(14983,140025756808960)] [INFORMATION]
.... datafile '/goldilocks_data/db/system_dict.dbf' assigned to PARALLEL_IO_GROUP_1
...
[2014-09-03 14:43:22.571562 THREAD(14983,140025280841472)] [INFORMATION]
.... LOAD DATAFILE(/goldilocks_data/db/system_dict.dbf)
....
```

데이터 파일을 메모리로 로드한 후에 복구를 수행한다.

```
[2014-09-03 14:43:23.537256 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REFINE TABLESPACE AND DATAFILE
[2014-09-03 14:43:23.631974 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] begin
[2014-09-03 14:43:23.634374 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] read checkpoint log - checkpoint log id(0,10128,13), oldest lsn(10512), system
scn(7)
[2014-09-03 14:43:23.756293 THREAD(14983,140025756808960)] [INFORMATION]
```

```
[RESTART REDO] ready to redo - start lid(0,10128,13), lsn(10512)
...
[2014-09-03 14:43:24.090755 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] end - restart lsn(10514), restart scn(7)
[2014-09-03 14:43:24.091551 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART UNDO] begin
[2014-09-03 14:43:24.091598 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART UNDO] end
```

복구가 완료되면 체크포인트를 수행하여 복구된 결과를 디스크 데이터 파일에 반영하고 인덱스를 생성한 후 OPEN 단계로 전이한다.

```
[2014-09-03 14:43:24.111878 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-03 14:43:24.129995 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:43:24.135864 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] end
[2014-09-03 14:43:24.144525 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] PRE-OPEN PHASE
[2014-09-03 14:43:24.202782 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] RECOVER TABLESPACE AND DATAFILE STATE
[2014-09-03 14:43:24.210158 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REFINER RELATIONS
[2014-09-03 14:43:24.210304 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REBUILD INDEXES
[2014-09-03 14:43:24.210375 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] OPEN PHASE
[2014-09-03 14:43:24.332064 THREAD(14983,140025756808960)] [INFORMATION]
[EVENT] system startup : SUCCESS
[2014-09-03 14:43:24.340843 THREAD(14981,139649517561600)] [INFORMATION]
[STARTUP] OPEN PHASE
```

- GOLDDILOCKS instance 종료 로그

데이터베이스 instance를 종료하려면 먼저 모든 데이터 파일들을 디스크에 반영한 후에 체크포인트를 수행하고 마스터 프로세스를 종료해야 한다.

```
[2014-09-03 14:48:03.467293 THREAD(15416,139855812097792)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
...
```

```
[2014-09-03 14:48:03.748958 THREAD(15416,139855908558592)] [INFORMATION]
[PAGE FLUSHER] flushed lsn(137496), flushed page count(9216)]
[2014-09-03 14:48:03.761055 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-03 14:48:03.780011 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:48:03.786387 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] end
[2014-09-03 14:48:03.791251 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] CLOSE
[2014-09-03 14:48:03.791383 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] POST CLOSE
[2014-09-03 14:48:03.824445 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] DISMOUNT
[2014-09-03 14:48:03.824518 THREAD(15416,139856430274304)] [INFORMATION]
[EVENT] system shutdown : SUCCESS
[2014-09-03 14:48:04.267130 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

\shutdown abort 를 이용하여 강제로 서버를 중지시킨 경우에는 체크포인트를 수행하지 않고 정상적인 서버 중지 절차를 수행하지 않는다.

```
[2014-09-03 14:51:45.353154 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN] skip CLOSE phase
[2014-09-03 14:51:45.678461 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN] skip DISMOUNT phase
[2014-09-03 14:51:45.678696 THREAD(8989,139949509089024)] [INFORMATION]
[EVENT] system shutdown : SUCCESS
[2014-09-03 14:51:45.678928 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

- 데이터베이스 운용 중 마스터 프로세스의 체크포인트, 로그 플러셔, 로그 아카이빙, ager, 병렬 IO, cleanup thread들의 로그

체크포인트는 체크포인트 시점에 디스크에 반영되지 않고 메모리에서만 변경된 데이터 파일을 모두 디스크에 반영한다. 만약 병렬 IO를 이용하는 경우에는 데이터 파일 단위로 병렬 IO가 수행된다. 체크포인트 로그는 '[CHECKPOINT] begin'에서 '[CHECKPOINT] end'까지가 한 세트이다.

[IO SLAVE]는 병렬 IO를 전담하는 IO thread에 의해 기록되는 로그이다. '[IO SLAVE] flush datafile (tablespace : 0, datafile : 0)' 로그는 ID가 '0'인 테이블스페이스의 데이터 파일과 ID가 '0'인 데이터 파일을 디스크에 반영한 후에 기록된다. 이러한 데이터 파일 플러시 로그는 체크포인트 할 때 데이터 파일 개수만큼 반복적으로 기록된다.

'[PAGE FLUSHER] flushed lsn(139039), flushed page count(9216)]'는 디스크에 반영된 최소 LSN이 139039 이고 9216 페이지가 디스크에 반영되었다는 의미이다. 또한 마지막 로그 LSN이 139039보다 작은 redo log file들을 아카이빙하고 체크포인트 로그와 제어 파일을 기록하여 디스크에 반영한다.

데이터베이스가 대용량이면 체크포인트 시간이 길어질 수 있다. [IO SLAVE] 로그를 추적하여 계속해서 데이터 파일을 기록하는 중인지 체크하고 만약 디스크 IO가 발생하지 않고 멈춘 경우에는 로그 아카이빙이 진행 중인지 검사하여 디스크 공간이 부족할 경우, 디스크 공간을 확보하여 로그 아카이빙이 정상적으로 진행될 수 있도록 한다.

만약 체크포인트가 실패하면 '[CHECKPOINT] CHECKPOINT was failed' 로그가 기록된다. 특히 로그 파일 스위치로 인한 체크포인트 시에는 체크포인트가 생략될 수 있고 이 때 '[CHECKPOINT] CHECKPOINT was skipped' 로그가 기록된다.

```
[2014-09-12 15:54:59.654427 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] begin
[2014-09-12 15:54:59.654798 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
[2014-09-12 15:54:59.835173 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 1, datafile : 0 )
[2014-09-12 15:54:59.893991 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 2, datafile : 0 )
[2014-09-12 15:54:59.926753 THREAD(13780,140493050603264)] [INFORMATION]
[PAGE FLUSHER] flushed lsn(138895), flushed page count(9216)]
[2014-09-12 15:54:59.926989 THREAD(13780,140492777965312)] [INFORMATION]
[ARCHIVING] stable lsn(139039)
[2014-09-12 15:54:59.933780 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] begin - checkpoint lid(0,55527,13), checkpoint lsn(139040), oldest lsn(139040)
[2014-09-12 15:54:59.933825 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] body - checkpoint lid(0,55527,77), checkpoint lsn(139041), active transaction
count(1)
[2014-09-12 15:54:59.933844 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] end - checkpoint lid(0,55527,155), checkpoint lsn(139042)
[2014-09-12 15:54:59.933859 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] flush redo log
[2014-09-12 15:54:59.936154 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-12 15:54:59.942850 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] end
```

로그 플러셔는 로그 버퍼를 디스크에 플러쉬하는 작업이 중지되었을 때와 중지된 플러쉬를 재개할 때 시스템 로그를 기록한다. 다음 로그 그룹이 재사용 가능한 로그 파일이 아닌 경우 재사용 가능할 때까지 로깅이 중지된다. 다음은 redo log file 시퀀스 번호가 34인 redo log file이 아직 아카이빙되지 않아서 로깅이 중지된 경우이다.

```
[2014-09-12 16:01:57.514303 THREAD(13780,14057333325568)] [INFORMATION]
[LOG FLUSHER] disable logging - blocked lfsn(34)
```

로그가 중지되면 트랜잭션도 중지되므로 즉각적인 조치가 필요하고, 체크포인트가 수행되어 아카이빙이 이루어지면 로그가 재개된다.

```
[2014-09-12 16:01:58.079236 THREAD(13780,140380267869952)] [INFORMATION]
[ARCHIVING] enable logging - blocked lfsn(34), inactivated lfsn(34)
```

로그 아카이빙 thread는 ACTIVE 상태의 redo log file을 아카이빙하고 시스템 로그를 기록한다. '[ARCHIVING] stable lsn(...)' 부터 '[ARCHIVING] inactivate group ...'까지가 한 세트이다. 그리고 데이터베이스가 archive log mode로 운영 중인 경우, redo log file을 아카이빙 하는 로그 '[ARCHIVELOG BEGIN] ...' 부터 '[ARCHIVELOG END] ...'가 기록된다.

```
[2014-09-02 17:41:56.762950 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVING] stable lsn(144143)
[2014-09-02 17:41:56.763549 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVELOG BEGIN] LOG(/goldilocks_data/wal/redo_0_0.log(8)) => ARCHIVE(/goldilocks_data/
archive_log/archive_8.log)
[2014-09-02 17:41:57.385936 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVELOG END] (/goldilocks_data/archive_log/archive_8.log) : SUCCESS
[2014-09-02 17:41:57.385987 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVING] inactivate group #0(8)
```

'[ARCHIVELOG BEGIN] ...' 로그 후에 'Archiving was failed - ...' 로그가 출력된 경우, 로그 아카이빙이 실패한 것이므로 즉시 해결해야 ACTIVE 상태의 redo log file을 재사용하여 서비스할 수 있다.

Age가 제거된 테이블과 테이블스페이스에 대해 에이징한 정보는 다음과 같이 기록된다. 테이블이 제거될 때 테이블의 lock도 함께 삭제되며 테이블의 scn과 에이징 시점의 에이징 가능한 scn 및 테이블 lock의 에이징 정보도 기록된다. 만약 테이블에 인덱스가 생성되어 있을 경우, 테이블을 제거할 때 인덱스도 함께 삭제된다.

```
[2014-09-03 12:13:56.539971 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging index - object scn(224), type(0), physical id(22634477649920)
[2014-09-03 12:13:56.540388 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging table - object scn(224), object view scn(225), type(0), physical id
(22630182682624)
[2014-09-03 12:13:56.540491 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging lock item - object scn(226), agable stmt scn(228), physical id(22630182682624)
```

테이블스페이스를 제거할 때 테이블스페이스의 scn과 에이징 시점의 에이징 가능한 scn 그리고 제거된 테이블스페이스의 ID를 기록한다.

```
[2014-09-03 12:13:56.540553 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging tablespace - object scn(224), object view scn(227), tablespace id(5)
```

비정상적으로 종료된 세션에 대해 cleanup thread가 정리한 정보는 다음과 같이 기록된다. 사용자 세션이 비정상 종료되더라도 세션의 자원을 정리하므로 데이터베이스 instance나 다른 사용자들은 계속해서 운영할 수 있다.

```
[2014-09-03 13:43:02.220139 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] snipe at zombie session - pid(7766), thread(139967223228160), program(gsql)
[2014-09-03 13:43:02.220211 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] cleaning session - env(3), session(4), transaction(FFFFFFFFFFFFFFFF), program(gsql),
pid(7766), thread(139967223228160)
[2014-09-03 13:43:02.220270 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] cleaning up 1 sessions
```

- 사용자에게 의한 테이블스페이스 생성, 제거, 변경 로그

사용자 테이블스페이스를 생성, 제거 또는 변경할 경우 시스템 로그에 기록된다. 테이블스페이스 관련 DDL은 TRACE_DDL ON/OFF에 관계없이 기본적으로 기록이 남는다. DDL 실패는 시스템 로그에 기록되지 않으므로 DDL 로그에 대한 자세한 정보와 실패 시 실패 원인을 알기 위해서는 TRACE_DDL을 ON하여 시스템을 운영하도록 한다.

```
[2014-09-15 10:26:41.649909 THREAD(24881,140468897289984)] [INFORMATION]
[TABLESPACE] Create Tablespace(7)
[2014-09-15 10:26:55.966385 THREAD(24881,140468897289984)] [INFORMATION]
[DATAFILE] add datafile(/home/zkyungoh/work/product/Gliese/home/db/TEST1.dbf)
[2014-09-15 10:27:11.325897 THREAD(24881,140468897289984)] [INFORMATION]
[DATAFILE] Drop Datafile(/home/zkyungoh/work/product/Gliese/home/db/TEST1.dbf)
...
[2014-09-15 10:32:00.669550 THREAD(24881,140468897289984)] [INFORMATION]
[TABLESPACE] drop tablespace ( 7 )
```

- 시스템 인터널 오류 및 인덱스 생성 실패 로그

GOLDBLOCKS 데이터베이스 시스템의 오류는 맞지만 정확한 원인을 정의할 수 없는 경우에는 인터널 에러가 발생한다. 인터널 에러가 발생한 경우, 에러를 발생시킨 SQL 문은 rollback되고 시스템과 다른 세션에 영향을 주지 않으므로 계속 서비스할 수 있다.

또한 에러를 발생시킨 SQL 문을 다시 실행하면 동일하게 실패할 수도 있고 실패했던 원인이 사라져 성공할 수도 있으므로 원인을 찾기 위해서는 실패 시점의 데이터베이스를 변경하지 말고 그대로 유지한 채 원인 분석을 요청해야 한다.

UNIQUE 인덱스를 생성할 때 테이블에 동일한 키가 존재하는 경우에는 인덱스 생성에 실패한다. 인덱스 생성에 실패하더라도 테이블과 테이블에 이미 생성된 인덱스에는 영향이 없으므로 서비스에도 영향을 미치지 않는다.


```
[2014-09-15 11:26:59.640345 THREAD(7819,140737354012416)] [INFORMATION]
Index creation failed ( physical id : 22638772617216, error code : 14016 )
```

XA 로그

분산 트랜잭션을 처리하기 위한 XA 트랜잭션 인터페이스 start, close, end, rollback, prepare, commit, recover, forget 등에 대한 성공, 실패 로그를 기록한다. GOLDILOCKS 데이터베이스는 XA 추적 로그를 기본적으로 기록하지 않도록 한다. XA 추적 로그를 기록하기 위해서는 다음과 같이 TRACE_XA를 ON시켜야 한다. XA 트랜잭션 인터페이스의 자세한 스펙은 **XA API References**를 참조한다.

```
gSQL> alter system set trace_xa = yes;
System altered.
```

'xa.trc'에 기록되는 XA 추적 로그는 다음과 같다. 먼저 수행된 XA 인터페이스가 기록되고 성공 (complete) 또는 실패 (failed) 여부가 기록된다. 수행된 세션의 세션 ID와 XA 트랜잭션 ID 등의 정보가 함께 기록되고 실패할 경우 **XA API References**에 정의된 에러 코드가 기록된다.

```
[2014-09-15 11:45:19.599018 THREAD(7966,139931504572160)] [INFORMATION]
xa_start() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:19.599360 THREAD(7966,139931504572160)] [INFORMATION]
xa_end() complete - session(4), xid(0.3231.00), flags(4000000)
[2014-09-15 11:45:19.599418 THREAD(7966,139931504572160)] [INFORMATION]
xa_prepare() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:22.864563 THREAD(7966,139931504572160)] [INFORMATION]
xa_recover() complete - session(4), xid(), flags(1000000)
[2014-09-15 11:45:22.864829 THREAD(7966,139931504572160)] [INFORMATION]
xa_commit() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:22.864887 THREAD(7966,139931504572160)] [INFORMATION]
xa_rollback() complete - session(4), xid(0.3232.00), flags(0)
[2014-09-15 11:45:22.885951 THREAD(7966,139931504572160)] [INFORMATION]
xa_forget() complete - session(4), xid(0.3230.00), flags(0)
[2014-09-15 11:45:22.886017 THREAD(7966,139931504572160)] [INFORMATION]
xa_forget() failed - session(4), xid(0.3231.00), flags(0), xa_error(-4)
```

DDL 로그

GOLDILOCKS 데이터베이스에서 발생한 모든 DDL (생성, 제거, 변경)에 대해 DDL을 발생시킨 세션과 전체 SQL 구문 및 성공 (success), 실패 (failure) 여부를 시스템 로그에 추가한다. GOLDILOCKS 데이터베이스는 DDL 로그를 기본적으로 기록하지 않도록 한다. DDL 추적 로그를 기록하기 위해서는 다음과 같이 TRACE_DDL을 ON시켜야 한다.

```
gSQL> alter system set trace_ddl = yes;
System altered.
```

예를 들어 다음과 같은 DDL 문으로 테이블스페이스를 생성하였을 때는 다음과 같은 DDL 로그가 기록된다.

```
gSQL> CREATE TABLESPACE TEST_TBS1
DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M,
          'TEST_TBS1_02.dbf' SIZE 10M,
          'TEST_TBS1_03.dbf' SIZE 10M;
Tablespace created.
```

```
[2014-09-15 12:26:29.209210 THREAD(8149,140267442067200)] [INFORMATION]
[SESSION:11][DDL success] CREATE TABLESPACE TEST_TBS1
DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M,
          'TEST_TBS1_02.dbf' SIZE 10M,
          'TEST_TBS1_03.dbf' SIZE 10M
[2014-09-15 12:26:29.209277 THREAD(8149,140267442067200)] [INFORMATION]
[SESSION:11][COMMIT with DDL]
```

DDL 문이 실패할 경우, 다음과 같이 'DDL failure'로 기록된다.

```
gSQL> ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_04.dbf' SIZE 10M;
ERR-42000(16130): file is already exist - '/home/zkyungoh/work/product/Gliese/home/db/TEST_TBS
1_04.dbf' :
ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_04.dbf' SIZE 10M
*
ERROR at line 1:
```

```
[2014-09-15 12:45:08.598789 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL failure] ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M
```

테이블, 인덱스에 대한 DDL 문도 동일한 방식으로 DDL 로그를 기록한다. 다음은 테이블과 인덱스를 생성 한 후에 COMMIT 한 DDL 로그이다.

```
gSQL> CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1;
Table created.
gSQL> CREATE INDEX T1X ON T1 ( I1 );
Index created.
gSQL> COMMIT;
Commit complete.
```

```
[2014-09-15 12:40:37.887952 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1
[2014-09-15 12:40:47.451806 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE INDEX T1X ON T1 ( I1 )
[2014-09-15 12:40:51.017975 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][COMMIT with DDL]
```

다음은 테이블과 인덱스를 생성한 후에 ROLLBACK 한 DDL 로그이다.

```
[2014-09-15 12:42:27.367722 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1
[2014-09-15 12:42:31.317436 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE INDEX T1X ON T1 ( I1 )
[2014-09-15 12:42:34.601738 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][ROLLBACK with DDL]
```

이중화 Trace 로그

GOLDILOCKS의 이중화 틀인 CYCLONE과 LOGMIRROR를 이용할 때 별도의 파일로 이중화 관련 추적 로그를 기록한다. 이중화 추적 로그는 CYCLONE의 **운영하기**와 LOGMIRROR의 **운영**을 참조한다.

Listener 로그

Listener 프로세스가 시작된 후 종료될 때까지 발생하는 오류와 정보를 listener 로그에 기록한다.

Listener 로그 포맷

Listener 로그는 다음과 같은 포맷으로 기록된다.

```
['로그 기록 날짜 및 시간' THREAD('프로세스 Id', 'thread handle')]
['로그 프리픽스'] '로그 바디'
```

- '로그 기록 날짜 및 시간'은 로그를 남긴 날짜와 시간이다.
- THREAD('프로세스 Id', 'thread handle')는 로그를 남긴 프로세스의 ID와 thread handle 정보이다.
- '로그 프리픽스'는 로그를 만든 주체나 기능을 의미하고 '로그 바디'는 세부 내용이다.

View를 이용한 성능 모니터링

여러 사용자가 동시에 데이터베이스에 접속하여 갱신할 수 있기 때문에 다중 사용자에게 대한 동시성 제어가 필요하다. SQL 문에 의한 명시적인 데이터의 동시성 뿐만 아니라 시스템을 위한 데이터, 공유된 자원들에 대해서도 동시성을 제

공해야 하는데 GOLDDILOCKS는 래치 (latch)를 이용하여 동시성을 제어한다.

Lock 기반의 동시성 제어기법은 서로 다른 트랜잭션에 의해 동일한 데이터들이 갱신될 때 데드락(deadlock)을 유발할 수 있고 GOLDDILOCKS의 래치 기반 동시성 제어도 데드락을 발생시킬 수 있다. 데드락이 발생할 경우 성능에 영향을 미치기 때문에 데드락이 발생한 래치를 처리할 수 있는 view를 제공한다.

V\$LOCK_WAIT를 이용하여 데드락을 유발한 트랜잭션들을 찾을 수 있는데 관리자는 이를 모니터링하여 데드락을 해제해야 한다. V\$LOCK_WAIT에 대한 자세한 내용은 **V\$LOCK_WAIT**를 참조한다. 또한 데드락의 원인인 래치 아 이템을 모니터링 하기 위해서는 **V\$LATCH**를 참조한다.

6.

GOLDILOCKS 데이터베이스의 구조 및 저장 구조

6.1 제어 파일 관리

GOLDDILOCKS 데이터베이스를 이용하기 위해서는 데이터베이스 instance를 생성해야 하며 데이터베이스 instance가 생성될 때 제어 파일이 생성된다. GOLDDILOCKS 다단계 시작의 nomount 단계에서 mount 단계로 올라갈 때 제어 파일에 기록된 정보를 이용하여 데이터베이스에서 사용할 파일들의 절대 경로와 파일 크기 등과 같은 정보를 파악한다. 제어 파일은 바이너리 파일이며 데이터베이스의 다음과 같은 정보들이 저장된다.

제어 파일 내용

표 6-1 GOLDDILOCKS 데이터베이스 시스템 정보

항목	설명
Data Store Mode	데이터베이스가 기동될 때 설정된 저장 모드이다. (TDS, CDS)
Server State	데이터베이스 instance의 상태이다. (NONE, RECOVERED, RECOVERING, SERVICE, SHUT DOWN)
Last Checkpoint Lsn	데이터베이스에서 마지막으로 수행된 체크포인트의 LSN이다.

표 6-2 로그 정보

항목	설명
Checkpoint Lid, Lsn	데이터베이스에서 마지막으로 수행된 체크포인트의 로그 정보 (lsn, 로그의 위치) 이다.
Last Inactivated Log File Sequence	마지막으로 inactive로 바뀐 로그 파일 sequence이다.
Archivelog Mode	운영 중인 데이터베이스의 archivelog mode이다.
Creation Time	데이터베이스 생성 시간이다.

데이터베이스 정보에는 데이터베이스의 운영 정보와 사용 중인 모든 테이블스페이스 정보, 데이터정보를 저장한다. 제어 파일에 저장된 데이터베이스 운영 정보는 다음과 같다.

표 6-3 데이터베이스 정보

항목	설명
Transaction Table Size	데이터베이스에서 사용 중인 최대 transaction table의 개수이다.
Undo Relation Count	데이터베이스에서 사용 중인 undo relation의 개수이다.
Tablespace Count	데이터베이스에 생성되어 사용 중인 테이블스페이스의 개수이다.
New Tablespace Id	다음에 생성될 테이블스페이스의 ID이다.

제어 파일에 저장된 테이블스페이스의 정보는 다음과 같다.

표 6-4 테이블스페이스 정보

항목	설명
Tablespace Id	테이블스페이스의 고유한 ID이다.
Attributes	테이블스페이스의 속성으로써 저장 장치 (메모리, 디스크), 영속성 보장여부 (temporary, persistent)와 테이블스페이스의 용도 (dictionary, undo, data, temporary) 등을 포함한다.
Page Count In Extent	Extent의 page 개수이다.
State	테이블스페이스의 상태 (CREATING, CREATED, DROPPING, DROPPED, AGING) 이다.
Relation Id	테이블스페이스의 pending 연산을 저장하기 위한 relation ID 이다.
New Data File Id	테이블스페이스에 새로운 데이터가 추가될 때 설정하기 위한 데이터 ID 이다.
Is Logging	테이블스페이스의 logging 모드이다. (LOGGING/ NOLOGGING)
Is Online	테이블스페이스의 online 여부이다. (ONLINE/ OFFLINE)
Data File Count	Tablespace가 사용 중인 데이터의 개수이다.
Offline Lsn	Offline 테이블스페이스를 online으로 변경하기 위해 복구가 필요한 경우 복구를 수행해야 하는 마지막 LSN이다.
Offline State	Offline 테이블스페이스의 상태이다. (CONSISTENT/ INCONSISTENT) CONSISTENT offline 테이블스페이스는 메모리에 존재하는 최신 데이터를 디스크의 데이터에 반영한 후 테이블스페이스의 상태를 offline으로 바꾸기 때문에 online으로 변경할 때 복구할 필요가 없다. 반면 INCONSISTENT offline 테이블스페이스를 online으로 변경할 때는 로그를 이용하여 복구를 수행한 후 online으로 변경해야 한다.

제어 파일에 저장된 데이터 파일의 정보는 다음과 같다.

표 6-5 데이터 파일 정보

항목	설명
Name	데이터가 저장된 절대 경로를 포함하는 데이터 이름이다.
State	데이터의 상태 (CREATING, CREATED, DROPPING, DROPPED, AGING) 이다.
Data File Id	테이블스페이스에서 고유한 데이터의 ID이다.
Auto Extend	데이터 파일이 가득 찼을 때 자동으로 확장할지 여부이다.
Size	데이터의 크기이다.
Next Size	데이터가 가득 찼을 때 확장할 크기이다.
Max Size	확장 가능한 데이터의 최대 크기이다.
Timestamp	데이터가 생성된 시점이다.
Checkpoint Lsn, Lid	데이터 파일에 마지막으로 체크포인트가 수행되었을 때의 체크포인트 로그 정보 (LSN, 로그 위치) 이다.
Creation Lsn, Lid	데이터 파일이 생성된 시점의 체크포인트 로그 정보 (lsn, 로그 위치) 이다.

데이터베이스에서 수행된 각 incremental backup의 정보를 저장한다. GOLDILOCKS 데이터베이스는 데이터베이스와 테이블스페이스에 대한 incremental backup을 지원하고 다음과 같은 incremental backup 정보를 제어 파일에 저장한다.

표 6-6 Incremental backup 정보

항목	설명
Backup Lsn, Lid	Backup 시작 시점에 마지막으로 수행된 체크포인트 로그 정보 (LSN, 로그 위치) 이다.
Begin Time	Incremental backup이 시작된 시점이다.
Completion Time	Incremental backup이 완료된 시점이다.
Tablespace Id	Incremental backup이 수행된 테이블스페이스의 고유 ID이다. 테이블스페이스에 대한 incremental backup의 경우 해당 테이블스페이스 ID가 기록되고 테이블스페이스에 대한 backup이 아닌 경우 최대 테이블스페이스 ID (65535)가 기록된다.
Level	수행된 incremental backup의 level 이다.
Object Type	Incremental backup의 대상이다. (데이터베이스, 제어 파일, 테이블스페이스)
Backup File Name	Incremental backup 파일의 이름이다.
Backup Option	Incremental backup의 option 이다. (cumulative/ differential)

제어 파일 다중화

제어 파일은 GOLDDILOCKS 데이터베이스의 물리적 구조와 데이터의 일관성에 대한 중요한 정보를 저장하고 있으므로 훼손되거나 실수로 삭제되는 등의 문제가 발생하는 경우 데이터베이스를 운영할 수 없다.

따라서 GOLDDILOCKS 데이터베이스는 최소 두 개 이상의 제어 파일을 생성하여 물리적으로 분리된 디스크에 유지할 것을 권장하고 최대 여덟 개까지 다중화를 지원한다. 제어 파일을 추가하려면 데이터베이스를 생성할 때 프로퍼티 파일에 다중화할 제어 파일 개수를 설정하고 각 제어 파일의 경로를 설정한다. 또는 운영 중에 제어 파일 다중화 개수에 대한 프로퍼티 값을 증가시키고 추가될 제어 파일의 경로를 설정하여 추가할 수도 있다.

제어 파일 훼손 시 대처 방안

데이터베이스 비정상 종료 등으로 인해 제어 파일이 훼손되었을 경우 다중화된 제어 파일 중 정상 제어 파일을 이용하여 훼손된 제어 파일을 복원한 후에 다시 시작한다.

만약 다중화된 제어 파일들이 모두 훼손되었을 경우, 백업된 제어 파일을 복원하여 archive redo log file과 redo log file까지 불완전 미디어 복구를 수행하여 재시작한다. 백업된 제어 파일을 이용한 불완전 복구는 시나리오별 복구 예제 중에 다중화된 모든 제어 파일이 훼손된 경우를 참조한다.

제어 파일 정보

제어 파일의 위치와 이름에 대한 정보를 조회하기 위해 성능 view인 V\$CONTROLFILE을 이용할 수 있다. 다음은 V\$CONTROLFILE를 이용하여 제어 파일의 이름을 조회하는 예이다.


```
gSQL> SELECT CONTROLFILE_NAME FROM V$CONTROLFILE;
```

```
CONTROLFILE_NAME
```

```
-----  
/goldilocks_data/wal/control_0.ctl
```

```
/goldilocks_data/wal/control_1.ctl
```

```
2 rows selected.
```

GOLDILOCKS의 dump tool인 **gdump**를 이용하면 제어 파일에 기록된 정확한 정보를 확인할 수 있다.

6.2 Redo Log File 관리

GOLDDILOCKS 데이터베이스는 영속성을 보장하기 위해 redo log file을 사용한다. 즉, 다양한 원인으로 인해 GOLDDILOCKS 데이터베이스가 비정상적으로 종료되었을 때 데이터 파일과 redo log를 이용하여 종료하기 직전의 데이터베이스 상태로 복귀할 수 있다.

이를 위해 GOLDDILOCKS 데이터베이스는 수행되는 모든 데이터베이스 갱신 연산들을 Write Ahead Logging (WAL) 정책을 이용하여 로그로 남긴다.

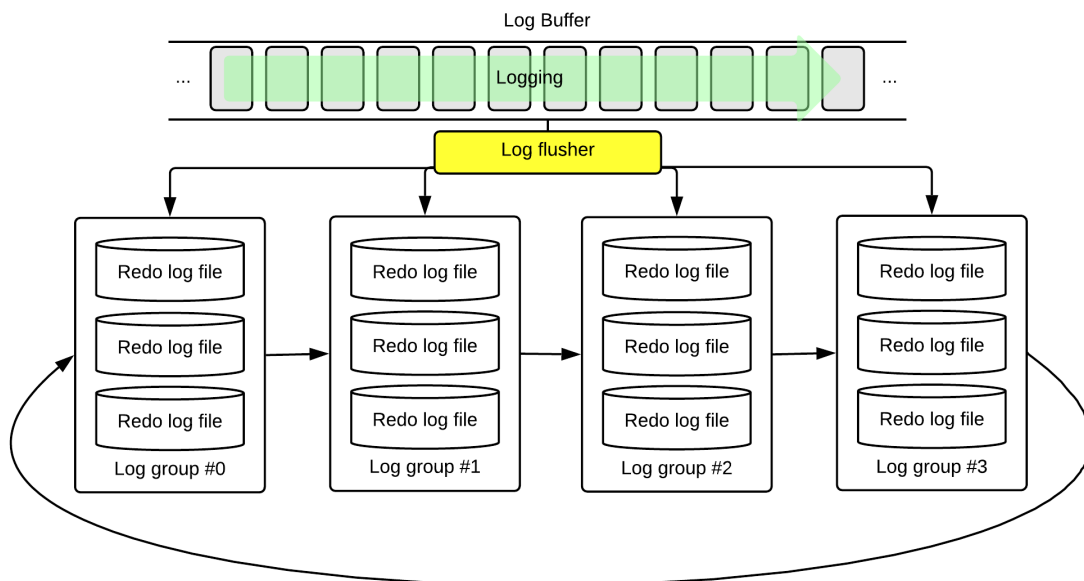
갱신 연산에 의해 갱신된 데이터를 데이터 파일에 기록하지 않고 갱신 연산에 대한 로그를 redo log file에 기록하는 이유는 데이터베이스 성능상 훨씬 효율적이기 때문이다. 갱신 연산이 발생할 때마다 데이터 파일을 기록하면 데이터 파일을 기록하기 위해 랜덤으로 접속하고 동일한 파일에 대한 갱신 연산들의 경합 등으로 인해 디스크 IO가 과도하게 발생한다. 이에 반해 갱신 로그는 갱신 데이터에 비해 크기가 작고 로그 파일의 마지막 위치에 계속 누적 (append)되는 방식으로 효율적인 디스크 IO를 수행한다.

또한 GOLDDILOCKS 데이터베이스는 공유 메모리상의 로그 버퍼를 이용하여 갱신 로그를 로그 버퍼에 기록한 후 로그 버퍼를 일괄적으로 로그 파일에 기록하여 더욱 효율적인 디스크 IO를 수행한다.

Redo Log File 구조

GOLDDILOCKS의 redo log buffer와 로그 파일은 circular 구조이다. 정의된 로그 그룹의 수만큼 미리 로그 파일을 생성하여 로그를 기록하고 하나의 로그 파일이 가득 차면 다음 로그 파일을 사용한다. 로그 파일이 모두 사용되면 이전에 사용되었던 로그 파일을 재사용한다. 로그 파일에는 여러 개의 멤버를 가진 하나의 로그 그룹이 circular 구조로 구성되어 있고 GOLDDILOCKS는 최소 네 개의 로그 그룹을 이용하여 로깅을 수행한다.

그림 1 GOLDDILOCKS redo log file, 로그 buffer 구조



Redo Log 그룹과 멤버

GOLDILOCKS 데이터베이스는 데이터베이스 운용 중에 발생하는 로그를 디스크 로그 파일에 기록하기 위해 로그 그룹과 멤버를 이용한다. 하나의 redo log file은 하나의 로그 그룹의 멤버가 되고 여러 개의 로그 멤버가 모여 로그 그룹을 구성한다. 로그 그룹을 여러 개의 로그 멤버로 구성하면 특정 디스크가 고장나거나 특정 로그 멤버가 훼손되었을 때 다른 로그 멤버를 이용할 수 있어 가용성이 높아진다.

시스템은 하나의 로그 그룹에 로그를 기록하다가 그 로그 그룹이 가득 차면 다음 로그 그룹을 사용하는 방식으로 circular 로그 그룹을 사용한다. 시스템이 사용하는 로그 그룹이 현재 로그 그룹에서 다음 로그 그룹으로 바뀌는 것을 로그 switching이라고 한다.

로그 그룹 및 멤버의 개수와 각각의 위치는 데이터베이스를 생성할 때 프로퍼티에 의해 설정되고 운용 중에 추가 또는 삭제 구문을 이용하여 변경할 수도 있다.

로그 그룹 상태

로그 그룹은 생성될 때 UNUSED 상태로 초기화되고 운용 중에 시스템에 의해 CURRENT, ACTIVE, INACTIVE 상태로 변경된다.

표 6-7 GOLDILOCKS 로그 그룹 상태

로그 그룹 상태	설명
UNUSED	생성 후 사용된 적이 없는 로그 그룹의 상태이다.
CURRENT	현재 시스템이 사용 중인 로그 그룹의 상태이다.
ACTIVE	CURRENT 로그 그룹이 switching 된 후 아직 재사용을 위해 준비되지 않은 상태이다.
INACTIVE	ACTIVE 상태의 로그 그룹이 재사용을 위해 준비된 상태이다.

ACTIVE 상태의 로그 그룹은 시스템에서 재사용할 수 없고 archive log thread에 의해 INACTIVE 상태로 변경된 후에 재사용할 수 있다. Archive log thread는 체크포인트가 완료되기 전에 이벤트에 의해 깨어나 ACTIVE 상태의 로그 그룹들을 INACTIVE 상태로 바꾼다. 이를 위해 시스템이 ARCHIVELOG 모드로 운용 중일 경우 로그 아카이빙 thread는 ACTIVE 상태의 로그 그룹의 로그 파일을 아카이빙한 후 INACTIVE 상태로 변경한다. 만약 NOARCHIVELOG 모드로 운영 중인 경우, 로그 그룹의 상태만 INACTIVE로 변경하여 즉시 재사용 할 수 있도록 한다.

로그 그룹 및 로그 멤버 추가

로그 그룹 추가

로그 그룹은 GOLDILOCKS 데이터베이스 다단계 startup 중 mount 단계에서만 추가할 수 있다. 사용 중인 로그 그룹에 새로운 로그 그룹을 추가하며 'CURRENT' 상태의 다음에 추가된다. 예를 들어 'abc.log'라는 파일 이름으로 파일 크기가 20 Mbyte인 새로운 로그 그룹을 그룹 ID 10으로 추가하는 경우 다음과 같이 수행할 수 있다.

```
ALTER DATABASE ADD LOGFILE GROUP 10 ('abc.log') SIZE 20M;
```

로그 멤버 추가

사용 중인 로그 그룹의 안정성을 위해 새로운 로그 멤버를 추가할 수 있는데 로그 그룹 추가와 마찬가지로 GOLDDILOCKS 데이터베이스 다단계 startup 중 mount 단계에서 가능하다. 예를 들어 로그 그룹 ID 10에 'test.log' 로그 파일을 추가하는 경우 다음과 같이 수행할 수 있다. 이 때 로그 그룹 내의 로그 멤버들은 동일한 파일 크기를 가지기 때문에 로그 파일의 크기를 기술하지 않는다.

```
ALTER DATABASE ADD LOGFILE MEMBER 'test.log' TO GROUP 10;
```

로그 멤버 이름 변경

사용 중인 로그 멤버의 위치와 파일 이름을 변경해야 하는 경우 mount 단계에서 RENAME 할 수 있다. 로그 멤버의 RENAME은 로그 멤버가 위치한 디스크가 물리적으로 고장났거나 성능상 로그 멤버를 다른 디스크로 옮겨야 하는 경우에 수행된다. 다음은 '/disk1/golddilocks_data/wal/redo_0_0.log'의 로그 파일을 '/disk2/golddilocks_data/wal/redo_0_0.log'로 RENAME하는 예이다.

```
ALTER DATABASE RENAME LOGFILE '/disk1/golddilocks_data/wal/redo_0_0.log' TO '/disk2/golddilocks_data/wal/redo_0_0.log';
```

로그 그룹 및 로그 멤버 제거

로그 그룹이나 로그 멤버의 수를 줄이거나 로그 파일이 위치한 디스크 고장 등으로 인해 사용 중인 로그 그룹과 멤버를 제거해야 할 경우, mount 단계에서 DROP 할 수 있다. 로그 그룹 10의 모든 로그 멤버를 제거하려면 다음과 같이 수행한다.

```
ALTER DATABASE DROP LOGFILE GROUP 10;
```

로그 멤버는 해당 로그 그룹에 최소 두 개 이상의 로그 멤버가 존재할 때만 제거할 수 있다. '/disk1/golddilocks_data/wal/redo_0_0.log' 로그 멤버를 제거하려면 다음과 같이 수행한다.

```
ALTER DATABASE DROP LOGFILE MEMBER '/disk1/golddilocks_data/wal/redo_0_0.log';
```

Redo Log File 훼손 시 대처 방안

시스템이 고장난 후에 redo log file이 훼손된 경우 재시작 복구에 실패하여 시스템을 구동할 수 없게 된다. 이 때 훼손된 로그 그룹에 정상적인 로그 멤버가 존재하는 경우, 정상 로그 멤버의 로그 파일을 훼손된 로그 파일에 복사하여 재시작 복구 및 시스템 구동을 수행할 수 있다.

만약 로그 그룹의 로그 파일들이 모두 훼손되었거나 로그 멤버가 하나밖에 없는 경우, **불완전 복구**를 수행하여 정상적인 로그 파일까지만 복구하여 재시작할 수 있다.

Redo Log File 정보

Redo log file의 위치와 이름에 대한 정보를 조회하기 위해 성능 view인 V\$LOGFILE을 이용할 수 있다. V\$LOGFILE를 이용하여 로그 파일 이름과 로그 파일이 속한 로그 그룹 ID, 로그 그룹의 상태, 파일 크기를 다음과 같이 조회할 수 있다.

```
gSQL> SELECT FILE_NAME, GROUP_ID, GROUP_STATE, FILE_SIZE FROM V$LOGFILE;
```

FILE_NAME	GROUP_ID	GROUP_STATE	FILE_SIZE
/disk1/goldilocks_data/wal/redo_0_0.log	0	INACTIVE	104857600
/disk1/goldilocks_data/wal/redo_1_0.log	1	CURRENT	104857600
/disk1/goldilocks_data/wal/redo_2_0.log	2	UNUSED	104857600
/disk1/goldilocks_data/wal/redo_3_0.log	3	UNUSED	104857600

4 rows selected.

6.3 Archive Redo Log File 관리

GOLDDILOCKS의 redo log file은 circular 로그 그룹을 이용하여 사용된 로그 그룹을 재사용하므로 백업을 이용한 미디어 복구를 위해서는 데이터베이스를 archive log mode로 운용해야 하는데 이 때 기록이 완료된 redo log file은 archive redo log file로 복사된다. GOLDDILOCKS 데이터베이스의 archive log mode에 대한 자세한 내용은 **archive log 모드**를 참조한다.

Archive Redo Log File 생성

Redo log file은 GOLDDILOCKS 데이터베이스의 시스템 thread인 log archiving thread에 의해 archive redo log file 디렉토리로 복사된다. Log archiving thread는 체크포인트를 수행할 때 체크포인트 thread에 의해 활성화되어 redo log file 중 아카이빙 되어야 할 대상을 찾아 아카이빙을 수행한다.

Archive redo log file은 ARCHIVELOG_DIR_1 프로퍼티에 설정된 디렉토리에 ARCHIVELOG_FILE 프로퍼티에 설정된 prefix와 각 redo log file의 sequence 번호를 이용한 archive redo log file 이름으로 생성된다.

Archive Redo Log File 보존과 삭제

Redo log file과 마찬가지로 archive redo log file도 임의로 삭제할 경우 백업을 이용한 미디어 복구에 실패할 수 있다. 따라서 archive redo log file은 백업 파일과 함께 보존해야 하며 백업 파일이 더 이상 필요 없을 때 백업을 이용한 미디어 복구를 위해 필요한 archive redo log file도 함께 제거할 수 있다.

백업은 가장 최근에 수행된 체크포인트에 의해 디스크로 내려간 데이터 파일을 복사하는 것이므로 그 백업을 이용하여 미디어 복구를 수행하기 위해서는 체크포인트 시점에 가장 오래된 LSN과 그 이후의 archive redo log file들이 필요하다. 체크포인트는 redo log file이 스위치될 때 시스템에 의해 수행되므로 CURRENT 상태의 redo log file을 제외한 모든 로그 파일에는 최소한 한 개 이상의 체크포인트 로그가 존재한다.

이를 이용하여 백업된 파일을 이용한 미디어 복구 시 필요한 archive redo log file을 구하는 방법은 다음과 같다.

1. 백업된 데이터 파일의 파일 헤더에 기록된 체크포인트 LSN을 구한다.
2. Archive redo log file을 덤프하여 체크포인트 LSN을 포함하는 archive redo log file을 구한다.
3. 백업을 이용한 미디어 복구 시 필요한 archive redo log file은 2에서 구한 archive redo log file의 직전 로그 파일부터가 된다.

증분 백업에 필요한 archive redo log file을 구하기 위해서는 제어 파일을 덤프하여 체크포인트 LSN을 구해야 하며 이후 과정은 전체 백업과 동일하다.

백업 파일이 더 이상 필요 없을 때 백업을 이용한 미디어 복구를 위해 필요한 archive redo log file도 함께 제거할 수 있다.

Archive Redo Log File 디렉토리 다중화

Archive redo log file을 ARCHIVELOG_DIR_1에 설정된 디렉토리에서 다른 미디어나 디렉토리로 옮기면 미디어 복구에 필요한 archive redo log file을 찾지 못하여 수행에 실패한다. 이 경우 옮겨진 archive redo log file을 ARCHIVELOG_DIR_1에 설정된 디렉토리로 다시 옮겨서 수행하거나 ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10에 archive redo log file이 존재하는 디렉토리를 설정하는 방식으로 미디어 복구를 위한 archive redo log file 디렉토리를 추가하여 수행할 수 있다. 단, 미디어 복구를 위해 ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10을 사용하기 위해서는 READABLE_ARCHIVELOG_DIR_COUNT를 디렉토리 개수만큼 설정해야 한다.

6.4 테이블스페이스 관리

데이터베이스에서 사용되는 모든 데이터들은 물리적인 디스크 파일에 저장되고 데이터의 효율적인 관리와 성능 향상을 위해 데이터베이스의 논리적인 구조를 이용한다. GOLDDILOCKS는 테이블스페이스, 세그먼트, extent, page 등과 같은 논리적 구조를 이용하여 디스크 사용 공간을 효율적으로 관리한다.

테이블스페이스는 여러 개의 데이터 파일을 포함할 수 있으며 각각의 테이블스페이스를 online/ offline 상태로 설정하면 데이터의 가용성을 향상시킬 수 있다. 또한 데이터 파일을 저장하는 디스크를 분산시켜 IO 성능을 향상시키고 물리적인 디스크의 IO 경합을 감소시킨다.

테이블스페이스 종류

GOLDDILOCKS의 테이블스페이스는 데이터베이스를 생성할 때 생성되어 GOLDDILOCKS 시스템만 사용하고 제어하는 SYSTEM 테이블스페이스와 사용자가 생성하고 사용하는 non SYSTEM 테이블스페이스가 있다.

SYSTEM 테이블스페이스

GOLDDILOCKS 데이터베이스가 생성될 때 생성되며 데이터베이스 운영을 위해 반드시 필요한 테이블스페이스이다. Dictionary 테이블스페이스, Undo 테이블스페이스, 그리고 system temporary 테이블스페이스가 있다.

Non SYSTEM 테이블스페이스

데이터를 저장하기 위한 테이블과 인덱스를 저장하기 위해 사용자가 임의로 생성하고 삭제할 수 있는 테이블스페이스들이다.

테이블스페이스 및 데이터 파일 관리

테이블스페이스 관리

테이블스페이스 상태 관리

GOLDDILOCKS 데이터베이스의 테이블스페이스는 online과 offline으로 상태를 나눌 수 있다. Offline 상태의 테이블스페이스에는 접근할 수 없다. 사용자가 특정 테이블스페이스를 임의로 offline 상태로 설정할 수 있고 시스템에 의해 비정상 상태의 테이블스페이스가 offline 될 수도 있다. 시스템 테이블스페이스는 offline 상태로 설정할 수 없다.

- Offline 테이블스페이스

GOLDILOCKS 데이터베이스는 테이블스페이스가 offline 상태가 되기 전에 테이블스페이스 내의 모든 데이터 파일을 디스크로 flush 한다. 데이터 파일을 flush 하려면 관련 로그들을 모두 디스크로 flush 해야 하기 때문에 이후에 online으로 상태가 바뀔 때 특별히 복구를 수행할 필요없고 단지 offline 테이블스페이스 상태에서 발생한 DDL들이 온라인 상태로 바뀔 때 적용된다.

데이터 파일을 flush하지 않고 offline으로 바꾸는 IMMEDIATE 모드를 이용하면 테이블스페이스를 즉시 offline으로 바꿀 수 있는데 이 경우 online으로 설정하려면 미디어 복구를 수행한 후에 online 상태로 바꾸어야 한다.

표 6-8 테이블스페이스 offline 옵션

옵션	설명	테이블스페이스를 online으로 바꿀 경우
NORMAL	테이블스페이스의 데이터 파일과 관련된 로그를 모두 디스크로 flush한 후 offline으로 바꾼다.	미디어 복구를 수행할 필요없다.
IMMEDIATE	테이블스페이스를 즉시 offline으로 바꾼다.	미디어 복구를 수행해야 한다.

GOLDILOCKS는 mount 단계에서 테이블스페이스를 offline으로 바꿀 수 있다. 이는 데이터베이스를 시작할 때 복구할 수 없는 테이블스페이스를 제외하고 시작하여 서비스를 수행함으로써 가용성을 높인다. Mount 단계에서 테이블스페이스를 offline하는 것은 서버가 정상적으로 종료되었거나 ARCHIVELOG 모드로 서버를 운용할 때만 가능하다.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;
```

```
Tablespace altered.
```

```
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
```

```
Tablespace altered.
```

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE IMMEDIATE;
```

```
Tablespace altered.
```

```
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
```

```
ERR-42000(14051): media recovery required - 'TEST_TBS'
```

```
gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;
```

```
Database altered.
```

```
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
```

```
Tablespace altered.
```

테이블스페이스 속성

GOLDDILOCKS 데이터베이스의 테이블스페이스 속성은 다음과 같다. 영속성 보장 여부를 구분하는 PERSISTENT, TEMPORARY 속성, 테이블스페이스와 저장되는 데이터의 종류에 따른 DATA, UNDO의 속성이 있다.

표 6-9 GOLDDILOCKS 데이터베이스의 테이블스페이스 속성

속성		설명
영속성	PERSISTENT	테이블스페이스에 저장된 데이터들에 대한 영속성을 지원한다. (복구 대상)
	TEMPORARY	테이블스페이스에 저장된 데이터들에 대한 영속성을 지원하지 않는다.
저장 데이터 종류	DATA	사용자가 입력한 데이터가 저장된다.
	UNDO	데이터베이스의 MVCC를 위해 필요한 데이터가 저장된다.
	DICT	데이터베이스 운영을 위한 dictionary 정보가 저장된다.
	TEMPORARY	SQL 처리를 위한 데이터들이 저장된다.

테이블스페이스 관리

- 사용자 테이블스페이스 생성

새로운 사용자 테이블스페이스를 생성한다. 테이블스페이스를 생성할 때 데이터베이스 인스턴스에서 사용 중인 테이블스페이스 이름은 고유해야 한다. 테이블스페이스 한 개당 최대 1,024 개의 데이터 파일을 가질 수 있고 각 데이터 파일에는 최대 30 Gbyte까지 저장할 수 있다. 데이터베이스에서 생성할 수 있는 최대 테이블스페이스의 개수는 시스템 테이블스페이스를 포함하여 65,535개이다.

데이터 파일 이름은 데이터 파일 절대 경로를 포함하여 데이터베이스에서 유일해야 하고, 이미 생성된 데이터 파일 중에 데이터베이스에서 사용하고 있지 않은 파일을 재사용하려면 'REUSE' 옵션을 사용한다. 테이블스페이스에서 사용할 익스텐트 (extent) 한 개의 크기는 64 Kbyte, 128 Kbyte, 256 Kbyte, 512 Kbyte, 1 Mbyte 중 하나를 선택할 수 있으며 기본값은 256 Kbyte 이다.

```
gSQL> CREATE TABLESPACE TEST_TBS DATAFILE
      '/goldilocks1/db/TEST_TBS1.dbf' SIZE 20M,
      '/goldilocks2/db/TEST_TBS2.dbf' SIZE 50M,
      '/goldilocks3/db/TEST_TBS3.dbf' SIZE 100M REUSE;
Tablespace created.
```

테이블스페이스를 생성할 때 테이블스페이스의 online/ offline 상태를 설정할 수 있고 LOGGING, NOLOGGING 테이블스페이스 속성을 설정할 수 있다.

- 테이블스페이스 제거

테이블스페이스가 더 이상 필요하지 않으면 테이블스페이스와 해당 데이터 파일을 제거할 수 있다. 테이블스페이스가 생성되면 테이블스페이스에 추가된 디스크 데이터 파일과 메모리가 생성되어 유지되기 때문에 사용하지 않는 테이블스페이스가 있으면 자원이 낭비되므로 삭제하는 것이 좋다.

```
gSQL> DROP TABLESPACE TEST_TBS;
Tablespace dropped.
```

기본적으로 테이블스페이스 제거는 테이블스페이스에 생성되어 사용 중인 테이블 인덱스까지 제거하지는 않기 때문에 만약 사용 중인 테이블이나 인덱스가 존재하는 테이블스페이스를 제거하려면 INCLUDING CONTENTS 옵션을 함께 사용해야 한다.

```
gSQL> DROP TABLESPACE TEST_TBS;
ERR-42000(16148): tablespace not empty, use INCLUDING CONTENTS option :
drop tablespace TEST_TBS
      *
ERROR at line 1:
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS;
Tablespace dropped.
```

그리고 테이블스페이스에 추가된 데이터 파일까지 삭제하려면 AND DATAFILES 옵션이 필요하다.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

- 테이블스페이스 크기 조정

테이블스페이스의 크기를 조정하기 위해서는 테이블스페이스에 데이터 파일을 추가하거나 삭제한다. 데이터베이스를 사용하는 중에 데이터를 저장할 공간이 부족할 경우 테이블스페이스에 새로운 데이터 파일을 추가하여 공간을 확보하고 사용하지 않는 테이블스페이스의 데이터 파일은 삭제하여 공간 낭비를 줄일 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD DATAFILE 'TEST_TBS2.dbf' SIZE 20M;
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf';
Tablespace altered.
```

테이블스페이스의 데이터 파일은 생성된 후 사용된 적이 없는 경우에만 삭제할 수 있다. 사용된 적이 있는 경우, 모든 데이터가 삭제되었더라도 데이터 파일을 삭제할 수 없다.

```
ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf';
ERR-42000(14044): datafile not empty
```

- 임시 테이블스페이스 관리

임시 테이블스페이스는 데이터 파일을 가지지 않고 지정된 크기만큼 메모리만 할당한다. 다음과 같이 임시 테이블스페이스를 생성할 수 있다.

```
gSQL> CREATE TEMPORARY TABLESPACE TEST_TBS MEMORY 'TEST_TEMP_TBS' SIZE 10M EXTSIZE 256K;
Tablespace created.
```

다음과 같이 임시 테이블스페이스의 메모리를 추가할 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD MEMORY 'TEST_TBS2' SIZE 10M;
Tablespace altered.
```

다음과 같이 임시 테이블스페이스에서 사용되지 않은 메모리를 삭제할 수 있다.

```
gSQL> ALTER TABLESPACE TEST_TBS DROP MEMORY 'TEST_TBS2';
Tablespace altered.
```

다음과 같이 임시 테이블스페이스를 제거할 수 있다.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

데이터 파일 이동

데이터 파일이 저장된 디스크를 옮기거나 디렉토리를 변경할 경우, 데이터베이스에 저장된 데이터 파일의 경로를 변경해야 한다.

다음은 'TEST_TBS' 테이블스페이스의 '/goldilocks1/db/TEST_TBS1.dbf' 데이터 파일을 '/goldilocks4/db/TEST_TBS1.dbf'로 이동하기 위해 경로를 변경하는 예이다.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE
      '/goldilocks1/db/TEST_TBS1.dbf' TO '/goldilocks4/db/TEST_TBS1.dbf';
Tablespace altered.
```

테이블스페이스 정보

데이터베이스에서 생성된 테이블스페이스의 정보를 보려면 **V\$TABLESPACE**를 참조한다.

```
gSQL> \DESC V$TABLESPACE
```

COLUMN_NAME	TYPE	IS_NULLABLE
TBS_NAME	VARCHAR(128)	FALSE
TBS_ID	NUMBER	FALSE
TBS_ATTR	VARCHAR(128)	FALSE
IS_LOGGING	BOOLEAN	FALSE
IS_ONLINE	BOOLEAN	FALSE
OFFLINE_STATE	VARCHAR(32)	FALSE

EXTENT_SIZE	NUMBER	FALSE
PAGE_SIZE	NUMBER	FALSE

6.5 데이터 파일 관리

데이터 파일 정합성

데이터 파일의 정합성은 디스크 고장이나 데이터베이스 결함, 사용자의 실수로 인해 깨어질 수 있다. 데이터 파일의 정합성이 깨어진 것을 인식하지 못한 채 데이터베이스를 이용하여 서비스하면 심각한 문제가 발생한다.

이를 위해 GOLDDILOCKS 데이터베이스는 데이터 파일의 각 페이지 별 checksum을 이용하여 데이터 파일의 정합성을 보장한다. LSN과 CRC 값을 이용하여 GOLDDILOCKS 데이터베이스의 페이지 checksum을 생성하고 각 페이지에 저장한다. 사용자는 **PAGE_CHECKSUM_TYPE** 값 설정을 통해 페이지 checksum 종류를 설정하고 기본값으로 LSN을 사용한다.

페이지 checksum은 데이터베이스를 시작할 때 데이터 파일을 메모리에 로드하는 시점에 체크되는데 만약 checksum 값에 오류가 발생할 경우 데이터베이스를 이용한 서비스가 불가능하여 시작할 수 없다.

예를 들어, 사용자가 생성한 테이블스페이스 TEST_TBS의 데이터 파일 TEST_TBS.dbf의 정합성이 깨어진 경우, 데이터베이스 시작은 다음과 같이 실패한다.

```
gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
```

데이터 파일의 정합성이 깨진 경우, 데이터 파일을 소유한 테이블스페이스를 offline 상태로 변경한 후에 데이터베이스를 시작하거나 해당 데이터 파일을 복구한 후에 데이터베이스를 시작할 수 있다.

다음은 테이블스페이스를 offline으로 변경한 후에 데이터베이스를 시작하는 방법이다.

```
gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE IMMEDIATE;
Tablespace altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

데이터 파일을 복구하려면 데이터 파일에 대한 전체 백업 혹은 증분 백업이 존재해야만 한다. 백업이 존재하는 경우 다음과 같이 복구를 수행한다.

```

gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
gSQL> ALTER DATABASE RECOVER DATAFILE 'TEST_TBS.dbf' CORRUPTION;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.

```

데이터 파일 정보

데이터베이스에서 사용 중인 데이터 파일의 정보를 보려면 **V\$DATAFILE**을 참조한다.

```

gSQL> \DESC V$DATAFILE

```

COLUMN_NAME	TYPE	IS_NULLABLE
TBS_NAME	VARCHAR(128)	FALSE
DATAFILE_NAME	VARCHAR(1024)	FALSE
CHECKPOINT_LSN	NUMBER	FALSE
CREATION_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	FALSE
FILE_SIZE	NUMBER	FALSE

7.

GOLDILOCKS 데이터베이스의 백업과 복구

본 장에서는 GOLDDILOCKS 데이터베이스의 백업과 복구에 대해 설명한다. 또한 백업과 복구를 위한 데이터베이스 ARCHIVELOG 모드에 대해 설명한다.

7.1 ARCHIVELOG 모드

GOLDDILOCKS 데이터베이스는 circular 로그 그룹을 이용하여 로깅을 수행한다. 최소 네 개의 로그 그룹으로 구성되는데 하나의 로그 그룹에 할당된 로그 파일이 소진되면 다음 로그 그룹을 이용하는 방식으로 로깅을 수행한다. 생성된 로그 그룹을 모두 소진하였을 경우 처음 로그 그룹을 재사용하는데 이 때 새로운 로그 파일을 생성하지 않고 이전에 기록된 로그 파일을 재사용한다.

NOARCHIVELOG 모드에서 로그 그룹이 재사용되면 이전에 기록된 로그는 잃어버린다. 따라서 NOARCHIVELOG 모드로 운영하는 경우, 운영자가 로깅이 완료된 로그 그룹을 관리하지 않으면 완료된 트랜잭션의 로그가 시간이 지나면 사라진다.

ARCHIVELOG 모드의 경우 로그 그룹의 로그 파일에 기록이 완료되어 다음 로그 그룹을 사용할 때 시스템이 로그 파일을 백업 (archive)한 후 재사용하기 때문에 기록이 완료된 로그는 임의로 지우지 않는 한 영구적으로 보존된다.

ARCHIVELOG 모드

백업을 이용한 복구를 위해서는 백업 시점 이후의 모든 로그 파일들이 필요한데 백업을 언제 사용할지 모르기 때문에 로그 파일을 재사용하기 전에 반드시 archive 해야 한다. 따라서 백업은 ARCHIVELOG 모드에서만 지원된다.

ARCHIVELOG 모드로 운용하는 경우, busy한 시스템에서 archiving으로 인한 끊김 현상이 발생할 수 있고 archive 로그 파일을 저장하기 위한 공간도 추가로 필요하다.

NOARCHIVELOG 모드

NOARCHIVELOG 모드로 운용하는 경우, 재사용 이전의 로그 파일이 존재하는지 확인할 수 없기 때문에 백업을 지원하지 않는다.

하지만 시스템이 로그 파일을 archiving 하지 않기 때문에 대량의 로그가 꾸준히 기록되는 상황에서 체크포인트 할 때 로그 archiving으로 인한 끊김 현상이 없고 archive 로그 파일들을 위한 저장 공간도 필요없다는 장점이 있다.

NOARCHIVELOG 모드는 데이터베이스를 생성할 때 'ARCHIVELOG_MODE' 프로퍼티 값에 따라 설정된다. 'ARCHIVELOG_MODE'가 0이면 NOARCHIVELOG 모드로, 1이면 ARCHIVELOG 모드로 데이터베이스가 생성된다. 이 프로퍼티는 데이터베이스를 생성할 때만 유효하고 운영 중에는 참조되지 않는다.

데이터베이스 운영 중에 ARCHIVELOG 모드를 변경하려면 GOLDILOCKS 데이터베이스의 시작 단계 중 mount 단계에서 다음 구문을 수행한다.

```
gSQL> ALTER DATABASE ARCHIVELOG;
```

```
Database altered.
```

```
gSQL> ALTER DATABASE NOARCHIVELOG;
```

```
Database altered.
```

데이터베이스에 설정된 ARCHIVELOG 모드를 확인하려면 성능 view V\$ARCHIVELOG의 ARCHIVELOG_MODE를 확인한다.

```
gSQL> SELECT ARCHIVELOG_MODE FROM V$ARCHIVELOG;
```

```
ARCHIVELOG_MODE
```

```
-----
```

```
NOARCHIVELOG
```

```
1 row selected.
```

7.2 백업과 복구

백업

백업/ 복구 목적

데이터베이스는 다양한 장애와 데이터 손실이 발생했을 경우 데이터를 보호하고 복구할 수 있다. 장애의 원인은 다양하고 특히 데이터베이스가 물리적으로 손상되거나 재해로 망가졌을 경우에 대비하여 복사본을 만들어 두어야 하는데 이를 백업이라고 한다.

여러가지 장애로 인해 데이터베이스 서비스가 불가능할 때 현재 데이터베이스나 백업을 이용하여 서비스 가능한 상태로 만들 수 있는데 이를 복구라고 한다. 이 중 현재 데이터베이스를 이용하여 복구하는 것을 재시작 복구 (restart recovery)라고 하고 백업을 이용하여 복구하는 것을 미디어 복구 (media recovery)라고 한다. GOLDDILOCKS는 미디어 복구를 자동 또는 수동으로 수행하여 백업 데이터 파일을 복원한 후에 재시작 복구로 미디어 복구와 재시작을 수행한다.

백업

데이터베이스 백업은 다음과 같이 물리적 백업과 논리적 백업으로 구분할 수 있다. 일반적으로 백업은 온라인 상태에서 데이터 파일에 대한 복사본을 만드는 것을 의미하며 본 절에서는 온라인 상태의 물리적 백업에 대해 설명한다.

표 7-1 데이터베이스 백업 유형

백업 유형	백업 형태	데이터베이스 상태	특징
물리적 백업	콜드 백업	오프라인	데이터 파일의 복사본을 생성한다. 백업 수행을 위해 서비스를 중단한다.
	핫 백업	온라인	데이터 파일의 복사본을 생성한다. 서비스를 운영하면서 백업을 수행할 수 있다. ARCHIVELOG mode에서만 가능하다.
논리적 백업	익스포트 백업	온라인	테이블 단위로 백업/ 복구를 수행한다. H/W, OS 구분 없이 export 할 수 있다.

GOLDDILOCKS가 서비스를 위해 사용하고 장애로 인해 손상되었을 때 복구해야 할 파일들은 데이터 파일들과 제어 파일이다. 제어 파일은 데이터베이스를 생성할 때 만들어지는 파일이며 데이터베이스 운용을 위해 필요한 정보들이 저장된다. 데이터 파일들은 실제 데이터가 저장되는 파일들로서 데이터베이스를 생성할 때 기본적으로 생성되는 시스템 테이블스페이스들의 데이터 파일과 사용자가 만든 테이블스페이스의 데이터 파일들이다. 이런 제어 파일이나 데이터 파일들 중 일부가 손상되면 백업된 파일들을 이용하여 복구한다.

즉, 복구를 위해 백업해야 하는 파일들은 제어 파일과 데이터 파일들이고 이를 위해 GOLDDILOCKS는 제어 파일 백업, 데이터베이스 백업, 테이블스페이스 백업을 지원한다.

백업하는 방식에 따라 전체 백업 (full backup)과 증분 백업 (incremental backup)으로 나뉜다. 전체 백업은 백업 시

점의 데이터 파일 복사본을 생성하는 것이고, 증분 백업은 이전 백업 이후에 변경된 부분만 백업하는 방식이다. 전체 백업의 경우 데이터 파일 복사본을 생성하기 때문에 백업할 때마다 데이터 파일과 동일한 크기의 복사본이 만들어지고 이로 인해 백업할 때마다 데이터베이스나 테이블스페이스 크기만큼의 저장 공간이 소모된다.

이에 반해 증분 백업은 이전 백업 이후에 변경된 부분만 백업하기 때문에 상대적으로 백업 크기가 작다는 장점이 있다.

표 7-2 전체 백업과 증분 백업

구분	전체 백업	증분 백업
백업 대상	- 데이터베이스: 데이터베이스가 사용 중인 전체 데이터 파일 - 테이블스페이스: 데이터베이스의 특정 테이블스페이스의 데이터 파일	
특징	- 데이터베이스나 테이블스페이스가 사용 중인 데이터 파일 전체를 백업한다. - 데이터 파일 한 개당 백업 파일 한 개가 생성된다. - 장애가 발생한 후에 필요한 데이터 파일을 적절한 백업을 통해 복원한 후에 복구한다.	- 데이터베이스나 테이블스페이스가 사용 중인 데이터 파일에서 이전에 백업된 후에 변경된 부분을 백업한다. - 백업 결과 변경된 부분이 기록된 한 개의 증분 백업 파일을 생성한다. - 장애가 발생한 후에 여러 개의 증분 백업을 이용한 복원을 통해 복구한다.

전체 백업

전체 백업을 이용하여 제어 파일과 데이터 파일 백업을 위한 데이터베이스 백업, 테이블스페이스 백업을 수행한다.

제어 파일 백업

다음과 같이 제어 파일을 백업한다. 제어 파일을 백업하려면 (절대 경로를 포함한) 백업 제어 파일 이름을 기술한다. 백업 제어 파일 이름만 기술하는 경우 LOG_DIR 프로퍼티에 설정된 경로에 백업 파일이 생성된다.

```
gSQL> ALTER DATABASE BACKUP CONTROLFILE TO '/goldilocks_data/backup/backup.ctl';
Database altered.
```

데이터베이스 백업

데이터베이스가 사용하는 모든 데이터 파일들을 백업하는 기능이다. 데이터 파일을 백업하려면 데이터 파일 복사본을 만드는 도중에 해당 파일에 쓰지 못하도록 해야 한다. 만약 복사하는 중에 파일에 쓰여지면 데이터 파일의 일관성이 깨어지고 더 나아가 한 페이지 내에서 일관성이 깨지는 상황이 발생할 수 있다. 이를 방지하기 위해 먼저 다음과 같이 데이터베이스를 백업할 수 있는 상태로 설정한다.

```
gSQL> ALTER DATABASE BEGIN BACKUP;
Database altered.
```

데이터베이스를 백업할 수 있는 상태에서 운영체제가 지원하는 파일 복사 기능으로 데이터 파일들의 복사본을 생성한 후에 다음과 같이 데이터베이스 백업이 완료되어 쓰기 가능한 상태로 설정한다.

```
gSQL> ALTER DATABASE END BACKUP;
Database altered.
```

테이블스페이스 백업

하나의 특정 테이블스페이스가 사용하는 데이터 파일들을 백업할 수 있는 기능이다. 데이터베이스 백업과 동일한 이유로 다음과 같이 백업할 테이블스페이스 이름 (tablespace_name)을 사용하여 백업할 수 있는 상태로 설정한다.

```
gSQL> ALTER TABLESPACE TEST_TBS BEGIN BACKUP;
Tablespace altered.
```

테이블스페이스를 백업할 수 있는 상태에서 운영체제가 지원하는 파일 복사 기능으로 테이블스페이스가 소유한 데이터 파일들의 복사본을 생성한 후에 다음과 같이 테이블스페이스 백업을 완료한다.

```
gSQL> ALTER TABLESPACE TEST_TBS END BACKUP;
Tablespace altered.
```

증분 백업

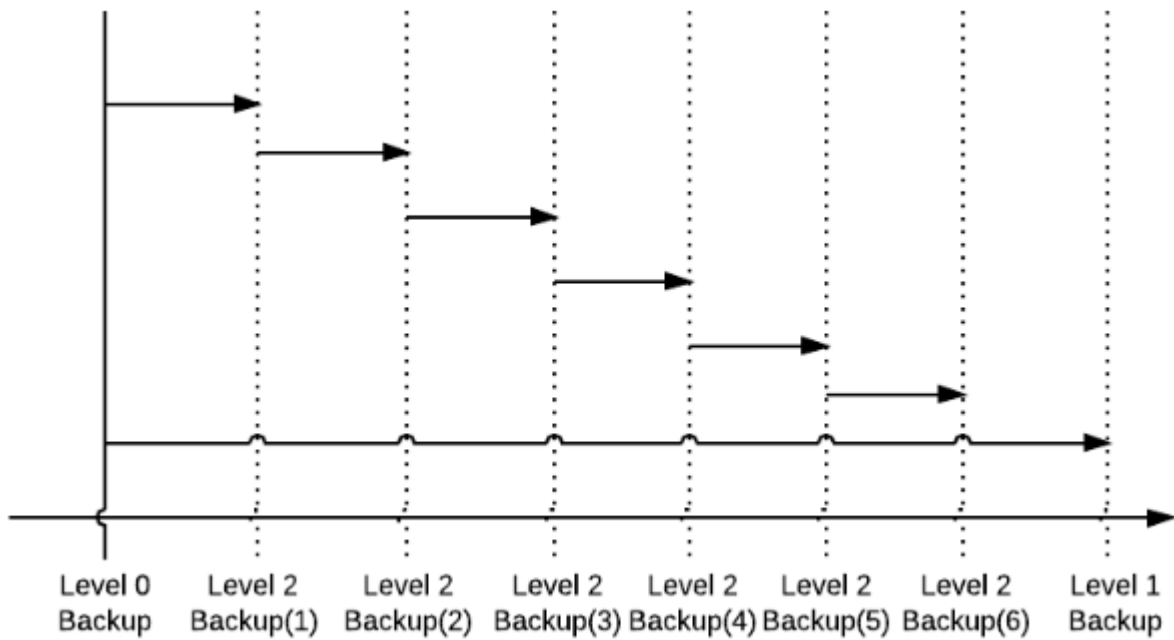
전체 백업과 동일하게 데이터베이스 단위와 테이블스페이스 단위의 증분 백업을 지원하는데 제어 파일 백업을 위한 별도의 증분 백업 기능은 없으며 데이터베이스를 증분 백업할 때 제어 파일도 함께 백업한다.

GOLDBLOCKS는 증분 백업을 위해 0부터 4까지의 증분 레벨을 지원한다. 처음으로 증분 백업을 수행하는 경우 반드시 레벨 0으로 백업을 수행하여 데이터 파일 전체를 백업한다. 그리고 이후 증분 백업을 수행할 때 레벨을 1이상으로 설정하여 이전 백업 이후 변경된 부분만 백업한다.

증분 백업할 레벨이 주어지면 백업 직전에 같거나 작은 레벨로 백업한 시점을 찾아서 그 이후에 변경된 부분을 백업한다.

예를 들어, 다음 그림에서 level 0 백업을 수행한 후 level 2로 백업하면 (1) level 0으로 백업한 후에 변경분만 백업하고 level 2 백업 (2)은 level 2 백업 (1) 이후에 변경된 부분만 백업한다. 마찬가지로 level 2 백업 (3), (4), (5), (6)은 이전에 level 2로 백업한 후에 변경된 부분만 백업하고 마지막으로 수행된 level 1 백업은 level 0 백업 후에 변경된 부분을 모두 백업한다.

그림 1 Incremental backup



데이터베이스 증분 백업

다음과 같이 데이터베이스 전체 데이터 파일을 대상으로 증분백업을 수행한다. 먼저 레벨 0으로 전체 데이터베이스의 데이터 파일을 백업한다.

```
gSQL> ALTER DATABASE BACKUP INCREMENTAL LEVEL 0;
Database altered.
```

그리고 레벨 1로 레벨 0 이후에 변경된 부분만 백업한다.

```
gSQL> ALTER DATABASE BACKUP INCREMENTAL LEVEL 1;
Database altered.
```

테이블스페이스 증분 백업

데이터베이스 증분 백업과 동일하게 먼저 레벨 0으로 테이블스페이스의 전체 데이터 파일을 백업한다.

```
gSQL> ALTER TABLESPACE TEST_TBS BACKUP INCREMENTAL LEVEL 0;
Tablespace altered.
```

그리고 레벨 1로 레벨 0 이후에 변경된 부분만 백업한다.

```
gSQL> ALTER TABLESPACE TEST_TBS BACKUP INCREMENTAL LEVEL 1;
Tablespace altered.
```

복구

데이터베이스는 장애가 발생하거나 데이터가 훼손되었을 때, 즉, 데이터의 일관성 (consistency)이 깨진 경우에 복구를 수행하여 데이터베이스 데이터의 일관성을 보장한다.

데이터베이스 장애 유형은 다음과 같다.

표 7-3 데이터베이스 장애 유형

장애 유형	원인 및 증상	해결책
Transaction Failure	논리적인 오류 (bad input, overflow, data not found)로 인해 transaction 수행에 실패했다. Deadlock 상태이다.	Transaction abort
System Crash	DBMS나 OS의 비정상 종료 (정전)로 인해 휘발성 저장 장치가 훼손되었다.	Restart recovery
Media Failure	비휘발성 저장 장치가 훼손되었다.	Restore, restart recovery

Transaction failure의 경우, 수행 중인 transaction을 abort하여 수행되었던 데이터베이스 갱신을 모두 rollback하고 획득한 lock item을 모두 release하여 해결한다.

System crash의 경우, 데이터베이스 process들이 종료되어 비휘발성 저장 장치에 반영되지 못한 채 휘발성 저장 장치에만 남아있던 내용들이 모두 사라진 상태이다. 따라서 데이터베이스를 startup하면서 비정상 종료 직전에 일관성 있던 데이터베이스의 상태로 복구하는데 이를 재시작 복구 (restart recovery)라고 한다. 재시작 복구는 장애 직전에 데이터베이스가 사용했던 제어 파일, 데이터 파일과 로그 파일 (redo log)을 이용하여 복구를 수행한다.

비휘발성 저장 장치까지 모두 훼손된 경우에는 복구를 수행할 제어 파일, 데이터 파일과 로그 파일까지 훼손되어 장애 직전의 데이터베이스의 파일을 이용해서 복구를 수행할 수 없는 상태가 된다. 이 경우, 이전에 받아 놓은 백업과 로그 파일 (archive log)을 이용하여 데이터베이스 파일을 복원한 후 복구를 수행한다.

GOLDBLOCKS는 완전 복구와 불완전 복구를 지원한다. 완전 복구는 로그 파일들을 이용하여 데이터 파일을 최신의 일관성있는 상태로 복구한다. 완전 복구의 대상은 데이터베이스, 테이블스페이스, 데이터 파일이며 테이블스페이스, 데이터 파일의 경우 데이터베이스 서비스 중에도 offline된 테이블스페이스를 복구할 수 있다. 완전 복구는 데이터베이스를 재시작할 때 수행되는 자동 복구와 GOLDBLOCKS가 지원하는 복구 구문을 이용한 수동 복구로 나뉜다.

불완전 복구는 데이터베이스에 대해서만 가능하며 특정 시점까지의 일관성 있는 상태로 복구한다. 불완전 복구는 수동으로만 수행되며 특정 시점까지 일괄적으로 불완전 복구하거나 복구 가능한 로그 파일을 사용자가 선택하여 해당 파일까지 복구하는 사용자 선택 불완전 복구를 수행한다.

완전 복구와 불완전 복구가 모두 필요한 경우, redo log와 archive log 파일들을 이용한다.

표 7-4 데이터베이스 복구의 분류

복구 분류	복구 대상	구분
완전 복구	데이터베이스, 테이블스페이스, 데이터 파일	- 자동복구 (재시작시 복구) - 수동복구 (데이터베이스/테이블스페이스/데이터파일 복구를 수동으로 수행)
		수동복구만 수행

복구 분류	복구 대상	구분
불완전 복구	데이터베이스	- 일괄 불완전 복구 - 사용자 선택 불완전 복구

자동 복구

자동 복구는 데이터베이스를 정상 또는 비정상 종료 후에 재시작할 때 수행되며 대부분 종료 직전에 사용 중이던 제어 파일, 데이터 파일, 로그 파일을 이용하여 수행한다. 특히 최신 데이터베이스 파일이 손상된 경우, 백업된 데이터베이스 파일을 복원하고 archive log 파일을 이용하여 수행한다.

복구는 analysis, redo, undo의 세 단계로 수행된다.

Analysis 단계

분석 단계에서 하는 일은 두 가지이다. 먼저 재시작 복구를 수행할 첫 번째 로그를 찾는다. 이를 위해 가장 최근에 수행된 checkpoint 로그를 참조하는데 가장 최근에 수행된 checkpoint 로그는 제어 파일에 기록된 로그 정보에서 구한다. 그 다음에 시스템의 transaction table을 초기화한다. Checkpoint 로그에 기록된 checkpoint 시 수행 중이던 transaction들의 정보를 이용하여 시스템의 transaction table을 초기화한다.

Restart Redo 단계

분석 단계에서 구한 재시작 복구를 위한 첫 번째 로그부터 redo log file에 기록된 마지막 로그까지 모두 리두 복구한다. 이 과정에서 transaction이 완료되거나 새로 시작되면 transaction table이 갱신된다.

Restart Undo 단계

Redo 복구가 끝난 후 완료되지 않고 transaction table에 남아 있는 transaction들을 모두 undo 하여 transaction rollback을 수행한다.

백업을 이용한 복구

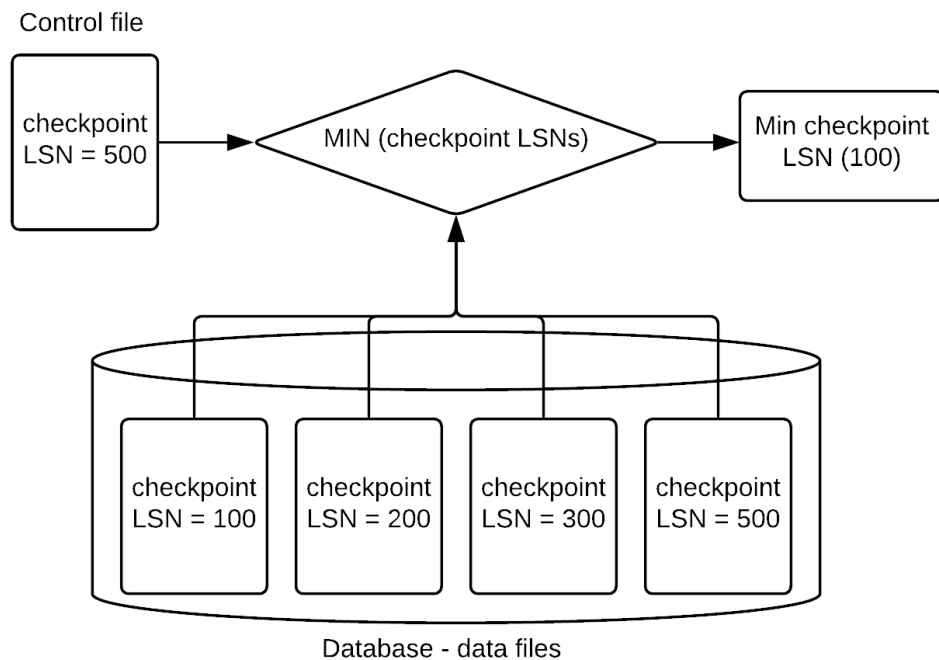
제어 파일, 데이터 파일, 로그 파일이 훼손되거나 존재하지 않는 경우, 백업된 파일을 이용하여 복원한 후 복구를 수행해야 한다. 백업된 제어 파일이나 데이터 파일을 이용하여 복구를 수행하는 경우, 복구를 시작할 로그를 찾는 방법이 복잡하다.

백업을 이용한 복구를 위한 분석

백업을 이용한 복구 분석 단계에서는 자동 복구와 동일하게 복구를 수행할 첫 번째 로그를 찾고 transaction table을 초기화한다. 복구를 시작할 첫 번째 로그를 찾는 방법은 모든 데이터 파일의 파일 헤더에 기록된 checkpoint LSN 중에 가장 오래된 LSN을 구한 후 제어 파일에 기록된 checkpoint LSN과 비교하여 최소값을 선택하는 것이다.

데이터 파일 헤더에 기록된 checkpoint LSN은 해당 데이터 파일이 checkpoint된 LSN을 저장하기 때문에 백업된 데이터 파일을 이용할 경우 가장 오래된 checkpoint LSN을 선택한 후 제어 파일의 checkpoint LSN과 비교하여 최소값을 선택하면 복구를 위한 최소 checkpoint LSN이 결정된다.

그림 2 Recovery를 위한 최소 checkpoint LSN을 구하는 과정



Archive Log File을 이용한 복구

Archive log file을 이용한 복구는 복구를 위해 구한 최소 checkpoint LSN이 archive log file에 존재할 경우, redo log file 뿐만 아니라 archive log file까지 이용하여 복구를 수행한다. 백업을 이용한 복구 대상은 데이터베이스, 테이블스페이스, 데이터 파일 단위이다. 데이터베이스 복구는 mount 단계에서만 수행되고 테이블스페이스와 데이터 파일 단위 복구는 mount와 open 단계에서 모두 수행할 수 있다.

- 백업된 데이터 파일 복원 (restore)

데이터 파일 복원은 전체 백업과 증분 백업을 통해 수행할 수 있는데 전체 백업을 이용한 데이터 파일 복원은 OS가 지원하는 파일 복사 명령어를 이용하여 사용자가 직접 수행하며 증분 백업을 이용한 복원은 GOLDBLOCKS가 지원하는 구문을 이용하여 수행한다. Open 단계에서 데이터 파일 복원을 수행하려면 해당 tablespace가 offline 상태여야 한다.

증분 백업을 이용한 데이터 파일 복원은 다음과 같이 수행된다.

```
gSQL> ALTER DATABASE RESTORE;
```

```
Database altered.
```

```
gSQL> ALTER DATABASE RESTORE TABLESPACE TEST_TBS;
```

```
Database altered.
```

- 데이터 파일 복원 후 수동 복구

데이터 파일을 복원한 후 복구를 위한 구문을 이용하여 다음과 같이 복구한다.

```
gSQL> ALTER DATABASE RECOVER;
```

```
Database altered.
```

```
gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;
```

```
Database altered.
```

불완전 복구

데이터베이스 운용 중에 발생하는 사용자 실수, 제어 파일 훼손, redo log file 및 archive log file의 훼손으로 인해 재시작 복구가 불가능할 뿐만 아니라 복구를 수행하더라도 데이터베이스의 일관성을 복구할 수 없는 경우에 특정 시점까지만 복구하는 불완전 복구 (incomplete recovery)도 지원한다.

불완전 복구가 필요한 경우는 다음과 같다.

제어 파일 훼손

제어 파일은 다중화되어 관리되므로 다중화된 파일이 모두 훼손되지만 않으면 훼손되지 않은 파일을 이용하여 복구할 수 있다. 그러나 모두 훼손된 경우에는 백업된 제어 파일을 이용하여 복구를 수행해야 하며 이 경우 제어 파일의 로그 정보가 변경될 수 있기 때문에 완전 복구를 수행할 수 없다. 따라서 복구가 가능한 시점까지 복구한다.

백업된 제어 파일 복원

제어 파일이 훼손된 경우 다중화로 관리 중인 다른 제어 파일을 복사하여 제어 파일을 최신 상태로 유지할 수 있지만 다중화된 제어 파일이 모두 훼손된 경우에는 백업된 제어 파일을 복원하여 복구를 수행한다. 백업된 제어 파일을 이용하여 복구하는 경우, 백업 이후에 변경된 로그 정보는 복구할 수 없다.

Redo Log File 훼손

GOLDILOCKS의 redo log file은 로그 그룹에서 여러 개의 로그 멤버로 관리되어 로그 파일의 훼손에 대비하지만, 특정 로그 그룹의 로그 멤버 전체가 훼손되면 redo log file을 이용하여 복구를 수행할 수 없어 훼손되지 않은 로그 파일까지 불완전 복구를 수행해야 한다.

Archive Log File 훼손

복구할 때 archive log file이 훼손된 경우에도 redo log file이 훼손된 경우와 동일하게 훼손되지 않은 로그 파일까지 불완전 복구를 수행해야 한다.

사용자 실수

중요한 테이블을 실수로 삭제하거나 테이블에 잘못된 데이터를 삽입/ 삭제/ 갱신하여 commit 한 경우에는 실수가 발생하기 이전으로 복구해야 한다.

GOLDBLOCKS 데이터베이스의 불완전 복구

GOLDBLOCKS 데이터베이스는 두 가지 불완전 복구를 지원한다. 하나는 운영자가 지정한 특정 시점까지 복구를 수행하는 것이고 다른 하나는 지정된 시스템과 운영자 사이에서 interactive하게 로그 파일 단위로 복구를 수행하는 것이다.

그리고 특정 시점까지의 불완전 복구는 특정 로그의 LSN, 특정 시간, 특정 SCN을 이용하여 복구가 필요한 특정 시점을 지정할 수 있다.

불완전 복구는 데이터베이스를 대상으로 mount 단계에서만 수행할 수 있다. 특정 테이블스페이스 단위의 불완전 복구는 데이터베이스의 일관성에 문제를 유발하므로 지원하지 않는다.

- 특정 LSN까지 불완전 복구

불완전 복구를 완료할 로그를 찾아 해당 로그의 LSN까지만 복구한다. 다음은 로그 LSN 1000까지만 불완전 복구하는 예이다.

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE 1000;
Database altered;
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE LSN 1000;
Database altered;
```

- 특정 SCN까지 불완전 복구

불완전 복구를 완료할 SCN을 찾아서 해당 로그의 SCN까지만 복구한다. 다음은 SCN 300까지만 불완전 복구하는 예이다.

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 300;
Database altered;
```

노트

SCN이 순서대로 로그에 쓰여지지 않기 때문에 SCN 300까지 수행했을 때 그 이상 복구될 수도 있다. 다음은 SCN이 역전된 경우에 복구하는 예이다.

로그: --- LSN 90 (SCN 3) -- LSN 91 (SCN 5) -- LSN 92 (SCN 4)

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 3;
→ LSN 90까지 복구된다.
```

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 4;
→ LSN 92까지 복구된다. (SCN 4가 있는 LSN 92까지 복구한다.)
```

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 5;
→ LSN 92까지 복구된다. (SCN 5까지 복구할 때 임의의 SCN 4와 SCN 5가 모두 복구된다.)
```

- 특정 time까지 불완전 복구

불완전 복구를 완료할 시간을 찾아서 특정 시간까지만 복구를 수행한다. 다음은 '2017-05-18 16:10:10.000000'까지 불완전 복구를 수행하는 예이다.

```
gSQL> ALTER DATABASE RECOVER UNTIL TIME '2017-05-18 16:10:10.000000';
Database altered;
```

- Interactive 불완전 복구

로그 파일이 훼손된 경우에는 훼손되기 직전 로그 파일까지만 복구한다. 이를 위해 GOLDILOCKS는 운영자에게 필요한 로그 파일을 제안하고 운영자는 GOLDILOCKS가 제안된 로그 파일을 이용하거나 새로운 로그 파일을 이용하여 불완전 복구를 수행한다.

다음은 GOLDILOCKS의 interactive 불완전 복구를 수행하는 예이다. 먼저 불완전 복구를 위한 BEGIN을 수행하면 GOLDILOCKS가 복구를 위해 필요한 로그 파일을 제안한다. 운영자는 GOLDILOCKS가 제안한 로그 파일을 그대로 사용하여 복구를 수행할 수도 있고 복구를 수행할 로그 파일을 직접 기술할 수도 있다.

```
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 139992)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_1.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 144143)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
```

불완전 복구 후 데이터베이스 재시작

불완전 복구를 완료한 후에는 정상적인 방법으로 데이터베이스를 재시작할 수 없다. 불완전 복구된 GOLDILOCKS 데이터베이스는 현재 redo log file과 관련이 없기 때문이다. 즉, 데이터베이스는 이전 시점이고 현재 redo log file은 그 이후에 발생한 로그의 기록이기 때문에 재시작하려면 redo log file을 리셋해야 한다. 따라서 불완전 복구 후에 GOLDILOCKS 데이터베이스를 재시작하려면 RESETLOGS 옵션을 사용한다.

```
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14083): must use RESETLOGS option for database open
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
```

System altered.

불완전 복구 시 주의사항

불완전 복구는 데이터베이스를 특정 시점까지만 복구하여 일관성을 가진 데이터베이스를 생성하는데 원하는 특정 시점을 한 번에 찾기는 쉽지 않다. 그리고 불완전 복구를 수행한 이후에는 redo log file이 모두 리셋되기 때문에 불완전 복구를 수행하기 전에 데이터베이스의 제어 파일, 데이터 파일, redo log file들을 모두 오프라인 백업한 후, 불완전 복구를 여러 번 수행하여 정확한 시점을 찾아야 한다.

Archive redo log file은 불완전 복구를 진행하는 동안에는 필요하지만 불완전 복구에 성공하면 새로운 데이터베이스가 되므로 이전 데이터베이스가 만든 archive redo log는 삭제한다.

복구 예

제어 파일 훼손

GOLDILOCKS 데이터베이스의 제어 파일은 데이터베이스의 물리적 구조와 데이터의 일관성에 대한 중요한 정보를 저장하고 있으므로 훼손되거나 실수로 삭제되는 등의 문제가 발생하면 데이터베이스를 운영할 수 없다.

이에 대비하여 GOLDILOCKS 데이터베이스는 최소 두 개에서 최대 여덟 개까지 제어 파일을 다중화하여 사용하기 때문에 유효한 제어 파일이 하나만 있으면 나머지 제어 파일을 복원한 후 데이터베이스를 재시작할 수 있다.

다중화된 유효한 제어 파일이 존재하는 경우

다중화된 제어파일 '/goldilocks_data/wal/control_1.ctl'이 훼손된 경우 다음과 같이 재시작에 실패한다.

```
gSQL> \STARTUP
ERR-HY000(14097): control file is corrupted - '/goldilocks_data/wal/control_1.ctl'
```

유효한 제어 파일 '/goldilocks_data/wal/control_0.ctl'을 '/goldilocks_data/wal/control_1.ctl'로 복사한 후 재시작에 실패했던 공유 메모리를 제거하고 재시작한다.

```
$ cp /goldilocks_data/wal/control_0.ctl /goldilocks_data/wal/control_1.ctl
gSQL> \SHUTDOWN
Shutdown success
gSQL> \STARTUP
Startup success
```

다중화된 모든 제어 파일이 훼손된 경우

다중화된 모든 제어 파일이 훼손된 경우 백업된 제어 파일을 이용하여 데이터베이스를 불완전 복구한 후에 재시작 할 수 있다. 백업된 제어 파일을 이용하여 제어 파일을 복구할 때 불완전 복구를 수행하는 이유는 제어 파일을 백업한 후에 데이터베이스의 물리적 구조가 변경될 수 있기 때문이다. 불완전 복구를 수행하더라도 archive log file과 redo log file이 모두 존재하기 때문에 GOLDILOCKS interactive 불완전 복구를 이용하여 운영자가 'CURRENT' 상태의 redo log file까지 수동으로 복구하여 완전 복구를 수행할 수 있다.

백업된 제어 파일은 OS의 복사 기능을 이용하여 다중화된 제어 파일로 모두 복사하거나 GOLDILOCKS 데이터베이스가 지원하는 제어 파일 복원 기능을 사용하여 다음과 같이 복원할 수도 있다. 제어 파일 복원은 GOLDILOCKS 데이터베이스 다단계 시작 중 NOMOUNT 단계에서만 수행할 수 있다.

```
gSQL> \STARTUP NOMOUNT
Startup success
gSQL> ALTER DATABASE RESTORE CONTROLFILE FROM '/goldilocks_data/backup/backup.ctl';
Database altered.
```

백업 제어 파일을 복원한 후 MOUNT 단계에서 다음과 같이 불완전 복구를 수행한다.

```
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137499)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_1.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137667)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_1_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_2.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137672)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered.
```

데이터 파일 훼손

데이터 파일이 훼손되었거나 삭제되었을 경우 백업된 데이터 파일을 이용하여 완전 복구를 수행한다. 전체 백업의 경우 백업 파일을 복사하여 복원하고, 증분 백업의 경우 GOLDILOCKS의 복원 기능을 이용하여 복원한다. 데이터 파일 복원과 복구는 데이터베이스 단위로 복원, 복구할 수도 있고 해당 데이터 파일의 테이블스페이스 단위로 복원, 복구할 수도 있다. 테이블스페이스 단위의 복원과 복구는 MOUNT, OPEN 단계에서 수행할 수 있는데 OPEN 단계에서 복원, 복구하기 위해서는 테이블스페이스가 반드시 OFFLINE 상태여야 한다.

- MOUNT 단계에서 전체 백업을 이용하여 훼손된 데이터 파일 복구

백업된 데이터 파일 (/goldilocks/backup/test.dbf)을 /goldilocks/db/test.dbf로 복사한 후 완전 복구를 수행한다.

```
$ cp /goldilocks/backup/test.dbf /goldilocks/db/test.dbf
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RECOVER;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

- OPEN 단계에서 전체 백업을 이용하여 훼손된 데이터 파일 복구

```
gSQL> SELECT IS_ONLINE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
IS_ONLINE
-----
FALSE
1 row selected.
$ cp /goldilocks/backup/test.dbf /goldilocks/db/test.dbf
gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;
Database altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

- MOUNT 단계에서 증분 백업을 이용하여 훼손된 데이터 파일 복구

```
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RESTORE;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

- OPEN 단계에서 증분 백업을 이용하여 훼손된 데이터 파일 복구

```
gSQL> SELECT IS_ONLINE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
IS_ONLINE
-----
FALSE
1 row selected.
gSQL> ALTER DATABASE RESTORE TABLESPACE TEST_TBS;
Database altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```


사용자 실수로 테이블 삭제 및 잘못된 삽입/ 삭제/ 갱신이 수행되었을 경우

TEST라는 테이블을 사용하던 중에 실수로 테이블을 삭제하였을 경우, GOLDILOCKS 데이터베이스는 테이블, 인덱스에 대한 DDL rollback 기능을 제공한다. 즉, 다음과 같이 테이블을 삭제하더라도 COMMIT 하지 않고 ROLLBACK하여 테이블 삭제를 취소할 수 있다.

```
gSQL> DROP TABLE TEST;
Table dropped.
gSQL> ROLLBACK;
Rollback complete.
gSQL> \DESC TEST
COLUMN_NAME TYPE          IS_NULLABLE
-----
I1          NUMBER(10,0) TRUE
I2          CHARACTER(10) TRUE
gSQL> DROP TABLE TEST;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> \DESC TEST
ERR-42000(16040): table or view does not exist :
SELECT * FROM TEST WHERE 1 = 0
          *
ERROR at line 1:
```

만약 테이블을 삭제한 후 COMMIT을 하였으면 ROLLBACK 할 수 없기 때문에 백업을 이용한 GOLDILOCKS 데이터베이스의 특정 시점까지의 불완전 복구 기능을 이용하여 테이블 삭제 이전까지만 복구하여 데이터베이스를 재시작한다.

불완전 복구를 위해 복구하려는 테이블은 삭제되기 이전 시점의 백업 파일을 이용하는데, 앞에서 설명하였듯이 여러 번의 과정을 거쳐 정확한 테이블 삭제 시점을 찾아야 하고 이 때 gdump 툴을 이용하여 로그 파일을 덤프해서 로그를 분석한다.

테이블이 삭제된 직후의 LSN이 1000이라고 가정할 경우, 다음과 같이 불완전 복구를 수행한다.

```
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE 1000;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered
gSQL> \DESC TEST
COLUMN_NAME TYPE          IS_NULLABLE
```

```

-----
I1          NUMBER(10,0)  TRUE
I2          CHARACTER(10) TRUE

```

로그 파일 훼손 (archive, redo log file)

- 복구 시 archive log file 훼손

데이터 파일이 훼손되어 백업 데이터 파일을 이용하여 복구를 수행하는 도중에 특정 archive log file이 훼손되어 복구를 끝까지 수행할 수 없다고 가정한다.

예를 들어, archive log file 'archive_0.log', 'archive_1.log', 'archive_2.log', 'archive_3.log'가 존재할 때 'archive_3.log'가 훼손되어 복구를 수행할 수 없는 경우에는 'archive_2.log' 까지만 불완전 복구를 수행한 후에 데이터베이스를 재시작한다.

```

gSQL> \STARTUP MOUNT
System altered
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 139992)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_3.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 194143)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered

```

- Redo log file 훼손

데이터베이스 서비스 중 장애가 발생하여 로그가 flush되던 CURRENT 로그 그룹이 훼손되었을 경우를 가정한다.

예를 들어, 다음과 같은 로그 그룹 상태에서 비정상 종료가 발생할 경우 로그 그룹 3, 0이 아직 아카이빙이 되지 않은 상태이기 때문에 수동 복구를 수행하고 불완전 복구를 완료한다.

표 7-5 Log group 상태

로그 그룹	로그 그룹 상태	로그 파일 시퀀스 번호	Prev last LSN
로그 그룹 0	ACTIVE	8	80000
로그 그룹 1	CURRENT	9	90000
로그 그룹 2	INACTIVE	6	60000
로그 그룹 3	ACTIVE	7	70000

```

gSQL> \STARTUP MOUNT
System altered
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 1000)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_7.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 70001)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_3_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_8.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 80001)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_0_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_9.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 90001)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered

```

데이터 파일이 로그보다 최신인 경우

GOLDILOCKS 데이터베이스에서 생성된 모든 테이블스페이스들의 데이터 파일들은 페이지로 구성되고, 각 페이지는 해당 페이지를 마지막으로 갱신한 트랜잭션이 기록한 로그의 LSN을 페이지 LSN으로 설정한다. 따라서 데이터 파일의 모든 페이지 LSN은 로그 그룹의 redo log file에 기록된 최신 로그의 LSN보다 작거나 같은 값을 가진다.

만약 데이터베이스를 재시작할 때 데이터 파일의 특정 페이지 LSN이 로그 그룹의 최신 로그 LSN보다 큰 값을 가지면 데이터베이스의 일관성이 깨어져 정상적인 서비스가 불가능하다. GOLDILOCKS 데이터베이스는 재시작할 때 데이터 파일과 로그를 체크하여 이런 비정상적인 상황이 발생하지 않도록 한다.

데이터 파일 중 최신 로그의 LSN 보다 큰 페이지 LSN 값을 가진 페이지가 존재하는 경우 다음과 같이 데이터베이스 재시작에 실패한다.

```

gSQL> \STARTUP
ERR-HY000(14114): exist inconsistent datafiles; need to restore more older backup datafiles or
more recent redo logfiles

```

이런 문제를 해결하기 위해서는 최신 로그 LSN보다 작은 페이지 LSN들로 구성된 백업 데이터 파일을 복원하거나 데이터 파일보다 큰 LSN의 로그가 기록된 로그 파일을 복원한 후 재시작해야 하며 복원할 대상 데이터 파일을 확인하기 위해서는 trace 파일을 확인해야 한다.

예를 들어 재시작에 실패했을 때 trace 파일에 다음과 같은 메시지가 출력되었다면 '/data/db/system_dic.dbf' 데이터 파일에 페이지 LSN이 '126787'인 페이지가 존재하고 이는 로그 파일의 최신 로그 LSN인 '126652'보다 큰 값이다. 따라서 재시작하기 위해서는 이전에 백업한 데이터 파일을 복원하거나 LSN이 '126787' 보다 크거나 같은 로그가 기록된 로그 파일을 복원하여 재시작해야 한다. 또한, 여러 개의 데이터 파일의 페이지가 로그 파일의 LSN보다 큰 LSN값을 가질 경우, 그 중 최대값보다 큰 로그 파일을 복원해야만 재시작과 서비스를 수행할 수 있다.

```
[2016-01-15 12:41:14.045679 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126787' of datafile '/data/db/system_dict.dbf' is more recent
than the latest redo log lsn '126652'.
[2016-01-15 12:41:14.045705 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126830' of datafile '/data/db/system_undo.dbf' is more recent
than the latest redo log lsn '126652'.
[2016-01-15 12:41:14.045729 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126829' of datafile '/data/db/test_log.dbf' is more recent than
the latest redo log lsn '126652'.
```

클러스터 환경에서 In Doubt 트랜잭션 복구

클러스터 환경의 트랜잭션은 두 개 이상의 cluster group에서 수행되는 global 트랜잭션, 하나의 cluster group에서 수행되는 domain 트랜잭션, 하나의 cluster member에서 수행되는 local 트랜잭션으로 구분된다.

Local 트랜잭션의 경우 local member의 로그만 이용하여 복구를 수행한다. Domain 트랜잭션의 경우 local member의 로그를 이용하여 복구하는데 트랜잭션 수행이 완료되지 않은 상태에서 비정상 종료된 경우에는 rollback하고 필요할 경우 rebalance를 통해 group 내의 member와 동기화한다.

Global 트랜잭션의 경우 2 phase commit protocol을 이용하여 commit 한다. GOLDDILOCKS의 global transaction에 대한 2 phase commit은 다음과 같이 수행된다.

- PREPARE phase
 - Driver member에서 PREPARE message를 모든 member들에 송신하고 응답 메시지를 기다린다.
 - 모든 member들로부터 PREPARE의 응답 메시지를 받으면 COMMIT phase로 넘어가고 그렇지 않고 하나의 member라도 실패하거나 응답이 없으면 rollback 한다.
- COMMIT phase
 - Driver member에서 모든 member들에게 COMMIT message를 송신하고 응답 메시지를 기다린다.
 - 실패하는 member가 있어도 commit이 수행된다.

COMMIT phase에서 commit log를 기록하지 못하고 비정상 종료된 member는 재시작될 때 'PREPARE' 상태의 global 트랜잭션이 commit 되었는지 rollback 되었는지 알 수 없기 때문에 다른 member들로부터 상태를 구해야 한다.

이를 위해 GOLDDILOCKS는 commit된 global transaction 정보를 MEM_TRANS_TBS에 트랜잭션 레코드 형태로 기록하고 해당 레코드에 새로운 레코드가 기록될 때 필요에 따라 이전 레코드를 commit.log에 기록한다. 이후 비정상 종료된 member는 log를 이용하여 재시작 복구 또는 수동 복구를 수행하고 'PREPARE' 상태인 in doubt 트랜잭

션이 남아 있으면 서비스 중인 member들로부터 해당 트랜잭션의 COMMIT/ ROLLBACK 정보를 얻어 복구한다.

8.

GOLDILOCKS 데이터베이스 이중화

8.1 이중화 소개

본 장에서는 CYCLONE과 LOGMIRROR에 대해 설명한다.

GOLDDILOCKS의 이중화는 CDC 방식을 사용하여 transaction을 이중화하는 CYCLONE과 원본 데이터베이스의 redo log file을 이중화하는 LOGMIRROR를 지원한다.

표 8-1 이중화 tool 소개

이름	이중화 대상	설명
CYCLONE	Transaction	CDC 방식을 사용하여 master에 반영된 transaction을 이중화하여 slave에 반영한다.
LOGMIRROR	Redo log file	Master 데이터베이스의 redo log file을 slave에 동일하게 이중화한다.

- CYCLONE
 - Change Data Capture (CDC) 방식으로 원본 데이터베이스의 redo log file을 분석하고 가공하여 원격 데이터베이스에 적용한다.
 - 데이터베이스의 redo log file에 저장된 내용을 분석하기 때문에 비동기 (async) 방식만 지원한다.
- LOGMIRROR
 - 원본 데이터베이스에 저장되는 redo log file을 원격지에 이중화한다.
 - 비동기 방식으로 동작하는 CYCLONE의 데이터 손실을 방지하기 위해 수행된다.

8.2 운영 방법

CYCLONE

일반적인 운영 방법과 옵션에 대한 자세한 내용은 CYCLONE을 참조한다.

노드 추가와 삭제

CYCLONE은 group 단위로 수행되며 이는 이중화 노드와 동일하다. 만약 노드를 추가할 경우에는 group을 추가하고 삭제할 경우에는 group을 삭제하도록 한다.

노드 추가 예

- 추가하려는 group 2를 master 환경 파일에 기록한다.
 - 그룹마다 고유의 PORT를 설정해야 한다.
 - 기본 master 환경 파일: \$GOLDILOCKS_DATA/conf/cyclone.master.conf
 - 다음은 기존에 운영 중인 group 1 노드이다.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        testTable1,
        testTable2
    )
}
```

- 다음은 추가하려는 group 2 노드이다.

```
GROUP_NAME = Group2
{
    PORT = 21103
    CAPTURE_TABLE =
    (
        testTable5,
```

```

        testTable6
    )
}

```

- 추가하려는 group 2를 slave 환경 파일에 기록한다.
 - 기존에 master에 추가한 group 2의 PORT와 동일해야 한다.
 - 기본 slave 환경 파일: \$GOLDBLOCKS_DATA/conf/cyclone.slave.conf
 - 다음은 기존에 운영 중인 group 1 노드이다.

```

...
...
GROUP_NAME = Group1
{
    PORT = 21102
    APPLY_TABLE =
    (
        testTable1 To testTable3,
        testTable2 To testTable4
    )
}

```

- 다음은 추가하려는 group 2 노드이다.

```

{
    PORT = 21103
    APPLY_TABLE =
    (
        testTable5 To testTable7,
        testTable6 To testTable8
    )
}

```

- 추가된 group 2 노드를 master 장비에서 실행하고 확인한다.

```

prompt> cyclone --master --start --group Group2
[GROUP2] Startup done as Master.
prompt> cyclone --master --status
=====
|          CYCLONE STATUS - MASTER          |
=====
GROUP1 Running...

```

```
GROUP2 Running...
```

- 추가된 group 2 노드를 slave 장비에서 실행하고 확인한다.

```
prompt> cyclone --slave --start --group Group2
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====
CYCLONE STATUS - SLAVE
=====
GROUP1 Running...
GROUP2 Running...
```

노드 삭제 예

- 삭제하려는 group 2 노드를 slave 장비에서 종료한다.

```
prompt> cyclone --slave --stop --group Group2
stop done.
prompt> cyclone --slave --status
=====
CYCLONE STATUS - SLAVE
=====
GROUP1 Running...
```

- 삭제하려는 group 2 노드를 master 장비에서 종료한다.

```
prompt> cyclone --master --stop --group Group2
stop done.
prompt> cyclone --master --status
=====
CYCLONE STATUS - MASTER
=====
GROUP1 Running...
```

- 삭제하려는 노드인 group 2를 master 환경 파일에서 제거한다.
 - 기본 master 환경 파일: \$GOLDILOCKS_DATA/conf/cyclone.master.conf

- 다음은 기존에 운영 중인 group 1 노드이다.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        testTable1,
        testTable2
    )
}
```

- Group 2 노드를 삭제한다.

```
GROUP_NAME = Group2
{
—— PORT = 21103
—— CAPTURE_TABLE =
—— (
—— testTable5,
—— testTable6
—— )
}
```

- 삭제하려는 노드인 group 2를 slave 환경 파일에서 제거한다.
 - 기본 slave 환경 파일: \$GOLDDILOCKS_DATA/conf/cyclone.slave.conf
 - 다음은 기존에 운영 중인 group1 노드이다.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    APPLY_TABLE =
    (
        testTable1 To testTable3,
        testTable2 To testTable4
    )
}
```

- Group 2 노드를 삭제한다.

```
GROUP_NAME = Group2
{
  PORT = 21103
  APPLY_TABLE =
  (
    testTable5 To testTable7,
    testTable6 To testTable8
  )
}
```

이중화 초기화

기존에 이중화를 수행했던 노드나 그룹에 포함된 table이 DDL 수행 등의 이유로 수행을 give up 했을 경우 이중화를 초기화할 수 있다. 특정 노드 또는 전체 노드를 초기화할 수 있다.

이중화 초기화는 slave에서 수행하고 있는 이중화를 --reset 옵션을 사용하여 재시작하는 방식으로 수행한다. Master에서는 특별한 작업을 수행하지 않아도 된다.

특정 노드의 이중화 초기화 예

- 초기화하려는 group 2 노드를 slave 장비에서 종료한다.

```
prompt> cyclone --slave --stop --group Group2
stop done.
prompt> cyclone --slave --status
=====
CYCLONE STATUS - SLAVE
=====
GROUP1 Running...
-----
```

- --reset 옵션을 사용하여 group 2 노드를 slave 장비에서 재시작한다.
 - Master 장비에서는 특정 작업을 수행하지 않아도 된다.
 - 현재 시점부터 이중화가 재시작된다.

```
prompt> cyclone --slave --start --reset --group Group2
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====
CYCLONE STATUS - SLAVE
```

```
=====
GROUP1 Running...
GROUP2 Running...
-----
```

전체 노드의 이중화 초기화 예

- Slave 장비에서 운영 중인 cyclone을 모두 종료한다.

```
prompt> cyclone --slave --stop
```

- --reset 옵션을 사용하여 slave 장비에서 재시작한다.
 - Master 장비에서는 특정 작업을 수행하지 않아도 된다.
 - 현재 시점부터 이중화가 시작된다.

```
prompt> cyclone --slave --start --reset
[GROUP1] Startup done as Slave.
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====
CYCLONE STATUS - SLAVE
=====
GROUP1 Running...
GROUP2 Running...
-----
```

LOGMIRROR

일반적인 운영 방법과 옵션에 대한 자세한 내용은 **LOGMIRROR**를 참조한다.

LOGMIRROR 상태 확인

LOGMIRROR와 연동할 경우 GOLDDILOCKS는 LOGMIRROR의 응답 대기 과정을 포함한다. 만약 LOGMIRROR가 응답을 기다리고 있다면 GOLDDILOCKS 역시 blocked 된 상태로 대기한다. 이런 상태는 v\$system_stat에서 확인할 수 있다.

```
gSQL> SELECT * FROM V$SYSTEM_STAT WHERE STAT_NAME='LOG_MIRROR_SYNC_STATE';
STAT_NAME          STAT_VALUE COMMENTS
-----
LOG_MIRROR_SYNC_STATE      0 logmirror sync state( 0 : sync, 1 : blocked )
```

1 row selected.

STAT_VALUE가 '0'이면 대기 상태가 아닌 일반적인 상황이고 '1'이면 응답을 기다리는 blocked 상태이다. 응답을 기다리는 상황에서 GOLDILOCKS의 서비스를 재개할 경우, LOG_MIRROR_TIMEOUT을 변경하여 LOGMIRROR 서비스를 중단할 수 있다.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_TIMEOUT = 20;
System altered.
```

이중화 초기화

LogMirror의 이중화 초기화는 수동으로 수행해야 한다. 이는 사용자가 옵션을 잘못 사용하여 데이터가 삭제되거나 복구 불능 상태가 되지 않도록 하기 위함이다.

노트

LOGMIRROR의 slave에는 control 파일과 redo log file이 저장되고 control 파일에는 운영에 필요한 정보가 저장 및 갱신된다.

이중화 초기화 예

- Slave 장비에서 운영 중인 LOGMIRROR를 종료한다.

```
prompt> logmirror --slave --stop
stop done.
```

- Master 장비에서 운영 중인 LOGMIRROR를 종료한다.

```
prompt> logmirror --master --stop
stop done.
```

- Slave 장비에서 이중화된 컨트롤 파일과 redo log file을 삭제한다.
 - 경로는 slave 환경 파일에 기술된 'LOG_PATH' 옵션을 참조한다.
 - 기본 LOGMIRROR slave 환경 파일: \$GOLDILOCKS_DATA/conf/logmirror.slave.conf

8.3 Trace Log

Trace log에 대한 자세한 정보는 다음과 같다.

이름	구분	파일 이름
CYCLONE	Master	cyclone_master_GROUP_NAME.trc
	Slave	cyclone_slave_GROUP_NAME.trc
LOGMIRROR	Master	LogMirror_master.trc
	Slave	LogMirror_slave.trc

CYCLONE 에러 메시지와 해결 방법

다음은 CYCLONE의 에러 메시지와 해결 방법이다.

에러 메시지	해결 방법
Service is not available	GOLDBLOCKS가 정상적으로 실행되어 있는지 확인한다.
table does not exist	환경 파일에 기술한 table 이름을 확인한다.
schema does not exist	환경 파일에 기술한 schema 이름을 확인한다.
previously added. Maybe duplicated	환경 파일에 기술한 table이 중복되었는지 확인한다.
table must have a primary key	환경 파일에 기술한 table에 primary key가 있는지 확인한다.
internal error occurred.	상세 에러를 확인한다.
table must set supplemental log	GOLDBLOCKS가 supplemental logging 되어 있는지 확인한다.
group XXX is already running	해당 group이 이미 실행되어 있는지 확인한다.
GOLDBLOCKS_DATA system environment is invalid	GOLDBLOCKS_DATA 환경 변수가 설정되어 있는지 확인한다.
log file reused or invalid. restart cyclone with '--reset' option	Redo log file이 재사용되었거나 아카이빙 되어있는 redo log file이 없다. Cyclone을 초기화하여 재시작해야 한다.
fail to analyze flow	비정상적인 redo log를 분석할 경우에 발생하는 에러이며 master와 slave의 GOLDBLOCKS 릴리즈 버전이 동일한지 확인한다.
Communication link failure	네트워크 상태를 확인한다. Cyclone을 재시작한다.
Master disconnect abnormally	네트워크 상태를 확인한다. Cyclone을 재시작한다.
Protocol error occurred	상세 에러를 확인한다.
Already slave connected	Slave가 이미 실행되어 있는지 확인한다.
Invalid group name	시작/ 종료할 때 명세한 그룹 이름을 확인한다. 환경 파일에 기술한 그룹 이름과 동일해야 한다.
Invalid capture information	기존의 운영 정보가 비정상적인 경우이며 cyclone을 초기화한 후에 재시작한다.
Redo log file read timeout	아카이빙 된 redo log file이 정상적으로 존재하는지 확인한다.
Invalid archive log file	아카이빙 된 해당 redo log file이 정상적이지 않은 경우이다. Cyclone을 초기화

에러 메시지	해결 방법
	고 재시작해야 한다.
Fail to write file	디스크의 여유 용량을 확인한 후에 cyclone을 재시작한다.
Invalid Meta File	Cyclone에서 관리하는 메타파일이 손상되었을 때 발생하는 에러이며 cyclone을 초기화하고 재시작해야 한다.
Redo log file does not exist	Master로 동작하는 GOLDDILOCKS가 정상적으로 운영되고 있는지 확인한다.
[APPLIER-INSERT] XXX	해당 이유로 인하여 INSERT에 실패했다.
[APPLIER-DELETE] XXX	해당 이유로 인하여 DELETE에 실패했다.
[APPLIER-UPDATE] XXX	해당 이유로 인하여 UPDATE에 실패했다.

LOGMIRROR 에러 메시지와 해결 방법

다음은 LOGMIRROR의 에러 메시지와 해결 방법이다.

에러 메시지	해결 방법
Service is not available	GOLDDILOCKS가 정상적으로 실행되어 있는지 확인한다.
Invalid Protocol value	해당 상세 메시지를 확인한다.
file does not exist	해당 파일이 정상적으로 존재하는지 확인한다.
invalid Control file	컨트롤 파일이 손상된 상태이다. LOGMIRROR를 초기화하고 재시작해야 한다.
Communication link failure	네트워크 상태를 확인한다. LOGMIRROR를 재시작한다.
GOLDDILOCKS_DATA system environment is invalid	GOLDDILOCKS_DATA 환경 변수가 설정되어 있는지 확인한다.
There is no Shared Memory Area for LogMirror	Master로 운영 중인 GOLDDILOCKS의 프로퍼티 중에 LOG_MIRROR_MODE가 정상적으로 enable 되었는지 확인한다.
Master disconnect abnormally	네트워크 상태를 확인한다. LOGMIRROR를 재시작해야 한다.
Invalid Log File	해당 파일이 정상적으로 존재하는지 확인한다.
Connection Information does not exist	환경 파일에 있는 GOLDDILOCKS 접속 정보가 정상적인지 확인한다.
Archive Log File does not exist	Master로 운영 중인 GOLDDILOCKS의 ARCHIVELOG_MODE가 정상적으로 설정되어 있는지 확인한다.

9.

Database Information

9.1 DICTIONARY_SCHEMA

DICTIONARY_SCHEMA 스키마는 시스템 내의 SQL 객체와 이와 관련된 정보를 얻기 위한 view나 테이블을 포함하고 있다.

노트

DICTIONARY_SCHEMA의 view와 테이블들은 open 단계부터 조회할 수 있다.

해당 view들을 사용하려면 다음과 같이 DictionarySchema.sql을 실행해야 한다.

- Standalone의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/DictionarySchema.sql
```

- Cluster의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
```

View나 테이블의 이름에 따라 다음과 같은 정보를 얻을 수 있다.

- ALL 계열 view
 - ALL_로 시작하는 이름을 가진 view
 - 현재 사용자가 접근 가능한 객체에 대한 정보
- DBA 계열 view
 - DBA_로 시작하는 이름을 가진 view
 - DBA 권한 (ACCESS CONTROL ON DATABASE)을 가지고 있는 현재 사용자의 모든 객체에 대한 정보
- USER 계열 view
 - USER_로 시작하는 이름을 가진 view
 - 현재 사용자가 소유한 객체에 대한 정보

ALL 계열 View

현재 사용자가 접근 가능한 객체에 대한 정보를 얻을 수 있다.

ALL_ALL_TABLES

ALL_ALL_TABLES describes the object tables and relational tables accessible to the current user.

표 9-1 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table

Column name	Data type	Description
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table
CHAIN_CNT	NUMBER	reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	reserved If an object table, owner of the type from which the table is created
TABLE_TYPE	VARCHAR(128)	reserved If an object table, type of the table
TEMPORARY	VARCHAR(1)	reserved Indicates whether the table is temporary (Y) or not (N)

Column name	Data type	Description
SECONDARY	VARCHAR(1)	reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	reserved Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

ALL_ARGUMENTS

ALL_ARGUMENTS lists all arguments of functions, procedures.

표 9-2 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or package
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in package
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data Type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

ALL_CATALOG

ALL_CATALOG displays the tables, views, synonyms, and sequences accessible to the current user.

표 9-3 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

ALL_CLUSTER_TABLES

ALL_CLUSTER_TABLES describes all cluster tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-4 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST S HARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is nul l)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)

ALL_COL_COMMENTS

ALL_COL_COMMENTS displays comments on the columns of the tables and views accessible to the current user.

표 9-5 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

ALL_COL_PRIVS

ALL_COL_PRIVS describes the object grants, for which the current user is the object owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-6 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_COL_PRIVS_MADE

ALL_COL_PRIVS_MADE describes the column object grants for which the current user is the object owner or grantor.

표 9-7 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_COL_PRIVS_RECD

ALL_COL_PRIVS_RECD describes the column object grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-8 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_CONSTRAINTS

ALL_CONSTRAINTS describes constraint definitions on tables accessible to the current user.

표 9-9 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential integrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
		reserved

Column name	Data type	Description
RELY	VARCHAR(32)	When NOT VALIDATED, indicates whether the constraint is to be taken into account for query rewrite (RELY) or not (NULL)
LAST_CHANGE	TIMESTAMP(2) WITHOUT TIME ZONE	When the constraint was last enabled or disabled
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	Indicates whether the constraint depends on a view (DEPEND ON VIEW) or not (NULL)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

ALL_CONS_COLUMNS

ALL_CONS_COLUMNS describes columns that are accessible to the current user and that are specified in constraints.

표 9-10 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

ALL_DB_PRIVS

ALL_DB_PRIVS describes the database grants, for which the current user is the grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-11 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_DB_PRIVS_MADE

ALL_DB_PRIVS_MADE describes the database grants for which the current user is the grantor.

표 9-12 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_DB_PRIVS_RECD

ALL_DB_PRIVS_RECD describes the database grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-13 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_DEPENDENCIES

ALL_DEPENDENCIES describes dependencies between objects accessible to the current user

표 9-14 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

ALL_GLOBAL_SECONDARY_INDEXES

ALL_GLOBAL_SECONDARY_INDEXES describes the global secondary indexes on the tables accessible to the current user.

노트

Cluster에서만 사용할 수 있다.

표 9-15 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index

ALL_GSI_PLACE

ALL_GSI_PLACE describes node placement of all global secondary indexes on the tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-16 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

ALL_INDEXES

ALL_INDEXES describes the indexes on the tables accessible to the current user.

표 9-17 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NON UNIQUE)
COMPRESSION	VARCHAR(32)	reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to this segment

Column name	Data type	Description
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
BLEVEL	NUMBER	reserved B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	reserved Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	reserved Average number of leaf blocks in which each distinct value in the index appears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	reserved Indicates the amount of order of the rows in the table based on the values of the index
STATUS	VARCHAR(32)	Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the index, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the indexes to be scanned, or DEFAULT
PARTITIONED	VARCHAR(3)	reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	Indicates whether the index is a secondary object created by the method of the Data Cartridge (Y) or not (N)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for index blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for index blocks
		reserved

Column name	Data type	Description
CELL_FLASH_CACHE	VARCHAR(32)	Cell flash cache hint to be used for index blocks
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
PCT_DIRECT_ACCESS	NUMBER	reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned indexes, indicates whether statistics were collected by analyzing the index as a whole (YES) or were estimated from statistics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR(3)	reserved Indicates whether redundant primary key columns are eliminated from secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	reserved Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGEMENT	VARCHAR(32)	reserved If this is a domain index, indicates whether the domain index is system-managed (SYSTEM_MANAGED) or user-managed (USER_MANAGED)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the index segment has been created (YES) or not (NO)

Column name	Data type	Description
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

ALL_IND_COLUMNS

ALL_IND_COLUMNS describes the columns of indexes on all tables accessible to the current user.

표 9-18 Column 정보

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the table or cluster
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

ALL_IND_PLACE

ALL_IND_PLACE describes node placement of the indexes on the tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-19 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(2) WITH OUT TIME ZONE	(deprecated)

ALL_NONSCHEMA_COMMENTS

ALL_NONSCHEMA_COMMENTS displays comments on all non-schema objects (database, authorizations, schemas, tablespaces) accessible to the current user.

표 9-20 Column 정보

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the non-schema object
OBJECT_TYPE	VARCHAR(32)	Type of the non-schema object: DATABASE, AUTHORIZATION, SCH EMA, TABLESPACE
COMMENTS	VARCHAR(1024)	Comments of the non-schema object

ALL_OBJECTS

ALL_OBJECTS describes all objects accessible to the current user.

표 9-21 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(2) WITH HOUR TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(2) WITH HOUR TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	* reserved Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether this is a secondary object created by the ODCIIndexCreate method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	• reserved Name of the edition in which the object is actual

ALL_PROCEEDURES

ALL_PROCEEDURES lists all function, procedures or package

표 9-22 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or package
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or package
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in package
OBJECT_ID	NUMBER	ID of a function, procedures or package
SUBPROGRAM_ID	NUMBER	ID of procedures in package
OVERLOAD	VARCHAR(32)	ID of overloading procedure in package
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggregate function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented using the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; otherwise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

ALL_PROC_PRIVS

ALL_PROC_PRIVS describes the procedure grants, for which the current user is the procedure owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-23 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_PROC_PRIVS_MADE

ALL_PROC_PRIVS_MADE describes the procedure grants for which the current user is the procedure owner or grantor.

표 9-24 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_PROC_PRIVS_RECD

ALL_PROC_PRIVS_RECD describes the procedure grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-25 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMAS

Identify the schemata in a catalog that are owned by given user or accessible to given user or role.

표 9-26 Column 정보

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

ALL_SCHEMA_PATH

ALL_SCHEMA_PATH describes the schema search order of the current user and PUBLIC, for naming resolution of unqualified SQL schema objects.

표 9-27 Column 정보

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the authorization
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the authorization

ALL_SCHEMA_PRIVS

ALL_SCHEMA_PRIVS describes the schema grants, for which the current user is the schema owner, grant or, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-28 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMA_PRIVS_MADE

ALL_SCHEMA_PRIVS_MADE describes the schema grants, for which the current user is the grantor.

표 9-29 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMA_PRIVS_RECD

ALL_SCHEMA_PRIVS_RECD describes the schema grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-30 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQUENCES

ALL_SEQUENCES describes all sequences accessible to the current user.

표 9-31 Column 정보

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

ALL_SEQ_PRIVS

ALL_SEQ_PRIVS describes the sequence grants, for which the current user is the sequence owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-32 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQ_PRIVS_MADE

ALL_SEQ_PRIVS_MADE describes the sequence grants for which the current user is the sequence owner or grantor.

표 9-33 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQ_PRIVS_RECD

ALL_SEQ_PRIVS_RECD describes the sequence grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-34 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SHARD_KEY_COLUMNS

ALL_SHARD_KEY_COLUMNS describes shard key columns of all shareded tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-35 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

ALL_SOURCE

ALL_SOURCE describes the text source of the stored objects accessible to the current user.

표 9-36 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the stored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

ALL_SYNONYMS

ALL_SYNONYMS describes all synonyms.

표 9-37 Column 정보

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

ALL_TABLES

ALL_TABLES describes the relational tables accessible to the current user.

표 9-38 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table
		reserved

Column name	Data type	Description
CHAIN_CNT	NUMBER	Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	reserved If a partitioned table, indicates whether row movement is enabled (

Column name	Data type	Description
		ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	reserved Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	reserved Result cache mode annotation for the table: the value in (NULL, DEFAULT, FORCE, MANUAL)

ALL_TAB_COLS

ALL_TAB_COLS describes the columns (including hidden columns) of the tables, views, and clusters accessible to the current user.

표 9-39 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	reserved Declaration length of the character type column
		reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	reserved Indicates existence/type of histogram
QUALIFIED_COLUMN_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

ALL_TAB_COLUMNS

ALL_TAB_COLUMNS describes the columns of the tables, views, and clusters accessible to the current user.

표 9-40 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
		reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

ALL_TAB_COMMENTS

ALL_TAB_COMMENTS displays comments on the tables and views accessible to the current user.

표 9-41 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

ALL_TAB_IDENTITY_COLS

ALL_TAB_IDENTITY_COLS describes all table identity columns.

표 9-42 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

ALL_TAB_PLACE

ALL_TAB_PLACE describes node placement of all cluster tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-43 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

ALL_TAB_SHARDS

ALL_TAB_SHARDS describes shard information of sharded tables accessible to the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-44 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed

ALL_TAB_PRIVS

ALL_TAB_PRIVS describes the object grants, for which the current user is the object owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-45 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

ALL_TAB_PRIVS_MADE

ALL_TAB_PRIVS_MADE describes the object grants for which the current user is the object owner or grantor.

표 9-46 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

ALL_TAB_PRIVS_RECD

ALL_TAB_PRIVS_RECD describes object grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-47 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

ALL_TBS_PRIVS

ALL_TBS_PRIVS describes the tablespace grants, for which the current user is the grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-48 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_TBS_PRIVS_MADE

ALL_TBS_PRIVS_MADE describes the tablespace grants for which the current user is the grantor.

표 9-49 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_TBS_PRIVS_RECD

ALL_TBS_PRIVS_RECD describes the tablespace grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-50 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_USERS

ALL_USERS lists all users of the database visible to the current user.

표 9-51 Column 정보

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
CREATED	TIMESTAMP(2) WITHOUT TIME ZONE	User creation timestamp

ALL_VIEWS

ALL_VIEWS describes the views accessible to the current user.

표 9-52 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the view
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	reserved Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

DBA 계열 View

DBA 권한 (ACCESS CONTROL ON DATABASE)을 가지고 있는 현재 사용자의 모든 객체에 대한 정보를 얻을 수 있다.

DBA_ALL_TABLES

DBA_ALL_TABLES describes all object tables and relational tables in the database.

표 9-53 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)

Column name	Data type	Description
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	reserved Number of used blocks in the table
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table
CHAIN_CNT	NUMBER	reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	reserved If an object table, owner of the type from which the table is created
		reserved

Column name	Data type	Description
TABLE_TYPE	VARCHAR(128)	If an object table, type of the table
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
		reserved

Column name	Data type	Description
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

DBA_ARGUMENTS

DBA_ARGUMENTS lists all arguments of functions, procedures.

표 9-54 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or package
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in package
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data Type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

DBA_CATALOG

DBA_CATALOG lists all tables, views, synonyms, and sequences in the database.

표 9-55 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

DBA_CLUSTER

DBA_CLUSTER describes all cluster members in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-56 Column 정보

Column name	Data type	Description
GROUP_ID	NUMBER	Group identifier of the cluster member
GROUP_NAME	VARCHAR(128)	Group name of the cluster member
MEMBER_ID	NUMBER	Member identifier of the cluster member
MEMBER_NAME	VARCHAR(128)	Member name of the cluster member
MEMBER_HOST	VARCHAR(128)	Host address of the cluster member
MEMBER_PORT	NUMBER	Port number of the cluster member

DBA_CLUSTER_COMMENTS

DBA_CLUSTER_COMMENTS displays comments on the cluster objects in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-57 Column 정보

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the cluster object
OBJECT_TYPE	VARCHAR(32)	Type of the cluster object: CLUSTER GROUP, CLUSTER MEMBER
COMMENTS	VARCHAR(1024)	Comment on the cluster object

DBA_CLUSTER_TABLES

DBA_CLUSTER_TABLES describes all cluster tables in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-58 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST S HARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is nul l)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)

DBA_COL_COMMENTS

DBA_COL_COMMENTS displays comments on the columns of all tables and views in the database.

표 9-59 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

DBA_COL_PRIVS

DBA_COL_PRIVS describes all column object grants in the database.

표 9-60 Column 정보

Column name	Data type	Description
GRANTOR	CHARACTER VARYING(128)	Name of the user who performed the grant
GRANTEE	CHARACTER VARYING(128)	Name of the user or role to whom access was granted
OWNER	CHARACTER VARYING(128)	Owner of the object
TABLE_SCHEMA	CHARACTER VARYING(128)	Schema of the object
TABLE_NAME	CHARACTER VARYING(128)	Name of the object
COLUMN_NAME	CHARACTER VARYING(128)	Name of the column
PRIVILEGE	CHARACTER VARYING(32)	Privilege on the column
GRANTABLE	CHARACTER VARYING(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_CONSTRAINTS

DBA_CONSTRAINTS describes all constraint definitions on all tables in the database.

표 9-61 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential integrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
RELY	VARCHAR(32)	reserved When NOT VALIDATED, indicates whether the constraint is to be taken into account for query rewrite (RELY) or not (NULL)
	TIMESTAMP(2) WITH	

Column name	Data type	Description
LAST_CHANGE	OUT TIME ZONE	When the constraint was last enabled or disabled
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	reserved Indicates whether the constraint depends on a view (DEPEND ON VIEW) or not (NULL)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

DBA_CONS_COLUMNS

DBA_CONS_COLUMNS describes all columns in the database that are specified in constraints.

표 9-62 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

DBA_DB_PRIVS

DBA_DB_PRIVS describes all database grants in the database.

표 9-63 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_DEPENDENCIES

DBA_DEPENDENCIES describes all dependencies between objects in the database

표 9-64 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

DBA_EXTENTS

DBA_EXTENTS describes the extents comprising the segments in all tablespaces in the database.

표 9-65 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the segment associated with the extent
SEGMENT_SCHEMA	VARCHAR(128)	Schema of the segment associated with the extent
SEGMENT_NAME	VARCHAR(128)	Name of the segment associated with the extent
PARTITION_NAME	VARCHAR(128)	reserved Object Partition Name (Set to NULL for non-partitioned objects)
SEGMENT_TYPE	VARCHAR(32)	Type of the segment: TABLE, INDEX
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	reserved Extent number in the segment
FILE_ID	NUMBER	reserved File identifier number of the file containing the extent
BLOCK_ID	NUMBER	reserved Starting block number of the extent
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in Oracle blocks
RELATIVE_FNO	NUMBER	reserved Relative file number of the first extent block

DBA_GLOBAL_SECONDARY_INDEXES

DBA_GLOBAL_SECONDARY_INDEXES describes all global secondary indexes in the database.

노트

Cluster에서만 사용할 수 있다.

표 9-66 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index

DBA_GSI_PLACE

DBA_GSI_PLACE describes node placement of all global secondary indexes in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-67 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

DBA_INDEXES

DBA_INDEXES describes all indexes in the database.

표 9-68 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NON UNIQUE)
COMPRESSION	VARCHAR(32)	reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to this segment

Column name	Data type	Description
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
BLEVEL	NUMBER	reserved B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	reserved Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	reserved Average number of leaf blocks in which each distinct value in the index appears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	reserved Indicates the amount of order of the rows in the table based on the values of the index
STATUS	VARCHAR(32)	reserved Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the index, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the indexes to be scanned, or DEFAULT
PARTITIONED	VARCHAR(3)	reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the index is a secondary object created by the method of the Data Cartridge (Y) or not (N)
		reserved

Column name	Data type	Description
BUFFER_POOL	VARCHAR(32)	Buffer pool to be used for index blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for index blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for index blocks
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
PCT_DIRECT_ACCESS	NUMBER	reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned indexes, indicates whether statistics were collected by analyzing the index as a whole (YES) or were estimated from statistics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR(3)	reserved Indicates whether redundant primary key columns are eliminated from secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	reserved Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGE		reserved If this is a domain index, indicates whether the domain index is syst

Column name	Data type	Description
MENT	VARCHAR(32)	em-managed (SYSTEM_MANAGED) or user-managed (USER_MANAGED)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the index segment has been created (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

DBA_IND_COLUMNS

DBA_IND_COLUMNS describes the columns of all the indexes on all tables and clusters in the database.

표 9-69 Column 정보

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the table or cluster
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	reserved Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

DBA_IND_PLACE

DBA_IND_PLACE describes node placement of all indexes in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-70 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(2) WITH OUT TIME ZONE	(deprecated)

DBA_NONSCHEMA_COMMENTS

DBA_NONSCHEMA_COMMENTS displays comments on all non-schema objects (database, authorizations, schemas, tablespaces).

표 9-71 Column 정보

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the non-schema object
OBJECT_TYPE	VARCHAR(32)	Type of the non-schema object: DATABASE, PROFILE, AUTHORIZATI ON, SCHEMA, TABLESPACE
COMMENTS	VARCHAR(1024)	Comments of the non-schema object

DBA_OBJECTS

DBA_OBJECTS describes all objects in the database.

표 9-72 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	Indicates whether this is a secondary object created by the ODCIIndexCreate method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	reserved Name of the edition in which the object is actual

DBA_PROCEEDURES

DBA_PROCEEDURES lists all function, procedures or package

표 9-73 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or package
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or package
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in package
OBJECT_ID	NUMBER	ID of a function, procedures or package
SUBPROGRAM_ID	NUMBER	ID of procedures in package
OVERLOAD	VARCHAR(32)	ID of overloading procedure in package
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggregate function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented using the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; otherwise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

DBA_PROC_PRIVS

DBA_PROC_PRIVS describes the procedure grants, for which the current user is the procedure owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-74 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_PROFILES

DBA_PROFILES displays all profiles and their limits.

표 9-75 Column 정보

Column name	Data type	Description
PROFILE_NAME	VARCHAR(128)	Profile name
RESOURCE_NAME	VARCHAR(128)	Resource name
RESOURCE_TYPE	VARCHAR(32)	Indicates whether the resource profile is a KERNEL or a PASSWORD parameter
LIMIT_VALUE	LONG VARCHAR	Limit placed on this resource for this profile
COMMON	VARCHAR(3)	Indicates whether a given profile is common. (YES or NO)

DBA_SCHEMAS

Identify the schemata in the database.

표 9-76 Column 정보

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

DBA_SCHEMA_PATH

DBA_SCHEMA_PATH describes the schema search order of all authorizations in the database.

표 9-77 Column 정보

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the authorization
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the authorization

DBA_SCHEMA_PRIVS

DBA_SCHEMA_PRIVS describes all schema grants in the database.

표 9-78 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_SEQUENCES

DBA_SEQUENCES describes all sequences in the database.

표 9-79 Column 정보

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	reserved Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

DBA_SEQ_PRIVS

DBA_SEQ_PRIVS describes all sequence grants in the database.

표 9-80 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_SHARD_KEY_COLUMNS

DBA_SHARD_KEY_COLUMNS describes shard key columns of all shareded tables in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-81 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

DBA_SOURCE

DBA_SOURCE describes the text source of the stored objects accessible to the current user.

표 9-82 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the stored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

DBA_STAT_SYSTEM

DBA_STAT_SYSTEM describes analyzed system statistics.

표 9-83 Column 정보

Column name	Data type	Description
CPU_OPS	NATIVE_BIGINT	OPS(operations per second) of CPU
NETWORK_IOPS	NATIVE_BIGINT	IOPS(I/O operations per second) of Cluster NETWORK
NETWORK_BUFSIZE	NATIVE_BIGINT	buffer size of Cluster NETWORK when analyzed
LAST_ANALYZED	TIMESTAMP(2) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

DBA_SYS_PRIVS

DBA_SYS_PRIVS describes all system (database, tablespace, schema) privileges in the database.

표 9-84 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the grantee
PRIVILEGE	VARCHAR(256)	System(database, tablespace, schema) privilege
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
ADMIN_OPTION	VARCHAR(3)	equal to GRANTABLE column

DBA_SYNONYMS

DBA_SYNONYMS describes all synonyms in the database.

표 9-85 Column 정보

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

DBA_TABLES

DBA_TABLES describes all relational tables in the database.

표 9-86 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	reserved Number of used blocks in the table
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table

Column name	Data type	Description
CHAIN_CNT	NUMBER	reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for table blocks
		reserved

Column name	Data type	Description
ROW_MOVEMENT	VARCHAR(32)	If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	reserved Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	reserved Result cache mode annotation for the table: the value in (NULL, DEFAULT, FORCE, MANUAL)

DBA_TABLESPACES

DBA_TABLESPACES describes all tablespaces in the database.

표 9-87 Column 정보

Column name	Data type	Description
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
BLOCK_SIZE	NUMBER	Tablespace block size
INITIAL_EXTENT	NUMBER	reserved Default initial extent size (in bytes)
NEXT_EXTENT	NUMBER	reserved Default incremental extent size (in bytes)
MIN_EXTENTS	NUMBER	reserved Default minimum number of extents
MAX_EXTENTS	NUMBER	reserved Default maximum number of extents
MAX_SIZE	NUMBER	reserved Default maximum size of segments
PCT_INCREASE	NUMBER	reserved Default percent increase for extent size
MIN_EXTLEN	NUMBER	reserved Minimum extent size for this tablespace (in bytes)
STATUS	VARCHAR(32)	Tablespace status: the value in (ONLINE, OFFLINE, READ ONLY)
CONTENTS	VARCHAR(32)	Tablespace contents: the value in (SYSTEM, DATA, TEMPORARY, UNDO)
LOGGING	VARCHAR(32)	Default logging attribute: LOGGING, NOLOGGING
FORCE_LOGGING	VARCHAR(3)	reserved Indicates whether the tablespace is under force logging mode (YES) or not (NO)
EXTENT_MANAGEMENT	VARCHAR(32)	reserved Indicates whether the extents in the tablespace are dictionary managed (DICTIONARY) or locally managed (LOCAL)
ALLOCATION_TYPE	VARCHAR(32)	reserved Type of extent allocation in effect for the tablespace: the value in (SYSTEM, UNIFORM, USER)
PLUGGED_IN	VARCHAR(3)	reserved Indicates whether the tablespace is plugged in (YES) or not (NO)
SEGMENT_SPACE_MANAGEMENT	VARCHAR(32)	reserved Indicates whether the free and used segment space in the tablespace is managed using free lists (MANUAL) or bitmaps (AUTO)
DEF_TAB_COMPRESSION	VARCHAR(32)	reserved Indicates whether default table compression is enabled (ENABLED)

Column name	Data type	Description
		or not (DISABLED)
RETENTION	VARCHAR(32)	reserved Undo tablespace retention: the value in (GUARANTEE, NOGUARANTEE, NOT APPLY)
BIGFILE	VARCHAR(3)	reserved Indicates whether the tablespace is a bigfile tablespace (YES) or a smallfile tablespace (NO)
PREDICATE_EVALUATION	VARCHAR(32)	reserved Indicates whether predicates are evaluated by host (HOST) or by storage (STORAGE)
ENCRYPTED	VARCHAR(3)	reserved Indicates whether the tablespace is encrypted (YES) or not (NO)
COMPRESS_FOR	VARCHAR(32)	reserved Indicates whether the tablespace is encrypted (YES) or not (NO)

DBA_TAB_COLS

DBA_TAB_COLS describes the columns (including hidden columns) of all tables, views, and clusters in the database.

표 9-88 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
		reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	reserved Indicates existence/type of histogram
QUALIFIED_COLUMN_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

DBA_TAB_COLUMNS

DBA_TAB_COLUMNS describes the columns of the tables, views, and clusters accessible to the current user.

표 9-89 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
		reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

DBA_TAB_COMMENTS

DBA_TAB_COMMENTS displays comments on all tables and views in the database.

표 9-90 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

DBA_TAB_IDENTITY_COLS

DBA_TAB_IDENTITY_COLS describes all table identity columns.

표 9-91 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

DBA_TAB_PLACE

DBA_TAB_PLACE describes node placement of all cluster tables in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-92 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

DBA_TAB_PRIVS

DBA_TAB_PRIVS describes all object grants in the database.

표 9-93 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

DBA_TAB_SHARDS

DBA_TAB_SHARDS describes shard information of all sharded tables in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-94 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed

DBA_TBS_PRIVS

DBA_TBS_PRIVS describes all tablespace grants in the database.

표 9-95 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_USERS

DBA_USERS describes all users of the database.

표 9-96 Column 정보

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
PASSWORD	VARCHAR(128)	encrypted password
ACCOUNT_STATUS	VARCHAR(32)	Account status: the value in (OPEN, EXPIRED, EXPIRED(GRACE), LOCKED(TIMED), LOCKED, EXPIRED & LOCKED(TIMED), EXPIRED(GRACE) & LOCKED(TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED)
LOCK_DATE	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp the account was locked if account status was LOCKED
EXPIRY_DATE	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp of expiration of the account
FAILED_LOGIN_ATTEMPTS	NUMBER	Consecutive failed login attempts count
DEFAULT_TABLESPACE	VARCHAR(128)	Default tablespace for data
TEMPORARY_TABLESPACE	VARCHAR(128)	Name of the default tablespace for temporary tables or the name of a tablespace group
INDEX_TABLESPACE	VARCHAR(128)	Default tablespace for index
CREATED	TIMESTAMP(2) WITH OUT TIME ZONE	User creation timestamp
PROFIL_NAME	VARCHAR(128)	User resource profile name
INITIAL_RSRC_CONSUMER_GROUP	VARCHAR(128)	reserved Initial resource consumer group for the user
EXTERNAL_NAME	VARCHAR(128)	reserved User external name
PASSWORD_VERSIONS	VARCHAR(32)	reserved Shows the list of versions of the password hashes (verifiers).
EDITIONS_ENABLED	VARCHAR(1)	Indicates whether editions have been enabled for the corresponding user (Y) or not (N).
AUTHENTICATION_TYPE	VARCHAR(32)	reserved Indicates the authentication mechanism for the user.

DBA_VIEWS

DBA_VIEWS describes all views in the database.

표 9-97 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the view
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

USER 계열 View

현재 사용자가 소유한 객체에 대한 정보를 얻을 수 있다.

USER_ALL_TABLES

USER_ALL_TABLES describes the object tables and relational tables owned by the current user.

표 9-98 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table

Column name	Data type	Description
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table
CHAIN_CNT	NUMBER	reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	reserved If an object table, owner of the type from which the table is created
TABLE_TYPE	VARCHAR(128)	reserved If an object table, type of the table
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
		reserved

Column name	Data type	Description
SECONDARY	VARCHAR(1)	Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	reserved Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

USER_ARGUMENTS

USER_ARGUMENTS lists all arguments of functions, procedures.

표 9-99 Column 정보

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in package
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data Type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

USER_CATALOG

USER_CATALOG lists tables, views, synonyms, and sequences owned by the current user.

표 9-100 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

USER_COL_COMMENTS

USER_COL_COMMENTS displays comments on the columns of the tables and views owned by the current user.

표 9-101 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

USER_CLUSTER_TABLES

USER_CLUSTER_TABLES describes all cluster tables owned by the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-102 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST S HARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is nul l)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)

USER_COL_PRIVS

USER_COL_PRIVS describes the column object grants for which the current user is the object owner, grantor, or grantee.

표 9-103 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_COL_PRIVS_MADE

USER_COL_PRIVS_MADE describes the column object grants for which the current user is the object owner.

표 9-104 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_COL_PRIVS_RECD

USER_COL_PRIVS_RECD describes the column object grants for which the current user is the grantee.

표 9-105 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_CONSTRAINTS

USER_CONSTRAINTS describes all constraint definitions on tables owned by the current user.

표 9-106 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential integrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
RELY	VARCHAR(32)	reserved When NOT VALIDATED, indicates whether the constraint is to be taken into account for query rewrite (RELY) or not (NULL)
	TIMESTAMP(2) WITH	

Column name	Data type	Description
LAST_CHANGE	OUT TIME ZONE	When the constraint was last enabled or disabled
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	• reserved Indicates whether the constraint depends on a view (DEPEND ON VIEW) or not (NULL)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

USER_CONS_COLUMNS

USER_CONS_COLUMNS describes columns that are owned by the current user and that are specified in constraint definitions.

표 9-107 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

USER_DEPENDENCIES

USER_DEPENDENCIES describes dependencies between objects accessible to the current user

표 9-108 Column 정보

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

USER_EXTENTS

USER_EXTENTS describes the extents comprising the segments owned by the current user's objects.

표 9-109 Column 정보

Column name	Data type	Description
SEGMENT_SCHEMA	VARCHAR(128)	Schema of the segment associated with the extent
SEGMENT_NAME	VARCHAR(128)	Name of the segment associated with the extent
PARTITION_NAME	VARCHAR(128)	reserved Object Partition Name (Set to NULL for non-partitioned objects)
SEGMENT_TYPE	VARCHAR(32)	Type of the segment: TABLE, INDEX
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	reserved Extent number in the segment
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in Oracle blocks

USER_GLOBAL_SECONDARY_INDEXES

USER_GLOBAL_SECONDARY_INDEXES describes the global secondary indexes on the tables owned by the current user.

노트

Cluster에서만 사용할 수 있다.

표 9-110 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index

USER_GSI_PLACE

USER_GSI_PLACE describes node placement of all global secondary indexes on the tables owned by the current user in the cluster system.

[노트](#)

Cluster에서만 사용할 수 있다.

표 9-111 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

USER_INDEXES

USER_INDEXES describes indexes owned by the current user.

표 9-112 Column 정보

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NON UNIQUE)
COMPRESSION	VARCHAR(32)	reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to this segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block

Column name	Data type	Description
LOGGING	VARCHAR(3)	Indicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
BLEVEL	NUMBER	reserved B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	reserved Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	reserved Average number of leaf blocks in which each distinct value in the index appears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	reserved Indicates the amount of order of the rows in the table based on the values of the index
STATUS	VARCHAR(32)	reserved Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the index, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the indexes to be scanned, or DEFAULT
PARTITIONED	VARCHAR(3)	reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the index is a secondary object created by the method of the Data Cartridge (Y) or not (N)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for index blocks

Column name	Data type	Description
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for index blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for index blocks
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
PCT_DIRECT_ACCESS	NUMBER	reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned indexes, indicates whether statistics were collected by analyzing the index as a whole (YES) or were estimated from statistics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR(3)	reserved Indicates whether redundant primary key columns are eliminated from secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGEMENT	VARCHAR(32)	reserved If this is a domain index, indicates whether the domain index is system-managed (SYSTEM_MANAGED) or user-managed (USER_MANAGED)

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the index segment has been created (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

USER_IND_COLUMNS

USER_IND_COLUMNS describes the columns of the indexes owned by the current user and columns of indexes on tables owned by the current user.

표 9-113 Column 정보

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	reserved Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

USER_IND_PLACE

USER_IND_PLACE describes node placement of the indexes owned by the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-114 Column 정보

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(2) WITH OUT TIME ZONE	(deprecated)

USER_OBJECTS

USER_OBJECTS describes all objects owned by the current user.

표 9-115 Column 정보

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	reserved Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether this is a secondary object created by the ODCIIndex Create method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	reserved Name of the edition in which the object is actual

USER_PROCEEDURES

USER_PROCEEDURES lists of procedures owned by the current user.

표 9-116 Column 정보

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or package
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in package
OBJECT_ID	NUMBER	ID of a function, procedures or package
SUBPROGRAM_ID	NUMBER	ID of procedures in package
OVERLOAD	VARCHAR(32)	ID of overloading procedure in package
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggreate function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented using the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; otherwise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

USER_PROC_PRIVS

USER_PROC_PRIVS describes the procedure grants for which the current user is the procedure owner, grantor, or grantee.

표 9-117 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_PROC_PRIVS_MADE

USER_PROC_PRIVS_MADE describes the procedure grants for which the current user is the procedure owner or grantor.

표 9-118 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_PROC_PRIVS_RECD

USER_PROC_PRIVS_RECD describes the procedure grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

표 9-119 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SCHEMAS

Identify the schemata in a catalog that are owned by current user.

표 9-120 Column 정보

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

USER_SCHEMA_PATH

USER_SCHEMA_PATH describes the schema search order of the current user, for naming resolution of unqualified SQL schema objects.

표 9-121 Column 정보

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the user
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the user

USER_SCHEMA_PRIVS

USER_SCHEMA_PRIVS describes the schema grants, for which the current user is the schema owner, grantor, or grantee.

표 9-122 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SCHEMA_PRIVS_MADE

USER_SCHEMA_PRIVS_MADE describes the schema grants for which the current user is the schema owner.

표 9-123 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SCHEMA_PRIVS_RECD

USER_SCHEMA_PRIVS_RECD describes the schema grants for which the current user is the grantee.

표 9-124 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQUENCES

USER_SEQUENCES describes all sequences owned by the current user.

표 9-125 Column 정보

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	reserved Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

USER_SEQ_PRIVS

USER_SEQ_PRIVS describes the sequence grants for which the current user is the sequence owner, grant or, or grantee.

표 9-126 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQ_PRIVS_MADE

USER_SEQ_PRIVS_MADE describes the sequence grants for which the current user is the sequence owner.

표 9-127 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQ_PRIVS_RECD

USER_SEQ_PRIVS_RECD describes the sequence grants for which the current user is the grantee.

표 9-128 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SHARD_KEY_COLUMNS

USER_SHARD_KEY_COLUMNS describes shard key columns of shared tables owned by the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-129 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

USER_SOURCE

USER_SOURCE describes the text source of the stored objects accessible to the current user.

표 9-130 Column 정보

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the stored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

USER_SYNONYMS

USER_SYNONYMS describes all synonyms owned by the current user.

표 9-131 Column 정보

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

USER_SYS_PRIVS

USER_SYS_PRIVS describes system (database, tablespace, schema) privileges granted to the current user or PUBLIC.

표 9-132 Column 정보

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user, or PUBLIC
PRIVILEGE	VARCHAR(256)	System(database, tablespace, schema) privilege
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
ADMIN_OPTION	VARCHAR(3)	equal to GRANTABLE column

USER_TABLES

USER_TABLES describes the relational tables owned by the current user.

표 9-133 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	reserved Name of the cluster
IOT_NAME	VARCHAR(128)	reserved Name of the index-organized table
STATUS	VARCHAR(32)	reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	reserved Percentage increase in extent size
FREELISTS	NUMBER	reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
EMPTY_BLOCKS	NUMBER	reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	reserved Average available free space in the table
		reserved Number of rows in the table that are chained from one data block to

Column name	Data type	Description
CHAIN_CNT	NUMBER	o another or that have migrated to a new block, requiring a link to p reserve the old rowid
AVG_ROW_LEN	NUMBER	reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	reserved Indicates the duration of a temporary table
SKIP_CORRUPT	VARCHAR(32)	reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	reserved Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	reserved Result cache mode annotation for the table: the value in (NULL, DEFAULT, FORCE, MANUAL)

USER_TABLESPACES

USER_TABLESPACES describes the tablespaces accessible to the current user.

표 9-134 Column 정보

Column name	Data type	Description
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
BLOCK_SIZE	NUMBER	Tablespace block size
INITIAL_EXTENT	NUMBER	reserved Default initial extent size (in bytes)
NEXT_EXTENT	NUMBER	reserved Default incremental extent size (in bytes)
MIN_EXTENTS	NUMBER	reserved Default minimum number of extents
MAX_EXTENTS	NUMBER	reserved Default maximum number of extents
MAX_SIZE	NUMBER	reserved Default maximum size of segments
PCT_INCREASE	NUMBER	reserved Default percent increase for extent size
MIN_EXTLEN	NUMBER	reserved Minimum extent size for this tablespace (in bytes)
STATUS	VARCHAR(32)	Tablespace status: the value in (ONLINE, OFFLINE, READ ONLY)
CONTENTS	VARCHAR(32)	Tablespace contents: the value in (SYSTEM, DATA, TEMPORARY, UNDO)
LOGGING	VARCHAR(32)	Default logging attribute: LOGGING, NOLOGGING
FORCE_LOGGING	VARCHAR(3)	reserved Indicates whether the tablespace is under force logging mode (YES) or not (NO)
EXTENT_MANAGEMENT	VARCHAR(32)	reserved Indicates whether the extents in the tablespace are dictionary managed (DICTIONARY) or locally managed (LOCAL)
ALLOCATION_TYPE	VARCHAR(32)	reserved Type of extent allocation in effect for the tablespace: the value in (SYSTEM, UNIFORM, USER)
SEGMENT_SPACE_MANAGEMENT	VARCHAR(32)	reserved Indicates whether the free and used segment space in the tablespace is managed using free lists (MANUAL) or bitmaps (AUTO)
DEF_TAB_COMPRESSION	VARCHAR(32)	reserved Indicates whether default table compression is enabled (ENABLED) or not (DISABLED)
		reserved

Column name	Data type	Description
RETENTION	VARCHAR(32)	Undo tablespace retention: the value in (GUARANTEE, NOGUARANTEE, NOT APPLY)
BIGFILE	VARCHAR(3)	• reserved Indicates whether the tablespace is a bigfile tablespace (YES) or a smallfile tablespace (NO)
PREDICATE_EVALUATION	VARCHAR(32)	• reserved Indicates whether predicates are evaluated by host (HOST) or by storage (STORAGE)
ENCRYPTED	VARCHAR(3)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)
COMPRESS_FOR	VARCHAR(32)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)

USER_TAB_COLS

USER_TAB_COLS describes the columns (including hidden columns) of the tables, views, and clusters owned by the current user.

표 9-135 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether column statistics were collected for the table

Column name	Data type	Description
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	reserved Indicates existence/type of histogram
QUALIFIED_COL_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

USER_TAB_COLUMNS

USER_TAB_COLUMNS describes the columns of the tables, views, and clusters owned by the current user.

표 9-136 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
GLOBAL_STATS	VARCHAR(3)	reserved For partitioned tables, indicates whether column statistics were collected for the table

Column name	Data type	Description
USER_STATS	VARCHAR(3)	reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

USER_TAB_COMMENTS

USER_TAB_COMMENTS displays comments on the tables and views owned by the current user.

표 9-137 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

USER_TAB_IDENTITY_COLS

USER_TAB_IDENTITY_COLS describes all table identity columns.

표 9-138 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

USER_TAB_PLACE

USER_TAB_PLACE describes node placement of cluster tables owned by the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-139 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

USER_TAB_PRIVS

USER_TAB_PRIVS describes the object grants for which the current user is the object owner, grantor, or grantee.

표 9-140 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

USER_TAB_PRIVS_MADE

USER_TAB_PRIVS_MADE describes the object grants for which the current user is the object owner.

표 9-141 Column 정보

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

USER_TAB_PRIVS_RECD

USER_TAB_PRIVS_RECD describes the object grants for which the current user is the grantee.

표 9-142 Column 정보

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

USER_TAB_SHARDS

USER_TAB_SHARDS describes shard information of sharded tables owned by the current user in the cluster system.

노트

Cluster에서만 사용할 수 있다.

표 9-143 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed

USER_USERS

USER_USERS describes the current user.

표 9-144 Column 정보

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
ACCOUNT_STATUS	VARCHAR(32)	Account status: the value in (OPEN, EXPIRED, EXPIRED(GRACE), LOCKED(TIMED), LOCKED, EXPIRED & LOCKED(TIMED), EXPIRED(GRACE) & LOCKED(TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED)
LOCK_DATE	TIMESTAMP(2) WITHOUT TIME ZONE	Timestamp the account was locked if account status was LOCKED
EXPIRY_DATE	TIMESTAMP(2) WITHOUT TIME ZONE	Timestamp of expiration of the account
DEFAULT_TABLESPACE	VARCHAR(128)	Default tablespace for data
TEMPORARY_TABLESPACE	VARCHAR(128)	Name of the default tablespace for temporary tables or the name of a tablespace group
INDEX_TABLESPACE	VARCHAR(128)	Default tablespace for index
CREATED	TIMESTAMP(2) WITHOUT TIME ZONE	User creation timestamp
INITIAL_RSRC_CONSUMER_GROUP	VARCHAR(128)	reserved Initial resource consumer group for the user
EXTERNAL_NAME	VARCHAR(128)	reserved User external name

USER_VIEWS

USER_VIEWS describes the views owned by the current user.

표 9-145 Column 정보

Column name	Data type	Description
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

기타 View

ALL 계열이나 DBA 계열, USER 계열이 아닌 view나 테이블이다.

AUDIT_POLICIES

AUDIT_POLICIES contains one row for each audit policy.

표 9-146 Column 정보

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
ENABLED	VARCHAR(3)	indicates whether the audit policy is enabled (YES) or not (NO)
CREATED_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	created time of the audit policy
MODIFIED_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	last modified time of the audit policy
COMMENTS	VARCHAR(1024)	comments of the audit policy

AUDIT_POLICY_OPTIONS

AUDIT_POLICY_OPTIONS describes all audit policies created in the database.

표 9-147 Column 정보

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
AUDIT_OPTION	VARCHAR(32)	auditing option defined in the audit policy
AUDIT_OPTION_TYPE	VARCHAR(32)	The values of AUDIT_OPTION_TYPE_NAME in ('DATABASE PRIVILEGE', 'SYSTEM ACTION', 'OBJECT ACTION')
OBJECT_SCHEMA	VARCHAR(128)	schema name, for an object-specific auditing option
OBJECT_NAME	VARCHAR(128)	object name, for an object-specific auditing option
OBJECT_TYPE	VARCHAR(32)	object type name, for an object-specific auditing option

AUDIT_POLICY_ENABLED

AUDIT_POLICY_ENABLE describes all the audit policies that are enable in the database.

표 9-148 Column 정보

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
ENABLED_OPT	VARCHAR(32)	enable option of the audit policy, the possible values are BY, EXCEPT
USER_NAME	VARCHAR(128)	user name for whom the audit policy is enable
WHEN_SUCCESS	VARCHAR(3)	indicates whether the audit policy is enable for auditing successful events or not
WHEN_FAILURE	VARCHAR(3)	indicates whether the audit policy is enable for auditing unsuccessful events or not

AUDIT_TRAIL

AUDIT_TRAIL displays audit records from the audit trail.

표 9-149 Column 정보

Column name	Data type	Description
MEMBER_NAME	VARCHAR(128)	cluster member name
SESSION_ID	NUMBER	session identifier
SESSION_SERIAL	NUMBER	session serial number
LOGON_USERNAME	VARCHAR(128)	logon user name of the user whose actions were audited
CURRENT_USERNAME	VARCHAR(128)	effective user for the statement execution
SERVER_PROCESS	NUMBER	server process identifier for the session
CLIENT_PROGRAM_NAME	VARCHAR(128)	client program used for session
CLIENT_USERNAME	VARCHAR(128)	client operating system user name for the session
CLIENT_PROCESS	NUMBER	client process identifier for the session
CLIENT_HOST	VARCHAR(128)	client host ip address for the session
CLIENT_PORT	NUMBER	client port number for the session
CLIENT_TERMINAL	VARCHAR(128)	client terminal name for the session
TRANSACTION_ID	NUMBER	transaction identifier
SCN	VARCHAR(128)	system change number (SCN) string of the query at the time of the event
GCN	NUMBER	global change number (GCN) of the query at the time of the event
DCN	NUMBER	domain change number (DCN) of the query at the time of the event
LCN	NUMBER	local change number (LCN) of the query at the time of the event
STMT_NO	NUMBER	numeric number for each statement run in a session
SQL_TEXT	LONG VARCHAR	SQL associated with the event
SQL_BINDS	LONG VARCHAR	list of bind variables, if any, associated with SQL_TEXT
RETURN_CODE	NUMBER	error code generated by the action, zero if the action succeeded
ERROR_MESSAGE	VARCHAR(1024)	error message generated by the action, null if the action succeeded
ENTRY_ID	NUMBER	audit trail entry identifier in the session
EVENT_TIMESTAMP	TIMESTAMP(2) WITHOUT TIME ZONE	timestamp of the creation of the audit trail entry in local time zone
POLICY_NAME	VARCHAR(128)	audit policy name that caused the current audit record
PRIVILEGE_USED	VARCHAR(32)	database privilege used to execute the action
ACTION_NAME	VARCHAR(32)	action name executed by the user

Column name	Data type	Description
OBJECT_TYPE	VARCHAR(32)	object type of object affected by the action
OBJECT_SCHEMA	VARCHAR(128)	schema name of object affected by the action
OBJECT_NAME	VARCHAR(128)	object name of object affected by the action

DATABASE_PROPERTIES

DATABASE_PROPERTIES lists permanent database properties.

표 9-150 Column 정보

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	Property name
PROPERTY_VALUE	VARCHAR(4000)	Property value
DESCRIPTION	VARCHAR(4000)	Property description

DBC_TABLE_TYPE_INFO

Identify the ODBC/JDBC table types available in this database.

표 9-151 Column 정보

Column name	Data type	Description
DBC_TABLE_TYPE_ID	NUMBER	number identifier of the table type in ODBC/JDBC
DBC_TABLE_TYPE	VARCHAR(128)	name of the table type in ODBC/JDBC
IS_SUPPORTED	BOOLEAN	is supported feature
COMMENTS	VARCHAR(1024)	comments of the table type

DICTIONARY

DICTIONARY contains descriptions of data dictionary tables and views.

표 9-152 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COMMENTS	VARCHAR(1024)	Text comment on the object

DICT_COLUMNS

DICT_COLUMNS contains descriptions of columns in data dictionary tables and views.

표 9-153 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object that contains the column
TABLE_NAME	VARCHAR(128)	Name of the object that contains the column
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Text comment on the column

IMPLEMENTATION_INFO

IMPLEMENTATION_INFO contains information about various aspects that are left implementation-defined.

표 9-154 Column 정보

Column name	Data type	Description
IMPLEMENTATION_INFO_ID	NUMBER	identifier of the implementation item
IMPLEMENTATION_INFO_NAME	VARCHAR(1024)	descriptive name of the implementation item
INTEGER_VALUE	NUMBER	Value of the implementation item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(1024)	Value of the implementation item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation item

IMPLEMENTATION_INFO_BASE

The IMPLEMENTATION_INFO_BASE table has one row for each implementation information item.

표 9-155 Column 정보

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the implementation item
SUB_ID	VARCHAR(32)	identifier string of the implementation item
NAME	VARCHAR(1024)	descriptive name of the implementation item
SUB_NAME	VARCHAR(1024)	descriptive name of the implementation item
IS_SUPPORTED	BOOLEAN	TRUE if the implementation item is supported, FALSE if not
INTEGER_VALUE	NUMBER	Value of the implementation item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(1024)	Value of the implementation item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation item

JDBC_CLIENT_PROPS

JDBC_CLIENT_PROPS is the set of jdbc client properties.

표 9-156 Column 정보

Column name	Data type	Description
NAME	VARCHAR(128)	property name
MAX_LEN	NATIVE_INTEGER	max length of a value
DEFAULT_VALUE	VARCHAR(128)	default value
DESCRIPTION	VARCHAR(256)	descripton on that property

PRODUCT

PRODUCT is about the product name, version for ODBC, JDBC interface.

표 9-157 Column 정보

Column name	Data type	Description
NAME	VARCHAR(32)	the product name
VERSION	VARCHAR(128)	product full version information
PRODUCT_VERSION	NUMBER	product version
MAJOR_VERSION	NUMBER	major version
MINOR_VERSION	NUMBER	minor version
PATCH_VERSION	NUMBER	patch version

SESSION_PRIVS

SESSION_PRIVS describes the privileges that are currently available to the user.

표 9-158 Column 정보

Column name	Data type	Description
PRIVILEGE	VARCHAR(256)	Name of the privilege

SUPPLEMENTAL_LOG_TABLE_INFO

SUPPLEMENTAL_LOG_TABLE_INFO describes table-level supplemental logging status.

표 9-159 Column 정보

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
SUPPLEMENTAL_LOG_DATA_PK	VARCHAR(32)	Status of table-level PRIMARY KEY COLUMNS supplemental logging: IMPLICIT, EXPLICIT, NO

Aliased Synonym

DICTIONARY_SCHEMA 내의 view나 테이블을 가리키는 public synonym이다.

COLS

COLS is a public synonym for USER_TAB_COLUMNS.

DICT

DICT is a public synonym for DICTIONARY.

IND

IND is a public synonym for USER_INDEXES.

OBJ

OBJ is a public synonym for USER_OBJECTS.

SEQ

SEQ is a public synonym for USER_SEQUENCES.

TABS

TABS is a public synonym for USER_TABLES.

9.2 INFORMATION_SCHEMA

INFORMATION_SCHEMA 스키마의 view들은 SQL 표준에서 정의한 INFORMATION_SCHEMA의 view들과 동일한 정보를 제공한다.

해당 view들을 사용하려면 다음과 같이 InformationSchema.sql 을 실행해야 한다.

- Standalone의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/InformationSchema.sql
```

- Cluster의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
```

노트

INFORMATION_SCHEMA의 view와 테이블들은 open 단계부터 조회할 수 있다.

COLUMNS

Identify the columns of tables defined in this catalog that are accessible to given user or role.

표 9-160 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the column
TABLE_OWNER	VARCHAR(128)	owner name of the column
TABLE_SCHEMA	VARCHAR(128)	schema name of the column
TABLE_NAME	VARCHAR(128)	table name of the column
COLUMN_NAME	VARCHAR(128)	column name
ORDINAL_POSITION	NUMBER	the ordinal position (> 0) of the column in the table
COLUMN_DEFAULT	LONG VARCHAR	the default for the column
IS_NULLABLE	BOOLEAN	is nullable of the column
DATA_TYPE	VARCHAR(128)	the standard name of the data type
CHARACTER_MAXIMUM_LENGTH	NUMBER	the maximum length in characters
CHARACTER_OCTET_LENGTH	NUMBER	the maximum length in octets
NUMERIC_PRECISION	NUMBER	the numeric precision of the numerical Data type
NUMERIC_PRECISION_RADIX	NUMBER	the radix (2 or 10) of the precision of the numerical data type
NUMERIC_SCALE	NUMBER	the numeric scale of the exact numerical data type
DATETIME_PRECISION	NUMBER	for a datetime or interval type, the value is the fractional seconds precision
INTERVAL_TYPE	VARCHAR(32)	for a interval type, the value is in (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND)
INTERVAL_PRECISION	NUMBER	for a interval type, the value is the leading precision
CHARACTER_SET_CATALOG	VARCHAR(128)	catalog name of the character set if is is a character string type
CHARACTER_SET_SCHEMA	VARCHAR(128)	schema name of the character set if is is a character string type
CHARACTER_SET_NAME	VARCHAR(128)	character set name of the character set if is is a character string type
COLLATION_CATALOG	VARCHAR(128)	catalog name of the applicable collation if is is a character string type
COLLATION_SCHEMA	VARCHAR(128)	schema name of the applicable collation if is is a character string type
COLLATION_NAME	VARCHAR(128)	collation name of the applicable collation if is is a character

Column name	Data type	Description
		string type
DOMAIN_CATALOG	VARCHAR(128)	catalog name of the domain used by the column being described
DOMAIN_SCHEMA	VARCHAR(128)	schema name of the domain used by the column being described
DOMAIN_NAME	VARCHAR(128)	domain name of the domain used by the column being described
UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined type of the data type being described
UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined type of the data type being described
UDT_NAME	VARCHAR(128)	user-defined type name of the user-defined type of the data type being described
SCOPE_CATALOG	VARCHAR(128)	catalog name of the referenceable table if DATA_TYPE is REF
SCOPE_SCHEMA	VARCHAR(128)	schema name of the referenceable table if DATA_TYPE is REF
SCOPE_NAME	VARCHAR(128)	scope name of the referenceable table if DATA_TYPE is REF
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality if DATA_TYPE is ARRAY
DTD_IDENTIFIER	NUMBER	data type descriptor identifier
IS_SELF_REFERENCING	BOOLEAN	is a self-referencing column
IS_IDENTITY	BOOLEAN	is an identity column
IDENTITY_GENERATION	VARCHAR(32)	for an identity column, the value is in (ALWAYS, BY DEFAULT)
IDENTITY_START	NUMBER	for an identity column, the start value of the identity column
IDENTITY_INCREMENT	NUMBER	for an identity column, the increment of the identity column
IDENTITY_MAXIMUM	NUMBER	for an identity column, the maximum value of the identity column
IDENTITY_MINIMUM	NUMBER	for an identity column, the minimum value of the identity column
IDENTITY_CYCLE	BOOLEAN	for an identity column, the cycle option
IS_GENERATED	BOOLEAN	is a generated column
GENERATION_EXPRESSION	VARCHAR(128)	for a generated column, the text of the generation expression
IS_SYSTEM_VERSION_START	BOOLEAN	is a system-version start column
IS_SYSTEM_VERSION_END	BOOLEAN	is a system-version end column
SYSTEM_VERSION_TIMESTAMP_GENERATION	VARCHAR(32)	for a system-version column, the value is ALWAYS
IS_UPDATABLE	BOOLEAN	is an updatable column
DECLARED_DATA_TYPE	VARCHAR(128)	the data type name that a user declared

Column name	Data type	Description
DECLARED_NUMERIC_PRECISION	NUMBER	the precision value that a user declared
DECLARED_NUMERIC_SCALE	NUMBER	the scale value that a user declared
COMMENTS	VARCHAR(1024)	comments of the column

COLUMN_PRIVILEGES

Identify the privileges on columns of tables defined in this catalog that are available to or granted by a given user or role.

표 9-161 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted column privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the column privilege being described is granted
TABLE_CATALOG	VARCHAR(128)	catalog name of the column on which the privilege being described was granted
TABLE_OWNER	VARCHAR(128)	table owner name of the column on which the privilege being described was granted
TABLE_SCHEMA	VARCHAR(128)	schema name of the column on which the privilege being described was granted
TABLE_NAME	VARCHAR(128)	table name of the column on which the privilege being described was granted
COLUMN_NAME	VARCHAR(128)	column name of the column on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (SELECT, INSERT, UPDATE, REFERENCES)
IS_GRANTABLE	BOOLEAN	is grantable

CONSTRAINT_COLUMN_USAGE

Identify the columns used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user or role.

표 9-162 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the column that participates in the constraint being described
TABLE_OWNER	VARCHAR(128)	owner name of the column that participates in the constraint being described
TABLE_SCHEMA	VARCHAR(128)	schema name of the column that participates in the constraint being described
TABLE_NAME	VARCHAR(128)	table name of the column that participates in the constraint being described
COLUMN_NAME	VARCHAR(128)	column name that participates in the constraint being described
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name

CONSTRAINT_TABLE_USAGE

Identify the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user or role.

표 9-163 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table that participates in the constraint being described
TABLE_OWNER	VARCHAR(128)	owner name of the table that participates in the constraint being described
TABLE_SCHEMA	VARCHAR(128)	schema name of the table that participates in the constraint being described
TABLE_NAME	VARCHAR(128)	table name that participates in the constraint being described
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name

INFORMATION_SCHEMA_CATALOG_NAME

Identify the catalog that contains the Information Schema.

표 9-164 Column 정보

Column name	Data type	Description
CATALOG_NAME	VARCHAR(128)	the name of catalog in which this Information Schema resides

KEY_COLUMN_USAGE

Identify the columns defined in this catalog that are constrained as keys and that are accessible by a given user or role.

표 9-165 Column 정보

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name
TABLE_CATALOG	VARCHAR(128)	catalog name of the column that participates in the constraint being described
TABLE_OWNER	VARCHAR(128)	owner name of the column that participates in the constraint being described
TABLE_SCHEMA	VARCHAR(128)	schema name of the column that participates in the constraint being described
TABLE_NAME	VARCHAR(128)	table name of the column that participates in the constraint being described
COLUMN_NAME	VARCHAR(128)	column name that participates in the constraint being described
ORDINAL_POSITION	NUMBER	the ordinal position of the specific column in the constraint being described. If the constraint described is a key of cardinality 1 (one), then the value of ORDINAL_POSITION is always 1 (one).
POSITION_IN_UNIQUE_CONSTRAINT	NUMBER	If the constraint being described is a foreign key constraint, then the value of POSITION_IN_UNIQUE_CONSTRAINT is the ordinal position of the referenced column corresponding to the referencing column being described, in the corresponding unique key constraint.

PARAMETERS

Identify the SQL parameters of SQL-invoked routines defined in this catalog that are accessible to a given user or role.

표 9-166 Column 정보

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	catalog name of the specific name of the SQL- invoked routine that contains the SQL parameter being described
SPECIFIC_OWNER	VARCHAR(128)	owner name of the specific name of the SQL- invoked routine that contains the SQL parameter being described
SPECIFIC_SCHEMA	VARCHAR(128)	schema name of the specific name of the SQL- invoked routine that contains the SQL parameter being described
SPECIFIC_NAME	VARCHAR(128)	specific name of the SQL- invoked routine that contains the SQL parameter being described
ORDINAL_POSITION	NUMBER	ordinal position of the SQL- invoked routine that contains the SQL parameter being described
PARAMETER_MODE	VARCHAR(32)	parameter mode of the SQL parameter being described
IS_RESULT	BOOLEAN	the parameter is RESULT parameter of type-preserving function
AS_LOCATOR	BOOLEAN	the parameter is passed as locator
PARAMETER_NAME	VARCHAR(128)	name of the SQL parameter being described
FROM_SQL_SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the from-sql routine for the input parameter being described
FROM_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the from-sql routine for the input parameter being described
FROM_SQL_SPECIFIC_NAME	VARCHAR(128)	specific name of the from-sql routine for the input parameter being described
TO_SQL_SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the to-sql routine for the input parameter being described
TO_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the to-sql routine for the input parameter being described
TO_SQL_SPECIFIC_NAME	VARCHAR(128)	specific name of the to-sql routine for the input parameter being described
DATA_TYPE	VARCHAR(128)	data type of the SQL parameter being described
CHARACTER_MAXIMUM_LENGTH	NUMBER	maximum length of the SQL parameter being described
CHARACTER_OCTET_LENGTH	NUMBER	maximum length in octets of the SQL parameter being described
CHARACTER_SET_CATALOG	VARCHAR(128)	character set catalog name of the data type of the SQL parameter being described
CHARACTER_SET_SCHEMA	VARCHAR(128)	character set schema name of the data type of the SQL parameter being described

Column name	Data type	Description
CHARACTER_SET_NAME	VARCHAR(128)	character set name of the data type of the SQL parameter being described
COLLATION_CATALOG	VARCHAR(128)	collation catalog name of the data type of the SQL parameter being described
COLLATION_SCHEMA	VARCHAR(128)	collation schema name of the data type of the SQL parameter being described
COLLATION_NAME	VARCHAR(128)	collation name of the data type of the SQL parameter being described
NUMERIC_PRECISION	NUMBER	precision of the data type of the SQL parameter being described
NUMERIC_PRECISION_RADIX	NUMBER	precision radix of the data type of the SQL parameter being described
NUMERIC_SCALE	NUMBER	scale of the data type of the SQL parameter being described
DATETIME_PRECISION	NUMBER	fractional second precisions of the data type of the SQL parameter being described
INTERVAL_TYPE	VARCHAR(32)	interval qualifier of the data type of the SQL parameter being described
INTERVAL_PRECISION	NUMBER	interval precision of the data type of the SQL parameter being described
UDT_CATALOG	VARCHAR(128)	catalog name of UDT of the data type of the SQL parameter being described
UDT_SCHEMA	VARCHAR(128)	schema name of UDT of the data type of the SQL parameter being described
UDT_NAME	VARCHAR(128)	name of UDT of the data type of the SQL parameter being described
SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable tables of the data type of the SQL parameter being described
SCOPE_SCHEMA	VARCHAR(128)	schema name of referenceable tables of the data type of the SQL parameter being described
SCOPE_NAME	VARCHAR(128)	name of referenceable tables of the data type of the SQL parameter being described
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality of the data type of the SQL parameter being described
DTD_IDENTIFIER	NUMBER	dtd identifier of the data type of the SQL parameter being described
DECLARED_DATA_TYPE	VARCHAR(128)	declared data type of the SQL parameter being described
DECLARED_NUMERIC_PRECISION	NUMBER	precision of declared data type of the SQL parameter being described
DECLARED_NUMERIC_SCALE	NUMBER	scale of declared data type of the SQL parameter being described
PARAMETER_DEFAULT	LONG VARCHAR	default value of the SQL parameter being described

REFERENTIAL_CONSTRAINTS

Identify the referential constraints defined on tables in this catalog that are accessible to a given user or role.

표 9-167 Column 정보

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the referential constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name who owns the referential constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the referential constraint being described
CONSTRAINT_NAME	VARCHAR(128)	referential constraint name
CONSTRAINT_TABLE_NAME	VARCHAR(128)	name of the table to which the referential constraint being described applies
CONSTRAINT_COLUMN_NAME	VARCHAR(128)	column name of the table to which the referential constraint being described applies
ORDINAL_POSITION	NUMBER	the ordinal position of the specific column in the referential constraint being described.
UNIQUE_CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_OWNER	VARCHAR(128)	owner name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_NAME	VARCHAR(128)	constraint name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_TABLE_NAME	VARCHAR(128)	table name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_COLUMN_NAME	VARCHAR(128)	column name of the unique or primary key constraint applied to the referenced column list being described
IS_PRIMARY_KEY	BOOLEAN	whether the constraint applied to the referenced column list being described, is primary key or not
MATCH_OPTION	VARCHAR(32)	the referential constraint that has a match option: the value in (SIMPLE, PARTIAL, FULL)
UPDATE_RULE	VARCHAR(32)	the referential constraint that has an update rule: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
DELETE_RULE	VARCHAR(32)	the referential constraint that has a delete rule: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
IS_DEFERRABLE	BOOLEAN	is a deferrable constraint
INITIALLY_DEFERRED	BOOLEAN	is an initially deferred constraint

ROUTINES

Identify the SQL-invoked routines in this catalog that are accessible to a given user or role.

표 9-168 Column 정보

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
ROUTINE_CATALOG	VARCHAR(128)	catalog name of the routine
ROUTINE_OWNER	VARCHAR(128)	owner name of the routine
ROUTINE_SCHEMA	VARCHAR(128)	schema name of the routine
ROUTINE_NAME	VARCHAR(128)	null
ROUTINE_TYPE	VARCHAR(128)	name of the routine
MODULE_CATALOG	VARCHAR(128)	module name of the routine
MODULE_SCHEMA	VARCHAR(128)	schema name of the module in which the routine is defined
MODULE_NAME	VARCHAR(128)	name of the module in which the routine is defined
UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined data type which defined the routine as a method function
UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined data type which defined the routine as a method function
UDT_NAME	VARCHAR(128)	name of the user-defined data type which defined the routine as a method function
DATA_TYPE	VARCHAR(128)	data type the routine returns
CHARACTER_MAXIMUM_LENGTH	NUMBER	maximum character length of data type the routine returns
CHARACTER_OCTET_LENGTH	NUMBER	maximum character length in octets of data type the routine returns
CHARACTER_SET_CATALOG	VARCHAR(128)	character set catalog name of data type the routine returns
CHARACTER_SET_SCHEMA	VARCHAR(128)	character set schema name of data type the routine returns
CHARACTER_SET_NAME	VARCHAR(128)	character set name of data type the routine returns
COLLATION_CATALOG	VARCHAR(128)	collation catalog name of data type the routine returns
COLLATION_SCHEMA	VARCHAR(128)	collation schema name of data type the routine returns
COLLATION_NAME	VARCHAR(128)	collation name of data type the routine returns
NUMERIC_PRECISION	NUMBER	precision of data type the routine returns
NUMERIC_PRECISION_RADIX	NUMBER	precision radix of data type the routine returns
NUMERIC_SCALE	NUMBER	scale of data type the routine returns

Column name	Data type	Description
DATETIME_PRECISION	NUMBER	fractional seconds precision of data type the routine returns
INTERVAL_TYPE	VARCHAR(32)	interval qualifier for data type the routine returns
INTERVAL_PRECISION	NUMBER	interval leading field precision of data type the routine returns
TYPE_UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined data type, which is the data type the routine returns
TYPE_UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined data type, which is the data type the routine returns
TYPE_UDT_NAME	VARCHAR(128)	name of the user-defined data type, which is the data type the routine returns
SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable table
SCOPE_SCHEMA	VARCHAR(128)	schema name of referenceable table
SCOPE_NAME	VARCHAR(128)	name of referenceable table
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality of data type the routine returns
DTD_IDENTIFIER	NUMBER	dtd identifier of data type the routine returns
ROUTINE_BODY	VARCHAR(32)	type of the routine body
ROUTINE_DEFINITION	LONG VARCHAR	catalog name of the routine
EXTERNAL_NAME	VARCHAR(128)	external name of the external routine
EXTERNAL_LANGUAGE	VARCHAR(32)	language of the external routine
PARAMETER_STYLE	VARCHAR(32)	SQL parameter passing style of the external routine
IS_DETERMINISTIC	BOOLEAN	the routine is deterministic or not
SQL_DATA_ACCESS	VARCHAR(32)	routine possibly contains SQL or access data
IS_NULL_CALL	BOOLEAN	routine returns NULL if any of parameter values are NULL
SQL_PATH	VARCHAR(1024)	described SQL PATH when the routine is defined
SCHEMA_LEVEL_ROUTINE	BOOLEAN	the routine is schema-level routine
MAX_DYNAMIC_RESULT_SETS	NUMBER	max result set count of the routine
IS_USER_DEFINED_CAST	BOOLEAN	the routine is a function that is a user-defined cast function
IS_IMPLICITLY_INVOCABLE	BOOLEAN	the user-defined cast function is implicitly invocable
SECURITY_TYPE	VARCHAR(32)	security type of the routine(DEFINER/INVOKER)
TO_SQL_SPECIFIC_CATALOG	VARCHAR(128)	catalog name of the to-sql routine of the result type of routine
TO_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	schema name of the to-sql routine of the result type of routine
TO_SQL_SPECIFIC_NAME	VARCHAR(128)	name of the to-sql routine of the result type of routine
AS_LOCATOR	BOOLEAN	return value of the routine is passed as locator
CREATED	TIMESTAMP(2) WITHOUT TIME ZONE	creation time of the routine
LAST_ALTERED	TIMESTAMP(2) WITHOUT TIME ZONE	most lately altered time of the routine
NEW_SAVEPOINT_LEVEL	BOOLEAN	specify new savepoint level or not

Column name	Data type	Description
IS_UDT_DEPENDENT	BOOLEAN	routine is dependent
RESULT_CAST_FROM_DATA_TYPE	VARCHAR(128)	data type which is specified in result cast clause of the routine definition
RESULT_CAST_AS_LOCATOR	BOOLEAN	locator indication which is specified in result cast clause of the routine definition
RESULT_CAST_CHAR_MAXIMUM_LENGTH	NUMBER	maximum character length of data type which is specified in result cast clause of the routine definition
RESULT_CAST_CHAR_OCTET_LENGTH	NUMBER	maximum character length in octets of data type which is specified in result cast clause of the routine definition
RESULT_CAST_CHAR_SET_CATALOG	VARCHAR(128)	character set catalog name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_CHAR_SET_SCHEMA	VARCHAR(128)	character set schema name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_CHARACTER_SET_NAME	VARCHAR(128)	character set name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_COLLATION_CATALOG	VARCHAR(128)	collation catalog name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_COLLATION_SCHEMA	VARCHAR(128)	collation schema name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_COLLATION_NAME	VARCHAR(128)	collation name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_NUMERIC_PRECISION	NUMBER	precision of data type which is specified in result cast clause of the routine definition
RESULT_CAST_NUMERIC_RADIX	NUMBER	precision radix of data type which is specified in result cast clause of the routine definition
RESULT_CAST_NUMERIC_SCALE	NUMBER	scale of data type which is specified in result cast clause of the routine definition
RESULT_CAST_DATETIME_PRECISION	NUMBER	fractional seconds precision of data type which is specified in result cast clause of the routine definition
RESULT_CAST_INTERVAL_TYPE	VARCHAR(32)	interval qualifier of data type which is specified in result cast clause of the routine definition
RESULT_CAST_INTERVAL_PRECISION	NUMBER	interval precision of data type which is specified in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_CATALOG	VARCHAR(128)	UDT catalog name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_SCHEMA	VARCHAR(128)	UDT schema name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_NAME	VARCHAR(128)	UDT name of data type which is specified in result cast clause of the routine definition
RESULT_CAST_SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable table described in result cast clause of the routine definition
RESULT_CAST_SCOPE_SCHEMA		schema name of referenceable table described in result cast clause of the routine definition

Column name	Data type	Description
HEMA	VARCHAR(128)	ause of the routine definition
RESULT_CAST_SCOPE_NAME	VARCHAR(128)	name of referenceable table described in result cast clause of the routine definition
RESULT_CAST_MAX_CARDINALITY	NUMBER	maximum cardinality of data type which is specified in result cast clause of the routine definition
RESULT_CAST_DTD_IDENTIFIER	NUMBER	dtd identifier of data type which is specified in result cast clause of the routine definition
DECLARED_DATA_TYPE	VARCHAR(128)	declared data type of the routine returns
DECLARED_NUMERIC_PRECISION	NUMBER	declared data type precision of the routine returns
DECLARED_NUMERIC_SCALE	NUMBER	declared data type scale of the routine returns
RESULT_CAST_FROM_DECLARED_DATA_TYPE	VARCHAR(128)	declared data type which is specified in result cast clause of the routine definition
RESULT_CAST_DECLARED_NUMERIC_PRECISION	NUMBER	declared data type precision which is specified in result cast clause of the routine definition
RESULT_CAST_DECLARED_NUMERIC_SCALE	NUMBER	declared data type scale which is specified in result cast clause of the routine definition

ROUTINE_PRIVILEGES

Identify the privileges on SQL-invoked routines defined in this catalog that are available to or granted by a given user or role.

표 9-169 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted routine privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the routine privilege being described is granted
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_NAME	VARCHAR(128)	specific name of the the SQL-invoked routine on which the privilege being described was granted
ROUTINE_CATALOG	VARCHAR(128)	routine catalog name of the SQL-invoked routine on which the privilege being described was granted
ROUTINE_OWNER	VARCHAR(128)	null
ROUTINE_SCHEMA	VARCHAR(128)	routine schema name of the the SQL-invoked routine on which the privilege being described was granted
ROUTINE_NAME	VARCHAR(128)	routine name of the the SQL-invoked routine on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (EXECUTE)
IS_GRANTABLE	BOOLEAN	is grantable

ROUTINE_ROUTINE_USAGE

Identify each SQL-invoked routine owned by a given user or role on which an SQL routine defined in this catalog is dependent.

표 9-170 Column 정보

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
ROUTINE_CATALOG	VARCHAR(128)	routine catalog name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_OWNER	VARCHAR(128)	routine owner name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_SCHEMA	VARCHAR(128)	routine schema name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_NAME	VARCHAR(128)	routine name of a routine contained in routine body of the SQL-invoked routine

ROUTINE_SEQUENCE_USAGE

Identify each external sequence generator owned by a given user or role on which some SQL routine defined in this catalog is dependent.

표 9-171 Column 정보

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence of contained in routine body of the SQL-invoked routine
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence of contained in routine body of the SQL-invoked routine
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence of contained in routine body of the SQL-invoked routine
SEQUENCE_NAME	VARCHAR(128)	sequence name of contained in routine body of the SQL-invoked routine

ROUTINE_TABLE_USAGE

Identify the tables owned by a given user or role on which SQL routines defined in this catalog are dependent.

표 9-172 Column 정보

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
TABLE_CATALOG	VARCHAR(128)	catalog name of the table of contained in routine body of the SQL-invoked routine
TABLE_OWNER	VARCHAR(128)	owner name of the table of contained in routine body of the SQL-invoked routine
TABLE_SCHEMA	VARCHAR(128)	schema name of the table of contained in routine body of the SQL-invoked routine
TABLE_NAME	VARCHAR(128)	table name of contained in routine body of the SQL-invoked routine

SCHEMATA

Identify the schemata in a catalog that are owned by given user or accessible to given user or role.

표 9-173 Column 정보

Column name	Data type	Description
CATALOG_NAME	VARCHAR(128)	catalog name of the schema
SCHEMA_NAME	VARCHAR(128)	schema name
SCHEMA_OWNER	VARCHAR(128)	authorization name who owns the schema
DEFAULT_CHARACTER_SET_CATALOG	VARCHAR(128)	catalog name of the default character set for columns and domains in the schemata
DEFAULT_CHARACTER_SET_SCHEMA	VARCHAR(128)	schema name of the default character set for columns and domains in the schemata
DEFAULT_CHARACTER_SET_NAME	VARCHAR(128)	character set name of the default character set for columns and domains in the schemata
SQL_PATH	VARCHAR(1024)	character representation of schema path specification
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	created time of the schema
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	last modified time of the schema
COMMENTS	VARCHAR(1024)	comments of the schema

SEQUENCES

Identify the external sequence generators defined in this catalog that are accessible to a given user or role.

표 9-174 Column 정보

Column name	Data type	Description
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence
SEQUENCE_NAME	VARCHAR(128)	sequence name
DATA_TYPE	VARCHAR(128)	the standard name of the data type
NUMERIC_PRECISION	NUMBER	the numeric precision of the numerical data type
NUMERIC_PRECISION_RADIX	NUMBER	the radix (2 or 10) of the precision of the numerical data type
NUMERIC_SCALE	NUMBER	the numeric scale of the exact numerical data type
START_VALUE	NUMBER	the start value of the sequence generator
MINIMUM_VALUE	NUMBER	the minimum value of the sequence generator
MAXIMUM_VALUE	NUMBER	the maximum value of the sequence generator
INCREMENT	NUMBER	the increment of the sequence generator
CYCLE_OPTION	BOOLEAN	cycle option
CACHE_SIZE	NATIVE_INTEGER	number of sequence numbers to cache
DECLARED_DATA_TYPE	VARCHAR(128)	the data type name that a user declared
DECLARED_NUMERIC_PRECISION	NUMBER	the precision value that a user declared
DECLARED_NUMERIC_SCALE	NUMBER	the scale value that a user declared
CREATED_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	created time of the sequence generator
MODIFIED_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	last modified time of the sequence generator
COMMENTS	VARCHAR(1024)	comments of the sequence generator

SQL_FEATURES

List the features and subfeatures of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation supports.

표 9-175 Column 정보

Column name	Data type	Description
FEATURE_ID	VARCHAR(32)	identifier string of the conformance element
FEATURE_NAME	VARCHAR(1024)	descriptive name of the conformance element
SUB_FEATURE_ID	VARCHAR(32)	identifier string of the subfeature, or a single space if not a subfeature
SUB_FEATURE_NAME	VARCHAR(1024)	descriptive name of the subfeature, or a single space if not a subfeature
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance element described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verified by testing, then the IS_VERIFIED_BY column shall contain information identifying the conformance test used to verify the conformance claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_IMPLEMENTATION_INFO

List the SQL-implementation information items defined in this ISO/IEC 9075 standard and, for each of the se, indicate the value supported by the SQL-implementation.

표 9-176 Column 정보

Column name	Data type	Description
IMPLEMENTATION_I NFO_ID	VARCHAR(32)	identifier string of the implementation information item
IMPLEMENTATION_I NFO_NAME	VARCHAR(1024)	descriptive name of the implementation information item
INTEGER_VALUE	NATIVE_INTEGER	value of the implementation information item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(32)	value of the implementation information item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation information it em

SQL_PACKAGES

List the packages of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation supports.

표 9-177 Column 정보

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the conformance element
NAME	VARCHAR(1024)	descriptive name of the conformance element
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance element described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verified by testing, then the IS_VERIFIED_BY column shall contain information identifying the conformance test used to verify the conformance claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_PARTS

List the parts of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation supports.

표 9-178 Column 정보

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the conformance element
NAME	VARCHAR(1024)	descriptive name of the conformance element
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance element described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verified by testing, then the IS_VERIFIED_BY column shall contain information identifying the conformance test used to verify the conformance claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_SIZING

List the sizing items of this ISO/IEC 9075 standard, for each of these, indicate the size supported by the SQL-implementation.

표 9-179 Column 정보

Column name	Data type	Description
SIZING_ID	NATIVE_INTEGER	identifier of the sizing item
SIZING_NAME	VARCHAR(1024)	descriptive name of the sizing item
SUPPORTED_VALUE	NATIVE_INTEGER	value of the sizing item, or 0 if the size is unlimited or cannot be determined, or null if the features for which the sizing item is applicable are not supported
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the sizing item

STATISTICS

Provide a list of statistics about a single table and the indexes associated with the table that are accessible to a given user or role.

표 9-180 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table
TABLE_OWNER	VARCHAR(128)	owner name of the table
TABLE_SCHEMA	VARCHAR(128)	schema name of the table
TABLE_NAME	VARCHAR(128)	table name of the table
STAT_TYPE	VARCHAR(32)	statistics type: the value in (TABLE STAT, INDEX CLUSTERED, INDEX HASHED, INDEX OTHER)
NON_UNIQUE	BOOLEAN	indicates whether the index does not allow duplicate values
INDEX_CATALOG	VARCHAR(128)	catalog name of the index
INDEX_OWNER	VARCHAR(128)	owner name of the index
INDEX_SCHEMA	VARCHAR(128)	schema name of the index
INDEX_NAME	VARCHAR(128)	name of the index
COLUMN_NAME	VARCHAR(128)	column name that participates in the index
ORDINAL_POSITION	NUMBER	ordinal position of the specific column in the index described
IS_ASCENDING_ORDER	BOOLEAN	index key column being described is sorted in ASCENDING(TRUE) or DESCENDING(FALSE) order
IS_NULLS_FIRST	BOOLEAN	the null values of the key column are sorted before(TRUE) or after(FALSE) non-null values
CARDINALITY	NUMBER	if STAT_TYPE is (TABLE TYPE), then this is the number of rows in the table; otherwise, it is the number of unique values in the index
PAGES	NUMBER	if STAT_TYPE is (TABLE TYPE), then this is the number of pages used for the table; otherwise, it is the number of pages used for the current index.
FILTER_CONDITION	VARCHAR(1024)	filter condition, if any.
COMMENTS	VARCHAR(1024)	if STAT_TYPE is (TABLE TYPE), then this is the table comments; otherwise, it is the index comments.

TABLES

Identify the tables defined in this catalog that are accessible to a given user or role.

표 9-181 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table
TABLE_OWNER	VARCHAR(128)	owner name of the table
TABLE_SCHEMA	VARCHAR(128)	schema name of the table
TABLE_NAME	VARCHAR(128)	table name of the table
TABLE_TYPE	VARCHAR(32)	the value is in (BASE TABLE, VIEW, GLOBAL TEMPORARY, LOCAL TEMPORARY, SYSTEM VERSIONED, FIXED TABLE, DUMP TABLE)
DBC_TABLE_TYPE	VARCHAR(32)	ODBC/JDBC table type: the value is in (TABLE, VIEW, GLOBAL TEMPORARY, LOCAL TEMPORARY, SYSTEM TABLE, ALIAS, SYNONYM)
TABLESPACE_NAME	VARCHAR(128)	tablespace name of the table, NULL if view
SYSTEM_VERSION_START_COLUMN_NAME	VARCHAR(128)	if the table is a system-versioned table, then the name of the system-version start column of the table
SYSTEM_VERSION_END_COLUMN_NAME	VARCHAR(128)	if the table is a system-versioned table, then the name of the system-version end column of the table
SYSTEM_VERSION_RETENTION_PERIOD	VARCHAR(32)	if the table is a system-versioned table, then the character representation of the value of the retention period of the table
SELF_REFERENCING_COLUMN_NAME	VARCHAR(128)	if the table is a typed table, then the name of the self-referencing column of the table
REFERENCE_GENERATION	VARCHAR(32)	if the table has a self-referencing column, the value is in (SYSTEM GENERATED, USER GENERATED, DERIVED)
USER_DEFINED_TYPE_CATALOG	VARCHAR(128)	if the table being described is a table of a structured type, the catalog name of the structured type
USER_DEFINED_TYPE_SCHEMA	VARCHAR(128)	if the table being described is a table of a structured type, the schema name of the structured type
USER_DEFINED_TYPE_NAME	VARCHAR(128)	if the table being described is a table of a structured type, the name of the structured type
IS_INSERTABLE_INTO	BOOLEAN	is an insertable-into table
IS_TYPED	BOOLEAN	is a typed table
COMMIT_ACTION	VARCHAR(32)	if the table is a temporary table, the value is in (DELETE, PRESERVE)
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	created time of the table
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	last modified time of the table
COMMENTS	VARCHAR(1024)	comments of the table

TABLE_CONSTRAINTS

Identify the table constraints defined on tables in this catalog that are accessible to a given user or role.

표 9-182 Column 정보

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	authorization name who owns the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint being described
CONSTRAINT_NAME	VARCHAR(128)	constraint name
TABLE_CATALOG	VARCHAR(128)	catalog name of the table to which the table constraint being described applies
TABLE_OWNER	VARCHAR(128)	authorization name who owns the table to to which the table constraint being described applies
TABLE_SCHEMA	VARCHAR(128)	schema name of the table to to which the table constraint being described applies
TABLE_NAME	VARCHAR(128)	table name of the table to to which the table constraint being described applies
CONSTRAINT_TYPE	VARCHAR(32)	the value is in (PRIMARY KEY, UNIQUE, FOREIGN KEY, NOT NULL, CHECK)
IS_DEFERRABLE	BOOLEAN	is a deferrable constraint
INITIALLY_DEFERRED	BOOLEAN	is an initially deferred constraint
ENFORCED	BOOLEAN	is an enforced constraint
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	created time of the constraint
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	last modified time of the constraint
COMMENTS	VARCHAR(1024)	comments of the constraint

TABLE_PRIVILEGES

Identify the privileges on tables defined in this catalog that are available to or granted by a given user or role.

표 9-183 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted table privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the table privilege being described is granted
TABLE_CATALOG	VARCHAR(128)	catalog name of the table on which the privilege being described was granted
TABLE_OWNER	VARCHAR(128)	table owner name of the table on which the privilege being described was granted
TABLE_SCHEMA	VARCHAR(128)	schema name of the table on which the privilege being described was granted
TABLE_NAME	VARCHAR(128)	table name on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (CONTROL, SELECT, INSERT, UPDATE, DELETE, REFERENCES, LOCK, INDEX, ALTER)
IS_GRANTABLE	BOOLEAN	is grantable
WITH_HIERARCHY	BOOLEAN	whether the privilege was granted WITH HIERARCHY OPTION or not

USAGE_PRIVILEGES

Identify the USAGE privileges on objects defined in this catalog that are available to or granted by a given user or role.

표 9-184 Column 정보

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted usage privileges, on the object of the type identified by OBJECT_TYPE
GRANTEE	VARCHAR(128)	authorization identifier of some user or role, or PUBLIC to indicate all users, to whom the usage privilege being described is granted
OBJECT_CATALOG	VARCHAR(128)	catalog name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_OWNER	VARCHAR(128)	owner name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_SCHEMA	VARCHAR(128)	schema name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_NAME	VARCHAR(128)	object name of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_TYPE	VARCHAR(32)	the value is in (DOMAIN, CHARACTER SET, COLLATION, TRANSLATION, SEQUENCE)
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (USAGE)
IS_GRANTABLE	BOOLEAN	is grantable

VIEWS

Identify the viewed tables defined in this catalog that are accessible to a given user or role.

표 9-185 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the viewed table
TABLE_OWNER	VARCHAR(128)	owner name of the viewed table
TABLE_SCHEMA	VARCHAR(128)	schema name of the viewed table
TABLE_NAME	VARCHAR(128)	view name of the viewed table
VIEW_DEFINITION	LONG VARCHAR	the character representation of the user-specified query expression contained in the corresponding view descriptor
CHECK_OPTION	VARCHAR(32)	the value is in (CASCADED, LOCAL, NONE)
IS_UPDATABLE	BOOLEAN	is an updatable view
INSERTABLE_INTO	BOOLEAN	is an insertable view
IS_TRIGGER_UPDATABLE	BOOLEAN	whether an update INSTEAD OF trigger is defined on the view or not
IS_TRIGGER_DELETABLE	BOOLEAN	whether a delete INSTEAD OF trigger is defined on the view or not
IS_TRIGGER_INSERTABLE_INTO	BOOLEAN	whether an insert INSTEAD OF trigger is defined on the view or not
IS_COMPILED	BOOLEAN	whether the view is compiled or not
IS_AFFECTED	BOOLEAN	whether the view is affected by modification of underlying object or not
COMMENTS	VARCHAR(1024)	comments of the view

VIEW_ROUTINE_USAGE

Identify each routine owned by a given user or role on which a view defined in this catalog is dependent.

표 9-186 Column 정보

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the viewed table
TABLE_OWNER	VARCHAR(128)	owner name of the viewed table
TABLE_SCHEMA	VARCHAR(128)	schema name of the viewed table
TABLE_NAME	VARCHAR(128)	view name of the viewed table
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of a routine contained in the query expression of the view being described
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of a routine contained in the query expression of the view being described
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of a routine contained in the query expression of the view being described
SPECIFIC_NAME	VARCHAR(128)	specific name of a routine contained in the query expression of the view being described

VIEW_TABLE_USAGE

Identify the tables on which viewed tables defined in this catalog and owned by a given user or role are dependent.

표 9-187 Column 정보

Column name	Data type	Description
VIEW_CATALOG	VARCHAR(128)	catalog name of the viewed table
VIEW_OWNER	VARCHAR(128)	owner name of the viewed table
VIEW_SCHEMA	VARCHAR(128)	schema name of the viewed table
VIEW_NAME	VARCHAR(128)	view name of the viewed table
TABLE_CATALOG	VARCHAR(128)	catalog name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_OWNER	VARCHAR(128)	owner name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_SCHEMA	VARCHAR(128)	schema name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_NAME	VARCHAR(128)	table name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described

9.3 PERFORMANCE_VIEW_SCHEMA

PERFORMANCE_VIEW_SCHEMA 스키마는 시스템의 현재 상태 정보를 조회할 수 있는 view들로 구성되어 있다.

해당 view들을 사용하려면 다음과 같이 PerformanceViewSchema.sql을 실행해야 한다.

- Standalone의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/PerformanceViewSchema
.sql
```

- Cluster의 경우

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.
.sql
```

PERFORMANCE_VIEW_SCHEMA의 view들은 startup 단계 (nomount, mount, open)별로 조회할 수 있는 정보가 다르다.

각각의 view를 어떤 시작 단계에서 조회할 수 있는지는 다음과 같은 질의를 통해 확인할 수 있다.

```
gSQL> select table_name, startup_phase from v$tables order by 1;
```

TABLE_NAME	STARTUP_PHASE
-----	-----
V\$AGABLE_INFO	OPEN
V\$ARCHIVELOG	MOUNT
V\$BACKUP	MOUNT
V\$BALANCER	OPEN
V\$COLUMNS	OPEN
V\$CONTROLFILE	MOUNT
V\$DATAFILE	MOUNT
V\$DB_FILE	MOUNT
V\$DISPATCHER	OPEN
V\$ERROR_CODE	NO_MOUNT
V\$INCREMENTAL_BACKUP	MOUNT
V\$INSTANCE	NO_MOUNT
V\$KEYWORDS	NO_MOUNT
V\$LATCH	NO_MOUNT
V\$LOCK_WAIT	OPEN
V\$LOGFILE	MOUNT
V\$PROCESS_MEM_STAT	NO_MOUNT
V\$PROCESS_SQL_STAT	NO_MOUNT

V\$PROCESS_STAT	NO_MOUNT
V\$PROPERTY	NO_MOUNT
TABLE_NAME	STARTUP_PHASE
-----	-----
V\$PSM_RESERVED_WORDS	NO_MOUNT
V\$QUEUE	OPEN
V\$RESERVED_WORDS	NO_MOUNT
V\$SESSION	NO_MOUNT
V\$SESSION_CONNECT_INFO	NO_MOUNT
V\$SESSION_EVENT	OPEN
V\$SESSION_MEM_STAT	NO_MOUNT
V\$SESSION_SQL_STAT	NO_MOUNT
V\$SESSION_STAT	NO_MOUNT
V\$SESSION_WAIT	OPEN
V\$SHARED_MODE	OPEN
V\$SHARED_SERVER	OPEN
V\$SHM_SEGMENT	NO_MOUNT
V\$SPROPERTY	NO_MOUNT
V\$SQLFN_METADATA	NO_MOUNT
V\$SQL_CACHE	NO_MOUNT
V\$SQL_COMMAND	OPEN
V\$SQL_HISTORY	NO_MOUNT
V\$STATEMENT	NO_MOUNT
V\$SYSTEM_EVENT	OPEN
TABLE_NAME	STARTUP_PHASE
-----	-----
V\$SYSTEM_MEM_STAT	NO_MOUNT
V\$SYSTEM_SQL_STAT	NO_MOUNT
V\$SYSTEM_STAT	NO_MOUNT
V\$TABLES	NO_MOUNT
V\$TABLESPACE	MOUNT
V\$TABLESPACE_STAT	OPEN
V\$TRANSACTION	OPEN
V\$WAIT_EVENT_CLASS_NAME	OPEN
V\$WAIT_EVENT_NAME	OPEN
V\$XA_TRANSACTION	OPEN

50 rows selected.

GV\$ Global View

Cluster에서는 거의 모든 V\$ view에 대응하는 GV\$ view를 제공한다.

V\$ view가 현재 접속한 서버의 정보를 조회하는 반면에 GV\$ view는 모든 서버의 정보를 조회한다.

GV\$ view는 V\$ view의 모든 column 정보를 포함하는데 이에 추가로 데이터를 획득한 서버 (cluster member)를 의미하는 ORIGIN_MEMBER_NAME column을 갖는다.

노트

Cluster에서만 사용할 수 있다.

예를 들어, V\$TRANSACTION 정보는 다음과 같이 현재 접속한 서버의 트랜잭션 정보를 보여준다.

```
gSQL> SELECT TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME FROM V$TRANSACTION;
TRANS_ID SESSION_ID TRANS_VIEW_SCN START_TIME
-----
40501296      48 1098.1.26      2017-04-07 17:14:01.912637
```

이에 반해, GV\$TRANSACTION 정보는 다음과 같이 모든 서버의 트랜잭션 정보를 보여준다.

```
gSQL> SELECT ORIGIN_MEMBER_NAME, TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME FROM GV$
TRANSACTION;
ORIGIN_MEMBER_NAME TRANS_ID SESSION_ID TRANS_VIEW_SCN START_TIME
-----
G1N1      40501296      48 1098.1.26      2017-04-07 17:14:01.912637
G2N2      40304688      48 1098.0.888      2017-04-07 17:14:55.134015
G2N1      42205232      48 1098.0.888      2017-04-07 17:14:55.135996
G1N2      40435760      48 1098.1.889      2017-04-07 17:14:01.910138
```

위의 예에서 ORIGIN_MEMBER_NAME 정보는 각 트랜잭션 정보를 G1N1, G2N1, G1N2, G2N2에 해당하는 cluster member로부터 획득했음을 알 수 있다.

특정 원격 서버에 대한 정보는 다음과 같이 ORIGIN_MEMBER_NAME column에 대한 조건을 사용하여 조회할 수 있다.

```
gSQL>
SELECT ORIGIN_MEMBER_NAME, TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME
FROM GV$TRANSACTION
WHERE ORIGIN_MEMBER_NAME IN ( 'G2N1', 'G3N2' );
```

ORIGIN_MEMBER_NAME	TRANS_ID	SESSION_ID	TRANS_VIEW_SCN	START_TIME
G3N2	32178224	48	1099.0.888	2017-04-07 17:33:10.934752
G2N1	42270768	48	1099.0.888	2017-04-07 17:31:14.726007

2 rows selected.

V\$AGABLE_INFO

The V\$AGABLE_INFO displays the system agable information.

표 9-188 Column 정보

Column name	Data type	Description
SCN	VARCHAR(32)	system scn
AGABLE_SCN	VARCHAR(32)	system agable scn
AGABLE_SCN_GAP	VARCHAR(32)	gap between system scn and agable scn
OLDEST_SESSION_ID	NUMBER	identifier of session blocking aging

V\$ARCHIVELOG

The V\$ARCHIVELOG displays information of log archiving.

표 9-189 Column 정보

Column name	Data type	Description
ARCHIVELOG_MODE	VARCHAR(32)	database log mode: the value in (NOARCHIVELOG, ARCHIVELOG)
LAST_ARCHIVED_LOG	NUMBER	sequence number of last archived log file
ARCHIVELOG_DIR	VARCHAR(1024)	archive destination path
ARCHIVELOG_FILE_PREFIX	VARCHAR(128)	file prefix name of the archived log

V\$AUDITABLE_DB_PRIVILEGES

The V\$AUDITABLE_DB_PRIVILEGES displays auditable database privileges.

표 9-190 Column 정보

Column name	Data type	Description
PRIVILEGE_ID	NUMBER	database privilege identifier
PRIVILEGE_NAME	VARCHAR(128)	database privilege name

V\$AUDITABLE_SYSTEM_ACTIONS

The V\$AUDITABLE_SYSTEM_ACTIONS displays auditable system actions.

표 9-191 Column 정보

Column name	Data type	Description
ACTION_ID	NUMBER	auditable system action identifier
ACTION_NAME	VARCHAR(128)	auditable system action name

V\$BACKUP

The V\$BACKUP displays information of backup.

표 9-192 Column 정보

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
BACKUP_STATUS	VARCHAR(16)	indicates whether the tablespace begin backup (ACTIVE) or not (I NACTIVE)
BACKUP_LSN	NUMBER	the last checkpoint lsn of tablespace when backup started

V\$BALANCER

The V\$BALANCER displays information of balancer.

표 9-193 Column 정보

Column name	Data type	Description
PROCESS_ID	NUMBER	balancer process identifier
CUR_CONNECTIONS	NUMBER	current number of connections
CONNECTIONS	NUMBER	total number of connections
CONNECTIONS_HIGHWATER	NUMBER	highest number of connections
MAX_CONNECTIONS	NUMBER	maximum connections
STATUS	VARCHAR(16)	status

V\$CLUSTER_DISPATCHER

The V\$CLUSTER_DISPATCHER displays cluster dispatcher information.

노트

Cluster에서만 사용할 수 있다.

표 9-194 Column 정보

Column name	Data type	Description
DISPATCHER_ID	NUMBER	dispatcher identifier
IS_SYNC	BOOLEAN	whether the dispatcher is sync or not
RX_BYTES	NUMBER	total amount of data that has received through the dispatcher
TX_BYTES	NUMBER	total amount of data that has transmitted through the dispatcher
RX_JOBS	NUMBER	the total number of jobs received
TX_JOBS	NUMBER	the total number of jobs transmitted

V\$CLUSTER_LOCATION

The V\$CLUSTER_LOCATION displays cluster location information.

노트

Cluster에서만 사용할 수 있다.

표 9-195 Column 정보

Column name	Data type	Description
MEMBER_NAME	VARCHAR(128)	member name
HOST	VARCHAR(128)	host address of a member
PORT	NUMBER	host port of a member

V\$CLUSTER_MEMBER

The V\$CLUSTER_MEMBER displays cluster member information.

노트

Cluster에서만 사용할 수 있다.

표 9-196 Column 정보

Column name	Data type	Description
MEMBER_ID	NUMBER	member identifier
MEMBER_POSITION	NUMBER	member position
STATUS	VARCHAR(64)	status of the member: the value in (ACTIVE, INACTIVE)
IS_GLOBAL_COORD	BOOLEAN	indicates whether a member is global coordnator (TRUE) or not (FALSE)
IS_GROUP_COORD	BOOLEAN	indicates whether a member is group coordnator (TRUE) or not (FALSE)

V\$COLUMNS

The V\$COLUMNS has one row for each column of all the performance views (views beginning with V\$).

V\$COLUMNS를 사용할 수 없는 nomount와 mount 단계에서 performance view의 column 정보를 조회하려면 아래 예제와 같이 \desc를 사용한다.

```
gSQL> \desc V$INSTANCE
COLUMN_NAME      TYPE                                IS_NULLABLE
-----
RELEASE_VERSION  VARCHAR(64)                        FALSE
STARTUP_TIME     TIMESTAMP(2) WITHOUT TIME ZONE    FALSE
INSTANCE_STATUS  VARCHAR(16)                        FALSE
```

표 9-197 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	owner name who owns the performance view
TABLE_SCHEMA	VARCHAR(128)	schema name of the performance view
TABLE_NAME	VARCHAR(128)	name of the performance view
COLUMN_NAME	VARCHAR(128)	column name
ORDINAL_POSITION	NUMBER	the ordinal position (> 0) of the column in the performance view
DATA_TYPE	VARCHAR(128)	the data type name that a user declared
DATA_PRECISION	NUMBER	the precision value that a user declared
DATA_SCALE	NUMBER	the scale value that a user declared
COMMENTS	VARCHAR(1024)	comments of the column

V\$CONTROLFILE

This view displays information about GOLDILOCKS control files.

표 9-198 Column 정보

Column name	Data type	Description
STATUS	VARCHAR(16)	control file status (VALID, CORRUPTED)
CONTROLFILE_NAME	VARCHAR(1152)	control file name (absolute path)
LAST_CHECKPOINT_LSN	NATIVE_BIGINT	the last checkpoint lsn
IS_PRIMARY	BOLLEAN	indicates whether the control file is primary

V\$DATAFILE

The V\$DATAFILE displays information of all datafiles.

표 9-199 Column 정보

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
DATAFILE_NAME	VARCHAR(1024)	datafile name (absolute path)
CHECKPOINT_LSN	NUMBER	LSN at last checkpoint (null if temporary tablespace)
CREATION_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	timestamp of the datafile creation
FILE_SIZE	NUMBER	datafile size (in bytes)
LOADED_CHECKPOINT_LSN	NUMBER	checkpoint LSN of the datafile loaded in memory
CORRUPT_PAGE_COUNT	NUMBER	number of corrupt pages in the datafile

V\$DB_FILE

The V\$DB_FILE displays a list of all files using in database.

표 9-200 Column 정보

Column name	Data type	Description
FILE_NAME	VARCHAR(1024)	file name
FILE_TYPE	VARCHAR(16)	file type

V\$DISPATCHER

The V\$DISPATCHER displays information of dispatchers.

표 9-201 Column 정보

Column name	Data type	Description
PROCESS_ID	NUMBER	dispatcher process identifier
RESPONSE_JOB_COUNT	NUMBER	response job count
ACCEPT	NUMBER	indicates whether this dispatcher is accepting new connections
START_TIME	NUMBER	process start time
CUR_CONNECTIONS	NUMBER	current number of connections
CONNECTIONS	NUMBER	total number of connections
CONNECTIONS_HIGHWATER	NUMBER	highest number of connections
MAX_CONNECTIONS	NUMBER	maximum connections
RECV_STATUS	VARCHAR(16)	receive status
RECV_BYTES	NUMBER	total bytes of received
RECV_UNITS	NUMBER	total units of received
RECV_IDLE	NUMBER	total idle time of receive (1/100 second)
RECV_BUSY	NUMBER	total busy time of receive (1/100 second)
SEND_STATUS	VARCHAR(16)	send status
SEND_BYTES	NUMBER	total bytes of sent
SEND_UNITS	NUMBER	total units of sent
SEND_IDLE	NUMBER	total idle time of send (1/100 second)
SEND_BUSY	NUMBER	total busy time of send (1/100 second)

V\$ERROR_CODE

The V\$ERROR_CODE displays a list of all GOLDDILOCKS error codes.

표 9-202 Column 정보

Column name	Data type	Description
ERROR_CODE	NUMBER	GOLDDILOCKS error code
SQL_STATE	VARCHAR(32)	standard SQLSTATE code
ERROR_MESSAGE	VARCHAR(1024)	error message

V\$GLOBAL_TRANSACTION

The V\$GLOBAL_TRANSACTION displays information on the currently active global transactions.

표 9-203 Column 정보

Column name	Data type	Description
GLOBAL_TRANS_ID	VARCHAR(1024)	global transaction identifier
LOCAL_TRANS_ID	NUMBER	local transaction identifier
GLOBAL_TRANS_STATE	VARCHAR(32)	state of the global transaction: the value in (NOTR, ACTIVE, I DLE, PREPARED, ROLLBACK_ONLY, HEURISTIC_COMPLETED)
ASSO_STATE	VARCHAR(32)	associate state of the global transaction: the value in (NOT_A SSOCIATED, ASSOCIATED, ASSOCIATION_SUSPENDED)
START_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	global transaction start time
IS_REPREPARABLE	BOOLEAN	indicates whether the global transaction is reparable

V\$INCREMENTAL_BACKUP

The V\$INCREMENTAL_BACKUP displays information about control files and datafiles in backup sets from the control file.

표 9-204 Column 정보

Column name	Data type	Description
BACKUP_NAME	VARCHAR(1024)	backup file name (absolute path)
BACKUP_SCOPE	VARCHAR(128)	incremental backup scope: the value in (database, tablespace, control)
INCREMENTAL_LEVEL	NUMBER	incremental backup level: the value in (0, 1, 2, 3, 4)
INCREMENTAL_TYPE	VARCHAR(32)	incremental backup type: the value in (DIFFERENTIAL, CUMULATIVE)
LSN	NUMBER	all changes up to checkpoint LSN are included in this backup
BEGIN_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	incremental backup beginning time
COMPLETION_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	incremental backup completion time

V\$INSTANCE

This view displays the state of the current instance.

노트

데이터베이스가 open 단계로 전이될 때 READ ONLY나 READ WRITE를 선택할 수 있는데, 만약 생략할 경우 DATABASE_ACCESS_MODE 프로퍼티에 설정된 값을 이용하여 DATA_ACCESS_MODE를 결정한다. 따라서 DATA_ACCESS_MODE는 nomount나 mount 단계에서 NONE으로 표시된다.

표 9-205 Column 정보

Column name	Data type	Description
RELEASE_VERSION	VARCHAR(64)	release version
STARTUP_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	time when the instance was started
INSTANCE_STATUS	VARCHAR(16)	status of the instance: the value in (STARTED, MOUNTED, OPEN)
DATA_ACCESS_MODE	VARCHAR(16)	data access mode of the instance: the value in (NONE, READ_ONLY, READ_WRITE)

V\$JOURNALING

The V\$JOURNALING displays journaling information.

노트

Cluster에서만 사용할 수 있다.

표 9-206 Column 정보

Column name	Data type	Description
TABLE_NAME	VARCHAR(128)	table name
SHARD_ID	NUMBER	shard identifier
RECORD_COUNT	NUMBER	journaled record count
TOTAL_SIZE	NUMBER	total size of journaled records (byte)

V\$KEYWORDS

The V\$KEYWORDS displays a list of all SQL keywords.

표 9-207 Column 정보

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword
IS_RESERVED	BOOLEAN	indicates whether the keyword cannot be used as an identifier (TRUE) or whether the keyword is not reserved (FALSE)

V\$LATCH

The V\$LATCH shows latch information.

표 9-208 Column 정보

Column name	Data type	Description
LATCH_DESCRIPTION	VARCHAR(64)	latch description
REF_COUNT	NUMBER	reference count
SPIN_LOCK	VARCHAR(3)	indicates whether the spin lock is locked (YES) or not (NO)
WAIT_COUNT	NUMBER	wait count
CURRENT_MODE	VARCHAR(32)	current latch mode: the value in (INITIAL, SHARED, EXCLUSIVE)

V\$LOGFILE

The V\$LOGFILE displays information of all redo log members.

표 9-209 Column 정보

Column name	Data type	Description
GROUP_ID	NUMBER	redo log group identifier
FILE_NAME	VARCHAR(1024)	name of the log member
GROUP_STATE	VARCHAR(32)	state of the log group: the value in (UNUSED, ACTIVE, CURRENT, IN ACTIVE)
FILE_SEQ	NUMBER	file sequence number of the log member
FILE_SIZE	NUMBER	file size of the log member (in bytes)

V\$LOCK_WAIT

This view lists the locks currently held and outstanding requests for a lock.

표 9-210 Column 정보

Column name	Data type	Description
GRANT_TRANS_ID	NUMBER	transaction identifier that holds the lock
REQUEST_TRANS_ID	NUMBER	transaction identifier that requests the lock

V\$PROCESS_STAT

The V\$PROCESS_STAT displays goldilocks process statistics.

표 9-211 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V\$PROCESS_MEM_STAT

The V\$PROCESS_MEM_STAT displays goldilocks process memory statistics.

표 9-212 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V\$PROCESS_SQL_STAT

The V\$PROCESS_SQL_STAT displays goldilocks process SQL statistics.

표 9-213 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V\$PROPERTY

The V\$PROPERTY displays a list of all properties at current session. Otherwise, the instance-wide value.

표 9-214 Column 정보

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	name of the property
DESCRIPTION	VARCHAR(2048)	description of the property
DATA_TYPE	VARCHAR(32)	data type of the property
STARTUP_PHASE	VARCHAR(32)	modifiable startup-phase: the value IN (NO MOUNT / MOUNT / OPEN & [BELOW ABOVE])
VALUE_UNIT	VARCHAR(32)	unit of the property value: the value in (NONE, BYTE, MS(milisc))
PROPERTY_VALUE	VARCHAR(2048)	property value for the session. otherwise, the instance-wide value
PROPERTY_SOURCE	VARCHAR(32)	source of the current property value: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
INIT_VALUE	VARCHAR(2048)	property init value for the session
INIT_SOURCE	VARCHAR(32)	source of the current property INIT_VALUE: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
MIN_VALUE	NUMBER	minimum value for property. null if type is varchar
MAX_VALUE	NUMBER	maximum value for property. null if type is varchar
SES_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SESSION or not: the value in (TRUE, FALSE)
SYS_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SYSTEM and when the change takes effect: the value in (NONE, FALSE, IMMEDIATE, DEFERRED)
IS_MODIFIABLE	VARCHAR(32)	property can be changed or not: the value in (TRUE, FALSE)

V\$PSM_RESERVED_WORDS

The V\$PSM_RESERVED_WORDS displays a list of all PSM reserved keywords. Reserved words cannot be used in variable name or procedure name.

표 9-215 Column 정보

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword

V\$QUEUE

The V\$QUEUE displays information of queue.

표 9-216 Column 정보

Column name	Data type	Description
TYPE	NUMBER	queue type (COMMON or DISPATCHER)
INDEX	NUMBER	index
QUEUED	NUMBER	number of items in the queue
WAIT	NUMBER	total time that all items in this queue have waited (1/100 second)
TOTALQ	VARCHAR(128)	total number of items that have ever been in the queue

V\$RESERVED_WORDS

The V\$RESERVED_WORDS displays a list of all SQL reserved keywords. Reserved words cannot be used in table name or column name.

표 9-217 Column 정보

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword

V\$SESSION

The V\$SESSION displays session information for each current session.

표 9-218 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
TRANS_ID	NUMBER	transaction identifier (-1 if inactive transaction)
CONNECTION_TYPE	VARCHAR(32)	connection type: the value in (DA, TCP)
USER_NAME	VARCHAR(128)	user name
SESSION_STATUS	VARCHAR(32)	status of the session: the value in (CONNECTED, SIGNED, SNIPED, DEAD)
SERVER_TYPE	VARCHAR(32)	server type: the value in (DEDICATED, SHARED)
PROCESS_ID	NUMBER	client process identifier
LOGON_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	logon time
PROGRAM_NAME	VARCHAR(128)	program name
CLIENT_ADDRESS	VARCHAR(1024)	client address (null if DA)
CLIENT_PORT	NUMBER	client port (0 if DA)
FAILOVER_TYPE	VARCHAR(13)	indicates whether and to what extent transparent application failover (TAF) is enabled for the session (NONE, SESSION)
FAILED_OVER	VARCHAR(3)	indicates whether the session is running in failover mode and failover has occurred (YES) or not (NO)
IS_AUDITED	VARCHAR(3)	indicates whether the session is audited (YES) or not (NO)

V\$SESSION_AUDIT

The V\$SESSION_AUDIT displays audited session information.

표 9-219 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
POLICY_NAME	VARCHAR(128)	active audit policy name
WHEN_SUCCESS	VARCHAR(3)	indicates whether the audit policy is enable for auditing successful events or not
WHEN_FAILURE	VARCHAR(3)	indicates whether the audit policy is enable for auditing unsuccessful events or not

V\$SESSION_CONNECT_INFO

The V\$SESSION_CONNECT_INFO displays information about network connections for the current session.

표 9-220 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
CLIENT_CHARSET	VARCHAR(40)	client character set

V\$SESSION_EVENT

The V\$SESSION_EVENT displays information on waits for an event by a session.

표 9-221 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	ID of the session
WAIT_EVENT_ID	NUMBER	Identifier of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	Name of the wait event
TOTAL_WAITS	NUMBER	Total number of waits for the event
TOTAL_TIMEOUTS	NUMBER	Total number of timeouts for the event
TIME_WAITED	NUMBER	Total amount of time waited for the event (microsecond)
AVERAGE_WAIT	NUMBER	Average amount of time waited for the event (microsecond)
MAX_WAIT	NUMBER	Maximum time waited for the event by the session (microsecond)
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V\$SESSION_STAT

The V\$SESSION_STAT displays session statistics.

표 9-222 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V\$SESSION_MEM_STAT

The V\$SESSION_MEM_STAT displays session memory statistics.

표 9-223 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V\$SESSION_SQL_STAT

The V\$SESSION_SQL_STAT displays session SQL statistics.

표 9-224 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V\$SESSION_WAIT

The V\$SESSION_WAIT displays the current or last wait for each session.

표 9-225 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	ID of the session
SEQ_NO	NUMBER	Identifier of the wait event
WAIT_EVENT_ID	NUMBER	Name of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	A number that uniquely identifies the current or last wait (incremented for each wait)
P1TEXT	VARCHAR(64)	Description of the first parameter for the wait event
P1	NUMBER	First wait event parameter (in decimal)
P1HEX	VARCHAR(32)	First wait event parameter (in hex)
P2TEXT	VARCHAR(64)	Description of the second parameter for the wait event
P2	NUMBER	Second wait event parameter (in decimal)
P2HEX	VARCHAR(32)	Second wait event parameter (in hex)
P3TEXT	VARCHAR(64)	Description of the third parameter for the wait event
P3	NUMBER	Third wait event parameter (in decimal)
P3HEX	VARCHAR(32)	Third wait event parameter (in hex)
STATE	VARCHAR(64)	Wait state
WAIT_TIME	NUMBER	If the session is currently waiting, then the value is time waited for the current wait. If the session is not in a wait, then the value is the duration of the last wait (in microseconds)
TIME_SINCE_LAST_WAIT	NUMBER	Time elapsed since the end of the last wait (in microseconds). If the session is currently in a wait, then the value is 0.
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V\$SHARED_MODE

The V\$SHARED_MODE displays information of shared mode.

표 9-226 Column 정보

Column name	Data type	Description
NAME	VARCHAR(128)	name
VALUE	VARCHAR(128)	value

V\$SHARED_SERVER

The V\$SHARED_SERVER displays information of shared servers.

표 9-227 Column 정보

Column name	Data type	Description
PROCESS_ID	NUMBER	shared server process identifier
PROCESSED_JOB_COUNT	NUMBER	processed job count
STATUS	VARCHAR(128)	status
IDLE	NUMBER	total idle time (1/100 second)
BUSY	NUMBER	total busy time (1/100 second)

V\$SHM_SEGMENT

The V\$SHM_SEGMENT displays a list of all shared memory segments.

표 9-228 Column 정보

Column name	Data type	Description
SHM_NAME	VARCHAR(32)	shared memory segment name
SHM_ID	NUMBER	shared memory segment identifier
SHM_SIZE	NUMBER	shared memory segment size (in bytes)
SHM_KEY	NUMBER	shared memory segment key
SHM_SEQ	NUMBER	shared memory segment sequence
SHM_ADDR	VARCHAR(32)	start address of the shared memory segment

V\$SPROPERTY

The V\$SPROPERTY displays a list of Properties. This is store a binary property file.

표 9-229 Column 정보

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	name of the property
DESCRIPTION	VARCHAR(2048)	description of the property
DATA_TYPE	VARCHAR(32)	data type of the property
STARTUP_PHASE	VARCHAR(32)	modifiable startup-phase: the value IN (NO MOUNT / MOUNT / OPEN & [BELOW ABOVE])
VALUE_UNIT	VARCHAR(32)	unit of the property value: the value in (NONE, BYTE, MS(milisc))
PROPERTY_VALUE	VARCHAR(2048)	property value stored in the binary property file
PROPERTY_SOURCE	VARCHAR(32)	source of the current property value: the value is BINARY_FILE
INIT_VALUE	VARCHAR(2048)	property init value for the system
INIT_SOURCE	VARCHAR(32)	source of the current property INIT_VALUE: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
MIN_VALUE	NUMBER	minimum value for property. null if type is varchar
MAX_VALUE	NUMBER	maximum value for property. null if type is varchar
SES_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SESSION or not: the value in (TRUE, FALSE)
SYS_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SYSTEM and when the change takes effect: the value in (NONE, FALSE, IMMEDIATE, DEFERRED)
IS_MODIFIABLE	VARCHAR(32)	property can be changed or not: the value in (TRUE, FALSE)

V\$SQLFN_METADATA

The V\$SQLFN_METADATA contains metadata about operators and built-in functions.

표 9-230 Column 정보

Column name	Data type	Description
FUNC_NAME	VARCHAR(128)	name of the built-in function
MINARGS	NUMBER	minimum number of arguments for the function
MAXARGS	NUMBER	maximum number of arguments for the function
IS_AGGREGATE	BOOLEAN	indicates whether the function is an aggregate function (TRUE) or not (FALSE)

V\$SQL_CACHE

The V\$SQL_CACHE lists statistics of shared SQL plan.

표 9-231 Column 정보

Column name	Data type	Description
SQL_HANDLE	NUMBER	SQL handle
HASH_VALUE	NUMBER	hash value of the SQL statement
REF_COUNT	NUMBER	count of prepared statements referencing the statement
PLAN_SIZE	NUMBER	the total plan size of the SQL statement (in bytes)
CLOCK_ID	NUMBER	clock identifier
PLAN_AGE	NUMBER	plan age
USER_NAME	VARCHAR(128)	user name
BIND_PARAM_COUNT	NUMBER	count of bind parameters
SQL_TEXT	LONG VARCHAR	SQL full text
PLAN_COUNT	NUMBER	physical plan count of the SQL statement
PLAN_ID	NUMBER	plan identifier
PLAN_SIZE	NUMBER	the total plan size of the SQL statement (in bytes)
PLAN_IS_ATOMIC	BOOLEAN	plan is atomic array insert or not
PLAN_TEXT	LONG VARCHAR	plan text for SQL statement

V\$SQL_COMMAND

The V\$SQL_COMMAND lists attribute information of each SQL command.

표 9-232 Column 정보

Column name	Data type	Description
COMMAND	VARCHAR(128)	SQL command
FROM_PHASE	VARCHAR(32)	executable from start-up phase
UNTIL_PHASE	VARCHAR(32)	executable until start-up phase
ACCESS_MODE	VARCHAR(32)	database access mode: values in (NONE, READ & WRITE, READ, READ & LOCK)
NEED_FETCH	VARCHAR(32)	the command is a query which has result set and need fetch
IS_DDL	VARCHAR(3)	the command is a DDL(Data Definition Language) or not
AUTO_COMMIT	VARCHAR(3)	the command is auto-commit or not
IS_CACHEABLE	VARCHAR(3)	the command is plan-cacheable or not
AUDIT_ACTION	VARCHAR(128)	auditable action name for the SQL command

V\$SQL_HISTORY

The V\$SQL_HISTORY displays information of SQLs.

표 9-233 Column 정보

Column name	Data type	Description
DRIVER_MEMBER_POS	NUMBER	driver member position
SESSION_ID	NUMBER	session identifier
START_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	statement start time
EXEC_TIME	NUMBER	execution time(us)
PREPARED	BOOLEAN	indicates whether the statement is prepared (YES) or not (NO)
SUCCESS	BOOLEAN	indicates whether the statement is success (YES) or not (NO)
STATUS	CHARACTER VARYING(16)	status of the statement: the value in (RUNNING, DONE)
SQL_TEXT	CHARACTER VARYING(1024)	first 1024 bytes of the SQL text for the statement

V\$STATEMENT

The V\$STATEMENT lists all statements.

표 9-234 Column 정보

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
STMT_ID	NUMBER	statement identifier in a session
STMT_VIEW_SCN	NUMBER	statement view scn
SQL_TEXT	VARCHAR(1024)	first 1024 bytes of the SQL text for the statement
START_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	statement start time

V\$SYSTEM_EVENT

The V\$SYSTEM_EVENT displays information on total waits for an event.

표 9-235 Column 정보

Column name	Data type	Description
WAIT_EVENT_ID	NUMBER	Identifier of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	Name of the wait event
TOTAL_WAITS	NUMBER	Total number of waits for the event
TOTAL_TIMEOUTS	NUMBER	Total number of timeouts for the event
TIME_WAITED	NUMBER	Total amount of time waited for the event (microsecond)
AVERAGE_WAIT	NUMBER	Average amount of time waited for the event (microsecond)
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V\$SYSTEM_STAT

The V\$SYSTEM_STAT displays system statistics.

표 9-236 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V\$SYSTEM_MEM_STAT

The V\$SYSTEM_MEM_STAT displays system memory statistics.

표 9-237 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V\$SYSTEM_SQL_STAT

The V\$SYSTEM_SQL_STAT displays system SQL statistics.

표 9-238 Column 정보

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V\$TABLES

The V\$TABLES contains the definitions of all the performance views (views beginning with V\$).

표 9-239 Column 정보

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	owner name who owns the performance view
TABLE_SCHEMA	VARCHAR(128)	schema name of the performance view
TABLE_NAME	VARCHAR(128)	name of the performance view
STARTUP_PHASE	VARCHAR(32)	visible startup phase of the performance view
CREATED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	available only in OPEN phase created time of the performance view
MODIFIED_TIME	TIMESTAMP(2) WITHOUT TIME ZONE	available only in OPEN phase modified time of the performance view
COMMENTS	VARCHAR(1024)	available only in OPEN phase comments of the performance view

V\$TABLESPACE

This view displays tablespace information.

표 9-240 Column 정보

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
TBS_ID	NUMBER	tablespace identifier
TBS_ATTR	VARCHAR(128)	tablespace attribute: the value in (device attribute (MEMORY) temporary attribute (TEMPORARY, PERSISTENT) usage attribute(DICT, UNDO, DATA, TEMPORARY))
IS_LOGGING	BOOLEAN	indicates whether the tablespace is a logging tablespace (YES) or not (NO)
IS_ONLINE	BOOLEAN	indicates whether the tablespace is ONLINE (YES) or OFFLINE (NO)
OFFLINE_STATE	VARCHAR(32)	indicates whether the tablespace can be taken online normally (CONSISTENT) or not (INCONSISTENT). null if the tablespace is ONLINE
EXTENT_SIZE	NUMBER	extent size of the tablespace (in bytes)
PAGE_SIZE	NUMBER	page size of the tablespace (in bytes)

V\$TABLESPACE_STAT

This view displays tablespace statistical information.

표 9-241 Column 정보

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
TBS_ID	NUMBER	tablespace identifier
TOTAL_EXT_COUNT	NUMBER	total extent count of the tablespace
USED_META_EXT_COUNT	NUMBER	meta extent count currently used on the tablespace
USED_DATA_EXT_COUNT	NUMBER	data extent count currently used on the tablespace
FREE_EXT_COUNT	NUMBER	free extent count of the tablespace
EXTENT_SIZE	NUMBER	extent size of the tablespace (in bytes)

V\$TRANSACTION

The V\$TRANSACTION lists the active transactions in the system.

표 9-242 Column 정보

Column name	Data type	Description
TRANS_ID	NUMBER	transaction identifier
SESSION_ID	NUMBER	session identifier (null if the global transaction is unassociated
TRANS_SLOT_ID	NUMBER	transaction slot identifier
PHYSICAL_TRANS_ID	NUMBER	physical transaction identifier
TRANS_STATE	VARCHAR(32)	transaction state: the value in (ACTIVE, BLOCK, PREPARE, COMMIT, ROLLBACK, IDLE, PRECOMMIT)
IS_GLOBAL	BOOLEAN	indicates whether the transaction is global or not
TRANS_ATTRIBUTE	VARCHAR(32)	transaction attribute: the value in (READ_ONLY, UPDATABLE, LOCK ABLE, UPDATABLE LOCKABLE)
ISOLATION_LEVEL	VARCHAR(32)	transaction isolation level: the value in (READ COMMITTED, SERIALI ZABLE)
TRANS_VIEW_SCN	NUMBER	transaction view scn
TCN	NUMBER	transaction change number
TRANS_SEQ	NUMBER	transaction sequence number
START_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	transaction start time

V\$WAIT_EVENT_CLASS_NAME

The V\$WAIT_EVENT_CLASS_NAME displays information about Class of wait event.

표 9-243 Column 정보

Column name	Data type	Description
CLASS_ID	NUMBER	Identifier of the class of the wait event
NAME	VARCHAR(64)	Name of the class of the wait event
DESCRIPTION	VARCHAR(128)	Description of the class of the wait event

V\$WAIT_EVENT_NAME

The V\$WAIT_EVENT_NAME displays information about wait events.

표 9-244 Column 정보

Column name	Data type	Description
CLASS_ID	NUMBER	Identifier of the wait event
NAME	VARCHAR(64)	Name of the wait event
DESCRIPTION	VARCHAR(128)	Description of the wait event
PARAMETER1	NUMBER	Description of the first parameter for the wait event
PARAMETER1	NUMBER	Description of the second parameter for the wait event
PARAMETER1	NUMBER	Description of the third parameter for the wait event
CLASS_ID	NUMBER	Identifier of the class of the wait event
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V\$XA_TRANSACTION

The V\$XA_TRANSACTION displays information on the currently active XA transactions.

표 9-245 Column 정보

Column name	Data type	Description
XA_TRANS_ID	VARCHAR(1024)	XA transaction identifier
LOCAL_TRANS_ID	NUMBER	local transaction identifier
XA_TRANS_STATE	VARCHAR(32)	state of the XA transaction: the value in (NOTR, ACTIVE, IDLE, PREPARED, ROLLBACK_ONLY, HEURISTIC_COMPLETED)
ASSO_STATE	VARCHAR(32)	associate state of the XA transaction: the value in (NOT_ASSOCIATED, ASSOCIATED, ASSOCIATION_SUSPENDED)
START_TIME	TIMESTAMP(2) WITH OUT TIME ZONE	XA transaction start time
IS_REPREPARABLE	BOOLEAN	indicates whether the XA transaction is reparable

10.

Server Property

10.1 Server Property 정보

Property는 다음 SQL 구문으로 변경할 수 있다.

- ALTER SYSTEM SET property_name
- ALTER SESSION SET property_name

Property 정보는 다음 view로 확인할 수 있다.

- V\$PROPERTY
- V\$SPROPERTY

본 매뉴얼의 property 기본 정보 각 항에 대한 설명은 다음과 같다.

항목	설명
이름	Property의 구분자이다.
요약	Property 요약 설명이다.
Data type	Property가 갖는 값의 데이터 타입이다.
적용단계	ALTER SYSTEM 또는 ALTER SESSION으로 변경할 수 있는 startup phase에 적용할 수 있다. • NONE: 적용할 수 있는 단계가 없다. (만약 변경 가능하지만 적용단계가 NONE인 경우에는 SCOPE = FILE 을 이용해야 한다.)
변경가능 여부	Property를 변경할 수 있는지 여부이다. • 해당 값이 TRUE일 경우, 변경 가능하다. • 해당 값이 FALSE일 경우, read-only만 가능하다.
ALTER SESSION 여부	ALTER SESSION SET property_name 구문으로 변경할 수 있는지 여부이다.
ALTER SYSTEM 여부	ALTER SYSTEM SET property_name 구문으로 변경할 수 있는지 여부이다. • IMMEDIATE: 수행 즉시 모든 SESSION에 변경된 값이 반영된다. • DEFERRED: 수행된 이후에 접속한 SESSION에만 변경된 값이 반영된다. 이미 접속된 SESSION에는 반영되지 않는다. • FALSE: 운영 중에는 변경된 값이 반영되지 않으며 restart 이후에 변경된 값이 반영된다. SCOPE=FILE로만 수행할 수 있다. • NONE: 변경할 수 없다.
MIN	데이터 타입이 BIGINT인 경우, property가 가질 수 있는 최소값이다. VARCHAR일 경우에는 N/A이다.
MAX	데이터 타입이 BIGINT인 경우, property가 가질 수 있는 최대값이다. VARCHAR일 경우에는 N/A이다.
기본값	해당 property가 갖는 기본값이다.

10.2 AGING_INTERVAL

기본 정보

항목	설명
이름	AGING_INTERVAL
요약	aging interval time(ms)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	100000000
기본값	10

설명

MVCC 기반의 database에서 이전 버전의 데이터를 지우는 ager thread가 처리할 job이 없을 때의 유틸 시간 (초)을 설정한다.

10.3 AGING_PLAN_INTERVAL

기본 정보

항목	설명
이름	AGING_PLAN_INTERVAL
요약	aging plan interval time(s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	31536000
기본값	0

설명

AGING_PLAN_INTERVAL 보다 오래된 SQL plan이 aging 대상이 된다.

10.4 ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10

기본 정보

항목	설명
이름	ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10
요약	archive log directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE(ARCHIVELOG_DIR_1), TRUE(ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10)
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/archive_log

설명

GOLDILOCKS 데이터베이스의 온라인 redo log file이 archive되는 디렉토리와 미디어 복구할 때 archive redo log file을 읽을 위치를 설정한다. 온라인 redo log file은 ARCHIVELOG_DIR_1에만 archive redo log file을 생성한다.

ARCHIVELOG_DIR_1은 시스템만 설정할 수 있고 ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10은 세션을 설정할 수 있다.

10.5 ARCHIVELOG_FILE

기본 정보

항목	설명
이름	ARCHIVELOG_FILE
요약	default archive log file
Data type	VARCHAR
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	archive

설명

온라인 redo log file을 archive 할 때 archive 디렉토리에 저장되는 목적 파일 이름의 prefix를 설정한다. Archive log file은 ARCHIVELOG_FILE에 설정된 prefix에 '_'와 파일 시퀀스, 'log' 확장자가 추가된 형태로 생성된다. 예를 들어, 파일 시퀀스가 0인 로그 파일은 'archive_0.log'으로 아카이빙된다.

10.6 ARCHIVELOG_MODE

기본 정보

항목	설명
이름	ARCHIVELOG_MODE
요약	archive log mode(0:disable, 1:enable)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	0

설명

Database를 생성할 때 적용되는 속성으로써 archive log mode를 다음 중 하나의 값으로 설정할 수 있다.

- 0: NOARCHIVELOG
- 1: ARCHIVELOG

Database가 생성된 후 운용되는 동안에는 archive log mode에 영향을 미치지 않고 mount 단계에서 ALTER DATABASE {ARCHIVELOG | NOARCHIVELOG}로 archive log mode를 변경할 수 있다.

10.7 BACKUP_DIR_1 ~ BACKUP_DIR_10

기본 정보

항목	설명
이름	BACKUP_DIR_1 ~ BACKUP_DIR_10
요약	backup directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE(BACKUP_DIR_1), TRUE(BACKUP_DIR_2 ~ BACKUP_DIR_10)
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/backup

설명

증분 백업이 수행될 때 백업 파일이 생성되고 증분 백업을 이용하여 파일을 복원할 때 백업 파일이 읽혀질 디렉토리를 설정한다. 증분 백업은 BACKUP_DIR_1에 설정된 디렉토리에만 생성된다.

BACKUP_DIR_1은 시스템만 설정할 수 있고 BACKUP_DIR_2 ~ BACKUP_DIR_10은 세션을 설정할 수 있다.

10.8 BLOCK_READ_COUNT

기본 정보

항목	설명
이름	BLOCK_READ_COUNT
요약	value count for a block read
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	65536
기본값	20

설명

SQL을 처리할 때 row의 묶음 단위인 BLOCK_READ_COUNT 단위로 row를 읽어 연산을 처리한다.

BLOCK_READ_COUNT는 연산을 수행할 때 한 번에 처리할 row의 개수를 의미하며 SQL 질의 처리에 참여하는 실행 노드간의 pipe-lining 처리의 기본 단위이다.

BLOCK_READ_COUNT 값이 크면 연산 처리 성능은 향상되지만 메모리 자원을 많이 사용한다. 따라서 10 ~ 100 사이의 값을 권장한다. 그 이상의 값을 사용하는 경우 자원 사용량은 비례하여 증가하지만 성능은 비례하여 향상되지 않는다.

10.9 BULK_IO_PAGE_COUNT

기본 정보

항목	설명
이름	BULK_IO_PAGE_COUNT
요약	page count for bulk IO operation
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	128
MAX	131072
기본값	3840

설명

서버를 재시작할 때 데이터 파일에 IO READ가 발생할 경우나 데이터 파일을 생성할 때 IO WRITE가 발생할 경우에 사용된다.

서버를 재시작하거나 데이터 파일을 생성할 때 $BULK\_IO\_PAGE\_COUNT * 8192$ 크기만큼 heap 메모리가 할당되며 세션의 PRIVATE_STATIC_AREA_SIZE가 그 크기보다 작을 경우 메모리 부족 에러가 발생할 수 있다. 이 경우에는 PRIVATE_STATIC_AREA_SIZE를 늘려주어야 한다.

10.10 CDISPATCHER_HOT_POLICY_INTERVAL

기본 정보

항목	설명
이름	CDISPATCHER_HOT_POLICY_INTERVAL
요약	cdispatcher dequeue interval for busy waiting (micro second)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	86400000000 (1day)
기본값	100000

설명

cdispatcher에서 dequeue 할 때 busy waiting하는 시간이다. Micro second 단위이며 이 값을 크게 하면 CPU를 많이 사용하는 대신 사용자 응답 시간 (latency)은 줄어든다.

10.11 CDISPATCHER_SOCKET_BUFFER_SIZE

기본 정보

항목	설명
이름	CDISPATCHER_SOCKET_BUFFER_SIZE
요약	cdispatcher socket buffer(sender, receiver) size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	64
MAX	100 Mega
기본값	32768

설명

cdispatcher socket buffer (송신자, 수신자)의 크기이다.

10.12 CDISPATCHER_SYNC_THREADS

기본 정보

항목	설명
이름	CDISPATCHER_SYNC_THREADS
요약	cdispatcher sync thread count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	32
기본값	1

설명

cdispatcher sync thread의 개수이다.

10.13 CDISPATCHER_THREADS

기본 정보

항목	설명
이름	CDISPATCHER_THREADS
요약	cdispatcher sender, receiver thread count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	32
기본값	1

설명

cdispatcher 송수신자의 thread 개수이다.

10.14 CHARACTER_SET

기본 정보

항목	설명
이름	CHARACTER_SET
요약	character set
Data type	VARCHAR
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	UTF8

설명

Database의 character set이다.

Database를 생성할 때 적용되는 속성으로써 다음 중 하나의 값을 설정할 수 있다.

Character set	설명
SQL_ASCII	ASCII standard
UTF8	Unicode, 8-bit
UHC	Unified Hangul code
GB18030	Chinese government standard

10.15 CHAR_LENGTH_UNITS

기본 정보

항목	설명
이름	CHAR_LENGTH_UNITS
요약	char length units
Data type	VARCHAR
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	OCTETS

설명

CHAR, VARCHAR와 같은 문자열 column을 정의하면서 다음과 같이 char length unit을 생략할 경우에 사용되는 char length units 값이다.

```
CREATE TABLE t1
(
  id   CHAR( 10 OCTETS ),
  name VARCHAR( 128 CHARACTERS ),
  addr VARCHAR( 128 )
);
```

- id CHAR(10 OCTETS)는 10 bytes를 의미한다.
- name VARCHAR(128 CHARACTERS)는 128 글자를 의미한다.
- addr VARCHAR(128)는 char length unit이 생략된 경우에 참조하는 프로퍼티 값이다.

Database 생성시 적용되는 속성으로 OCTETS나 CHARACTERS 중 하나의 값을 설정할 수 있다. OCTETS는 byte 수를 의미하고 CHARACTERS는 문자의 개수를 의미한다.

노트

SQL 표준은 기본값을 CHARACTERS로 정의하고 있으며 타 DBMS 들의 char length unit 기본값은 다음과 같다.

- Oracle과 DB2는 OCTETS를 사용한다.
- MS-SQL, MySQL, PostgreSQL는 CHARACTERS를 사용한다.

10.16 CHECK_DEDICATE_CONNECTION_INTERVAL

기본 정보

항목	설명
이름	CHECK_DEDICATE_CONNECTION_INTERVAL
요약	check dedicate socket
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000
기본값	1000

설명

C/S dedicate 환경에서 client가 접속을 강제로 종료했을 경우, 이를 검사하는 주기이다. Dedicate server (gserver)가 socket을 확인하여 끊어졌으면 종료한다. 기본값은 1,000 millisecond (1초)이다.

10.17 CLIENT_MAX_COUNT

기본 정보

항목	설명
이름	CLIENT_MAX_COUNT
요약	Maximum session count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	12
MAX	65535
기본값	128

설명

접속할 수 있는 세션의 최대 개수를 설정한다.

10.18 CLIENT_NUMA_POLICY

기본 정보

항목	설명
이름	CLIENT_NUMA_POLICY
요약	client numa policy(0: by modular, 1: by statistics , 2: by manual)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	2
기본값	0

설명

Client 프로세스들을 NUMA 노드들에 분배하기 위한 정책을 결정한다. CLIENT_NUMA_POLICY 프로퍼티는 NUMA 프로퍼티가 on 되어있을 때 동작한다.

- 0: 세션 ID를 모듈러 (modular)해서 연결할 NUMA 노드를 결정한다.
- 1: 통계정보를 바탕으로 가장 조금 연결되어 있는 NUMA 노드에 우선적으로 연결한다.
- 2: C/S client는 TCP_CLIENT_NUMA_NODE 프로퍼티에 의해서 결정되고, D/A client는 DA_CLIENT_NUMA_NODE 프로퍼티에 의해서 결정된다.

10.19 CLOSE_PSM_CHILD_STMTS

기본 정보

항목	설명
이름	CLOSE_PSM_CHILD_STMTS
요약	close child statements of PSM at the end of each execution
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

매 실행 마지막에 PSM의 child 구문을 close 한다.

10.20 CLUSTER_ASYNC_COMMIT

기본 정보

항목	설명
이름	CLUSTER_ASYNC_COMMIT
요약	enable asynchronous commit in cluster system
Data type	BOOLEAN
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	YES

설명

Cluster system에서 내부적으로 commit protocol을 async 모드로 처리할 것인지 여부를 설정한다.

노트

이 프로퍼티가 켜져 있으면 각 노드별로 commit을 비동기 처리하기 때문에 일시적으로 노드별 consistency가 깨어질 수 있다. 반면에 이 프로퍼티가 꺼져 있으면 commit 할 때마다 동기화하기 때문에 성능이 저하될 수 있다. 따라서 원하는 용도에 맞춰서 사용할 프로퍼티를 적절하게 설정해야 한다.

10.21 CLUSTER_ASYNC_REPLICATION

기본 정보

항목	설명
이름	CLUSTER_ASYNC_REPLICATION
요약	enable asynchronous replication
Data type	BOOLEAN
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	YES

설명

Cluster system에서 내부적으로 replication을 async 모드로 처리할 것인지 여부를 설정한다.

노트

이 프로퍼티가 켜져 있으면 각 노드별로 데이터를 비동기적으로 반영하기 때문에 어떤 노드에 접속하여 작업을 수행하는지에 따라 응답시간이 차이난다. 반면에 이 프로퍼티가 꺼져 있으면 데이터를 변경할 때마다 동기화하기 때문에 전체 성능이 저하될 수 있다. 따라서 원하는 용도에 맞춰서 사용할 프로퍼티를 적절하게 설정해야 한다.

10.22 CLUSTER_CM_BUFFER_COUNT

기본 정보

항목	설명
이름	CLUSTER_CM_BUFFER_COUNT
요약	communication buffer count for cluster
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	256
기본값	4

설명

Cluster의 communication buffer 개수이다.

10.23 CLUSTER_CM_BUFFER_SIZE

기본 정보

항목	설명
이름	CLUSTER_CM_BUFFER_SIZE
요약	communication buffer size for cluster
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	10 Mega
MAX	32 Giga
기본값	10 Mega

설명

Cluster의 communication buffer 크기이다.

10.24 CLUSTER_CM_READ_BUFFER_SIZE

기본 정보

항목	설명
이름	CLUSTER_CM_READ_BUFFER_SIZE
요약	communication read block size
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	8192
MAX	10485760
기본값	65536

설명

Communication read block의 크기이다.

10.25 CLUSTER_COMMIT_SLAVES

기본 정보

항목	설명
이름	CLUSTER_COMMIT_SLAVES
요약	number of commit slaves
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	8
기본값	1

설명

Commit slave의 번호이다.

10.26 CLUSTER_COMMIT_STREAM_ISOLATION

기본 정보

항목	설명
이름	CLUSTER_COMMIT_STREAM_ISOLATION
요약	isolate cluster dispatcher stream for commit protocol
Data type	BOOLEAN
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	0

설명

Cluster system에서 내부적으로 commit 처리 흐름을 다른 protocol 처리와 분리하여 수행할 것인지 여부를 설정한다. 시스템 환경에 따라 commit 처리를 분리할 경우 성능이 향상될 수 있다.

10.27 CLUSTER_CONNECTION

기본 정보

항목	설명
이름	CLUSTER_CONNECTION
요약	connection mode for cluster (socket:0, rdma:1)
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	0: socket

설명

Cluster의 connection mode이다. (socket: 0, rdma:1)

10.28 CLUSTER_CONNECTION_TIMEOUT_SEC

기본 정보

항목	설명
이름	CLUSTER_CONNECTION_TIMEOUT_SEC
요약	connection timeout for cluster
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	86400
기본값	5

설명

Cluster의 connection timeout 이다.

10.29 CLUSTER_DATA_SYNC_SERVERS

기본 정보

항목	설명
이름	CLUSTER_DATA_SYNC_SERVERS
요약	count of data synchronization server
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	128
기본값	3

설명

Data synchronization server의 개수이다.

10.30 CLUSTER_DISPATCHER_IN_QUEUE_SIZE

기본 정보

항목	설명
이름	CLUSTER_DISPATCHER_IN_QUEUE_SIZE
요약	in-queue size for cluster dispatcher
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1024
MAX	32768
기본값	1024

설명

Cluster dispatcher in-queue의 크기이다.

10.31 CLUSTER_DISPATCHER_NUMA_STREAM_MAP

기본 정보

항목	설명
이름	CLUSTER_DISPATCHER_NUMA_STREAM_MAP
요약	numa stream map for cluster dispatcher
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	'x': no binding

설명

Cluster 디스패처들이 연결될 NUMA 노드를 결정한다. CLUSTER_DISPATCHER_NUMA_STREAM_MAP 프로퍼티는 NUMA 프로퍼티가 on되어 있을 때 동작한다.

노트

만약 CLUSTER_COMMIT_STREAM_ISOLATION 프로퍼티가 on 되어 있다면 0번 스트림은 commit stream의 NUMA 노드로 설정된다.

다음은 디스패처가 세 개인 경우의 예이다. 0번 스트림은 NUMA 노드 0번에 연결하고 1번 스트림은 NUMA 노드 1번에 연결하며 2번 스트림은 NUMA 노드 2번에 연결한다.

```
CLUSTER_DISPATCHER_NUMA_STREAM_MAP = '0:1:2'
```

10.32 CLUSTER_DISPATCHER_OUT_QUEUE_SIZE

기본 정보

항목	설명
이름	CLUSTER_DISPATCHER_OUT_QUEUE_SIZE
요약	out-queue size for cluster dispatcher
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1024
MAX	32768
기본값	1024

설명

Cluster dispatcher의 out-queue 크기이다.

10.33 CLUSTER_HEARTBEAT_INTERVAL

기본 정보

항목	설명
이름	CLUSTER_HEARTBEAT_INTERVAL
요약	interval seconds for health checking of cluster
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	86400
기본값	3

설명

Cluster의 상태를 점검하는 주기 (초)이다. 0은 비활성화 상태를 의미한다.

10.34 CLUSTER_HEARTBEAT_RETRY_COUNT

기본 정보

항목	설명
이름	CLUSTER_HEARTBEAT_RETRY_COUNT
요약	retry count for health checking of cluster
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	65536
기본값	5

설명

Cluster 상태 점검을 다시 시도하는 횟수이다.

10.35 CLUSTER_IGNORE_INACTIVE_MEMBER

기본 정보

항목	설명
이름	CLUSTER_IGNORE_INACTIVE_MEMBER
요약	ignore in-active member for cluster
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	NO

설명

Cluster의 in-active 멤버를 무시한다.

10.36 CLUSTER_MAX_PACKET_SIZE

기본 정보

항목	설명
이름	CLUSTER_MAX_PACKET_SIZE
요약	maximum packet size for cluster session
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	10 Mega
MAX	32 Giga
기본값	100 Mega

설명

원격 프로토콜이 한 번에 전송할 수 있는 패킷의 최대 크기를 결정한다. 원격으로 전송해야 하는 column의 크기가 CLUSTER_MAX_PACKET_SIZE 프로퍼티 크기를 초과할 경우, 해당 프로퍼티를 column 크기보다 크게 설정해야 한다.

10.37 CLUSTER_MAX_PAYLOAD_SIZE

기본 정보

항목	설명
이름	CLUSTER_MAX_PAYLOAD_SIZE
요약	maximum packet payload size for cluster session (byte)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	524288
MAX	33554432
기본값	524288

설명

원격으로 전송되는 클러스터 패킷은 여러 개의 piece로 나뉘어 전달될 수 있는데 CLUSTER_MAX_PAYLOAD_SIZE 프로퍼티는 하나의 piece에 저장할 수 있는 데이터의 최대 크기를 설정한다.

10.38 CLUSTER_PACKET_ALLOCATION_TIMEOUT

기본 정보

항목	설명
이름	CLUSTER_PACKET_ALLOCATION_TIMEOUT
요약	a time limit (sec) for how long statemets will wait to allocate packet memory
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	100000000
기본값	3

설명

Cluster 패킷 구성에 필요한 메모리를 할당할 때 기다릴 수 있는 최대 시간 (초)을 설정한다.

10.39 CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT

기본 정보

항목	설명
이름	CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT
요약	a time limit of session fatal policy to wait for a response from the cluster protocol
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	0

설명

Cluster에서 protocol을 전송한 후에 응답이 올 때까지 최대로 대기하는 시간이다. 제한된 시간 내에 응답이 없을 경우 GOLDILOCKS는 protocol에 따라 session을 종료시키거나, 응답이 없는 원격 cluster member를 failover 시킨다. Session을 종료시키는 정책을 사용하려면 `CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT` property를 사용하여 제한 시간을 설정하고, failover 시키는 정책을 사용하려면 `CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT` property를 사용하여 제한 시간을 설정한다.

10.40 CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT

기본 정보

항목	설명
이름	CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT
요약	a time limit of failover policy to wait for a response from the cluster protocol
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1000000000
기본값	0

설명

Cluster에서 protocol을 전송한 후에 응답이 올 때까지 최대로 대기하는 시간이다. 제한된 시간 내에 응답이 없을 경우 GOLDILOCKS는 protocol에 따라 session을 종료시키거나, 응답이 없는 원격 cluster member를 failover 시킨다. Failover 시키는 정책을 사용하려면 `CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT` property를 사용하여 제한 시간을 설정하고, session을 종료시키는 정책을 사용하려면 `CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT` property를 사용하여 제한 시간을 설정한다.

10.41 CLUSTER_SERVER_RESPONSE_QUEUE_SIZE

기본 정보

항목	설명
이름	CLUSTER_SERVER_RESPONSE_QUEUE_SIZE
요약	response queue size for cluster server
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	30
MAX	32768
기본값	30

설명

원격 서버로부터 응답을 받기 위한 queue의 최대 크기를 설정한다.

10.42 CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY

기본 정보

항목	설명
이름	CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY
요약	split brain resolution policy for cluster system
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	2
기본값	0

설명

Cluster system에서 split-brain 상황을 해결하기 위한 정책을 설정한다. 1 이상의 값으로 설정할 경우 해결 방안을 Locator에게 질의한다.

노트

Locator에게 한 질의에 timeout이 발생하면 CLUSTER_SPLIT_BRAIN_RETRY_COUNT만큼 질의를 시도한다. 재시도에 실패하면 속성값이 1인 경우에는 failover를 강제로 진행하고 속성값이 2인 경우에는 fatal 종료한다.

10.43 CLUSTER_SPLIT_BRAIN_RETRY_COUNT

기본 정보

항목	설명
이름	CLUSTER_SPLIT_BRAIN_RETRY_COUNT
요약	retry count for split brain resolution policy(1 ~ 65536)
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	65536
기본값	4

설명

Cluster system에서 CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY가 1 이상으로 설정되었을 경우에 사용된다. Locator에게 보낸 질의에 응답이 없을 경우, 질의를 다시 시도하는 횟수를 설정한다.

10.44 COMMITTER_HOT_POLICY_INTERVAL

기본 정보

항목	설명
이름	COMMITTER_HOT_POLICY_INTERVAL
요약	committer deque interval for busy waiting
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	86400000000 (1 day)
기본값	0 (cold 정책)

설명

Commit cserver가 commit protocol 메시지를 읽기 위해 deque 할 때 busy waiting의 기준 시간 간격을 설정한다. 만약 1000000 (1초)로 설정할 경우, 이전 deque에 성공한 이후 다시 deque를 시도할 때까지 1 초를 경과하지 않았다면 deque에서의 대기시간 (timeout)을 0으로 설정하여 busy waiting 한다.

10.45 CONTROL_FILE_0 ~ CONTROL_FILE_7

기본 정보

항목	설명
이름	CONTROL_FILE_0
요약	control file name
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/wal/control_0.ctl

설명

Control file이 훼손되면 database를 사용할 수 없기 때문에 안정성을 위해 control file을 다중화해야 하는데 이 때 각 control file이 저장될 디렉토리와 파일 이름을 설정한다.

10.46 CONTROL_FILE_COUNT

기본 정보

항목	설명
이름	CONTROL_FILE_COUNT
요약	control file count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	2
MAX	8
기본값	2

설명

Control file이 훼손되면 database를 사용할 수 없기 때문에 안정성을 위해 control file을 다중화한다. CONTROL_FILE_COUNT는 control file의 다중화 개수를 설정하며 최소 두 개에서 최대 여덟 개까지 다중화할 수 있다.

10.47 CONTROL_FILE_TEMP_NAME

기본 정보

항목	설명
이름	CONTROL_FILE_TEMP_NAME
요약	temporary file name for control file
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/backup/control.tmp

설명

Database를 운용하는 중에 control file은 수시로 변경되고 필요할 경우 임시로 복사본을 만들 수도 있다. CONTROL_FILE_TEMP_NAME은 control file이 임시로 저장되는 디렉토리와 파일 이름을 설정한다.

10.48 COORDINATOR_COMMIT_WRITE_MODE

기본 정보

항목	설명
이름	COORDINATOR_COMMIT_WRITE_MODE
요약	coordinator commit write mode(0:disable, 1:wait mode)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

조정자 (coordinator)에 적용되는 commit write mode 이다. 만약 TRANSACTION_COMMIT_WRITE_MODE가 "no wait"이고 해당 프로퍼티가 "wait" 인 경우라면 조정자 노드는 "wait"으로 동작하고 그 외 노드들은 "no wait"으로 동작한다.

10.49 CSERVERS

기본 정보

항목	설명
이름	CSERVERS
요약	number of cserver processes
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	512
기본값	10

설명

cserver 프로세스의 번호이다.

10.50 DATABASE_ACCESS_MODE

기본 정보

항목	설명
이름	DATABASE_ACCESS_MODE
요약	database access mode (0: read only, 1: read write)
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	1

설명

Database를 시작할 때 접근 모드를 설정한다.

- 0: Database 조회만 할 수 있고 삽입/ 갱신/ 삭제는 불가능하다.
- 1: Database를 조회/ 삽입/ 삭제/ 갱신할 수 있다.

10.51 DATABASE_INSTANCE_NAME

기본 정보

항목	설명
이름	DATABASE_INSTANCE_NAME
요약	database instance name
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	GOLDILOCKS

설명

데이터베이스의 instance 이름이다.

10.52 DATA_STORE_MODE

기본 정보

항목	설명
이름	DATA_STORE_MODE
요약	Data store mode(cds:1,tds:2)
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	2
기본값	2

설명

Database의 저장 방식을 설정한다.

- 1: CDS 모드는 다중 사용자에게 대한 동시성은 지원하지만 영속성은 보장하지 않는다. 즉, data 삽입/ 삭제/ 갱신을 비롯하여 database를 변경하는 모든 연산에 대한 로그를 기록하지 않기 때문에 장애가 발생할 경우 복구할 수도 없다.
- 2: TDS 모드는 다중 사용자에게 대한 동시성 및 로그를 이용한 영속성을 보장한다.

10.53 DA_CLIENT_NUMA_NODE

기본 정보

항목	설명
이름	DA_CLIENT_NUMA_NODE
요약	numa node for DA clients
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	-1
MAX	63
기본값	-1

설명

Direct Access (D/A) 세션이 바인드 될 NUMA 노드 ID를 설정한다. DA_CLIENT_NUMA_NODE는 NUMA 프로퍼티가 on으로 설정된 경우에 작동한다.

10.54 DDL_AUTOCOMMIT

기본 정보

항목	설명
이름	DDL_AUTOCOMMIT
요약	DDL auto commit
Data type	BOOL
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

Autocommit이 적용되지 않는 DDL에 대한 autocommit 여부를 설정한다. 예를 들어, table의 생성과 변경에는 autocommit이 적용되지 않기 때문에 DDL_AUTOCOMMIT이 '0'인 경우 rollback을 수행하여 table 생성과 변경을 철회할 수 있다. 이에 반해 DDL_AUTOCOMMIT을 '1'로 설정하면 autocommit이 적용되지 않는 DDL들이 즉시 commit 된다.

10.55 DDL_LOCK_TIMEOUT

기본 정보

항목	설명
이름	DDL_LOCK_TIMEOUT
요약	a time limit (sec) for how long DDL statemets will wait
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	0

설명

동일한 객체에 대해 동시에 DDL이 발생했을 때의 lock 대기 시간을 의미한다.

기본값은 0 sec로써 DDL이 발생했을 때 lock을 기다리지 않는다.

다음과 같이 테이블에 대한 구조 변경 작업이 동시에 발생할 경우 다른 트랜잭션이 종료될 때까지 대기하지 않고 DDL_LOCK_TIMEOUT 시간만큼만 대기한다.

- Transaction A

```
ALTER TABLE t1 ADD COLUMN ( new_column NUMBER );
```

- Transaction B

```
TRUNCATE TABLE t1;
```

대기 시간이 DDL_LOCK_TIMEOUT을 초과할 경우 다음과 같은 에러가 발생한다.

```
gSQL> TRUNCATE TABLE t1;
ERR-HYT00(14026): resource busy or timeout expired
```

10.56 DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION

기본 정보

항목	설명
이름	DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION
요약	specifies whether or not create global secondary index at table creation
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	YES

설명

Cluster system에서 테이블을 생성할 때 global secondary index를 생성할지 여부를 설정한다. Global secondary index를 생성하지 않은 테이블에 대한 non-deterministic 질의는 실패한다. NO로 설정한 상태에서 테이블을 생성한 후에 별도로 global secondary index를 생성할 수도 있다.

10.57 DEFAULT_INDEX_LOGGING

기본 정보

항목	설명
이름	DEFAULT_INDEX_LOGGING
요약	default logging flag of indexes
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

인덱스를 생성할 때 사용자가 명시적으로 LOGGING 속성을 지정하지 않은 경우, LOGGING 속성은 DEFAULT_INDEX_LOGGING 프로퍼티 값으로 설정된다. 만약 인덱스가 LOGGING 테이블스페이스에 생성되면 반드시 LOGGING 속성이 설정되어야 한다

10.58 DEFAULT_INDEX_PCTFREE

기본 정보

항목	설명
이름	DEFAULT_INDEX_PCTFREE
요약	default pctfree value of indexes (%)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	99
기본값	0

설명

인덱스를 생성할 때 사용자가 명시적으로 PCTFREE 구문을 지정하지 않은 경우 PCTFREE는 DEFAULT_INDEX_PCTFREE 프로퍼티 값으로 설정된다.

10.59 DEFAULT_INITRANS

기본 정보

항목	설명
이름	DEFAULT_INITRANS
요약	default initrans value of tables
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	32
기본값	4

설명

테이블이나 인덱스를 생성할 때 사용자가 명시적으로 INITRANS 구문을 지정하지 않은 경우 INITRANS는 DEFAULT_INITRANS 프로퍼티 값으로 설정된다.

10.60 DEFAULT_MAXTRANS

기본 정보

항목	설명
이름	DEFAULT_MAXTRANS
요약	default maxtrans value of tables
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	32
기본값	8

설명

테이블이나 인덱스를 생성할 때 사용자가 명시적으로 MAXTRANS 구문을 지정하지 않은 경우 MAXTRANS는 DEFAULT_MAXTRANS 프로퍼티 값으로 설정된다.

10.61 DEFAULT_PCTFREE

기본 정보

항목	설명
이름	DEFAULT_PCTFREE
요약	default pctfree value of tables (%)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	99
기본값	10

설명

테이블을 생성할 때 사용자가 명시적으로 PCTFREE 구문을 지정하지 않은 경우 PCTFREE는 DEFAULT_PCTFREE 프로퍼티 값으로 설정된다.

10.62 DEFAULT_PCTUSED

기본 정보

항목	설명
이름	DEFAULT_PCTUSED
요약	default pctused value of tables (%)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	99
기본값	60

설명

테이블을 생성할 때 사용자가 명시적으로 PCTUSED 구문을 지정하지 않은 경우 PCTUSED는 DEFAULT_PCTUSED 프로퍼티 값으로 설정된다.

10.63 DEFAULT_REMOVAL_BACKUP_FILE

기본 정보

항목	설명
이름	DEFAULT_REMOVAL_BACKUP_FILE
요약	default removal flag of incremental backup files
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

백업 목록을 삭제할 때 백업 파일을 삭제할지 여부를 지정한다.

10.64 DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST

기본 정보

항목	설명
이름	DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST
요약	default removal flag of obsolete incremental backup lists
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

INCREMENTAL BACKUP을 수행할 때 obsolete 된 이전 백업 목록의 삭제 여부를 설정한다.

10.65 DEFAULT_SHARDING

기본 정보

항목	설명
이름	DEFAULT_SHARDING
요약	default sharding strategy (0: cloned, 1: hash sharding)
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

테이블을 생성할 때 sharding strategy를 정의하지 않을 경우 사용할 기본 sharding strategy를 설정한다.

다음은 CREATE TABLE 구문을 수행하는 예이다.

```
CREATE TABLE t1
(
  id    INTEGER,
  name  VARCHAR(128)
);
```

DEFAULT_SHARDING 값이 0 (cloned) 인 경우, 테이블을 다음과 같은 개념의 cloned table로 생성한다.

```
CREATE TABLE t1
(
  id    INTEGER,
  name  VARCHAR(128)
)
CLONED
AT CLUSTER WIDE
```

;

DEFAULT_SHARDING 값이 1 (hash sharding) 인 경우, 테이블을 다음과 같은 개념의 hash-sharded table 로 생성한다.

```
CREATE TABLE t1
(
  id    INTEGER,
  name  VARCHAR(128)
)
SHARDING BY HASH (id)
SHARD COUNT 24
AT CLUSTER WIDE
;
```

DEFAULT_SHARDING 값이 1 (hash sharding) 이고 <table sharding strategy>를 기술하지 않은 경우, 다음과 같은 순서로 hash sharding key를 결정한다.

1. PRIMARY KEY 제약 조건을 정의한 경우, primary key를 sharding key로 사용한다.

- 원본

```
CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(128) );
```

- 해석

```
CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(128) )
SHARDING BY HASH(id)
SHARD COUNT 24
AT CLUSTER WIDE;
```

2. UNIQUE 제약 조건을 정의한 경우, 첫 번째로 기술한 UNIQUE 제약 조건을 sharding key로 사용한다.

- 원본

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) UNIQUE );
```

- 해석

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) UNIQUE )
SHARDING BY HASH(name)
SHARD COUNT 24
AT CLUSTER WIDE;
```

3. Key 제약 조건이 없는 경우, 다음과 같은 data type이 아닌 첫 번째 column을 sharding key로 사용한다.

- 제외되는 data type: LONG VARCHAR, LONG VARBINARY, BOOLEAN

- 원본

```
CREATE TABLE t1 ( is_man BOOLEAN, id INTEGER, name VARCHAR(128) );
```

- 해석

```
CREATE TABLE t1 ( is_man BOOLEAN, id INTEGER, name VARCHAR(128) )  
    SHARDING BY HASH(id)  
    SHARD COUNT 24  
    AT CLUSTER WIDE;
```

10.66 DISABLE_DDL_CDC_GIVEUP

기본 정보

항목	설명
이름	DISABLE_DDL_CDC_GIVEUP
요약	disable DDL which caused CDC give-up
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

CDC의 give up에 영향을 미치는 supplemental log 대상 테이블에 대한 DDL 수행을 금지한다.
 관련 DDL은 표 44-1 DDL 구문에 따른 give up 발생 및 절차에 따른 허용 여부를 참조한다.

10.67 DISABLE_UPDATE_PK_CDC_GIVEUP

기본 정보

항목	설명
이름	DISABLE_UPDATE_PK_CDC_GIVEUP
요약	disable UPDATE primary key which caused CDC give-up
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

CDC give up을 유발한 UPDATE primary key를 비활성화 한다.

10.68 DISALLOWED_PROTOCOL_TARGETTYPE

기본 정보

항목	설명
이름	DISALLOWED_PROTOCOL_TARGETTYPE
요약	disallowed TARGETTYPE protocol
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

TARGETTYPE protocol을 허용하지 않는다.

10.69 DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL

기본 정보

항목	설명
이름	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL
요약	disallowed TARGETTYPE_WITH_ALL protocol
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

TARGETTYPE_WITH_ALL protocol을 허용하지 않는다.

10.70 DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME

기본 정보

항목	설명
이름	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME
요약	disallowed TARGETTYPE_WITH_NAME protocol
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

TARGETTYPE_WITH_NAME protocol을 허용하지 않는다.

10.71 DISPATCHER_CM_BUFFER_SIZE

기본 정보

항목	설명
이름	DISPATCHER_CM_BUFFER_SIZE
요약	communication buffer size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	10485760
MAX	34359738368
기본값	31457280

설명

Shared 모드에서 사용하는 전체 communication buffer 크기로서 Shared Static Area (SSA) 내에 할당되어 사용된다.

10.72 DISPATCHER_CM_UNIT_SIZE

기본 정보

항목	설명
이름	DISPATCHER_CM_UNIT_SIZE
요약	communication unit size
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1024
MAX	10485760
기본값	1024

설명

Shared 모드에서 dispatcher가 관리하는 unit의 크기이다. 이 크기가 크면 메모리가 낭비되고 이 크기가 작으면 성능이 저하될 수 있다.

Shared 모드에서 통신 packet의 최대 크기로 설정된다.

10.73 DISPATCHER_CONNECTIONS

기본 정보

항목	설명
이름	DISPATCHER_CONNECTIONS
요약	maximum number of connections for each dispatcher
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	10
MAX	32768
기본값	950

설명

Shared 모드에서 하나의 dispatcher가 관리할 수 있는 최대 connection (client)의 개수이다. 시스템에서 지원하는 최대값이 설정값보다 작으면 내부적으로 시스템 최대값으로 설정된다.

10.74 DISPATCHER_HOT_POLICY_INTERVAL

기본 정보

항목	설명
이름	DISPATCHER_HOT_POLICY_INTERVAL
요약	dispatcher dequeue interval for busy waiting
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	86400000000: 1day
기본값	100000: 0.1 second

설명

Busy waiting에 대한 dispatcher dequeue 주기이다. (micro second)

10.75 DISPATCHER_LOAD_BALANCING

기본 정보

항목	설명
이름	DISPATCHER_LOAD_BALANCING
요약	load balancing algorithm for shared mode (0: number of clients, 1: round robin)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

Shared 모드에서 client에 접속할 때 dispatcher를 할당하는 알고리즘이다.

- 0: 현재 연결된 client 수가 적은 dispatcher에 할당한다.
- 1: 순차적으로 dispatcher에 할당한다.

10.76 DISPATCHER_NUMA_STREAM_MAP

기본 정보

항목	설명
이름	DISPATCHER_NUMA_STREAM_MAP
요약	numa stream map for dispatcher
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	'x': no binding

설명

디스패처들이 연결될 NUMA 노드를 결정한다. 해당 프로퍼티는 NUMA 프로퍼티가 on으로 설정된 경우에 작동한다.

다음은 디스패처가 세 개인 경우의 예이다. 0번 스트림은 NUMA 노드 0번에 연결하고, 1번 스트림은 NUMA 노드 1번에 연결하고, 2번 스트림은 NUMA 노드 2번에 연결한다.

```
DISPATCHER_NUMA_STREAM_MAP = '0:1:2'
```

10.77 DISPATCHER_QUEUE_SIZE

기본 정보

항목	설명
이름	DISPATCHER_QUEUE_SIZE
요약	dispatcher queue size
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1024
MAX	32768
기본값	1024

설명

Shared 모드에서 dispatcher와 shared-server 간의 통신을 위한 queue 크기를 설정한다.

10.78 DISPATCHER_REQUEST_MINI_QUEUE_COUNT

기본 정보

항목	설명
이름	DISPATCHER_REQUEST_MINI_QUEUE_COUNT
요약	count of mini queue per request queue
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	16
기본값	4

설명

각 request queue의 mini queue 개수이다.

10.79 DISPATCHER_RESPONSE_MINI_QUEUE_COUNT

기본 정보

항목	설명
이름	DISPATCHER_RESPONSE_MINI_QUEUE_COUNT
요약	count of mini queue per response queue
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	16
기본값	4

설명

각 response queue의 mini queue 개수이다.

10.80 DISPATCHERS

기본 정보

항목	설명
이름	DISPATCHERS
요약	number of dispatcher processes
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	256
기본값	2

설명

Shared 모드를 사용할 때 dispatcher process 개수를 설정한다.
Open 단계에서는 alter system을 사용하여 값을 줄일 수 없다.

10.81 FETCH_FAILOVER

기본 정보

항목	설명
이름	FETCH_FAILOVER
요약	enable fetch failover
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	NO

설명

Fetch failover를 활성화한다.

10.82 GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY

기본 정보

항목	설명
이름	GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY
요약	allowed session dependent features in global connection
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	TRUE
MIN	0
MAX	1
기본값	NO

설명

Global connection에서 session dependent한 정보를 포함한 질의 수행 지원 여부를 설정한다.

10.83 GLOBAL_JOURNAL_BUFFER_SIZE

기본 정보

항목	설명
이름	GLOBAL_JOURNAL_BUFFER_SIZE
요약	global journal buffer size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1024
MAX	10 Giga
기본값	1 Mega

설명

Global journal의 buffer 크기이다.

10.84 GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE

기본 정보

항목	설명
이름	GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE
요약	global journal buffer total max size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1 Mega
MAX	100 Giga
기본값	64 Mega

설명

Global journal buffer의 최대 사이즈의 합이다.

10.85 GLOBAL_PROPERTY_LOCK_TIMEOUT

기본 정보

항목	설명
이름	GLOBAL_PROPERTY_LOCK_TIMEOUT
요약	a time limit(second) for how long global property lock statements will wait
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0 (infinite)
MAX	100000000
기본값	0

설명

Global property를 변경할 때 동시성을 제어하기 위해 lock 하는데 이 때 해당 lock 하기 위해 대기하는 시간을 설정한다.

10.86 GLOBAL_TRANSACTION_COMMIT_WRITE_MODE

기본 정보

항목	설명
이름	GLOBAL_TRANSACTION_COMMIT_WRITE_MODE
요약	global transaction commit write mode(0:no_wait, 1:wait, 2:disable)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	2
기본값	2

설명

Global transaction의 commit write mode를 변경하기 위한 프로퍼티이다. TRANSACTION_COMMIT_WRITE_MODE는 모든 트랜잭션들에 적용되는 반면에 이 프로퍼티는 global transaction에만 적용된다. 만약 해당 프로퍼티가 2로 설정되면 TRANSACTION_COMMIT_WRITE_MODE 값을 따른다.

- 0: no wait
- 1: wait
- 2: TRANSACTION_COMMIT_WRITE_MODE 값을 따른다.

10.87 GLOBAL_TRANSACTION_ISOLATION_SCOPE

기본 정보

항목	설명
이름	GLOBAL_TRANSACTION_ISOLATION_SCOPE
요약	isolation scope for global transaction(0:system, 1:group)
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

Transaction이 두 개 이상의 cluster group에 걸쳐 데이터를 변경한 경우 이를 global transaction으로 처리할지 아니면 다수의 domain transaction으로 처리할지 결정하는 프로퍼티이다.

- 0: Global transaction으로 처리
- 1: 다수의 domain transaction으로 처리

노트

이 프로퍼티가 1인 경우에는 cluster group마다 독립적인 트랜잭션으로 commit하기 때문에 트랜잭션 원자성 (transaction atomicity)을 보장하지 않는다.

10.88 GLOBAL_TRANSACTION_LOG_DIR

기본 정보

항목	설명
이름	GLOBAL_TRANSACTION_LOG_DIR
요약	default global transaction log directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/wal

설명

Global transaction log의 기본 directory 이다.

10.89 GLOBAL_TRANSACTION_LOG_FILE_SIZE

기본 정보

항목	설명
이름	GLOBAL_TRANSACTION_LOG_FILE_SIZE
요약	global transaction log file size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	20 Mega
MAX	10 Giga
기본값	100 Mega

설명

Global transaction log의 file 크기이다.

10.90 GMASTER_NUMA_NODE

기본 정보

항목	설명
이름	GMASTER_NUMA_NODE
요약	numa node for gmaster process
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	-1
MAX	63
기본값	-1

설명

gmaster 데몬이 사용할 NUMA node의 ID를 설정한다. GMASTER_NUMA_NODE 프로퍼티는 NUMA 프로퍼티가 on으로 설정된 경우에 작동한다.

10.91 GMON_AUTOSTART

기본 정보

항목	설명
이름	GMON_AUTOSTART
요약	Indicate whether gmon process automatically starts or not (0 1)
Data type	BOOLEAN
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	1

설명

gmon 프로세스를 자동으로 시작시킬지 여부를 설정한다.

10.92 HINT_ERROR

기본 정보

항목	설명
이름	HINT_ERROR
요약	enable hint error
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

Hint 구문에 대한 syntax 에러 및 validation 에러 체크 여부를 설정한다.

10.93 IDLE_TIMEOUT

기본 정보

항목	설명
이름	IDLE_TIMEOUT
요약	idle timeout(s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

C/S 세션에서 최대로 대기할 수 있는 IDLE 시간을 설정하며 해당 IDLE 시간을 초과하면 TIMEOUT 에러가 발생한다.

- 0: 무한 대기를 의미하며 TIMEOUT이 발생하지 않는다.

10.94 INDEX_BUILD_PARALLEL_FACTOR

기본 정보

항목	설명
이름	INDEX_BUILD_PARALLEL_FACTOR
요약	index build parallel factor
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	64
기본값	0

설명

인덱스를 생성할 때 병렬화 개수 (parallel factor)를 지정한다.

- 0: 시스템의 코어 개수로 지정된다.

10.95 INDEX_TREE_MERGE_PARALLEL_FACTOR

기본 정보

항목	설명
이름	INDEX_BUILD_PARALLEL_FACTOR
요약	parallel factor for merging sub-trees
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	64
기본값	0

설명

인덱스를 생성할 때 sub-tree를 합병하기 위한 병렬화 개수 (parallel factor)를 지정한다. 만약 해당 값이 INDEX_BUILD_PARALLEL_FACTOR 보다 큰 경우에는 INDEX_BUILD_PARALLEL_FACTOR를 사용한다.

- 0: INDEX_BUILD_PARALLEL_FACTOR를 따른다.

10.96 INST_ALLOCATOR_COUNT

기본 정보

항목	설명
이름	INST_ALLOCATOR_COUNT
요약	memory allocator count for instant tables or indexes
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	3
MAX	128
기본값	3

설명

인스턴트 블록을 할당 또는 삭제하는 연산의 병렬성을 높이기 위한 프로퍼티이다.

10.97 INST_TABLE_BLOCK_SIZE

기본 정보

항목	설명
이름	INST_TABLE_BLOCK_SIZE
요약	a block size of instant tables or indexes
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	8192
MAX	1048576
기본값	16384

설명

인스턴트 블록의 크기를 결정한다. 만약 인스턴트 레코드의 고정영역 크기가 인스턴트 블록의 크기를 초과하는 경우 다음과 같은 에러가 발생한다.

```
gSQL> SELECT DISTINCT * FROM T1, T1, T1, T1, T1, T1, T1, T1, T1, T1;
ERR-HY000(14098): maximum record length(16360) exceeds
```

10.98 IN_DOUBT_DECISION

기본 정보

항목	설명
이름	IN_DOUBT_DECISION
요약	decision for in-doubt transaction
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	2
기본값	2

설명

분산 트랜잭션의 in-doubt 트랜잭션을 commit 할지 rollback 할지 결정한다.

- 1: Commit
- 2: Rollback

10.99 JOURNAL_TEMP_DIR

기본 정보

항목	설명
이름	JOURNAL_TEMP_DIR
요약	journaling temporary directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/journal

설명

Journaling의 임시 디렉토리이다.

10.100 KEEPALIVE_IDLE_TIME

기본 정보

항목	설명
이름	KEEPALIVE_IDLE_TIME
요약	tcp keepalive idle time for checking dead client session (sec)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	16383
기본값	300

설명

Keep alive packet을 송신하기 전에 client와 server 간 TCP packet의 송수신없이 지속되는 시간 (idle) 이다. 즉, KEEPALIVE_IDLE_TIME에 설정된 초 동안 TCP packet 교환이 이루어지지 않으면 server 측에서 dead connection을 감지하기 위해 keep alive mechanism을 수행한다.

10.101 LOCAL_CLUSTER_MEMBER

기본 정보

항목	설명
이름	LOCAL_CLUSTER_MEMBER
요약	local cluster member name
Data type	VARCHAR
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	N/A
MAX	N/A
기본값	'G1N1'

설명

Local cluster member의 이름이다.

10.102 LOCAL_CLUSTER_MEMBER_HOST

기본 정보

항목	설명
이름	LOCAL_CLUSTER_MEMBER_HOST
요약	host name of local cluster member
Data type	VARCHAR
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	N/A
MAX	N/A
기본값	'127.0.0.1'

설명

Local cluster member의 host 이름이다.

10.103 LOCAL_CLUSTER_MEMBER_PORT

기본 정보

항목	설명
이름	LOCAL_CLUSTER_MEMBER_PORT
요약	listen port of local cluster member
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	1024
MAX	49151
기본값	10101

설명

Local cluster member의 listen port 이다.

10.104 LOCAL_JOURNAL_BUFFER_SIZE

기본 정보

항목	설명
이름	LOCAL_JOURNAL_BUFFER_SIZE
요약	local journal buffer size
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1024
MAX	10737418240 (10 Giga)
기본값	65536

설명

Local journal buffer의 크기이다.

10.105 LOCATION_FILE

기본 정보

항목	설명
이름	LOCATION_FILE
요약	location file name
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/wal/location.ctl

설명

Location file의 이름이다.

10.106 LOCATOR_QUERY_TIMEOUT

기본 정보

항목	설명
이름	LOCATOR_QUERY_TIMEOUT
요약	timeout for waiting locator response (sec)
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	20

설명

Cluster system이 split-brain 상황에 대한 해결 방안을 locator에게 질의한 후에 응답을 기다리는 시간 (초)을 설정한다. CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY를 1 이상으로 설정했을 때만 사용할 수 있는 프로퍼티이다.

10.107 LOCK_HASH_TABLE_SIZE

기본 정보

항목	설명
이름	LOCK_HASH_TABLE_SIZE
요약	lock manager hash table size (bucket 개수)
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	2
MAX	1000000
기본값	65519

설명

잠금 관리자 (lock manager)가 관리하는 hash table의 최대 크기를 설정한다.

10.108 LOG_BLOCK_SIZE

기본 정보

항목	설명
이름	LOG_BLOCK_SIZE
요약	log block size(byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	512
MAX	4096
기본값	512

설명

LOG_BLOCK_SIZE는 log buffer가 disk의 log file로 flush 되는 최소 크기이고 512, 1024, 2048, 4096 중 하나의 값으로 설정되어야 한다.

10.109 LOG_BUFFER_SIZE

기본 정보

항목	설명
이름	LOG_BUFFER_SIZE
요약	default log buffer size(byte)
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1048576
MAX	10737418240
기본값	10485760

설명

Database에서 DML 및 DDL 연산을 수행하여 생성한 redo log들은 공유 메모리 공간인 log buffer에 저장되고, LOG_BUFFER_SIZE를 참조하여 log buffer의 메모리 크기를 설정한다.

10.110 LOG_DIR

기본 정보

항목	설명
이름	LOG_DIR
요약	default log directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/wal

설명

Log buffer에 기록된 log는 database의 연속성을 보장하기 위해 비휘발성 저장 장치에 있는 log file로 flush 되고 LOG_DIR은 log file의 경로를 설정한다.

10.111 LOG_FILE_SIZE

기본 정보

항목	설명
이름	LOG_FILE_SIZE
요약	log file size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	20 Mbyte
MAX	60 Gbyte
기본값	100 Mbyte

설명

Database에서 사용되는 log file의 크기를 설정한다. 이는 database를 생성할 때만 참조되며 이후에는 log file size를 변경할 수 없다.

10.112 LOG_GROUP_COUNT

기본 정보

항목	설명
이름	LOG_GROUP_COUNT
요약	initial count of log group
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	4
MAX	254
기본값	4

설명

Database에서 사용되는 log group의 개수를 설정한다. 이는 database를 생성할 때만 참조되며 이후에는 영향을 미치지 않는다. Database를 생성한 후에 log group을 추가하거나 제거하는 기능은 별도의 구문으로 지원한다.

10.113 LOG_MIRROR_MODE

기본 정보

항목	설명
이름	LOG_MIRROR_MODE
요약	LogMirror Mode (1:Enable, 0:Disable)
Data type	BOOLEAN
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

데이터베이스를 시작할 때 redo log 복제 tool인 LogMirror를 운영할 때 필요한 shared memory를 구성하기 위한 프로퍼티이다.

LogMirror를 수행하려면 반드시 enable 되어야 한다.

Shared memory의 크기는 LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE 프로퍼티로 변경할 수 있다.

10.114 LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE

기본 정보

항목	설명
이름	LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE
요약	shared memory size for LogMirror (byte)
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	10485760 (10M)
MAX	1073741824 (1G)
기본값	104857600 (100M)

설명

Redo log 복제 tool인 LogMirror에 사용될 shared memory의 크기를 설정하는 프로퍼티이다.
LOG_MIRROR_MODE가 enable 된 상태에서에만 적용된다.

10.115 LOG_MIRROR_TIMEOUT

기본 정보

항목	설명
이름	LOG_MIRROR_TIMEOUT
요약	logmirror retry timeout (sec)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	0

설명

LogMirror의 응답을 기다리는 시간이다.

만약 0일 경우 무한정 대기하며 그렇지 않을 경우 설정한 값만큼 대기하다가 TIMEOUT이 발생하고 LogMirror service를 중단한다. 이 후 서버는 정상적으로 운영된다.

LOG_MIRROR_MODE가 enable 된 상태에서만 적용된다.

10.116 LOG_SYNC_INTERVAL

기본 정보

항목	설명
이름	LOG_SYNC_INTERVAL
요약	interval for synchronize log (s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	10000
기본값	3

설명

GOLDILOCKS의 log flusher는 log buffer의 내용을 disk log file로 flush하는 system thread이다. Log flusher가 유휴상태에서 깨어나면 flush해야 할 log가 있는지 확인하여 있을 경우 flush를 수행한다. 이 때 LOG_SYNC_INTERVAL에 설정된 시간 내에 flush를 하지 않았다면 현재 log buffer의 마지막 block까지 flush를 수행하여 log buffer와 log file을 동기화한다.

10.117 LOG_SYNC_INTERVAL_MSEC

기본 정보

항목	설명
이름	LOG_SYNC_INTERVAL_MSEC
요약	milli-second interval for synchronize log
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

Log를 동기화하는 millisecond 단위의 주기이다.

10.118 MAX_GROUP_COUNT

기본 정보

항목	설명
이름	MAX_GROUP_COUNT
요약	maximum group count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	8192
기본값	32

설명

그룹의 최대 개수이다.

10.119 MAX_JOURNAL_FILE_SIZE

기본 정보

항목	설명
이름	MAX_JOURNAL_FILE_SIZE
요약	maximum journal file size
Data type	BIGINT
적용단계	MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1099511627776 (1 Terabytes)
기본값	0 (no limit)

설명

Cluster system에서 journaling이 발생할 경우 내부적으로 journaling data를 저장할 global journaling file 의 최대 크기 (quota)를 설정한다.

10.120 MAX_NODE_COUNT

기본 정보

항목	설명
이름	MAX_NODE_COUNT
요약	maximum node count
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	1
MAX	8192
기본값	64

설명

최대 노드 개수이다.

10.121 MAXIMUM_CONCURRENT_ACTIVITIES

기본 정보

항목	설명
이름	MAXIMUM_CONCURRENT_ACTIVITIES
요약	maximum number of active statements that the driver can support for a connection
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	65535
기본값	1024

설명

동시에 수행 가능한 statement 개수를 설정한다.

10.122 MAXIMUM_FLANGE_COUNT

기본 정보

항목	설명
이름	MAXIMUM_FLANGE_COUNT
요약	maximum flange count in a plan clock
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	128
MAX	65535
기본값	1024

설명

Plan clock에서 확장할 수 있는 flange의 최대 개수이다.

10.123 MAXIMUM_FLUSH_LOG_BLOCK_COUNT

기본 정보

항목	설명
이름	MAXIMUM_FLUSH_LOG_BLOCK_COUNT
요약	maximum number of log block count to be flushing
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1000
MAX	2000000
기본값	100000

설명

Log buffer의 내용을 disk의 log file에 flush 할 때 한 번의 write 연산으로 flush 할 log block의 최대 개수를 설정한다.

10.124 MAXIMUM_FLUSH_PAGE_COUNT

기본 정보

항목	설명
이름	MAXIMUM_FLUSH_PAGE_COUNT
요약	maximum number of page count to be flushing
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	8192
기본값	1024

설명

GOLDILOCKS의 datafile은 checkpoint와 특정 DDL문에 의해 disk에 flush 된다. Datafile을 flush 하기 위해 한 번의 write 연산으로 flush 할 data page의 최대 개수를 설정한다.

10.125 MAXIMUM_JOURNAL_REPLAY_COUNT

기본 정보

항목	설명
이름	MAXIMUM_JOURNAL_REPLAY_COUNT
요약	maximum number of replaying journals for rebalance table
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	2
MAX	1024
기본값	2

설명

클러스터 환경에서 온라인 테이블 리밸런스를 DML과 병행하여 수행할 수 있는데 이 때 DML은 변경된 내용을 journal log로 남긴다. 테이블 리밸런스는 테이블을 동기화하는 동안 발생한 journal log를 1차로 반영하고, 다시 journal log를 반영하는 동안 누적된 journal log를 2차로 반영하는데, 이런 방식으로 최대 몇차까지 journal log를 반영할 것 인지를 설정한다.

10.126 MAXIMUM_NAMED_CURSOR_COUNT

기본 정보

항목	설명
이름	MAXIMUM_NAMED_CURSOR_COUNT
요약	maximum number of named cursor that the driver can support for a connection
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	100000
기본값	128

설명

하나의 session 내에서 사용할 수 있는 named cursor의 최대 개수이다.

다음과 같은 경우에 named cursor가 생성된다.

- SQLSetCursorName(), SQLGetCursorName() 함수 등을 이용해 named cursor를 선언한 경우

```
{
    ...
    SQLSetCursorName( stmt,
                      "my_cursor",
                      SQL_NTS );
    ...
}
```

- SQLExecDirect(), SQLPrepare() 함수 등에서 DECLARE cursor 구문을 사용한 경우

```
{
    ...
    SQLExecDirect( stmt,
                  "DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name",
```

```

        SQL_NTS );
    ...
}

```

- Embedded SQL에서 DECLARE cursor FOR UPDATE 구문을 사용한 경우

```

{
    ...
    EXEC SQL DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name FOR UPDATE;
    ...
    EXEC SQL OPEN my_cursor;
    ...
    EXEC SQL FETCH my_cursor INTO :data;
    EXEC SQL DELETE FROM tab_name WHERE CURRENT OF my_cursor;
    ...
    EXEC SQL CLOSE my_cursor;
}

```

노트

Embedded SQL에서 다음과 같이 FOR UPDATE가 없는 DECLARE CURSOR 구문은 session에 named cursor를 생성하지 않는다.

```

{
    ...
    EXEC SQL DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name;
    ...
}

```

10.127 MAXIMUM_SESSION_CM_BUFFER_SIZE

기본 정보

항목	설명
이름	MAXIMUM_SESSION_CM_BUFFER_SIZE
요약	maximum communication bytes per shared mode session
Data type	BIGINT
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1048576
MAX	1073741824
기본값	20971520

설명

Shared mode로 접속한 하나의 session에서 사용 가능한 최대 buffer size를 설정한다.
자세한 내용은 DISPATCHER_CM_BUFFER_SIZE를 참조한다.

10.128 MEASURE_CLUSTER_LATENCY

기본 정보

항목	설명
이름	MEASURE_CLUSTER_LATENCY
요약	measure cluster latency
Data type	BOOL
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

Measure cluster의 latency 이다.

10.129 MEMORY_MERGE_RUN_COUNT

기본 정보

항목	설명
이름	MEMORY_MERGE_RUN_COUNT
요약	merge run count for memory index
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	2
MAX	64
기본값	32

설명

Bottom-up 방식의 memory B-tree index는 테이블의 모든 key를 추출하여 특정 block size (MEMORY_SORT_RUN_SIZE) 단위로 정렬하고, 정렬된 block들을 병합한 후 internal node를 만드는 방식으로 생성된다. MEMORY_MERGE_RUN_COUNT는 한 번에 병합할 정렬된 block들의 개수를 설정한다.

10.130 MEMORY_SORT_RUN_SIZE

기본 정보

항목	설명
이름	MEMORY_SORT_RUN_SIZE
요약	sort run size for memory index(byte)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	8192
MAX	32768
기본값	8192

설명

Bottom-up 방식의 memory B-tree index는 테이블의 모든 key를 추출하여 특정 block size (MEMORY_SORT_RUN_SIZE) 단위로 정렬하고, 정렬된 block들을 병합한 후 internal node를 만드는 방식으로 생성된다. MEMORY_SORT_RUN_SIZE는 정렬할 block 한 개의 크기를 설정한다.

10.131 MINIMUM_UNDO_PAGE_COUNT

기본 정보

항목	설명
이름	MINIMUM_UNDO_PAGE_COUNT
요약	minimum undo page count
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	16
MAX	1048576
기본값	16

설명

DML은 이전 image를 저장하기 위해 undo page를 사용한다. DML당 undo segment를 하나씩 사용하여 undo page를 소모하는데, 만약 할당받은 undo segment의 page를 모두 소진하였을 경우 다른 undo segment의 page를 가져와서 사용할 수 있다. MINIMUM_UNDO_PAGE_COUNT는 undo page가 부족할 때 page를 가져올 undo segment를 찾기 위한 최소 undo page 수이다. 즉, undo page가 부족할 때, MINIMUM_UNDO_PAGE_COUNT 보다 많은 page를 보유한 undo segment에서만 page를 가져올 수 있다.

10.132 MIN_SAMPLE_ROW_COUNT

기본 정보

항목	설명
이름	MIN_SAMPLE_ROW_COUNT
요약	minimum sampling row count for analyze table
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	9223372036854775807 (INT64_MAX)
기본값	100000

설명

샘플링을 이용해서 **ANALYZE TABLE**을 수행할 때의 최소 샘플링 row 건수이다.

10.133 NET_BUFFER_SIZE

기본 정보

항목	설명
이름	NET_BUFFER_SIZE
요약	TCP network buffer size (byte)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	1024
MAX	1073741824
기본값	32768

설명

TCP 통신 buffer size를 설정한다.

Dedicated 모드에서는 통신 packet의 최대 크기로 설정된다.

Shared 모드에서는 DISPATCHER_CM_UNIT_SIZE가 사용된다.

10.134 NLS_DATE_FORMAT

기본 정보

항목	설명
이름	NLS_DATE_FORMAT
요약	nls date format
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	YYYY-MM-DD

설명

NLS_DATE_FORMAT은 TO_CHAR와 TO_DATE 함수의 default date format을 지정한다.

10.135 NLS_TIME_FORMAT

기본 정보

항목	설명
이름	NLS_TIME_FORMAT
요약	nls time format
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	HH24:MI:SS.FF6

설명

NLS_TIME_FORMAT은 TO_CHAR와 TO_TIME 함수의 default time format을 지정한다.

10.136 NLS_TIME_WITH_TIME_ZONE_FORMAT

기본 정보

항목	설명
이름	NLS_TIME_WITH_TIME_ZONE_FORMAT
요약	nls time with time zone format
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	HH24:MI:SS.FF6 TZH:TZM

설명

NLS_TIME_WITH_TIME_ZONE_FORMAT은 TO_CHAR와 TO_TIME_WITH_TIME_ZONE 함수의 default time with time zone format을 지정한다.

10.137 NLS_TIMESTAMP_FORMAT

기본 정보

항목	설명
이름	NLS_TIMESTAMP_FORMAT
요약	nls timestamp format
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	YYYY-MM-DD HH24:MI:SS.FF6

설명

NLS_TIMESTAMP_FORMAT은 TO_CHAR와 TO_TIMESTAMP 함수의 default timestamp format을 지정한다.

10.138 NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT

기본 정보

항목	설명
이름	NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT
요약	nls timestamp with time zone format
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	YYYY-MM-DD HH24:MI:SS.FF6 TZh:TzM

설명

NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT은 TO_CHAR와 TO_TIMESTAMP_WITH_TIME_ZONE 함수의 default timestamp with time zone format을 지정한다.

10.139 NUMA

기본 정보

항목	설명
이름	NUMA
요약	enable numa
Data type	BOOLEAN
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	0
MAX	1
기본값	NO

설명

NUMA를 활성화/ 비활성화한다.

노트

AIX에서 NUMA 속성을 사용하려면 사용자 계정을 변경해야 한다. 다음 명령을 루트 사용자로 실행한다.

```
# chuser "capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE" <username>
```

여기에서 <username>은 루트가 아닌 AIX 사용자 계정이다.

변경 사항을 적용하려면 로그아웃한 후에 다시 로그인해야 한다.

10.140 NUMA_MAP

기본 정보

항목	설명
이름	NUMA_MAP
요약	numa node map
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	'x': no binding

설명

시스템의 CPU core들을 NUMA 노드에 연결하기 위한 map을 설정한다. 해당 프로퍼티는 NUMA 프로퍼티가 on으로 설정된 경우에 작동한다.

다음은 시스템의 core가 네 개인 경우의 예이다.

- 0과 1은 NUMA 노드 0번에 연결하고, core 2와 3은 NUMA 노드 1번에 연결한다.

NUMA_MAP = '0:0:1:1' # core

- 0과 1은 NUMA 노드 0번에 연결하고, core 2와 3은 NUMA 노드 1번, core 1과 3을 NUMA 노드 2번에 연결한다.

NUMA_MAP = '0:0,2:1:1,2' # core

10.141 OFFLINE_MEMBER_AFTER_FAILOVER

기본 정보

항목	설명
이름	OFFLINE_MEMBER_AFTER_FAILOVER
요약	Aromatically offline member after failover
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	YES

설명

시스템의 백그라운드 프로세스 (gmaster)는 노드 장애에 따른 failover를 완료한 후에 장애 멤버를 자동으로 오프라인시킨다.

만약 NO로 설정되어서 장애 멤버가 오프라인되지 않았다면 장애 멤버를 시스템에 다시 조인시키기 전에 다음 구문을 실행해야 한다.

```
gSQL> ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS;
```

```
Database altered.
```

10.142 ONLINE_JOURNAL_REPLAY_THRESHOLD

기본 정보

항목	설명
이름	ONLINE_JOURNAL_REPLAY_THRESHOLD
요약	threshold bytes for replaying journals without table lock
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10737418240 (10G)
기본값	1048576 (1M)

설명

클러스터 환경에서 온라인 테이블 리밸런스는 수행 중에 발생한 DML이 남긴 journal log를 여러 번에 걸쳐 반영한다. Journal log를 반영하는 최대 차수는 MAXIMUM_JOURNAL_REPLAY_COUNT가 결정하지만 journal log를 반영할 때 남은 journal log의 양이 많지 않을 경우에는 굳이 최대 차수만큼 반복하지 않고 즉시 테이블에 EXCLUSIVE lock 하여 마지막 journal log를 반영하기 위한 journal log의 threshold 값으로 사용한다.

10.143 OS_GROUP_ACCESS

기본 정보

항목	설명
이름	OS_GROUP_ACCESS
요약	enable access database with OS group permission
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	NO

설명

동일한 group의 다른 user가 D/A로 접속하려면 이 설정을 YES로 변경해야 한다. 또한 시스템 상의 umask도 0002로 변경해야 한다.

10.144 PACKET_COMPRESSION_THRESHOLD

기본 정보

항목	설명
이름	PACKET_COMPRESSION_THRESHOLD
요약	The size limit at which packets are compressed(bytes)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	32
MAX	2113929216
기본값	2113929216

설명

클라이언트로 보낼 통신 데이터의 크기가 PACKET_COMPRESSION_THRESHOLD 보다 클 경우, 통신 데이터를 압축한다.

10.145 PAGE_CHECKSUM_TYPE

기본 정보

항목	설명
이름	PAGE_CHECKSUM_TYPE
요약	page checksum type (0:LSN, 1:CRC)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

Datafile의 각 page들에 대한 물리적 정합성을 보장하기 위해 checksum을 사용한다. GOLDILOCKS는 LSN, CRC 방식의 page checksum을 지원한다.

- 0: LSN
- 1: CRC

10.146 PARALLEL_IO_FACTOR

기본 정보

항목	설명
이름	PARALLEL_IO_FACTOR
요약	parallel load factor
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	16
기본값	1

설명

데이터베이스를 시작할 때 데이터 파일을 병렬 로딩하고 체크포인트 할 때 데이터 파일을 병렬 기록하기 위한 thread 개수를 설정한다.

10.147 PARALLEL_IO_GROUP_1 ~ PARALLEL_IO_GROUP_16

기본 정보

항목	설명
이름	PARALLEL_IO_GROUP_1
요약	parallel load group 1
Data type	VARCHAR
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/db

설명

Datafile의 병렬 IO를 위한 group directory를 설정한다. 즉, PARALLEL_IO_FACTOR 수만큼 group을 설정하여 각 group에 속한 datafile 별로 병렬 IO를 수행한다.

10.148 PARALLEL_LOAD_FACTOR

기본 정보

항목	설명
이름	PARALLEL_LOAD_FACTOR
요약	parallel load factor
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	64
기본값	1

설명

데이터베이스를 시작할 때 데이터 파일의 메모리를 적재한 후에 병렬 작업을 위한 thread 개수를 설정한다.

10.149 PENDING_LOG_BUFFER_COUNT

기본 정보

항목	설명
이름	PENDING_LOG_BUFFER_COUNT
요약	default pending log buffer count
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	32
기본값	4

설명

여러 트랜잭션이 동시에 실행될 경우 log buffer에 대한 경쟁을 줄이기 위해 pending log buffer를 사용하며, PENDING_LOG_BUFFER_COUNT는 동시에 사용할 수 있는 pending log buffer 개수를 설정한다.

10.150 PLAN_CACHE

기본 정보

항목	설명
이름	PLAN_CACHE
요약	caching sql plan
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	YES

설명

Plan cache 사용 여부를 설정한다.

10.151 PLAN_CACHE_SIZE

기본 정보

항목	설명
이름	PLAN_CACHE_SIZE
요약	sql plan cache size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	20971520
MAX	1099511627776
기본값	104857600

설명

Plan cache에 사용할 메모리 크기를 설정한다.

10.152 PRIVATE_STATIC_AREA_SIZE

기본 정보

항목	설명
이름	PRIVATE_STATIC_AREA_SIZE
요약	Private Static Area Size (byte)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	104857600
MAX	34359738368
기본값	104857600

설명

세션에서 할당할 수 있는 최대 heap 메모리 크기를 지정한다.

10.153 PROCESS_MAX_COUNT

기본 정보

항목	설명
이름	PROCESS_MAX_COUNT
요약	Process Max Count
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	12
MAX	65535
기본값	128

설명

시스템에서 사용할 수 있는 최대 프로세스 (thread) 개수를 지정한다.

시스템 프로세스 생성

- D/A 또는 C/S dedicated 모드로 접속할 때마다 프로세스가 생성된다.
- C/S shared 모드는 기본적인 balancer, dispatcher, shared-server가 프로세스이고 client에서 접속할 때는 프로세스가 생성되지 않는다.

10.154 QUERY_TIMEOUT

기본 정보

항목	설명
이름	QUERY_TIMEOUT
요약	query timeout (s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

세션에서 받은 명령어를 처리할 수 있는 최대 시간을 지정하며 만약 해당 시간을 초과하면 TIMEOUT 에러가 발생합니다.

- 0: 무한 대기를 의미하며 TIMEOUT 에러가 발생하지 않는다.

10.155 READABLE_ARCHIVELOG_DIR_COUNT

기본 정보

항목	설명
이름	READABLE_ARCHIVELOG_DIR_COUNT
요약	readable archive log directory count
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	10
기본값	1

설명

미디어 복구 시 archive redo log가 존재하는 디렉토리의 개수를 설정한다.

10.156 READABLE_BACKUP_DIR_COUNT

기본 정보

항목	설명
이름	READABLE_BACKUP_DIR_COUNT
요약	readable backup directory count
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	10
기본값	1

설명

중분 백업을 이용하여 파일을 복원할 때 중분 백업이 존재하는 디렉토리의 개수를 설정한다.

10.157 REBALANCE_BLOCK_READ_COUNT

기본 정보

항목	설명
이름	REBALANCE_BLOCK_READ_COUNT
요약	block read count for rebalance
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	DEFERRED
MIN	1
MAX	65536
기본값	100

설명

Rebalance 하기 위해 block을 읽어들이는 횟수이다.

10.158 RECOMPILE_CHECK_MINIMUM_PAGE_COUNT

노트

3.1 이후로 지원하지 않는다.

기본 정보

항목	설명
이름	RECOMPILE_CHECK_MINIMUM_PAGE_COUNT
요약	minimum page count for recompile check
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	10000
기본값	64

설명

Page count 변경에 의해 plan이 recompile 되었는지 여부를 체크하기 위해 minimum page count를 설정한다.

10.159 RECOMPILE_PAGE_PERCENT

노트

3.1 이후로 지원하지 않는다.

기본 정보

항목	설명
이름	RECOMPILE_PAGE_PERCENT
요약	recompile page percent
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1000
기본값	30

설명

Page count가 변경되어 plan을 recompile 할 때의 page percentage를 설정한다. 이 값이 0인 경우 page count 변경에 따른 recompile을 하지 않는다.

10.160 RECOVERY_LOG_BUFFER_SIZE

기본 정보

항목	설명
이름	RECOVERY_LOG_BUFFER_SIZE
요약	default log buffer size for recovery
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	786432
MAX	32 Mega
기본값	10 Mega

설명

복구를 위한 기본 log buffer 크기이다.

10.161 REDO_LOG_COMPRESSION_THRESHOLD

기본 정보

항목	설명
이름	REDO_LOG_COMPRESSION_THRESHOLD
요약	The size limit at which redo log are compressed(bytes)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	32
MAX	2113929216
기본값	256

설명

생성된 REDO LOG 의 크기가 REDO_LOG_COMPRESSION_THRESHOLD 값보다 클 경우, REDO LOG를 압축한다.

10.162 REFINE_RELATION

기본 정보

항목	설명
이름	REFINE_RELATION
요약	refine aged relations
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	YES

설명

이 속성을 NO로 하면 서버를 재시작할 때 REFINE RELATION 과정을 수행하지 않는다.

해당 프로퍼티는 REFINE RELATION 도중에 문제가 발생한 경우에 사용할 수 있으며 삭제되었지만 REFINE 하지 못한 RELATION (테이블이나 인덱스)들의 공간은 재사용할 수 없다. 문제를 해결한 이후 해당 프로퍼티를 YES로 설정하고 재시작하면 삭제하지 못했던 RELATION들에 대해 REFINE을 시도한다.

10.163 SESSION_FATAL_BEHAVIOR

기본 정보

항목	설명
이름	SESSION_FATAL_BEHAVIOR
요약	session fatal behavior
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

Session fatal이 발생할 때 fatal을 유발한 thread만 종료시킬지 아니면 프로세스 자체를 종료시킬지 결정한다.

- 0: Fatal을 유발한 thread만 종료한다.
- 1: 프로세스를 종료한다. 해당 프로세스 내에 다수의 세션이 동시에 수행되고 있다면 모든 세션들이 데이터베이스 사용을 끝낸 후에 프로세스를 종료한다.

10.164 SESSION_MEMORY_INIT_SIZE

기본 정보

항목	설명
이름	SESSION_MEMORY_INIT_SIZE
요약	initial memory size for session
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	131072 (128K)
MAX	1073741824 (1G)
기본값	131072 (128K)

설명

세션에서 사용할 공유 메모리를 미리 할당할 크기를 설정한다.

10.165 SESSION_MEMORY_SHRINK_THRESHOLD

기본 정보

항목	설명
이름	SESSION_MEMORY_SHRINK_THRESHOLD
요약	threshold bytes to attempt to shrink session memory allocator (byte)
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	131072 (128K)
MAX	1073741824 (1G)
기본값	131072 (128K)

설명

세션에서 사용한 동적 공유 메모리를 해제할 때 세션에서 사용하지 않는 동적 공유 메모리를 시스템에 반납할지 여부를 판단하기 위한 경계값을 설정한다. 즉, 사용하지 않는 메모리 중 설정된 값보다 큰 크기의 메모리 청크가 있으면 시스템에 반납한다.

10.166 SHARED_MEMORY_ADDRESS

기본 정보

항목	설명
이름	SHARED_MEMORY_ADDRESS
요약	shared memory address
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	1610612736

설명

Shared Static Area (SSA)의 주소를 지정한다.

10.167 SHARED_MEMORY_STATIC_KEY

기본 정보

항목	설명
이름	SHARED_MEMORY_STATIC_KEY
요약	Shared Memory Static KEY
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	542353

설명

서버를 구동할 때 Shared Static Area (SSA) 공간을 할당하면서 사용되는 shared memory key 값을 지정한다.

10.168 SHARED_MEMORY_STATIC_NAME

기본 정보

항목	설명
이름	SHARED_MEMORY_STATIC_NAME
요약	Shared Memory Static Name
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	_STATIC

설명

서버를 구동할 때 Shared Static Area (SSA) 공간을 할당하면서 사용되는 shared memory name을 지정한다.

10.169 SHARED_MEMORY_STATIC_SIZE

기본 정보

항목	설명
이름	SHARED_MEMORY_STATIC_SIZE
요약	Shared Memory Static Size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	104857600
MAX	1099511627776
기본값	629145600

설명

Shared Static Area (SSA)의 크기를 지정한다.

10.170 SHARED_REQUEST_QUEUE_COUNT

기본 정보

항목	설명
이름	SHARED_REQUEST_QUEUE_COUNT
요약	count of global request queue
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	1
MAX	16
기본값	1

설명

Shared 모드의 dispatcher에서 shared-server로 요청하는 queue 개수를 설정한다. 여러 dispatcher가 사용자의 작업 요청을 shared-server에 할당할 때 사용하는 queue로써 일반적으로 load-balance를 위해 하나를 사용한다. 그러나 dispatcher와 shared-server가 많아지면 queue에 경합이 발생하여 성능이 저하될 수 있으므로 이 값을 늘려서 사용한다. 이 값이 커지면 load-balance가 비효율적으로 될 수 있고 dead-lock이 발생할 가능성이 커진다.

10.171 SHARED_SERVERS

기본 정보

항목	설명
이름	SHARED_SERVERS
요약	number of shared-server processes
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	2048
기본값	10

설명

Shared 모드에서 shared-server process 개수를 설정한다.

Open 단계에서는 alter system으로 값을 줄일 수 없다.

10.172 SHARED_SESSION

기본 정보

항목	설명
이름	SHARED_SESSION
요약	to enable shared session
Data type	BOOL
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	YES

설명

Shared 모드를 활성화할지 여부를 설정한다. 이 값을 NO로 설정하면 load-balancer (gbalancer), dispatcher (gdi spatcher), shared-server (gserver)가 실행되지 않는다.

10.173 SNAPSHOT_STATEMENT_TIMEOUT

기본 정보

항목	설명
이름	SNAPSHOT_STATEMENT_TIMEOUT
요약	snapshot statement timeout (s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	22118400 (1 year)
기본값	22118400 (1 year)

설명

Snapshot read가 필요로 하는 statement의 최대 유지 시간을 설정한다. 설정된 시간을 초과한 snapshot statement들에는 TIMEOUT 에러가 발생한다.

10.174 SQL_HISTORY_SIZE

기본 정보

항목	설명
이름	SQL_HISTORY_SIZE
요약	history size for SQLs
Data type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	100000
기본값	0

설명

SQLs의 이력 (history) 크기이다.

10.175 SQL_HISTORY_TYPE

기본 정보

항목	설명
이름	SQL_HISTORY_TYPE
요약	history type for SQLs
Data Type	BIGINT
적용단계	NO MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	2
기본값	0

설명

SQLs의 이력 (history) 타입이다.

- 0: Direct-execute
- 1: Prepare-execute
- 2: All

10.176 SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY

기본 정보

항목	설명
이름	SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY
요약	supplemental log data of primary key columns be logged in redo log files
Data type	BOOLEAN
적용단계	MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

Database 내의 모든 변경 내용에 대한 supplemental log를 기록한다.

10.177 SYSTEM_LOGGER_DIR

기본 정보

항목	설명
이름	SYSTEM_LOGGER_DIR
요약	system logger directory
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/trc

설명

Trace 로그 메시지가 기록되는 디스크 경로를 지정한다.

10.178 SYSTEM_MEMORY_AUX_TABLESPACE_SIZE

기본 정보

항목	설명
이름	SYSTEM_MEMORY_AUX_TABLESPACE_SIZE
요약	default system memory auxiliary tablespace size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	32 Mega
MAX	30 Giga
기본값	200 Mega

설명

데이터베이스를 생성할 때 초기 MEM_AUX_TBS 테이블스페이스 크기를 결정한다.

10.179 SYSTEM_MEMORY_DATA_TABLESPACE_SIZE

기본 정보

항목	설명
이름	SYSTEM_MEMORY_DATA_TABLESPACE_SIZE
요약	default system memory data tablespace size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	32 Mega
MAX	30 Giga
기본값	200 Mega

설명

데이터베이스를 생성할 때 초기 MEM_DATA_TBS 테이블스페이스 크기를 결정한다.

10.180 SYSTEM_MEMORY_DICT_TABLESPACE_SIZE

기본 정보

항목	설명
이름	SYSTEM_MEMORY_DICT_TABLESPACE_SIZE
요약	default dictionary tablespace size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	256 Mega
MAX	30 Giga
기본값	256 Mega

설명

데이터베이스를 생성할 때 초기 DICTIONARY_TBS 테이블스페이스의 크기를 결정한다.

10.181 SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE

기본 정보

항목	설명
이름	SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE
요약	default system memory temporary tablespace size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	32 Mega
MAX	30 Giga
기본값	200 Mega

설명

데이터베이스를 생성할 때 초기 MEM_TEMP_TBS 테이블스페이스의 크기를 결정한다.

10.182 SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE

기본 정보

항목	설명
이름	SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE
요약	default system memory undo tablespace size (byte)
Data type	BIGINT
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	NONE
MIN	32 Mega
MAX	30 Giga
기본값	32 Mega

설명

데이터베이스를 생성할 때 초기 MEM_UNDO_TBS 테이블스페이스의 크기를 결정한다.

10.183 SYSTEM_TABLESPACE_DIR

기본 정보

항목	설명
이름	SYSTEM_TABLESPACE_DIR
요약	system tablespace directory
Data type	VARCHAR
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	N/A
MAX	N/A
기본값	<GOLDILOCKS_DATA>/db

설명

데이터베이스를 생성할 때 초기 시스템 테이블스페이스들이 저장되는 경로를 지정한다.

10.184 SYSTEM_UDS_DIR

기본 정보

항목	설명
이름	SYSTEM_UDS_DIR
요약	system unix domain socket directory
Data type	VARCHAR
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	'/tmp'

설명

Unix domain socket 파일이 생성되는 directory를 설정한다.

DB system 이외에 glsnr 등과 같은 unix domain socket에 대한 디렉토리 설정은 별도의 configuration file에서 관리된다.

최대 설정 크기는 60 byte이다. (Unix domain socket 파일의 절대 경로 (디렉토리 + 파일 이름)의 최대 크기는 OS마다 다르지만 보통 100 byte 내외이다.)

10.185 TCP_CLIENT_NUMA_NODE

기본 정보

항목	설명
이름	TCP_CLIENT_NUMA_NODE
요약	numa node for TCP clients
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	-1
MAX	63
기본값	-1

설명

Client server 세션이 바인드 될 NUMA 노드 ID를 설정한다. 해당 프로퍼티는 NUMA 프로퍼티가 on으로 설정된 경우에 작동한다.

10.186 TCP_NODELAY

기본 정보

항목	설명
이름	TCP_NODELAY
요약	no delays in buffer flushing within the TCP/IP protocol stack
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	1
기본값	YES

설명

C/S 방식 (TCP socket)으로 client에 data 전송할 때의 socket TCP_NODELAY 옵션을 설정한다.
빠른 latency가 필요하지 않고 network 부하를 줄이고 싶은 경우에는 NO로 설정한다.

10.187 TEMP_SEGMENT_CACHE_SIZE

기본 정보

항목	설명
이름	TEMP_SEGMENT_CACHE_SIZE
요약	the number of segments to be cached for global temporary tables and indexes in each session
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	DEFERRED
MIN	0
MAX	4294967295
기본값	3

설명

Global temporary table이나 global temporary index segment가 drop 될 때 tablespace에 반납하지 않고 session에서 caching 할 segment 개수를 지정한다. Segment cache에 존재하는 segment는 향후 global temporary table이나 global temporary index에서 재사용된다.

- 0: Session에서 global temporary table이나 global temporary index의 segment cache를 사용하지 않는다.
- 1 ~ 4294967295: Session에서 global temporary table이나 global temporary index의 segment cache를 주어진 개수만큼 유지한다.

10.188 TEMP_UNDO_ENABLED

기본 정보

항목	설명
이름	TEMP_UNDO_ENABLED
요약	enables writing undo records of global temporary tables to the temp table space
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

Global temporary table에 대한 undo 레코드의 로깅 위치를 지정한다.

- 0 (FALSE): 데이터베이스의 기본 undo tablespace에 undo 레코드를 기록한다.
- 1 (TRUE): 데이터베이스의 기본 temporary tablespace에 undo 레코드를 기록한다.

10.189 TIMED_STATISTICS

기본 정보

항목	설명
이름	TIMED_STATISTICS
요약	timed statistics
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	2
기본값	0

설명

Wait event를 측정하는지 여부이다.

v\$sqlsystem_event, v\$sqlsession_event, v\$sqlsession_wait table에 wait event와 관련된 통계 기록을 남기고 싶은 경우 설정한다.

- 0: 통계 기록을 남기지 않는다.
- 1: 통계 기록을 남긴다.
- 2: High precision timer를 이용하여 통계 기록을 남긴다.

10.190 TIMEZONE

기본 정보

항목	설명
이름	TIMEZONE
요약	timezone
Data type	VARCHAR
적용단계	NONE
변경가능 여부	FALSE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	N/A
MAX	N/A
기본값	+09:00

설명

Database의 time zone 값이다.

Database가 생성될 때 적용되는 속성으로써 -14:00 ~ +14:00 범위의 값을 사용할 수 있다.

10.191 TRACE_ALTER_SYSTEM

기본 정보

항목	설명
이름	TRACE_ALTER_SYSTEM
요약	write trace messages for ALTER SYSTEM
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

ALTER SYSTEM 구문을 실행할 때 해당 SQL 문장을 trace file (<GOLDILOCKS_DATA>/trc/system.trc)에 기록한다.

시스템 변경에 대한 기록을 남기려면 TRACE_ALTER_SYSTEM 프로퍼티를 ON으로 설정한다.

TRACE_ALTER_SYSTEM 프로퍼티는 SELECT 질의, INSERT, UPDATE, DELETE 등의 구문 수행과는 무관하므로 성능에 영향을 주지 않는다.

10.192 TRACE_DDL

기본 정보

항목	설명
이름	TRACE_DDL
요약	write trace messages for DDL
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

Data Definition Language (DDL) 구문을 실행할 때 해당 SQL 문장을 trace file (<GOLDILOCKS_DATA>/trc/system.trc)에 기록한다.

테이블 생성, 삭제, 변경 등과 같은 SQL 문을 실행했을 때 이에 대한 기록을 남기려면 TRACE_DDL 프로퍼티를 ON으로 설정한다.

TRACE_DDL 프로퍼티는 DDL 구문 수행에만 영향을 주며, SELECT 질의, INSERT, UPDATE, DELETE 등의 구문 수행과는 무관하므로 성능에 영향을 주지 않는다.

10.193 TRACE_LOG_ID

기본 정보

항목	설명
이름	TRACE_LOG_ID
요약	trace log ID
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	0

설명

질의를 수행할 때 해당 질의에 대한 실행 계획 정보와 기타 정보를 trace directory (<GOLDILOCKS_DATA>/trc/) 아래에 있는 trace file (opt_p[프로세스ID]_s[세션ID].trc)에 기록한다.

질의에 대한 SQL 구문과 실행 계획, 수행시간 등에 대한 기록을 남기려면 아래 표의 flag 정보를 조합하여 설정한다.

표 10-1 TRACE_LOG_ID에 대한 flag 정보

정보	Flag(on)	Flag(off)
성공한 SQL 질의 출력 여부	100000	0
실패한 SQL 질의 출력 여부	10000	0
실행 계획 출력 여부	1000	0
실행 형태 (direct/prepare) 출력 여부	100	0
Bind 값 출력 여부	10	0
구간별 수행시간 출력 여부	1	0

"성공한 SQL 질의 출력" + "실행 계획 출력" + "Bind 값 출력" 하려면 TRACE_LOG_ID 값을 101010으로 설정한다.

10.194 TRACE_LOG_MSGBUF_SIZE

기본 정보

항목	설명
이름	TRACE_LOG_MSGBUF_SIZE
요약	memory buffer size for trace log message
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	FALSE
MIN	8192
MAX	10485760
기본값	8192

설명

Trace logfile에 기록할 log message를 구성하는데 사용되는 heap memory buffer의 크기를 설정한다.

10.195 TRACE_LOG_TIME_DETAIL

기본 정보

항목	설명
이름	TRACE_LOG_TIME_DETAIL
요약	detail trace log time
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

Trace log를 기록할 때 시간 정확도를 높일지 여부를 설정한다.

이 값이 OFF로 설정된 경우, 10 ms의 정확도를 가지며, ON으로 설정된 경우 1 us의 정확도를 가진다.

10.196 TRACE_LOGGER

기본 정보

항목	설명
이름	TRACE_LOGGER
요약	trace log type (1:file, 2:file & remote)
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	2
기본값	1

설명

Trace log를 기록할 대상을 설정한다.
1이면 file에 기록하고 2이면 remote로 파일에 기록한다.
Remote로 기록하면 gtrclogger에서 원격으로 trace log를 수집하여 파일에 기록한다.

10.197 TRACE_LOGGER_REMOTE_HOST

기본 정보

항목	설명
이름	TRACE_LOGGER_REMOTE_HOST
요약	remote host for trace logger
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	255255255255
기본값	127000000001

설명

TRACE_LOGGER를 2로 설정하여 remote로 trace log를 전송할 host를 설정한다.

10.198 TRACE_LOGGER_REMOTE_PORT

기본 정보

항목	설명
이름	TRACE_LOGGER_REMOTE_PORT
요약	remote port for trace logger
Data type	BIGINT
적용단계	NONE
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1024
MAX	49151
기본값	21470

설명

TRACE_LOGGER를 2로 설정하여 remote로 trace log를 전송할 port를 설정한다.

10.199 TRACE_LOGIN

기본 정보

항목	설명
이름	TRACE_LOGIN
요약	write login trace messages for user
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

로그인 할 때 해당 접속 정보를 trace file (<GOLDILOCKS_DATA>/trc/login.trc)에 기록한다.
로그인 할 때 이에 대한 기록을 남기려면 TRACE_LOGIN 프로퍼티를 ON으로 설정한다.

10.200 TRACE_LONG_RUN_CURSOR

기본 정보

항목	설명
이름	TRACE_LONG_RUN_CURSOR
요약	write trace SQL for cursor life-time over specific time (mili-sec. 0 ~ 10000000)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

Cursor의 lifetime이 프로퍼티에 지정된 시간보다 긴 SQL 문장을 trace file (<GOLDILOCKS_DATA>/trc/system.trc)에 기록한다.

- 값의 의미
 - 단위: millisecond
 - 0 값: 정보를 기록하지 않는다.
 - 권장값: 20 (millisecond) 이상
 - 10 ms 주기의 time tic을 이용하여 수행시간을 측정하므로 20 이상의 값을 권장한다.
- 커서의 lifetime이 1초 이상인 SQL 구문을 기록한다.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_CURSOR = 1000;
```

- 기본값으로 복원한다.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_CURSOR TO DEFAULT;
```

다음과 같이 커서를 오래 유지하는 사용자 프로그램을 추적하기 위해 사용한다.

```
int main()
{
    ...
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT name FROM t1 WHERE pk = :s_id;
    EXEC SQL OPEN cur1
    EXEC SQL FETCH cur1 INTO :s_name;
```

- 사용자 logic으로 인해 ager가 오랜 시간 자원을 정리하지 못한다.

```
long_run_user_logic( s_name );
    EXEC SQL CLOSE cur1;
    ...
}
```


10.201 TRACE_LONG_RUN_SQL

기본 정보

항목	설명
이름	TRACE_LONG_RUN_SQL
요약	write trace for long-run SQL over specific execution time (mili-sec. 0 ~ 10 000000)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

구문의 수행시간이 프로퍼티에 지정된 시간보다 긴 SQL 문장을 trace file (<GOLDILOCKS_DATA>/trc/system.trc)에 기록한다.

- 값의 의미
 - 단위: millisecond
 - 0 값: 정보를 기록하지 않는다.
 - 권장값: 20 (millisecond) 이상
 - 10 ms 주기의 time tic을 이용하여 수행시간을 측정하므로 20 이상의 값을 권장한다.
- 수행시간이 1초 이상인 SQL 구문을 기록한다.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_SQL = 1000;
```

- 기본값으로 복원한다.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_SQL TO DEFAULT;
```

10.202 TRACE_XA

기본 정보

항목	설명
이름	TRACE_XA
요약	logging trace log for xa interfaces
Data type	BOOLEAN
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	NO

설명

XA 인터페이스를 사용할 때 추적 메시지를 출력할지 여부를 지정한다. 메시지는 'SYSTEM_LOGGER_DIR/xa.trc'에 출력된다.

10.203 TRANSACTION_ALLOCATION_TIMEOUT

기본 정보

항목	설명
이름	TRANSACTION_ALLOCATION_TIMEOUT
요약	a time limit (sec) for allocating transaction slot
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	3

설명

Transaction slot을 할당할 때의 최대 대기 시간이다.

대기 시간이 TRANSACTION_ALLOCATION_TIMEOUT을 초과할 경우 다음과 같은 에러가 발생한다.

```
gSQL> insert into t1 values(1);
ERR-HYT00(14129): transaction allocation time exceeded
```

10.204 TRANSACTION_COMMIT_WRITE_MODE

기본 정보

항목	설명
이름	TRANSACTION_COMMIT_WRITE_MODE
요약	transaction commit write mode (0:no_wait, 1:wait)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	1
기본값	0

설명

TRANSACTION_COMMIT_WRITE_MODE는 트랜잭션이 완료될 때 트랜잭션이 생성한 log를 disk log file에 flush 할지 여부를 설정한다. 즉, TRANSACTION_COMMIT_WRITE_MODE가 '1'이면 log를 트랜잭션 완료 시점에 disk log file에 flush해야 하고, 그렇지 않은 경우 log flush 여부와 관계없이 트랜잭션을 완료한다.

만약 TRANSACTION_COMMIT_WRITE_MODE를 '0'으로 설정하여 시스템을 운용하는 경우에 트랜잭션을 COMMIT 한 후 log flush가 되지 않은 상태에서 GOLDILOCKS가 비정상적으로 종료되면 기록되지 않은 log로 인해 최신 data를 잃어버리게 된다.

따라서 모든 트랜잭션이 완료되었을 때 반드시 database에 남아 있어야 하는 경우 TRANSACTION_COMMIT_WRITE_MODE를 '1'로 설정하여 시스템을 운용하거나, TRANSACTION_COMMIT_WRITE_MODE를 '0'으로 설정한 후 트랜잭션이 완료되는 시점에 명시적으로 'ALTER SYSTEM FLUSH LOGS'문을 수행하여 log를 flush해야 한다.

- 0: no wait
- 1: wait

10.205 TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT

기본 정보

항목	설명
이름	TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT
요약	트랜잭션이 기록할 수 있는 최대 undo 페이지 개수
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1
MAX	838860800
기본값	838860800

설명

트랜잭션이 기록할 수 있는 최대 undo 페이지 개수를 의미한다. 최소값은 1로 8 Kbyte이며 최대값은 838860800으로 100 Gbyte이다.

10.206 TRANSACTION_TABLE_SIZE

기본 정보

항목	설명
이름	TRANSACTION_TABLE_SIZE
요약	transaction table size
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	64
MAX	10240
기본값	1024

설명

Database에서 수행되는 최대 트랜잭션 테이블의 수를 설정한다. Database를 갱신할 때 트랜잭션의 ACID를 보장하기 위해 트랜잭션 테이블을 할당한다. 트랜잭션 테이블의 수를 변경하려면 database를 재시작해야 하며, 이전에 설정된 값보다 더 큰 수로만 변경할 수 있다.

더 작은 수로 변경하면 재시작에 실패하는데 예를 들어, 1024로 설정된 값을 512로 변경하면 다음과 같이 재시작에 실패한다. 이 경우 1024보다 크거나 같은 수로 설정하면 재시작에 성공한다.

```
gSQL> ALTER SYSTEM SET TRANSACTION_TABLE_SIZE = 512 SCOPE = FILE;
System altered.
gSQL> \CONNECT sys gliese as sysdba
gSQL> \SHUTDOWN
Shutdown success
```

- 재시작 실패

```
gSQL> \STARTUP
ERR-HY000(14118): TRANSACTION_TABLE_SIZE property value must be equal to or greater than
'1024'
gSQL> ALTER SYSTEM SET TRANSACTION_TABLE_SIZE = 1024 SCOPE = FILE;
```

```
System altered.  
gSQL> \SHUTDOWN  
Shutdown success  
gSQL> \STARTUP  
Startup success
```

10.207 TRANSACTION_TIMEOUT

기본 정보

항목	설명
이름	TRANSACTION_TIMEOUT
요약	transaction timeout(s)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	10000000
기본값	0

설명

Transaction이 활성화되어 있는 시간을 설정한다. Transaction이 장시간 활성화되어 있을 때 발생할 수 있는 부작용을 방지하기 위해 사용한다. Transaction이 정해진 시간을 초과할 경우, gmaster 데몬이 해당 transaction을 소유한 세션을 자동으로 종료시킨다.

10.208 UNDO_RELATION_ALLOCATION_TIMEOUT

기본 정보

항목	설명
이름	UNDO_RELATION_ALLOCATION_TIMEOUT
요약	a time limit (sec) for allocating undo relation
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	TRUE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	100000000
기본값	3

설명

Undo relation을 할당할 때의 최대 대기 시간이다.

대기 시간이 UNDO_RELATION_ALLOCATION_TIMEOUT을 초과할 경우 다음과 같은 에러가 발생한다.

```
gSQL> insert into t1 values(1);
ERR-HYT00(14130): undo relation allocation time exceeded
```

10.209 UNDO_RELATION_COUNT

기본 정보

항목	설명
이름	UNDO_RELATION_COUNT
요약	undo relation count
Data type	BIGINT
적용단계	NO_MOUNT 이하
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	8
MAX	10240
기본값	128

설명

Database에서 사용할 undo relation의 수를 설정한다. Undo relation은 DML을 수행하는 트랜잭션이 undo segment를 사용하도록 하기 위해 할당한다. Undo relation의 수를 변경하기 위해서는 database를 재시작해야 하며 이전에 설정된 값보다 더 큰 수로만 변경할 수 있다.

더 작은 수로 변경하면 재시작에 실패하는데 예를 들어, 128로 설정된 값을 64로 변경하면 다음과 같이 재시작에 실패한다. 이 경우 128보다 크거나 같은 수로 설정하면 재시작에 성공한다.

```
gSQL> ALTER SYSTEM SET UNDO_RELATION_COUNT = 64 SCOPE = FILE;
System altered.
gSQL> \CONNECT sys gliese as sysdba
gSQL> \SHUTDOWN
Shutdown success
```

- 재시작 실패

```
gSQL> \STARTUP
ERR-HY000(14119): UNDO_RELATION_COUNT property value must be equal to or greater than '128'
gSQL> ALTER SYSTEM SET UNDO_RELATION_COUNT = 128 SCOPE = FILE;
System altered.
```

```
gSQL> \SHUTDOWN  
Shutdown success  
gSQL> \STARTUP  
Startup success
```

10.210 UNDO_SHRINK_THRESHOLD

기본 정보

항목	설명
이름	UNDO_SHRINK_THRESHOLD
요약	threshold bytes to attempt to shrink undo segment(byte)
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	1048576
MAX	107374182400
기본값	10485760

설명

Ager thread는 주기적으로 (10초) undo segment의 공간을 검사하여 공간을 많이 차지하고 있을 경우, 일부분을 테이블스페이스로 반환한다. 이 프로퍼티는 한 번에 반환해야 할 바이트 단위의 크기를 의미한다.

10.211 USE_LARGE_PAGES

기본 정보

항목	설명
이름	USE_LARGE_PAGES
요약	use large pages
Data type	BIGINT
적용단계	NO_MOUNT 이상
변경가능 여부	TRUE
ALTER SESSION 여부	FALSE
ALTER SYSTEM 여부	IMMEDIATE
MIN	0
MAX	2
기본값	0

설명

HugePage를 사용한다. USE_LARGE_PAGES를 사용하려면 먼저 장비에 HugePage를 설정해야 한다.

- 0: Large page를 사용하지 않는다.
- 1: Large page를 사용한다. 만약 공유 메모리 할당에 실패할 경우에는 에러가 발생한다.
- 2: Large page를 사용하여 할당을 시도한다. 만약 공유 메모리 할당에 실패할 경우에는 regular page를 사용하여 메모리를 할당한다.

노트

리눅스 커널 2.6.32-573 이상에서만 사용할 수 있다.

파트 III.

SQL Manual

11. SQL Elements	855
11.1 Syntax Elements	856
Identifiers	856
Literals	856
Null Value	869
Comments	869
SQL Reserved Words and Keywords	870
호환성	875
11.2 Data Type	877
숫자 타입	877
CHARACTER STRING 타입	879
BINARY STRING 타입	879
날짜/ 시간 타입	880
INTERVAL 타입	880
BOOLEAN 타입	880
ROWID 타입	880
타입간 비교	881
타입간 변환	888
타입간 조합	894
호환성	899
11.3 Format 문자열	903
Number Format 문자열	903
Datetime Format 문자열	906
11.4 Expressions	916
Boolean Value Expression	916
CASE Expression	918
CAST specification	920
Scalar Subquery Expression	921
호환성	921
11.5 Pseudo Columns	923
ROWID Pseudo Column	923
CLUSTER_GROUP_ID Pseudo Column	924
CLUSTER_MEMBER_ID Pseudo Column	925
CLUSTER_GROUP_NAME Pseudo Column	925
CLUSTER_MEMBER_NAME Pseudo Column	926
호환성	926
11.6 Operators	927
Arithmetic Operator	927
Concatenation Operator	928

Set Operator	928
호환성	929
11.7 Functions	930
Single Row Function	930
Aggregate Function	935
호환성	936
11.8 Conditions	938
Condition	938
Comparison Conditions	938
Logical Conditions	942
Null Condition	943
Compound Condition	944
Pattern-matching Conditions	944
BETWEEN Condition	945
IN Condition	946
EXISTS Condition	947
호환성	948
11.9 Built-in Data Type References	950
Aliases of Built-in Data Types	950
BINARY	953
BINARY VARYING	954
BINARY LONG VARYING	955
BOOLEAN	956
CHARACTER	957
CHARACTER VARYING	958
CHARACTER LONG VARYING	959
DATE	960
FLOAT	961
INTERVAL	962
NATIVE_BIGINT	966
NATIVE_DOUBLE	967
NATIVE_INTEGER	968
NATIVE_REAL	969
NATIVE_SMALLINT	970
NUMBER	971
NUMERIC	972
ROWID	973
TIME	974
TIMESTAMP	975

11.10 Built-in Function References	976
* (MULTIPLICATION)	976
+ (ADDITION)	978
+ (POSITIVE)	980
- (NEGATIVE)	981
- (SUBTRACTION)	982
/ (DIVISION)	985
(CONCATENATE)	987
ABS	988
ACOS	989
ADDDATE	990
ADDTIME	991
ADD_MONTHS	992
ASCII	993
ASIN	994
ATAN	995
ATAN2	996
AVG	997
BITAND	998
BITNOT	999
BITOR	1,000
BITXOR	1,001
BIT_LENGTH	1,002
BYTE_LENGTH	1,003
CASE2	1,004
CBRT	1,006
CEIL	1,007
CHAR_LENGTH	1,008
CHR	1,009
CLOCK_DATE	1,010
CLOCK_LOCALTIME	1,011
CLOCK_LOCALTIMESTAMP	1,012
CLOCK_TIME	1,013
CLOCK_TIMESTAMP	1,014
COALESCE	1,015
CONCAT	1,016
CONCATENATE	1,017
COS	1,018
COT	1,019

COUNT	1,020
COUNT(*)	1,021
CURRENT_CATALOG	1,022
CURRENT_DATE	1,023
CURRENT_SCHEMA	1,024
CURRENT_TIME	1,025
CURRENT_TIMESTAMP	1,026
CURRENT_USER	1,027
CURRVAL	1,028
DATEADD	1,029
DATEDIFF	1,031
DATE_ADD	1,033
DATE_PART	1,034
DECODE	1,035
DEGREES	1,037
DIGEST	1,038
DUMP	1,039
EXP	1,040
EXTRACT	1,041
FACTORIAL	1,043
FLOOR	1,044
FROM_BASE64	1,045
GREATEST	1,046
HEX	1,047
INITCAP	1,048
INSTR	1,049
LAST_DAY	1,051
LAST_IDENTITY_VALUE	1,052
LEAST	1,055
LENGTH	1,056
LENGTHB	1,057
LN	1,058
LOCALTIME	1,059
LOCALTIMESTAMP	1,060
LOCAL_GROUP_ID	1,061
LOCAL_GROUP_NAME	1,062
LOCAL_MEMBER_ID	1,063
LOCAL_MEMBER_NAME	1,064
LOG	1,065

LOGON_USER	1,066
LOWER	1,067
LPAD	1,068
LTRIM	1,069
MAX	1,070
MIN	1,071
MOD	1,072
MONTHS_BETWEEN	1,073
NEXT_DAY	1,075
NEXTVAL	1,077
NULLIF	1,078
NVL	1,079
NVL2	1,080
OCTET_LENGTH	1,081
OVERLAY	1,082
PI	1,084
POSITION	1,085
POWER	1,086
RADIANS	1,087
RANDOM	1,088
REPEAT	1,089
REPLACE	1,090
REVERSE	1,091
ROUND(number)	1,092
ROUND(date)	1,093
ROWID_GRID_BLOCK_ID	1,095
ROWID_GRID_BLOCK_SEQ	1,096
ROWID_MEMBER_ID	1,097
ROWID_OBJECT_ID	1,098
ROWID_PAGE_ID	1,099
ROWID_ROW_NUMBER	1,100
ROWID_SHARD_ID	1,101
ROWID_TABLESPACE_ID	1,102
ROWNUM	1,103
RPAD	1,107
RTRIM	1,108
SESSION_ID	1,109
SESSION_SERIAL	1,110
SESSION_USER	1,111

SHARD_GROUP_ID	1,112
SHARD_GROUP_NAME	1,113
SHARD_ID	1,114
SHARD_NAME	1,115
SHIFT_LEFT	1,116
SHIFT_RIGHT	1,117
SIGN	1,118
SIN	1,119
SPLIT_PART	1,120
SQRT	1,121
STATEMENT_DATE	1,122
STATEMENT_LOCALTIME	1,123
STATEMENT_LOCALTIMESTAMP	1,124
STATEMENT_TIME	1,125
STATEMENT_TIMESTAMP	1,126
STATEMENT_VIEW_SCN	1,127
STATEMENT_VIEW_SCN_DCN	1,128
STATEMENT_VIEW_SCN_GCN	1,129
STATEMENT_VIEW_SCN_LCN	1,130
STDDEV	1,131
STDDEV_POP	1,133
STDDEV_SAMP	1,135
SUBSTR	1,137
SUBSTRB	1,138
SUBSTRING	1,139
SUM	1,141
SYSDATE	1,142
SYS_EXTRACT_UTC	1,143
SYSTIME	1,144
SYSTIMESTAMP	1,145
TAN	1,146
TO_BASE64	1,147
TO_CHAR(datetime)	1,148
TO_CHAR(number)	1,150
TO_DATE	1,151
TO_NATIVE_DOUBLE	1,152
TO_NATIVE_REAL	1,153
TO_NUMBER	1,154
TO_TIME	1,155

TO_TIME_TZ	1,156
TO_TIME_WITH_TIME_ZONE	1,157
TO_TIMESTAMP	1,159
TO_TIMESTAMP_TZ	1,160
TO_TIMESTAMP_WITH_TIME_ZONE	1,161
TRANSACTION_DATE	1,163
TRANSACTION_LOCALTIME	1,164
TRANSACTION_LOCALTIMESTAMP	1,165
TRANSACTION_TIME	1,166
TRANSACTION_TIMESTAMP	1,167
TRANSLATE	1,168
TRIM	1,170
TRUNC(number)	1,172
TRUNC(date)	1,173
UPPER	1,175
UNHEX	1,176
UNHEX_TO_CHARSTR	1,177
USER_ID	1,178
UUID	1,179
VAR_POP	1,180
VAR_SAMP	1,182
VARIANCE	1,184
VERSION	1,186
WIDTH_BUCKET	1,187
12. SQL Languages	1,189
12.1 Data Definition Language	1,190
DDL 관련 구문	1,190
DDL 개념	1,190
DDL과 트랜잭션	1,191
12.2 Data Manipulation Language	1,199
DML 관련 구문	1,199
DML 개념	1,199
데이터 추가	1,200
데이터 삭제	1,201
데이터 갱신	1,202
커서를 이용한 데이터 조작	1,203
DML Query	1,204
12.3 Data Query Language	1,207
Query 관련 구문	1,207

Query 개념	1,207
기본 Query	1,208
SET 연산자	1,210
조인	1,211
결과 집합 그룹 (group by)	1,216
결과 집합 정렬 (order by)	1,217
부질의 (Subquery)	1,218
12.4 Control Language	1,220
Control Language 관련 구문	1,220
Transaction Control	1,220
Session Control	1,222
System Control	1,223
12.5 Cluster의 SQL 처리	1,225
Cluster의 DDL 처리	1,225
Cluster의 SELECT 처리	1,227
Cluster의 DML 처리	1,260
13. SQL Objects	1,273
13.1 Database	1,274
데이터베이스 관련 구문	1,274
데이터베이스 구성 객체	1,275
Built-in 객체	1,277
13.2 Profile	1,280
Profile 관련 구문	1,280
Profile 개념	1,280
13.3 Audit Policy	1,286
Audit Policy 관련 구문	1,286
사용 예	1,286
Audit Policy 개념	1,290
13.4 Authorization	1,313
Authorization 관련 구문	1,313
User 개념	1,314
객체의 생성과 권한	1,315
권한 (Privilege)	1,318
13.5 Schema	1,323
Schema 관련 구문	1,323
Schema 개념	1,323
User와 Schema	1,324
Schema Path	1,325
PUBLIC 스키마	1,329

User와 Schema 활용 예	1,330
13.6 Tablespace	1,333
Tablespace 관련 구문	1,333
Tablespace 개념	1,333
13.7 Table	1,336
테이블 관련 구문	1,336
테이블 개념	1,337
Global Temporary Table	1,339
Cluster 테이블	1,340
13.8 Index	1,341
Index 관련 구문	1,341
Index 개념	1,341
UNIQUE 개념	1,343
13.9 View	1,345
View 관련 구문	1,345
View 개념	1,345
13.10 Sequence	1,348
Sequence 관련 구문	1,348
Sequence 개념	1,348
Cluster Sequence	1,350
13.11 Synonym	1,352
Synonym 관련 구문	1,352
Synonym 개념	1,352
13.12 Stored Procedure	1,355
Stored Procedure 관련 구문	1,355
Stored Procedure 개념	1,355
13.13 Stored Function	1,357
Stored Function 관련 구문	1,357
Stored Function 개념	1,357
14. Cluster Objects	1,359
14.1 Cluster System	1,360
Cluster System 관련 구문	1,360
Cluster System 개념	1,360
Cluster System의 가용성	1,362
Cluster System 확장	1,364
14.2 Cluster Group	1,367
Cluster Group 관련 구문	1,367
Cluster Group 개념	1,367
14.3 Cluster Member	1,369

Cluster Member 관련 구문	1,369
Cluster Member 개념	1,369
14.4 Cluster Location	1,371
Cluster Location 관련 구문	1,371
Cluster Location 개념	1,371
14.5 Cluster Table과 Shard	1,372
Shard 관련 구문	1,372
Cluster Table 유형	1,372
Cloned Table	1,373
Hash-sharded Table	1,375
Range-sharded Table	1,377
List-sharded Table	1,379
Cluster Table 재배치	1,381
14.6 Global Secondary Index	1,388
Global Secondary Index 관련 구문	1,388
Global Secondary Index 개념	1,388
15. SQL Tuning	1,391
15.1 SQL Tuning 개요	1,392
15.2 SQL 처리 과정	1,393
SQL Parser	1,394
Plan Cache Check	1,394
SQL Validation	1,394
Optimization	1,395
Plan Generation	1,395
Plan Cache Registration	1,396
Execution	1,397
15.3 Query Optimizer	1,399
개요	1,399
Query Transformation	1,399
Access Paths	1,406
Join	1,411
Cluster	1,427
통계 정보	1,430
Optimizer 조정	1,430
15.4 SQL 실행 계획	1,432
개요	1,432
출력	1,432
읽기	1,435
15.5 SQL Trace Log	1,470

개요	1,470
출력	1,470
출력 형식	1,471
출력 예	1,473
16. SQL References	1,477
16.1 ALTER AUDIT POLICY	1,478
기능	1,478
구문	1,478
사용 범위 및 접근 권한	1,478
구문 규칙 및 파라미터	1,479
설명	1,479
사용 예	1,480
호환성	1,480
참조	1,480
16.2 ALTER CLUSTER GROUP name ADD MEMBER	1,481
기능	1,481
구문	1,481
사용 범위 및 접근 권한	1,481
구문 규칙 및 파라미터	1,481
설명	1,482
사용 예	1,482
호환성	1,483
참조	1,483
16.3 ALTER CLUSTER GROUP name OFFLINE MEMBER	1,484
기능	1,484
구문	1,484
사용 범위 및 접근 권한	1,484
구문 규칙 및 파라미터	1,484
설명	1,485
사용 예	1,485
호환성	1,485
참조	1,485
16.4 ALTER CLUSTER LOCATION	1,487
기능	1,487
구문	1,487
사용 범위 및 접근 권한	1,487
구문 규칙 및 파라미터	1,487
설명	1,488
사용 예	1,488

호환성	1,488
참조	1,488
16.5 ALTER DATABASE ADD LOGFILE	1,489
기능	1,489
구문	1,489
사용 범위 및 접근 권한	1,489
구문 규칙 및 파라미터	1,490
설명	1,490
사용 예	1,491
호환성	1,491
참조	1,491
16.6 ALTER DATABASE ARCHIVELOG	1,492
기능	1,492
구문	1,492
사용 범위 및 접근 권한	1,492
구문 규칙 및 파라미터	1,492
설명	1,492
사용 예	1,493
호환성	1,493
참조	1,493
16.7 ALTER DATABASE BACKUP	1,494
기능	1,494
구문	1,494
사용 범위 및 접근 권한	1,494
구문 규칙 및 파라미터	1,495
설명	1,496
사용 예	1,496
호환성	1,497
참조	1,497
16.8 ALTER DATABASE CLEAR AUDIT TRAIL	1,498
기능	1,498
구문	1,498
사용 범위 및 접근 권한	1,498
설명	1,498
Audit Trail 저장	1,498
사용 예	1,499
호환성	1,499
참조	1,499
16.9 ALTER DATABASE CLEAR PASSWORD HISTORY	1,500

기능	1,500
구문	1,500
사용 범위 및 접근 권한	1,500
설명	1,500
사용 예	1,501
호환성	1,501
참조	1,501
16.10 ALTER DATABASE DELETE BACKUP	1,502
기능	1,502
구문	1,502
사용 범위 및 접근 권한	1,502
구문 규칙 및 파라미터	1,502
설명	1,503
사용 예	1,503
호환성	1,503
참조	1,504
16.11 ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	1,505
기능	1,505
구문	1,505
사용 범위 및 접근 권한	1,505
설명	1,505
사용 예	1,506
호환성	1,506
참조	1,506
16.12 ALTER DATABASE DROP LOGFILE	1,507
기능	1,507
구문	1,507
사용 범위 및 접근 권한	1,507
구문 규칙 및 파라미터	1,507
설명	1,508
사용 예	1,508
호환성	1,508
참조	1,509
16.13 ALTER DATABASE MOVE SHARD	1,510
기능	1,510
구문	1,510
사용 범위 및 접근 권한	1,510
구문 규칙 및 파라미터	1,510
설명	1,511

사용 예	1,512
호환성	1,512
참조	1,512
16.14 ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS	1,513
기능	1,513
구문	1,513
사용 범위 및 접근 권한	1,513
구문 규칙 및 파라미터	1,513
설명	1,514
사용 예	1,514
호환성	1,514
참조	1,514
16.15 ALTER DATABASE REBALANCE	1,515
기능	1,515
구문	1,515
사용 범위 및 접근 권한	1,515
구문 규칙 및 파라미터	1,515
설명	1,516
사용 예	1,516
호환성	1,517
참조	1,517
16.16 ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP	1,518
기능	1,518
구문	1,518
사용 범위 및 접근 권한	1,518
구문 규칙 및 파라미터	1,518
설명	1,519
사용 예	1,519
호환성	1,520
참조	1,520
16.17 ALTER DATABASE RENAME LOGFILE	1,521
기능	1,521
구문	1,521
사용 범위 및 접근 권한	1,521
구문 규칙 및 파라미터	1,521
설명	1,522
사용 예	1,522
호환성	1,522
참조	1,522

16.18 ALTER DATABASE RECOVER	1,523
기능	1,523
구문	1,523
사용 범위 및 접근 권한	1,524
구문 규칙 및 파라미터	1,524
설명	1,526
사용 예	1,526
호환성	1,527
참조	1,527
16.19 ALTER DATABASE REGISTER	1,528
기능	1,528
구문	1,528
사용 범위 및 접근 권한	1,528
구문 규칙 및 파라미터	1,528
설명	1,529
사용 예	1,529
호환성	1,529
참조	1,529
16.20 ALTER DATABASE RESET LOCAL CLUSTER MEMBER	1,530
기능	1,530
구문	1,530
사용 범위 및 접근 권한	1,530
설명	1,530
사용 예	1,531
호환성	1,531
참조	1,531
16.21 ALTER DATABASE RESTORE	1,532
기능	1,532
구문	1,532
사용 범위 및 접근 권한	1,532
구문 규칙 및 파라미터	1,532
설명	1,533
사용 예	1,533
호환성	1,534
참조	1,534
16.22 ALTER INDEX	1,535
기능	1,535
구문	1,535
사용 범위 및 접근 권한	1,535

구문 규칙 및 파라미터	1,535
설명	1,536
사용 예	1,536
호환성	1,536
16.23 ALTER INDEX name AGING	1,537
기능	1,537
구문	1,537
사용 범위 및 접근 권한	1,537
구문 규칙 및 파라미터	1,537
설명	1,537
사용 예	1,538
호환성	1,538
참조	1,538
16.24 ALTER INDEX name STORAGE	1,540
기능	1,540
구문	1,540
사용 범위 및 접근 권한	1,540
구문 규칙 및 파라미터	1,541
설명	1,543
사용 예	1,543
호환성	1,543
참조	1,543
16.25 ALTER INDEX name RENAME TO	1,544
기능	1,544
구문	1,544
사용 범위 및 접근 권한	1,544
구문 규칙 및 파라미터	1,544
설명	1,545
사용 예	1,545
호환성	1,545
참조	1,545
16.26 ALTER PROFILE	1,546
기능	1,546
구문	1,546
사용 범위 및 접근 권한	1,546
구문 규칙 및 파라미터	1,547
사용 예	1,548
호환성	1,549
참조	1,549

16.27 ALTER SEQUENCE	1,550
기능	1,550
구문	1,550
사용 범위 및 접근 권한	1,551
구문 규칙 및 파라미터	1,551
설명	1,553
사용 예	1,553
호환성	1,554
참조	1,554
16.28 ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;	1,555
기능	1,555
구문	1,555
설명	1,555
사용 예	1,555
호환성	1,555
참조	1,556
16.29 ALTER SESSION SET property_name	1,557
기능	1,557
구문	1,557
구문 규칙 및 파라미터	1,557
설명	1,557
사용 예	1,558
호환성	1,558
참조	1,558
16.30 ALTER SYSTEM CHECKPOINT	1,559
기능	1,559
구문	1,559
사용 범위 및 접근 권한	1,559
구문 규칙 및 파라미터	1,559
설명	1,560
사용 예	1,560
호환성	1,560
16.31 ALTER SYSTEM CLEANUP PLAN	1,561
기능	1,561
구문	1,561
사용 범위 및 접근 권한	1,561
구문 규칙 및 파라미터	1,561
설명	1,562
사용 예	1,562

호환성	1,562
16.32 ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER	1,563
기능	1,563
구문	1,563
사용 범위 및 접근 권한	1,563
구문 규칙 및 파라미터	1,563
설명	1,564
사용 예	1,564
호환성	1,564
16.33 ALTER SYSTEM JOIN DATABASE	1,565
기능	1,565
구문	1,565
사용 범위 및 접근 권한	1,565
설명	1,565
사용 예	1,566
호환성	1,566
참조	1,567
16.34 ALTER SYSTEM {MOUNT OPEN} DATABASE	1,568
기능	1,568
구문	1,568
사용 범위 및 접근 권한	1,568
구문 규칙 및 파라미터	1,568
사용 예	1,569
호환성	1,570
참조	1,570
16.35 ALTER SYSTEM [KILL DISCONNECT] SESSION	1,571
기능	1,571
구문	1,571
사용 범위 및 접근 권한	1,571
구문 규칙 및 파라미터	1,571
설명	1,572
사용 예	1,572
호환성	1,573
16.36 ALTER SYSTEM RECONNECT GLOBAL CONNECTION	1,574
기능	1,574
구문	1,574
사용 범위 및 접근 권한	1,574
설명	1,574
사용 예	1,574

호환성	1,575
16.37 ALTER SYSTEM RESET property_name	1,576
기능	1,576
구문	1,576
사용 범위 및 접근 권한	1,576
구문 규칙 및 파라미터	1,576
설명	1,577
사용 예	1,577
호환성	1,578
참조	1,578
16.38 ALTER SYSTEM SET property_name	1,579
기능	1,579
구문	1,579
사용 범위 및 접근 권한	1,579
구문 규칙 및 파라미터	1,579
설명	1,581
사용 예	1,581
호환성	1,581
참조	1,582
16.39 ALTER SYSTEM SWITCH LOGFILE	1,583
기능	1,583
구문	1,583
사용 범위 및 접근 권한	1,583
구문 규칙 및 파라미터	1,583
설명	1,584
사용 예	1,584
호환성	1,584
참조	1,584
16.40 ALTER TABLE	1,585
기능	1,585
구문	1,585
사용 범위 및 접근 권한	1,585
구문 규칙 및 파라미터	1,586
설명	1,588
사용 예	1,588
호환성	1,588
16.41 ALTER TABLE name ADD COLUMN	1,589
기능	1,589
구문	1,589

사용 범위 및 접근 권한	1,589
구문 규칙 및 파라미터	1,590
설명	1,591
사용 예	1,591
호환성	1,592
참조	1,592
16.42 ALTER TABLE name SET UNUSED COLUMN	1,593
기능	1,593
구문	1,593
사용 범위 및 접근 권한	1,593
구문 규칙 및 파라미터	1,593
설명	1,594
사용 예	1,594
호환성	1,595
참조	1,595
16.43 ALTER TABLE name ALTER COLUMN	1,596
기능	1,596
구문	1,596
사용 범위 및 접근 권한	1,597
구문 규칙 및 파라미터	1,597
설명	1,605
사용 예	1,606
호환성	1,607
참조	1,607
16.44 ALTER TABLE name RENAME COLUMN	1,608
기능	1,608
구문	1,608
사용 범위 및 접근 권한	1,608
구문 규칙 및 파라미터	1,608
설명	1,609
사용 예	1,609
호환성	1,609
참조	1,609
16.45 ALTER TABLE name ADD CONSTRAINT	1,610
기능	1,610
구문	1,610
사용 범위 및 접근 권한	1,610
구문 규칙 및 파라미터	1,611
설명	1,611

사용 예	1,611
호환성	1,612
참조	1,612
16.46 ALTER TABLE name DROP CONSTRAINT	1,613
기능	1,613
구문	1,613
사용 범위 및 접근 권한	1,613
구문 규칙 및 파라미터	1,614
설명	1,614
사용 예	1,614
호환성	1,615
참조	1,615
16.47 ALTER TABLE name ALTER CONSTRAINT	1,616
기능	1,616
구문	1,616
사용 범위 및 접근 권한	1,616
구문 규칙 및 파라미터	1,617
설명	1,618
사용 예	1,618
호환성	1,618
16.48 ALTER TABLE name RENAME CONSTRAINT	1,619
기능	1,619
구문	1,619
사용 범위 및 접근 권한	1,619
구문 규칙 및 파라미터	1,619
설명	1,620
사용 예	1,620
호환성	1,620
참조	1,621
16.49 ALTER TABLE name ADD GLOBAL SECONDARY INDEX	1,622
기능	1,622
구문	1,622
사용 범위 및 접근 권한	1,623
구문 규칙 및 파라미터	1,623
설명	1,626
사용 예	1,626
호환성	1,627
참조	1,627
16.50 ALTER TABLE name DROP GLOBAL SECONDARY INDEX	1,628

기능	1,628
구문	1,628
사용 범위 및 접근 권한	1,628
구문 규칙 및 파라미터	1,628
설명	1,629
사용 예	1,629
호환성	1,629
참조	1,629
16.51 ALTER TABLE name ALTER GLOBAL SECONDARY INDEX	1,630
기능	1,630
구문	1,630
사용 범위 및 접근 권한	1,630
구문 규칙 및 파라미터	1,631
설명	1,633
사용 예	1,633
호환성	1,633
참조	1,633
16.52 ALTER TABLE name MOVE SHARD	1,634
기능	1,634
구문	1,634
사용 범위 및 접근 권한	1,634
구문 규칙 및 파라미터	1,634
설명	1,635
사용 예	1,635
호환성	1,636
참조	1,636
16.53 ALTER TABLE name REBALANCE	1,637
기능	1,637
구문	1,637
사용 범위 및 접근 권한	1,637
구문 규칙 및 파라미터	1,637
설명	1,638
사용 예	1,638
호환성	1,638
16.54 ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list	1,639
기능	1,639
구문	1,639
사용 범위 및 접근 권한	1,639
구문 규칙 및 파라미터	1,639

설명	1,640
사용 예	1,640
호환성	1,640
16.55 ALTER TABLE name SPLIT SHARD	1,641
기능	1,641
구문	1,641
사용 범위 및 접근 권한	1,641
구문 규칙 및 파라미터	1,642
설명	1,643
사용 예	1,644
호환성	1,644
참조	1,644
16.56 ALTER TABLE name RENAME SHARD	1,645
기능	1,645
구문	1,645
사용 범위 및 접근 권한	1,645
구문 규칙 및 파라미터	1,645
설명	1,646
사용 예	1,646
호환성	1,646
참조	1,646
16.57 ALTER TABLE name READ { ONLY WRITE }	1,648
기능	1,648
구문	1,648
사용 범위 및 접근 권한	1,648
구문 규칙 및 파라미터	1,648
설명	1,649
사용 예	1,649
호환성	1,650
참조	1,650
16.58 ALTER TABLE name RENAME TO	1,651
기능	1,651
구문	1,651
사용 범위 및 접근 권한	1,651
구문 규칙 및 파라미터	1,651
설명	1,652
사용 예	1,652
호환성	1,652
참조	1,652

16.59 ALTER TABLE name STORAGE	1,653
기능	1,653
구문	1,653
사용 범위 및 접근 권한	1,653
구문 규칙 및 파라미터	1,654
설명	1,654
사용 예	1,654
호환성	1,655
참조	1,655
16.60 ALTER TABLE name ADD SUPPLEMENTAL LOG	1,656
기능	1,656
구문	1,656
사용 범위 및 접근 권한	1,656
구문 규칙 및 파라미터	1,656
설명	1,657
사용 예	1,657
호환성	1,657
참조	1,657
16.61 ALTER TABLE name DROP SUPPLEMENTAL LOG	1,658
기능	1,658
구문	1,658
사용 범위 및 접근 권한	1,658
구문 규칙 및 파라미터	1,658
설명	1,659
사용 예	1,659
호환성	1,659
16.62 ALTER TABLESPACE	1,660
기능	1,660
구문	1,660
사용 범위 및 접근 권한	1,660
구문 규칙 및 파라미터	1,660
설명	1,661
사용 예	1,661
호환성	1,661
참조	1,662
16.63 ALTER TABLESPACE name RENAME TO	1,663
기능	1,663
구문	1,663
사용 범위 및 접근 권한	1,663

구문 규칙 및 파라미터	1,663
설명	1,664
사용 예	1,664
호환성	1,664
참조	1,664
16.64 ALTER TABLESPACE name BACKUP	1,665
기능	1,665
구문	1,665
사용 범위 및 접근 권한	1,665
구문 규칙 및 파라미터	1,666
설명	1,667
사용 예	1,667
호환성	1,667
참조	1,667
16.65 ALTER TABLESPACE name [ONLINE OFFLINE]	1,668
기능	1,668
구문	1,668
사용 범위 및 접근 권한	1,668
구문 규칙 및 파라미터	1,668
설명	1,669
사용 예	1,669
호환성	1,670
참조	1,670
16.66 ALTER TABLESPACE name ADD [DATAFILE MEMORY]	1,671
기능	1,671
구문	1,671
사용 범위 및 접근 권한	1,671
구문 규칙 및 파라미터	1,672
설명	1,673
사용 예	1,673
호환성	1,673
참조	1,673
16.67 ALTER TABLESPACE name DROP [DATAFILE MEMORY]	1,674
기능	1,674
구문	1,674
사용 범위 및 접근 권한	1,674
구문 규칙 및 파라미터	1,674
설명	1,675
사용 예	1,675

호환성	1,675
참조	1,676
16.68 ALTER TABLESPACE name RENAME DATAFILE	1,677
기능	1,677
구문	1,677
사용 범위 및 접근 권한	1,677
구문 규칙 및 파라미터	1,677
설명	1,678
사용 예	1,678
호환성	1,678
참조	1,679
16.69 ALTER USER	1,680
기능	1,680
구문	1,680
사용 범위 및 접근 권한	1,681
구문 규칙 및 파라미터	1,681
설명	1,684
사용 예	1,684
호환성	1,685
참조	1,685
16.70 ALTER VIEW	1,686
기능	1,686
구문	1,686
사용 범위 및 접근 권한	1,686
구문 규칙 및 파라미터	1,686
설명	1,687
사용 예	1,687
호환성	1,688
참조	1,688
16.71 ANALYZE SYSTEM	1,689
기능	1,689
구문	1,689
사용 범위 및 접근 권한	1,689
구문 규칙 및 파라미터	1,689
설명	1,690
사용 예	1,690
호환성	1,690
참조	1,690
16.72 ANALYZE TABLE	1,691

기능	1,691
구문	1,691
사용 범위 및 접근 권한	1,691
구문 규칙 및 파라미터	1,692
설명	1,695
사용 예	1,695
호환성	1,697
참조	1,697
16.73 AUDIT POLICY	1,698
기능	1,698
구문	1,698
사용 범위 및 접근 권한	1,698
구문 규칙 및 파라미터	1,698
설명	1,699
사용 예	1,702
호환성	1,703
참조	1,703
16.74 CLOSE cursor_name	1,704
기능	1,704
구문	1,704
구문 규칙 및 파라미터	1,704
설명	1,704
사용 예	1,704
호환성	1,705
참조	1,705
16.75 COMMENT ON name IS	1,706
기능	1,706
구문	1,706
사용 범위 및 접근 권한	1,706
구문 규칙 및 파라미터	1,708
설명	1,709
사용 예	1,710
호환성	1,710
16.76 COMMIT	1,711
기능	1,711
구문	1,711
구문 규칙 및 파라미터	1,711
설명	1,712
사용 예	1,713

호환성	1,713
참조	1,713
16.77 CREATE AUDIT POLICY	1,714
기능	1,714
구문	1,714
사용 범위 및 접근 권한	1,714
구문 규칙 및 파라미터	1,715
설명	1,717
사용 예	1,718
호환성	1,719
참조	1,719
16.78 CREATE CLUSTER GROUP	1,721
기능	1,721
구문	1,721
사용 범위 및 접근 권한	1,721
구문 규칙 및 파라미터	1,721
설명	1,722
사용 예	1,722
호환성	1,723
참조	1,723
16.79 CREATE CLUSTER LOCATION	1,724
기능	1,724
구문	1,724
사용 범위 및 접근 권한	1,724
구문 규칙 및 파라미터	1,724
설명	1,725
사용 예	1,725
호환성	1,725
참조	1,725
16.80 CREATE INDEX	1,726
기능	1,726
구문	1,726
사용 범위 및 접근 권한	1,727
구문 규칙 및 파라미터	1,727
설명	1,731
사용 예	1,731
호환성	1,732
참조	1,733
16.81 CREATE PROFILE	1,734

기능	1,734
구문	1,734
사용 범위 및 접근 권한	1,734
구문 규칙 및 파라미터	1,735
설명	1,739
사용 예	1,743
호환성	1,744
참조	1,744
16.82 CREATE SCHEMA	1,745
기능	1,745
구문	1,745
사용 범위 및 접근 권한	1,745
구문 규칙 및 파라미터	1,746
설명	1,747
사용 예	1,748
호환성	1,748
참조	1,749
16.83 CREATE SEQUENCE	1,750
기능	1,750
구문	1,750
사용 범위 및 접근 권한	1,751
구문 규칙 및 파라미터	1,751
설명	1,753
사용 예	1,754
호환성	1,754
참조	1,754
16.84 CREATE SYNONYM	1,756
기능	1,756
구문	1,756
사용 범위 및 접근 권한	1,756
구문 규칙 및 파라미터	1,757
설명	1,758
사용 예	1,758
호환성	1,759
참조	1,759
16.85 CREATE TABLE	1,760
기능	1,760
구문	1,760
사용 범위 및 접근 권한	1,765

구문 규칙 및 파라미터	1,766
설명	1,785
사용 예	1,786
호환성	1,793
참조	1,794
16.86 CREATE TABLE AS SELECT	1,795
기능	1,795
구문	1,795
사용 범위 및 접근 권한	1,797
구문 규칙 및 파라미터	1,797
설명	1,798
사용 예	1,798
호환성	1,800
참조	1,800
16.87 CREATE GLOBAL TEMPORARY TABLE	1,801
기능	1,801
구문	1,801
사용 범위 및 접근 권한	1,801
구문 규칙 및 파라미터	1,802
설명	1,802
사용 예	1,803
호환성	1,804
참조	1,805
16.88 CREATE TABLESPACE	1,806
기능	1,806
구문	1,806
사용 범위 및 접근 권한	1,806
구문 규칙 및 파라미터	1,806
설명	1,807
사용 예	1,807
호환성	1,807
참조	1,807
16.89 CREATE MEMORY DATA TABLESPACE	1,808
기능	1,808
구문	1,808
사용 범위 및 접근 권한	1,808
구문 규칙 및 파라미터	1,809
설명	1,810
사용 예	1,810

호환성	1,811
참조	1,811
16.90 CREATE MEMORY TEMPORARY TABLESPACE	1,812
기능	1,812
구문	1,812
사용 범위 및 접근 권한	1,812
구문 규칙 및 파라미터	1,813
설명	1,814
사용 예	1,814
호환성	1,814
참조	1,814
16.91 CREATE USER	1,816
기능	1,816
구문	1,816
사용 범위 및 접근 권한	1,816
구문 규칙 및 파라미터	1,817
설명	1,819
사용 예	1,819
호환성	1,821
참조	1,821
16.92 CREATE VIEW	1,822
기능	1,822
구문	1,822
사용 범위 및 접근 권한	1,822
구문 규칙 및 파라미터	1,823
설명	1,824
사용 예	1,825
호환성	1,826
참조	1,826
16.93 DECLARE cursor_name	1,827
기능	1,827
구문	1,827
사용 범위 및 접근 권한	1,828
구문 규칙 및 파라미터	1,828
설명	1,832
사용 예	1,834
호환성	1,838
참조	1,838
16.94 DELETE FROM	1,840

기능	1,840
구문	1,840
사용 범위 및 접근 권한	1,840
구문 규칙 및 파라미터	1,841
설명	1,842
사용 예	1,842
호환성	1,843
참조	1,843
16.95 DELETE FROM name RETURNING	1,844
기능	1,844
구문	1,844
사용 범위 및 접근 권한	1,844
구문 규칙 및 파라미터	1,845
설명	1,846
사용 예	1,846
호환성	1,847
참조	1,847
16.96 DELETE FROM name RETURNING .. INTO	1,848
기능	1,848
구문	1,848
사용 범위 및 접근 권한	1,848
구문 규칙 및 파라미터	1,849
설명	1,850
사용 예	1,850
호환성	1,850
참조	1,850
16.97 DELETE FROM name WHERE CURRENT OF cursor_name	1,852
기능	1,852
구문	1,852
사용 범위 및 접근 권한	1,852
구문 규칙 및 파라미터	1,852
설명	1,853
사용 예	1,853
호환성	1,854
참조	1,854
16.98 DROP AUDIT POLICY	1,856
기능	1,856
구문	1,856
사용 범위 및 접근 권한	1,856

구문 규칙 및 파라미터	1,856
설명	1,856
사용 예	1,857
호환성	1,857
참조	1,857
16.99 DROP CLUSTER GROUP	1,858
기능	1,858
구문	1,858
사용 범위 및 접근 권한	1,858
구문 규칙 및 파라미터	1,858
설명	1,859
사용 예	1,859
호환성	1,859
참조	1,859
16.100 DROP CLUSTER LOCATION	1,860
기능	1,860
구문	1,860
사용 범위 및 접근 권한	1,860
구문 규칙 및 파라미터	1,860
설명	1,860
사용 예	1,861
호환성	1,861
참조	1,861
16.101 DROP INDEX	1,862
기능	1,862
구문	1,862
사용 범위 및 접근 권한	1,862
구문 규칙 및 파라미터	1,862
설명	1,863
사용 예	1,863
호환성	1,863
참조	1,863
16.102 DROP PROFILE	1,864
기능	1,864
구문	1,864
사용 범위 및 접근 권한	1,864
구문 규칙 및 파라미터	1,864
사용 예	1,865
호환성	1,865

참조	1,865
16.103 DROP SCHEMA	1,866
기능	1,866
구문	1,866
사용 범위 및 접근 권한	1,866
구문 규칙 및 파라미터	1,866
설명	1,867
사용 예	1,867
호환성	1,867
참조	1,868
16.104 DROP SEQUENCE	1,869
기능	1,869
구문	1,869
사용 범위 및 접근 권한	1,869
구문 규칙 및 파라미터	1,869
설명	1,870
사용 예	1,870
호환성	1,870
참조	1,870
16.105 DROP SYNONYM	1,871
기능	1,871
구문	1,871
사용 범위 및 접근 권한	1,871
구문 규칙 및 파라미터	1,871
설명	1,872
사용 예	1,872
호환성	1,872
참조	1,872
16.106 DROP TABLE	1,873
기능	1,873
구문	1,873
사용 범위 및 접근 권한	1,873
구문 규칙 및 파라미터	1,873
설명	1,874
사용 예	1,874
호환성	1,875
참조	1,876
16.107 DROP TABLESPACE	1,877
기능	1,877

구문	1,877
사용 범위 및 접근 권한	1,877
구문 규칙 및 파라미터	1,877
설명	1,879
사용 예	1,879
호환성	1,879
참조	1,879
16.108 DROP USER	1,880
기능	1,880
구문	1,880
사용 범위 및 접근 권한	1,880
구문 규칙 및 파라미터	1,880
설명	1,882
사용 예	1,882
호환성	1,882
참조	1,883
16.109 DROP VIEW	1,884
기능	1,884
구문	1,884
사용 범위 및 접근 권한	1,884
구문 규칙 및 파라미터	1,884
설명	1,885
사용 예	1,885
호환성	1,885
참조	1,885
16.110 EXECUTE statement_name	1,886
기능	1,886
구문	1,886
사용 범위 및 접근 권한	1,886
구문 규칙 및 파라미터	1,886
설명	1,889
사용 예	1,889
호환성	1,889
참조	1,890
16.111 EXECUTE IMMEDIATE 'sql_string'	1,891
기능	1,891
구문	1,891
사용 범위 및 접근 권한	1,891
구문 규칙 및 파라미터	1,891

설명	1,892
사용 예	1,892
호환성	1,893
참조	1,893
16.112 FETCH cursor_name	1,894
기능	1,894
구문	1,894
구문 규칙 및 파라미터	1,894
설명	1,896
사용 예	1,896
호환성	1,899
참조	1,899
16.113 GRANT privileges TO	1,900
기능	1,900
구문	1,900
구문 규칙 및 파라미터	1,903
설명	1,908
사용 예	1,909
호환성	1,910
참조	1,911
16.114 INSERT INTO	1,912
기능	1,912
구문	1,912
사용 범위 및 접근 권한	1,912
구문 규칙 및 파라미터	1,913
설명	1,914
사용 예	1,914
호환성	1,916
참조	1,916
16.115 INSERT INTO name RETURNING	1,917
기능	1,917
구문	1,917
사용 범위 및 접근 권한	1,917
구문 규칙 및 파라미터	1,918
설명	1,919
사용 예	1,919
호환성	1,920
참조	1,920
16.116 INSERT INTO name RETURNING .. INTO	1,921

기능	1,921
구문	1,921
사용 범위 및 접근 권한	1,921
구문 규칙 및 파라미터	1,922
설명	1,923
사용 예	1,923
호환성	1,924
참조	1,924
16.117 LOCK TABLE	1,925
기능	1,925
구문	1,925
사용 범위 및 접근 권한	1,925
구문 규칙 및 파라미터	1,926
설명	1,927
사용 예	1,927
호환성	1,927
참조	1,928
16.118 NOAUDIT POLICY	1,929
기능	1,929
구문	1,929
사용 범위 및 접근 권한	1,929
구문 규칙 및 파라미터	1,929
설명	1,930
사용 예	1,933
호환성	1,933
참조	1,933
16.119 OPEN cursor_name	1,934
기능	1,934
구문	1,934
사용 범위 및 접근 권한	1,934
구문 규칙 및 파라미터	1,934
설명	1,935
사용 예	1,935
호환성	1,936
참조	1,937
16.120 PREPARE statement_name	1,938
기능	1,938
구문	1,938
사용 범위 및 접근 권한	1,938

구문 규칙 및 파라미터	1,938
설명	1,941
사용 예	1,941
호환성	1,942
참조	1,942
16.121 RELEASE SAVEPOINT savepoint_specifier	1,943
기능	1,943
구문	1,943
구문 규칙 및 파라미터	1,943
설명	1,943
사용 예	1,943
호환성	1,944
참조	1,944
16.122 REVOKE privileges FROM	1,945
기능	1,945
구문	1,945
구문 규칙 및 파라미터	1,945
설명	1,946
사용 예	1,947
호환성	1,948
참조	1,948
16.123 ROLLBACK	1,949
기능	1,949
구문	1,949
구문 규칙 및 파라미터	1,949
설명	1,950
사용 예	1,950
호환성	1,951
참조	1,952
16.124 SAVEPOINT savepoint_specifier	1,953
기능	1,953
구문	1,953
구문 규칙 및 파라미터	1,953
설명	1,953
사용 예	1,953
호환성	1,955
참조	1,955
16.125 SELECT	1,956
query expression	1,956

query specification	1,960
select list	1,965
from clause	1,968
joined table	1,975
where clause	1,981
group by clause	1,983
having clause	1,986
order by clause	1,988
offset limit clause	1,992
set operator	1,997
subquery	2,003
hint clause	2,007
16.126 SELECT .. FOR UPDATE	2,037
기능	2,037
구문	2,037
사용 범위 및 접근 권한	2,037
구문 규칙 및 파라미터	2,038
설명	2,039
사용 예	2,040
호환성	2,041
16.127 SELECT .. INTO	2,042
기능	2,042
구문	2,042
사용 범위 및 접근 권한	2,042
구문 규칙 및 파라미터	2,042
설명	2,043
사용 예	2,044
16.128 SELECT .. INTO .. FOR UPDATE	2,045
기능	2,045
구문	2,045
사용 범위 및 접근 권한	2,045
구문 규칙 및 파라미터	2,046
설명	2,048
사용 예	2,049
참조	2,050
16.129 SET CONSTRAINTS	2,051
기능	2,051
구문	2,051
사용 범위 및 접근 권한	2,051

구문 규칙 및 파라미터	2,051
설명	2,052
사용 예	2,059
호환성	2,059
참조	2,059
16.130 SET SESSION AUTHORIZATION user_identifier	2,060
기능	2,060
구문	2,060
사용 범위 및 접근 권한	2,060
구문 규칙 및 파라미터	2,060
설명	2,061
사용 예	2,061
호환성	2,061
16.131 SET SESSION CHARACTERISTICS AS transaction_mode	2,062
기능	2,062
구문	2,062
구문 규칙 및 파라미터	2,062
설명	2,063
사용 예	2,063
호환성	2,063
참조	2,063
16.132 SET TIME ZONE	2,064
기능	2,064
구문	2,064
구문 규칙 및 파라미터	2,064
설명	2,064
사용 예	2,065
호환성	2,065
16.133 SET TRANSACTION transaction_mode	2,066
기능	2,066
구문	2,066
구문 규칙 및 파라미터	2,066
설명	2,067
사용 예	2,067
호환성	2,067
참조	2,067
16.134 TRUNCATE TABLE	2,068
기능	2,068
구문	2,068

사용 범위 및 접근 권한	2,068
구문 규칙 및 파라미터	2,068
설명	2,069
사용 예	2,069
호환성	2,070
16.135 UPDATE	2,071
기능	2,071
구문	2,071
사용 범위 및 접근 권한	2,071
구문 규칙 및 파라미터	2,072
설명	2,073
사용 예	2,074
호환성	2,075
16.136 UPDATE name RETURNING	2,076
기능	2,076
구문	2,076
사용 범위 및 접근 권한	2,077
구문 규칙 및 파라미터	2,077
설명	2,078
사용 예	2,079
호환성	2,079
16.137 UPDATE name RETURNING .. INTO	2,080
기능	2,080
구문	2,080
사용 범위 및 접근 권한	2,081
구문 규칙 및 파라미터	2,081
설명	2,082
사용 예	2,082
호환성	2,083
16.138 UPDATE name WHERE CURRENT OF cursor_name	2,084
기능	2,084
구문	2,084
사용 범위 및 접근 권한	2,084
구문 규칙 및 파라미터	2,084
설명	2,085
사용 예	2,085
호환성	2,087
참조	2,087

11.

SQL Elements

11.1 Syntax Elements

Identifiers

Identifier는 ordinary identifier와 delimited identifier로 나뉜다.

Ordinary identifier는 문자 또는 문자와 숫자로 구성된 identifier로써 내부적으로 모든 문자를 대문자로 치환하여 사용한다. 따라서 대소문자를 구분하지 않는다.

다음은 ordinary identifier의 예이다.

```
GOLDILOCKS
GoldiLocks
```

Delimited identifier는 double quote (")를 시작과 끝에 기술한 문자 또는 문자와 숫자로 구성된 identifier로써 내부적으로 해당 문자를 모두 기술한 그대로 사용한다. 따라서 delimited identifier를 사용할 경우 대소문자를 구분한다.

다음은 delimited identifier의 예이다.

```
"GOLDILOCKS"
"GoldiLocks"
```

Literals

Literals는 null이 아닌 값을 기술한 것이다.

Text Literals

Text literals는 string이나 binary string을 기술한 것이다.

String의 시작과 끝에 single quote (')를 작성하여 string에 대한 text literals를 사용할 수 있다. Double quote (") 뿐만 아니라 single quote (')를 제외한 모든 문자열이 single quote (') 안에 작성하는 string이 될 수 있다. 만약 string에 single quote (')를 사용하려면 single quote (')를 공백없이 연속으로 두 번 작성해야 한다. String에는 최대 4,000 문자까지 작성할 수 있다.

다음은 string에 대한 text literals를 작성하는 예이다.

```
'GOLDILOCKS'
'Sunje's DBMS'
```

Binary string에 대한 text literals는 x' 또는 X'로 시작하고 '로 끝나는 16진수의 string을 기술한다. 16진수의 string은 각 자리에 0 ~ 9, A (a) ~ F (f)에 해당하는 문자만 기술할 수 있으며, 두 자리의 문자가 하나의 byte를 의미하므로 항상 짝수 자리를 작성해야 한다. Binary string은 최대 4,000 문자까지 작성할 수 있다.

다음은 binary string에 대한 text literals를 작성하는 예이다.

```
x'001f'
X'FF0A'
x'aF37BBc013'
```

Numeric Literals

Numeric literals는 숫자타입의 literals를 작성하는 형식으로써 정수나 소수점 이하의 자리를 갖는 숫자를 작성할 수 있다. Numeric literals에 대한 문법은 다음과 같다.

```
[ + | - ] <digits> [ . <digits> ] [ E | e [ + | - ] <digits> ] [ f | F | d | D ]
```

- 제일 앞에 나오는 +, - 는 숫자 전체의 부호가 양수인지 음수인지를 나타낸다. +, - 는 생략 가능하며 생략시 +로 간주된다.
- <digits>에는 0 ~ 9 사이의 숫자들을 공백없이 나열할 수 있고, .을 이용하여 소수점 이하의 숫자까지 작성할 수 있다.
- 첫 번째 <digits> 뒤에는 exponent 형식으로 작성할 수 있는데, 이를 위해 exponent를 의미하는 E 또는 e를 작성한다. 그 뒤에는 exponent의 부호인 +, -를 작성하고 다음으로 <digits>를 작성한다. Exponent의 부호도 생략 가능하며 생략시 +로 간주한다.
- 마지막으로 f, F, d, D 등의 문자가 숫자 뒤에 올 수 있는데 이 문자는 BINARY_FLOAT, BINARY_DOUBLE의 숫자임을 알려준다. 이런 문자를 기술하지 않는 경우 NUMBER 타입의 숫자인 것으로 간주한다.

다음은 numeric literals를 작성하는 예이다.

```
20
+123.45
0.03
+1.23E-02
-1.5
10f
+123.45F
1.2E-3F
-22d
123.45D
-1.23E+05D
```

Datetime Literals

Datetime literals는 날짜/ 시간 타입에 대한 literals를 작성하는 형식이다. Datetime value는 string literal을 사용하여 지정하거나 TO_* 함수(TO_DATE 등)를 이용해 character 또는 numeric value를 변환하여 지정할 수도 있다.

날짜/시간 타입에는 DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 이 있다.

Date Literals

Date literals는 DATE 'string literal' 또는 TO_DATE(string_literal [, format]) 의 형태로 작성할 수 있다.

- DATE 'string literal'
 - Date 타입 format은 'YYYY-MM-DD' 이다.
 - DATE '2002-07-15'
- TO_DATE(string_literal [, format])
 - Format을 지정하지 않은 경우, date 타입의 format은 NLS_DATE_FORMAT이고,
 - Format을 지정한 경우, 지정된 format을 적용한다.
- Date 타입은 년월일시분초를 포함한다. (fractional seconds (소수점이하초)는 제외)
- 날짜를 기술하지 않으면, 현재 월의 첫째날로 설정된다.
- 시분초를 기술하지 않으면 기본값인 자정으로 설정된다.
 - HH24 format인 경우 '00:00:00' 이다.
 - HH12 format인 경우 '12:00:00' 이다.
- 시분초가 포함된 DATE value의 시분초를 기본값인 자정으로 설정하려면 TRUNC(date)를 이용해야 한다.
 - 예를 들어 TRUNC(SYSDATE): SYSDATE는 년월일시분초가 모두 포함된 값이다.
- Date value들을 시분초를 제외하고 년월일값만 비교하고자 하는 경우, TRUNC 함수를 이용해서 시분초값을 자정으로 설정해야 한다.

자세한 내용은 TO_DATE, Datetime Format 문자열, NLS_DATE_FORMAT을 참조한다.

다음은 date literals를 작성하는 예이다.

DATE '2002-07-15'

- Format을 지정하지 않은 경우로 NLS_DATE_FORMAT = 'YYYY-MM-DD' 일 때

TO_DATE('2002-07-15')

- Format을 지정한 경우

TO_DATE('15-JUL-02', 'DD-MON-RR')

TO_DATE('2002-07-15 00:00:00', 'YYYY-MM-DD HH24:MI:SS')

TO_DATE('2002-07-15 13:25:30', 'YYYY-MM-DD HH24:MI:SS')

- DATE 타입에 날짜를 기술하지 않은 경우 (현재 월의 첫째날로 저장)

```
gSQL> SELECT TO_DATE( '2000-07', 'YYYY-MM' ) FROM DUAL;
TO_DATE( '2000-07', 'YYYY-MM' )
-----
2000-07-01
```

- DATE 타입에 시분초를 기술하지 않은 경우 (시분초값이 자정으로 저장됨)

```
gSQL> SELECT
      TO_CHAR( DATE'2002-07-15', 'YYYY-MM-DD HH24:MI:SS' ) AS RESULT
    FROM DUAL;
RESULT
-----
2002-07-15 00:00:00
```

- 시분초가 포함된 DATE value(SYSDATE)의 시분초를 기본값인 자정으로 설정한 경우

```
gSQL> SELECT
      TO_CHAR( SYSDATE,
              'YYYY-MM-DD HH24:MI:SS' ) AS RESULT_SYSDATE,
      TO_CHAR( TRUNC( SYSDATE ),
              'YYYY-MM-DD HH24:MI:SS' ) AS RESULT_TRUNC_SYSDATE
    FROM DUAL;
RESULT_SYSDATE      RESULT_TRUNC_SYSDATE
-----
2014-08-19 10:06:49 2014-08-19 00:00:00
```

- Date value에서 시분초를 제외하고 년월일만 비교하는 경우

```
gSQL> SELECT
      TO_DATE( '2002-08-12' ) =
      TRUNC( TO_DATE( '2002-08-12 23:59:59', 'YYYY-MM-DD HH24:MI:SS' ) )
    AS RESULT FROM DUAL;
RESULT
-----
TRUE
```

Time Literals

Time literals는 TIME'string literal' 또는 TO_TIME(string_literal [, format])의 형태로 작성할 수 있다.

- TIME'string literal'

- Time 타입 format은 'HH24:MI:SS[.FF6]]'이다.
- TIME'15:30:59.999999'
- TO_TIME(string_literal [, format])
 - Format을 지정하지 않은 경우, time 타입의 format은 NLS_TIME_FORMAT 이고,
 - Format을 지정한 경우, 지정된 format을 적용한다.

Time 타입은 시분초, fractional seconds (소수점이하초)를 포함한다.

Fractional seconds는 최대 여섯 자리 숫자의 형식을 지정하여 작성할 수 있다.

자세한 내용은 TO_TIME, Datetime Format 문자열, NLS_TIME_FORMAT을 참조한다.

다음은 time literals를 작성하는 예이다.

```
TIME '15:30:59.999999'
```

- Format을 지정하지 않은 경우로써 NLS_TIME_FORMAT = 'HH24:MI:SS.FF6' 일 때

```
TO_TIME( '15:30:59.999999' )
```

- Format을 지정한 경우

```
TO_TIME( '09.45.03.546873 AM', 'HH12.MI.SS.FF6 AM' )
```

```
TO_TIME( '09:45:03', 'HH12:MI:SS' )
```

Time with Time Zone Literals

Time with time zone literals는 TIME'string literal', TIME WITH TIME ZONE'string literal' 또는 TO_TIME_WITH_TIME_ZONE(string_literal [, format]) , TO_TIME_TZ(string_literal [, format])의 형태로 작성할 수 있다.

- TIME'string literal' 또는 TIME WITH TIME ZONE'string literal'
 - Time with time zone 타입의 format은 'HH24:MI:SS[.FF6]] TZH:TZM' 이다.
 - TIME'15:30:59.999999 +09:00'
 - TIME WITH TIME ZONE'15:30:59.999999 +09:00'
- TO_TIME_WITH_TIME_ZONE(string_literal [, format])
 - Format을 지정하지 않은 경우, time with time zone 타입의 format은 NLS_TIME_WITH_TIME_ZONE_FORMAT 이고,
 - Format을 지정한 경우, 지정된 format을 적용한다.

Time with time zone 타입은 시분초, fractional seconds (소수점이하초), time zone offset (time zone hour, time zone minute)를 포함한다.

Fractional seconds는 최대 여섯 자리 숫자의 형식을 지정하여 작성할 수 있다.

자세한 내용은 TO_TIME_WITH_TIME_ZONE, Datetime Format 문자열, NLS_TIME_WITH_TIME_ZONE_FORMAT을 참조한다.

다음은 time with time zone literals를 작성하는 예이다.

```
TIME '15:30:59.999999 +09:00'
TIME WITH TIME ZONE '15:30:59.999999 +09:00'
```

- Format을 지정하지 않은 경우로써 NLS_TIME_WITH_TIME_ZONE_FORMAT = 'HH24:MI:SS.FF6 TZh:TZM' 일 때

```
TO_TIME_WITH_TIME_ZONE( '15:30:59.999999 +09:00' )
TO_TIME_TZ( '15:30:59.999999 +09:00' )
```

- Format을 지정한 경우

```
TO_TIME_WITH_TIME_ZONE( '09.45.03.546873 +09:00 AM',
                        'HH12.MI.SS.FF6 TZh:TZM AM' )
```

Timestamp Literals

Timestamp literals는 `TIMESTAMP'string literal'` 또는 `TO_TIMESTAMP(string_literal [, format])`의 형태로 작성할 수 있다.

- `TIMESTAMP'string literal'`
 - Timestamp 타입의 format은 'YYYY-MM-DD HH24:MI:SS[.FF6]' 이다.
 - `TIMESTAMP'2002-07-15 15:39:59.999999'`
- `TO_TIMESTAMP(string_literal [, format])`
 - Format을 지정하지 않은 경우, timestamp 타입의 format은 NLS_TIMESTAMP_FORMAT이고
 - Format을 지정한 경우, 지정된 format을 적용한다.

Timestamp 타입은 년월일, 시분초, fractional seconds (소수점이하초)를 포함한다.

Fractional seconds는 최대 여섯 자리 숫자의 형식을 지정하여 작성할 수 있다.

자세한 내용은 `TO_TIMESTAMP`, `Datetime Format 문자열`, `NLS_TIMESTAMP_FORMAT`을 참조한다.

다음은 timestamp literals를 작성하는 예이다.

```
TIMESTAMP '2002-07-15 15:39:59.999999'
```

- Format을 지정하지 않은 경우로써 NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6' 일 때

```
TO_TIMESTAMP( '2002-07-15 15:39:59.999999' )
```

- Format을 지정한 경우

```
TO_TIMESTAMP( '15-JUL-02 11.06.30.123456 AM',
              'DD-MON-RR HH12.MI.SS.FF6 AM' )
```

Timestamp with Time Zone Literals

Timestamp with time zone literals는 `TIMESTAMP'string literal'`, `TIMESTAMP WITH TIME ZONE'string literal'` 또는 `TO_TIMESTAMP_WITH_TIME_ZONE(string_literal [, format] )`, `TO_TIMESTAMP_TZ(string_literal [, format])`의 형태로 작성할 수 있다.

- `TIMESTAMP'string literal'` 또는 `TIMESTAMP WITH TIME ZONE'string literal'`
 - Timestamp with time zone 타입의 format은 'YYYY-MM-DD HH24:MI:SS[.FF6]] TZH:TZM'이다.
 - `TIMESTAMP'2002-07-15 15:39:59.999999 +09:00'`
 - `TIMESTAMP WITH TIME ZONE'2002-07-15 15:39:59.999999 +09:00'`
- `TO_TIMESTAMP_WITH_TIME_ZONE(string_literal [, format] )`
 - Format을 지정하지 않은 경우, timestamp with time zone 타입의 format은 `NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT` 이고,
 - Format을 지정한 경우, 지정된 format을 적용한다.

Timestamp with time zone 타입은 년월일, 시분초, fractional seconds (소수점이하초), time zone offset (time zone hour, timezone minute)을 포함한다.

Fractional seconds는 최대 여섯 자리 숫자의 형식을 지정하여 작성할 수 있다.

자세한 내용은 `TO_TIMESTAMP_WITH_TIME_ZONE`, `Datetime Format 문자열`, `NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT`을 참조한다.

다음은 timestamp with time zone literals를 작성하는 예이다.

```
TIMESTAMP'2002-07-15 15:39:59.999999 +09:00'
TIMESTAMP WITH TIME ZONE'2002-07-15 15:39:59.999999 +09:00'
```

- FORMAT을 지정하지 않은 경우로써 `NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'` 일 때

```
TO_TIMESTAMP_WITH_TIME_ZONE( '2002-07-15 15:39:59.999999 +09:00' )
TO_TIMESTAMP_TZ( '2002-07-15 15:39:59.999999 +09:00' )
```

- FORMAT을 지정한 경우

```
TO_TIMESTAMP_WITH_TIME_ZONE( '15-JUL-02 11.06.30.123456 +09:00 AM',
                              'DD-MON-RR HH12.MI.SS.FF6 TZH:TZM AM' )
TO_TIMESTAMP_TZ( '15-JUL-02 11.06.30.123456 +09:00 AM',
                  'DD-MON-RR HH12.MI.SS.FF6 TZH:TZM AM' )
```


Interval Literals

Interval literals는 시간의 간격을 지정한다.

Interval은 크게 두 가지로 분류되며 다음과 같이 표현한다.

- Year-month INTERVAL values
 - YEAR와 MONTH를 포함한다.
 - Display string 표현: 'year-month'
- Day-time INTERVAL values
 - DAY, HOUR, MINUTE, SECOND (fractional seconds 포함)를 포함한다.
 - Display string 표현: 'day hour:minute:second.fractional_seconds'
- Interval value의 부호는 string 표현의 제일 앞에 한 번만 지정할 수 있다.
 예: INTERVAL'+3 11:22:33.999999'DAY TO SECOND (O)
 INTERVAL'-3 +11:22:33.999999'DAY TO SECOND (X)

Interval 타입 목록은 다음과 같다.

- INTERVAL YEAR (leading precision)
- INTERVAL MONTH (leading precision)
- INTERVAL YEAR (leading precision) TO MONTH
- INTERVAL DAY (leading precision)
- INTERVAL HOUR (leading precision)
- INTERVAL MINUTE (leading precision)
- INTERVAL SECOND (leading precision[, fractional seconds precision])
- INTERVAL DAY (leading precision) TO HOUR
- INTERVAL DAY (leading precision) TO MINUTE
- INTERVAL DAY (leading precision) TO SECOND (fractional seconds precision)
- INTERVAL HOUR (leading precision) TO MINUTE
- INTERVAL HOUR (leading precision) TO SECOND (fractional seconds precision)
- INTERVAL MINUTE (leading precision) TO SECOND (fractional seconds precision)

Leading precision

- 해당 field의 자리수로써 2 ~ 6까지 지정할 수 있으며, 지정하지 않을 경우의 기본값은 2이다.
- Leading field 값이 leading precision 값을 초과하면 에러를 반환한다.

Fractional seconds precision

- Fractional seconds의 자리수로써 0 ~ 6까지 지정할 수 있으며, 지정하지 않을 경우의 기본값은 6이다.
- Fractional second field 값이 fractional seconds precision 값을 초과하면 반올림된다.

자세한 내용은 INTERVAL, INTERVAL * TO * 에서 두 번째 이후 field의 precision과 값의 범위 를 참조한다.

Interval literals의 사용 예

다음은 interval literals를 사용하는 예들이다.

Interval YEAR

다음은 interval YEAR literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'YEAR INTERVAL'01-00'YEAR	1 year	+01-00
INTERVAL'100'YEAR	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL'100'YEAR(3)	100 year	+100-00
INTERVAL'+999999'YEAR(6)	999999 year	+999999-00
INTERVAL'-999999'YEAR(6)	-(999999 year)	-999999-00

Interval MONTH

다음은 interval MONTH literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'MONTH INTERVAL'00-01'MONTH	1 month	+00-01
INTERVAL'100'MONTH	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL'100'MONTH(3)	8 year 4 month	+008-04
INTERVAL'+999999'MONTH(6)	83333 year 3 month	+083333-03
INTERVAL'-999999'MONTH(6)	-(83333 year 3 month)	-083333-03

Interval YEAR TO MONTH

다음은 interval YEAR TO MONTH literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1-06'YEAR TO MONTH	1 year 6 month	+01-06
INTERVAL'1-12'YEAR TO MONTH	Month value가 11을 초과하여 예러가 반환된다.	-
INTERVAL'100-11'YEAR TO MONTH	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL'100-11'YEAR(3) TO MONTH	100 year 11 month	+100-11
INTERVAL'+999999-11'YEAR(6) TO MONTH	999999 year 11 month	+999999-11
INTERVAL'-999999-11'YEAR(6) TO MONTH	-(999999 year 11 month)	-999999-11

Interval DAY

다음은 interval DAY literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'DAY INTERVAL'01 00:00:00'DAY	1 day	+01 00:00:00
INTERVAL'100'DAY	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL'100'DAY(3)	100 day	+100 00:00:00
INTERVAL'+999999'DAY(6)	999999 day	+999999 00:00:00
INTERVAL'-999999'DAY(6)	-(999999 day)	-999999 00:00:00

Interval HOUR

다음은 interval HOUR literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'HOUR INTERVAL'00 01:00:00'HOUR	1 hour	+00 01:00:00
INTERVAL'1000'HOUR(3)	Leading precision 3을 초과하여 예러가 반환된다.	-
INTERVAL'1000'HOUR(4)	41 day 16 hour	+0041 16:00:00
INTERVAL'+999999'HOUR(6)	41666 day 15 hour	+041666 15:00:00
INTERVAL'-999999'HOUR(6)	-(41666 day 15 hour)	-041666 15:00:00

Interval MINUTE

다음은 interval MINUTE literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'MINUTE INTERVAL'00 00:01:00'MINUTE	1 minute	+00 00:01:00
INTERVAL'12345'MINUTE(4)	Leading precision 4를 초과하여 예러가 반환된다.	-
INTERVAL'12345'MINUTE(5)	8 day 13 hour 45 minute	+00008 13:45:00
INTERVAL'+999999'MINUTE(6)	694 day 10 hour 39 minute	+000694 10:39:00
INTERVAL'-999999'MINUTE(6)	-(694 day 10 hour 39 minute)	-000694 10:39:00

Interval SECOND

다음은 interval SECOND literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1'SECOND INTERVAL'00 00:00:01.000000'SECOND	1 second	+00 00:00:01.000000
INTERVAL'100'SECOND	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL'99.9999999'SECOND	Fractional seconds가 반올림되어 1 00 second가 되므로 leading preci	-

예	설명	Display string
INTERVAL'99.9999999'SECOND(2,6)	sion 2를 초과하여 에러가 반환된다.	
INTERVAL'99.9999999'SECOND(3)	1 minute 40 second	+000 00:01:40.000000
INTERVAL'29.506167'SECOND(2, 2)	29.51 second	+00 00:00:29.51
INTERVAL'999999.999999'SECOND(6,6)	11day 13 hour 46 minute 39.99 9999 second	+000011 13:46:39.999999
INTERVAL'-999999.999999'SECOND(6,6)	-(11day 13 hour 46 minute 39.9 99999 second)	-000011 13:46:39.999999

Interval DAY TO HOUR

다음은 interval DAY TO HOUR literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1 23'DAY TO HOUR INTERVAL'01 23:00:00'DAY TO HOUR	1 day 23 hour	+01 23:00:00
INTERVAL'1 24'DAY TO HOUR	Hour value가 23을 초과한 invalid 값으로써 에러를 반환한다.	-
INTERVAL'100 23'DAY TO HOUR	Leading precision 2를 초과하여 에러가 반환된다.	-
INTERVAL'100 23'DAY(3) TO HOUR	100 day 23 hour	+100 23:00:00
INTERVAL'+999999 23'DAY(6) TO HOUR	999999 day 23 hour	+999999 23:00:00
INTERVAL'-999999 23'DAY(6) TO HOUR	-(999999 day 23 hour)	-999999 23:00:00
INTERVAL'-999999 +23'DAY(6) TO HOUR	부호 지정 오류로 인한 에러이다.	-

Interval DAY TO MINUTE

다음은 interval DAY TO MINUTE literals를 사용하는 예이다.

예	설명	Display string
INTERVAL'1 23:59'DAY TO MINUTE INTERVAL'01 23:59:00'DAY TO MINUTE	1 day 23 hour 59 second	+01 23:59:00
INTERVAL'1 24:59'DAY TO MINUTE	Hour value가 23을 초과한 invalid 값으로써 에러가 반환된다.	-
INTERVAL'1 23:60'DAY TO MINUTE	Minute value가 59를 초과한 invalid 값으로써 에러가 반환된다.	-
INTERVAL'100 23:59'DAY TO MINUTE	Leading precision 2를 초과하여 에러가 반환된다.	-
INTERVAL'100 23:59'DAY(3) TO MINUTE	100 day 23 hour 59 minute	+100 23:59:00
INTERVAL'+999999 23:59'DAY(6) TO MINUTE	999999 day 23 hour 59 minute	+999999 23:59:00
INTERVAL'-999999 23:59'DAY(6) TO MINUTE	-(999999 day 23 hour 59 minute)	-999999 23:59:00

Interval DAY TO SECOND

다음은 interval DAY TO SECOND literals를 사용하는 예이다.

예	설명	Display string
INTERVAL '1 23:59:59.999999'DAY TO SECOND	1 day 23 hour 59 minute 59.999999 second	+01 23:59:59.999999
INTERVAL '1 24:59:59.999999'DAY TO SECOND	Hour value가 23을 초과하여 예러가 반환된다.	-
INTERVAL '1 23:60:59.999999'DAY TO SECOND	Minute value가 59를 초과하여 예러가 반환된다.	-
INTERVAL '1 23:59:60.999999'DAY TO SECOND	Second value가 60을 초과하여 예러가 반환된다.	-
INTERVAL '99 23:59:59.999999'DAY TO SECOND	Fractional seconds가 반올림되어 100 day가 되므로 leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL '99 23:59:59.999999'DAY(3) TO SECOND	100 day	+100 00:00:00.000000
INTERVAL '1 11:22:33.567890'DAY(2) TO SECOND(2)	1 day 11 hour 22 minute 33.57 second	+01 11:22:33.57
INTERVAL '+999999 23:59:59.999999'DAY(6) TO SECOND(6)	999999 day 23 hour 59 minute 59.999999 hour	+999999 23:59:59.999999
INTERVAL '-999999 23:59:59.999999'DAY(6) TO SECOND(6)	-(999999 day 23 hour 59 minute 59.999999 hour)	-999999 23:59:59.999999

Interval HOUR TO MINUTE

다음은 interval HOUR TO MINUTE literals를 사용하는 예이다.

예	설명	Display string
INTERVAL '23:59'HOUR TO MINUTE INTERVAL '00 23:59:00'HOUR TO MINUTE	23 hour 59 minute	+00 23:59:00
INTERVAL '23:60'HOUR TO MINUTE	Minute value가 59를 초과하여 예러가 반환된다.	-
INTERVAL '100:59'HOUR TO MINUTE	Leading precision 2를 초과하여 예러가 반환된다.	-
INTERVAL '100:59'HOUR(3) TO MINUTE	4 day 4 hour 59 minute	+004 04:59:00
INTERVAL '+999999:59'HOUR(6) TO MINUTE	41666 day 15 hour 59 minute	+041666 15:59:00
INTERVAL '-999999:59'HOUR(6) TO MINUTE	-(41666 day 15 hour 59 minute)	-041666 15:59:00

Interval HOUR TO SECOND

다음은 interval HOUR TO SECOND literals를 사용하는 예이다.

예	설명	Display string
INTERVAL '23:59:59.999999'HOUR TO SECOND INTERVAL '00 23:59:59.999999'HOUR TO SECON D	23 hour 59 minute 59.99999 9 second	+00 23:59:59.999999
INTERVAL '23:60:59.999999'HOUR TO SECOND	Minute value가 59를 초과하여 에러가 반환된다.	-
INTERVAL '23:59:60.999999'HOUR TO SECOND	Second value가 59를 초과하여 에러가 반환된다.	-
INTERVAL '99:59:59.9999999'HOUR TO SECOND	Fractional seconds가 반올림되 어 100 hour가 되므로 leading precision 2를 초과하여 에러가 반환된다.	-
INTERVAL '99:59:59.9999999'HOUR(3) TO SECO ND	4 day 4 hour	+004 04:00:00.000000
INTERVAL '11:22:29.569'HOUR(3) TO SECOND(1)	11 hour 22 minute 29.6 seco nd	+000 11:22:29.6
INTERVAL '+9999999:59:59.999999'HOUR(6) TO S ECOND(6)	41666 day 15 hour 59 minute 59.999999 second	+041666 15:59:59.999999
INTERVAL '-9999999:59:59.999999'HOUR(6) TO S ECOND(6)	-(41666 day 15 hour 59 minu te 59.999999 second)	-041666 15:59:59.999999

Interval MINUTE TO SECOND

다음은 interval MINUTE TO SECOND literals를 사용하는 예이다.

예	설명	Display string
INTERVAL '15:23.123456'MINUTE TO SECOND INTERVAL '00 00:15:23.123456'MINUTE TO SECO ND	15 minute 23.123456 second	+00 00:15:23.123456
INTERVAL '15:60.123456'MINUTE TO SECOND	second value가 59를 초과하여 에러가 반환된다.	-
INTERVAL '99:59.999999'MINUTE TO SECOND(2)	fractional seconds가 반올림되 어 100 minute가 되므로, leadin g precision 2를 초과하여 에러가 반환된다.	-
INTERVAL '99:59.999999'MINUTE(3) TO SECOND (2)	1 hour 40 minute	+000 01:40:00.00
INTERVAL '+9999999:59.999999'MINUTE(6) TO SE COND(6)	694 day 10 hour 39 minute 5 9.999999 second	+000694 10:39:59.999999
INTERVAL '-9999999:59.999999'MINUTE(6) TO SE	-(694 day 10 hour 39 minute	

예	설명	Display string
COND(6)	59.999999 second)	-000694 10:39:59.999999

Null Value

Null value는 알 수 없는 값 또는 정의되지 않은 값이다. 모든 data type 값이 null value가 될 수 있다.

Boolean type의 unknown 값은 null value로 대체되어 표현된다.

Null value는 keyword로 정의되어 있으며 대소문자 구분없이 사용한다.

다음은 null value를 기술하는 예이다.

```
NULL
Null
```

Comments

Single Line Comments

Single line comments는 -- 또는 //로 시작하는 comment 이다. Single line comments는 해당 comment 기호 뒤부터 해당 라인의 끝까지를 comment로 처리한다.

다음은 single line comments를 사용하는 예이다.

```
gSQL> SELECT I1, -- I2, I3,
2 I4, I5
3 FROM T1;
I1      I4      I5
-----
column i1 column i4 column i5
1 row selected.
gSQL> SELECT I1, // I2, I3,
2 I4, I5
3 FROM T1;
I1      I4      I5
-----
column i1 column i4 column i5
1 row selected.
```

Multiple Line Comments

Multiple line comments는 /*로 시작해서 */로 끝나는 comment 이다. Multiple line comments는 /*부터 */까지를 comment로 설정하며 여러 라인에 걸쳐 comment를 기술할 수 있다.

다음은 multiple line comments를 사용하는 예이다.

```
gSQL> SELECT I1, I2, I3, I4, I5
2 /* TABLE T1에 대하여
3   모든 COLUMN들을 출력한다. */
4 FROM T1;
I1      I2      I3      I4      I5
-----
column i1 column i2 column i3 column i4 column i5
1 row selected.
```

Hint Comments

Hint comments는 /*+로 시작하고, */로 끝나는 comment 이다. Hint comments는 multiple line comments와 비슷하지만 시작 기호에 +가 더 있다는 점이 다르다. Hint comments에서 시작기호의 *와 + 사이에 공백이 존재하면 multiple line comments로 처리되는 것에 주의한다.

Hint comments는 다른 comment들과 달리 사용 가능한 위치가 SELECT 키워드의 바로 다음으로 지정되어 있다. Hint comments에는 사용자가 GOLDILOCKS의 optimizer에게 처리 방법 등을 지정하는 내용이 기술되어 있는데 자세한 내용은 **hint clause**를 참조한다.

다음은 hint comments를 사용하는 예이다.

```
gSQL> SELECT /*+ FULL(T1) */ * FROM T1;
I1      I2      I3      I4      I5
-----
column i1 column i2 column i3 column i4 column i5
1 row selected.
```

SQL Reserved Words and Keywords

SQL Reserved Words

GOLDILOCKS에는 SQL reserved words로 지정된 reserved word가 있으며, 해당 SQL reserved words들은 해당 사용 위치가 아닌 곳에서 single quote (') 없이 사용할 수 없다. 하지만 single quote (')를 이용한 SQL reserved words의 사용은 권장하지 않는다.

다음은 GOLDILOCKS SQL reserved words인데 * 표시한 것은 SQL standard에서 명시한 reserved words이다. 해당 리스트는 **V\$RESERVED_WORDS** view를 통해 검색할 수 있다.

ABSOLUTE
ACCESS
ALL *
ALLOCATE *
ALTER *
AND *
ANY *
ARE *
AS *
ASYMMETRIC *
AT *
AUTHORIZATION *
BEGIN *
BETWEEN *
BOTH *
BY *
CALL *
CASE *
CHECK *
CLOSE *
COLUMN *
COMMENT
COMMIT *
CONNECT *
CONSTRAINT *
CREATE *
CROSS *
CURRENT *
CURRENT_CATALOG *
CURRENT_DATE *
CURRENT_DEFAULT_TRANSFORM_GROUP *
CURRENT_PATH *
CURRENT_ROLE *
CURRENT_ROW *
CURRENT_SCHEMA *
CURRENT_TIME *
CURRENT_TIMESTAMP *
CURRENT_TRANSFORM_GROUP_FOR_TYPE *
CURRENT_USER *

DATABASE
DEALLOCATE *
DECLARE *
DEFAULT *
DELETE *
DEREF *
DESCRIBE *
DETERMINISTIC *
DISCONNECT *
DISTINCT *
DROP *
ELSE *
END *
END_EXEC *
ESCAPE *
EXCEPT *
EXEC *
EXECUTE *
EXISTS *
FALSE *
FETCH *
FILTER *
FIRST
FOR *
FOREIGN *
FREE *
FROM *
FULL *
FUNCTION *
GET *
GLOBAL *
GRANT *
GROUP *
HAVING *
HOLD *
IDENTIFIED
IF
IMMEDIATE
IN *
INDICATOR *
INNER *

INOUT *
INSERT *
INTERSECT *
INTO *
IS *
JOIN *
LAST
LEADING *
LEFT *
LIKE *
LIMIT
LOCAL *
LOCALTIME *
LOCALTIMESTAMP *
MATCH *
MEMBER *
MERGE *
MINUS
NATURAL *
NEW *
NEXT
NOT *
NULL *
OF *
OFFSET *
OLD *
ON *
OPEN *
OR *
ORDER *
OUT *
PREPARE *
PRIMARY *
PRIOR
PROCEDURE *
PROFILE
REF *
REFERENCES *
RELATIVE
RELEASE *
RENAME

RETURN *
RETURNING
RETURNS *
REVOKE *
RIGHT *
ROLLBACK *
ROW *
ROWID
ROWS *
ROW_NUMBER *
SAVEPOINT *
SELECT *
SESSION_USER *
SET *
SOME *
SQL *
SQLEXCEPTION *
SQLSTATE *
SQLWARNING *
START *
SYMMETRIC *
SYNONYM
SYSDATE
SYSTEM *
SYSTEM_USER *
SYSTIME
SYSTIMESTAMP
TABLE *
THEN *
TO *
TRAILING *
TRIGGER *
TRUE *
TRUNCATE *
UNION *
UNIQUE *
UNKNOWN *
UPDATE *
UPPER *
USER *
USING *

VALUES *

VIEW

WHEN *

WHENEVER *

WHERE *

WINDOW *

WITH *

WITHOUT *

SQL Keywords

GOLDILOCKS SQL keywords는 reserved word가 아니다. 그러나 GOLDILOCKS 내부적으로 사용하는 keyword 이므로 GOLDILOCKS SQL keywords를 사용할 경우 결과의 가독성이 떨어질 수 있어 사용을 권장하지 않는다.

GOLDILOCKS SQL keywords 목록은 **V\$KEYWORDS** view를 통해 검색할 수 있다.

호환성

Syntax element에 대한 SQL 표준 호환성은 다음과 같다.

표 11-1 SQL 표준 호환성

Feature ID	설명	지원 여부
E021-03	Character literals	O
E131	Null value support (nulls in lieu of values)	O
E161	SQL comments using leading double minus	O
F051-01	DATE data type (including support of DATE literal)	O
F051-02	TIME data type (including support of TIME literal) with fractional seconds precision of at least 0	O
F051-03	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6	O
F271	Compound character literals	X
F383	Set column not null clause	O
F391	Long identifiers	X
F392	Unicode escapes in identifiers	X
F393	Unicode escapes in literals	X
T023	Compound binary literals	X
T024	Spaces in binary literals	X
T101	Enhanced nullability determination	X
T351	Bracketed comments	X
T591	UNIQUE constraints of possibly null columns	O

Feature ID	설명	지원 여부
X041	Basic table mapping: null absent	X
X042	Basic table mapping: null as nil	X
X051	Advanced table mapping: null absent	X
X052	Advanced table mapping: null as nil	X
X170	XML null handling options	X
X400	Name and identifier mapping	X

11.2 Data Type

숫자 타입

숫자 타입은 크게 저장 방식과 소수부 표현 방식에 따라 구분할 수 있다.

- 저장 방식에 따른 분류
 - 십진 숫자 타입
 - 십진수 숫자를 100 진법 형태로 저장하는 방식
 - 타입 예: NUMBER, NUMERIC, FLOAT
 - 이진 숫자 타입
 - C 언어의 숫자를 그대로 저장하는 방식
 - 타입 예: NATIVE_INTEGER, NATIVE_DOUBLE
- 소수부 표현 방식에 따른 분류
 - 고정 소수점 타입 (exact numeric)
 - Scale이 고정되어 있는 숫자 타입
 - 타입 예: NUMERIC(precision, scale), NATIVE_INTEGER
 - 부동 소수점 타입 (approximate numeric)
 - Scale이 정해지지 않은 숫자 타입
 - 타입 예: FLOAT (precision), NATIVE_DOUBLE

십진 숫자 타입

유효 숫자의 정밀도를 나타내는 precision과 소수점의 범위를 나타내는 scale이 십진수 (decimal)를 기반으로 한다.

십진 고정 소수점 타입

십진 고정 소수점 타입은 SQL에서 정의한 타입이다.

표 11-2 십진 고정 소수점 타입

Type	Decimal precision	Decimal scale	참조
NUMBER(p)	p	0	NUMBER
NUMBER(p, s)	p	s	NUMBER
NUMERIC(p)	p	0	NUMERIC
NUMERIC(p, s)	p	s	NUMERIC
DECIMAL(p)	p	0	NUMERIC 타입의 alias
DECIMAL(p, s)	p	s	NUMERIC 타입의 alias
DEC(p)	p	0	NUMERIC 타입의 alias

Type	Decimal precision	Decimal scale	참조
DEC(p, s)	p	s	NUMERIC 타입의 alias
SMALLINT	5	0	NUMBER 타입의 alias
INTEGER	10	0	NUMBER 타입의 alias
BIGINT	19	0	NUMBER 타입의 alias
INT2	5	0	NUMBER 타입의 alias
INT4	10	0	NUMBER 타입의 alias
INT8	19	0	NUMBER 타입의 alias

십진 부동 소수점 타입

십진 부동 소수점 타입은 SQL에서 정의한 타입이다.

표 11-3 십진 부동 소수점 타입

Type	Decimal precision	Decimal scale	참조
NUMBER	38	N/A	NUMBER
FLOAT(p)	$\text{ceil}(\log_{10} 2^p)$	N/A	FLOAT
REAL	$\text{ceil}(\log_{10} 2^{24}) = 8$	N/A	FLOAT 타입의 alias
DOUBLE	$\text{ceil}(\log_{10} 2^{53}) = 16$	N/A	FLOAT 타입의 alias
FLOAT4	$\text{ceil}(\log_{10} 2^{24}) = 8$	N/A	FLOAT 타입의 alias
FLOAT8	$\text{ceil}(\log_{10} 2^{53}) = 16$	N/A	FLOAT 타입의 alias

이진 숫자 타입

유효 숫자의 정밀도를 나타내는 precision과 소수점의 범위를 나타내는 scale이 이진수 (binary)를 기반으로 한다.

이진 고정 소수점 타입

이진 고정 소수점 타입은 C 언어의 signed integer 계열 타입을 참조한다.

Sign bit를 표시하기 위해 1 bit를 사용하고 나머지 bit들은 precision을 표현하는데 사용하며 scale을 표현할 때는 bit를 사용하지 않는다.

표 11-4 이진 고정 소수점 타입

Type	Binary precision	Binary scale	참조
NATIVE_SMALLINT	15	0	NATIVE_SMALLINT
NATIVE_INTEGER	31	0	NATIVE_INTEGER
NATIVE_BIGINT	63	0	NATIVE_BIGINT

이진 부동 소수점 타입

이진 부동 소수점 타입은 C 언어의 float과 double 타입을 참조한다.

Sign bit를 표시하기 위해 1 bit를 사용하고 나머지 bit들은 precision과 scale을 표현하기 위해 사용한다.

표 11-5 이진 부동 소수점 타입

Type	Binary precision	Binary scale	참조
NATIVE_REAL	23	8	NATIVE_REAL
NATIVE_DOUBLE	52	11	NATIVE_DOUBLE

노트

이진 부동 소수점 타입에 대한 precision과 scale은 OS 및 compiler의 환경에 따라 변동될 수 있다.

CHARACTER STRING 타입

CHARACTER STRING 타입은 가변길이 문자열 여부와 문자열의 최대 길이에 따라 구분할 수 있다.

- 가변길이 문자열 여부에 따른 분류
 - 고정 길이 문자열
 - CHARACTER
 - 가변 길이 문자열
 - CHARACTER VARYING, CHARACTER LONG VARYING
- 문자열의 최대 길이에 따른 분류
 - 2000 [characters 또는 bytes]
 - CHARACTER
 - 4000 [characters 또는 bytes]
 - CHARACTER VARYING
 - 100 megabytes
 - CHARACTER LONG VARYING

BINARY STRING 타입

BINARY STRING 타입은 가변길이 이진 문자열 여부와 이진 문자열의 최대 길이에 따라 구분할 수 있다.

- 가변길이 이진 문자열 여부에 따른 분류
 - 고정 길이 이진 문자열

- **BINARY**
 - 가변 길이 이진 문자열
 - **BINARY VARYING, BINARY LONG VARYING**
- 이진 문자열의 최대 길이 따른 분류
 - 2000
 - **BINARY**
 - 4000
 - **BINARY VARYING**
 - 100 megabytes
 - **BINARY LONG VARYING**

날짜/ 시간 타입

날짜/ 시간 타입은 년, 월, 일, 시, 분, 초, time zone offset을 각 타입의 표현방식에 맞게 지정한다.
 날짜/ 시간 타입에는 **DATE, TIME, TIMESTAMP** 타입이 있다.

INTERVAL 타입

INTERVAL 타입은 시간 간격을 지정한다.

년, 월, 일, 시, 분, 초의 시간 간격을 각 타입의 표현방식에 맞게 지정한다.

INTERVAL 타입은 값의 표현 범위에 따라 YEAR TO MONTH 계열과 DAY TO SECOND 계열로 구분할 수 있다.

BOOLEAN 타입

BOOLEAN 타입은 TRUE, FALSE, UNKNOWN 값을 저장하며 UNKNOWN일 경우 null 값으로 표현한다. Condition으로 사용된 모든 expression들은 BOOLEAN 값을 반환하며, BOOLEAN type으로 정의된 column 또는 value는 condition으로 사용할 수 있다.

자세한 내용은 **BOOLEAN**을 참조한다.

ROWID 타입

데이터베이스에 저장된 모든 레코드는 각기 다른 위치정보를 가지고 있으며, 각각의 레코드를 구분하기 위해 레코드 식별자 (ROWID)를 사용한다.

ROWID 타입은 레코드 식별자 (ROWID)를 저장 관리하기 위한 타입이다.

ROWID pseudo column을 사용한 질의를 통해 레코드 식별자 (ROWID)를 얻을 수 있다.

자세한 내용은 **ROWID**를 참조한다.

타입간 비교

두 타입간의 비교는 하나의 대표 타입을 기준으로 수행된다. 비교 대상 타입이 대표 타입과 다를 경우 타입 변환을 통해 비교할 수도 있다.

표 11-6 타입간 비교를 위한 대표 타입에서는 두 타입간의 비교를 위한 대표 타입을 정의한다.

다음 표에서는 각 대표 타입별로 비교를 위해 대상 타입들을 변환하는 것에 대해 설명한다.

- 표 11-7 VC 에서의 비교를 위한 타입 변환
- 표 11-8 LC 에서의 비교를 위한 타입 변환
- 표 11-9 VB 에서의 비교를 위한 타입 변환
- 표 11-10 LB 에서의 비교를 위한 타입 변환
- 표 11-11 NB 에서의 비교를 위한 타입 변환
- 표 11-12 ND 에서의 비교를 위한 타입 변환
- 표 11-13 NU 에서의 비교를 위한 타입 변환
- 표 11-14 DA 에서의 비교를 위한 타입 변환
- 표 11-15 TI 에서의 비교를 위한 타입 변환
- 표 11-16 TZ 에서의 비교를 위한 타입 변환
- 표 11-17 TS 에서의 비교를 위한 타입 변환
- 표 11-18 SZ 에서의 비교를 위한 타입 변환
- 표 11-19 YM 에서의 비교를 위한 타입 변환
- 표 11-20 DS 에서의 비교를 위한 타입 변환
- 표 11-21 B0 에서의 비교를 위한 타입 변환
- 표 11-22 RI 에서의 비교를 위한 타입 변환

팁

다음은 타입간 비교를 위해 사용하는 약어이다.

- "VC" : CHARACTER VARYING
- "LC" : CHARACTER LONG VARYING
- "VB" : BINARY VARYING
- "LB" : BINARY LONG VARYING
- "NB" : NATIVE_BIGINT
- "ND" : NATIVE_DOUBLE

- "NU" : NUMBER
- "DA" : DATE
- "TI" : TIME
- "TZ" : TIME WITH TIMEZONE
- "TS" : TIMESTAMP
- "SZ" : TIMESTAMP WITH TIMEZONE
- "YM" : INTERVAL YEAR TO MONTH
- "DS" : INTERVAL DAY TO SECOND
- "BO" : BOOLEAN
- "RI" : ROWID

노트

타입간 비교 표에서 built-in data type은 축약된 단어를 double quote (")로 묶어서 표기한다.

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE
- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

표 11-6 타입간 비교를 위한 대표 타입

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBLE	NUMB	NUMERIC	FLOAT	DATE	TIME	TIMESTAMP	TIMESTAMP	TIMESTAMP	INTERVAL	INTERVAL	BLOB	ROWID
CHAR	VC	VC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
VARCHAR	VC	VC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
LONG VARCHAR	LC	LC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
BINARY				VB	VB	LB																	
VARBINARY				VB	VB	LB																	
LONG VARBINARY				LB	LB	LB																	
NATIVE_SMALLINT	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_INTEGER	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_BIGINT	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_REAL	NU	NU	NU				ND	ND	ND	ND	ND	NU	NU	NU									
NATIVE_DOUBLE	ND	ND	ND				ND	ND	ND	ND	ND	ND	ND	ND									
NUMB																				YM	DS		

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARCHAR	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC
ER	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU								
NUMERIC	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU					YM	DS		
FLOAT	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU					YM	DS		
DATE	DA	DA	DA												DA			TS	SZ			
TIME	TI	TI	TI													TI	TZ					
TIME_TZ	TZ	TZ	TZ													TZ	TZ					
TIMES TAMP	TS	TS	TS												TS			TS	SZ			
TIMES TAMP_TZ	SZ	SZ	SZ												SZ			SZ				
INTER VAL_Y M	YM	YM	YM				YM	YM	YM			YM	YM	YM					YM			
INTER VAL_DS	DS	DS	DS				DS	DS	DS			DS	DS	DS					DS			
BOOLEAN	B0	B0	B0																	B0		
ROWID	RI	RI	RI																		RI	

표 11-7 VC 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	CHAR (변환 없음)
VARCHAR	VARCHAR (변환 없음)

표 11-8 LC 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	CHAR (변환 없음)
VARCHAR	VARCHAR (변환 없음)
LONG VARCHAR	LONG VARCHAR (변환 없음)

표 11-9 VB 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
BINARY	BINARY (변환 없음)
VARBINARY	VARBINARY (변환 없음)

표 11-10 LB 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
BINARY	BINARY (변환 없음)
VARBINARY	VARBINARY (변환 없음)
LONG VARBINARY	LONG VARBINARY (변환 없음)

표 11-11 NB 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	NATIVE_BIGINT
VARCHAR	NATIVE_BIGINT
LONG VARCHAR	NATIVE_BIGINT
NATIVE_SMALLINT	NATIVE_SMALLINT (변환 없음)
NATIVE_INTEGER	NATIVE_INTEGER (변환 없음)
NATIVE_BIGINT	NATIVE_BIGINT (변환 없음)

표 11-12 ND 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	NATIVE_DOUBLE
VARCHAR	NATIVE_DOUBLE
LONG VARCHAR	NATIVE_DOUBLE
NATIVE_SMALLINT	NATIVE_SMALLINT (변환 없음)
NATIVE_INTEGER	NATIVE_INTEGER (변환 없음)
NATIVE_BIGINT	NATIVE_BIGINT (변환 없음)
NATIVE_REAL	NATIVE_REAL (변환 없음)
NATIVE_DOUBLE	NATIVE_DOUBLE (변환 없음)
NUMBER	NUMBER (변환 없음)
NUMERIC	NUMERIC (변환 없음)
FLOAT	FLOAT (변환 없음)

표 11-13 NU 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	NUMBER
VARCHAR	NUMBER
LONG VARCHAR	NUMBER
NATIVE_SMALLINT	NATIVE_SMALLINT (변환 없음)
NATIVE_INTEGER	NATIVE_INTEGER (변환 없음)
NATIVE_BIGINT	NATIVE_BIGINT (변환 없음)
NATIVE_REAL	NATIVE_REAL (변환 없음)
NATIVE_DOUBLE	NATIVE_DOUBLE (변환 없음)
NUMBER	NUMBER (변환 없음)
NUMERIC	NUMERIC (변환 없음)
FLOAT	FLOAT (변환 없음)

표 11-14 DA 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	DATE
VARCHAR	DATE
LONG VARCHAR	DATE
DATE	DATE (변환 없음)

표 11-15 TI 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	TIME
VARCHAR	TIME
LONG VARCHAR	TIME
TIME	TIME (변환 없음)

표 11-16 TZ 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	TIME_TZ
VARCHAR	TIME_TZ
LONG VARCHAR	TIME_TZ
TIME	TIME_TZ
TIME_TZ	TIME_TZ (변환 없음)

표 11-17 TS 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	TIMESTAMP
VARCHAR	TIMESTAMP

원본 타입	변환 타입
LONG VARCHAR	TIMESTAMP
DATE	DATE (변환 없음)
TIMESTAMP	TIMESTAMP (변환 없음)

표 11-18 SZ 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	TIMESTAMP_TZ
VARCHAR	TIMESTAMP_TZ
LONG VARCHAR	TIMESTAMP_TZ
DATE	TIMESTAMP_TZ
TIMESTAMP	TIMESTAMP_TZ
TIMESTAMP_TZ	TIMESTAMP_TZ (변환 없음)

표 11-19 YM 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	INTERVAL_YM
VARCHAR	INTERVAL_YM
LONG VARCHAR	INTERVAL_YM
NATIVE_SMALLINT	INTERVAL_YM
NATIVE_INTEGER	INTERVAL_YM
NATIVE_BIGINT	INTERVAL_YM
NUMBER	INTERVAL_YM
NUMERIC	INTERVAL_YM
FLOAT	INTERVAL_YM
INTERVAL_YM	INTERVAL_YM (변환 없음)

표 11-20 DS 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	INTERVAL_DS
VARCHAR	INTERVAL_DS
LONG VARCHAR	INTERVAL_DS
NATIVE_SMALLINT	INTERVAL_DS
NATIVE_INTEGER	INTERVAL_DS
NATIVE_BIGINT	INTERVAL_DS
NUMBER	INTERVAL_DS
NUMERIC	INTERVAL_DS
FLOAT	INTERVAL_DS
INTERVAL_DS	INTERVAL_DS (변환 없음)

표 11-21 B0 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	BOOLEAN
VARCHAR	BOOLEAN
LONG VARCHAR	BOOLEAN
BOOLEAN	BOOLEAN (변환 없음)

표 11-22 RI 에서의 비교를 위한 타입 변환

원본 타입	변환 타입
CHAR	ROWID
VARCHAR	ROWID
LONG VARCHAR	ROWID
ROWID	ROWID (변환 없음)

타입간 변환

타입간 변환은 내부 변환 (implicit type conversion)과 외부 변환 (explicit type conversion)으로 구분된다.

- 내부 변환 (implicit type conversion)은 select, insert, delete, update를 위한 expression, operator, function, condition 등에서 발생한다.
- 외부 변환 (explicit type conversion)은 CAST operator를 통해 이루어진다.

표 11-23 타입간 변환에서는 한 data type에서 다른 data type으로의 변환 가능 여부를 설명한다.

노트

타입간 변환 표에서 built-in data type은 축약된 문자열을 double quote (")로 묶어서 표기한다.

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE
- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

표 11-23 타입간 변환

Data type	C H A R	V A R C H A R	L O N G V A R C H A R	B I N A R Y	V A R B I N A R Y	L O N G V A R B I N A R Y	N A T I V E	N A T I V E	N A T I V E	N A T I V E	N A T I V E	N U M B E R	N U M E R I C	F L O A T	D A T E	T I M E	T I M E	T I M E S T A M P	T I M E S T A T M P	I N T E R V A L	I N T E R V A L	B O L E A N	R O W I D	
							S M A L L I N T	I N T E G E R	B I G I N T	R E A L	D O U B L E						T Z	T A M P	T M P	Y M	D S			
CHAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
VARC HAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
LONG VARC HAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
BINAR Y				O	O	O																		
VARBI NARY				O	O	O																		
LONG VARBI NARY				O	O	O																		
NATIV E_SMA LLINT	O	O	O				O	O	O	O	O	O	O	O							O	O		
NATIV E_INTE GER	O	O	O				O	O	O	O	O	O	O	O							O	O		
NATIV E_BIGI NT	O	O	O				O	O	O	O	O	O	O	O							O	O		
NATIV E_REA L	O	O	O				O	O	O	O	O	O	O	O										
NATIV E_DOUB LE	O	O	O				O	O	O	O	O	O	O	O										

Data type	C H A R	V A R C H A R	L O N G V A R C H A R	B I N A R Y	V A R B I N A R Y	L O N G V A R B I N A R Y	N A T I V E	N A T I V E	N A T I V E	N A T I V E	N A T I V E	N U M B E R	N U M E R I C	F L O A T	D A T E	T I M E	T I M E	T I M E S T A M P	T I M E S T A M P	I N T E R V A L	I N T E R V A L	B O O L E A N	R O W I D
NUMB ER	O	O	O				O	O	O	O	O	O	O	O						O	O		
NUME RIC	O	O	O				O	O	O	O	O	O	O	O						O	O		
FLOAT	O	O	O				O	O	O	O	O	O	O	O						O	O		
DATE	O	O	O												O			O	O				
TIME	O	O	O													O	O						
TIME_ TZ	O	O	O													O	O						
TIMES TAMP	O	O	O												O	O		O	O				
TIMES TAMP_ TZ	O	O	O												O	O	O	O	O				
INTERV AL_Y M	O	O	O				O	O	O			O	O	O						O			
INTERV AL_DS	O	O	O				O	O	O			O	O	O							O		
BOOLE AN	O	O	O																			O	
ROWI D	O	O	O																				O

- CHARACTER 타입으로의 변환

- 원본 타입이 CHARACTER 타입인 경우

- 원본 타입의 precision이 CHARACTER 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 precision이 CHARACTER 타입의 precision과 같은 경우 문자열이 갱신되지 않는다.

- 원본 타입의 precision이 CHARACTER 타입의 precision보다 작은 경우 precision 차이만큼 문자열 뒤에 공백 문자가 추가된다.
 - 원본 타입이 CHARACTER VARYING 타입 또는 CHARACTER LONG VARYING 타입인 경우
 - 원본 타입의 문자열 길이가 CHARACTER 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 문자열 길이가 CHARACTER 타입의 precision과 같은 경우 문자열이 갱신되지 않는다.
 - 원본 타입의 문자열 길이가 CHARACTER 타입의 precision보다 작은 경우 precision 차이만큼 문자열 뒤에 공백 문자가 추가된다.
 - 원본 타입이 숫자형 타입, 날짜/시간 타입, INTERVAL 타입, BOOLEAN 타입, ROWID 타입인 경우
 - 원본 타입의 변환 문자열 길이가 CHARACTER 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 변환 문자열 길이가 CHARACTER 타입의 precision과 같으면 문자열이 갱신되지 않는다.
 - 원본 타입의 변환 문자열 길이가 CHARACTER 타입의 precision보다 작은 경우 precision 차이만큼 문자열 뒤에 공백 문자가 추가된다.
- CHARACTER VARYING 타입으로의 변환
 - 원본 타입이 CHARACTER 타입인 경우
 - 원본 타입의 precision이 CHARACTER VARYING 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 precision이 CHARACTER VARYING 타입의 precision과 같거나 작은 경우 문자열이 갱신되지 않는다.
 - 원본 타입이 CHARACTER VARYING 타입, CHARACTER LONG VARYING 타입인 경우
 - 원본 타입의 문자열의 길이가 CHARACTER VARYING 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 문자열의 길이가 CHARACTER VARYING 타입의 precision과 같거나 작은 경우 문자열이 갱신되지 않는다.
 - 원본 타입이 숫자형 타입, 날짜/시간 타입, INTERVAL 타입, BOOLEAN 타입, ROWID 타입인 경우
 - 원본 타입의 변환 문자열 길이가 CHARACTER VARYING 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 변환 문자열 길이가 CHARACTER VARYING 타입의 precision과 같거나 작은 경우 문자열이 갱신되지 않는다.
- CHARACTER LONG VARYING 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - 원본 타입의 문자열이 갱신되지 않는다.
 - 원본 타입이 숫자형 타입, 날짜/시간 타입, INTERVAL 타입, BOOLEAN 타입, ROWID 타입인 경우
 - 원본 타입의 변환 문자열이 갱신되지 않는다.
- BINARY 타입으로의 변환
 - 원본 타입이 BINARY 타입인 경우
 - 원본 타입의 precision이 BINARY 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 precision이 BINARY 타입의 precision과 같은 경우 이진 문자열이 갱신되지 않는다.
 - 원본 타입의 precision이 BINARY 타입의 precision보다 작은 경우 precision 차이만큼 이진 문자열 뒤에 X'00' 문자가 추가된다.
 - 원본 타입이 BINARY VARYING 타입 또는 BINARY LONG VARYING 타입인 경우
 - 원본 타입의 이진 문자열의 길이가 BINARY 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 이진 문자열의 길이가 BINARY 타입의 precision과 같은 경우 이진 문자열이 갱신되지 않

는다.

- 원본 타입의 이진 문자열의 길이가 BINARY 타입의 precision보다 작은 경우 precision 차이만큼 이진 문자열 뒤에 '00' 문자가 추가된다.

- BINARY VARYING 타입으로의 변환

- 원본 타입이 BINARY 타입인 경우
 - 원본 타입의 precision이 BINARY VARYING 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 precision이 BINARY VARYING 타입의 precision과 같거나 작은 경우 이진 문자열이 갱신되지 않는다.
- 원본 타입이 BINARY VARYING 타입, BINARY LONG VARYING 타입인 경우
 - 원본 타입의 이진 문자열의 길이가 BINARY VARYING 타입의 precision보다 큰 경우 에러가 발생한다.
 - 원본 타입의 이진 문자열의 길이가 BINARY VARYING 타입의 precision과 같거나 작은 경우 이진 문자열이 갱신되지 않는다.

- BINARY LONG VARYING 타입으로의 변환

- 원본 타입이 BINARY STRING 타입인 경우 원본 타입의 이진 문자열이 갱신되지 않는다.

- 숫자형 타입으로의 변환

- 원본 타입이 CHARACTER STRING 타입인 경우
 - 숫자 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 precision 및 scale에 의해 overflow가 발생하거나 반올림 될 수 있다.
- 원본 타입이 숫자형 타입인 경우
 - 변환 타입에 정의된 precision 및 scale에 의해 overflow가 발생하거나 반올림 될 수 있다.
- 원본 타입이 INTERVAL 타입인 경우
 - 원본 타입이 single field (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)인 경우만 숫자형 타입으로 변환할 수 있다.
 - 변환 타입에 정의된 precision 및 scale에 의해 overflow가 발생하거나 반올림 될 수 있다.

- DATE 타입으로의 변환

- 원본 타입이 CHARACTER STRING 타입인 경우
 - DATE format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 값의 범위에 의해 overflow가 발생할 수 있다.
- 원본 타입이 DATE 타입, TIMESTAMP 타입인 경우
 - 에러가 발생하지 않는다.
- 원본 타입이 TIMESTAMP WITH TIME ZONE 타입인 경우
 - time zone offset을 고려하여 DATE 타입 값으로 변환한다.

- TIME 타입으로의 변환

- 원본 타입이 CHARACTER STRING 타입인 경우
 - TIME format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 값의 범위에 의해 반올림될 수 있다.
- 원본 타입이 TIME 타입, TIMESTAMP 타입인 경우
 - 에러가 발생하지 않는다.

- 원본 타입이 TIME WITH TIME ZONE 타입, TIMESTAMP WITH TIME ZONE 타입인 경우
 - time zone offset을 고려하여 TIME 타입 값으로 변환한다.
- TIME WITH TIME ZONE 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - TIME WITH TIME ZONE format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 값의 범위에 의해 반올림될 수 있다.
 - 원본 타입이 TIME 타입, TIME WITH TIME ZONE 타입, TIMESTAMP WITH TIME ZONE 타입인 경우
 - time zone offset을 고려하여 TIME WITH TIME ZONE 타입 값으로 변환한다.
- TIMESTAMP 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - TIMESTAMP format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 값의 범위에 의해 overflow가 발생하거나 반올림될 수 있다.
 - 원본 타입이 DATE 타입, TIMESTAMP 타입인 경우
 - 에러가 발생하지 않는다.
 - 원본 타입이 TIMESTAMP WITH TIME ZONE 타입인 경우
 - time zone offset을 고려하여 TIMESTAMP 타입 값으로 변환한다.
- TIMESTAMP WITH TIME ZONE 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - TIMESTAMP WITH TIME ZONE format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 변환 타입에 정의된 값의 범위에 의해 overflow가 발생하거나 반올림될 수 있다.
 - 원본 타입이 DATE 타입, TIMESTAMP 타입, TIMESTAMP WITH TIME ZONE 타입인 경우
 - time zone offset을 고려하여 TIMESTAMP WITH TIME ZONE 타입 값으로 변환한다.
- INTERVAL YEAR TO MONTH 계열 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - year-month interval literal 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 자세한 내용은 **Interval Literals**를 참조한다.
 - 변환 타입에 정의된 precision에 의해 overflow가 발생할 수 있다.
 - 원본 타입이 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, NUMBER, NUMERIC, FLOAT 타입인 경우
 - 변환 타입이 single field (YEAR, MONTH)이어야 한다.
 - 변환 타입에 정의된 precision에 의해 overflow가 발생할 수 있다.
 - 원본 타입이 INTERVAL YEAR TO MONTH 계열 타입인 경우
 - 변환 타입에 정의된 precision에 의해 overflow가 발생할 수 있다.
- INTERVAL DAY TO SECOND 계열 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - day-time interval literal 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 자세한 내용은 **Interval Literals**를 참조한다.

- 변환 타입에 정의된 precision에 의해 overflow가 발생하거나 반올림될 수 있다.
- 원본 타입이 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, NUMBER, NUMERIC, FLOAT 타입인 경우
 - 변환 타입이 single field (DAY, HOUR, MINUTE, SECOND)이어야 한다.
 - 변환 타입에 정의된 precision에 의해 overflow가 발생하거나 반올림이 발생할 수 있다.
- 원본 타입이 INTERVAL DAY TO SECOND 계열 타입인 경우
 - 변환 타입에 정의된 precision에 의해 overflow가 발생하거나 반올림이 발생할 수 있다.
- BOOLEAN 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - 대소문자 구분 없이 문자열이 "TRUE" 또는 "FALSE"인 경우에 변환할 수 있다. (앞뒤에 공백이 있는 경우에도 변환 가능)
 - 원본 타입이 BOOLEAN 타입인 경우
 - 에러가 발생하지 않는다.
- ROWID 타입으로의 변환
 - 원본 타입이 CHARACTER STRING 타입인 경우
 - ROWID format 형식에 맞지 않는 문자열이 오는 경우 에러가 발생한다.
 - 원본 타입이 ROWID 타입인 경우
 - 에러가 발생하지 않는다.

타입간 조합

타입간 조합이 필요한 경우

CASE 연산자와 집합 연산자 (**set operator**)는 다수의 expression을 연산 결과로 가진다.

다음 예와 같이 각 expression들이 서로 다른 타입을 가질 경우, 결과 타입을 결정해 주어야 한다.

- CASE 연산자

```
SELECT CASE expr WHEN expr THEN char(3)
          WHEN expr THEN char(5)
          ELSE char(1)
        END
FROM t1;
```

- 집합 연산자

```
SELECT float_column
FROM t1
UNION ALL
```



```
SELECT number_precision_column
FROM t2
UNION ALL
SELECT native_integer_column
FROM t3;
```

타입간 조합에 의한 결과 타입 결정에 적용되는 규칙이 있는데 다음은 그 규칙을 적용하는 예이다.

- 결과 타입 조합 규칙
 - 집합 연산자 (**set operator**)
 - CASE 연산자
 - CASE Expression
 - COALESCE
 - NULLIF

결과 타입 조합 규칙

각 expression의 data type은 조합 가능한 동일한 계열의 타입이어야 한다.

- 결과 타입 조합 규칙이 적용되는 예
 - 집합 연산자 (**set operator**)
 - CASE 연산자
 - CASE Expression
 - COALESCE
 - NULLIF

결과 타입 조합 규칙에 따른 결과 타입은 아래 표에서 설명한다.

팁

다음은 결과 타입 조합 규칙을 설명하기 위해 사용하는 약어이다.

- "VC" : CHARACTER VARYING
- "LC" : CHARACTER LONG VARYING
- "VB" : BINARY VARYING
- "LB" : BINARY LONG VARYING
- "NS" : NATIVE_SMALLINT
- "NI" : NATIVE_INTEGER
- "NB" : NATIVE_BIGINT
- "NR" : NATIVE_REAL
- "ND" : NATIVE_DOUBLE

- "FL" : FLOAT
- "NU" : NUMBER
- "DA" : DATE
- "TI" : TIME
- "TZ" : TIME WITH TIMEZONE
- "TS" : TIMESTAMP
- "SZ" : TIMESTAMP WITH TIMEZONE
- "YM" : INTERVAL YEAR TO MONTH
- "DS" : INTERVAL DAY TO SECOND
- "BO" : BOOLEAN
- "RI" : ROWID

노트

결과 타입 조합 규칙에 따른 결과 타입 표에서 built-in data type은 축약된 문자열을 double quote (")로 묶어서 표기한다.

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "BINARY": BINARY
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE
- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

표 11-24 결과 타입 조합 규칙에 따른 결과 타입

Data type	CHAR	VARCHAR	LONG	BINARY	VARBINARY	LONG	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	DATE	TIME	TIME	TIMESTAMP	TIMESTAMP	INTERVAL	INTERVAL	BOOLEAN	ROWID
ER							NU	NU	NU	ND	ND	NU	NU	NU									
NUMERIC							NU	NU	NU	ND	ND	NU	NU	NU									
FLOAT							NU	NU	NU	ND	ND	NU	NU	FL									
DATE															DA			TS	SZ				
TIME																TI	TZ						
TIME_TZ																TZ	TZ						
TIMESTAMP															TS			TS	SZ				
TIMESTAMP_TZ															SZ			SZ	SZ				
INTERVAL_YM																				YM			
INTERVAL_DS																					DS		
BOOLEAN																						BO	
ROWID																							RI

- 모두 CHAR 타입인 경우의 결과 타입 결정
 - Length가 다를 경우, VARCHAR
 - Length가 같을 경우, CHAR
- 모두 BINARY 타입인 경우의 결과 타입 결정
 - Length가 다를 경우, VARBINARY

- Length가 같을 경우, BINARY
- INTERVAL YEAR TO MONTH 형의 결과 타입 결정
 - YEAR와 MONTH가 혼재되어 있다면, INTERVAL YEAR TO MONTH
 - YEAR만 존재할 경우, INTERVAL YEAR
 - MONTH만 존재할 경우, INTERVAL MONTH
- INTERVAL DAY TO SECOND 형의 결과 타입 결정
 - Start field는 각 대상 expression 중 가장 큰 범위의 start field
 - End field는 각 대상 expression 중 가장 작은 범위의 end field
 - 예: {INTERVAL DAY, INTERVAL HOUR} → INTERVAL DAY TO HOUR
- 결과 타입의 precision, scale 결정
 - CHARACTER STRING 형
 - 대상 expression 중 최대 문자 길이
 - BINARY STRING 형
 - 대상 expression 중 최대 문자 길이
 - 숫자형
 - 해당 타입의 value를 최대 수용할 수 있는 범위로 지정한다.
 - TIME/TIMESTAMP 형
 - 대상 expression 중 최대 fractional seconds precision
 - INTERVAL YEAR TO MONTH
 - 대상 expression 중 최대 leading precision
 - INTERVAL DAY TO SECOND
 - Leading precision은 start field의 maximum leading precision
 - Fractional seconds precision은 maximum fractional seconds precision
- 자세한 내용은 **타입간 비교**를 참조한다.

호환성

Data type에 대한 SQL 표준 호환성은 다음과 같다.

표 11-25 SQL 표준 호환성

Feature ID	설명	지원 여부
B033	Untyped SQL-invoked function arguments	X
E011-01	INTEGER and SMALLINT data types	O
E011-02	REAL, DOUBLE PRECISION, and FLOAT data types	O
E011-03	DECIMAL and NUMERIC data types	X
E011-04	Arithmetic operators	O
E011-05	Numeric comparison	O
E011-06	Implicit casting among the numeric data types	O
E021-01	CHARACTER data type	O

Feature ID	설명	지원 여부
E021-02	CHARACTER VARYING data type	O
E021-03	Character literals	O
E021-04	CHARACTER_LENGTH function	O
E021-05	OCTET_LENGTH function	O
E021-06	SUBSTRING function	O
E021-07	Character concatenation	O
E021-08	UPPER and LOWER functions	O
E021-09	TRIM function	O
E021-10	Implicit casting among the fixed-length and variable-length character string types	O
E021-11	POSITION function	O
E021-12	Character comparison	O
E071-05	Columns combined via table operators need not have exactly the same data type	O
F051-01	DATE data type (including support of DATE literal)	O
F051-02	TIME data type (including support of TIME literal) with fractional seconds precision of at least 0	O
F051-03	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6	O
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	X
F051-05	Explicit CAST between datetime types and character string types	O
F054	TIMESTAMP in DATE type precedence list	X
F382	Alter column data type	O
F611	Indicator data types	X
F741	Referential MATCH types	X
J521	JDBC data types	X
J622	external Java types	X
S011-01	USER_DEFINED_TYPES view	X
S023	Basic structured types	X
S024	Enhanced structured types	X
S025	Final structured types	X
S026	Self-referencing structured types	X
S041	Basic reference types	X
S043	Enhanced reference types	X
S051	Create table of type	X
S071	SQL paths in function and type name resolution	X
S091-01	Arrays of built-in data types	X
S091-02	Arrays of distinct types	X
S092	Arrays of user-defined types	X
S094	Arrays of reference types	X

Feature ID	설명	지원 여부
S161	Subtype treatment	X
S162	Subtype treatment for references	X
S201-02	Array as result type of functions	X
S231	Structured type locators	X
S261	Specific type method	X
S272	Multisets of user-defined types	X
S274	Multisets of reference types	X
S281	Nested collection types	X
S401	Distinct types based on array types	X
S402	Distinct types based on distinct types	X
T021	BINARY and VARBINARY data types	O
T022	Advanced support for BINARY and VARBINARY data types	O
T031	BOOLEAN data type	O
T041	Basic LOB data type support	X
T042	Extended LOB data type support	X
T051	Row types	X
T071	BIGINT data type	O
T201	Comparable data types for referential constraints	X
T322	Declared data type attributes	X
X010	XML type	X
X011	Arrays of XML type	X
X012	XMultisets of XML type	X
X013	Distinct types of XML type	X
X014	Attributes of XML type	X
X015	Fields of XML type	X
X181	XML(DOCUMENT(UNTYPED)) type	X
X182	XML(DOCUMENT(ANY)) type	X
X190	XML(SEQUENCE) type	X
X191	XML(DOCUMENT(XMLSCHEMA)) type	X
X192	XML(CONTENT(XMLSCHEMA)) type	X
X231	XML(CONTENT(UNTYPED)) type	X
X232	XML(CONTENT(ANY)) type	X
X251	Persistent XML values of XML(DOCUMENT(UNTYPED)) type	X
X252	Persistent XML values of XML(DOCUMENT(ANY)) type	X
X253	Persistent XML values of XML(CONTENT(UNTYPED)) type	X
X254	Persistent XML values of XML(CONTENT(ANY)) type	X
X255	Persistent XML values of XML(SEQUENCE) type	X
X256	Persistent XML values of XML(DOCUMENT(XMLSCHEMA)) type	X
X257	Persistent XML values of XML(CONTENT(XMLSCHEMA)) type	X
X260	XML type: ELEMENT clause	X

Feature ID	설명	지원 여부
X261	XML type: NAMESPACE without ELEMENT clause	X
X263	XML type: NO NAMESPACE with ELEMENT clause	X
X264	XML type: schema location	X
X410	Alter column data type: XML type	X

11.3 Format 문자열

Format 문자열은 숫자 타입이나 날짜/ 시간 타입을 문자열로 변환하거나 문자열을 숫자 타입이나 날짜/ 시간 타입으로 변환하기 위한 형식을 정의한 문자열이다.

- 숫자 타입이나 날짜/시간 타입을 문자열로 변환할 때 다음과 같은 형식으로 문자열을 표현한다.
 - `TO_CHAR( number ), TO_CHAR( datetime )`
 - 숫자 타입: `TO_CHAR( 1234.56, 'S9,999.99' ) → '+1,234.56'`
 - 날짜/시간 타입: `TO_CHAR( SYSDATE, 'YYYY-MM-DD' ) → '2012-07-15'`
- 문자열을 숫자 타입이나 날짜/ 시간 타입으로 변환할 때 다음과 같은 형식으로 문자열을 표현한다.
 - `TO_NUMBER, TO_NATIVE_REAL, TO_NATIVE_DOUBLE`
 - `TO_DATE`
 - `TO_TIMESTAMP, TO_TIMESTAMP_WITH_TIME_ZONE`
 - `TO_TIME, TO_TIME_WITH_TIME_ZONE`
 - 숫자 타입: `TO_NUMBER( '+1,234.56', 'S9,999.99' ) → NUMBER TYPE`
 - 날짜/ 시간 타입: `TO_DATE( '2012-07-15', 'YYYY-MM-DD' ) → DATE TYPE`

Format 문자열은 다음과 같은 타입으로 구분된다.

- 숫자 타입: **Number Format 문자열**
- 날짜/ 시간 타입: **Datetime Format 문자열**

Number Format 문자열

Number format 문자열은 숫자 타입을 문자열로 변환하거나 문자열을 숫자 타입으로 변환하기 위한 형식을 정의한 문자열이다.

Number format 문자열은 `TO_CHAR( number ), TO_NUMBER, TO_NATIVE_REAL, TO_NATIVE_DOUBLE` 함수의 인자로 사용된다.

Number format 문자열에는 표현하고자 하는 형식에 따라 여러 개의 format element를 지정할 수 있다.

모든 number format element는 그 형식에 맞게 반올림하여 적용한다.

변환하고자 하는 value의 소수점 이전 digit 개수가 format 문자열에 지정된 숫자의 자리수보다 큰 경우, '#' 문자로 대체된다.

MI, S, PR의 부호를 표현하는 format element를 지정하지 않은 경우, 음수는 숫자 앞에 '-' 부호를 양수는 공백을 반환하는 방식으로 부호를 표현한다.

표 11-26 Number format elements

Format element	예제	설명
, (comma)	9,999	지정한 위치에 comma를 반환한다. Comma를 여러 개 지정할 수 있다. Format 문자열은 comma로 시작할 수 없고 소수점 (.) 이후에도 올 수 없다.
. (period)	99.99	지정한 위치에 소수점 (.)을 반환한다. Format 문자열 내에서 소수점은 한 번만 지정할 수 있다.
\$	\$9999	숫자 앞에 \$ 기호를 반환한다.
0	0999 9990	숫자 앞이나 끝에 0을 반환한다. 변환하고자 하는 value의 digit 개수가 format 문자열의 0 위치까지의 digit 개수보다 작은 경우, 차이나는 부분을 0으로 채워 반환한다.
9	9999	부호와 명시된 9의 개수에 맞게 공백과 숫자를 반환한다. 변환하고자 하는 value의 digit 개수가 명시된 9의 개수보다 작은 경우, 차이나는 부분을 공백으로 채워 반환한다. 음수인 경우 숫자 앞에 '-' 부호를 양수는 공백을 반환하는 방식으로 부호를 표현한다. Format 문자열 소수점 이전의 정수부로 표현되는 값이 0인 경우, 이 0은 공백으로 반환된다. 예: TO_CHAR(0.123, '9.999') → .123 예: TO_CHAR(0, '9') → 0
B	B9999	값이 0이 되는 경우, 공백을 반환한다.
EEEE	9.9EEEE	지수 표기법으로 반환한다. Format 문자열의 맨 마지막에 오거나, S, MI, PR 앞에 올 수 있다. Comma (,)와 함께 지정할 수 없다.
MI	9999MI	음수인 경우 숫자 끝에 '-' 를 양수인 경우 공백을 반환한다. Format 문자열의 마지막에만 지정할 수 있고, S, PR과 함께 지정할 수 없다.
PR	9999PR	음수인 경우 꺾쇠 괄호 안에 숫자를 반환한다. <숫자> 양수인 경우 숫자 앞뒤에 공백을 반환한다. Format 문자열의 마지막에만 지정할 수 있고, S, MI와 함께 지정할 수 없다.
RN rn	RN rn	로마 숫자를 대문자로 반환한다. 로마 숫자를 소문자로 반환한다. 1 ~ 3999 사이의 숫자에서만 반환된다. FM format element 이외의 다른 format element와는 함께 지정할 수 없다. TO_NUMBER 함수에는 사용할 수 없다.
S	S9999 9999S	양수인 경우 숫자 앞에 '+' 부호를 음수인 경우 '-' 부호를 반환한다. (S9999) 양수인 경우 숫자 끝에 '+' 부호를 음수인 경우 '-' 부호를 반환한다. (9999S) Format 문자열의 맨처음 또는 맨마지막에만 지정할 수 있다. MI, PR과 함께 지정할 수 없다.
V	999V99	V format element 뒤에 오는 9의 digit 개수가 n일 때, value에 10^n 을 곱한 값을 반환한다. 소수점 (.)과 함께 지정할 수 없다. TO_NUMBER 함수에는 사용할 수 없다.
		지정된 X digit 수에 맞게 공백과 16 진수를 반환한다. 정수값을 16 진수로 반환한다. (정수가 아닌 경우, 반올림하여 정수값을 만든다.) XXXX는 16 진수 대문자를 xxxx는 16 진수 소문자를 반환한다.

Format element	예제	설명
X	XXXX xxxx	변환된 16 진수의 digit 개수가 명시된 X의 개수보다 작은 경우, 차이나는 부분을 공백으로 채워 반환한다. 0과 양의 정수만 처리하고, 음수인 경우 '#' 문자로 대체된다. Format element 0 및 FM과만 함께 지정할 수 있으며, 다른 format element와는 함께 지정할 수 없다.
FM	FM	앞 뒤 공백을 제거하여 왼쪽 정렬되는 효과를 반환한다. 숫자 앞뒤에 붙는 공백을 제거한다. 9 format element에 의해 소수점 이하에 추가된 0을 제거한다.

다음은 number format 문자열을 사용하는 예이다.

```

TO_CHAR( 12345, '99,999' )           : ' 12,345'
TO_CHAR( 123456789, '999,999,999' ) : ' 123,456,789'
TO_CHAR( 12.345, '99.999' )           : ' 12.345'
TO_CHAR( 1234.56, '$9,999.99' )       : ' $1,234.56'
TO_CHAR( 123, '0999999' )             : ' 000123'
TO_CHAR( 0.2, '0.9' )                 : ' 0.2'
TO_CHAR( 123.45, '999999.99' )        : '    123.45'
TO_CHAR( -123.45, '999999.99' )       : '   -123.45'
TO_CHAR( 123.45, 'FM999999.99' )      : '123.45'
TO_CHAR( -123.45, 'FM999999.99' )     : '-123.45'
TO_CHAR( 12345.67, '999.99' )         : '#####'
TO_CHAR( 123.100567, '999.999' )      : ' 123.101'
TO_CHAR( 0.2, '90.99' )               : '  0.20'
TO_CHAR( 0.2, '99.99' )               : '   .20'
TO_CHAR( 0, '90.99' )                 : '  0.00'
TO_CHAR( 0, 'B90.99' )                : '      '
TO_CHAR( 123.45, '9.9EEEE' )          : '  1.2E+02'
TO_CHAR( 123.45, '999.99MI' )         : '123.45 '
TO_CHAR( -123.45, '999.99MI' )        : '123.45-'
TO_CHAR( 123.45, '999.99PR' )         : ' 123.45 '
TO_CHAR( -123.45, '999.99PR' )        : '<123.45>'
TO_CHAR( 123, 'RN' )                  : '      CXXIII'
TO_CHAR( 123, 'rn' )                  : '      cxxiii'
TO_CHAR( 123, 'FMRN' )                : 'CXXIII'
TO_CHAR( 4000, 'RN' )                 : '#####'
TO_CHAR( 123.45, 'S999.99' )          : '+123.45'
TO_CHAR( -123.45, 'S999.99' )         : '-123.45'
TO_CHAR( 123.45, '999.99S' )          : '123.45+'
TO_CHAR( -123.45, '999.99S' )         : '123.45-'

```

```

TO_CHAR( 123.45, '999V999' )      : ' 123450'
TO_CHAR( 123, 'XX' )              : ' 7B'
TO_CHAR( 123, 'xx' )              : ' 7b'
TO_CHAR( 45678, 'XXXXXX' )        : '      B26E'
TO_CHAR( 45678, 'FMXXXXXX' )      : 'B26E'
TO_CHAR( 123.45, '99,999.999999' ) : '      123.450000'
TO_CHAR( 123.45, 'FM99,999.999999' ) : '123.45'

```

Datetime Format 문자열

Datetime format 문자열은 날짜/ 시간 타입을 문자열로 변환하거나 문자열을 날짜/ 시간 타입으로 변환하기 위한 형식을 정의한 문자열이다.

Datetime format 문자열은 `TO_CHAR( datetime )`, `TO_DATE`, `TO_TIMESTAMP`, `TO_TIMESTAMP_WITH_TIME_ZONE`, `TO_TIME`, `TO_TIME_WITH_TIME_ZONE` 함수의 인자로 사용된다.

날짜/ 시간 타입의 경우, format 문자열을 지정하지 않으면 default 값으로 처리되는데, 각 타입의 default 값은 session property인 `NLS*_FORMAT`에 지정된 값이다.

- DATE: `NLS_DATE_FORMAT`
- TIMESTAMP: `NLS_TIMESTAMP_FORMAT`
- TIMESTAMP WITH TIME ZONE: `NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT`
- TIME: `NLS_TIME_FORMAT`
- TIME WITH TIME ZONE: `NLS_TIME_WITH_TIME_ZONE_FORMAT`

`NLS*_FORMAT` 값들은 `ALTER SESSION SET property_name` 구문으로 변경할 수 있다.

Datetime format 문자열에는 표현하고자 하는 형식에 따라 여러 개의 format element를 지정할 수 있다.

표 11-27 Datetime format elements

Format element	TO_* datetime 사용여부	설명
- / , . ; : "text" 특수문자	Y	지정된 위치에 format element의 문자를 반환한다.
AD	Y	서기

Format element	TO_* datetime 사용여부	설명
A.D.		
AM A.M.	Y	오전
BC B.C.	Y	기원전
CC	N	세기 네 자리 연도 중 마지막 두 자리가 01 ~ 99 이면 처음 두 자리에 1을 더한 값을 반환한다. (예: 2005년도일 경우 21) 네 자리 연도 중 마지막 두 자리가 00 이면, 처음 두 자리 값이 반환된다. (예: 2000년도일 경우 20)
D	Y	일주일 중 몇 번째 날인지를 반환한다. (1 ~ 7) 일요일 (1) ~ 토요일 (7)
DAY Day day	Y	요일을 반환한다. (예: SUNDAY) - DAY: 모두 대문자로 반환한다. - Day: 첫 번째 문자만 대문자로 반환하고 그 외는 소문자로 반환한다. - day: 모두 소문자로 반환한다.
DD	Y	달의 몇 번째 날인지를 반환한다. (1 ~ 31)
DDD	Y	연도의 몇 번째 날인지를 반환한다. (1 ~ 366)
DY Dy dy	Y	요일의 약어를 반환한다. (예: SUN) - DY: 모두 대문자로 반환한다. - Dy: 첫 번째 문자만 대문자로 반환하고 그 외는 소문자로 반환한다. - dy: 모두 소문자로 반환한다.
FF[1..6]	Y	Fractional seconds를 FF 이후에 지정한 숫자 (1 ~ 6)의 개수만큼 반환한다. 숫자를 지정하지 않은 경우, 기본값은 6이다. (FF는 FF6과 같다.) Fractional seconds의 digit 개수가 FF 이후에 지정한 숫자보다 많으면 버림처리된다. Fractional seconds의 digit 개수가 FF 이후에 지정한 숫자보다 적으면 지정된 숫자에 맞추어 0이 추가된다. DATE 타입에서는 사용할 수 없다.
HH HH12	Y	시간 (1 ~ 12)
HH24	Y	시간 (0 ~ 23)
IW	N	ISO 8601 표준에 정의된 calendar week (1 ~ 52주 또는 1 ~ 53주)로 지정된 연도의 첫 번째 목요일이 있는 주가 첫 번째 주가 된다. - Calendar week는 monday부터 시작한다. - First calendar week는 1월 4일을 포함한다. - First calendar week는 12월 29, 30, 31을 포함할 수 있다. - Last calendar week는 1월 1, 2, 3을 포함할 수 있다.
IYYY	N	ISO 8601 표준에 정의된 calendar week를 수용하는 4자리 연도이다.
IYY		ISO 8601 표준에 정의된 calendar week를 수용하는 3자리 연도이다.

Format element	TO_* datetime 사용여부	설명
IY I	N	ISO 8601 표준에 정의된 calendar week를 수용하는 2자리 연도이다. ISO 8601 표준에 정의된 calendar week를 수용하는 1자리 연도이다.
J	Y	BC 4714-11-24 일부터 경과한 날짜를 반환한다.
MI	Y	분 (0 ~ 59)
MM	Y	월 (01 ~ 12), 1월 (01) ~ 12월 (12)
MON Mon mon	Y	월의 약어 (예: JAN) 이다. - MON: 모두 대문자로 반환한다. - Mon: 첫 번째 문자만 대문자로 반환하고 그 외는 소문자로 반환한다. - mon: 모두 소문자로 반환한다.
MONTH Month month	Y	월의 이름 (예: JANUARY) 이다. - MONTH: 모두 대문자로 반환한다. - Month: 첫 번째 문자만 대문자로 반환하고 그 외는 소문자로 반환한다. - month: 모두 소문자로 반환한다.
PM P.M.	Y	오후
Q	N	연도의 분기 (1 ~ 4) 이다. 1월에서 3월 (1) ~ 10월에서 12월 (4) 이다.
RM Rm rm	Y	월을 로마숫자로 반환한다. (예: I) - RM: 모두 대문자로 반환한다. - Rm: 첫 번째 문자만 대문자로 반환하고 그 외는 소문자로 반환한다. - rm: 모두 소문자로 반환한다.
RR	Y	조정된 두 자리 연도이다. RR로 표현된 두 자리 연도를 네 자리 연도로 표현하는 방법은 다음과 같다. - RR로 표현된 두 자리 연도가 00 ~ 49인 경우 - 현재 연도의 마지막 두 자리가 00 ~ 50 이면, - 현재 연도의 처음 두 자리와 RR로 표현된 두 자리 연도 - 현재 연도의 마지막 두 자리가 51 ~ 99 이면, (현재 연도의 처음 두 자리 + 1)와 RR로 표현된 두 자리 연도 - RR로 표현된 두 자리 연도가 50 ~ 99 인 경우 - 현재 연도의 마지막 두 자리가 00 ~ 50 이면, (현재 연도의 처음 두 자리 - 1)와 RR로 표현된 두 자리 연도 - 현재 연도의 마지막 두 자리가 51 ~ 99 이면, - 현재 연도의 처음 두 자리와 RR로 표현된 두 자리 연도
RRRR	Y	조정된 네 자리 연도이다. 네 자리 또는 두 자리로 입력받을 수 있다. 두 자리로 입력받을 경우, RR과 동일하게 처리된다.
SS	Y	초 (0 ~ 59)
SSSSS	Y	지난 자정을 기준으로 경과된 초 (0 ~ 86399) 이다.
TZH	Y	Time zone hour 이다. DATE, TIMESTAMP, TIME 타입에서는 사용할 수 없고, TIMESTAMP WITH TIME

Format element	TO_* datetime 사용여부	설명
		ZONE, TIME WITH TIME ZONE 타입에서만 사용할 수 있다.
TZM	Y	Time zone minute 이다. DATE, TIMESTAMP, TIME 타입에서는 사용할 수 없고, TIMESTAMP WITH TIME ZONE, TIME WITH TIME ZONE 타입에서만 사용할 수 있다.
WW	N	연도의 몇 번째 주 (1 ~ 53) 인지를 반환한다. 첫 번째 주 1은 1월 1일부터 7일까지이다.
W	N	월의 몇 번째 주 (1 ~ 5)인지 반환한다. 첫 번째 주 1은 월의 1일부터 7일까지이다.
Y,YYY	Y	Comma가 포함된 Y,YYY 형식의 연도를 반환한다.
YYYY SYYYY	Y	네자리 연도이다. SYYYY는 BC 연도인 경우 '-' 부호를 반환한다.
YYY YY Y	Y	- YYY: 현재 연도의 마지막 3자리 연도이다. - YY: 현재 연도의 마지막 2자리 연도이다. - Y: 현재 연도의 마지막 1자리 연도이다.

다음은 datetime format 문자열을 사용하는 예이다.

* - / , . ; : "text" Special character

- TO_CHAR(TO_DATE('2012-07-15 03:30:30', 'YYYY-MM-DD HH12:MI:SS'),
'YYYY/MM/DD HH12:MI:SS')
==> '2012/07/15 03:30:30'
- TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'),
'YYYY"year" DDD"th day"')
==> '2012year 197th day'

* AD

- TO_CHAR(TO_DATE('2012-07-15 AD', 'YYYY-MM-DD AD'), 'YYYY AD')
==> '2012 AD'
- TO_CHAR(TO_DATE('0001-01-01 BC', 'YYYY-MM-DD AD'), 'YYYY AD')
==> '0001 BC'

* BC

- TO_CHAR(TO_DATE('0001-01-01 BC', 'YYYY-MM-DD BC'), 'YYYY BC')
==> '0001 BC'
- TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'YYYY BC')
==> '2012 AD'

* AM

- TO_CHAR(TO_DATE('2012-07-15 03:30:30 AM',
'YYYY-MM-DD HH12:MI:SS AM'),

```

        'HH12:MI:SS AM' )
    ==> '03:30:30 AM'
    • TO_CHAR( TO_DATE( '2012-07-15 21:30:30', 'YYYY-MM-DD HH24:MI:SS' ),
        'HH12:MI:SS AM' )
    ==> '09:30:30 PM'

```

* PM

```

    • TO_CHAR( TO_DATE( '2012-07-15 03:30:30',
        'YYYY-MM-DD HH24:MI:SS' ),
        'HH12:MI:SS PM' )
    ==> '03:30:30 AM'
    • TO_CHAR( TO_DATE( '2012-07-15 09:30:30 PM',
        'YYYY-MM-DD HH12:MI:SS PM' ),
        'HH12:MI:SS PM' )
    ==> '09:30:30 PM'

```

* CC

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'CC' )
    ==> '21'

```

* D

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'D' )
    ==> '1'

```

* DD

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DD' )
    ==> '15'

```

* DDD

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DDD' )
    ==> '197'

```

* DAY

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DAY' )
    ==> 'SUNDAY'
    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Day' )
    ==> 'Sunday'
    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'day' )
    ==> 'sunday'

```

* DY

```

    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DY' )
    ==> 'SUN'
    • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Dy' )
    ==> 'Sun'

```


- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'dy' )`
`==> 'sun'`

* FF[1 ... 6]

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 03:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'FF')`
`==> '123456'`
- `TO_CHAR(TO_TIMESTAMP('2012-07-15 03:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'FF5')`
`==> '12345'`
- `TO_CHAR(TO_TIMESTAMP('2012-07-15 03:30:45.9',
'YYYY-MM-DD HH24:MI:SS.FF1') ,
'FF6')`
`==> '9000000'`

* HH HH12 HH24

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 03:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'HH12')`
`==> '03'`
- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'HH12')`
`==> '11'`
- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'HH24')`
`==> '23'`

* IW

- `TO_CHAR( DATE'2016-01-01', 'IW' )`
`==> 53`
- `TO_CHAR( DATE'2014-12-30', 'IW' )`
`==> 01`

* IYYY

- `TO_CHAR( DATE'2016-01-01', 'IYYY' )`
`==> 2015`
- `TO_CHAR( DATE'2014-12-30', 'IYYY' )`

```
==> 2015
```

```
* IYY
```

- `TO_CHAR( DATE'2016-01-01', 'IYY' )`

```
==> 015
```

```
* IY
```

- `TO_CHAR( DATE'2016-01-01', 'IY' )`

```
==> 15
```

```
* I
```

- `TO_CHAR( DATE'2016-01-01', 'I' )`

```
==> 5
```

```
* J
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'J' )`

```
==> '2456124'
```

- `TO_CHAR( TO_DATE( '2456124', 'J' ), 'YYYY-MM-DD' )`

```
==> '2012-07-15'
```

```
* MI
```

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
 'YYYY-MM-DD HH24:MI:SS.FF6'),
 'MI')`

```
==> '30'
```

```
* MM
```

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
 'YYYY-MM-DD HH24:MI:SS.FF6'),
 'MM')`

```
==> '07'
```

```
* MON
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'MON' )`

```
==> 'JUL'
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Mon' )`

```
==> 'Jul'
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'mon' )`

```
==> 'jul'
```

```
* MONTH
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'MONTH' )`

```
==> 'JULY'
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Month' )`

```
==> 'July'
```

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'month' )`
`==> 'july'`

* Q

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Q' )`
`==> '3'`

* RM

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'RM' )`
`==> 'VII'`
- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Rm' )`
`==> 'vii'`
- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'rm' )`
`==> 'vii'`

* RR, RRRR (현재연도가 2014 인 경우)

- `TO_CHAR( TO_DATE( '49-07-15', 'RR-MM-DD' ), 'RRRR' )`
`==> '2049'`
- `TO_CHAR( TO_DATE( '49-07-15', 'RR-MM-DD' ), 'YYYY' )`
`==> '2049'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RR-MM-DD' ), 'RRRR' )`
`==> '1950'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RR-MM-DD' ), 'YYYY' )`
`==> '1950'`
- `TO_CHAR( TO_DATE( '50-07-15', 'YY-MM-DD' ), 'RRRR' )`
`==> '2050'`
- `TO_CHAR( TO_DATE( '49-07-15', 'RRRR-MM-DD' ), 'YYYY' )`
`==> '2049'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RRRR-MM-DD' ), 'YYYY' )`
`==> '1950'`

* RR, RRRR (현재연도가 2051 인 경우)

- `TO_CHAR( TO_DATE( '49-07-15', 'RR-MM-DD' ), 'RRRR' )`
`==> '2149'`
- `TO_CHAR( TO_DATE( '49-07-15', 'RR-MM-DD' ), 'YYYY' )`
`==> '2149'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RR-MM-DD' ), 'RRRR' )`
`==> '2050'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RR-MM-DD' ), 'YYYY' )`
`==> '2050'`
- `TO_CHAR( TO_DATE( '50-07-15', 'YY-MM-DD' ), 'RRRR' )`
`==> '2050'`

- `TO_CHAR( TO_DATE( '49-07-15', 'RRRR-MM-DD' ), 'YYYY' )`
`==> '2149'`
- `TO_CHAR( TO_DATE( '50-07-15', 'RRRR-MM-DD' ), 'YYYY' )`
`==> '2050'`

* SS

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'SS')`
`==> '45'`

* SSSSS

- `TO_CHAR(TO_TIMESTAMP('2012-07-15 23:30:45.123456',
'YYYY-MM-DD HH24:MI:SS.FF6'),
'SSSSS')`
`==> '84645'`

* TZH

- `TO_CHAR(TO_TIMESTAMP_TZ('2012-07-15 23:30:45.123456 +09:00',
'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'),
'TZH')`
`==> '+09'`

* TZM

- `TO_CHAR(TO_TIMESTAMP_TZ('2012-07-15 23:30:45.123456 +09:00',
'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'),
'TZM')`
`==> '00'`
- `TO_CHAR(TO_TIMESTAMP_TZ('2012-07-15 23:30:45.123456 +09:00',
'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'),
'TZH:TZM')`
`==> '+09:00'`

* WW

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'WW' )`
`==> '29'`

* W

- `TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'W' )`
`==> '3'`

* Y,YYY

- `TO_CHAR( TO_DATE( '2,012-07-15', 'Y,YYY-MM-DD' ), 'Y,YYY' )`
`==> '2,012'`

* YYYY

- TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'YYYY')
==> '2012'

* SYYYY

- TO_CHAR(TO_DATE('-0001-01-01', 'SYYYY-MM-DD'), 'SYYYY')
==> '-0001'

* YYY

- TO_CHAR(TO_DATE('012-07-15', 'YYY-MM-DD'), 'YYY')
==> '012'
- TO_CHAR(TO_DATE('012-07-15', 'YYY-MM-DD'), 'YYYY')
==> '2012' ((현재 연도 / 1000년)이 2 인 경우)

* YY

- TO_CHAR(TO_DATE('12-07-15', 'YY-MM-DD'), 'YY')
==> '12'
- TO_CHAR(TO_DATE('12-07-15', 'YY-MM-DD'), 'YYYY')
==> '2012' ((현재 연도 / 100년)이 20 인 경우)
==> '2112' ((현재 연도 / 100년)이 21 인 경우)

* Y

- TO_CHAR(TO_DATE('2-07-15', 'Y-MM-DD'), 'Y')
==> '2'
- TO_CHAR(TO_DATE('12-07-15', 'YY-MM-DD'), 'YYYY')
==> '2012' ((현재 연도 / 10년)이 201 인 경우)
==> '2052' ((현재 연도 / 10년)이 205 인 경우)

11.4 Expressions

Expression은 데이터 값을 얻기 위한 value, operator, function 들의 조합이다.

Expression이 사용될 수 있는 SQL 구문의 위치는 다음과 같다.

- SELECT target절
- SELECT의 GROUP BY절
- SELECT의 ORDER BY절
- SELECT의 WHERE절, HAVING절
- INSERT VALUES절
- UPDATE SET절
- INSERT, DELETE, UPDATE의 RETURN절

Expression 형태는 다음과 같이 다양하다.

- Simple expression
- Compound expression
- Boolean value expression
- Case expression
- Datetime expression
- Scalar subquery expression
- Sequence manipulation expression

Simple expression: Column, pseudo columns, literals, null value

Compound expression: 여러 개의 expression의 조합

자세한 내용은 다음을 참조한다.

- **Null Value**
- **Literals**
- **Pseudo Columns**
- **Operators**
- **Functions**

Boolean Value Expression

구문

```

<boolean value expression> ::=
    <boolean term>
    | <boolean value expression> OR <boolean term>
<boolean term> ::=

```

```

    <boolean factor>
    | <boolean term> AND <boolean factor>
<boolean factor> ::=
    [ NOT ] <boolean test>
<boolean test> ::=
    <boolean primary> [ IS [ NOT ] <truth value> ]
<truth value> ::=
    TRUE
    | FALSE
    | UNKNOWN
<boolean primary> ::=
    <column>
    <condition>
    | <boolean predicand>
<boolean predicand> ::=
    <parenthesized boolean value expression>
    | <nonparenthesized value expression primary>
<parenthesized boolean value expression> ::=
    <left paren> <boolean value expression> <right paren>

```

설명

<boolean value expression>은 boolean value를 기술한다. Boolean value를 갖는 <boolean primary>는 <column>과 <condition>, <boolean predicand>가 있다. <column>의 경우 BOOLEAN type으로 선언되어야 하고 CAST를 이용하여 boolean value를 반환할 수도 있다.

<boolean value expression>은 AND나 OR, NOT 등과 같은 논리 연산자와 함께 사용할 수 있으며 boolean value만의 연산자인 IS, IS NOT을 지원한다.

<boolean test>에 기술된 IS, IS NOT 연산자는 <boolean primary>에 기술된 boolean value가 <truth value>인 TRUE, FALSE, UNKNOWN 중 하나와 일치하는지 여부를 판단한다.

자세한 내용은 **Conditions**를 참조한다.

사용 예

```

gSQL> SELECT * FROM T1 WHERE CAST('TRUE' AS BOOLEAN);
I1
-----
TRUE
FALSE
null

```

```

3 rows selected.
gSQL> SELECT * FROM T1 WHERE I1;
I1
----
TRUE
1 row selected.
gSQL> SELECT * FROM T1 WHERE I1 IS TRUE;
I1
----
TRUE
1 row selected.
gSQL> SELECT * FROM T1 WHERE I1 IS NOT FALSE;
I1
----
TRUE
null
2 rows selected.
gSQL> SELECT * FROM T1 WHERE I1 IS UNKNOWN;
I1
----
null
1 row selected.

```

CASE Expression

구문

```

<case expression> ::=
    <simple case>
    | <searched case>
<simple case> ::=
    CASE expr WHEN comparison_expr THEN result
        [ WHEN comparison_expr THEN result ... ]
        [ ELSE result ]
    END
<searched case> ::=
    CASE WHEN condition THEN result
        [ WHEN condition THEN result ... ]
        [ ELSE result ]

```


END

설명

CASE 문에 기술된 순서대로 WHEN ... THEN 절을 평가한다.

비교 결과가 FALSE이면, TRUE가 나올 때까지 이후의 WHEN ... THEN 절을 평가한다.

비교 결과가 TRUE이면, result를 반환하고 이후는 평가하지 않는다.

- Simple case

CASE expr과 WHEN ... THEN 절의 comparison_expr을 equal 연산 (expr = comparison_expr)으로 평가한다.

- Searched case

WHEN ... THEN 절의 condition을 평가한다.

WHEN 절을 평가한 결과가 모두 FALSE인 경우에는 ELSE 절의 result를 반환한다.

ELSE 절이 생략된 경우, result로 NULL을 반환한다.

THEN 또는 ELSE 절의 result에 여러 type이 오는 경우, **결과 타입 조합 규칙**에 따라 result type을 결정한다.

자세한 내용은 다음을 참조한다.

- COALESCE
- NULLIF

사용 예

- Simple case

```
gSQL> SELECT I1,
           CASE I1 WHEN 1 THEN 'ONE'
                  WHEN 2 THEN 'TWO'
                  ELSE 'NUMBER'
           END AS CASE_RESULT1,
           CASE I1 WHEN 1 THEN 'ONE'
                  WHEN 2 THEN 'TWO'
           END AS CASE_RESULT2
FROM T1;
I1 CASE_RESULT1 CASE_RESULT2
-- -----
1 ONE          ONE
2 TWO          TWO
3 NUMBER       null
3 rows selected.
```

- Searched case

```

gSQL> SELECT I1,
           CASE WHEN I1 = 1 THEN 'ONE'
                WHEN I1 = 2 THEN 'TWO'
                ELSE 'NUMBER'
           END AS CASE_RESULT1,
           CASE WHEN I1 = 1 THEN 'ONE'
                WHEN I1 = 2 THEN 'TWO'
           END AS CASE_RESULT2
FROM T1;
I1 CASE_RESULT1 CASE_RESULT2
-- -----
1 ONE          ONE
2 TWO          TWO
3 NUMBER       null
3 rows selected.

```

CAST specification

구문

```
CAST( expression AS data_type )
```

설명

CAST는 expression의 데이터 타입을 지정된 data_type의 데이터 타입으로 변환한다.

사용 예

```

gSQL> SELECT CAST( '1-2' AS INTERVAL YEAR TO MONTH ) AS RESULT FROM DUAL;
RESULT
-----
+01-02
1 row selected.

```

Scalar Subquery Expression

Scalar subquery expression은 하나의 column을 갖는 row 하나를 결과값으로 반환하는 subquery 이다. Scalar subquery expression의 결과값은 subquery의 <select list>에 기술한 값이다.

만약 subquery가 0개의 row를 반환한다면 결과값은 NULL이며, 둘 이상의 row를 반환한다면 에러로 처리된다.

Scalar subquery expression은 expression을 기술하는 대부분의 위치에 기술할 수 있는데 subquery를 기술할 때는 반드시 괄호로 묶어야 한다. 함수 등의 인자로 사용되어 괄호 안에 scalar subquery expression이 기술되는 경우에도 함수의 괄호와 별도로 subquery를 위한 괄호로 묶어야 하며 그렇지 않은 경우 에러로 처리된다.

다음은 scalar subquery expression을 사용하는 예이다.

```
gSQL> select * from dual where dummy = (select * from dual);
DUMMY
-----
X
1 row selected.
gSQL> select sum(select 1 from dual) from dual;
ERR-42000(40000): syntax error
select sum(select 1 from dual) from dual
.....^    ^
Error at line 1
gSQL> select sum((select 1 from dual)) from dual;
SUM((SELECT 1 FROM DUAL))
-----
1
1 row selected.
```

호환성

Expression에 대한 SQL 표준 호환성은 다음과 같다.

표 11-28 SQL 표준 호환성

Feature ID	설명	지원 여부
E121-03	Value expressions in ORDER BY clause	O
F051-05	Basic date and time Explicit CAST between datetime types and character string types	O
F201	CAST function	O
F261-01	Simple CASE	O

Feature ID	설명	지원 여부
F261-02	Searched CASE	O
F261-03	NULLIF	O
F261-04	COALESCE	O
F263	Comma-separated predicates in simple CASE expression	X
F301	CORRESPONDING in query expressions	X
F385	Drop column generation expression clause	X
F561	Full value expressions	X
F846	Octet support in regular expression operators	X
F847	Nonconstant regular expressions	X
F850	Top-level <order by clause> in <query expression>	O
F855	Nested <order by clause> in <query expression>	O
F856	Nested <fetch first clause> in <query expression>	O
F857	Top-level <fetch first clause> in <query expression>	O
F861	Top-level <result offset clause> in <query expression>	O
F863	Nested <result offset clause> in <query expression>	O
S091-03	Arrays expressions	X
S111	ONLY in query expressions	X
T121	WITH (excluding RECURSIVE) in query expression	X
T581	Regular expression substring function	X

11.5 Pseudo Columns

Pseudo column은 function과 유사하지만, pseudo column을 수행할 때 row 단위로 매번 다른 값을 반환할 수 있다는 점에서 table의 column과도 유사하다.

표 11-29 지원되는 pseudo column

이름	설명	참고
CURRVAL	Sequence와 관련있는 pseudo column이다.	CURRVAL
NEXTVAL	Sequence와 관련있는 pseudo column이다.	NEXTVAL
ROWNUM	조건을 만족하는 row의 번호이다.	ROWNUM
ROWID	데이터베이스 내의 레코드 식별자를 반환한다.	ROWID Pseudo Column
CLUSTER_GROUP_ID	레코드가 저장된 group의 식별자를 반환한다.	CLUSTER_GROUP_ID Pseudo Column
CLUSTER_MEMBER_ID	레코드가 저장된 member의 식별자를 반환한다.	CLUSTER_MEMBER_ID Pseudo Column
CLUSTER_GROUP_NAME	레코드가 저장된 group의 이름을 반환한다.	CLUSTER_GROUP_NAME Pseudo Column
CLUSTER_MEMBER_NAME	레코드가 저장된 member의 이름을 반환한다.	CLUSTER_MEMBER_NAME Pseudo Column

ROWID Pseudo Column

ROWID pseudo column은 레코드 식별자로서 데이터베이스 내 각 레코드의 식별정보를 반환한다.

ROWID는 system에 따라 데이터베이스 내의 위치정보를 식별하기 위한 다음 정보들을 가진다.

Standalone system

- OBJECT_ID
- TABLESPACE_ID
- PAGE_ID
- PAGE 내 OFFSET

Cluster system

- GRID_BLOCK_SEQUENCE
- GRID_BLOCK_ID
- MEMBER_ID
- SHARD_ID

ROWID를 검색할 때 base 64 encoding으로 내부에 저장된 정보들을 A-Z, a-z, 0-9, +, / 의 value로 변환하여 출력한다.

ROWID 내에 저장된 데이터베이스 내 주소를 식별하기 위한 각각의 정보들은 ROWID-related functions 를 통해 얻을 수 있다.

레코드가 삭제되는 경우, 이 레코드의 주소는 새로 추가되는 레코드에 다시 할당될 수 있다.

ROWID pseudo column은 SELECT만 할 수 있고 INSERT, UPDATE, DELETE는 할 수 없다.

자세한 내용은 **ROWID, ROWID-related Functions**를 참조한다.

다음은 ROWID pseudo column을 검색하는 예이다.

```
gSQL> SELECT ROWID FROM T1;

                ROWID
-----
AAAAAAAAAFpEAACAAEAkAAA
AAAAAAAAAFpEAACAAEAkAAB
AAAAAAAAAFpEAACAAEAkAAC
AAAAAAAAAFpEAACAAEAkAAD
AAAAAAAAAFpEAACAAEAkAAE
5 rows selected.
```

CLUSTER_GROUP_ID Pseudo Column

CLUSTER_GROUP_ID pseudo column은 레코드가 저장된 server의 group 식별자를 반환한다.

CLUSTER_GROUP_ID pseudo column은 SELECT만 할 수 있고 INSERT, UPDATE, DELETE는 할 수 없다.

노트

Cluster system에서 유효한 정보이다.

다음은 CLUSTER_GROUP_ID pseudo column을 검색하는 예이다.

```
gSQL> SELECT T1.C1, T1.CLUSTER_GROUP_ID FROM T1;
C1 T1.CLUSTER_GROUP_ID
-- -----
A          1
B          2
C          3
3 rows selected.
```

CLUSTER_MEMBER_ID Pseudo Column

CLUSTER_MEMBER_ID pseudo column은 레코드가 저장된 server의 member 식별자를 반환한다.

CLUSTER_MEMBER_ID pseudo column은 SELECT만 할 수 있고 INSERT, UPDATE, DELETE는 할 수 없다.

노트

Cluster system에서 유효한 정보이다.

다음은 CLUSTER_MEMBER_ID pseudo column을 검색하는 예이다.

```
gSQL> SELECT T1.C1, T1.CLUSTER_MEMBER_ID FROM T1;
C1 T1.CLUSTER_MEMBER_ID
-- -----
A          1
B          3
C          5
3 rows selected.
```

CLUSTER_GROUP_NAME Pseudo Column

CLUSTER_GROUP_NAME pseudo column은 레코드가 저장된 server의 group 이름을 반환한다.

CLUSTER_GROUP_NAME pseudo column은 SELECT만 할 수 있고 INSERT, UPDATE, DELETE는 할 수 없다.

노트

Cluster system에서 유효한 정보이다.

다음은 CLUSTER_GROUP_NAME pseudo column을 검색하는 예이다.

```
gSQL> SELECT T1.C1, T1.CLUSTER_GROUP_NAME FROM T1;
C1 T1.CLUSTER_GROUP_NAME
-- -----
A  G1
```

```
B G2
C G3
3 rows selected.
```

CLUSTER_MEMBER_NAME Pseudo Column

CLUSTER_MEMBER_NAME pseudo column은 레코드가 저장된 server의 member 이름을 반환한다.

CLUSTER_MEMBER_NAME pseudo column은 SELECT만 할 수 있고 INSERT, UPDATE, DELETE는 할 수 없다.

노트

Cluster system에서 유효한 정보이다.

다음은 CLUSTER_MEMBER_NAME pseudo column을 검색하는 예이다.

```
gSQL> SELECT T1.C1, T1.CLUSTER_MEMBER_NAME FROM T1;
C1 T1.CLUSTER_MEMBER_NAME
-- -----
A G1N1
B G2N1
C G3N1
3 rows selected.
```

호환성

Pseudo column에 대한 SQL 표준 호환성은 다음과 같다.

표 11-30 SQL 표준 호환성

Feature ID	설명	지원 여부
T176	Sequence generator support	O
T177	Sequence generator support: simple restart option	O

11.6 Operators

Operator는 구문상에서 하나 이상의 특정 기호 또는 keyword로 표현되며, 하나 이상의 argument들을 가지고 기능을 수행한다.

Operator에는 다음과 같이 다양한 형태가 있다.

- Arithmetic operator
- Concatenation operator
- Set operator

Arithmetic Operator

구문

```

<arithmetic operator> ::=
    <value term>
    | <expression> + <value term>
    | <expression> - <value term>
<value term> ::=
    <value factor>
    | <value term> * <value factor>
    | <value term> / <value factor>
<value factor> ::=
    <expression>
    | + <expression>
    | - <expression>
  
```

설명

Arithmetic operator는 숫자형, 날짜/ 시간, INTERVAL 타입의 산술 연산을 수행한다.

Arithmetic operator의 우선순위는 다음과 같다.

1. + (POSITIVE), - (NEGATIVE)
2. * (MULTIPLICATION), / (DIVISION)
3. + (ADDITION), - (SUBTRACTION)

Concatenation Operator

구문

```
<concatenation operator> ::=
    <expression> || <expression>
```

설명

Concatenation operator는 CHARACTER STRING 타입이나 BINARY STRING 타입의 value 사이를 연결한 문자열을 반환한다.

자세한 내용은 || (CONCATENATE), CONCATENATE를 참조한다.

Set Operator

구문

```
<set operator> ::=
    <set operator term>
    | <subquery> UNION [ ALL | DISTINCT ] <set operator term>
    | <subquery> EXCEPT [ ALL | DISTINCT ] <set operator term>
    | <subquery> MINUS [ ALL | DISTINCT ] <set operator term>
<set operator term> ::=
    <subquery>
    | <subquery> INTERSECT [ ALL | DISTINCT ] <set operator term>
```

설명

set operator는 부질의 (subquery) 결과들에 대한 집합 (set) 연산을 수행한다.

INTERSECT ALL/ DISTINCT는 다른 set operator 보다 우선한다.

표 11-31 Set operators

Operator	설명
UNION ALL	Subquery 결과들에서 중복을 제거하지 않은 합집합이다.
UNION DISTINCT	Subquery 결과들에서 중복을 제거한 합집합이다.
EXCEPT ALL	Subquery 결과들에서 중복을 제거하지 않은 차집합이다.
EXCEPT DISTINCT	Subquery 결과들에서 중복을 제거한 차집합이다.

Operator	설명
MINUS ALL	EXCEPT ALL과 동일하다.
MINUS DISTINCT	EXCEPT DISTINCT와 동일하다.
INTERSECT ALL	Subquery 결과들에서 중복을 제거하지 않은 교집합이다.
INTERSECT DISTINCT	Subquery 결과들에서 중복을 제거한 교집합이다.

호환성

Operator에 대한 SQL 표준 호환성은 다음과 같다.

표 11-32 SQL 표준 호환성

Feature ID	설명	지원 여부
E011-04	Arithmetic operators	O
E021-07	Character concatenation	O
E071-01	UNION DISTINCT table operator	O
E071-02	UNION ALL table operator	O
E071-03	EXCEPT DISTINCT table operator	O
E071-05	Columns combined via table operators need not have exactly the same data type	O
E071-06	Table operators in subqueries	O
F041-08	All comparison operators are supported (rather than just =)	O
F302-01	INTERSECT DISTINCT table operator	O
F302-02	INTERSECT ALL table operator	O
F304	EXCEPT ALL table operator	O
F846	Octet support in regular expression operators	X
J571	NEW operator	X

11.7 Functions

Operator와 기능상으로는 유사하지만, function 이름 뒤에 괄호를 사용하여 argument들을 명시한다. Function은 0개 이상의 argument를 포함할 수 있다.

Function 형태는 다음과 같이 구분된다.

- Single row function
- Aggregate function

Single Row Function

Single row function은 table이나 view의 매 row 마다 각각 하나의 결과 row를 생성하는 function이다.

Single row function의 형태는 다음과 같이 구분된다.

- Numeric function
- Character string function returning character values
- Character string function returning number values
- Datetime function
- General comparison function
- Conversion function
- Conditional function
- NULL-related function
- ROWID-related function
- Encryption function
- System information function

Numeric Function

Numeric function은 숫자형 값을 입력받아 숫자형 결과를 반환하는 function이다.

Numeric function의 종류는 다음과 같다.

- ABS
- ACOS
- ASIN
- ATAN
- ATAN2
- BITAND
- BITNOT

- BITOR
- BITXOR
- CBRT
- CEIL
- COS
- COT
- DEGREES
- EXP
- FACTORIAL
- FLOOR
- LN
- LOG
- MOD
- PI
- POWER
- RADIANS
- RANDOM
- ROUND(number)
- SHARD_ID
- SHIFT_LEFT
- SHIFT_RIGHT
- SIGN
- SIN
- SQRT
- TAN
- TRUNC(number)
- WIDTH_BUCKET

Character String Functions Returning Character Values

Character string functions returning character values는 CHARACTER STRING형 값을 입력받아 CHARACTER STRING형 결과를 반환하는 function이다.

Character string functions returning character value의 종류는 다음과 같다.

- CHR
- CONCAT
- CONCATENATE
- INITCAP
- LOWER
- LPAD
- LTRIM

- OVERLAY
- REPEAT
- REPLACE
- REVERSE
- RPAD
- RTRIM
- SPLIT_PART
- SUBSTR
- SUBSTRB
- TRANSLATE
- TRIM
- UPPER

Character String Functions Returning Number Values

Character string functions returning number value는 CHARACTER STRING형 값을 입력받아 숫자형 결과를 반환하는 function이다.

Character string functions returning number value의 종류는 다음과 같다.

- ASCII
- BIT_LENGTH
- BYTE_LENGTH
- CHAR_LENGTH
- INSTR
- LENGTH
- LENGTHB
- OCTET_LENGTH
- POSITION

Datetime Functions

Datetime function은 DATE/ TIME/ TIMESTAMP/ INTERVAL 형 값을 입력받아 DATE/ TIME TIMESTAMP/ INTERVAL형 결과를 반환하는 function이다.

Datetime function의 종류는 다음과 같다.

- ADDDATE
- ADDTIME
- ADD_MONTHS
- DATEADD
- DATEDIFF

- DATE_ADD
- DATE_PART
- EXTRACT
- LAST_DAY
- MONTHS_BETWEEN

General Comparison Functions

General comparison function은 value 집합에 대한 최소값 또는 최대값을 구하는 function이다.

General comparison function의 종류는 다음과 같다.

- GREATEST
- LEAST

Conversion Functions

Conversion function은 특정 data type으로의 값을 설정하는 function이다.

Conversion function 종류는 다음과 같다.

- TO_CHAR(datetime)
- TO_CHAR(number)
- TO_DATE
- TO_NATIVE_DOUBLE
- TO_NATIVE_REAL
- TO_NUMBER
- TO_TIME
- TO_TIME_TZ
- TO_TIME_WITH_TIME_ZONE
- TO_TIMESTAMP
- TO_TIMESTAMP_TZ
- TO_TIMESTAMP_WITH_TIME_ZONE

Conditional Functions

Conditional function은 조건에 따라 특정값을 결과로 반환하는 function이다.

Conditional function의 종류는 다음과 같다.

- CASE2
- DECODE

NULL-related Functions

NULL-related function은 입력값이 NULL값인지 여부에 따라 특정값을 결과로 반환하는 function이다.

NULL-related function의 종류는 다음과 같다.

- COALESCE
- NULLIF
- NVL
- NVL2

ROWID-related Functions

ROWID-related function은 ROWID에 대한 정보를 얻기 위한 function이다.

ROWID-related function의 종류는 다음과 같다.

- Standalone에서 유효한 function
 - ROWID_OBJECT_ID
 - ROWID_TABLESPACE_ID
 - ROWID_PAGE_ID
 - ROWID_ROW_NUMBER
- Cluster에서 유효한 function
 - ROWID_GRID_BLOCK_ID
 - ROWID_GRID_BLOCK_SEQ
 - ROWID_MEMBER_ID
 - ROWID_SHARD_ID

Encryption Functions

Encryption function은 주어진 plain text를 특정 알고리즘으로 encrypt/ decrypt 하거나 hash한 결과값을 반환하는 function이다.

Encryption function에 대한 자세한 내용은 DIGEST를 참조한다.

System Information Functions

System information function은 session과 system에 대한 정보를 얻기 위한 function이다.

System information function의 종류는 다음과 같다.

- CLOCK_DATE
- CLOCK_LOCALTIME

- CLOCK_LOCALTIMESTAMP
- CURRENT_CATALOG
- CURRENT_DATE
- CURRENT_SCHEMA
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- LAST_IDENTITY_VALUE
- LOCALTIME
- LOCALTIMESTAMP
- LOGON_USER
- SESSION_ID
- SESSION_SERIAL
- SESSION_USER
- STATEMENT_DATE
- STATEMENT_LOCALTIME
- STATEMENT_LOCALTIMESTAMP
- STATEMENT_TIME
- STATEMENT_TIMESTAMP
- STATEMENT_VIEW_SCN
- SYSDATE
- SYSTIME
- SYSTIMESTAMP
- TRANSACTION_DATE
- TRANSACTION_LOCALTIME
- TRANSACTION_LOCALTIMESTAMP
- TRANSACTION_TIME
- TRANSACTION_TIMESTAMP
- USER_ID
- VERSION

Aggregate Function

Aggregate function은 여러 row에 대해 하나의 결과 row를 생성하는 function이다.

Aggregation function의 종류는 다음과 같다.

- COUNT
- COUNT(*)
- SUM

- AVG
- MIN
- MAX
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- VAR_POP
- VAR_SAMP
- VARIANCE

호환성

Function에 대한 SQL 표준 호환성은 다음과 같다.

표 11-33 SQL 표준 호환성

Feature ID	설명	지원 여부
B033	Untyped SQL-invoked function arguments	X
E021-04	CHARACTER_LENGTH function	O
E021-05	OCTET_LENGTH function	O
E021-06	SUBSTRING function	O
E021-08	UPPER and LOWER functions	O
E021-09	TRIM function	O
E021-11	POSITION function	O
E091-01	AVG	O
E091-02	COUNT	O
E091-03	MAX	O
E091-04	MIN	O
E091-05	SUM	O
E091-06	ALL quantifier	O
E091-07	DISTINCT quantifier	O
F131-03	Set functions supported in queries with grouped views	O
F201	CAST function	O
F441	Extended set function support	X
F442	Mixed column references in set functions	X
F801	Full set function	X
F842	OCCURRENCES_REGEX function	X
F843	POSITION_REGEX function	X
S071	SQL paths in function and type name resolution	X
S201-02	Array as result type of functions	X

Feature ID	설명	지원 여부
S211	User-defined cast functions	X
S241	Transform functions	X
T041-03	POSITION, LENGTH, LOWER, TRIM, UPPER, and SUBSTRING functions for LOB data types	X
T312	OVERLAY function	O
T321-01	User-defined functions with no overloading	X
T326	Table functions	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
T433	Multiargument GROUPING function	X
T441	ABS and MOD functions	O
T571	Array-returning external SQL-invoked functions	X
T572	Multiset-returning external SQL-invoked functions	X
T581	Regular expression substring function	X
T614	NTILE function	X
T615	LEAD and LAG functions	X
T616	Null treatment option for LEAD and LAG functions	X
T617	FIRST_VALUE and LAST_VALUE functions	X
T618	NTH_VALUE function	X
T619	Nested window functions	X
T621	Enhanced numeric functions	O

11.8 Conditions

Condition

TRUE, FALSE, UNKNOWN으로 평가되는 식이다.

Condition이 사용될 수 있는 SQL 구문의 위치는 다음과 같다.

- DELETE, UPDATE의 WHERE 절
- SELECT의 WHERE, HAVING 절
- 그 외 BOOLEAN TYPE이 위치할 수 있는 곳

Condition의 종류는 다음과 같다.

- Comparison condition
- Logical condition
- Null condition
- Compound condition
- Pattern-matching condition
- Between condition
- In condition
- Exists condition

표 11-34 Condition 우선순위

우선순위	Condition 종류
1	조건절에 쓰여진 연산자들
2	=, !=, <, >, <=, >=
3	IS [NOT] NULL, [NOT] BETWEEN, [NOT] IN, LIKE, EXISTS
4	NOT
5	AND
6	OR

Comparison Conditions

양쪽 조건을 비교하여 TRUE, FALSE, UNKNOWN 값의 boolean 타입을 반환한다.

표 11-35 Comparison condition

Condition	설명
=	두 식이 같은지 여부를 검사한다.
!=, <>	두 식이 서로 같지 않은지 여부를 검사한다.
>	두 식을 비교하여 큰 지 여부를 검사한다.
<	두 식을 비교하여 작은 지 여부를 검사한다.
>=	두 식을 비교하여 크거나 같은지 검사한다.
<=	두 식을 비교하여 작거나 같은지 검사한다.
ANY, SOME	왼쪽 expr이 오른쪽 expr_list (또는 subquery 결과) 중 하나 이상을 만족하는 조건이면 TRUE를 반환한다. 오른쪽 subquery의 결과가 없는 경우, FALSE를 반환한다.
ALL	왼쪽 expr이 오른쪽 expr_list (또는 subquery 결과)를 모두 만족하는 조건이면 TRUE를 반환한다. 오른쪽 subquery의 결과가 없는 경우, TRUE를 반환한다.

자세한 내용은 **타입간 비교**를 참조한다.

〈 Simple Comparison Conditions 〉

구문

```

<simple_comparison_condition> ::=
    <expr>          <comparison_operator> <expr>
  | <expr>          <comparison_operator> ( <subquery> )
  | ( <subquery> ) <comparison_operator> <expr>
  | ( <subquery> ) <comparison_operator> ( <subquery> )
  | ( <expr_list> ) <comparison_operator> ( <expr_list> )
  | ( <expr_list> ) <comparison_operator> ( <subquery> )
  | ( <subquery> ) <comparison_operator> ( <expr_list> )
  | ( <subquery> ) <comparison_operator> ( <subquery> )

<comparison_operator> ::=
    < = >
  | < != >
  | < < >
  | < > >
  | < <= >
  | < >= >

<expr_list> ::=
    <expr>
  | <expr>, ... , <expr>
  | ( <expr> )

```

| (<expr> , ... , <expr>)

자세한 내용은 **Scalar Subquery Expression**을 참조한다.

설명

comparison_operator 양쪽에 expr_list 또는 subquery가 오는 경우, 비교되는 expr의 개수 또는 subquery target의 개수는 동일해야 한다.

Subquery가 오는 경우, 결과 레코드는 한 건이어야 한다.

사용 예

표 11-36 Simple comparison condition의 예

Condition	결과
'abc' = 'abc'	TRUE
'abc' != 'abc'	FALSE
'abc' < 'abc'	FALSE
'abc' <= 'abc'	TRUE
'abc' > 'abc'	FALSE
'abc' >= 'abc'	TRUE
(1, 2, 3) = (1, 2, 3)	TRUE
(1, 2, 3) = (1, 2, 4)	FALSE
(1, 2, 3) != (4, 5, 6)	TRUE
(1, 2, 3) != (1, 2, 3)	FALSE
(1, 2, 3) < (1, 2, 4)	TRUE
(1, 2, 3) < (1, 2, 3)	FALSE
(1, 2, 3) <= (1, 2, 4)	TRUE
(1, 2, 3) <= (1, 2, 2)	FALSE
(1, 2, 3) > (1, 2, 2)	TRUE
(1, 2, 3) > (1, 2, 4)	FALSE
(1, 2, 3) >= (1, 2, 2)	TRUE
(1, 2, 3) >= (1, 2, 4)	FALSE

< Group Comparison Conditions >

구문

```

<group_comparison_condition> ::=
    <expr>          <comparison_operator> <quantifier> ( <expr_list> )
  | <expr>          <comparison_operator> <quantifier> ( <subquery> )
  | ( <expr_list> ) <comparison_operator> <quantifier> ( <expr_list_list> )

```

```

| ( <expr_list> ) <comparison_operator> <quantifier> ( <subquery> )
| ( <subquery> ) <comparison_operator> <quantifier> ( <expr_list> )
| ( <subquery> ) <comparison_operator> <quantifier> ( <expr_list_list> )
| ( <subquery> ) <comparison_operator> <quantifier> ( <subquery> )
<comparison_operator> ::=
    < = >
    | < != >
    | < < >
    | < > >
    | < <= >
    | < >= >
<quantifier> ::=
    ALL
    | ANY
    | SOME
<expr_list> ::=
    <expr>
    | <expr>, ... , <expr>
    | ( <expr> )
    | ( <expr> , ... , <expr> )
<expr_list_list> ::=
    <expr_list>
    | <expr_list>, ... , <expr_list>

```

자세한 내용은 **Scalar Subquery Expression**을 참조한다.

설명

comparison_operator 양쪽에 expr_list 또는 subquery가 오는 경우, 비교되는 expr의 개수 또는 subquery target의 개수는 동일해야 한다.

comparison_operator 왼쪽에 subquery가 오는 경우, 결과 레코드는 한 건이어야 한다.

comparison_operator 오른쪽에 subquery가 오는 경우, 결과 레코드는 여러 건일 수 있다.

사용 예

표 11-37 Group comparison condition의 예

Condition	결과
1 =any (1, 2, 3, 4, 5)	TRUE
1 =any (1, 2, null, 4, 5)	TRUE
1 =any (2, null, 4, 5)	NULL
1 =any (100, 2, 3, 4, 5)	FALSE
1 =all (1, +1, 1E+0)	TRUE

Condition	결과
1 =all (1, +1, 1E+0, null)	NULL
1 =all (1, 2, 3, 4, 5)	FALSE
(1, 2) =any ((0, 1), (1, 2), (3, 4))	TRUE
(1, 2) =any ((0, 1), (1, 2), (null, null))	TRUE
(1, 2) =any ((0, 1), (2, 3), (3, 4))	FALSE
(1, 2) =all ((1, 2), (+1, +2), (1E+0, 2E+0))	TRUE
(1, 2) =all ((1, 2), (+1, +2), (null, null))	NULL
(1, 2) =all ((0, 1), (2, 3), (3, 4))	FALSE
comparison_operator 오른쪽 subquery의 결과 레코드가 0인 경우	
('X') =any (select dummy from dual where dummy = 'Y')	FALSE
('X') =all (select dummy from dual where dummy = 'Y')	TRUE

Logical Conditions

Logical condition으로는 AND, OR, NOT이 있다.

AND

구문

⟨boolean value expression⟩ AND ⟨boolean value expression⟩

설명

표 11-38 AND boolean operator의 truth table

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR

구문

⟨boolean value expression⟩ OR ⟨boolean value expression⟩

설명

표 11-39 OR boolean operator의 truth table

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

NOT

구문

NOT <boolean value expression>

설명

표 11-40 NOT boolean operator의 truth table

expr	NOT
True	False
False	True
Unknown	Unknown

Null Condition

구문

<expr> IS [NOT] NULL

설명

expr의 결과가 NULL 값인지 여부를 검사한다.

표 11-41 Is Null 조건의 결과표

expr	IS NULL	IS NOT NULL
NULL	True	False
NOT NULL	False	True

Compound Condition

여러 조건들이 결합되어 만들어진 조건식이다.

```
compound_condition ::=
    ( condition )
  | NOT condition
  | condition < AND | OR > condition
```

Pattern-matching Conditions

LIKE Condition

구문

```
like_condition ::=
    string [NOT] LIKE pattern [ ESCAPE escape_character ]
```

설명

string이 지정된 pattern과 일치하는지 검사한다.

인자 string, pattern, escape_character에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입 또는 문자 타입으로 변환될 수 있는 타입이 올 수 있다.

string, pattern, escape_character가 NULL인 경우, NULL이 결과로써 반환된다.

escape_character가 생략되었을 경우, default 값은 없다.

escape_character가 명시된 경우, escape_character는 한 개 문자여야 한다.

pattern에 '_' 또는 '%'를 포함하지 않으면, equal 연산 (string = pattern)과 동일하게 처리된다.

pattern에 '_' 또는 '%'가 포함되면, string에서 다음과 같이 일치여부를 판단한다.

- '_': 임의의 한 개 문자와 대응한다.
- '%': 0개 이상의 문자를 가진 임의의 문자열과 대응한다.

pattern에 포함된 '_' 또는 '%'를 문자로 비교하고자 하는 경우, ESCAPE절을 사용한다.

escape_character를 지정하고, 지정된 escape_character를 pattern의 '_' 또는 '%' 앞에 기술한다.

사용 예

```
gSQL> SELECT 'hello%' LIKE 'h%o!%' ESCAPE '!' AS RESULT FROM DUAL;
RESULT
```

TRUE

- 'represent' LIKE 'represent' => TRUE
- 'represent' LIKE ' represent ' => FALSE
- 'represent' LIKE 'REPRESENT' => FALSE
- 'represent' LIKE 'r_pr_s_nt' => TRUE
- 'represent' LIKE 're%t' => TRUE
- 'represent' LIKE 'rep' => FALSE
- 'summer_vacation' LIKE 'summer_vacation' ESCAPE '\\' => TRUE
- NULL LIKE 'summer_vacation' ESCAPE '\\' => NULL
- 'summer_vacation' LIKE NULL ESCAPE '\\' => NULL
- 'summer_vacation' LIKE 'summer_vacation' ESCAPE NULL => NULL

BETWEEN Condition

구문

```
<between condition> ::=
    <expr1> [ NOT ] BETWEEN [ ASYMMETRIC | SYMMETRIC ] <expr2> AND <expr3>
```

설명

expr1이 expr2와 expr3 범위 내의 조건인지 검사한다.

ASYMMETRIC이나 SYMMETRIC이 생략된 경우, default는 ASYMMETRIC 이다.

expr1, expr2, expr3의 data type이 다른 경우, conversion이 수행된다.

자세한 내용은 **타입간 비교**, **타입간 변환**을 참조한다.

표 11-42 Between 구문 동치

A	B
X BETWEEN ASYMMETRIC Y AND Z	X BETWEEN Y AND Z
X BETWEEN Y AND Z	X >= Y AND X <= Z
X NOT BETWEEN Y AND Z	NOT(X BETWEEN Y AND Z)
X BETWEEN SYMMETRIC Y AND Z	((X BETWEEN Y AND Z) OR (X BETWEEN Z AND Y))
X NOT BETWEEN SYMMETRIC Y AND Z	NOT(X BETWEEN SYMMETRIC Y AND Z)

사용 예

표 11-43 Between 구문의 예

Condition		결과
BETWEEN [ASYMMETRIC]	3 BETWEEN 1 AND 5	TRUE
	NULL BETWEEN 1 AND 5	NULL
	3 BETWEEN NULL AND 5	
	3 BETWEEN 1 AND NULL	
	3 BETWEEN 5 AND 1	FALSE
BETWEEN SYMMETRIC	3 BETWEEN SYMMETRIC 1 AND 5	TRUE
	NULL BETWEEN SYMMETRIC 1 AND 5	NULL
	3 BETWEEN SYMMETRIC NULL AND 5	
	3 BETWEEN SYMMETRIC 1 AND NULL	
	3 BETWEEN SYMMETRIC 5 AND 1	TRUE

IN Condition

구문

```

<in_condition> ::=
    <expr>          [NOT] IN ( <expr_list> )
  | <expr>          [NOT] IN ( <subquery> )
  | ( <expr_list> ) [NOT] IN ( <expr_list_list> )
  | ( <expr_list> ) [NOT] IN ( <subquery> )
  | ( <subquery> ) [NOT] IN ( <expr_list> )
  | ( <subquery> ) [NOT] IN ( <expr_list_list> )
  | ( <subquery> ) [NOT] IN ( <subquery> )

<expr_list> ::=
    <expr>
  | <expr>, ... , <expr>
  | ( <expr> )
  | ( <expr> , ... , <expr> )

<expr_list_list> ::=
    <expr_list>
  | <expr_list>, ... , <expr_list>

```

설명

IN condition은 =ANY와 동일한 결과를 반환한다.

NOT IN condition은 !=ALL과 동일한 결과를 반환한다.

자세한 내용은 **Comparison Conditions**를 참조한다.

사용 예

표 11-44 IN condition의 예

Condition	결과
1 IN (1, 2, 3, 4, 5)	TRUE
1 IN (1, 2, null, 4, 5)	TRUE
1 IN (2, null, 4, 5)	NULL
1 IN (100, 2, 3, 4, 5)	FALSE
NULL IN (1, 2, 3)	NULL
1 NOT IN (2, 3, 4, 5)	TRUE
1 NOT IN (2, null, 4, 5)	NULL
1 NOT IN (1, 2, null, 4, 5)	FALSE
1 NOT IN (100, 2, 3, 4, 5)	TRUE
NULL NOT IN (1, 2, 3)	NULL

EXISTS Condition

구문

```
exists_conditions ::=
    EXISTS ( subquery )
```

설명

Subquery의 결과 레코드 존재 여부를 검사한다.

Subquery의 결과 레코드가 존재하면 TRUE를 반환하고, 결과 레코드가 존재하지 않으면 FALSE를 반환한다.

사용 예

```
gSQL> SELECT * FROM DUAL WHERE EXISTS ( SELECT * FROM DUAL );
DUMMY
```

```

-----
X
1 row selected.
gSQL> SELECT * FROM DUAL
      WHERE EXISTS ( SELECT * FROM DUAL WHERE DUMMY = 'Y' );
no rows selected.

```

호환성

Condition에 대한 SQL 표준 호환성은 다음과 같다.

표 11-45 SQL 표준 호환성

Feature ID	설명	지원 여부
E061-01	Comparison predicate	O
E061-02	BETWEEN predicate	O
E061-03	IN predicate with list of values	O
E061-04	LIKE predicate	O
E061-05	LIKE predicate: ESCAPE clause	O
E061-06	NULL predicate	O
E061-07	Quantified comparison predicate	O
E061-08	EXISTS predicate	O
E061-09	Subqueries in comparison predicate	O
E061-11	Subqueries in IN predicate	O
E061-12	Subqueries in quantified comparison predicate	O
E061-13	Correlated subqueries	O
E061-14	Search condition	O
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	X
F053	OVERLAPS predicate	X
F263	Comma-separated predicates in simple CASE expression	X
F291	UNIQUE predicate	X
F481	Expanded NULL predicate	O
F841	LIKE_REGEX predicate	X
P008	Comma-separated predicates in a CASE statement Extended CASE	X
S151	Type predicate	X
T141	SIMILAR predicate	X
T151	DISTINCT predicate	X
T152	DISTINCT predicate with negation	X
T461	Symmetric BETWEEN predicate	O
T501	Enhanced EXISTS predicate	X

Feature ID	설명	지원 여부
T631	IN predicate with one list element	X
X090	XML document predicate	X
X091	XML content predicate	X
X141	IS VALID predicate: data-driven case	X
X142	IS VALID predicate: ACCORDING TO clause	X
X143	IS VALID predicate: ELEMENT clause	X
X144	IS VALID predicate: schema location	X
X145	IS VALID predicate outside check constraints	X
X151	IS VALID predicate with DOCUMENT option	X
X152	IS VALID predicate with CONTENT option	X
X153	IS VALID predicate with SEQUENCE option	X
X155	IS VALID predicate: NAMESPACE without ELEMENT clause	X
X157	IS VALID predicate: NO NAMESPACE with ELEMENT clause	X

11.9 Built-in Data Type References

Aliases of Built-in Data Types

- **BIGINT**
 - **NUMBER(19,0)**와 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **BINARY**
 - 자세한 내용은 **BINARY**를 참조한다.
- **BINARY VARYING**
 - 자세한 내용은 **BINARY VARYING**을 참조한다.
- **BINARY LONG VARYING**
 - 자세한 내용은 **BINARY LONG VARYING**을 참조한다.
- **BOOLEAN**
 - 자세한 내용은 **BOOLEAN**을 참조한다.
- **CHAR**
 - **CHARACTER**와 동일하다.
 - 자세한 내용은 **CHARACTER**를 참조한다.
- **CHARACTER**
 - 자세한 내용은 **CHARACTER**를 참조한다.
- **CHARACTER VARYING**
 - 자세한 내용은 **CHARACTER VARYING**을 참조한다.
- **CHARACTER LONG VARYING**
 - 자세한 내용은 **CHARACTER LONG VARYING**을 참조한다.
- **DATE**
 - 자세한 내용은 **DATE**를 참조한다.
- **DEC**
 - **NUMERIC**과 동일하다.
 - 자세한 내용은 **NUMERIC**을 참조한다.
- **DECIMAL**
 - **NUMERIC**과 동일하다.
 - 자세한 내용은 **NUMERIC**을 참조한다.
- **DOUBLE**
 - **FLOAT(53)**과 동일하다.
 - 자세한 내용은 **FLOAT**을 참조한다.
- **DOUBLE PRECISION**
 - **FLOAT(53)**과 동일하다.
 - 자세한 내용은 **FLOAT**을 참조한다.
- **FLOAT**

- 자세한 내용은 **FLOAT**을 참조한다.
- **FLOAT4**
 - **FLOAT(24)**과 동일하다.
 - 자세한 내용은 **FLOAT**을 참조한다.
- **FLOAT8**
 - **FLOAT(53)**과 동일하다.
 - 자세한 내용은 **FLOAT**을 참조한다.
- **INT**
 - **NUMBER(10,0)**과 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **INT2**
 - **NUMBER(5,0)**과 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **INT4**
 - **NUMBER(10,0)**와 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **INT8**
 - **NUMBER(19,0)**와 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **INTEGER**
 - **NUMBER(10,0)**와 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **INTERVAL**
 - 자세한 내용은 **INTERVAL**을 참조한다.
- **LONG BINARY VARYING**
 - **BINARY LONG VARYING**과 동일하다.
 - 자세한 내용은 **BINARY LONG VARYING**을 참조한다.
- **LONG CHAR VARYING**
 - **CHARACTER LONG VARYING**과 동일하다.
 - 자세한 내용은 **CHARACTER LONG VARYING**을 참조한다.
- **LONG CHARACTER VARYING**
 - **CHARACTER LONG VARYING**과 동일하다.
 - 자세한 내용은 **CHARACTER LONG VARYING**을 참조한다.
- **LONG VARCHAR**
 - **CHARACTER LONG VARYING**과 동일하다.
 - 자세한 내용은 **CHARACTER LONG VARYING**을 참조한다.
- **NATIVE_BIGINT**
 - 자세한 내용은 **NATIVE_BIGINT**를 참조한다.
- **NATIVE_DOUBLE**
 - 자세한 내용은 **NATIVE_DOUBLE**을 참조한다.
- **NATIVE_INTEGER**

- 자세한 내용은 **NATIVE_INTEGER**를 참조한다.
- **NATIVE_REAL**
 - 자세한 내용은 **NATIVE_REAL**을 참조한다.
- **NATIVE_SMALLINT**
 - 자세한 내용은 **NATIVE_SMALLINT**를 참조한다.
- **NUMBER**
 - 자세한 내용은 **NUMBER**를 참조한다.
- **NUMERIC**
 - 자세한 내용은 **NUMERIC**을 참조한다.
- **ROWID**
 - 자세한 내용은 **ROWID**를 참조한다.
- **SMALLINT**
 - **NUMBER(5,0)**와 동일하다.
 - 자세한 내용은 **NUMBER**를 참조한다.
- **TIME**
 - 자세한 내용은 **TIME**을 참조한다.
- **TIMESTAMP**
 - 자세한 내용은 **TIMESTAMP**를 참조한다.
- **VARBINARY**
 - **BINARY VARYING**과 동일하다.
 - 자세한 내용은 **BINARY VARYING**을 참조한다.
- **VARCHAR**
 - **CHARACTER VARYING**과 동일하다.
 - 자세한 내용은 **CHARACTER VARYING**을 참조한다.
- **VARCHAR2**
 - **CHARACTER VARYING**과 동일하다.
 - 자세한 내용은 **CHARACTER VARYING**을 참조한다.

BINARY

구문

BINARY [(length)]

구문 규칙 및 파라미터

- length: Binary string의 길이이다.
 - 범위: 1 ~ 2000
 - 기본값: 1

설명

고정 길이의 binary string을 저장한다.

명시된 length 보다 저장되는 binary string의 길이가 짧은 경우, 나머지 부분은 X'00'로 저장된다.

- 저장 공간의 크기: length 값에 해당하는 bytes

참조

- BINARY VARYING
- BINARY LONG VARYING

BINARY VARYING

구문

BINARY VARYING (length)

구문 규칙 및 파라미터

- length: Binary string의 최대 길이이다.
 - 범위: 1 ~ 4000

설명

가변 길이의 binary string을 저장한다.

- 저장 공간의 크기: 저장할 binary string의 byte 크기
- Alias name: VARBINARY

참조

- BINARY
- BINARY LONG VARYING

BINARY LONG VARYING

구문

BINARY LONG VARYING

설명

긴 가변 binary string 값을 저장한다.

- 최대 저장 크기: 100 megabytes
- 저장 공간의 크기: 저장할 binary string의 byte 크기
- Alias names: LONG BINARY VARYING, LONG VARBINARY

Key의 column으로 사용할 수 없어 다음과 같은 제약이 있다.

- 인덱스의 key column으로 사용할 수 없다.
- ORDER BY 절의 expression으로 사용할 수 없다.
- GROUP BY 절의 expression으로 사용할 수 없다.
- DISTINCT 절의 expression으로 사용할 수 없다.
- UNION, INTERSECT, EXCEPT 절의 expression 으로 사용할 수 없다.

참조

- BINARY
- BINARY VARYING

BOOLEAN

구문

BOOLEAN

설명

TRUE 또는 FALSE 값을 저장한다.

- 저장 공간의 크기: 1 byte

CHARACTER

구문

CHARACTER [(length [CHARACTERS | OCTETS | CHAR | BYTE])]

구문 규칙 및 파라미터

- length: 문자열의 길이이다.
 - 범위: 1 ~ 2000
 - 기본값: 1
- [CHARACTERS | OCTETS | CHAR | BYTE]: length의 단위이다.
 - CHARACTERS
 - 문자의 개수이다.
 - CHAR는 CHARACTERS와 동일하다.
 - OCTETS
 - 바이트의 개수이다.
 - BYTE는 OCTETS와 동일하다.
 - 생략할 경우, database를 생성할 때 설정한 **CHAR_LENGTH_UNITS** 프로퍼티 값을 따른다.

설명

고정 길이의 문자열을 저장한다.

명시된 length 보다 저장되는 string의 길이가 짧은 경우, 나머지 부분에는 공백이 저장된다.

- 저장 공간의 크기: length 값에 해당하는 bytes 크기
- Alias name: CHAR

참조

- CHARACTER VARYING
- CHARACTER LONG VARYING

CHARACTER VARYING

구문

CHARACTER VARYING (length [CHARACTERS | OCTETS | CHAR | BYTE])

구문 규칙 및 파라미터

- length: 문자열의 길이이다.
 - Range: 1 ~ 4000
- [CHARACTERS | OCTETS | CHAR | BYTE]: length의 단위이다.
 - CHARACTERS
 - 문자의 개수이다.
 - CHAR는 CHARACTERS 와 동일하다.
 - OCTETS
 - 바이트의 개수이다.
 - BYTE는 OCTETS와 동일하다.
 - 생략할 경우, database를 생성할 때 설정한 **CHAR_LENGTH_UNITS** 프로퍼티 값을 따른다.

설명

가변 길이의 문자열을 저장한다.

- 저장 공간의 크기 : 저장할 문자열의 byte 크기
- Alias names: VARCHAR, VARCHAR2

참조

- CHARACTER
- CHARACTER LONG VARYING

CHARACTER LONG VARYING

구문

CHARACTER LONG VARYING

설명

긴 가변 문자열 값을 저장한다.

- 최대 저장 크기: 100 megabytes
- 저장 공간의 크기: 저장할 문자열의 byte 크기
- Alias names: LONG CHARACTER VARYING, LONG CHAR VARYING, LONG VARCHAR

Key의 column으로 사용할 수 없어 다음과 같은 제약이 있다.

- 인덱스의 key column으로 사용할 수 없다.
- ORDER BY 절의 expression으로 사용할 수 없다.
- GROUP BY 절의 expression으로 사용할 수 없다.
- DISTINCT 절의 expression으로 사용할 수 없다.
- UNION, INTERSECT, EXCEPT 절의 expression으로 사용할 수 없다.

참조

- CHARACTER
- CHARACTER VARYING

DATE

구문

DATE

설명

YEAR, MONTH, DAY, HOUR, MINUTE, SECOND (fractional seconds 제외)를 포함하는 날짜 형식이다.

- 값의 범위: '4714-11-24 BC' ~ '9999-12-31 AD' 범위의 날짜값
- 저장 공간의 크기: 8 bytes

참조

- Date Literals
- TIME
- TIMESTAMP

FLOAT

구문

Float [(precision)]

구문 규칙 및 파라미터

- precision: 유효 숫자의 이진 정밀도이다.
 - precision의 범위: 1 ~ 126
 - 기본값: 126

설명

이진 정밀도를 가지는 부동 소수점 값을 저장한다.

- 지수 범위: 1E-130 ~ 1E+125
- 저장 공간의 크기: (정수자리수 + 1) / 2 + (소수자리수 + 1) / 2 + 1 (Exponent와 sign 정보)

NUMBER, **NUMERIC** 타입과 달리 이진 precision 값을 갖는다.

- Alias names: REAL = FLOAT(24), DOUBLE = FLOAT(53), DOUBLE PRECISION = FLOAT(53), FLOAT4 = FLOAT(24), FLOAT8 = FLOAT(53)

참조

- NUMBER**
- NUMERIC**

INTERVAL

구문

```

<interval_type> ::=
    INTERVAL YEAR [ ( leading_precision ) ]
  | INTERVAL MONTH [ ( leading_precision ) ]
  | INTERVAL DAY [ ( leading_precision ) ]
  | INTERVAL HOUR [ ( leading_precision ) ]
  | INTERVAL MINUTE [ ( leading_precision ) ]
  | INTERVAL SECOND [ ( leading_precision [ , fractional_seconds_precision ] ) ]
  | INTERVAL YEAR [ ( leading_precision ) ] TO MONTH
  | INTERVAL DAY [ ( leading_precision ) ] TO HOUR
  | INTERVAL DAY [ ( leading_precision ) ] TO MINUTE
  | INTERVAL DAY [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]
  | INTERVAL HOUR [ ( leading_precision ) ] TO MINUTE
  | INTERVAL HOUR [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]
  | INTERVAL MINUTE [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]

```

구문 규칙 및 파라미터

- INTERVAL YEAR [(leading_precision)]: YEAR의 기간을 저장한다.
 - leading_precision
 - YEAR의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL MONTH [(leading_precision)]: MONTH의 기간을 저장한다.
 - leading_precision
 - MONTH의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL YEAR [(leading_precision)] TO MONTH: YEAR 와 MONTH 의 기간을 저장한다.
 - leading_precision
 - YEAR의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL DAY [(leading_precision)]: DAY의 기간을 저장한다.

- leading_precision
 - DAY의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL HOUR [(leading_precision)]: HOUR의 기간을 저장한다.
 - leading_precision
 - HOUR의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL MINUTE [(leading_precision)]: MINUTE의 기간을 저장한다.
 - leading_precision
 - MINUTE의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL SECOND [(leading_precision [, fractional_seconds_precision])]: SECOND의 기간을 저장한다.
 - leading_precision
 - SECOND의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
 - fractional_seconds_precision
 - fractional seconds의 digit 수
 - 값의 범위: 0 ~ 6
 - 기본값: 6
- INTERVAL DAY [(leading_precision)] TO HOUR: DAY 와 HOUR 의 기간을 저장한다.
 - leading_precision
 - DAY의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL DAY [(leading_precision)] TO MINUTE: DAY, HOUR, MINUTE 의 기간을 저장한다.
 - leading_precision
 - DAY의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL DAY [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: DAY, HOUR, MINUTE, SECOND의 기간을 저장한다.
 - leading_precision

- DAY의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- fractional_seconds_precision
 - fractional seconds의 digit 수
 - 값의 범위: 0 ~ 6
 - 기본값: 6
- INTERVAL HOUR [(leading_precision)] TO MINUTE: HOUR 와 MINUTE 의 기간을 저장한다.
 - leading_precision
 - HOUR의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
- INTERVAL HOUR [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: HOUR, MINUTE, SECOND 의 기간을 저장한다.
 - leading_precision
 - HOUR의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
 - fractional_seconds_precision
 - fractional seconds의 digit 수
 - 값의 범위: 0 ~ 6
 - 기본값: 6
- INTERVAL MINUTE [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: MINUTE 와 SECOND 의 기간을 저장한다.
 - leading_precision
 - MINUTE의 digit 수
 - 값의 범위: 2 ~ 6
 - 기본값: 2
 - fractional_seconds_precision
 - fractional seconds의 digit 수
 - 값의 범위: 0 ~ 6
 - 기본값: 6

설명

INTERVAL 타입은 값의 표현 범위에 따라 다음과 같이 YEAR TO MONTH 계열과 DAY TO SECOND 계열로 구분할 수 있다.

- YEAR TO MONTH 계열: 저장공간의 크기는 8 byte이다.

- INTERVAL YEAR
- INTERVAL MONTH
- INTERVAL YEAR TO MONTH
- DAY TO SECOND 계열: 저장 공간의 크기는 16 byte이다.
 - INTERVAL DAY
 - INTERVAL HOUR
 - INTERVAL MINUTE
 - INTERVAL SECOND
 - INTERVAL DAY TO HOUR
 - INTERVAL DAY TO MINUTE
 - INTERVAL DAY TO SECOND
 - INTERVAL HOUR TO MINUTE
 - INTERVAL HOUR TO SECOND
 - INTERVAL MINUTE TO SECOND

leading_precision이 지정된 field에 지정된 digit 이상의 숫자가 올 경우, 에러를 반환한다.

fractional_seconds_precision이 지정된 field에 지정된 digit 이상의 숫자가 올 경우, 반올림된다.

표 11-46 INTERVAL * TO * 에서 두 번째 이후 field의 precision과 값의 범위

Field	Precision	값의 범위
MONTH	2	0 ~ 11
HOUR	2	0 ~ 23
MINUTE	2	0 ~ 59
SECOND(소수점 이전값)	2	0 ~ 59

참조

자세한 내용은 **Interval Literals**를 참조한다.

NATIVE_BIGINT

구문

NATIVE_BIGINT

설명

Signed 8 byte 정수를 저장한다.

C 언어의 long long (8 bytes integer)과 동일하다.

- 값의 범위: -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
- 저장 공간의 크기: 8 bytes

NATIVE_DOUBLE

구문

NATIVE_DOUBLE

설명

Double precision floating-point number (8 bytes)를 저장한다.

C 언어의 double과 동일하다.

- 지수값의 범위: 1E-307 ~ 1E+308
- 저장 공간의 크기: 8 bytes

NATIVE_INTEGER

구문

NATIVE_INTEGER

설명

Signed 4 byte 정수를 저장한다.

C 언어의 integer (4 bytes)와 동일하다.

- 값의 범위: -2,147,483,648 ~ +2,147,483,647
- 저장 공간의 크기: 4 bytes

NATIVE_REAL

구문

NATIVE_REAL

설명

Single precision floating-point number (4 bytes)를 저장한다.

C 언어의 float과 동일하다.

- 지수값의 범위: $1\text{E}-37 \sim 1\text{E}+37$
- 저장 공간의 크기: 4 bytes

NATIVE_SMALLINT

구문

NATIVE_SMALLINT

설명

Signed 2 byte 정수를 저장한다.

C 언어의 short과 동일하다.

- 값의 범위: -32,768 ~ 32,767
- 저장 공간의 크기: 2 bytes

NUMBER

구문

NUMBER [(precision [, scale])]

구문 규칙 및 파라미터

- NUMBER: precision과 scale이 없는 부동 소수점 숫자를 저장한다.
 - 유효 숫자의 범위: 38
 - 지수값의 범위: 1E-130 ~ 1E+125
 - FLOAT(126)과 동일하다.
- NUMBER(precision): precision의 유효 자리수를 가지는 정수를 저장한다.
 - Precision의 범위: 1 ~ 38
 - Scale의 값: 0
 - FLOAT 타입과 달리 십진 precision 값을 갖는다.
 - NUMBER(precision, 0), NUMERIC(precision, 0)과 동일하다.
- NUMBER(precision, scale): precision과 scale을 갖는 고정 소수점 숫자를 저장한다.
 - Precision의 범위: 1 ~ 38
 - Scale의 범위: -84 ~ 127
 - FLOAT 타입과 달리 십진 precision 값을 갖는다.
 - NUMERIC(precision, scale)과 동일하다.
 - Alias names: SMALLINT = NUMBER(5,0), INTEGER = NUMBER(10,0), BIGINT = NUMBER(19,0), INT2 = NUMBER(5,0), INT4 = NUMBER(10,0), INT8 = NUMBER(19,0)

설명

NUMBER 타입은 NUMERIC 타입과 비슷하지만 precision과 scale을 모두 생략할 경우 NUMBER 타입은 precision과 scale이 정해지지 않은 부동 소수점 숫자를 저장한다는 점이 다르다.

- precision, scale이 없는 NUMBER: 부동 소수점 숫자
- precision, scale이 없는 NUMERIC: NUMERIC(38,0)의 고정 소수점 숫자
- 저장 공간의 크기: (정수자리수 + 1) / 2 + (소수자리수 + 1) / 2 + 1 (Exponent와 sign 정보)

참조

- FLOAT
- NUMERIC

NUMERIC

구문

NUMERIC [(precision [, scale])]

구문 규칙 및 파라미터

- precision: 유효 숫자의 십진 정밀도이다.
 - precision의 범위: 1 ~ 38
 - 기본값: 38
- scale: 소수점의 범위이다.
 - scale의 범위: -84 ~ 127
 - 기본값: 0

설명

precision과 scale을 갖는 고정 소수점 숫자를 저장한다.

precision과 scale을 생략할 경우, 다음과 같은 의미를 갖는다.

- NUMERIC = NUMERIC(38,0)
- NUMERIC(p) = NUMERIC(p,0)

NUMBER 타입은 NUMERIC 타입과 비슷하지만 precision과 scale을 모두 생략할 경우 NUMBER 타입은 precision과 scale이 정해지지 않은 부동 소수점 숫자를 저장한다는 점이 다르다.

- precision, scale이 없는 NUMBER: 부동 소수점 숫자
- precision, scale이 없는 NUMERIC: NUMERIC(38,0)의 고정 소수점 숫자
- 저장 공간의 크기: (정수자리수 + 1) / 2 + (소수자리수 + 1) / 2 + 1 (Exponent와 sign 정보)

참조

- FLOAT
- NUMBER

ROWID

구문

ROWID

설명

ROWID 타입은 레코드 식별자 (ROWID)를 저장한다.

레코드 식별자 (ROWID)는 데이터베이스 내 각 레코드의 식별정보이다.

ROWID pseudo column을 조회하면 각각의 레코드식별자 (ROWID) 값을 확인할 수 있는데, 이 ROWID pseudo column은 ROWID data type 정보를 갖는다.

Standalone system에서의 ROWID 타입의 구성요소는 다음과 같다.

- OBJECT_ID
- TABLESPACE_ID
- PAGE_ID
- PAGE 내 OFFSET

Cluster system에서의 ROWID 타입의 구성요소는 다음과 같다.

- GRID_BLOCK_SEQUENCE
- GRID_BLOCK_ID
- MEMBER_ID
- SHARD_ID

ROWID는 A~Z, a~z, 0~9, +, / 를 포함할 수 있는 base 64 value로 저장된다.

ROWID-related functions를 이용하여 ROWID 각각의 구성요소 정보를 얻을 수 있다.

- 저장 공간의 크기: 16 bytes

참조

- ROWID Pseudo Column
- ROWID-related Functions

TIME

구문

TIME [(fractional_seconds_precision)] [WITH TIME ZONE | WITHOUT TIME ZONE]

구문 규칙 및 파라미터

- fractional_seconds_precision: fractional seconds의 유효자리 수이다.
 - fractional_seconds_precision의 범위: 0 ~ 6
 - 기본값: 6
- [WITH TIME ZONE | WITHOUT TIME ZONE]: TIME ZONE 값 저장 여부이다.
 - WITH TIME ZONE: time zone을 포함한 시간
 - WITHOUT TIME ZONE: time zone을 포함하지 않는 시간
 - 기본값: WITHOUT TIME ZONE

설명

HOUR, MINUTE, SECOND를 가지는 시간을 저장한다.

- 저장 공간의 크기
 - TIME WITHOUT TIME ZONE: 8 bytes
 - TIME WITH TIME ZONE: 12 bytes

참조

- Time Literals
- Time with Time Zone Literals
- DATE
- TIMESTAMP

TIMESTAMP

구문

```
TIMESTAMP [ ( fractional_seconds_precision ) ] [ WITH TIME ZONE | WITHOUT TIME ZONE ]
```

구문 규칙 및 파라미터

- fractional_seconds_precision: fractional seconds의 유효자리 수이다.
 - fractional_seconds_precision의 범위: 0 ~ 6
 - 기본값: 6
- [WITH TIME ZONE | WITHOUT TIME ZONE]: TIME ZONE 값 저장 여부이다.
 - WITH TIME ZONE: time zone을 포함한 시간
 - WITHOUT TIME ZONE: time zone을 포함하지 않는 시간
 - 기본값: WITHOUT TIME ZONE

설명

YEAR, MONTH, DATE, HOUR, MINUTE, SECOND를 가지는 시간을 저장한다.

- 저장 공간의 크기
 - TIMESTAMP WITHOUT TIME ZONE: 8 bytes
 - TIMESTAMP WITH TIME ZONE: 12 bytes

참조

- Timestamp Literals
- Timestamp with Time Zone Literals
- DATE
- TIME

11.10 Built-in Function References

* (MULTIPLICATION)

구문

`expr1 * expr2`

설명

expr1과 expr2의 곱하기 연산결과를 반환한다.

곱하기 연산의 종류와 결과 타입은 다음 표와 같다.

자세한 내용은 **타입간 변환**을 참조한다.

표 11-47 숫자형 * 연산

expr1 (expr2)	expr2 (expr1)	결과 타입
NATIVE INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

표 11-48 INTERVAL * 연산

expr1 (expr2)	expr2 (expr1)	결과 타입
INTERVAL YEAR TO MONTH	숫자형 타입	INTERVAL YEAR TO MONTH (결과 타입은 interval type 이다.)
INTERVAL DAY TO SECOND	숫자형 타입	INTERVAL DAY TO SECOND (결과 타입은 interval type 이다.)
자세한 내용은 표에 표기된 INTERVAL 타입이 포함하는 INTERVAL 세부타입 을 참조한다.		

표 11-49 표에 표기된 INTERVAL 타입이 포함하는 INTERVAL 세부타입

INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
	- INTERVAL DAY - INTERVAL HOUR - INTERVAL MINUTE

INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
• INTERVAL YEAR • INTERVAL MONTH • INTERVAL YEAR TO MONTH	• INTERVAL SECOND • INTERVAL DAY TO HOUR • INTERVAL DAY TO MINUTE • INTERVAL DAY TO SECOND • INTERVAL HOUR TO MINUTE • INTERVAL HOUR TO SECOND • INTERVAL MINUTE TO SECOND

사용 예

```
gSQL> SELECT INTERVAL '1-2' YEAR TO MONTH * 2 AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
+000002-04
```

```
1 row selected.
```

```
gSQL> SELECT INTERVAL '1 01:02:03.400000' DAY TO SECOND * 2 AS RESULT
```

```
FROM DUAL;
```

```
RESULT
```

```
-----
```

```
+000002 02:04:06.800000
```

```
1 row selected.
```

+ (ADDITION)

구문

```
expr1 + expr2
```

설명

expr1과 expr2의 더하기 연산 결과를 반환한다.

더하기 연산의 종류와 결과 타입은 다음 표와 같다.

자세한 내용은 **타입간 변환**을 참조한다.

표 11-50 숫자형 + 연산

expr1 (expr2)	expr2 (expr1)	결과 타입
NATIVE INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

표 11-51 (DATETIME/INTERVAL) + 연산

expr1 (expr2)	expr2 (expr1)	결과 타입
DATE	NUMERIC	DATE
DATE	INTERVAL YEAR TO MONTH	DATE
DATE	INTERVAL DAY	DATE
DATE	INTERVAL DAY TO SECOND	TIMESTAMP
TIME	NUMERIC	TIME
TIME	INTERVAL YEAR TO MONTH	TIME
TIME	INTERVAL DAY TO SECOND	TIME
TIME WITH TIME ZONE	NUMERIC	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL DAY TO SECOND	TIME WITH TIME ZONE
TIMESTAMP	NUMERIC	TIMESTAMP
TIMESTAMP	INTERVAL YEAR TO MONTH	TIMESTAMP
TIMESTAMP	INTERVAL DAY TO SECOND	TIMESTAMP
TIMESTAMP WITH TIME ZONE	NUMERIC	TIMESTAMP WITH TIME ZONE

expr1 (expr2)	expr2 (expr1)	결과 타입
TIMESTAMP WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND	TIMESTAMP WITH TIME ZONE
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH (expr1과 expr2의 interval 범위를 모두 포함할 수 있는 타입으로 결정된다.)
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND (expr1과 expr2의 interval 범위를 모두 포함할 수 있는 타입으로 결정된다.)
자세한 내용은 표에 표기된 INTERVAL 타입이 포함하는 INTERVAL 세부타입 을 참조한다.		

사용 예

```

gSQL> SELECT TO_DATE( '2012-05-05', 'YYYY-MM-DD' ) + 5 AS RESULT FROM DUAL;
RESULT
-----
2012-05-10
1 row selected.
gSQL> SELECT
    TO_DATE( '2012-05-05', 'YYYY-MM-DD' ) + INTERVAL'01-01'YEAR TO MONTH
    AS RESULT
    FROM DUAL;
RESULT
-----
2013-06-05
1 row selected.
gSQL> SELECT
    INTERVAL'01-01'YEAR TO MONTH + INTERVAL'02-10'YEAR TO MONTH
    AS RESULT
    FROM DUAL;
RESULT
-----
+000003-11
1 row selected.

```

+ (POSITIVE)

구문

```
+ expr
```

설명

expr에 + 부호를 표시한다.

사용 예

```
gSQL> SELECT +3 AS RESULT1, +(-3) AS RESULT2 FROM DUAL;  
RESULT1 RESULT2  
-----  
3      -3  
1 row selected.
```

- (NEGATIVE)

구문

- expr

설명

expr에 - 부호를 표시한다.

사용 예

```
gSQL> SELECT -3 AS RESULT1, -(-3) AS RESULT2 FROM DUAL;  
RESULT1 RESULT2  
-----  
      -3        3  
1 row selected.
```

- (SUBTRACTION)

구문

```
expr1 - expr2
```

설명

expr1과 expr2의 뺄셈 연산 결과를 반환한다.

뺄셈 연산의 종류와 결과 타입은 다음 표와 같다.

자세한 내용은 **타입간 변환**을 참조한다.

표 11-52 숫자형 - 연산

expr1	expr2	결과 타입
NATIVE INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE 계열 • NATIVE_REAL • NATIVE_DOUBLE	NATIVE DOUBLE 계열 • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

표 11-53 (DATETIME/INTERVAL) - 연산

expr1	expr2	결과 타입
DATE	DATE	NUMBER
DATE	숫자 타입	DATE
DATE	INTERVAL YEAR TO MONTH	DATE
DATE	INTERVAL DAY	DATE
DATE	INTERVAL DAY TO SECOND	TIMESTAMP
TIME	TIME	INTERVAL DAY TO SECOND
TIME	숫자 타입	TIME
TIME	INTERVAL YEAR TO MONTH	TIME
TIME	INTERVAL DAY TO SECOND	TIME
TIME WITH TIME ZONE	숫자 타입	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL DAY TO SECOND	TIME WITH TIME ZONE
TIMESTAMP	TIMESTAMP	INTERVAL DAY TO SECOND
TIMESTAMP	숫자 타입	TIMESTAMP

expr1	expr2	결과 타입
TIMESTAMP	INTERVAL YEAR TO MONTH	TIMESTAMP
TIMESTAMP	INTERVAL DAY TO SECOND	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND
TIMESTAMP WITH TIME ZONE	숫자 타입	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND	TIMESTAMP WITH TIME ZONE
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH (expr1과 expr2의 interval 범위를 모두 포함할 수 있는 타입으로 결정된다.)
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND (expr1과 expr2의 interval 범위를 모두 포함할 수 있는 타입으로 결정된다.)
자세한 내용은 표에 표기된 INTERVAL 타입이 포함하는 INTERVAL 세부타입 을 참조한다.		

사용 예

```

gSQL> SELECT
    TO_DATE( '2012-05-05' ) - TO_DATE( '2012-01-01' ) AS RESULT
FROM DUAL;

RESULT
-----
    125
1 row selected.
gSQL> SELECT TO_DATE( '2012-05-05' ) - 3 AS RESULT FROM DUAL;
RESULT
-----
2012-05-02
1 row selected.
gSQL> SELECT
    TO_DATE( '2012-05-05' ) - INTERVAL'01-02'YEAR TO MONTH AS RESULT
FROM DUAL;

RESULT
-----
2011-03-05
1 row selected.
gSQL> SELECT
    INTERVAL'05-11'YEAR TO MONTH - INTERVAL'02-01'YEAR TO MONTH
    AS RESULT
FROM DUAL;

```

RESULT

+000003-10

1 row selected.

```
gSQL> SELECT INTERVAL'15 23:59:59.999999'DAY TO SECOND
        - INTERVAL'10 23:59:59.999999'DAY TO SECOND AS RESULT
        FROM DUAL;
```

RESULT

+000005 00:00:00.000000

1 row selected.

/ (DIVISION)

구문

```
expr1 / expr2
```

설명

expr1과 expr2의 나누기 연산 결과를 반환한다.

나누기 연산의 종류와 결과 타입은 다음 표와 같다.

자세한 내용은 [타입간 변환](#)을 참조한다.

표 11-54 숫자형/ 연산

expr1	expr2	결과 타입
NATIVE INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER	NUMBER
NATIVE_DOUBLE	NATIVE_DOUBLE	NATIVE_DOUBLE

표 11-55 (DATETIME/INTERVAL)/ 연산

expr1	expr2	결과 타입
INTERVAL YEAR TO MONTH	숫자 타입	INTERVAL YEAR TO MONTH (결과 타입은 interval type이다.)
INTERVAL DAY TO SECOND	숫자 타입	INTERVAL DAY TO SECOND (결과 타입은 interval type이다.)

자세한 내용은 [표에 표기된 INTERVAL 타입이 포함하는 INTERVAL 세부타입](#)을 참조한다.

사용 예

```
gSQL> SELECT INTERVAL'20-10'YEAR TO MONTH / 2 AS RESULT FROM DUAL;
RESULT
-----
+000010-05
1 row selected.
gSQL> SELECT INTERVAL'02 02:04:06.800000'DAY TO SECOND / 2 AS RESULT
FROM DUAL;
```

RESULT

+000001 01:02:03.400000

1 row selected.

|| (CONCATENATE)

구문

```
str1 || str2
```

설명

CONCATENATE는 str1과 str2를 연결한 문자열을 반환한다.

str1과 str2 중 하나가 null인 경우 null이 아닌 나머지 str이 반환되고, str1과 str2가 모두 null인 경우 NULL이 반환된다.

인자로는 character string 타입 또는 binary string 타입으로 변환될 수 있는 타입이 올 수 있다.

자세한 내용은 [타입간 변환](#)을 참조한다.

CONCAT, CONCATENATE의 alias 이다.

결과 타입은 다음 표와 같다.

표 11-56 || (CONCATENATE)의 결과 타입

Data type	CHAR	VARCHAR	LONG VARCHAR
CHAR	CHAR	VARCHAR	LONG VARCHAR
VARCHAR	VARCHAR	VARCHAR	LONG VARCHAR
LONG VARCHAR	LONG VARCHAR	LONG VARCHAR	LONG VARCHAR
Data type	BINARY	VARBINARY	LONG VARBINARY
BINARY	BINARY	VARBINARY	LONG VARBINARY
VARBINARY	VARBINARY	VARBINARY	LONG VARBINARY
LONG VARBINARY	LONG VARBINARY	LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT 'DATA' || 'BASE' AS RESULT1,
            'DATA' || NULL  AS RESULT2,
            NULL || NULL   AS RESULT3
      FROM DUAL;
RESULT1  RESULT2  RESULT3
-----
DATABASE DATA    null
1 row selected.
```

ABS

구문

ABS(num)

설명

ABS는 num의 절대값을 반환한다.

인자 num에는 숫자 타입 또는 숫자로 변환될 수 있는 타입이 올 수 있다.

사용 예

```
gSQL> SELECT ABS(-1) AS RESULT1, ABS(1) AS RESULT2 FROM DUAL;
```

```
RESULT1 RESULT2
```

```
-----
```

```
1      1
```

```
1 row selected.
```

ACOS

구문

ACOS(num)

설명

ACOS 함수는 num의 arc cosine 값을 반환한다.
인자 num은 -1 이상 1 이하의 값이어야 한다.
0 ~ pi 사이의 라디안 값을 반환한다.

사용 예

```
gSQL> SELECT ACOS( 1 ) FROM DUAL;  
ACOS( 1 )  
-----  
          0  
1 row selected.
```

ADDDATE

구문

```
ADDDATE( date, INTERVAL expr unit )
ADDDATE( expr, days )
```

설명

ADDDATE는 입력받은 첫 번째 인자에 두 번째 인자를 더하기 연산하여 그 결과를 반환한다.

입력 인자의 값이 하나라도 NULL이면 결과값도 NULL이다.

첫 번째 인자에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있고, 두 번째 인자에는 INTERVAL 또는 숫자 타입이 올 수 있다.

결과 타입은 (DATETIME/INTERVAL) + 연산과 동일하다.

사용 예

```
gSQL> SELECT ADDDATE( TO_DATE( '2012-12-12', 'YYYY-MM-DD' ), 1 ) AS RESULT
      FROM DUAL;
RESULT
-----
2012-12-13
1 row selected.
gSQL> SELECT ADDDATE( TO_DATE( '2012-12-12', 'YYYY-MM-DD' ),
      INTERVAL '01-01' YEAR TO MONTH ) AS RESULT
      FROM DUAL;
RESULT
-----
2014-01-12
1 row selected.
```


ADDTIME

구문

ADDTIME(expr1, expr2)

설명

ADDTIME은 입력받은 expr2를 expr1에 더하여 그 결과를 반환한다.

expr1이나 expr2가 NULL이면 결과값은 NULL이다.

expr1에는 TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE TYPE이 올 수 있고, expr2에는 INTERVAL DAY TO SECOND TYPE이 올 수 있다.

결과 타입은 (DATETIME/INTERVAL) + 연산과 동일하다.

사용 예

```
gSQL> SELECT
      ADDTIME( TO_TIMESTAMP( '2001-05-10 11:22:33',
                           'YYYY-MM-DD HH24:MI:SS' ),
              INTERVAL'0 01:02:03.999999'DAY TO SECOND ) AS RESULT
    FROM DUAL;

RESULT
-----
2001-05-10 12:24:36.999999
1 row selected.
```

ADD_MONTHS

구문

ADD_MONTHS(date, number)

설명

ADD_MONTHS는 date에 number 숫자만큼의 달을 더한 값을 반환한다.

입력 인자의 값이 하나라도 NULL이면 결과값도 NULL이다.

만약 ADD_MONTHS 연산 후에 날짜가 그 달의 마지막 날보다 큰 경우에는 마지막 날짜로 조정한다.

인자 date에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있으며 인자 number에는 숫자 타입이 올 수 있다.

결과 타입은 입력 인자 date 타입과 관계없이 항상 DATE 타입이다.

사용 예

```
gSQL> SELECT
      ADD_MONTHS( TO_DATE( '2001-07-31', 'YYYY-MM-DD' ), 1 ) AS RESULT1,
      ADD_MONTHS( TO_DATE( '2001-07-31', 'YYYY-MM-DD' ), 2 ) AS RESULT2
    FROM DUAL;
RESULT1    RESULT2
-----
2001-08-31 2001-09-30
1 row selected.
```

ASCII

구문

ASCII(char)

설명

char의 첫 번째 문자에 대한 database character set code를 십진수로 반환한다.

char에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입 또는 문자 타입으로 변환될 수 있는 타입이 올 수 있고, 반환되는 타입은 NUMBER 이다.

사용 예

```
gSQL> SELECT ASCII( 'G' ) AS RESULT FROM DUAL;  
RESULT  
-----  
       71  
1 row selected.
```

ASIN

구문

ASIN(num)

설명

ASIN 함수는 num의 arc sin 값을 반환한다.
인자 num은 -1 이상 1 이하의 값이어야 한다.
 $-\pi/2 \sim \pi/2$ 사이의 라디안 값을 반환한다.

사용 예

```
gSQL> SELECT ASIN( 0 ) FROM DUAL;  
ASIN( 0 )  
-----  
          0  
1 row selected.
```

ATAN

구문

ATAN(num)

설명

ATAN 함수는 num의 arc tangent 값을 반환한다.

num 값 범위의 제한은 없으며, $-\pi/2 \sim \pi/2$ 사이의 라디안 값을 반환한다.

사용 예

```
gSQL> SELECT ATAN( 1 ) FROM DUAL;  
          ATAN( 1 )  
-----  
.785398163397448  
1 row selected.
```

ATAN2

구문

```
ATAN2( num1, num2 )
```

설명

ATAN2 함수는 num1과 num2의 arc tangent 값을 반환한다.

인자 num1 값 범위의 제한은 없으며, $-\pi \sim \pi$ 사이의 라디안 값을 반환한다.

사용 예

```
gSQL> SELECT ATAN2( 1, 0 ) FROM DUAL;  
      ATAN2( 1, 0 )  
-----  
1.5707963267949  
1 row selected.
```

AVG

구문

```
AVG( [ ALL | DISTINCT ] num )
```

설명

Aggregation 함수로써 expr 들의 평균값을 얻는데 사용된다.

ALL을 명시한 경우, 모든 값들에 대한 aggregation을 수행한다.

DISTINCT를 명시한 경우, 중복 제거한 값들에 대한 aggregation을 수행한다.

ALL이나 DISTINCT를 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

사용 예

```
gSQL> SELECT AVG(c1) FROM t1;
AVG(C1)
-----
      2
1 row selected.
```

BITAND

구문

```
BITAND( num1, num2 )
```

설명

num1과 num2의 비트에 대한 AND 연산 결과를 반환한다.

입력 인자에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환할 때 소수점은 TRUNCATE 된다.

결과 타입은 NATIVE_BIGINT이다.

사용 예

```
gSQL> SELECT BITAND( 5, 3 ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
1
```

```
1 row selected.
```


BITNOT

구문

BITNOT(num)

설명

num의 비트에 대해 NOT 연산한 결과를 반환한다.

입력 인자에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환할 때 소수점은 TRUNCATE 된다.

결과 타입은 다음과 같다.

- 입력 인자가 NATIVE_SMALLINT인 경우, NATIVE_SMALLINT
- 입력 인자가 NATIVE_INTEGER인 경우, NATIVE_INTEGER
- 입력 인자가 NATIVE_BIGINT인 경우, NATIVE_BIGINT

사용 예

```
gSQL> SELECT BITNOT( 5 ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
      -6
```

```
1 row selected.
```

BITOR

구문

```
BITOR( num1, num2 )
```

설명

num1과 num2의 비트에 대해 OR 연산한 결과를 반환한다.

입력 인자에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환할 때 소수점은 TRUNCATE 된다.

결과 타입은 NATIVE_BIGINT이다.

사용 예

```
gSQL> SELECT BITOR( 5, 3 ) FROM DUAL;  
BITOR( 5, 3 )  
-----  
              7  
1 row selected.
```

BITXOR

구문

```
BITXOR( num1, num2 )
```

설명

num1과 num2의 비트에 대해 XOR 연산한 결과를 반환한다.

입력 인자에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환할 때 소수점은 TRUNCATE 된다.

결과 타입은 NATIVE_BIGINT이다.

사용 예

```
gSQL> SELECT BITXOR( 5, 3 ) FROM DUAL;  
BITXOR( 5, 3 )  
-----  
6  
1 row selected.
```

BIT_LENGTH

구문

```
BIT_LENGTH( str )
```

설명

BIT_LENGTH는 str의 비트 수를 반환한다.

사용 예

```
gSQL> SELECT BIT_LENGTH( 'LIKE' ) AS RESULT FROM DUAL;  
RESULT  
-----  
32  
1 row selected.
```

BYTE_LENGTH

구문

```
BYTE_LENGTH( str )
```

설명

OCTET_LENGTH의 alias이다.

자세한 내용은 **OCTET_LENGTH**, **LENGTHB**를 참조한다.

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT BYTE_LENGTH( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
      FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----
                        12
1 row selected.
```

- Multi byte character set (예: UTF8): 2 byte character

```
gSQL> SELECT BYTE_LENGTH( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
-----
                        4
1 row selected.
```

CASE2

구문

```
CASE2( condition1, result1
      [, condition2, result2
        , ...
        , conditionN, resultN ]
      [, default ] )
```

설명

CASE2는 기술된 순서대로 condition을 평가한다.

비교 결과가 FALSE이면 TRUE가 나올 때까지 평가한다.

비교 결과가 TRUE이면 대응되는 result를 반환하고, 이후는 평가하지 않는다.

비교 결과가 모두 FALSE인 경우에는 default를 반환하고, default가 생략된 경우에는 NULL을 반환한다.

결과 타입은 result1 (첫 번째 result)의 데이터 타입으로 결정된다.

result1 (첫 번째 result)의 데이터 타입이 숫자형인 경우와 문자형인 경우에는 각각 result1, ..., resultN의 범위를 포함할 수 있는 타입으로 결정된다.

result1 (첫 번째 result)가 CHAR 타입 또는 NULL인 경우, 결과 타입은 VARCHAR 이다.

CASE2는 CASE로 동일하게 표현할 수 있다.

- CASE2(condition1, res1, condition2, res2)

```
CASE WHEN condition1 THEN res1
      WHEN condition2 THEN res2
      ELSE NULL
END
```

- CASE2(condition1, res1, condition2, res2, default)

```
CASE WHEN condition1 THEN res1
      WHEN condition2 THEN res2
      ELSE default
END
```

사용 예

```
gSQL> SELECT I1,  
           CASE2( I1 = 1, 'ONE', I1 = 2, 'TWO' ) AS CASE2_RESULT1,  
           CASE2( I1 = 1, 'ONE', I1 = 2, 'TWO', 'NUMBER' ) AS CASE2_RESULT2  
           FROM T1;  
I1 CASE2_RESULT1 CASE2_RESULT2  
-- -----  
1 ONE           ONE  
2 TWO           TWO  
3 null          NUMBER  
3 rows selected.
```

CBRT

구문

CBRT(num)

설명

num의 세제곱근을 반환한다.

인자값 num이 NULL이면 결과값도 NULL이다.

사용 예

```
gSQL> SELECT CBRT( 27 ) FROM DUAL;
```

```
CBRT( 27 )
```

```
-----
```

```
3
```

```
1 row selected.
```


CEIL

구문

```
CEIL( num )  
CEILING( num )
```

설명

CEIL 함수는 num 보다 크거나 같은 가장 작은 정수를 반환한다.

사용 예

```
gSQL> SELECT CEIL( 3.5 ) AS RESULT1, CEIL( -3.5 ) AS RESULT2 FROM DUAL;  
RESULT1 RESULT2  
-----  
4      -3  
1 row selected.
```

CHAR_LENGTH

구문

```
CHAR_LENGTH( str )  
CHARACTER_LENGTH( str )
```

설명

CHAR_LENGTH는 str에 대해 character set에 따른 문자수를 반환한다.

str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입 또는 문자 타입으로 변환될 수 있는 타입이 올 수 있고, 반환되는 타입은 NATIVE_BIGINT 이다.

str의 타입이 CHARACTER 타입이면 공백문자 (trailing blank)를 포함하여 계산한다.

str이 NULL이면 NULL이 반환된다.

LENGTH의 alias이다.

사용 예

Multi byte character set: (예:UTF8)

```
gSQL> SELECT CHAR_LENGTH( 'αβ-SUMMER' ) AS RESULT FROM DUAL;  
RESULT  
-----  
          9  
1 row selected.
```

CHR

구문

CHR(num)

설명

num에 대응하는 database character set code 내의 character를 반환한다.

입력 인자에는 숫자형 타입이 올 수 있으며 결과 타입은 VARCHAR 이다.

사용 예

```
gSQL> SELECT CHR(71) FROM DUAL;  
CHR(71)  
-----  
G  
1 row selected.
```

CLOCK_DATE

구문

CLOCK_DATE()

설명

함수가 호출될 때마다 현재 날짜 (DATE type) 값을 얻는다.

현재 날짜를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_DATE(): 트랜잭션 내의 모든 날짜값이 동일하다.
- STATEMENT_DATE(): 하나의 SQL 문장 내에서 모든 날짜 값이 동일하다.
- CLOCK_DATE(): 함수가 호출될 때마다 현재 날짜 값을 얻는다.

사용 예

Row 마다 다른 값을 가질 수 있다.

```
gSQL> SELECT CLOCK_DATE() FROM t1;
CLOCK_DATE()
-----
2013-12-12
2013-12-12
2013-12-13
3 rows selected.
```

CLOCK_LOCALTIME

구문

CLOCK_LOCALTIME()

설명

함수가 호출될 때마다 TIME_ZONE이 없는 현재 시간 (TIME WITHOUT TIME_ZONE type) 값을 얻는다.

현재 시간을 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_LOCALTIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_LOCALTIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

Row 마다 시간값이 다를 수 있다.

```
gSQL> SELECT CLOCK_LOCALTIME() FROM t1;
CLOCK_LOCALTIME()
-----
14:42:05.470757
14:42:05.470759
14:42:05.470759
3 rows selected.
```

CLOCK_LOCALTIMESTAMP

구문

CLOCK_LOCALTIMESTAMP()

설명

함수가 호출될 때마다 TIME_ZONE이 없는 현재 TIMESTAMP (TIMESTAMP WITHOUT TIME_ZONE type) 값을 얻는다.

현재 TIMESTAMP를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_LOCALTIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_LOCALTIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

Row 마다 값이 다를 수 있다.

```
gSQL> SELECT CLOCK_LOCALTIMESTAMP() FROM t1;
CLOCK_LOCALTIMESTAMP()
-----
2013-12-12 14:46:17.309206
2013-12-12 14:46:17.309209
2013-12-12 14:46:17.309209
```

CLOCK_TIME

구문

CLOCK_TIME()

설명

함수가 호출될 때마다 TIME_ZONE이 있는 현재 시간 (TIME WITH TIME_ZONE type) 값을 얻는다.

현재 시간을 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_TIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_TIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_TIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

Row 마다 시간값이 다를 수 있다.

```
gSQL> SELECT CLOCK_TIME() FROM t1;
CLOCK_TIME()
-----
14:48:21.052324 +09:00
14:48:21.052326 +09:00
14:48:21.052327 +09:00
3 rows selected.
```

CLOCK_TIMESTAMP

구문

CLOCK_TIMESTAMP()

설명

함수가 호출될 때마다 TIME_ZONE이 있는 현재 TIMESTAMP (TIMESTAMP WITH TIME_ZONE type) 값을 얻는다.

현재 TIMESTAMP를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_TIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_TIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_TIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

Row 마다 값이 다를 수 있다.

```
gSQL> SELECT CLOCK_TIMESTAMP() FROM t1;
CLOCK_TIMESTAMP()
-----
2013-12-12 14:49:45.051709 +09:00
2013-12-12 14:49:45.051714 +09:00
2013-12-12 14:49:45.051714 +09:00
3 rows selected.
```


COALESCE

구문

```
COALESCE( expr1, ..., exprN )
```

설명

expr list들 중에 null이 아닌 첫 번째 expr을 반환한다.

expr list들이 모두 null인 경우에는 null을 반환한다.

expr은 두 개 이상이어야 한다.

expr list에 여러 type들이 오는 경우에는 **결과 타입 조합 규칙**에 따라 result type이 결정된다.

COALESCE는 CASE를 사용하여 동일하게 표현할 수 있다.

- COALESCE(expr1, expr2)

```
CASE WHEN expr1 IS NOT NULL THEN expr1
      ELSE expr2
END
```

- COALESCE(expr1, expr2, ..., exprN)

```
CASE WHEN expr1 IS NOT NULL THEN expr1
      ELSE COALESCE( expr2, ..., exprN )
END
```

사용 예

```
gSQL> SELECT COALESCE( NULL, 1, 2 ) FROM DUAL;
COALESCE( NULL, 1, 2 )
-----
1
1 row selected.
gSQL> SELECT COALESCE( NULL, NULL, NULL ) FROM DUAL;
COALESCE( NULL, NULL, NULL )
-----
null
1 row selected.
```

CONCAT

구문

```
CONCAT( str1, str2, ... )
```

설명

|| (CONCATENATE)의 alias 이다.

CONCAT 함수의 argument로써 2 ~ 254 개까지 설정할 수 있다.

자세한 내용은 || (CONCATENATE), CONCATENATE를 참조한다.

사용 예

```
gSQL> SELECT CONCAT( 'DATA', 'BASE' ) AS RESULT FROM DUAL;  
RESULT  
-----  
DATABASE  
1 row selected.
```

CONCATENATE

구문

```
CONCATENATE( str1, str2, ... )
```

설명

|| (CONCATENATE) 의 alias 이다.

CONCATENATE 함수의 argument로 2 ~ 254 개까지 설정할 수 있다.

자세한 내용은 **CONCAT**, **|| (CONCATENATE)** 를 참조한다.

사용 예

```
gSQL> SELECT CONCATENATE( 'DATA', 'BASE' ) AS RESULT FROM DUAL;  
RESULT  
-----  
DATABASE  
1 row selected.
```

COS

구문

COS(num)

설명

num 의 COSINE 값을 반환한다.

인자값 num이 NULL이면 결과값도 NULL이다.

사용 예

```
gSQL> SELECT COS( 0 ) FROM DUAL;
```

```
COS( 0 )
```

```
-----
```

```
1
```

```
1 row selected.
```

COT

구문

COT(num)

설명

num의 COTANGENT 값을 반환한다.
인자값 num이 NULL이면 결과값도 NULL이다.

사용 예

```
gSQL> SELECT COT( 1 ) FROM DUAL;  
          COT( 1 )  
-----  
.642092615934331  
1 row selected.
```

COUNT

구문

```
COUNT( [ ALL | DISTINCT ] expr )
```

설명

Aggregation 함수로써 expr이 NULL 값이 아닌 row의 개수를 얻는다.

ALL을 명시한 경우, 모든 값들에 대한 aggregation을 수행한다.

DISTINCT을 명시한 경우, 중복 제거한 값들에 대한 aggregation을 수행한다.

ALL이나 DISTINCT를 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

사용 예

```
gSQL> SELECT COUNT(c1) FROM t1;
COUNT(C1)
-----
          3
1 row selected.
```

COUNT(*)

구문

COUNT(*)

설명

Aggregation 함수로써 row의 개수를 얻는다.

별도의 expression을 지정하지 않으므로 값의 NULL 여부와 무관하다.

사용 예

```
gSQL> SELECT COUNT(*) FROM t1;
```

```
COUNT(*)
```

```
-----
```

```
4
```

```
1 row selected.
```

CURRENT_CATALOG

구문

CURRENT_CATALOG [()]

설명

catalog name (database 이름)을 얻는다.

사용 예

```
gSQL> SELECT CURRENT_CATALOG FROM dual;  
CURRENT_CATALOG  
-----  
TEST_DB  
1 row selected.
```


CURRENT_DATE

구문

```
CURRENT_DATE [( )]  
STATEMENT_DATE()
```

설명

현재 날짜 (DATE type) 값을 얻는다.

CURRENT_DATE는 SQL 표준 함수이다.

현재 날짜를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_DATE(): 트랜잭션 내의 모든 날짜값이 동일하다.
- CURRENT_DATE, STATEMENT_DATE(): 하나의 SQL 문장 내에서 모든 날짜 값이 동일하다.
- CLOCK_DATE(): 함수가 호출될 때마다 현재 날짜 값을 얻는다.

사용 예

```
gSQL> SELECT CURRENT_DATE FROM t1;  
CURRENT_DATE  
-----  
2013-12-12  
2013-12-12  
2013-12-12  
3 rows selected.
```

CURRENT_SCHEMA

구문

CURRENT_SCHEMA [()]

설명

사용자의 현재 SCHEMA를 얻는다.

사용 예

```
gSQL> SELECT CURRENT_SCHEMA FROM dual;  
CURRENT_SCHEMA  
-----  
PUBLIC  
1 row selected.
```

CURRENT_TIME

구문

```
CURRENT_TIME [( )]
STATEMENT_TIME()
```

설명

Session 시간을 기준으로 현재 TIME WITH TIME ZONE type 값을 얻는다.

CURRENT_TIME은 SQL 표준 함수이다.

현재 날짜를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_TIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- CURRENT_TIME, STATEMENT_TIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_TIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

모든 row가 같은 시간값을 갖는다.

```
gSQL> SELECT CURRENT_TIME FROM t1;
CURRENT_TIME
-----
16:27:10.116396 +09:00
16:27:10.116396 +09:00
16:27:10.116396 +09:00
3 rows selected.
```

CURRENT_TIMESTAMP

구문

```
CURRENT_TIMESTAMP [()]  
STATEMENT_TIMESTAMP()
```

설명

Session 시간을 기준으로 TIMESTAMP WITH TIME ZONE type 값을 얻는다.

CURRENT_TIMESTAMP는 SQL 표준 함수이다.

현재 TIMESTAMP를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_TIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- CURRENT_TIMESTAMP, STATEMENT_TIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_TIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT CURRENT_TIMESTAMP FROM t1;  
CURRENT_TIMESTAMP  
-----  
2013-12-12 16:34:55.649632 +09:00  
2013-12-12 16:34:55.649632 +09:00  
2013-12-12 16:34:55.649632 +09:00  
3 rows selected.
```

CURRENT_USER

구문

CURRENT_USER [()]

설명

현재 사용자를 반환한다.

사용자 정보는 다음과 같은 세 가지 형태로 관리된다.

- Logon user: login을 실행한 user로서 connection을 닫을 때까지 유지된다.
- Session user: 최초 logon user와 동일하지만 SET SESSION AUTHORIZATION 구문을 이용해 변경할 수 있다.
- Current user: 일반적으로 session user와 동일하지만 PSM, view 등을 사용할 때 접근 제어등을 위해 시스템 내부적으로 잠시 변경된다.
 - Session user와 current user는 unix system의 real user와 effective user의 차이와 유사한 개념이다.

사용 예

```
% gsql sys gliese
gSQL> SET SESSION AUTHORIZATION test;
Session set.
gSQL> SELECT
    LOGON_USER() AS result1,
    SESSION_USER() AS result2,
    CURRENT_USER() AS result3
FROM DUAL;
RESULT1 RESULT2 RESULT3
-----
SYS      TEST      TEST
1 row selected.
```

CURRVAL

구문

```
seq_name.CURRVAL  
CURRVAL(seq_name)
```

설명

시퀀스 객체의 현재 값을 얻는다.

최소 한 번은 NEXTVAL(seq_name) 등으로 시퀀스 값을 설정해야 한다.

사용 예

```
gSQL> SELECT seq.CURRVAL FROM dual;  
SEQ.CURRVAL  
-----  
1  
1 row selected.
```

DATEADD

구문

DATEADD(datepart, number, date)

설명

date의 지정된 datepart에 number를 더한 값을 반환한다.

number가 소수점인 경우 반올림되지 않는다.

date에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE 타입이 올 수 있다.

number 또는 date가 NULL인 경우에는 결과값도 NULL이다.

인자로 받은 date의 타입과 동일한 결과 타입이 반환된다.

표 11-57 datepart에 사용 가능한 형식문자열

datepart	설명
YEAR	년
QUARTER	분기
MONTH	월
DAYOFYEAR	일
DAY	요일
WEEK	주
WEEKDAY	평일
HOUR	시
MINUTE	분
SECOND	초
MILLISECOND	밀리세컨드
MICROSECOND	마이크로세컨드

사용 예

```
gSQL> SELECT
    DATEADD( YEAR, 1, TO_DATE( '2013-05-14', 'YYYY-MM-DD' ) ) AS RESULT
FROM DUAL;

RESULT
-----
2014-05-14
```

1 row selected.

gSQL> SELECT

DATEADD(MONTH, 13, TO_DATE('2013-05-14', 'YYYY-MM-DD')) AS RESULT

FROM DUAL;

RESULT

2014-06-14

1 row selected.

gSQL> SELECT

DATEADD(DAY, 397, TO_DATE('2013-05-14', 'YYYY-MM-DD')) AS RESULT

FROM DUAL;

RESULT

2014-06-15

1 row selected.

DATEDIFF

구문

DATEDIFF(datepart, startdate, enddate)

설명

enddate에서 startdate를 뺀 값을 지정된 datepart로 반환한다.

startdate 또는 enddate가 NULL이면 결과값도 NULL이다.

startdate와 enddate에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME TYPE이 올 수 있다.

결과 타입은 NUMBER이다.

표 11-58 datepart에 사용 가능한 형식 문자열

datepart	설명
YEAR	년
QUARTER	분기
MONTH	월
DAYOFYEAR	일
DAY	요일
HOUR	시
MINUTE	분
SECOND	초
MILLISECOND	밀리세컨트
MICROSECOND	마이크로세컨드

사용 예

```
gSQL> SELECT
    DATEDIFF( YEAR,
              TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
              TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
FROM DUAL;

RESULT
-----
      1
1 row selected.
gSQL> SELECT
```

```
        DATEDIFF( MONTH,
                TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
                TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
    FROM DUAL;
RESULT
-----
      13
1 row selected.
gSQL> SELECT
        DATEDIFF( DAY,
                TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
                TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
    FROM DUAL;
RESULT
-----
     397
1 row selected.
```

DATE_ADD

구문

```
DATE_ADD( date, INTERVAL expr unit )
```

설명

ADDDATE(date, INTERVAL expr unit)와 동일한 함수이다.

사용 예

```
gSQL> SELECT
      DATE_ADD( TO_DATE( '2012-01-02', 'YYYY-MM-DD' ),
                INTERVAL '2-2' YEAR TO MONTH ) AS RESULT
    FROM DUAL;

RESULT
-----
2014-03-02
1 row selected.
```

DATE_PART

구문

```
DATE_PART( field, datetime )
```

설명

DATE_PART는 EXTRACT 함수와 결과값이 같은 함수로써 입력한 datetime 타입에서 지정된 field를 찾아 반환한다.

인자 field에는 문자 literal만 올 수 있으며, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, TIMEZONE_MINUTE를 문자 literal로 지정할 수 있다.

인자 datetime에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL 타입이 올 수 있다.

field가 datetime의 범위에 있지 않는 경우 에러를 반환한다.

또한, DATE 타입인 경우에는 field에 YEAR, MONTH, DAY 만 올 수 있고, 그 외에는 에러를 반환한다.

반환되는 타입은 NUMBER이다.

자세한 내용은 EXTRACT를 참조한다.

사용 예

```
gSQL> SELECT
    DATE_PART( 'DAY', TO_DATE( '2012-01-02', 'YYYY-MM-DD' ) ) AS RESULT
FROM DUAL;

RESULT
-----
      2
1 row selected.

gSQL> SELECT
    DATE_PART( 'YEAR', INTERVAL'9-11'YEAR TO MONTH ) AS RESULT
FROM DUAL;

RESULT
-----
      9
1 row selected.
```

DECODE

구문

```
DECODE( expr, comparison_expr1, result1
        [, comparison_expr2, result2
        , ...
        , comparison_exprN, resultN ]
        [, default ] )
```

설명

DECODE 문에 기술된 순서대로 expr과 comparison_expr를 equal 연산으로 평가한다.

비교 결과가 FALSE이면 TRUE가 나올 때까지 평가한다.

비교 결과가 TRUE면 대응되는 result를 반환하고 이후는 평가하지 않는다.

expr과 comparison_expr이 같거나, expr과 comparison_expr이 모두 null인 경우, (null = null)에는 TRUE로 평가되어 대응되는 result를 반환한다.

평가한 결과가 모두 FALSE인 경우에는 default를 반환하고, default가 생략된 경우, NULL을 반환한다.

- expr과 comparison_expr 비교

모든 expr, comparison_expr1, ..., comparison_exprN은 comparison_expr1 (첫 번째 comparison_expr)의 데이터 타입으로 변환하여 비교한다.

comparison_expr1 (첫 번째 comparison_expr)이 문자형인 경우와 숫자형인 경우, 각각 expr, comparison_expr1, ..., comparison_exprN에 기술된 타입들의 범위를 포함할 수 있는 타입이 된다.

expr, comparison_expr1, ..., comparison_exprN에 기술된 타입이 모두 CHAR 타입인 경우, VARCHAR 타입으로 비교가 수행된다.

- 결과 타입

결과 타입은 result1 (첫 번째 result)의 데이터 타입이 된다.

result1 (첫 번째 result)의 데이터 타입이 숫자형인 경우와 문자형인 경우는 각각 result1, ..., resultN에 기술된 타입들의 범위를 포함할 수 있는 타입이 된다.

result1 (첫 번째 result)가 CHAR 타입 또는 NULL인 경우, 결과 타입은 VARCHAR가 된다.

DECODE는 CASE를 사용해 동일하게 표현할 수 있다.

- DECODE(expr, comp_expr1, res1, comp_expr2, res2)

```
CASE WHEN (expr = comp_expr1) OR (expr IS NULL AND comp_expr1 IS NULL ) THEN res1
      WHEN (expr = comp_expr2) OR (expr IS NULL AND comp_expr2 IS NULL ) THEN res2
      ELSE NULL
END
```

- `DECODE( expr, comp_expr1, res1, comp_expr2, res2, default )`

```

CASE WHEN (expr = comp_expr1) OR (expr IS NULL AND comp_expr1 IS NULL ) THEN res1
      WHEN (expr = comp_expr2) OR (expr IS NULL AND comp_expr2 IS NULL ) THEN res2
      ELSE default
END

```

사용 예

```

gSQL> SELECT I1,
           DECODE( I1, 1, 'ONE',
                  2, 'TWO',
                  NULL, 'NULL VALUE',
                  'DEFAULT VALUE' ) AS DECODE_RESULT
      FROM T1;
I1 DECODE_RESULT
----
1 ONE
2 TWO
null NULL VALUE
3 DEFAULT VALUE
4 rows selected.

```

DEGREES

구문

DEGREES(radians)

설명

라디안 단위로 표시된 각도 radians를 도 단위로 변환한 값을 반환한다.

사용 예

```
gSQL> SELECT DEGREES( PI() ) AS RESULT FROM DUAL;  
RESULT  
-----  
      180  
1 row selected.
```

DIGEST

구문

DIGEST(data, type)

설명

data를 주어진 type으로 hash한 결과값을 VARBINARY 타입으로 반환한다.

data 타입을 다음 규칙에 따라 입력할 때 implicit conversion이 발생할 수 있다.

- BINARY, VARBINARY 타입 data는 VARBINARY로 입력한다.
- LONG VARBINARY 타입 data는 LONG VARBINARY로 입력한다.
- LONG VARCHAR 타입 data는 LONG VARCHAR로 입력한다.
- 그 외의 모든 타입 data는 모두 VARCHAR 타입으로 implicit conversion하여 입력한다.

DIGEST 함수가 지원하는 hash type은 다음과 같다.

- 'SHA1'의 결과는 20 byte varbinary이다.
- 'SHA224'의 결과는 28 byte varbinary이다.
- 'SHA256'의 결과는 32 byte varbinary이다.
- 'SHA384'의 결과는 48 byte varbinary 이다.
- 'SHA512'의 결과는 64 byte varbinary 이다.

결과를 VARBINARY 타입으로 반환하기 때문에 결과를 hexadecimal 문자로 보려면 HEX 함수를 씌워서 사용해야 하는데 이 경우 그 길이는 원본의 두 배가 된다.

사용 예

```
gSQL> SELECT HEX( DIGEST( 'my password', 'SHA256' ) ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----  
BB14292D91C6D0920A5536BB41F3A50F66351B7B9D94C804DFCE8A96CA1051F2
```

```
1 row selected.
```


DUMP

구문

DUMP(expr)

설명

DUMP 함수는 expr의 내부 표현정보를 반환한다.

내부 표현정보는 데이터 타입, 길이 (byte length), 데이터 정보로 보여준다.

expr에는 모든 타입이 올 수 있으며, 반환되는 타입은 CHARACTER VARYING이다.

사용 예

```
gSQL> SELECT DUMP( 'DUMP' ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
Type=CHAR Len=4 : Str=68,85,77,80
```

```
1 row selected.
```

EXP

구문

EXP(num)

설명

EXP 함수는 e (자연로그 베이스)의 num의 제곱값을 반환한다.

사용 예

```
gSQL> SELECT EXP( 1 ) AS RESULT FROM DUAL;  
          RESULT  
-----  
2.71828182845905  
1 row selected.
```

EXTRACT

구문

```
EXTRACT( <field> FROM datetime )
<field> ::=
    YEAR
    | MONTH
    | DAY
    | HOUR
    | MINUTE
    | SECOND
    | TIMEZONE_HOUR
    | TIMEZONE_MINUTE
```

설명

EXTRACT는 입력한 datetime 타입에서 지정된 field를 찾아 반환한다.

인자 datetime에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL 타입이 올 수 있다.

field가 datetime의 범위에 있지 않는 경우에는 에러를 반환한다.

또, DATE 타입인 경우에는 field에 YEAR, MONTH, DAY만 올 수 있고, 그 외의 경우에는 에러를 반환한다. 반환되는 타입은 NUMBER이다.

EXTRACT 함수의 결과는 **DATE_PART**와 동일하다.

사용 예

```
gSQL> SELECT
    EXTRACT( SECOND FROM TO_TIMESTAMP( '2012-12-13 01:23:44.5',
                                         'YYYY-MM-DD HH24:MI:SS.FF1' ) )
    AS RESULT
FROM DUAL;
RESULT
-----
    44.5
1 row selected.
gSQL> SELECT
    EXTRACT( YEAR FROM CAST('2-3' AS INTERVAL YEAR TO MONTH) ) AS RESULT
```

```
FROM DUAL;  
RESULT  
-----  
2  
1 row selected.
```

FACTORIAL

구문

FACTORIAL(num)

설명

FACTORIAL 함수는 1 ~ num 까지의 연속된 자연수를 차례로 곱한 값을 반환한다.

사용 예

```
gSQL> SELECT FACTORIAL( 5 ) AS RESULT FROM DUAL;  
RESULT  
-----  
      120  
1 row selected.
```

FLOOR

구문

```
FLOOR( num )
```

설명

FLOOR 함수는 num 보다 크지 않은 가장 큰 정수를 반환한다.

사용 예

```
gSQL> SELECT FLOOR(42.8) AS RESULT1, FLOOR(-42.8) AS RESULT2 FROM DUAL;  
RESULT1 RESULT2  
-----  
42      -43  
1 row selected.
```


GREATEST

구문

```
GREATEST( expr1 [, expr2, ... exprn ] )
```

설명

GREATEST 함수는 인자로 받은 expr들 중에 가장 큰 값을 반환한다.

인자로 받은 expr들 중 하나라도 NULL인 경우, 결과값은 NULL이다.

결과 타입은 expr1 (첫 번째 expr)의 데이터 타입이다.

expr1 (첫 번째 expr)의 데이터 타입이 숫자형인 경우와 문자형인 경우는 각각 expr1, ..., exprN의 범위를 포함할 수 있는 타입으로 결정된다.

expr1, ..., exprN에 모두 CHAR타입이 기술된 경우, 모든 expr은 VARCHAR 타입으로 비교되고, 결과 타입은 VARCHAR 타입으로 결정된다.

사용 예

```
gSQL> SELECT GREATEST( 100, 0, 200, 150, 1 ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
200
```

```
1 row selected.
```


HEX

구문

HEX(str)

설명

인자 str을 16진수 문자로 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입 또는 문자 타입으로 변환될 수 있는 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

결과 타입은 CHARACTER VARYING 또는 CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이다.

str이 NULL이면 결과값도 NULL이다.

HEX 함수의 인자로 숫자타입이 오는 경우에는 에러를 반환한다.

10진수 숫자를 16진수로 변환하고자 하는 경우에는 'X' number format을 이용한 TO_CHAR() 함수를 사용할 수 있다.

예: TO_CHAR(255, 'XX')

자세한 내용은 UNHEX를 참조한다.

사용 예

```
gSQL> SELECT HEX( 'abc' ) FROM DUAL;
HEX( 'abc' )
-----
616263
1 row selected.
```

INITCAP

구문

INITCAP(str)

설명

INITCAP 함수는 주어진 문자열 str의 각 단어들의 첫 번째 문자를 대문자로 변환하고 첫 번째 문자 이후의 문자를 소문자로 변환하여 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입이 올 수 있다.

문자열의 각 단어는 white space, 알파벳 또는 숫자가 아닌 문자로 구분한다.

str이 NULL이면 결과값도 NULL이다.

인자로 받는 str의 타입과 동일한 타입이 반환된다.

사용 예

```
gSQL> SELECT INITCAP( 'hi GLIESE' ) AS RESULT FROM DUAL;  
RESULT  
-----  
Hi Gliese  
1 row selected.
```

INSTR

구문

```
INSTR( str, substr [, position [, occurrence ] ] )
```

설명

INSTR 함수는 str의 position부터 시작해서 occurrence 번째의 substr을 찾아 그 위치를 반환한다.

인자 str, substr에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

인자 position, occurrence에는 숫자 타입이 올 수 있다.

position과 occurrence가 생략된 경우, default 값은 1이다.

position과 occurrence는 1부터 시작하며, character set에 따른 문자 단위로 계산된다. (byte 단위가 아님)

position은 str에서 substr 검색을 시작할 처음 위치를 의미하며, 0이 아닌 정수값이어야 한다.

- 양수인 경우: str의 앞부분에서부터 오른쪽으로 계속 비교해나가면서 substr을 찾을 때까지 position 위치를 검색한다.
- 음수인 경우: str의 뒷부분에서부터 왼쪽으로 계속 비교해나가면서 substr을 찾을 때까지 position 위치를 검색한다.
- 0인 경우: 결과값은 0이다.

occurrence는 str에서 substr이 반복된 횟수를 의미하며, 양의 정수이어야 한다.

사용 예

```
gSQL> SELECT INSTR( 'INSTR( STR, SUBSTR )', 'SUB' ) AS RESULT1,
                INSTR( 'INSTR( STR, SUBSTR )', 'SUB', 5 ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
-----
      13      13
1 row selected.
gSQL> SELECT INSTR( 'INSTR( STR, SUBSTR )', 'STR', 6, 2 ) AS RESULT1,
                INSTR( 'INSTR( STR, SUBSTR )', 'STR', -6, 2 ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
-----
```

	16	3
	1 row selected.	

LAST_DAY

구문

LAST_DAY(date)

설명

LAST_DAY 함수는 date에 포함된 월의 마지막 날짜를 반환한다.

인자 date에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있다.
반환되는 타입은 인자 date의 타입에 상관없이 항상 DATE이다.

사용 예

```
gSQL> SELECT
      LAST_DAY( TO_DATE( '2012-07-10', 'YYYY-MM-DD' ) ) AS RESULT FROM DUAL;
RESULT
-----
2012-07-31
1 row selected.
```

LAST_IDENTITY_VALUE

구문

```
LAST_IDENTITY_VALUE()
```

설명

현재 session에서 identity column을 위해 자동으로 생성한 최근 값으로써 결과 타입은 NATIVE_BIGINT 이다.

자동으로 생성한 값이 없을 경우 null을 반환한다.

MS-SQL의 @@IDENTITY, MySQL의 LAST_INSERT_ID()와 유사한 기능으로, 다수의 테이블에 대해 DML을 수행할 경우 다음과 같이 마지막으로 변경한 테이블에 의해 값이 결정되므로 주의하여 사용하여야 한다.

```
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' );
1 row created.
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
-----
12

1 row selected.
gSQL> INSERT INTO t2(name) VALUES ( 'leekmo' );
1 row created.
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
-----
2

1 row selected.
```

INSERT 할 때 생성된 identity column 값을 얻으려면 다음과 같이 **INSERT INTO name RETURNING .. INTO** 구문을 사용한다.

```
gSQL> CREATE TABLE t1 ( id INTEGER GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(32) );
Table created.
gSQL> \var v1 integer
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' ) RETURN id INTO :v1;
V1
--
1
1 row created.
```

사용 예

다음은 LAST_IDENTITY_VALUE() 함수를 사용하는 예이다.

```
gSQL> CREATE TABLE t1 ( id  INTEGER GENERATED BY DEFAULT AS IDENTITY,
                        name VARCHAR(32) );
Table created.
gSQL> COMMIT;
Commit complete.
```

- 현재 session에서 생성한 identity value가 없다.

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
-----
                        null
1 row selected.
```

- identity value (1)를 자동으로 생성한다.

```
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' );
1 row created.
```

- Result: 1

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
-----
                        1
1 row selected.
```

- Default 값으로 identity value (2)가 자동 생성된다.

```
gSQL> UPDATE t1 SET id = DEFAULT;
1 row updated.
```

- Result: 2

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
-----
                        2
1 row selected.
```

- 사용자가 입력한 값으로 identity value가 자동으로 생성되지 않는다.

```
INSERT INTO t1 VALUES ( 100, 'jhkim' );
```

1 row updated.

- Result: 2

```
SELECT LAST_IDENTITY_VALUE() FROM dual;
```

```
LAST_IDENTITY_VALUE()
```

```
-----
```

```
2
```

1 row selected.

LEAST

구문

```
LEAST( expr1 [, expr2, ... exprn ] )
```

설명

LEAST 함수는 인자로 받은 expr들 중에 가장 작은 값을 반환한다.

인자로 받은 expr들 중 하나라도 NULL인 경우, 결과값은 NULL이다.

결과 타입은 expr1 (첫 번째 expr)의 데이터 타입에 따라 결정된다.

expr1 (첫 번째 expr)의 데이터 타입이 숫자형인 경우와 문자형인 경우에는 각각 expr1, ..., exprN의 범위를 포함할 수 있는 타입으로 결정된다.

expr1, ..., exprN에 모두 CHAR 타입이 기술된 경우, 모든 expr은 VARCHAR 타입으로 비교되고, 결과 타입은 VARCHAR 타입이 된다.

사용 예

```
gSQL> SELECT LEAST( 100, 0, 200, 150, 1 ) AS RESULT FROM DUAL;
RESULT
-----
      0
1 row selected.
```

LENGTH

구문

```
LENGTH( str )
```

설명

CHAR_LENGTH의 alias 이다.

사용 예

Multi byte character set: (예: UTF8)

```
gSQL> SELECT LENGTH( 'αβ-SUMMER' ) AS RESULT FROM DUAL;  
RESULT  
-----  
          9  
1 row selected.
```

LENGTHB

구문

LENGTHB(str)

설명

OCTET_LENGTH의 alias 이다.

자세한 내용은 OCTET_LENGTH, BYTE_LENGTH를 참조한다.

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT LENGTHB( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
      FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----
                        12
1 row selected.
```

- Multi byte character set (예: UTF8): 2 byte character

```
gSQL> SELECT LENGTHB( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
-----
                        4
1 row selected.
```

LN

구문

LN(num)

설명

LN 함수는 num의 자연 로그 값을 반환하는 함수이다.
num은 0보다 큰 값이어야 한다.

사용 예

```
gSQL> SELECT LN( 2.71828182845905 ) AS RESULT FROM DUAL;  
RESULT  
-----  
1  
1 row selected.
```

LOCALTIME

구문

```
LOCALTIME [()]
STATEMENT_LOCALTIME()
```

설명

Session 시간을 기준으로 현재 TIME WITHOUT TIME ZONE type 값을 얻는다.

LOCALTIME은 SQL 표준 함수이다.

현재 시간을 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- LOCALTIME, STATEMENT_LOCALTIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_LOCALTIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

각 row의 시간값이 모두 같다.

```
gSQL> SELECT LOCALTIME FROM t1;
LOCALTIME
-----
16:17:08.592459
16:17:08.592459
16:17:08.592459
3 rows selected.
```

LOCALTIMESTAMP

구문

```
LOCALTIMESTAMP [()]  
STATEMENT_LOCALTIMESTAMP()
```

설명

Session 시간을 기준으로 현재 TIMESTAMP WITHOUT TIME ZONE type 값을 얻는다.

LOCALTIMESTAMP는 SQL 표준 함수이다.

현재 TIMESTAMP를 얻는 함수들 간에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- LOCALTIMESTAMP, STATEMENT_LOCALTIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_LOCALTIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT LOCALTIMESTAMP FROM t1;  
LOCALTIMESTAMP  
-----  
2013-12-12 16:21:51.790614  
2013-12-12 16:21:51.790614  
2013-12-12 16:21:51.790614  
3 rows selected.
```

LOCAL_GROUP_ID

구문

`LOCAL_GROUP_ID()`

설명

사용자 질의를 받아 처리하는 server에 대한 cluster group ID를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT LOCAL_GROUP_ID() FROM DUAL;  
LOCAL_GROUP_ID()  
-----  
1  
1 row selected.
```

LOCAL_GROUP_NAME

구문

LOCAL_GROUP_NAME()

설명

사용자 질의를 받아 처리하는 server에 대한 cluster group name을 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT LOCAL_GROUP_NAME() FROM DUAL;  
LOCAL_GROUP_NAME()  
-----  
G1  
1 row selected.
```


LOCAL_MEMBER_ID

구문

LOCAL_MEMBER_ID()

설명

사용자 질의를 받아 처리하는 server에 대한 cluster member ID를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT LOCAL_MEMBER_ID() FROM DUAL;  
LOCAL_MEMBER_ID()  
-----  
1  
1 row selected.
```

LOCAL_MEMBER_NAME

구문

`LOCAL_MEMBER_NAME()`

설명

사용자 질의를 받아 처리하는 server에 대한 cluster member name을 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT LOCAL_MEMBER_NAME() FROM DUAL;
```

```
LOCAL_MEMBER_NAME()
```

```
-----
```

```
G1N1
```

```
1 row selected.
```

LOG

구문

```
LOG( num2 )  
LOG( num1, num2 )
```

설명

LOG 함수는 밑이 num1인 num2의 로그값을 반환한다.
num1이 생략된 경우에는 밑이 10으로 계산된 값이 반환된다.
num1은 1과 0이 아닌 양수이어야 하고, num2는 양수이어야 한다.

사용 예

```
gSQL> SELECT LOG( 100 ) AS RESULT1, LOG( 4, 16 ) AS RESULT2 FROM DUAL;  
RESULT1 RESULT2  
-----  
2        2  
1 row selected.
```

LOGON_USER

구문

LOGON_USER()

설명

로그인 사용자를 반환한다.

사용자 정보는 다음과 같은 세 가지 형태로 관리된다

- Logon user: login을 실행한 user로서 connection을 닫을 때까지 유지된다.
- Session user: 최초 logon user와 동일하지만 SET SESSION AUTHORIZATION 구문을 이용해 변경가능하다.
- Current user: 일반적으로 session user와 동일하지만 PSM, view 등을 사용할 때 접근 제어등을 위해 시스템 내부적으로 잠시 변경된다.
 - Session user와 current user는 unix system의 real user와 effective user의 차이와 유사한 개념이다.

사용 예

```
% gsql test test
gSQL> SELECT LOGON_USER() AS result FROM DUAL;
RESULT
-----
TEST
1 row selected.
```

LOWER

구문

LOWER(str)

설명

LOWER 함수는 str의 소문자를 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입이 올 수 있다.

str이 NULL이면 결과값도 NULL이다.

인자 str과 동일한 타입이 반환된다.

사용 예

```
gSQL> SELECT LOWER( 'SPRING' ) AS RESULT FROM DUAL;  
RESULT  
-----  
spring  
1 row selected.
```

LPAD

구문

LPAD(str, length, [, fill])

설명

LPAD 함수는 string의 길이가 length가 될 때까지 str의 왼쪽에 문자열 fill을 추가한 값을 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

인자 length에는 숫자 타입이 올 수 있다.

length는 문자의 개수를 의미하며, 최대 범위는 결과 타입의 최대 PRECISION이다.

fill이 생략된 경우, 공백 문자가 추가된다.

str이 length보다 길이가 긴 경우에는 str을 length만큼 잘라서 반환한다.

str, length, fill이 하나라도 NULL이면 결과값도 NULL이며, length가 0 또는 음수인 경우에도 결과값은 NULL이다.

결과 타입은 다음 표와 같다.

표 11-59 LPAD의 결과 타입

인자 str의 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT LPAD( 'aa', 5, 'b' ) AS RESULT FROM DUAL;  
RESULT  
-----  
bbbaa  
1 row selected.
```

LTRIM

구문

LTRIM(trim_source [, trim_character])

설명

LTRIM 함수는 trim_source에서 trim_character를 왼쪽 방향에서 비교하여 일치하는 문자가 없을 때까지 제거한 결과를 반환한다.

인자 trim_character와 trim_source에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

trim_character, trim_source 중 하나라도 NULL이면, 결과값은 NULL이다.
trim_character가 생략된 경우에는 기본적으로 single blank space (' ')가 지정된다.

결과 타입은 다음과 같다.

표 11-60 LTRIM의 결과 타입

trim_source, trim_character 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT LTRIM( '____LTRIM', '_' ) AS RESULT FROM DUAL;
RESULT
-----
LTRIM
1 row selected.
```

MAX

구문

```
MAX( [ ALL | DISTINCT ] expr )
```

설명

Aggregation 함수로써 row들의 expr 중 최대값을 얻는다.

ALL을 명시한 경우, 모든 값들에 대해 aggregation을 수행한다.

DISTINCT를 명시한 경우, 중복을 제거한 값들에 대해 aggregation을 수행한다.

ALL이나 DISTINCT를 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

MAX는 ALL과 DISTINCT에 영향을 받지 않고 동일한 결과를 낸다.

사용 예

```
gSQL> SELECT MAX(c1) FROM t1;
MAX(C1)
-----
      3
1 row selected.
```


MIN

구문

```
MIN( [ ALL | DISTINCT ] expr )
```

설명

Aggregation 함수로써 row들의 expr 중 최소값을 얻는다.

ALL을 명시한 경우, 모든 값들에 대해 aggregation을 수행한다.

DISTINCT를 명시한 경우, 중복 제거한 값들에 대해 aggregation을 수행한다.

ALL이나 DISTINCT를 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

MIN은 ALL과 DISTINCT에 영향을 받지 않고 동일한 결과를 낸다.

사용 예

```
gSQL> SELECT MIN(c1) FROM t1;
```

```
MIN(C1)
```

```
-----
```

```
1
```

```
1 row selected.
```

MOD

구문

```
MOD( num1, num2 )
```

설명

MOD는 num1을 num2로 나눈 나머지를 반환한다.

인자 num1, num2에는 숫자타입이 올 수 있다.

num2가 0이면 에러를 반환한다.

사용 예

```
gSQL> SELECT MOD(5, 4) AS RESULT1, MOD(-5, 4) AS RESULT2 FROM DUAL;
```

```
RESULT1 RESULT2
```

```
-----
```

```
1      -1
```

```
1 row selected.
```

MONTHS_BETWEEN

구문

```
MONTHS_BETWEEN( date1, date2 )
```

설명

MONTHS_BETWEEN은 date2와 date1 사이의 일수를 31로 나눈 개월 수를 반환한다.

date1 또는 date2가 NULL이면 결과도 NULL이다.

인자 date1, date2에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있다.

결과 타입은 NUMBER 이다.

노트

date1과 date2 모두에 동일한 날짜가 포함되어 있거나 (예: 2014-01-15 와 2014-02-15) 월의 마지막 날짜가 포함되어 있는 경우 (예: 2014-08-31와 2014-09-30), 타임스탬프 구간 (있는 경우)의 일치 여부와 상관없이 정수 결과를 반환한다.

사용 예

```
gSQL> SELECT
      MONTHS_BETWEEN('2018-01-18', '2018-01-17')
    FROM DUAL;
MONTHS_BETWEEN('2018-01-18', '2018-01-17')
-----
                        3.225806451612903E-2
1 row selected.
gSQL> SELECT
      MONTHS_BETWEEN('2018-02-17', '2018-01-17')
    FROM DUAL;
MONTHS_BETWEEN('2018-02-17', '2018-01-17')
-----
                        1
1 row selected.
gSQL> SELECT
      MONTHS_BETWEEN('2018-02-28', '2018-01-31')
```

```
FROM DUAL;  
MONTHS_BETWEEN('2018-02-28', '2018-01-31')
```

1

1 row selected.

NEXT_DAY

구문

NEXT_DAY(date, day)

설명

인자로 주어진 date (날짜)를 지나 처음으로 도래하는 day (요일)의 날짜를 구한다.

두 번째 인자 day에는 day를 지칭하는 스트링 또는 숫자가 올 수 있다.

- 스트링: SUNDAY ~ SATURDAY 또는 SUN ~ SAT
- 숫자: 1 (sunday) ~ 7 (saturday)

반환되는 타입은 date의 입력 타입에 상관없이 항상 DATE 타입이다.

결과값의 시분초는 입력 인자 date의 시분초를 동일하게 반환한다.

사용 예

```
gSQL> SELECT NEXT_DAY( TO_DATE( '2010-05-01', 'YYYY-MM-DD' ),
                        'SUNDAY' ) AS RESULT1
FROM DUAL;
```

RESULT1

2010-05-02

```
gSQL> SELECT NEXT_DAY( TO_DATE( '2010-05-01', 'YYYY-MM-DD' ),
                        'SUN' ) AS RESULT1
FROM DUAL;
```

RESULT1

2010-05-02

```
gSQL> SELECT NEXT_DAY( TO_DATE( '2010-05-01', 'YYYY-MM-DD' ),
                        1 ) AS RESULT1
FROM DUAL;
```

RESULT1

2010-05-02

[illegible]

FROM DUAL;
RESULT2

2010-05-02 00:00:00

NEXTVAL

구문

```
seq_name.NEXTVAL
NEXTVAL( seq_name )
NEXT VALUE FOR seq_name
```

설명

시퀀스 객체의 다음 값을 얻는다.

사용 예

```
gSQL> CREATE SEQUENCE seq;
Sequence created.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT seq.NEXTVAL FROM dual;
SEQ.NEXTVAL
-----
          1
1 row selected.
gSQL> SELECT NEXTVAL( seq ) FROM dual;
NEXTVAL( SEQ )
-----
          2
1 row selected.
gSQL> SELECT NEXT VALUE FOR seq FROM dual;
NEXT VALUE FOR SEQ
-----
          3
1 row selected.
```

NULLIF

구문

```
NULLIF( expr1, expr2 )
```

설명

expr1과 expr2가 같으면 null을 반환하고, 같지 않으면 첫 번째 인자인 expr1을 반환한다.

expr1과 expr2의 타입이 다를 경우, **결과 타입 조합 규칙**에 따라 result type이 결정된다.

NULLIF는 CASE를 사용하여 동일하게 표현할 수 있다.

- NULLIF(expr1, expr2)

```
CASE WHEN expr1 = expr2 THEN NULL
      ELSE expr1
END
```

사용 예

```
gSQL> SELECT NULLIF( 'SUN', 'SUN' ) AS RESULT1,
              NULLIF( 'SUN', 'MOON' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
-----
null    SUN
1 row selected.
```


NVL

구문

```
NVL( expr1, expr2 )
```

설명

expr1이 null이 아니면 expr1을 반환하고, expr1이 null이면 expr2를 반환한다.

결과 타입은 expr1의 데이터 타입에 따라 결정된다.

expr1에 NULL이 기술된 경우에는 expr2의 타입에 따라 결과 타입이 결정된다.

expr1의 데이터 타입이 숫자형 타입인 경우와 문자형 타입인 경우는 각각 expr1, expr2의 범위를 포함할 수 있는 타입으로 결정된다.

expr1, expr2의 데이터 타입이 모두 CHAR 타입인 경우, 결과 타입은 VARCHAR로 결정된다.

사용 예

```
gSQL> SELECT I1, NVL( I1, 0 ) FROM T1;
  I1 NVL( I1, 0 )
----
  1             1
null            0
2 rows selected.
```

NVL2

구문

```
NVL2( expr1, expr2, expr3 )
```

설명

expr1이 null이 아니면 expr2를 반환하고, expr1이 null이면 expr3을 반환한다.

결과 타입은 expr2의 데이터 타입에 따라 결정된다.

expr2에 NULL이 기술된 경우에는 expr3의 타입에 따라 결과 타입이 결정된다.

expr2의 데이터 타입이 숫자형인 경우와 문자형인 경우에는 각각 expr2, expr3의 범위를 포함할 수 있는 타입으로 결정된다.

expr2, expr3의 데이터 타입이 모두 CHAR 타입인 경우, 결과 타입은 VARCHAR가 된다.

사용 예

```
gSQL> SELECT I1, NVL2( I1, I1 * 1000, 0 ) FROM T1;
      I1 NVL2( I1, I1 * 1000, 0 )
-----
      1                1000
    null                0
2 rows selected.
```

OCTET_LENGTH

구문

```
OCTET_LENGTH( str )
BYTE_LENGTH( str )
LENGTHB( str )
```

설명

OCTET_LENGTH는 str의 바이트 수를 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

str의 타입이 CHARACTER면 공백문자도 계산에 포함된다.

str이 NULL이면 결과값도 NULL이다.

OCTET_LENGTH의 alias로는 **BYTE_LENGTH**와 **LENGTHB** 함수가 있다.

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT OCTET_LENGTH( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
        FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----
                        12
1 row selected.
```

- Multi byte character set (예: UTF8): 2 byte character

```
gSQL> SELECT OCTET_LENGTH( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
-----
                        4
1 row selected.
```

OVERLAY

구문

OVERLAY(str1 PLACING str2 FROM start_position [FOR string_length])

설명

OVERLAY 함수는 str1의 start_position부터 string_length까지의 문자를 str2로 치환한 결과를 반환한다.

인자 str1과 str2에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARCATER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다. 인자 start_position과 string_length에는 숫자 타입이 올 수 있다.

- OVERLAY 결과는 다음과 같다.
 - FOR가 지정된 경우
SUBSTRING(str1 FROM 1 FOR (start_position - 1))
|| str2
|| SUBSTRING(str1 FROM (start_position + string_length)
 - FOR가 생략된 경우
SUBSTRING(str1 FROM 1 FOR (start_position - 1))
|| str2
|| SUBSTRING(str1 FROM (start_position + CHAR_LENGTH(str2))

자세한 내용은 **SUBSTRING**을 참조한다.

결과 타입은 다음 표와 같다.

표 11-61 OVERLAY의 결과 타입

str1, str2 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT
    OVERLAY( 'RESULT_OF_XXX_FUNC' PLACING 'OVERLAY' FROM 11 FOR 3 )
    AS RESULT
```

```
FROM DUAL;  
RESULT  
-----  
RESULT_OF_OVERLAY_FUNC  
1 row selected.
```

PI

구문

PI()

설명

PI는 " π " constant를 반환한다.

사용 예

```
gSQL> SELECT PI() AS RESULT FROM DUAL;  
          RESULT  
-----  
3.141592653589793E+0  
1 row selected.
```

POSITION

구문

```
POSITION( str1 IN str2 )
```

설명

POSITION 함수는 str2에서 첫 번째 str1을 찾아 그 위치를 반환하는 함수이다.

str1과 str2에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

str2에서 str1을 찾을 수 없는 경우, 리턴값 0이 반환된다.

str2에서 str1을 찾은 경우, 1을 시작으로 그 찾은 위치를 반환한다.

반환되는 위치값은 CHARACTER 단위로 계산된 값이다. (byte 단위가 아님)

str1 또는 str2가 NULL이면, 반환되는 값도 NULL 이다.

사용 예

```
gSQL> SELECT POSITION( 'CHAR' IN 'LONG CHAR 2000' ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
6
```

```
1 row selected.
```

POWER

구문

```
POWER( num1, num2 )
```

설명

POWER 함수는 num1에 num2를 제공한 값을 반환한다.

인수 num1과 num2에는 숫자 타입이 올 수 있다.

num1이 음수라면 num2는 정수여야 한다.

num1 또는 num2의 값이 NULL이면, 결과값도 NULL이다.

사용 예

```
gSQL> SELECT POWER( 2, 3 ) AS RESULT FROM DUAL;  
RESULT  
-----  
      8  
1 row selected.
```


RADIANS

구문

RADIANS(degrees)

설명

RADIANS 함수는 degrees의 라디안을 반환한다.
인자 degrees에는 숫자 타입이 올 수 있다.

사용 예

```
gSQL> SELECT RADIANS( 180 ) AS RESULT FROM DUAL;  
          RESULT  
-----  
3.14159265358979  
1 row selected.
```

RANDOM

구문

`RANDOM( min, max )`

설명

RANDOM은 min 이상 max 이하의 random 값을 반환한다.
인자 min, max에는 숫자 타입이 올 수 있다.

사용 예

```
gSQL> SELECT RANDOM( 1, 100 ) AS RESULT FROM DUAL;  
          RESULT  
-----  
34.1870528003201  
1 row selected.
```

REPEAT

구문

REPEAT(str, num)

설명

REPEAT 함수는 num에 지정된 수만큼 str을 반복한 string을 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARCATER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

인자 num에는 숫자 타입이 올 수 있다.

str 또는 num 중의 하나라도 NULL이면, 결과값도 NULL이다.

num의 값이 0 또는 음수인 경우에도 결과값은 NULL이다.

결과 타입은 다음 표와 같다.

표 11-62 REPEAT의 결과 타입

str 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT REPEAT( 'ab', 3 ) AS RESULT FROM DUAL;  
RESULT  
-----  
ababab  
1 row selected.
```

REPLACE

구문

REPLACE(str, from, to)

설명

REPLACE는 str string 내의 모든 from string을 to string으로 치환하여 반환한다.

인자 str, from, to에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

str 값이 NULL인 경우, 결과값은 NULL이다.

from 값이 NULL인 경우, str 값을 변환하지 않고 반환한다.

to 값이 생략되었거나 NULL인 경우, str에서 from을 제거한 값이 반환된다.

결과 타입은 다음 표와 같다.

표 11-63 REPLACE의 결과 타입

str 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

사용 예

```
gSQL> SELECT REPLACE( 'HI GLIESE', 'HI', 'HELLO' ) AS RESULT FROM DUAL;  
RESULT  
-----  
HELLO GLIESE  
1 row selected.
```

REVERSE

구문

REVERSE(str)

설명

REVERSE는 str의 문자를 역순으로 반환한다.

인자로 character string 타입 또는 binary string 타입으로 변환될 수 있는 타입이 올 수 있으며 character string 타입은 해당 문자 단위로, binary string 타입은 byte 단위로 수행된다.

str이 NULL일 경우 NULL을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-64 REVERSE 인자와 결과 타입

str	결과 타입
CHAR	CHAR
VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY	BINARY
VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT REVERSE( 'GOLDILOCKS' ) AS RESULT FROM DUAL;
RESULT
-----
SKCOLIDLOG
1 row selected.
gSQL> SELECT REVERSE( '선재소프트 2018' ) AS RESULT FROM DUAL;
RESULT
-----
8102 트프소재선
1 row selected.
```

ROUND(number)

구문

```
ROUND( num [, scale ] )
```

설명

ROUND는 scale을 기준으로 num을 반올림한 값을 반환한다.

인자 num, scale에는 숫자 타입이 올 수 있다.

scale이 생략된 경우, scale은 0이 되어 ROUND(num, 0)과 같이 수행된다.

scale이 양수인 경우 소수점 오른쪽 자리수를 기준으로 반올림되고, scale이 음수인 경우 소수점 왼쪽 자리수를 기준으로 반올림된다.

사용 예

```
gSQL> SELECT ROUND( 152.4282, 2 ) AS RESULT FROM DUAL;  
RESULT  
-----  
152.43  
1 row selected.  
gSQL> SELECT ROUND( 152.4282, -2 ) AS RESULT FROM DUAL;  
RESULT  
-----  
200  
1 row selected.
```

ROUND(date)

구문

ROUND(date [, fmt])

설명

ROUND(date) 함수는 date를 지정된 fmt 단위로 반올림한 값을 반환한다.

인자 date에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있다.

인자 fmt에는 CHARACTER, CHARACTER VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

결과 타입은 인자로 받은 date 타입과 관계없이 항상 DATE 타입이다.

fmt가 생략되었을 경우의 기본값은 DAY 이며, 사용 가능한 형식문자열은 다음 표와 같다.

표 11-65 fmt에 사용 가능한 형식문자열

문자열	설명
CC, SCC	51년부터 반올림하여 네자리 연도로 표현한다. (예: XX01)
YYYY, YEAR, SYYYY, SYEAR, YY Y, YY, Y	7월 1일부터 반올림한다.
IYYY, IYY, IY, I	ISO 8601 표준에 정의된 calendar week를 수용하는 연도로써 7월 1일부터 반올림한다.
Q	분기의 두 번째 달의 16일부터 반올림한다.
MONTH, MON, MM, RM	16일부터 반올림한다.
WW	연도의 1월 1일부터 한 주가 시작되며 해당 WEEK의 수요일 오후 12시부터 반올림한다.
IW	ISO 8601 표준에 정의된 calendar week (1 ~ 52주 또는 1 ~ 53주)로써 목요일 오후 12시부터 반올림한다.
W	월의 1일을 한 주로 시작해서 해당 WEEK의 수요일 오후 12시부터 반올림한다.
DDD, DD, J	오후 12시부터 반올림한다.
DAY, DY, D	WEEK의 수요일 오후 12시부터 반올림한다.
HH, HH12, HH24	30 분부터 반올림한다.
MI	30 초부터 반올림한다.

사용 예

```
gSQL> SELECT
    ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'CC' ) AS RESULT
FROM DUAL;
```

RESULT

2101-01-01

1 row selected.

gSQL> SELECT

```
ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'YYYY' ) AS RESULT
FROM DUAL;
```

RESULT

2052-01-01

1 row selected.

gSQL> SELECT

```
ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'MONTH' ) AS RESULT
FROM DUAL;
```

RESULT

2051-08-01

1 row selected.

gSQL> SELECT

```
ROUND( TO_TIMESTAMP( '2001-05-05 15:22:33.999999',
                      'YYYY-MM-DD HH24:MI:SS.FF6' ) ) AS RESULT
FROM DUAL;
```

RESULT

2001-05-06

1 row selected.

ROWID_GRID_BLOCK_ID

구문

`ROWID_GRID_BLOCK_ID( rowid )`

설명

GRID block ID를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT C1, ROWID_GRID_BLOCK_ID( ROWID ) FROM T1;
C1 ROWID_GRID_BLOCK_ID( ROWID )
-- -----
1                               52
2                               52
3                               52
3 rows selected.
```

ROWID_GRID_BLOCK_SEQ

구문

```
ROWID_GRID_BLOCK_SEQ( rowid )
```

설명

GRID block sequence를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT C1, ROWID_GRID_BLOCK_SEQ( ROWID ) FROM T1;  
C1 ROWID_GRID_BLOCK_SEQ( ROWID )  
-- -----  
1 747465  
2 747466  
3 747467  
3 rows selected.
```

ROWID_MEMBER_ID

구문

```
ROWID_MEMBER_ID( rowid )
```

설명

Member ID를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT C1, ROWID_MEMBER_ID( ROWID ) FROM T1;
C1 ROWID_MEMBER_ID( ROWID )
-- -----
1 1
2 1
3 1
3 rows selected.
```

ROWID_OBJECT_ID

구문

```
ROWID_OBJECT_ID( rowid )
```

설명

Object ID를 반환하는 함수이다.

노트

Cluster system에서 유효하지 않은 정보이다.

사용 예

```
gSQL> SELECT ROWID_OBJECT_ID( t1.ROWID ) FROM t1;  
ROWID_OBJECT_ID( T1.ROWID )  
-----  
22012  
22012  
22012  
22012  
4 rows selected.
```

ROWID_PAGE_ID

구문

`ROWID_PAGE_ID( rowid )`

설명

Page ID를 반환하는 함수이다.

노트

Cluster system에서 유효하지 않은 정보이다.

사용 예

```
gSQL> SELECT ROWID_PAGE_ID( t1.ROWID ) FROM t1;
ROWID_PAGE_ID( T1.ROWID )
-----
                        8227
                        8227
                        8227
                        8227
4 rows selected.
```

ROWID_ROW_NUMBER

구문

```
ROWID_ROW_NUMBER( rowid )
```

설명

Row number를 반환하는 함수이다.

노트

Cluster system에서 유효하지 않은 정보이다.

사용 예

```
gSQL> SELECT ROWID_ROW_NUMBER( t1.ROWID ) FROM t1;
ROWID_ROW_NUMBER( T1.ROWID )
-----
                                0
                                1
                                2
                                3
4 rows selected.
```

ROWID_SHARD_ID

구문

```
ROWID_SHARD_ID( rowid )
```

설명

Shard ID를 반환하는 함수이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT C1, ROWID_SHARD_ID( ROWID ) FROM T1;  
C1 ROWID_SHARD_ID( ROWID )  
-- -----  
1 0  
2 1  
3 2  
3 rows selected.
```

ROWID_TABLESPACE_ID

구문

```
ROWID_TABLESPACE_ID( rowid )
```

설명

Tablespace ID를 반환하는 함수이다.

노트

Cluster system에서 유효하지 않은 정보이다.

사용 예

```
gSQL> SELECT ROWID_TABLESPACE_ID( t1.ROWID ) FROM t1;
ROWID_TABLESPACE_ID( T1.ROWID )
-----
2
2
2
2
4 rows selected.
```


ROWNUM

구문

ROWNUM

설명

WHERE 조건을 만족하는 row에 1부터 순차적으로 번호를 부여한다.

Oracle과의 호환성을 위해 WHERE 절에 ROWNUM 사용을 허용한다.

그러나 질의 결과 개수를 제한하려면 다음과 같이 SQL 표준의 **offset limit clause**를 사용할 것을 권장한다.

- (비표준) ROWNUM을 이용하여 결과 개수 기술

```
gSQL> SELECT * FROM t1 WHERE ROWNUM <= 3;
C1
--
A
B
C
3 rows selected.
```

- (SQL 표준) FETCH 구문을 이용하여 결과 개수 기술

```
gSQL> SELECT * FROM t1 FETCH 3;
C1
--
A
B
C
3 rows selected.
```

질의 결과의 일부 범위를 제한하고자 할 때도 다음과 같이 OFFSET, FETCH 구문을 사용할 것을 권장한다.

- (비표준) ROWNUM을 이용하여 결과 개수의 범위 기술

```
gSQL> SELECT c1
      FROM ( SELECT ROWNUM rn, c1
            FROM t1 )
      WHERE rn BETWEEN 2 AND 3;
```

```
C1
--
B
C
2 rows selected.
```

- (SQL 표준) OFFSET, FETCH 구문을 이용하여 결과 개수의 범위 기술

```
gSQL> SELECT c1 FROM t1 OFFSET 1 FETCH 2;
C1
--
B
C
2 rows selected.
```

결과 개수 제한 이외의 용도로 ROWNUM을 WHERE 절에 사용하는 것은 권장하지 않는다.

다음 예와 같이 모호한 조건 (WHERE c1 < ROWNUM + 3)을 사용할 경우, 실행 방법에 따라 동일한 질의에 대한 결과가 달라질 수 있다.

- 데이터 생성

```
CREATE TABLE t1 ( c1 INTEGER );
CREATE INDEX t1_idx ON t1(c1);
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (2);
INSERT INTO t1 VALUES (3);
INSERT INTO t1 VALUES (4);
INSERT INTO t1 VALUES (5);
COMMIT;
```

- Oracle의 경우

```
SQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
  ROWNUM      C1
-----
      1         1
      2         2
SQL> DROP INDEX t1_idx;
```

- 인덱스가 삭제되었다.

```
SQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
```

ROWNUM	C1
1	1
2	2
3	3
4	4
5	5

- GOLDDILOCKS의 경우

```
gSQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
```

```
ROWNUM C1
```

```
----- --
1 1
2 2
3 3
4 4
5 5
```

5 rows selected.

```
gSQL> DROP INDEX t1_idx;
```

Index dropped.

```
gSQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
```

```
ROWNUM C1
```

```
----- --
1 1
2 2
3 3
4 4
5 5
```

5 rows selected.

사용 예

```
gSQL> SELECT ROWNUM, c1 FROM t1;
```

```
ROWNUM C1
```

```
----- --
1 A
2 B
3 C
4 D
```

5 E

5 rows selected.

RPAD

구문

```
RPAD( str, length, [, fill] )
```

설명

RPAD 함수는 string의 길이가 length가 될 때까지 str의 오른쪽에 문자열 fill을 추가한 값을 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

인자 length에는 숫자 타입이 올 수 있다.

length는 문자의 개수를 의미하며 최대 범위는 결과 타입의 최대 PRECISION이다.

fill이 생략된 경우, 공백 문자가 추가된다.

str이 length보다 길이가 긴 경우에는 str을 length만큼 잘라서 반환한다.

str, length, fill 중 하나라도 NULL이면 결과값도 NULL이며, length가 0 또는 음수인 경우에도 결과값은 NULL이다.

결과 타입은 다음 표와 같다.

표 11-66 RPAD의 결과 타입

인자 str의 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT RPAD( 'aa', 5, 'b' ) AS RESULT FROM DUAL;
RESULT
-----
aabbb
1 row selected.
```

RTRIM

구문

RTRIM(trim_source [, trim_character])

설명

RTRIM 함수는 trim_source에서 trim_character를 오른쪽 방향에서 비교하여 일치하는 문자가 없을 때까지 제거한 결과를 반환한다.

인자 trim_character와 trim_source에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

trim_character, trim_source 중 하나라도 NULL이면, 결과값은 NULL이다.
trim_character가 생략된 경우에는 기본적으로 single blank space (' ')가 지정된다.

결과 타입은 다음과 같다.

표 11-67 RTRIM의 결과 타입

trim_source, trim_character 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT RTRIM( 'RTRIM_____', ' ' ) AS RESULT FROM DUAL;
RESULT
-----
RTRIM
1 row selected.
```

SESSION_ID

구문

`SESSION_ID()`

설명

현재 session의 ID를 얻는다.

사용 예

```
gSQL> SELECT SESSION_ID() FROM dual;
SESSION_ID()
-----
          4
1 row selected.
```

SESSION_SERIAL

구문

SESSION_SERIAL()

설명

현재 session의 serial 번호를 얻는다.

사용 예

```
gSQL> SELECT SESSION_SERIAL() FROM dual;  
SESSION_SERIAL()  
-----  
16  
1 row selected.
```


SESSION_USER

구문

```
SESSION_USER[()]
```

설명

세션 사용자를 반환한다.

사용자 정보는 다음과 같은 세 가지 형태로 관리된다.

- Logon user: login을 실행한 user로서 connection을 닫을 때까지 유지된다.
- Session user: 최초 logon user와 동일하지만 SET SESSION AUTHORIZATION 구문을 이용해 변경할 수 있다.
- Current user: 일반적으로 session user와 동일하지만 PSM, view 등을 사용할 때 접근 제어등을 위해 시스템 내부적으로 잠시 변경된다.
 - Session user와 current user는 unix system의 real user와 effective user의 차이와 유사한 개념이다.

사용 예

```
% gsql sys gliese
gSQL> SET SESSION AUTHORIZATION test;
Session set.
gSQL> SELECT LOGON_USER() AS result1, SESSION_USER() AS result2 FROM DUAL;
RESULT1 RESULT2
-----
SYS      TEST
1 row selected.
```

SHARD_GROUP_ID

구문

```
SHARD_GROUP_ID( table_name, shard_key_value [ , ... ] )
```

설명

SHARD_GROUP_ID 함수는 table_name에 shard strategy가 정의된 경우, shard_key_value 값을 저장할 수 있는 shard를 관리하는 group ID를 반환한다.

입력 인자 table_name은 identifier로 기술하여야 하며 table_name에 해당하는 객체가 base table이 아니거나 shard strategy가 정의되지 않은 경우 에러가 발생한다.

입력 인자 shard_key_value는 table_name에 정의된 shard strategy의 shard key column 순으로 나열되어야 한다. shard_key_value 개수와 shard key column 개수가 일치하지 않을 경우, 에러가 발생한다.

결과 타입은 NATIVE_BIGINT이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT T1.C1, SHARD_GROUP_ID( T1, T1.C1 ) FROM T1;
```

```
C1 SHARD_GROUP_ID( T1, T1.C1 )
```

```
-- -----
```

```
A 1
```

```
B 2
```

```
C 3
```

```
3 rows selected.
```

```
gSQL> SELECT SHARD_GROUP_ID( T1, 'B' ) FROM DUAL;
```

```
SHARD_GROUP_ID( T1, 'B' )
```

```
-----
```

```
2
```

```
1 row selected.
```

SHARD_GROUP_NAME

구문

```
SHARD_GROUP_NAME( table_name, shard_key_value [ , ... ] )
```

설명

SHARD_GROUP_NAME 함수는 table_name에 shard strategy가 정의된 경우, shard_key_value 값을 저장할 수 있는 shard를 관리하는 group NAME을 반환한다.

입력 인자 table_name은 identifier로 기술하여야 하며 table_name에 해당하는 객체가 base table이 아니거나 shard strategy가 정의되지 않은 경우, 에러가 발생한다.

입력 인자 shard_key_value는 table_name에 정의된 shard strategy의 shard key column 순으로 나열되어야 한다. shard_key_value 개수와 shard key column 개수가 일치하지 않을 경우, 에러가 발생한다.

결과 타입은 VARCHAR이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT T1.C1, SHARD_GROUP_NAME( T1, T1.C1 ) FROM T1;
C1 SHARD_GROUP_NAME( T1, T1.C1 )
-- -----
A  G1
B  G2
C  G3
3 rows selected.
gSQL> SELECT SHARD_GROUP_NAME( T1, 'B' ) FROM DUAL;
SHARD_GROUP_NAME( T1, 'B' )
-----
G2
1 row selected.
```

SHARD_ID

구문

```
SHARD_ID( table_name, shard_key_value [ , ... ] )
```

설명

SHARD_ID 함수는 table_name에 shard strategy가 정의된 경우, shard_key_value 값을 저장할 수 있는 shard에 대한 ID를 반환한다.

입력 인자 table_name은 identifier로 기술하여야 하며 table_name에 해당하는 객체가 base table이 아니거나 shard strategy가 정의되지 않은 경우, 에러가 발생한다.

입력 인자 shard_key_value는 table_name에 정의된 shard strategy의 shard key column 순으로 나열되어야 한다. shard_key_value 개수와 shard key column 개수가 일치하지 않을 경우, 에러가 발생한다.

결과 타입은 NATIVE_BIGINT이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT T1.C1, SHARD_ID( T1, T1.C1 ) FROM T1;
C1 SHARD_ID( T1, T1.C1 )
-- -----
A                0
B                1
C                2
3 rows selected.
gSQL> SELECT SHARD_ID( T1, 'B' ) FROM DUAL;
SHARD_ID( T1, 'B' )
-----
1
1 row selected.
```

SHARD_NAME

구문

```
SHARD_NAME( table_name, shard_key_value [ , ... ] )
```

설명

SHARD_NAME 함수는 table_name에 shard strategy가 정의된 경우, shard_key_value 값을 저장할 수 있는 shard에 대한 NAME을 반환한다.

입력 인자 table_name은 identifier로 기술하여야 하며 table_name에 해당하는 객체가 base table이 아니거나 shard strategy가 정의되지 않은 경우, 에러가 발생한다.

입력 인자 shard_key_value는 table_name에 정의된 shard strategy의 shard key column 순으로 나열되어야 한다. shard_key_value 개수와 shard key column 개수가 일치하지 않을 경우, 에러가 발생한다.

결과 타입은 VARCHAR이다.

노트

Cluster system에서 유효한 정보이다.

사용 예

```
gSQL> SELECT T1.C1, SHARD_NAME( T1, T1.C1 ) FROM T1;
C1 SHARD_NAME( T1, T1.C1 )
-- -----
A  S1
B  S2
C  S3
3 rows selected.
gSQL> SELECT SHARD_NAME( T1, 'B' ) FROM DUAL;
SHARD_NAME( T1, 'B' )
-----
S2
1 row selected.
```

SHIFT_LEFT

구문

```
SHIFT_LEFT( num, cnt )
```

설명

SHIFT_LEFT 함수는 num을 cnt 비트만큼 왼쪽으로 이동시킨 값을 반환한다.

입력 인자 num, cnt에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환하면 소수점은 TRUNCATE 된다.

연산 시 cnt는 6 bit로 마스킹하여 6 bit 범위 내의 값으로 처리한다.

결과 타입은 NATIVE_BIGINT이다.

사용 예

```
gSQL> SELECT SHIFT_LEFT( 7, 3 ) AS RESULT FROM DUAL;  
RESULT  
-----  
      56  
1 row selected.
```

SHIFT_RIGHT

구문

```
SHIFT_RIGHT( num, cnt )
```

설명

SHIFT_RIGHT 함수는 num을 cnt 비트만큼 오른쪽으로 이동시킨 값을 반환한다.

입력 인자 num, cnt에는 NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT 또는 NATIVE_BIGINT로 변환될 수 있는 타입이 올 수 있다.

NATIVE_BIGINT 타입으로 변환하면 소수점은 TRUNCATE 된다.

연산 시 cnt는 6 bit로 마스킹하여 6 bit 범위내의 값으로 처리한다.

결과 타입은 NATIVE_BIGINT이다.

사용 예

```
gSQL> SELECT SHIFT_RIGHT( 56, 3 ) AS RESULT FROM DUAL;
RESULT
-----
      7
1 row selected.
```

SIGN

구문

SIGN(num)

설명

SIGN 함수는 num의 부호를 반환한다.

인자 num에는 숫자타입이 올 수 있다.

반환값은 다음과 같다.

- num < 0 이면 -1
- num = 0 이면 0
- num > 0 이면 1

사용 예

```
gSQL> SELECT SIGN(-10) AS RESULT1,  
              SIGN(0) AS RESULT2,  
              SIGN(10) AS RESULT3 FROM DUAL;
```

```
RESULT1 RESULT2 RESULT3
```

```
-----
```

```
      -1         0         1
```

```
1 row selected.
```


SIN

구문

SIN(num)

설명

SIN 함수는 num의 sine 값을 반환한다.

사용 예

```
gSQL> SELECT SIN( 0 ) AS RESULT FROM DUAL;  
RESULT  
-----  
      0  
1 row selected.
```

SPLIT_PART

구문

SPLIT_PART(string, delimiter, field)

설명

SPLIT_PART 함수는 string 내에서 delimiter로 지정된 문자를 구분자로 하여 field의 문자열을 반환한다.

인자 string, delimiter에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

인자 field에는 숫자 타입이 올 수 있다.

string, delimiter, field 중에 하나라도 NULL인 경우에는 결과값도 NULL이다.
field에는 1 이상의 숫자값만 올 수 있고, 0 또는 음수일 경우에는 에러를 반환한다.

결과 타입은 다음 표와 같다.

표 11-68 SPLIT_PART의 결과 타입

string 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

사용 예

```
gSQL> SELECT SPLIT_PART( 'AB;CD;EF;GH', ';', 3 ) AS RESULT FROM DUAL;
RESULT
-----
EF
1 row selected.
```

SQRT

구문

SQRT(num)

설명

SQRT 함수는 num의 제곱근을 반환한다.

인자 num에는 숫자 타입이 올 수 있는데 음수가 아닌 0 이상의 값이어야 한다.

사용 예

```
gSQL> SELECT SQRT( 9 ) AS RESULT FROM DUAL;  
RESULT  
-----  
          3  
1 row selected.
```

STATEMENT_DATE

구문

```
STATEMENT_DATE()  
CURRENT_DATE [()]
```

설명

현재 날짜 (DATE type) 값을 얻는다.

현재 날짜를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_DATE(): 트랜잭션 내의 모든 날짜값이 동일하다.
- STATEMENT_DATE(): 하나의 SQL 문장 내에서 모든 날짜 값이 동일하다.
- CLOCK_DATE(): 함수가 호출될 때마다 현재 날짜 값을 얻는다.

사용 예

```
gSQL> SELECT STATEMENT_DATE() AS result FROM t1;  
RESULT  
-----  
2013-12-12  
2013-12-12  
2013-12-12  
3 rows selected.
```

STATEMENT_LOCALTIME

구문

```
STATEMENT_LOCALTIME()  
LOCALTIME [()]
```

설명

Session 시간을 기준으로 현재 TIME WITHOUT TIME ZONE type 값을 얻는다.

LOCALTIME은 SQL 표준 함수이다.

현재 시간을 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_LOCALTIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_LOCALTIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

모든 row가 같은 시간값을 갖는다.

```
gSQL> SELECT STATEMENT_LOCALTIME() AS result FROM t1;  
RESULT  
-----  
16:18:50.775870  
16:18:50.775870  
16:18:50.775870  
3 rows selected.
```

STATEMENT_LOCALTIMESTAMP

구문

```
STATEMENT_LOCALTIMESTAMP()  
LOCALTIMESTAMP [()]
```

설명

Session 시간을 기준으로 현재 TIMESTAMP WITHOUT TIME ZONE type 값을 얻는다.

LOCALTIMESTAMP는 SQL 표준 함수이다.

현재 TIMESTAMP를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_LOCALTIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_LOCALTIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT STATEMENT_LOCALTIMESTAMP() FROM t1;  
STATEMENT_LOCALTIMESTAMP()  
-----  
2013-12-12 16:23:39.782187  
2013-12-12 16:23:39.782187  
2013-12-12 16:23:39.782187  
3 rows selected.
```

STATEMENT_TIME

구문

```
STATEMENT_TIME()  
CURRENT_TIME [( )]
```

설명

TIME_ZONE이 있는 현재 TIME (TIME WITH TIME_ZONE type) 값을 얻는다.

CURRENT_TIME은 SQL 표준 함수이다.

현재 시간을 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_TIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_TIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_TIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

모든 row가 같은 시간값을 갖는다.

```
gSQL> SELECT STATEMENT_TIME() AS result FROM t1;  
RESULT  
-----  
16:28:19.268513 +09:00  
16:28:19.268513 +09:00  
16:28:19.268513 +09:00  
3 rows selected.
```

STATEMENT_TIMESTAMP

구문

```
STATEMENT_TIMESTAMP()  
CURRENT_TIMESTAMP [()]
```

설명

TIME_ZONE이 있는 현재 TIMESTAMP (TIMESTAMP WITH TIME_ZONE type) 값을 얻는다.

CURRENT_TIMESTAMP는 SQL 표준 함수이다.

현재 TIMESTAMP를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_TIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_TIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_TIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

모든 row가 같은 값을 갖는다.

```
gSQL> SELECT STATEMENT_TIMESTAMP() AS result FROM t1;  
RESULT  
-----  
2013-12-12 16:36:11.032957 +09:00  
2013-12-12 16:36:11.032957 +09:00  
2013-12-12 16:36:11.032957 +09:00  
3 rows selected.
```


STATEMENT_VIEW_SCN

구문

STATEMENT_VIEW_SCN()

설명

현재 STATEMENT의 VIEW SCN을 얻는다.

사용 예

```
gSQL> SELECT STATEMENT_VIEW_SCN() FROM dual;  
STATEMENT_VIEW_SCN()  
-----  
17697.658.17880  
1 row selected.
```

STATEMENT_VIEW_SCN_DCN

구문

```
STATEMENT_VIEW_SCN_DCN()
```

설명

현재 STATEMENT의 VIEW SCN의 Domain Change Number (DCN) 값을 얻는다.

사용 예

```
gSQL> SELECT STATEMENT_VIEW_SCN_DCN() FROM dual;
STATEMENT_VIEW_SCN_DCN()
-----
                        658
1 row selected.
```

STATEMENT_VIEW_SCN_GCN

구문

STATEMENT_VIEW_SCN_GCN()

설명

현재 STATEMENT의 VIEW SCN의 Global Change Number (GCN) 값을 얻는다.

사용 예

```
gSQL> SELECT STATEMENT_VIEW_SCN_GCN() FROM dual;  
STATEMENT_VIEW_SCN_GCN()  
-----  
17697  
1 row selected.
```

STATEMENT_VIEW_SCN_LCN

구문

```
STATEMENT_VIEW_SCN_LCN()
```

설명

현재 STATEMENT의 VIEW SCN의 Local Change Number (LCN) 값을 얻는다.

사용 예

```
gSQL> SELECT STATEMENT_VIEW_SCN_LCN() FROM dual;  
STATEMENT_VIEW_SCN_LCN()  
-----  
17880  
1 row selected.
```

STDDEV

구문

```
STDDEV( [ ALL | DISTINCT ] expr )
```

설명

Aggregation 함수로써 expr set의 표준편차 (standard deviation)를 얻는다.

ALL을 명시한 경우, 모든 값들에 대해 수행하고, DISTINCT를 명시한 경우 중복을 제거한 값들에 대해 수행한다. 둘 다 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

NULL 값을 제외하고 DISTINCT로 중복을 제거한 후의 expr set의 개수가 한 개일 경우, **VARIANCE**와 같이 0을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-69 STDDEV 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

GOLDILOCKS는 다음과 같이 표준편차를 계산한다.

- expr set의 개수가 1이면 0을 반환한다.
- expr set의 개수가 1보다 크면 **STDDEV_SAMP(expr)** 값을 반환한다.

노트

표준편차는 분산의 양의 제곱근으로써 분산에 루트를 씌워서 계산한다.
즉, STDDEV 함수는 **VARIANCE** 함수에 루트를 씌운 것과 같다.

```
STDDEV( [ ALL ] expr )
= SQRT( VARIANCE( [ ALL ] expr ) )
```

```
STDDEV( DISTINCT expr )  
= SQRT( VARIANCE( DISTINCT expr ) )
```

사용 예

```
gSQL> SELECT STDDEV(c1) FROM t1;  
      STDDEV(C1)  
-----  
11.4978258814438  
1 row selected.  
gSQL> SELECT STDDEV(ALL c1) FROM t1;  
      STDDEV(ALL C1)  
-----  
11.4978258814438  
1 row selected.  
gSQL> SELECT STDDEV(DISTINCT c1) FROM t1;  
      STDDEV(DISTINCT C1)  
-----  
13.2759180473518  
1 row selected.
```

STDDEV_POP

구문

STDDEV_POP(expr)

설명

Aggregation 함수로써 expr set의 모 표준편차 (population standard deviation)를 얻는다.
NULL 값을 제외한 expr set의 개수가 한 개일 때 0을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-70 STDDEV_POP의 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

노트

모 표준편차는 모 분산의 양의 제곱근으로써 모 분산에 루트를 씌워서 계산한다.
즉, STDDEV_POP 함수는 VAR_POP 함수에 루트를 씌운 것과 같다.

STDDEV_POP(expr)
= SQRT(VAR_POP(expr))

사용 예

```
gSQL> SELECT STDDEV_POP(c1) FROM t1;
STDDEV_POP(C1)
-----
10.283968105746
```

1,134 | SQL Elements

1 row selected.

STDDEV_SAMP

구문

STDDEV_SAMP(expr)

설명

Aggregation 함수로써 expr set의 표본 표준편차 (sample standard deviation)를 얻는다. NULL 값을 제외한 expr set의 개수가 한 개일 때 NULL값을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-71 STDDEV_SAMP의 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

노트

표본 표준편차는 표본 분산의 양의 제곱근으로써 표본 분산에 루트를 씌워서 계산한다. 즉, STDDEV_SAMP 함수는 **VAR_SAMP** 함수에 루트를 씌운 것과 같다.

```
STDDEV_SAMP( expr )
= SQRT( VAR_SAMP( expr ) )
```

사용 예

```
gSQL> SELECT STDDEV_SAMP(c1) FROM t1;
STDDEV_SAMP(C1)
-----
11.4978258814438
```

1,136 | SQL Elements

1 row selected.

SUBSTR

구문

```
SUBSTR( str FROM start_position [ FOR string_length ] )
SUBSTR( str, start_position [ , string_length ] )
```

설명

SUBSTRING의 alias 이다.

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTR( 'DATABASE MANAGEMENT SYSTEM', 10, 10 ) AS RESULT
    FROM DUAL;
RESULT
-----
MANAGEMENT
1 row selected.
```

- Multi byte character set (예: UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTR( ' “ αβ ≠ AB ” ', 2, 5 ) AS RESULT FROM DUAL;
RESULT
-----
αβ ≠ AB
1 row selected.
```

SUBSTRB

구문

```
SUBSTRB( str, start_position [ , string_length ] )
```

설명

SUBSTRB 함수는 str에 대해 start_position으로부터 string_length 범위의 문자를 추출하여 반환한다.

SUBSTRB 함수는 start_position과 string_length가 byte 단위로 계산된다는 점 외에는 SUBSTRING 함수와 동일하다.

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTRB( 'DATABASE MANAGEMENT SYSTEM', 10, 10 ) AS RESULT
    FROM DUAL;

RESULT
-----
MANAGEMENT
1 row selected.
```

- Multi byte character set (예: UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTRB( ' “ αβ ≠ AB ” ', 4, 11 ) AS RESULT FROM DUAL;

RESULT
-----
αβ ≠ AB
1 row selected.
```

SUBSTRING

구문

```
SUBSTRING( str FROM start_position [ FOR string_length ] )
SUBSTRING( str, start_position [ , string_length ] )
```

설명

SUBSTRING 함수는 str에 대해 start_position으로부터 string_length 범위의 문자를 추출하여 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

인자 start_position과 string_length에는 숫자 타입이 올 수 있다.

str, start_position과 string_length 중의 하나라도 NULL인 경우, 결과값은 NULL이다.

start_position과 string_length는 1부터 시작하며, character set에 따른 문자 단위로 계산된다. (byte 단위가 아님)

start_position이 0인 경우, 1로 처리된다.

start_position이 양수인 경우, str의 앞부분에서부터 position의 위치를 찾는다.

start_position이 음수인 경우, str의 뒷부분에서부터 position의 위치를 찾는다.

string_length가 생략된 경우, start_position으로부터 str의 마지막 문자까지 반환한다.

string_length에 0 또는 음수가 오는 경우, 결과값은 NULL이다.

start_position > (str의 길이) 이면, 결과값은 NULL이다.

(str의 길이 + start_position) < 0 이면, 결과값은 NULL이다.

SUBSTR의 alias이다.

자세한 내용은 **SUBSTR**과 **SUBSTRB**를 참조한다.

결과 타입은 다음 표와 같다.

표 11-72 SUBSTRING의 결과 타입

str	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

- Multi byte character set (예: UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTRING( 'DATABASE MANAGEMENT SYSTEM' FROM 10 FOR 10 ) AS RESULT
FROM DUAL;
RESULT
-----
MANAGEMENT
1 row selected.
```

- Multi byte character set (예: UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTRING( ' “ αβ ≠ AB ” ' FROM 2 FOR 5 ) AS RESULT FROM DUAL;
RESULT
-----
αβ ≠ AB
1 row selected.
```

SUM

구문

```
SUM( [ ALL | DISTINCT ] expr )
```

설명

Aggregation 함수로써 expr 값들의 합을 얻는다.

ALL을 명시한 경우, 모든 값들에 대해 aggregation을 수행한다.

DISTINCT를 명시한 경우, 중복을 제거한 값들에 대해 aggregation을 수행한다.

ALL이나 DISTINCT를 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

사용 예

```
gSQL> SELECT SUM(c1) FROM t1;
SUM(C1)
-----
      6
1 row selected.
```

SYSDATE

구문

SYSDATE

설명

Database server가 위치하는 OS 시간을 기준으로 현재의 DATE type 값을 얻는다.

사용 예

```
gSQL> SELECT SYSDATE FROM t1;
SYSDATE
-----
2013-12-12
2013-12-12
2013-12-12
3 rows selected.
```


SYS_EXTRACT_UTC

구문

```
SYS_EXTRACT_UTC( datetime_with_timezone )
```

설명

SYS_EXTRACT_UTC는 UTC (Coordinated Universal Time—formerly Greenwich Mean Time) 값을 반환한다. timezone이 명시되지 않은 경우, session time zone으로 계산된다.

입력 인자에는 time, time with time zone, timestamp, timestamp with time zone 타입이 올 수 있다. 결과 타입은 time 또는 timestamp 타입이다.

사용 예

```
gSQL> SELECT
      SYS_EXTRACT_UTC(
        TO_TIMESTAMP_TZ( '2017-05-25 21:13:32.123456 +09:00',
                        'YYYY-MM-DD HH24:MI:SS.FF6 TZh:TZM' )
      ) AS RESULT
    FROM DUAL;
RESULT
-----
2017-05-25 12:13:32.123456
1 row selected.
```

SYSTIME

구문

SYSTIME

설명

Database server가 위치하는 OS 시간을 기준으로 TIME ZONE이 있는 현재 시간 (TIME WITH TIME ZONE type) 값을 얻는다.

사용 예

```
gSQL> SELECT SYSTIME FROM t1;
SYSTIME
-----
16:30:46.954941 +09:00
16:30:46.954941 +09:00
16:30:46.954941 +09:00
3 rows selected.
```

SYSTIMESTAMP

구문

SYSTIMESTAMP

설명

Database server가 위치하는 OS 시간을 기준으로 TIME ZONE이 있는 현재 TIMESTAMP WITH TIME ZONE type 값을 얻는다.

사용 예

```
gSQL> SELECT SYSTIMESTAMP FROM t1;
SYSTIMESTAMP
-----
2013-12-12 16:37:34.432241 +09:00
2013-12-12 16:37:34.432241 +09:00
2013-12-12 16:37:34.432241 +09:00
3 rows selected.
```

TAN

구문

TAN(num)

설명

TAN 함수는 num의 tangent 값을 라디안 단위로 반환한다.

사용 예

```
gSQL> SELECT TAN( 1 ) AS RESULT FROM DUAL;  
          RESULT  
-----  
1.5574077246549  
1 row selected.
```

TO_BASE64

구문

```
TO_BASE64( str )
```

설명

TO_BASE64는 str을 base64 인코딩으로 변환한 문자를 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입 또는 문자 타입으로 변환될 수 있는 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

결과 타입은 CHARACTER VARYING 또는 CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이다.

str이 NULL이면 결과값도 NULL이다.

Base64 인코딩은 8 비트 바이너리 데이터를 ascii 영역으로 구성된 64개의 문자로 표현한다.

64개의 문자는 A~Z, a~z, 0~9, +, / 로 구성된다.

6 bit를 하나의 문자로 표현하며, 세 개의 문자 (24 bit)를 하나의 단위로 네 개의 문자로 표현한다.

인코딩 된 문자가 네 개의 문자를 채우지 못하면 나머지는 '=' 로 채운다.

인코딩 된 문자가 76개를 넘으면 newline이 추가되어 여러 라인으로 나누어진다.

Base64 인코딩 된 문자의 디코딩은 FROM_BASE64() 함수를 이용한다.

Base64를 디코딩 할 때는 newline, carriage return, tab, space가 무시된다.

자세한 내용은 **FROM_BASE64**를 참조한다.

사용 예

```
gSQL> SELECT TO_BASE64( 'abc' ), TO_BASE64( 'abcd' ) FROM DUAL;
TO_BASE64( 'abc' ) TO_BASE64( 'abcd' )
-----
YWJj                YWJjZA==
1 row selected.
```

TO_CHAR(datetime)

구문

```
TO_CHAR( datetime [, fmt ] )
```

설명

TO_CHAR(datetime) 함수는 datetime을 명시된 fmt 형식의 문자열로 변환하여 반환한다.

인자 datetime에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL 타입이 올 수 있다.

인자 fmt에는 CHARACTER, CHARACTER VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, 다음과 같은 default format 형식을 따른다.

- DATE: **NLS_DATE_FORMAT**
- TIMESTAMP: **NLS_TIMESTAMP_FORMAT**
- TIMESTAMP WITH TIME ZONE: **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**
- TIME: **NLS_TIME_FORMAT**
- TIME WITH TIME ZONE: **NLS_TIME_WITH_TIME_ZONE_FORMAT**

인자 datetime이 INTERVAL 타입인 경우, fmt와 무관하게 string으로 변환하여 반환한다.

fmt에 지정할 수 있는 문자열은 **Datetime Format 문자열**을 참조한다.

결과 타입은 CHARACTER VARYING 이다.

사용 예

다음은 fmt가 생략되고, NLS_DATE_FORMAT = 'YYYY-MM-DD' 인 경우의 예이다.

```
gSQL> SELECT
      TO_CHAR( TO_DATE( '2012-03-15', 'YYYY-MM-DD' ) ) AS RESULT
    FROM DUAL;

RESULT
-----
2012-03-15
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT
      TO_CHAR( TO_DATE('2012-03-15','YYYY-MM-DD'), 'DD-MON-YY' ) AS RESULT
    FROM DUAL;
RESULT
-----
15-MAR-12
1 row selected.
```

TO_CHAR(number)

구문

```
TO_CHAR( number [ , fmt ] )
```

설명

TO_CHAR(number) 함수는 number를 명시된 fmt 형식의 문자열로 변환하여 반환한다.

인자 number에는 숫자 타입이 올 수 있다.

인자 fmt에는 CHARACTER, CHARACTER VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, 모든 유효 숫자를 문자열로 변환하여 반환한다.

fmt에 지정할 수 있는 문자열은 **Number Format 문자열**을 참조한다.

결과 타입은 CHARACTER VARYING 이다.

사용 예

```
gSQL> SELECT TO_CHAR( 12500000 ) AS RESULT FROM DUAL;
RESULT
-----
12500000
1 row selected.
gSQL> SELECT TO_CHAR( 12500000, 'S999,999,999' ) AS RESULT FROM DUAL;
RESULT
-----
+12,500,000
1 row selected.
```


TO_DATE

구문

```
TO_DATE( str [, fmt ] )
```

설명

TO_DATE 함수는 명시된 fmt 형식의 문자열 str을 DATE 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, NLS_DATE_FORMAT은 default format 형식이 되고, 이 경우 str은 default format 형식의 문자열이어야 한다.

fmt에 지정할 수 있는 문자열은 **Datetime Format 문자열**을 참조한다.

자세한 내용은 NLS_DATE_FORMAT을 참조한다.

결과 타입은 DATE 이다.

사용 예

다음은 fmt가 생략되었고, NLS_DATE_FORMAT = 'YYYY-MM-DD' 인 경우의 예이다.

```
gSQL> SELECT TO_DATE( '2009-07-29' ) AS RESULT FROM DUAL;
RESULT
-----
2009-07-29
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT TO_DATE( '29-JUL-09', 'DD-MON-YY' ) AS RESULT FROM DUAL;
RESULT
-----
2009-07-29
1 row selected.
```

TO_NATIVE_DOUBLE

구문

```
TO_NATIVE_DOUBLE( str [, fmt ] )
```

설명

TO_NATIVE_DOUBLE 함수는 명시된 fmt 형식의 문자열 str을 NATIVE_DOUBLE 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

str과 fmt 중 하나라도 NULL이면, 결과값도 NULL이다.

fmt에 지정할 수 있는 문자열은 **Number Format 문자열**을 참조한다.

결과 타입은 NATIVE_DOUBLE이다.

사용 예

```
gSQL> SELECT TO_NATIVE_DOUBLE( '123.45' ) AS RESULT1,  
            TO_NATIVE_DOUBLE( '+123.45', 'S999.99' ) AS RESULT2  
      FROM DUAL;  
RESULT1 RESULT2  
-----  
123.45  123.45  
1 row selected.
```

TO_NATIVE_REAL

구문

```
TO_NATIVE_REAL( str [, fmt ] )
```

설명

TO_NATIVE_REAL 함수는 명시된 fmt 형식의 문자열 str을 NATIVE_REAL 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

str과 fmt 중 하나라도 NULL이면, 결과값도 NULL이다.

fmt에 지정할 수 있는 문자열은 **Number Format 문자열**을 참조한다.

결과 타입은 NATIVE_REAL이다.

사용 예

```
gSQL> SELECT TO_NATIVE_REAL( '123.45' ) AS RESULT1,
              TO_NATIVE_REAL( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
-----
123.45  123.45
```

TO_NUMBER

구문

```
TO_NUMBER( str [, fmt] )
```

설명

TO_NUMBER 함수는 명시된 fmt 형식의 문자열 str을 NUMBER 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

str과 fmt 중 하나라도 NULL이면 결과값도 NULL이다.

fmt에 지정할 수 있는 문자열은 **Number Format 문자열**을 참조한다.

결과 타입은 NUMBER이다.

사용 예

```
gSQL> SELECT TO_NUMBER( '123.45' ) AS RESULT1,
              TO_NUMBER( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
-----
123.45  123.45
1 row selected.
```

TO_TIME

구문

```
TO_TIME( str [, fmt ] )
```

설명

TO_TIME 함수는 명시된 fmt 형식의 문자열 str을 TIME 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, NLS_TIME_FORMAT이 default format 형식이 되고, 이 경우 str은 default format 형식의 문자열이어야 한다.

fmt에 지정할 수 있는 문자열은 **Datetime Format 문자열**을 참조한다.

자세한 내용은 NLS_TIME_FORMAT을 참조한다.

결과 타입은 TIME 이다.

사용 예

다음은 fmt가 생략되었고, NLS_TIME_FORMAT = 'HH24:MI:SS.FF6' 인 경우의 예이다.

```
gSQL> SELECT TO_TIME( '11:22:33.999999' ) AS RESULT FROM DUAL;
RESULT
-----
11:22:33.999999
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT
      TO_TIME( '112233.999999/P.M.', 'HH12MISS.FF6/P.M.' ) AS RESULT
FROM DUAL;
RESULT
-----
23:22:33.999999
1 row selected.
```

TO_TIME_TZ

구문

```
TO_TIME_TZ( str [, fmt ] )
```

설명

TO_TIME_WITH_TIME_ZONE의 alias이다.

자세한 내용은 TO_TIME_WITH_TIME_ZONE과 NLS_TIME_WITH_TIME_ZONE_FORMAT을 참조한다.

사용 예

다음은 fmt가 생략되었고, NLS_TIME_WITH_TIME_ZONE_FORMAT = 'HH24:MI:SS.FF6 TZH:TZM' 인 경우의 예이다.

```
gSQL> SELECT TO_TIME_TZ( '11:22:33.999999 +09:00' ) AS RESULT FROM DUAL;
RESULT
-----
11:22:33.999999 +09:00
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT TO_TIME_TZ( '11:22:33.999999 +09:00 PM',
                        'HH12:MI:SS.FF6 TZH:TZM PM' ) AS RESULT
FROM DUAL;
RESULT
-----
23:22:33.999999 +09:00
1 row selected.
```



```
AS RESULT  
FROM DUAL;
```

```
RESULT
```

```
-----
```

```
23:22:33.999999 +09:00
```

```
1 row selected.
```


TO_TIMESTAMP

구문

```
TO_TIMESTAMP( str [, fmt ] )
```

설명

TO_TIMESTAMP 함수는 명시된 fmt 형식의 문자열 str을 TIMESTAMP 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, NLS_TIMESTAMP_FORMAT이 default format 형식이 되고, 이 경우 str은 default format 형식의 문자열이어야 한다.

fmt에 지정할 수 있는 문자열은 **Datetime Format 문자열**을 참조한다.

자세한 내용은 NLS_TIMESTAMP_FORMAT을 참조한다.

결과 타입은 TIMESTAMP 이다.

사용 예

다음은 fmt가 생략되었고, NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6' 인 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP( '2009-07-29 11:22:33.999999' ) AS RESULT
    FROM DUAL;

RESULT
-----
2009-07-29 11:22:33.999999
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP( '090729 112233999999 PM', 'YMMDD HH12MISSFF6 PM' )
    AS RESULT
    FROM DUAL;

RESULT
-----
2009-07-29 23:22:33.999999
1 row selected.
```

TO_TIMESTAMP_TZ

구문

```
TO_TIMESTAMP_TZ( str [, fmt ] )
```

설명

TO_TIMESTAMP_WITH_TIME_ZONE의 alias이다.

자세한 내용은 TO_TIMESTAMP_WITH_TIME_ZONE과 NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT을 참조한다.

사용 예

다음은 fmt가 생략되었고, NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' 인 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP_TZ( '2009-07-29 11:22:33.999999 +09:00' ) AS RESULT
    FROM DUAL;
RESULT
-----
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP_TZ( '29-JUL-09 11:22:33.999999 +09:00',
                      'DD-MON-RR HH12:MI:SS.FF6 TZH:TZM' ) AS RESULT
    FROM DUAL;
RESULT
-----
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

TO_TIMESTAMP_WITH_TIME_ZONE

구문

```
TO_TIMESTAMP_WITH_TIME_ZONE( str [, fmt ] )
TO_TIMESTAMP_TZ( str [, fmt ] )
```

설명

TO_TIMESTAMP_WITH_TIME_ZONE 함수는 명시된 fmt 형식의 문자열 str을 TIMESTAMP WITH TIME ZONE 타입으로 변환하여 반환한다.

인자 str과 fmt에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

fmt가 생략된 경우, NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT이 default format 형식이 되고, 이 경우 str은 default format 형식의 문자열이어야 한다.

fmt에 지정할 수 있는 문자열은 **Datetime Format 문자열**을 참조한다.

자세한 내용은 NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT을 참조한다.

TO_TIMESTAMP_WITH_TIME_ZONE의 alias로는 TO_TIMESTAMP_TZ 함수가 있다.

결과 타입은 TIMESTAMP WITH TIME ZONE 이다.

사용 예

다음은 fmt가 생략되었고, NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' 인 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP_WITH_TIME_ZONE( '2009-07-29 11:22:33.999999 +09:00' )
      AS RESULT
      FROM DUAL;

RESULT
-----
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

다음은 fmt가 지정된 경우의 예이다.

```
gSQL> SELECT
      TO_TIMESTAMP_WITH_TIME_ZONE( '29-JUL-09 11:22:33.999999 +09:00',
```

```
'DD-MON-RR HH12:MI:SS.FF6 TZH:TZM' )
```

```
AS RESULT
```

```
FROM DUAL;
```

```
RESULT
```

```
-----
```

```
2009-07-29 11:22:33.999999 +09:00
```

```
1 row selected.
```

TRANSACTION_DATE

구문

TRANSACTION_DATE()

설명

Session 시간을 기준으로 현재 날짜 (DATE type) 값을 얻는다.

현재 날짜를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_DATE(): 트랜잭션 내의 모든 날짜값이 동일하다.
- STATEMENT_DATE(): 하나의 SQL 문장 내에서 모든 날짜 값이 동일하다.
- CLOCK_DATE(): 함수가 호출될 때마다 현재 날짜 값을 얻는다.

사용 예

동일한 transaction 내에서는 날짜 값이 항상 같다.

```
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
-----
2013-12-12
1 row selected.
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
-----
2013-12-12
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
-----
2013-12-13
1 row selected.
```

TRANSACTION_LOCALTIME

구문

TRANSACTION_LOCALTIME()

설명

Session 시간을 기준으로 TIME_ZONE이 없는 현재 시간 (TIME WITHOUT TIME_ZONE type) 값을 얻는다.

현재 시간을 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_LOCALTIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_LOCALTIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

동일한 transaction 내에서는 시간값이 항상 같다.

```
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----
16:43:24.391834
1 row selected.
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----
16:43:24.391834
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----
16:43:32.651833
1 row selected.
```

TRANSACTION_LOCALTIMESTAMP

구문

TRANSACTION_LOCALTIMESTAMP()

설명

Session 시간을 기준으로 TIME_ZONE이 없는 현재 TIMESTAMP (TIMESTAMP WITHOUT TIME_ZONE type) 값을 얻는다.

현재 TIMESTAMP를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_LOCALTIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_LOCALTIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_LOCALTIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

동일한 transaction 내에서는 TIMESTAMP 값이 항상 같다.

```
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
-----
2013-12-12 16:43:32.651833
1 row selected.
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
-----
2013-12-12 16:43:32.651833
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
-----
2013-12-12 16:46:07.831834
1 row selected.
```

TRANSACTION_TIME

구문

TRANSACTION_TIME()

설명

Session 시간을 기준으로 TIME_ZONE이 있는 현재 시간 (TIME WITH TIME_ZONE type) 값을 얻는다.

현재 시간을 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_TIME(): 트랜잭션 내의 모든 시간값이 동일하다.
- STATEMENT_TIME(): 하나의 SQL 문장 내에서 모든 시간값이 동일하다.
- CLOCK_TIME(): 함수가 호출될 때마다 현재 시간값을 얻는다.

사용 예

동일한 transaction 내에서는 시간값이 항상 같다.

```
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
-----
16:46:07.831834 +09:00
1 row selected.
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
-----
16:46:07.831834 +09:00
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
-----
16:48:00.691827 +09:00
1 row selected.
```


TRANSACTION_TIMESTAMP

구문

TRANSACTION_TIMESTAMP()

설명

Session 시간을 기준으로 TIME_ZONE이 있는 현재 TIMESTAMP (TIMESTAMP WITH TIME_ZONE type) 값을 얻는다.

현재 TIMESTAMP를 얻는 함수들 사이에는 다음과 같은 차이가 있다.

- TRANSACTION_TIMESTAMP(): 트랜잭션 내의 모든 TIMESTAMP 값이 동일하다.
- STATEMENT_TIMESTAMP(): 하나의 SQL 문장 내에서 모든 TIMESTAMP 값이 동일하다.
- CLOCK_TIMESTAMP(): 함수가 호출될 때마다 현재 TIMESTAMP 값을 얻는다.

사용 예

동일한 transaction 내에서는 TIMESTAMP 값이 항상 같다.

```
gSQL> SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-----
2013-12-12 16:48:00.691827 +09:00
1 row selected.
gSQL> SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-----
2013-12-12 16:48:00.691827 +09:00
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-----
2013-12-12 16:49:26.291827 +09:00
1 row selected.
```

TRANSLATE

구문

TRANSLATE(string, from, to)

설명

TRANSLATE 함수는 string에서 from의 문자와 일치하는 모든 문자를 그에 대응하는 to 문자로 치환하여 반환한다.

인자 string, from, to에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

string, from, to 중의 하나라도 NULL이면 결과값도 NULL이다.

string에 from의 문자와 일치하지 않는 문자는 치환되지 않는다.

string에 from의 문자와 일치하는 문자는 그에 대응하는 to 문자로 치환된다.

from 문자의 개수가 to 문자의 개수보다 많은 경우, to 문자와 대응되지 않는 from 문자들은 string에서 제거된 후 반환된다.

from에 동일한 문자가 여러 번 쓰인 경우, 첫 번째 매핑된 문자로 치환된다.

결과 타입은 다음 표와 같다.

표 11-73 TRANSLATE의 결과 타입

string 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

사용 예

```
gSQL> SELECT TRANSLATE( '12345', '15', 'FL' ) AS RESULT FROM DUAL;
RESULT
-----
F234L
1 row selected.
gSQL> SELECT TRANSLATE( 'ABC12345', 'ABC12345', 'XYZ' ) AS RESULT FROM DUAL;
RESULT
-----
XYZ
1 row selected.
gSQL> SELECT TRANSLATE( 'ABC12345ABC', 'ABCABCABC', 'XYZ^&*xyz' ) AS RESULT
```

```
FROM DUAL;  
RESULT  
-----  
XYZ12345XYZ  
1 row selected.
```

TRIM

구문

```
TRIM([ [ LEADING | TRAILING | BOTH ] [trim_character] FROM ] trim_source)
```

설명

TRIM 함수는 trim_source에서 trim_character를 LEADING, TRAILING, BOTH 방향에서 비교하여 일치되는 문자가 없을 때까지 제거한 결과를 반환한다.

인자 trim_character와 trim_source에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입과 BINARY, BINARY VARYING, BINARY LONG VARYING과 같은 BINARY 문자 타입이 올 수 있다.

trim_character, trim_source 중 하나라도 NULL이면 결과값은 NULL이다.

- [LEADING | TRAILING | BOTH]
 - LEADING: trim_source의 앞부분부터 trim_character를 제거한다.
 - TRAILING: trim_source의 뒷부분부터 trim_character를 제거한다.
 - BOTH: trim_source의 앞뒤 양방향에서 trim_character를 제거한다.
- trim_character는 한 개의 문자만 허용된다.
- trim_character가 생략된 경우에는 single blank space (' ')가 기본적으로 지정된다.
- FROM이 지정된 경우
 - [LEADING | TRAILING | BOTH] 또는 trim_character 또는 [LEADING | TRAILING | BOTH] trim_character가 지정되어야 한다.
 - 예: TRIM(LEADING FROM ' abc') , TRIM('x' FROM 'xabc') , TRIM(LEADING 'x' FROM 'xabc')
 - [LEADING | TRAILING | BOTH]이 생략된 경우에는 BOTH가 기본적으로 지정된다.
- FROM이 생략된 경우
 - TRIM(trim_source)인 경우이며 TRIM(BOTH ' ' FROM trim_source)과 동일하게 수행된다.

결과 타입은 다음 표와 같다.

표 11-74 TRIM의 결과 타입

trim_character, trim_source 타입	결과 타입
CHAR 또는 VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY 또는 VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

사용 예

```
gSQL> SELECT TRIM( LEADING '_' FROM '___TRIM FUNCTION___' ) AS RESULT
        FROM DUAL;
```

```
RESULT
```

```
-----
```

```
TRIM FUNCTION___
```

```
1 row selected.
```

```
gSQL> SELECT TRIM( TRAILING '_' FROM '___TRIM FUNCTION___' ) AS RESULT
        FROM DUAL;
```

```
RESULT
```

```
-----
```

```
___TRIM FUNCTION
```

```
1 row selected.
```

```
gSQL> SELECT TRIM( BOTH '_' FROM '___TRIM FUNCTION___' ) AS RESULT
        FROM DUAL;
```

```
RESULT
```

```
-----
```

```
TRIM FUNCTION
```

```
1 row selected.
```

TRUNC(number)

구문

```
TRUNC( num [ , scale ] )
```

설명

TRUNC(number) 함수는 scale 기준으로 num을 버림한 값을 반환한다.

인자 num과 scale에는 숫자 타입이 올 수 있다.

scale이 생략된 경우, scale은 0이 되어 TRUNC(num, 0)일 때와 같이 실행된다.

scale이 양수인 경우, 소수점 오른쪽 자리수를 기준으로 버림한다.

scale이 음수인 경우, 소수점 왼쪽 자리수를 기준으로 버림한다.

사용 예

```
gSQL> SELECT TRUNC( 142.4282, 2 ) AS RESULT FROM DUAL;  
RESULT  
-----  
142.42  
1 row selected.  
gSQL> SELECT TRUNC( 142.4282, -2 ) AS RESULT FROM DUAL;  
RESULT  
-----  
100  
1 row selected.
```

TRUNC(date)

구문

TRUNC(date [, fmt])

설명

TRUNC(date) 함수는 date를 지정된 fmt 단위로 버림한 값을 반환한다.

인자 date에는 DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 타입이 올 수 있다.

인자 fmt에는 CHARACTER, CHARACTER VARYING과 같은 CHARACTER 문자 타입이 올 수 있다.

결과 타입은 인자로 받은 date 타입과 관계없이 항상 DATE 타입이다.

fmt가 생략되었을 경우의 기본값은 DAY이며, 사용 가능한 형식문자열은 다음 표와 같다.

표 11-75 fmt에 사용 가능한 형식문자열

형식문자열	설명
CC, SCC	세기
YYYY, YEAR, SYYYY, SYEAR, YYY, YY, Y	년
IYYY, IYY, IY, I	ISO 8601 표준에 정의된 calendar week를 수용하는 연도
Q	분기
MONTH, MON, MM, RM	월
WW	연도의 1월 1일을 한 주로 시작하는 주
IW	ISO 8601 표준에 정의된 calendar week (1~52주 또는 1~53주)로 지정된 연도의 첫 번째 목요일이 있는 주가 첫 번째 주
W	월의 1일을 한 주로 시작하는 주
DDD, DD, J	일
DAY, DY, D	요일
HH, HH12, HH24	시
MI	분

사용 예

```
gSQL> SELECT
      TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'CC' ) AS RESULT
FROM DUAL;

RESULT
-----
```

```
2001-01-01
```

```
1 row selected.
```

```
gSQL> SELECT
```

```
    TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'YYYY' ) AS RESULT
  FROM DUAL;
```

```
RESULT
```

```
-----
```

```
2051-01-01
```

```
1 row selected.
```

```
gSQL> SELECT
```

```
    TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'MONTH' ) AS RESULT
  FROM DUAL;
```

```
RESULT
```

```
-----
```

```
2051-07-01
```

```
1 row selected.
```

```
gSQL> SELECT
```

```
    TRUNC( TO_TIMESTAMP( '2001-05-05 11:22:33.999999',
                        'YYYY-MM-DD HH24:MI:SS.FF6' ) ) AS RESULT
  FROM DUAL;
```

```
RESULT
```

```
-----
```

```
2001-05-05
```

```
1 row selected.
```


UPPER

구문

UPPER(str)

설명

UPPER 함수는 str의 대문자를 반환한다.

인자 str에는 CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING과 같은 문자 타입이 올 수 있다.

str이 NULL이면 결과값도 NULL이다.

반환되는 타입은 인자 str과 동일한 타입이다.

사용 예

```
gSQL> SELECT UPPER( 'spring' ) AS RESULT FROM DUAL;  
RESULT  
-----  
SPRING  
1 row selected.
```

UNHEX

구문

UNHEX(str)

설명

인자 str은 16진수 문자이며, 이를 각 byte로 표현하여 binary string으로 반환한다.

입력 인자에는 CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있고, 결과 타입은 BINARY VARYING이나 BINARY LONG VARYING과 같은 BINARY 문자 타입이다.

str이 NULL이면 결과값도 NULL이다.

str에 16진수 범위에 속하지 않는 문자가 포함되면 에러를 반환한다.

자세한 내용은 **HEX**를 참조한다.

사용 예

```
gSQL> SELECT UNHEX( HEX( 'abc' ) ) FROM DUAL;  
UNHEX( HEX( 'abc' ) )  
-----  
616263  
1 row selected.
```

UNHEX_TO_CHARSTR

구문

UNHEX_TO_CHARSTR(str)

설명

인자 str은 16진수 문자이며 이를 각 byte로 표현하여 character string으로 반환한다.

입력 인자에는 CHARACTER VARYING, CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이 올 수 있고, 결과 타입은 CHARACTER VARYING이나 CHARACTER LONG VARYING과 같은 CHARACTER 문자 타입이다.

str이 NULL이면 결과값도 NULL이다.

str에 16진수 범위에 속하지 않는 문자가 포함되면 에러를 반환한다.

인자 str가 어떤 데이터의 16진수 문자표현인지 알 수 없으므로 character string으로 반환할 때 현재 적용할 수 있는 character set을 적용하여 결과값을 반환한다.

현재 적용할 수 있는 character set에 포함되지 않는 경우, 에러를 반환한다.

자세한 내용은 HEX와 UNHEX 를 참조한다.

사용 예

```
gSQL> SELECT UNHEX_TO_CHARSTR( '616263' ) FROM DUAL;
UNHEX_TO_CHARSTR( '616263' )
-----
abc
1 row selected.
gSQL> SELECT UNHEX_TO_CHARSTR( HEX( 'abc' ) ) FROM DUAL;
UNHEX_TO_CHARSTR( HEX( 'abc' ) )
-----
abc
1 row selected.
```

USER_ID

구문

USER_ID ()

설명

현재 사용자의 number ID를 얻는다.

노트

Cluster system에서는 접속한 server에 따라 다른 값을 가질 수 있다.
현재 사용자의 이름을 얻는 **CURRENT_USER** 함수 사용을 권장한다.

사용 예

```
% gsql test test
gSQL> SELECT USER_ID() FROM dual;
USER_ID()
-----
        6
1 row selected.
```

UUID

구문

UUID()

설명

UUID 함수는 전역고유식별자 (Universal Unique Identifier) 를 생성하여 반환한다.
반환되는 타입은 VARBINARY 이며 내부적으로 16 바이트로 구성된다.

사용 예

```
gSQL> SELECT HEX( UUID() ) FROM DUAL;  
HEX( UUID() )  
-----  
E6F0A5C2387511E8B95259E479C2FD50  
1 row selected.
```

VAR_POP

구문

VAR_POP(expr)

설명

Aggregation 함수로써 expr set의 모 분산 (population variance)을 얻는다.
NULL 값을 제외한 expr set의 개수가 한 개일 때 0을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-76 VAR_POP 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

노트

모 분산은 모 집단 (전체)의 분산이며 분산은 편차 제곱의 평균이다. 즉, 데이터의 각 값에서 모 평균 (전체의 평균)을 빼고 제곱해서 모두 더한 뒤 모 집단의 데이터 개수로 나눈다.
이는 각 관찰값들이 평균으로부터 얼마나 많이 퍼져있는지 파악하는데 사용된다.

자세한 내용은 STDDEV_POP 을 참조한다.

사용 예

```
gSQL> SELECT VAR_POP(c1) FROM t1;
VAR_POP(C1)
-----
105.76
```

1 row selected.

VAR_SAMP

구문

VAR_SAMP(expr)

설명

Aggregation 함수로써 expr set의 표본 분산 (sample variance)을 얻는다.
NULL 값을 제외한 expr set의 개수가 한 개일 때 NULL값을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-77 VAR_SAMP 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

노트

모 집단 (전체)을 다루는 모 분산과 달리, 표본 분산은 추출한 표본으로 평균과 편차를 다룬다. 즉, 데이터의 각 값에서 표본의 평균을 빼고 제곱해서 모두 더한 뒤, 표본 집단의 데이터 개수 - 1로 나눈다.
이는 모 집단의 분산을 추정하는 데 사용된다.

자세한 내용은 STDDEV_SAMP 을 참조한다.

사용 예

```
gSQL> SELECT VAR_SAMP(c1) FROM t1;  
VAR_SAMP(C1)  
-----  
132.2
```


1 row selected.

VARIANCE

구문

VARIANCE([ALL | DISTINCT] expr)

설명

Aggregation 함수로써 expr set의 분산 (variance)을 얻는다.

ALL을 명시한 경우 모든 값들에 대해 수행하고, DISTINCT를 명시한 경우 중복을 제거한 값들에 대해 수행한다. 둘 다 명시하지 않은 경우, ALL을 명시한 것과 동일하게 처리한다.

NULL 값을 제외하고 DISTINCT로 중복을 제거한 후의 expr set의 개수가 한 개일 경우, 0을 반환한다.

인자와 결과 타입은 다음 표와 같다.

표 11-78 VARIANCE 인자와 결과 타입

expr	결과 타입
NATIVE_INTEGER 계열 - NATIVE_SMALLINT - NATIVE_INTEGER - NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE 계열 - NATIVE_REAL - NATIVE_DOUBLE	NATIVE_DOUBLE

노트

GOLDILOCKS는 분산을 다음과 같이 계산한다.

- expr set의 개수가 1이면 0을 반환한다.
- expr set의 개수가 1보다 크면 **VAR_SAMP(expr)** 값을 반환한다.

자세한 내용은 **STDDEV** 를 참조한다.

사용 예

```
gSQL> SELECT VARIANCE(c1) FROM t1;
VARIANCE(C1)
-----
          132.2
1 row selected.
gSQL> SELECT VARIANCE(ALL c1) FROM t1;
VARIANCE(ALL C1)
-----
          132.2
1 row selected.
gSQL> SELECT VARIANCE(DISTINCT c1) FROM t1;
VARIANCE(DISTINCT C1)
-----
          176.25
1 row selected.
```

VERSION

구문

VERSION()

설명

제품의 version string을 얻는다.

사용 예

```
gSQL> SELECT VERSION() FROM dual;
```

```
VERSION()
```

```
-----
```

```
Release Name.X.X.X revision(XXXXX)
```

```
1 row selected.
```

WIDTH_BUCKET

구문

```
WIDTH_BUCKET( num, min, max, cnt )
```

설명

WIDTH_BUCKET 함수는 명시된 min, max 범위에서 cnt와 동일한 넓이를 갖는 구간을 생성하고, num이 속하는 구간의 위치를 반환한다.

인자 num, min, max, cnt에는 숫자 타입이 올 수 있다.

min, max는 구간에 대한 범위를 의미하며, min, max 값이 같은 경우에는 에러를 반환한다.

cnt는 구간 개수를 의미하고 양의 정수이어야 하며 0 이거나 음수인 경우에는 에러를 반환한다.

구간의 위치에는 1부터 시작하는 번호가 부여된다.

num, min, max, cnt 중 하나라도 NULL인 경우, 결과값도 NULL이다.

사용 예

```
gSQL> SELECT WIDTH_BUCKET( 5, 1, 20, 5 ) AS RESULT FROM DUAL;
```

```
RESULT
```

```
-----
```

```
2
```

```
1 row selected.
```


12.

SQL Languages

Structured Query Language (SQL)는 다음과 같이 구분할 수 있다.

- Data Definition Language: 데이터 정의 언어
- Data Manipulation Language: 데이터 조작 언어
- Data Query Language: 데이터 질의 언어
- Control Language: 제어 언어

12.1 Data Definition Language

DDL 관련 구문

관련 내용은 다음을 참조한다.

- Non-schema object DDL
 - 데이터베이스 관련 구문
 - Profile 관련 구문
 - Audit Policy 관련 구문
 - Authorization 관련 구문
 - Schema 관련 구문
 - Tablespace 관련 구문
- SQL schema object DDL
 - 테이블 관련 구문
 - Index 관련 구문
 - View 관련 구문
 - Sequence 관련 구문
 - Synonym 관련 구문
- Cluster object DDL
 - Cluster System 관련 구문
 - Cluster Group 관련 구문
 - Cluster Member 관련 구문
 - Cluster Location 관련 구문
 - Global Secondary Index 관련 구문

DDL 개념

Data Definition Language (DDL)는 SQL 객체를 생성 (CREATE), 제거 (DROP), 변경 (ALTER)하는 SQL 언어이다. 데이터베이스를 구성하는 SQL 객체는 다음 표와 같으며 각 객체에 대한 자세한 설명은 참조 링크 항목을 참조한다.

표 12-1 SQL 객체의 종류

구분	객체	객체 설명	참조 링크
	Profile	Password 관리 정책을 정의한 객체	Profile
	Audit policy	SQL 감사 정책을 정의한 객체	Audit Policy

구분	객체	객체 설명	참조 링크
Non-schema 객체	User	권한의 집합으로 이루어진 사용자 객체	Authorization
	Schema	테이블 등 SQL schema 객체를 포함하는 논리적 위치	Schema
	Tablespace	테이블, 인덱스 등의 객체를 저장하는 물리적 저장소	Tablespace
	Public synonym	공용 synonym	Public Synonym
SQL schema 객체	Table	데이터가 저장되는 물리적 릴레이션	Table
	View	질의로 구성된 논리적 릴레이션	View
	Index	질의 성능을 향상시키기 위한 색인 객체	Index
	Sequence	순차 번호를 생성하는 객체	Sequence
	Synonym	객체에 대한 대체명을 선언한 객체	Synonym
	Stored procedure	사용자 정의 procedure 객체	Stored Procedure
	Stored function	사용자 정의 function 객체	Stored Function
Cluster 객체	Cluster group	Cluster member의 집합	Cluster Group
	Cluster member	Cluster system을 구성하는 data server	Cluster Member
	Cluster location	Cluster member의 위치 정보 객체	Cluster Location
	Shard	Cluster table을 수평으로 분할한 row들의 집합	Cluster Table과 Shard
	Global secondary index	Cluster의 row 식별자를 위한 색인	Global Secondary Index

DDL과 트랜잭션

GOLDILOCKS에서는 데이터를 추가 (INSERT), 삭제 (DELETE), 갱신 (UPDATE)하는 DML 뿐 아니라, 객체를 생성 (CREATE), 제거 (DROP), 변경 (ALTER)하는 DDL 역시 트랜잭션에 포함된다. 많은 DBMS에서 DDL을 실행할 때 암묵적으로 COMMIT을 수행하는 반면에 GOLDILOCKS에서는 DDL도 트랜잭션에 포함시켜 트랜잭션의 원자성 (atomicity)과 일관성 (consistency)을 보장한다.

이러한 기능은 데이터베이스 migration 작업이나 tool 설치와 같은 배치성 DDL을 원자적으로 수행해야 하는 경우나, 사용자가 실수로 DROP TABLE, TRUNCATE TABLE과 같은 구문을 실행하는 바람에 rollback을 통해 이를 복구해야 하는 경우에 유용하다.

DDL 구문의 속성이 auto-commit인 경우, 구문을 수행할 때 자동으로 commit을 수행하며 auto-commit 이 아닌 경우에는 구문을 수행한 후에도 트랜잭션을 rollback 할 수 있다. DDL의 auto-commit 여부는 **V\$SQL_COMMAND** view를 사용하여 다음과 같이 조회할 수 있다.

```
gSQL>
SELECT command, auto_commit
FROM V$SQL_COMMAND
WHERE is_ddl = 'YES';
```

COMMAND	AUTO_COMMIT
ALTER AUDIT POLICY	YES
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER	YES
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER	YES
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	YES
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS	YES
ALTER DATABASE RESET LOCAL CLUSTER MEMBER	YES
ALTER DATABASE ADD LOGFILE GROUP	YES
ALTER DATABASE ADD LOGFILE MEMBER	YES
ALTER DATABASE DROP LOGFILE GROUP	YES
ALTER DATABASE DROP LOGFILE MEMBER	YES
ALTER DATABASE RENAME LOGFILE	YES
ALTER DATABASE ARCHIVELOG	YES
ALTER DATABASE NOARCHIVELOG	YES
ALTER DATABASE CLEAR PASSWORD HISTORY	NO
ALTER FUNCTION	NO
ALTER INDEX AGING	NO
ALTER INDEX .. STORAGE	NO
ALTER INDEX .. RENAME	NO
ALTER PROCEDURE	NO
ALTER PROFILE	YES
ALTER SEQUENCE	YES
ALTER SYSTEM SWITCH LOGFILE	YES
ALTER TABLE .. ADD COLUMN	NO
ALTER TABLE .. SET UNUSED COLUMN	NO
ALTER TABLE .. ALTER COLUMN .. SET DEFAULT	NO
ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT	NO
ALTER TABLE .. ALTER COLUMN .. SET NOT NULL	NO
ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL	NO
ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE	YES
ALTER TABLE .. ALTER COLUMN .. AS IDENTITY	NO
ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY	NO
ALTER TABLE .. RENAME COLUMN	NO
ALTER TABLE .. STORAGE	NO
ALTER TABLE .. ADD CONSTRAINT	NO
ALTER TABLE .. ALTER CONSTRAINT	NO
ALTER TABLE .. DROP CONSTRAINT	NO
ALTER TABLE .. RENAME CONSTRAINT	NO
ALTER TABLE .. RENAME TO ..	NO
ALTER TABLE .. REBALANCE ..	YES

ALTER TABLE .. MOVE SHARD .. TO CLUSTER GROUP ..	YES
ALTER TABLE .. SPLIT SHARD .. INTO .. AT CLUSTER GROUP ..	YES
ALTER TABLE .. RENAME SHARD .. TO ..	NO
ALTER TABLE .. ADD SUPPLEMENTAL LOG	NO
ALTER TABLE .. ADD GLOBAL SECONDARY INDEX	NO
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX	NO
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX AGING	NO
ALTER TABLE .. DROP GLOBAL SECONDARY INDEX	NO
ALTER TABLE .. DROP SUPPLEMENTAL LOG	NO
ALTER TABLE .. READ ONLY	YES
ALTER TABLE .. READ WRITE	YES
ALTER TABLESPACE .. ADD	YES
ALTER TABLESPACE .. DROP	YES
ALTER TABLESPACE .. ONLINE	YES
ALTER TABLESPACE .. OFFLINE	YES
ALTER TABLESPACE .. RENAME TO	YES
ALTER TABLESPACE .. RENAME { DATAFILE MEMORY }	YES
ALTER USER	YES
ALTER USER .. IDENTIFIED BY	YES
ALTER VIEW	NO
ANALYZE SYSTEM COMPUTE STATISTICS	NO
ANALYZE SYSTEM DELETE STATISTICS	NO
ANALYZE TABLE .. [COMPUTE ESTIMATE] STATISTICS	YES
ANALYZE TABLE .. DELETE STATISTICS	NO
AUDIT POLICY	YES
COMMENT ON .. IS	NO
CREATE AUDIT POLICY	YES
CREATE CLUSTER GROUP	YES
CREATE FUNCTION	NO
CREATE INDEX	NO
CREATE PROCEDURE	NO
CREATE PROFILE	YES
CREATE SCHEMA	YES
CREATE SEQUENCE	YES
CREATE SYNONYM	NO
CREATE TABLE	NO
CREATE TABLE ... AS SELECT	NO
CREATE TABLESPACE	YES
CREATE USER	YES
CREATE VIEW	NO
DROP AUDIT POLICY	YES

DROP CLUSTER GROUP	YES
DROP FUNCTION	NO
DROP INDEX	NO
DROP PROCEDURE	NO
DROP PROFILE	YES
DROP SCHEMA	YES
DROP SEQUENCE	YES
DROP SYNONYM	NO
DROP TABLE	NO
DROP TABLESPACE	YES
DROP USER	YES
DROP VIEW	NO
GRANT .. ON DATABASE	NO
GRANT .. ON TABLESPACE	NO
GRANT .. ON SCHEMA	NO
GRANT .. ON TABLE	NO
GRANT USAGE ON ..	NO
GRANT .. ON PROCEDURE	NO
NOAUDIT POLICY	YES
REVOKE .. ON DATABASE	NO
REVOKE .. ON TABLESPACE	NO
REVOKE .. ON SCHEMA	NO
REVOKE .. ON TABLE	NO
REVOKE USAGE ON ..	NO
REVOKE .. ON PROCEDURE	NO
TRUNCATE TABLE	NO
106 rows selected.	

다음은 테이블 관련 DDL 구문이 트랜잭션에 포함되어 commit 또는 rollback 되는 경우와 다른 트랜잭션에 미치는 영향에 대해 설명하는 예들이다. 다음 예들을 통해 DDL을 포함한 트랜잭션도 트랜잭션의 원자성 (atomicity)을 보장하며 DDL을 포함한 트랜잭션이 commit 되기 전이거나 rollback 될 경우 다른 트랜잭션들이 영향을 받지 않는 트랜잭션의 읽기 일관성 (consistency)을 보장한다는 것을 확인할 수 있다.

객체 생성과 트랜잭션

- 테이블 생성 트랜잭션이 commit/ rollback 되기 전

다음 예와 같이 테이블을 생성한 후에 트랜잭션을 commit 하지 않은 경우, DDL을 수행한 트랜잭션 내에서는 테이블의 데이터를 조작할 수 있지만 다른 트랜잭션에서는 테이블을 생성한 트랜잭션이 commit 될 때까지 해당 테이블을 조회할 수 없다. 즉, INSERT 구문과 마찬가지로 CREATE TABLE 구문도 트랜잭션이 commit 되기 전까지 다른 트랜잭션에서 조회할 수 없다.

- 트랜잭션 A: t1 테이블을 생성한 후 트랜잭션을 commit 하지 않는다.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

위와 같이 트랜잭션 A에서 t1 테이블을 생성하고 commit 하지 않은 경우, 아래와 같이 다른 세션에서 수행하는 트랜잭션 B에서는 t1 테이블을 조회할 수 없으며 아직 commit 되지 않은 t1 테이블과 동일한 이름의 테이블을 생성할 수 없다.

- 트랜잭션 B: 트랜잭션 A가 commit 되기 전까지 t1 테이블을 조회할 수 없다.

```
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
          *
ERROR at line 1:
```

- 트랜잭션 A가 rollback 되기 전까지 t1 테이블을 생성할 수 없다.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
ERR-HYT00(14026): resource busy or timeout expired
```

- 테이블 생성 트랜잭션이 commit 된 후

트랜잭션 A가 commit 될 경우, 다음과 같이 트랜잭션 B는 t1 테이블을 조회할 수 있고 테이블 생성 구문은 t1 테이블이 존재하고 있음을 알리는 유효성 에러를 반환한다.

- 트랜잭션 B: 트랜잭션 A가 commit 된 후에 다음과 같이 테이블을 조회할 수 있다.

```
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

- 트랜잭션 A가 commit 된 후에 다음과 같이 유효성 에러를 반환한다.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
ERR-42000(16005): name 'PUBLIC.T1' is already used by an existing object :
CREATE TABLE t1 ( emp_no INTEGER )
      *
ERROR at line 1:
```

- 테이블 생성 트랜잭션이 rollback 된 후

트랜잭션 A가 rollback 될 경우, t1 테이블의 생성 역시 rollback 되며 다음과 같이 트랜잭션 B가 t1 테이블을 생성할 수 있다.

- 트랜잭션 B: 트랜잭션 A가 rollback 되어 t1 테이블 생성 이전과 동일한 상태가 된다.

```
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
      *
ERROR at line 1:
```

- 트랜잭션 A가 rollback 되어 t1 테이블을 생성할 수 있다.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
Table created.
```

객체 제거와 트랜잭션

- 테이블 제거 트랜잭션이 commit/ rollback 되기 전

테이블을 제거하고 트랜잭션을 commit 하지 않은 경우, DROP TABLE 구문을 수행한 트랜잭션이 commit 되기 전까지 다른 트랜잭션에서는 제거된 테이블을 조회할 수 있다. 즉, DELETE 구문과 마찬가지로 DROP TABLE 구문도 트랜잭션이 commit 되기 전까지, 다른 트랜잭션에서는 테이블이 제거되기 이전 상태를 조회하게 된다.

다음은 트랜잭션 A가 t1 테이블을 제거하고 새로운 테이블 t1을 생성한 후 트랜잭션을 commit 하지 않은 상태이다.

- 트랜잭션 A: DROP 하기 전 t1 테이블에 두 개의 column을 가진 row 하나가 있다.

```
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

- 기존 테이블 t1을 제거한다.

```
gSQL> DROP TABLE t1;
```

```
Table dropped.
```

- 새로운 테이블 t1을 생성한다.

```
gSQL> CREATE TABLE t1 ( addr VARCHAR(128) );
```

```
Table created.
```

- 새로운 테이블 t1에 새로운 row를 생성한다.

```
gSQL> INSERT INTO t1 VALUES ( 'Seoul, Korea' );
```

```
1 row created.
```

```
gSQL> SELECT * FROM t1;
```

```
ADDR
```

```
-----
```

```
Seoul, Korea
```

```
1 row selected.
```

트랜잭션 A가 commit 되지 않은 상태에서, 다음과 같이 트랜잭션 B가 조회를 수행하면 트랜잭션 A가 실행되기 이전의 상태 정보가 조회된다. 즉, DELETE 구문과 마찬가지로 DROP TABLE 구문도 트랜잭션이 commit 되기 전까지는 다른 트랜잭션에 영향을 미치지 않는다.

- 트랜잭션 B: 트랜잭션 B는 트랜잭션 A가 수행되기 이전의 테이블을 조회한다.

```
gSQL> SELECT * FROM t1;
```

```
ID NAME
```

```
-- -----
```

```
1 leekmo
```

```
1 row selected.
```

- 테이블 제거 트랜잭션이 commit 된 후

트랜잭션 A가 commit 될 경우, 다음과 같이 트랜잭션 B가 새로 생성된 t1 테이블을 조회한다.

- Transaction B: 트랜잭션 A가 commit 된 후에 새로 생성된 테이블 t1을 조회한다.

```
gSQL> SELECT * FROM t1;
```

```
ADDR
```

```
-----
```

```
Seoul, Korea
```

```
1 row selected.
```

- 테이블 제거 트랜잭션이 rollback 된 후

트랜잭션 A가 rollback 될 경우, 다음과 같이 트랜잭션 B는 트랜잭션 A가 수행되기 이전의 테이블 t1을 조회한다. 즉, 트랜잭션 A가 rollback 될 경우 트랜잭션 B가 조회하는 데이터는 rollback 된 트랜잭션의 영향을 받지 않는다.

- 트랜잭션 B: 트랜잭션 A가 rollback 된 경우, 트랜잭션 B는 트랜잭션 A를 수행하기 이전의 상태 정보를 조회한다.

```
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

객체 변경과 트랜잭션

테이블 생성, 제거와 마찬가지로 테이블의 구조를 변경하는 ALTER TABLE 구문 역시 트랜잭션의 원자성 (atomicity) 과 일관성 (consistency)을 보장한다. 다음과 같이 테이블에 column을 추가한 트랜잭션이 commit 되기 전까지 다른 트랜잭션은 기존의 테이블을 조회하게 된다. 즉, UPDATE 구문과 마찬가지로 ALTER TABLE 구문도 트랜잭션이 commit 되기 전까지 다른 트랜잭션이 조회를 수행하면 DDL 수행 전의 정보가 조회된다.

- Transaction A: 새로운 UPDATE_TIME column을 추가한다.

```
gSQL> ALTER TABLE t1 ADD COLUMN ( update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
Table altered.
```

- 추가된 column을 포함하는 t1 테이블이 조회된다.

```
gSQL> select * from t1;
ID NAME    UPDATE_TIME
-- -----
1 leekmo 2014-07-10 12:50:33.540495
1 row selected.
```

다음과 같이 트랜잭션 A를 commit 하기 전에 트랜잭션 B를 수행하면 column이 추가되기 이전의 테이블이 조회된다.

- Transaction B: 트랜잭션 A가 commit 되지 않았으므로 추가된 column UPDATE_TIME은 조회되지 않는다.

```
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```


12.2 Data Manipulation Language

DML 관련 구문

관련 내용은 다음을 참조한다.

- INSERT 관련 구문
 - INSERT INTO
 - INSERT INTO name RETURNING
 - INSERT INTO name RETURNING .. INTO
- UPDATE 관련 구문
 - UPDATE
 - UPDATE name RETURNING
 - UPDATE name RETURNING .. INTO
 - UPDATE name WHERE CURRENT OF cursor_name
- DELETE 관련 구문
 - DELETE FROM
 - DELETE FROM name RETURNING
 - DELETE FROM name RETURNING .. INTO
 - DELETE FROM name WHERE CURRENT OF cursor_name
- SELECT 관련 구문: SELECT .. INTO
- Dynamic SQL 관련 구문
 - EXECUTE IMMEDIATE 'sql_string'
 - PREPARE statement_name
 - EXECUTE statement_name

DML 개념

Data Manipulation Language (DML)는 추가 (INSERT), 삭제 (DELETE), 갱신 (UPDATE)과 같이 테이블 데이터를 조작하거나 질의하는 SQL 언어이다.

질의에 대해서는 **Data Query Language** 장에서 다루며 본 장에서는 데이터를 갱신하는 DML 구문만 다룬다.

DDL 구문이 SQL 객체의 구조를 변경하는 언어인 반면에 DML은 객체의 내용을 조작하는 언어이다. 예를 들어 ALTER TABLE 구문은 테이블의 구조를 변경하는 반면에 INSERT 구문은 테이블에 하나 이상의 row 를 추가하는 구문이다.

테이블에 데이터 추가, 삭제, 갱신 등을 수행하는 데이터 조작 구문은 크게 다음과 같이 구분할 수 있다.

표 12-2 데이터 조작 구문

구분	구문 유형	설명
추가	INSERT .. VALUES	하나의 row를 테이블에 추가한다.
	INSERT .. SELECT	검색 결과를 테이블에 추가한다.
	INSERT .. RETURN .. INTO	추가한 row의 값을 변수로 설정한다.
	INSERT .. RETURN	질의 결과로 추가한 row들을 검색한다.
삭제	DELETE .. WHERE	조건에 부합하는 row들을 삭제한다.
	DELETE .. WHERE CURRENT OF	커서 위치에 있는 row를 삭제한다.
	DELETE .. RETURN .. INTO	삭제한 row의 값을 변수로 설정한다.
	DELETE .. RETURN	질의 결과로 삭제한 row들을 검색한다.
갱신	UPDATE .. WHERE	조건에 부합하는 row들을 갱신한다.
	UPDATE .. WHERE CURRENT OF	커서의 위치에 있는 row를 갱신한다.
	UPDATE .. RETURN .. INTO	갱신한 row의 값을 변수로 설정한다.
	UPDATE .. RETURN	질의 결과로 갱신한 row들을 검색한다.

데이터 추가

테이블에 데이터를 추가할 때 row 단위로 데이터를 추가한다. INSERT 구문을 통해 하나 이상의 row를 추가할 수 있으며, 일부 column의 데이터를 생략하더라도 row의 모든 column에 데이터가 완성된 형태로 테이블에 row를 추가한다.

다음 예제 테이블을 참조한다.

```
CREATE TABLE t1
(
    id    NUMBER(10,0),
    name  VARCHAR(128),
    addr  VARCHAR(1024) DEFAULT 'n/a'
);
```

가장 기본적인 row 추가 방법은 다음과 같다.

```
INSERT INTO t1 VALUES ( 1, 'leekmo', 'Seoul, Korea' );
```

VALUES 절에 나열된 값들은 테이블을 생성할 때 나열한 column 순서와 동일하게 입력된다. 그러나 위와 같은 구문에 column이 추가되거나 제거될 경우 예기치 않은 오류가 발생할 수 있어 다음 구문과 같이 column 이름을 명시하는 것이 바람직하다.

```
INSERT INTO t1 (id, name, addr) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
INSERT INTO t1 (name, addr, id) VALUES ( 'leekmo', 'Seoul, Korea', 1 );
```

위의 두 INSERT 구문은 column의 순서만 다르게 나열했을 뿐, 동일한 데이터를 가진 row를 추가한 구문이다.

테이블의 모든 column을 나열하지 않을 경우, 지정하지 않은 column에는 column의 기본값을 설정하여 row를 완성한다. 다음과 같이 구문 내에서 사용되지 않은 addr column에는 테이블을 생성할 때 정의한 기본값인 'n/a'가 저장된다.

```
INSERT INTO t1 ( id, name ) VALUES ( 1, 'leekmo' );
INSERT INTO t1 ( id, name ) SELECT id, name FROM emp;
```

Column의 기본값을 명시적으로 사용하고자 할 경우, 다음과 같이 DEFAULT를 명시할 수 있다.

```
INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', DEFAULT );
```

모든 column에 기본값을 사용하고자 할 경우, 다음과 같이 두 가지 형태의 구문을 사용할 수 있다.

```
INSERT INTO t1 ( id, name, addr ) VALUES ( DEFAULT, DEFAULT, DEFAULT );
INSERT INTO t1 DEFAULT VALUES;
```

다음과 같이 하나의 INSERT 구문을 이용해 다수의 row를 추가할 수 있다. 아래 구문은 하나의 INSERT 구문으로 세 개의 새 row들을 추가하는 예이다.

```
INSERT INTO t1 (id, name, addr) VALUES
( 1, 'leekmo', 'Seoul, Korea' ),
( 2, 'mkkim', 'Seoul, Korea' ),
( 3, 'xcom', 'Inchon, Korea' );
```

다음과 같이 SELECT 질의 결과를 이용해 다수의 row들을 추가할 수 있다. 아래 예는 입사한 지 3년 이상인 직원들을 구하여 테이블 t1에 추가하는 구문이다.

```
INSERT INTO t1 ( id, name, addr )
SELECT id, name, addr
FROM emp
WHERE DATEDIFF( YEAR, SYSDATE, join_date ) >= 3;
```

데이터 삭제

데이터를 추가할 때와 마찬가지로 데이터를 삭제할 때도 row 단위로 테이블에서 제거한다. Row를 삭제할 때는 WHERE 조건을 이용하거나 row의 id (ROWID)를 이용할 수 있다.

다음은 WHERE 조건을 만족하는 row들을 삭제하는 예이다.

```
DELETE FROM t1 WHERE id = 1;
```

다음은 ROWID 값을 이용해 해당 row를 삭제하는 예이다.

```
gSQL> SELECT rowid FROM t1 WHERE id = 1;
                ROWID
-----
AAAAAAAAFNHAACAAAAiAAA
1 row selected.
gSQL> DELETE FROM t1 WHERE ROWID = 'AAAAAAAAFNHAACAAAAiAAA';
1 row deleted.
```

다음과 같이 WHERE 절이 없는 DELETE 구문은 테이블의 전체 row들을 삭제한다. WHERE 절이 없는 DELETE 구문은 전체 row를 제거한다는 점에서 TRUNCATE TABLE 구문과 기능이 동일하지만 TRUNCATE TABLE 구문을 사용할 것을 권장한다.

```
DELETE FROM t1;
TRUNCATE TABLE t1;
```

데이터 갱신

UPDATE 구문을 이용해 데이터를 갱신한다. 하나 이상의 row 들을 갱신할 수 있으며 하나 또는 다수의 column들을 갱신할 수 있다. UPDATE 구문에 기술되지 않은 다른 column들은 영향을 받지 않는다.

다음은 조건에 부합하는 row들의 한 column을 갱신하는 예이다.

```
UPDATE t1 SET page_view = page_view + 1 WHERE id = 1;
```

다음은 다수의 column을 갱신하는 UPDATE 구문으로써 두 구문의 의미는 동일하다.

```
UPDATE t1 SET page_view = page_view + 1, status = 'F' WHERE id = 1;
UPDATE t1 SET (page_view, status) = (page_view + 1, 'F') WHERE id = 1;
```

Column 값을 기본값으로 설정하려면 다음과 같이 DEFAULT를 사용한다.

```
UPDATE t1 SET addr = DEFAULT WHERE id = 1;
```

커서를 이용한 데이터 조작

커서는 질의를 수행하고 질의 결과를 조작하는 세션 객체이다. 커서를 이용하여 질의 결과 집합을 갱신 하거나 삭제할 수 있다.

다음은 updatable cursor를 선언하고 이를 이용하여 현재 cursor 위치의 row를 갱신하거나 삭제하는 예이다.

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1 FOR UPDATE;
```

```
Cursor declared.
```

```
gSQL> OPEN cur1;
```

```
Cursor is open.
```

```
gSQL> \var v_id INTEGER
```

```
gSQL> \var v_data VARCHAR(128)
```

```
gSQL> FETCH cur1 INTO :v_id, :v_data;
```

```
V_ID V_DATA
```

```
----
```

```
1 data_1
```

```
1 row fetched.
```

```
gSQL> FETCH cur1 INTO :v_id, :v_data;
```

```
V_ID V_DATA
```

```
----
```

```
2 data_2
```

```
1 row fetched.
```

```
gSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
```

```
1 row deleted.
```

```
gSQL> FETCH cur1 INTO :v_id, :v_data;
```

```
V_ID V_DATA
```

```
----
```

```
3 data_3
```

```
1 row fetched.
```

```
gSQL> UPDATE t1 SET id = id + :v_id WHERE CURRENT OF cur1;
```

```
1 row updated.
```

```
gSQL> COMMIT;
```

```
Commit complete.
```

```
gSQL> SELECT * FROM t1 ORDER BY 1;
```

```
ID DATA
```

```
--
```

```
1 data_1
```

```
6 data_3
```

```
2 rows selected.
```

위의 예와 같이 **DECLARE cursor_name** 구문을 이용하여 FOR UPDATE 커서를 선언하고 **OPEN cursor_name** 구문을 이용하여 해당 커서를 연다. **FETCH cursor_name** 구문을 이용해 커서를 원하는 위치로 이동시키고 **DELETE FROM name WHERE CURRENT OF cursor_name** 구문을 이용하여 해당 위치의 row를 삭제하거나 **UPDATE name WHERE CURRENT OF cursor_name** 구문을 이용하여 row를 갱신할 수 있다.

FOR UPDATE 커서는 **CLOSE cursor_name** 구문을 이용하여 닫거나 commit을 수행할 때 트랜잭션 종료와 함께 닫힌다.

DML Query

데이터를 갱신하는 DML 구문을 수행할 때 RETURNING 절을 이용하여 갱신된 데이터를 조회할 수 있다. DML에 대한 RETURNING 절은 SELECT와 마찬가지로 data 조회를 가능하게 하므로 DML 구문과 SELECT 구문 두 가지 모두 수행할 것을 하나의 DML query로 대체할 수 있다.

다음은 **INSERT INTO name RETURNING** 구문을 이용하여 데이터를 추가하고 해당 결과를 조회하는 예이다. 다음 예와 같이 SYSDATE 함수를 통해 입력된 join_date 값을 하나의 구문으로 조회할 수 있다.

```
gSQL> INSERT INTO t1 ( id, join_date ) VALUES ( 1, SYSDATE ) RETURNING id, join_date;
ID JOIN_DATE
--
1 2014-07-18
1 row created.
```

다음은 **DELETE FROM name RETURNING** 구문을 이용하여 데이터를 삭제하고 삭제된 row를 조회하는 예이다. RETURNING 절에 연산을 사용하여 데이터를 가공할 수 있다.

```
gSQL> DELETE FROM t1 RETURNING ( id || ': ' || join_date ) AS id_and_join_date;
ID_AND_JOIN_DATE
-----
1: 2014-07-18
1 row deleted.
```

다음은 **UPDATE name RETURNING** 구문을 이용하여 갱신된 row 값을 기준으로 조회하는 예이다. OLD 절을 이용할 경우 변경되기 이전의 값을 조회할 수 있다.

- Row를 갱신하고 갱신된 값을 조회한다.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURNING page_view;
PAGE_VIEW
-----
102
1 row updated.
```

- Row를 갱신하고 갱신되기 전 값을 조회한다.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURNING OLD page_view;
PAGE_VIEW
-----
      102
1 row updated.
```

DML에 사용하는 RETURNING 절은 SELECT query와 마찬가지로 다수의 질의 결과를 조회할 수 있는 반면에 row 하나에 대해서만 DML을 수행할 경우에는 RETURNING INTO 절을 이용하여 호스트 변수 값을 얻어올 수 있다. 이 경우, **SELECT .. INTO** 구문과 마찬가지로 갱신된 row의 개수가 한 건 이하여야 한다.

다음은 각 DML의 RETURNING .. INTO 절을 이용하여 호스트 변수 값을 설정하는 예이다.

- Host 변수를 선언한다.

```
gSQL> \var v_id          INTEGER
gSQL> \var v_page_view BIGINT
gSQL> \var v_date        DATE
```

- Row를 추가한 후 host 변수에 값을 얻어온다.

```
gSQL> INSERT INTO t1 ( id, join_date ) VALUES ( 1, SYSDATE ) RETURNING join_date INTO :v_date;
V_DATE
-----
2014-07-18 16:57:11.000000
1 row created.
```

- Row를 갱신한 후 host 변수에 값을 얻어온다.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURN page_view INTO :v_page_view;
V_PAGE_VIEW
-----
      101
1 row updated.
```

- Row를 삭제한 후 host 변수에 값을 얻어온다.

```
gSQL> DELETE FROM t1 WHERE id = 1 RETURN id, page_view INTO :v_id, :v_page_view;
V_ID V_PAGE_VIEW
----
    1      101
1 row deleted.
```

DML query와 관련된 자세한 내용은 다음 구문들을 참조한다.

- **INSERT INTO name RETURNING**
- **INSERT INTO name RETURNING .. INTO**
- **DELETE FROM name RETURNING**
- **DELETE FROM name RETURNING .. INTO**
- **UPDATE name RETURNING**
- **UPDATE name RETURNING .. INTO**

12.3 Data Query Language

Query 관련 구문

관련 내용은 다음을 참조한다.

- SELECT query 관련 구문
 - SELECT
 - SELECT .. FOR UPDATE
- DML query 관련 구문
 - INSERT INTO name RETURNING
 - UPDATE name RETURNING
 - DELETE FROM name RETURNING
- Cursor 관련 구문
 - DECLARE cursor_name
 - OPEN cursor_name
 - FETCH cursor_name
 - CLOSE cursor_name
 - DELETE FROM name WHERE CURRENT OF cursor_name
 - UPDATE name WHERE CURRENT OF cursor_name

Query 개념

Query는 하나 이상의 table 또는 view의 데이터를 검색하는 일련의 연산이다. Query를 이용하여 저장되어 있는 데이터 중 원하는 조건을 만족하는 데이터만 원하는 형태로 가공하여 가져올 수 있다.

Query는 ';'으로 구분되는 모든 SQL 구문 중에 최상위에 있는 SELECT 구문이다. 최상위 SELECT 구문은 그 안에 다시 SELECT 구문을 포함할 수 있는데 이 때 최상위 SELECT 구문에 포함된 하위 SELECT 구문들을 subquery라고 한다.

GOLDILOCKS에서 query는 크게 SELECT query와 DML query, cursor로 나뉜다. SELECT query는 SELECT 구문을 사용하여 결과를 반환하는 query이며, DML query는 INSERT, DELETE, UPDATE 구문에 RETURNING 구문을 사용하여 결과를 반환하는 query이다. Cursor는 한 번 질의한 결과 집합을 임시로 저장하고 저장된 결과 집합에서 임의의 row에 접근하여 원하는 결과를 가져오는 query이다. SELECT query와 DML query는 한 번의 질의로 결과를 얻을 수 있다. 하지만 cursor는 DECLARE cursor로 명시된 SELECT 구문을 OPEN cursor에서 수행하고 그 결과 집합을 CLOSE cursor를 호출할 때까지 유지하여 FETCH cursor로 결과 집합의 임의 row에 접근해가며 반복적으로 원하는 결과를 얻는다.

본 장에서는 SELECT query와 DML query, cursor에 대해 설명한다.

기본 Query

가장 기본적인 query의 형태는 SELECT <select list> FROM <table expression>이다. SELECT 키워드와 FROM 키워드 사이에 존재하는 <select list>에는 <table expression>에 기술한 table 또는 view에 대해 반환되는 결과 row들에 포함될 하나 이상의 column 또는 expression들을 명시한다.

```
SELECT n_name
       , INITCAP( n_name )
FROM nation
WHERE n_regionkey = 1;
```

N_NAME	INITCAP(N_NAME)
ARGENTINA	Argentina
BRAZIL	Brazil
CANADA	Canada
PERU	Peru
UNITED STATES	United States

5 rows selected.

<table expression>에는 하나 이상의 table 또는 view들을 기술할 수 있는데 이 때 두 table 또는 view에 동일한 column이 존재할 수 있다. 이 경우, <select list>에 해당 column을 기술하려면 해당 table 또는 view의 이름을 함께 기술하여야 한다. Table이나 column을 기술할 때 schema 이름과 table 이름을 명확하게 기술해주는 것이 좋다.

- 잘못된 예

```
SELECT n_name
FROM nation AS n
       , v_nation AS v
WHERE n.n_nationkey = v.n_nationkey
      AND v.n_regionkey = 1;
```

ERR-42000(16142): column ambiguously defined :

```
SELECT n_name
      *
```

ERROR at line 1:

- 올바른 예

```

SELECT n.n_name
  FROM nation AS n
       , v_nation AS v
 WHERE n.n_nationkey = v.n_nationkey
       AND v.n_regionkey = 1;
N_NAME
-----
ARGENTINA
BRAZIL
CANADA
PERU
UNITED STATES
5 rows selected.

```

〈select list〉는 alias name을 지원한다. 이는 콤마 (,) 리스트로 구분된 각 column들 중에 출력할 column 이름을 갱신한다. Alias name은 〈order by clause〉에서만 사용할 수 있고 그 외의 구문에서는 사용할 수 없다.

```

SELECT p_type
       , p_retailprice * 0.9 AS discount_price
  FROM part
 ORDER BY discount_price
 FETCH 5;
   2      3      4      5
P_TYPE          DISCOUNT_PRICE
-----
PROMO BURNISHED COPPER          810.9
ECONOMY BRUSHED NICKEL          810.9
LARGE BRUSHED BRASS             811.8
LARGE BRUSHED NICKEL            811.8
PROMO ANODIZED STEEL            811.8
5 rows selected.

```

SELECT 키워드와 FROM 키워드 사이에는 〈select list〉 이외에 〈hint clause〉와 〈set quantifier〉가 올 수 있다. **hint clause**는 query 실행 계획을 사용자가 직접 제어하는 구문이다. 〈set quantifier〉는 결과로 반환되는 row들 간에 중복되는 데이터를 제거하는 구문으로써 자세한 내용은 **query specification** 절을 참조한다.

- hint 사용 예

```

SELECT
  /*+ INDEX( part ) */
  p_type
  , p_retailprice

```

```

FROM part
WHERE p_partkey = 100;
P_TYPE                P_RETAILPRICE
-----
ECONOMY ANODIZED TIN      1000.1
1 row selected.

```

- <set quantifier> 사용 예

```

SELECT DISTINCT
    o_orderpriority
FROM orders;
O_ORDERPRIORITY
-----
5-LOW
2-HIGH
3-MEDIUM
1-URGENT
4-NOT SPECIFIED
5 rows selected.

```

SET 연산자

SET 연산자는 둘 이상의 query들의 결과 집합을 하나의 결과 집합으로 결합한다. SET 연산자에는 UNION, EXCEPT, INTERSECT 등이 있으며 EXCEPT와 동일하게 동작하는 MINUS 연산자를 제공한다. 각각의 SET 연산자에는 ALL이 나 DISTINCT와 같은 추가 옵션이 있는데 생략할 경우 DISTINCT로 간주한다.

SET 연산자를 이용하여 둘 이상의 query들을 기술하는 경우 기본적으로 왼쪽에 기술한 query부터 오른쪽에 기술한 query 순으로 순차적으로 처리하며 괄호를 사용하여 처리 순서를 명확하게 지정한 경우 해당 query부터 처리한다.

```

SELECT n_name
FROM nation
WHERE n_nationkey < 10
INTERSECT
( SELECT n_name
  FROM nation
  WHERE n_regionkey = 1
UNION ALL
SELECT n_name
FROM nation

```

```

        WHERE n_regionkey = 2 );
N_NAME
-----
BRAZIL
ARGENTINA
INDONESIA
INDIA
CANADA
5 rows selected.

```

SET 연산자에 기술된 각 query들은 모두 동일한 개수의 target을 가져야 하며, 각 query와 동일한 위치에 있는 target은 같은 그룹에 속하는 data type을 가져야 한다.

SET 연산자에는 최종 결과 집합을 정렬하기 위한 <order by clause>이 있고 SET 연산자에 속한 각각의 query들에는 query를 자체적으로 정렬하기 위한 <order by clause>가 있을 수 있다.

SET 연산자에 대한 자세한 내용은 **set operator** 절을 참고한다.

조인

조인은 <from clause>에 기술한 둘 이상의 table 또는 view의 각 row들을 결합하는 질의이다. 조인 조건이 없는 조인 연산은 두 table 또는 view의 왼쪽 결과에 있는 모든 row들과 오른쪽 결과의 모든 row들을 각각 row 하나로 결합한 row들을 결과로써 반환한다.

<from clause>에 둘 이상의 table 또는 view를 기술하여 조인할 때, 두 table 또는 view에 같은 이름의 column이 있다면 <select list>와 <where clause> 등의 구문들에 table 이름 등을 사용하여 column을 명확하게 구분해야 한다. 그렇지 않은 경우 validation 에러가 발생한다.

조인은 조인 조건이 있는 경우와 없는 경우로 나뉠 수 있다. 조인 조건이란 조인에 참여하는 서로 다른 두 table 또는 view의 column들을 비교하는 조건을 말한다. 조인 조건이 없으면 두 table 또는 view에 있는 각각의 row들을 row 하나로 결합한 결과가 반환되며, 조인 조건이 존재하는 경우 두 table 또는 view에 있는 각각의 row들 중 조인 조건을 만족하는 것들만 row 하나로 결합한 결과가 반환된다.

동등비교(=)를 이용한 조인 조건이 있는 경우 equi-join이라 말한다. 조인 조건 중 equi-join에 해당하는 조건들은 optimizer에서 join을 위한 최적화를 하는데 중요한 요소이다.

조인 연산 중에 <from clause>에 동일한 table만 존재하는 경우 self-join이라 말하며, <select list> 등에 column을 기술하기 위하여 각 table에 alias 이름을 기술하고 column에 table의 alias를 이용한다.

CROSS JOIN

CROSS JOIN은 조인 조건이 존재하지 않는 조인 연산으로써 cartesian product라고도 한다. CROSS JOIN은 table 또는 view의 row들을 각각 다른 table 또는 view의 row들과 결합한 결과를 반환한다.

```
SELECT a.r_name
       , b.r_name
  FROM region AS a
       , region AS b
 FETCH 5;
```

R_NAME	R_NAME
AFRICA	AFRICA
AFRICA	AMERICA
AFRICA	ASIA
AFRICA	EUROPE
AFRICA	MIDDLE EAST

5 rows selected.

INNER JOIN

INNER JOIN은 둘 이상의 table 또는 view에 대해 조인 조건을 만족하는 row들을 반환하는 조인 연산이다. INNER JOIN은 <from clause>에 명시적으로 inner join을 명시한 경우와 <from clause>에 콤마 (,) 리스트로 table 또는 view를 나열하고 <where clause>에서 이 table 또는 view에 대한 조인 조건을 기술한 경우 모두를 말한다. <from clause>에 inner join을 명시한 경우 <where clause>에도 조인 조건이 있다면 이 둘을 구분하지 않고 하나의 조인조건으로 묶어서 처리한다.

- INNER JOIN 구문을 사용한 예

```
SELECT n_name
  FROM region INNER JOIN nation ON r_regionkey = n_regionkey
 WHERE r_name = 'AFRICA';
```

N_NAME
ALGERIA
ETHIOPIA
KENYA
MOROCCO
MOZAMBIQUE

5 rows selected.

- 콤마 (,) 리스트로 나열한 예

```
SELECT n_name
  FROM region
       , nation
```

```
WHERE r_regionkey = n_regionkey
      AND r_name = 'AFRICA';
```

```
N_NAME
```

```
-----
```

```
ALGERIA
```

```
ETHIOPIA
```

```
KENYA
```

```
MOROCCO
```

```
MOZAMBIQUE
```

```
5 rows selected.
```

OUTER JOIN

OUTER JOIN이란 둘 이상의 table 또는 view에 대해 조인 조건을 만족하는 row들을 반환하고 추가적으로 OUTER JOIN의 방향에 따라 한쪽 또는 양쪽의 table 또는 view에 대하여 조인 조건을 만족하지 않는 row들을 반환하는 조인 연산이다.

OUTER JOIN에는 LEFT OUTER JOIN과 RIGHT OUTER JOIN, FULL OUTER JOIN이 있다. 세 OUTER JOIN 모두 조인 조건을 만족하는 row를 반환한다. 다만, LEFT OUTER JOIN은 조인 조건을 만족하지 않는 왼쪽 row에 대하여 오른쪽 row 부분을 모두 NULL로 채워 결과로 반환하고 RIGHT OUTER JOIN은 조인 조건을 만족하지 않는 오른쪽 row에 대하여 왼쪽 row 부분을 모두 NULL로 채워 결과로 반환한다. FULL OUTER JOIN의 경우 LEFT OUTER JOIN과 RIGHT OUTER JOIN에서 추가적으로 반환하는 결과들 모두를 결과로 반환한다.

```
SELECT r_name
       , n_name
FROM   region LEFT OUTER JOIN nation
       ON r_regionkey = n_regionkey AND r_name = 'AFRICA';
```

```
R_NAME
```

```
N_NAME
```

```
-----
```

```
AFRICA          ALGERIA
```

```
AFRICA          ETHIOPIA
```

```
AFRICA          KENYA
```

```
AFRICA          MOROCCO
```

```
AFRICA          MOZAMBIQUE
```

```
AMERICA         null
```

```
ASIA            null
```

```
EUROPE          null
```

```
MIDDLE EAST    null
```

```
9 rows selected.
```

GOLDILOCKS에서는 Oracle과의 호환성을 위해 SQL 표준에서는 지원하지 않지만 Oracle에서 지원하는 outer join operator (+)를 지원한다. Outer join operator (+)는 <from clause>에 table들을 콤마 (,) 리스트로 나열하고 <wh

ere clause>에 조인 조건들의 outer node가 될 column에 (+)를 추가한다.

Outer join operator (+)를 사용하는 경우 column에 추가하는 (+)는 다음 예제에서처럼 반드시 column의 오른쪽에 기술하여야 한다.

```
select * from t1, t2 where t1.i1 = t2.i1(+);
```

Outer join operator (+)를 사용하기 위한 구문 규칙은 다음과 같다.

- <join outer operator>는 <where clause> 이외의 구문에 기술할 수 없다.
- <join outer operator>는 <joined table> 대상인 <table reference>의 <column reference>에 대해서만 사용할 수 있다.
- <join outer operator>를 포함한 <value expression>과 OR logical operator를 사용한 다른 조건과 결합될 수 없다.
- <join outer operator>를 포함한 <column reference>를 IN function의 인자로 사용할 수 없다.
- 하나의 <table reference>는 다수의 outer join의 null-generated table로 사용될 수 없다. (Outer join의 제약 조건)
- 두 테이블 이상의 outer join이 실행되는 경우 left outer join 순서로 나열하여 제일 왼쪽부터 outer join을 수행한다.
- 두 테이블 이상의 outer join이 실행되고 한 테이블에 다수의 테이블이 outer join으로 묶이는 경우, optimizer가 계산한 순서대로 outer join을 실행한다.

다음과 같은 경우에는 outer join operator (+)를 기술하더라도 무시된다.

- <join outer operator>는 두 테이블간의 join condition으로 사용할 수 있는 <value expression>에 대해서 사용할 수 있으며, join condition으로 사용할 수 없을 경우 무시되며 이 때 error나 warning은 출력하지 않는다.
- Outer query에 대한 <column reference>에 기술된 <join outer operator>는 무시되고 error나 warning은 출력하지 않는다.
- <join outer operator>를 이용하여 두 <table reference>를 outer join하는 경우, null-generated table들에 속한 모든 <column reference>에 <join outer operator>를 기술해야 한다. 그렇지 않은 경우 두 <table reference>의 join은 inner join으로 간주하여 처리되며, 그에 대한 error나 warning은 출력하지 않는다.

GOLDILOCKS와 Oracle의 outer join operator (+)에는 다음과 같은 차이가 있다.

- Quantified comparison (in, = any, = all = row 등)
 - GOLDILOCKS: 해당 연산을 validation error로 처리한다.
 - Oracle: 해당 연산을 and/ or로 풀어낸 다음 validation 체크를 수행한다.
- And 절 하위에 or 절이 오는 경우
 - GOLDILOCKS: 하위 or 절에 나타나는 column들도 validation에 적용한다.
 - Oracle: 하위 or 절에 나타나는 column들은 validation에서 무시한다.
- 조건절에 부질의 (subquery)가 포함되는 경우
 - GOLDILOCKS: Outer join으로 풀리면서 해당 조건은 join condition으로 처리된다.
 - Oracle: Outer join으로 풀리지만 해당 조건은 where filter로 처리된다.

노트

GOLDILOCKS의 outer join operator (+)는 Oracle과의 호환성을 위해 지원하는 기능이며 <from clause>에 OUTER JOIN을 기술하는 방법을 사용할 것을 권장한다. (참고로 Oracle에서도 <from clause>에 OUTER JOIN을 기술하는 방법을 사용할 것을 권장하고 있다.)

NATURAL JOIN

NATURAL JOIN이란 둘 이상의 table 또는 view의 이름이 같은 column들에 대해 동등비교 (=) 조건을 조인 조건으로 사용하는 조인 연산이다. 동일한 이름의 column들에 대한 조인조건을 내부적으로 만들어 사용한다는 것을 제외하고는 INNER JOIN과 동일하다.

```
SELECT r_name
  FROM region a NATURAL JOIN region b;
```

R_NAME
AFRICA
AMERICA
ASIA
EUROPE
MIDDLE EAST

5 rows selected.

SEMI JOIN

SEMI JOIN이란 오른쪽 row들 중에 조인 조건을 만족하는 row들의 왼쪽 row를 결과로 반환하는 조인 연산이다. SEMI JOIN은 왼쪽 row와 오른쪽 row를 결합한 row를 결과로 반환하는 다른 조인 연산과 달리 왼쪽 row만 결과로 반환한다.

- Semi join이 사용되는 예

```
SELECT r_name
  FROM region
 WHERE r_regionkey IN ( SELECT n_regionkey
                        FROM nation
                        WHERE n_nationkey < 5 );
```

R_NAME
AFRICA
AMERICA

MIDDLE EAST

3 rows selected.

ANTI-SEMI JOIN

ANTI-SEMI JOIN이란 오른쪽 row 들 중에 조인 조건을 만족하지 않는 row들의 왼쪽 row를 결과로 반환하는 조인 연산이다. ANTI-SEMI JOIN은 SEMI JOIN과 마찬가지로 왼쪽 row만 결과로 반환한다.

- Anti-semi join이 사용되는 예

```
SELECT r_name
FROM region
WHERE r_regionkey NOT IN ( SELECT n_regionkey
                           FROM nation
                           WHERE n_nationkey < 5 );
```

R_NAME
ASIA
EUROPE

2 rows selected.

조인 연산에 대한 자세한 내용은 **joined table**절을 참고한다.

결과 집합 그룹 (group by)

하나 이상의 column을 기준으로 동일한 column들을 갖는 row들을 하나의 그룹으로 하여 연산을 처리하기 위해서는 <group by clause>를 사용한다. <group by clause>는 그룹을 구분하기 위해 column들을 콤마 (,)로 나열하며, GOLDILOCKS는 이를 기준으로 그룹에 대한 연산을 수행한다.

```
SELECT
    o_orderpriority
    , MIN( o_totalprice ) AS min_price
    , MAX( o_totalprice ) AS max_price
FROM orders
GROUP BY
    o_orderpriority;
```

O_ORDERPRIORITY	MIN_PRICE	MAX_PRICE
5-LOW	857.71	530604.44
2-HIGH	896.8	522720.61

```

3-MEDIUM          875.52 508668.52
1-URGENT           866.9 544089.09
4-NOT SPECIFIED    884.82 555285.16
5 rows selected.

```

〈group by clause〉를 기술한 경우 〈select list〉에는 〈group by clause〉에 기술한 column들과 집계 함수만 올 수 있다.

〈group by clause〉에 column이 아닌 상수를 기술하거나 괄호만 기술할 경우, 각 row에 동일한 값을 갖는 가상의 column이 있는 empty grouping set로 간주하고 해당 column을 기준으로 그룹화가 진행된다. 이는 〈group by clause〉 없이 〈having clause〉만 기술하는 경우에도 적용된다.

〈having clause〉를 사용하여 〈group by clause〉에 의해 그룹이 구분된 결과들 중에서 특정 row들만 가져오기 위한 조건을 기술할 수 있다. 〈having clause〉는 그룹 각각에 대한 조건을 기술하는데 집계 연산을 이용한 조건 등을 기술할 수 있다.

```

SELECT
    o_orderpriority
  , MIN( o_totalprice ) AS min_price
  , MAX( o_totalprice ) AS max_price
FROM orders
GROUP BY
    o_orderpriority
HAVING
    MIN( o_totalprice ) < 870;
O_ORDERPRIORITY MIN_PRICE MAX_PRICE
-----
5-LOW           857.71 530604.44
1-URGENT        866.9 544089.09
2 rows selected.

```

그룹에 대한 자세한 내용은 **group by clause** 절을 참조한다.

결과 집합 정렬 (order by)

검색된 결과 집합을 원하는 column들을 기준으로 정렬하고 싶은 경우에 〈order by clause〉를 사용한다. 〈order by clause〉는 LONG type을 제외한 모든 column을 기준으로 정렬할 수 있다.

〈order by clause〉에 양의 정수값을 기술하는 경우 〈select list〉에 존재하는 target들 중 정수값에 해당하는 위치의 column을 의미한다. 〈order by clause〉에 기술할 수 있는 양의 정수값의 범위는 1부터 〈select list〉에 존재하는 target의 개수까지이다.

```

SELECT
    o_orderpriority
  , MIN( o_totalprice ) AS min_price
  , MAX( o_totalprice ) AS max_price
FROM orders
GROUP BY
    o_orderpriority
ORDER BY 1;
O_ORDERPRIORITY MIN_PRICE MAX_PRICE
-----
1-URGENT          866.9 544089.09
2-HIGH            896.8 522720.61
3-MEDIUM         875.52 508668.52
4-NOT SPECIFIED   884.82 555285.16
5-LOW             857.71 530604.44
5 rows selected.

```

<order by clause>에 기술한 column의 data type이 numeric data인 경우에는 숫자 비교에 의해 정렬되고 character data인 경우에는 문자 비교에 의해 정렬된다.

<order by clause>의 각 column들은 정렬 방향인 ASC와 DESC 옵션을 사용하여 기술할 수 있으며, 생략할 경우에는 ASC가 기술된 것으로 간주한다.

정렬에 대한 자세한 내용은 **order by clause** 절을 참조한다.

부질의 (Subquery)

부질의는 여러 단계로 구성된 검색 요청을 수행한다. 여러 단계로 구성된 검색 요청이란 현재 query 결과가 하위 query 결과에 따라 결정되는 것을 말한다. 예를 들어 특정 그룹에 속한 사람들의 평균 나이보다 나이가 많은 사람들을 검색하려면 먼저 특정 그룹에 속한 사람들의 평균 나이를 구하는 query를 수행하고, 이를 이용하여 평균 나이보다 많은 사람들을 검색하는 query를 수행하는 식이다.

```

SELECT e_name
FROM emp
WHERE e_age > ( SELECT AVG(e_age)
                FROM emp
                WHERE e_dept = 'RND' );

```

부질의는 <from clause>와 <where clause>에 사용될 수 있으며, <from clause>에 사용되는 부질의를 inline view라 하고, <where clause>에 사용되는 부질의를 nested subquery라 한다.

Nested subquery를 사용하는 경우 nested subquery의 table 또는 view의 column 이름과 nested subquery를 포함하는 query의 table 또는 view의 column 이름이 같을 수 있다. 이 때 nested subquery의 <select list>에 column 이름만 기술한 경우 nested subquery에 존재하는 table 또는 view의 column을 참조한다. 만약 nested subquery의 <select list>에 nested subquery의 table 또는 view에 존재하지 않는 column 이름이 사용된 경우 nested subquery를 포함하고 있는 query table 또는 view에 있는 해당 column 이름을 참조한다.

```
SELECT r_name
  FROM region
 WHERE EXISTS ( SELECT *
                  FROM nation
                  WHERE n_nationkey < 5           /* nation.n_nationkey */
                  AND n_regionkey = r_regionkey ) /* nation.n_regionkey = region.r_regionkey
*/
;
```

Optimizer는 <where clause>에 존재하는 부질의인 nested subquery를 nested subquery가 포함된 query에 unnest하고 SEMI JOIN 또는 ANTI-SEMI JOIN 으로 처리할 수 있다. 이는 nested subquery를 최적화하는 것으로 IN, NOT IN, EXISTS, NOT EXISTS, quantify operator 등에 부질의가 존재하는 경우 optimizer가 unnest에 대한 cost를 계산하여 판단한다. 만약 사용자가 해당 nested subquery를 강제로 unnest하고 싶다면 nested subquery의 <hint clause>를 사용할 수 있다.

자세한 내용은 nested subquery의 unnest에 대한 **hint clause**를 참조한다.

```
SELECT r_name
  FROM region
 WHERE r_regionkey IN ( SELECT /*+ UNNEST */
                        n_regionkey
                        FROM nation
                        WHERE n_nationkey < 5 );
```

부질의에 대한 자세한 내용은 <subquery> 절을 참조한다.

12.4 Control Language

Control Language 관련 구문

Transaction control 관련 구문

- COMMIT
- ROLLBACK
- LOCK TABLE
- SAVEPOINT savepoint_specifier
- RELEASE SAVEPOINT savepoint_specifier

Session control 관련 구문

- ALTER SESSION SET property_name
- SET SESSION AUTHORIZATION user_identifier
- SET SESSION CHARACTERISTICS AS transaction_mode
- SET TIME ZONE
- SET TRANSACTION transaction_mode

System control 관련 구문

- ALTER SYSTEM CHECKPOINT
- ALTER SYSTEM {MOUNT | OPEN} DATABASE
- ALTER SYSTEM [KILL | DISCONNECT] SESSION
- ALTER SYSTEM SET property_name
- ALTER SYSTEM RECONNECT GLOBAL CONNECTION
- ALTER SYSTEM SWITCH LOGFILE

Transaction Control

트랜잭션 제어 구문은 DML 구문과 DDL 구문으로 인한 갱신 사항을 트랜잭션 내에서 관리하기 위한 구문이다. 트랜잭션 제어 구문 중에 COMMIT은 갱신 사항을 영속적으로 보존하고 ROLLBACK은 갱신 사항을 철회한다.

트랜잭션 제어 구문은 크게 다음과 같이 분류할 수 있다.

표 12-3 트랜잭션 제어 구문

구문	설명	참조 링크
COMMIT	트랜잭션 정상 종료	COMMIT
ROLLBACK	트랜잭션 철회	ROLLBACK
SAVEPOINT	저장점 생성	SAVEPOINT savepoint_specifier
RELEASE SAVEPOINT	저장점 제거	RELEASE SAVEPOINT savepoint_specifier

구문	설명	참조 링크
LOCK TABLE	Table-level lock 설정	LOCK TABLE
SET TRANSACTION	트랜잭션 속성 제어 (읽기/쓰기, isolation level)	SET TRANSACTION transaction_mode
SET CONSTRAINTS	지연가능한 제약 조건의 검사시점 제어	SET CONSTRAINTS

트랜잭션은 데이터를 갱신하는 DML이나 SQL 객체를 변경하는 DDL 구문을 최초로 실행할 때 자동으로 생성된다. 단, SELECT나 제어 구문을 실행할 때는 트랜잭션이 생성되지 않는다.

트랜잭션 철회 (ROLLBACK)는 전체 철회 (total rollback)와 부분 철회 (partial rollback)로 구분되며 부분 철회에는 명시적 방법과 암시적 방법이 있다. 전체 철회는 ROLLBACK 구문으로 실행하는데 트랜잭션 내에서 수행한 DML, DDL의 모든 변경 내용을 이전 상태로 복구한다.

사용자는 다음 예와 같이 savepoint를 사용한 ROLLBACK 구문을 이용하여 명시적으로 부분 철회를 실행한다. 다음 예에서는 저장점 sp1과 sp2를 선언하고 이를 이용하여 명시적으로 부분 철회하고 있다.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> SAVEPOINT sp1;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> SAVEPOINT sp2;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom' );
1 row created.
gSQL> ROLLBACK TO SAVEPOINT sp2;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
2 mkkim
2 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
```

```
-- -----
1 leekmo
1 row selected.
gSQL> ROLLBACK WORK;
Rollback complete.
gSQL> SELECT * FROM t1;
no rows selected.
```

구문을 실행할 때 에러가 발생하면 해당 구문의 갱신 사항만을 철회하며 이를 암시적 부분 rollback이라고 한다. 다음은 암시적 부분 rollback의 예로써, unique 제약 조건을 위반한 경우 해당 INSERT 구문만 철회되고 트랜잭션의 이전 갱신 사항은 그대로 유지된다.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id);
Table altered.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 4, 'egonspace' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 1, 'jhkim' );
ERR-40002(16057): unique constraint (PUBLIC.T1_UK) violated
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
2 mkkim
3 xcom
4 egonspace
4 rows selected.
```

Session Control

세션은 데이터베이스에 접속한 사용자의 상태 정보를 관리하는 논리적 객체이다. 세션 제어 구문은 세션의 속성을 변경한다.

세션 제어 구문은 다음과 같이 분류할 수 있다.

표 12-4 세션 제어 구문

구문	설명	참조 링크
SET SESSION CHARACTERISTICS	세션 내의 트랜잭션 속성 제어	SET SESSION CHARACTERISTICS AS transaction_mode

구문	설명	참조 링크
SET TIME ZONE	세션 time zone 변경	SET TIME ZONE
SET SESSION AUTHORIZATION	세션 사용자 변경	SET SESSION AUTHORIZATION user_identifier
ALTER SESSION SET	세션 프로퍼티 값 변경	ALTER SESSION SET property_name

트랜잭션 제어 구문인 SET TRANSACTION 구문과 세션 제어 구문인 SET SESSION CHARACTERISTICS 모두 트랜잭션의 속성을 제어하는 구문이지만 다음과 같은 차이가 있다. SET TRANSACTION 구문은 다음에 수행할 하나의 트랜잭션에만 적용되는 반면, SET SESSION CHARACTERISTICS 구문은 해당 세션에서 이후 발생하는 모든 트랜잭션에 적용된다.

다음은 SET TIME ZONE 구문을 사용하여 time zone을 변경한 후에 현재 날짜/ 시간을 얻어오는 CURRENT_TIMESTAMP 함수를 사용한 결과이다.

```
gSQL> SELECT CURRENT_TIMESTAMP FROM DUAL;
CURRENT_TIMESTAMP
-----
2014-07-21 11:42:49.828276 +09:00
1 row selected.
gSQL> SET TIME ZONE '+00:00';
Session set.
gSQL> SELECT CURRENT_TIMESTAMP FROM DUAL;
CURRENT_TIMESTAMP
-----
2014-07-21 02:43:03.437940 +00:00
1 row selected.
```

System Control

시스템 제어 구문은 데이터베이스 시스템을 관리하는 구문으로써 다음과 같이 구분할 수 있다.

표 12-5 시스템 제어 구문

구문	설명	참조 링크
ALTER SYSTEM {OPEN MOUNT} DATABASE	데이터베이스를 구동한다.	ALTER SYSTEM {MOUNT OPEN} DATABASE
ALTER SYSTEM CHECKPOINT	체크포인트를 수행한다.	ALTER SYSTEM CHECKPOINT
ALTER SYSTEM KILL SESSION	특정 세션을 강제로 종료한다.	ALTER SYSTEM [KILL DISCONNECT] SESSION
ALTER SYSTEM SWITCH LOGFILE	로그 파일을 전환한다.	ALTER SYSTEM SWITCH LOGFILE
ALTER SYSTEM SET	시스템 프로퍼티를 설정한다.	ALTER SYSTEM SET property_name
ALTER SYSTEM RESET	시스템 프로퍼티를 제거한다.	ALTER SYSTEM RESET property_name

다음은 데이터베이스에 연결되어 있는 세션들을 조회하고 이 중 특정 세션을 강제로 종료하는 예이다.

```
gSQL> SELECT USER_NAME, SESSION_ID, SERIAL_NO, SESSION_STATUS, PROGRAM_NAME FROM V$SESSION  
WHERE USER_NAME = 'TEST';
```

```
USER_NAME SESSION_ID SERIAL_NO SESSION_STATUS PROGRAM_NAME
```

```
-----  
TEST          62         49 CONNECTED      gsql  
TEST          65        109 CONNECTED      gsqlnet  
TEST          66        130 CONNECTED      gsql
```

```
3 rows selected.
```

```
gSQL> ALTER SYSTEM DISCONNECT SESSION 65, 109;
```

```
System altered.
```

12.5 Cluster의 SQL 처리

본 장에서는 cluster 환경에서의 다양한 SQL 구문 처리 과정에 대해 설명한다.

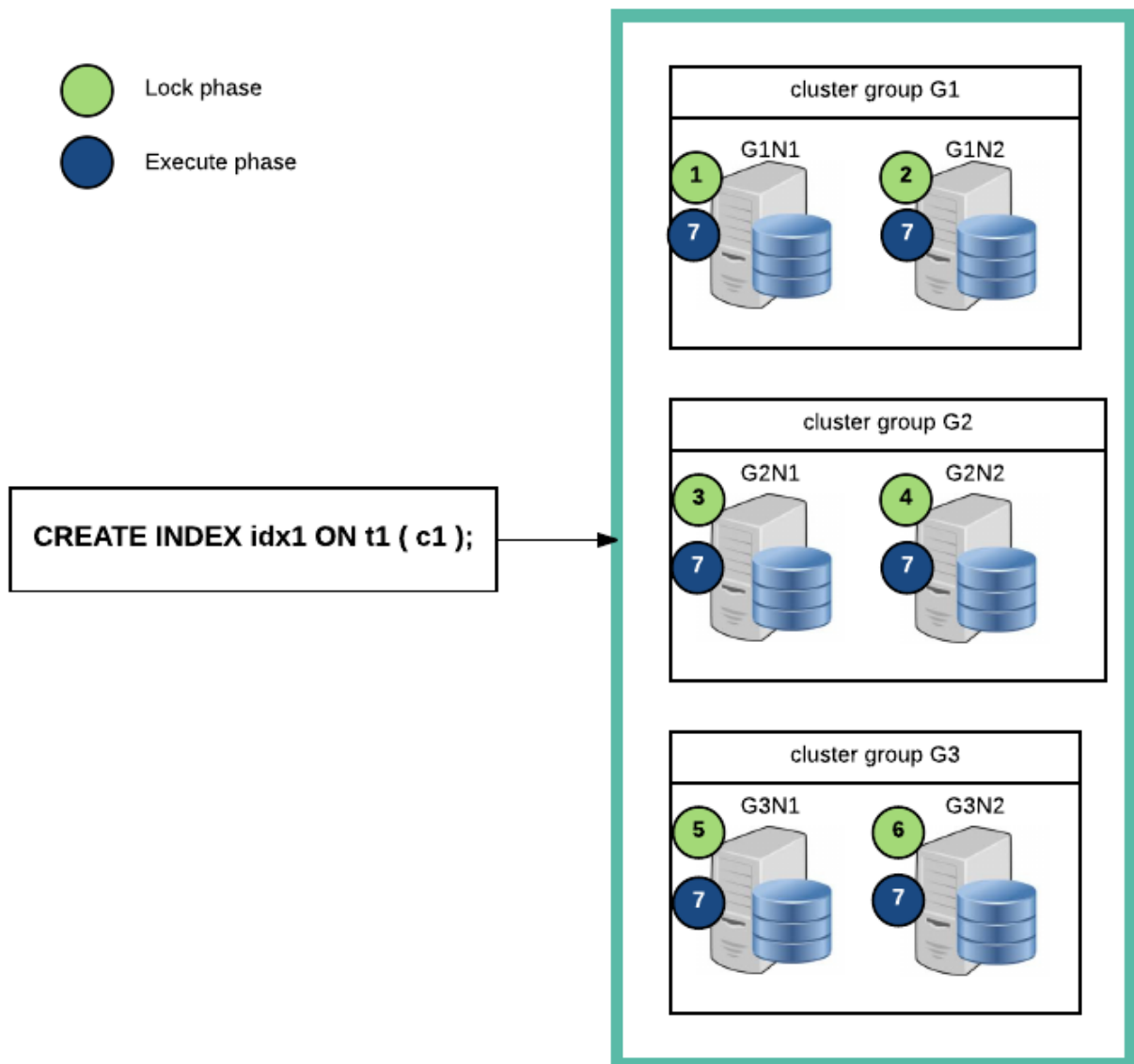
Cluster의 DDL 처리

Cluster의 DDL 처리 과정

GOLDILOCKS cluster에는 별도의 meta server가 없으며 사용자는 cluster system을 구성하는 모든 cluster member에서 DDL을 수행할 수 있다.

Cluster 환경에서 DDL은 아래 그림과 같은 절차에 따라 실행된다.

그림 1 Cluster의 DDL 처리 과정



DDL은 lock phase와 execution phase로 나뉘어 처리된다. Lock phase는 DDL 수행에 필요한 lock을 획득하는 단계로써 모든 cluster member들에 대해 순차적으로 DDL을 수행한다. Execute phase에서는 모든 cluster member에 대해 동시에 DDL을 처리한다.

모든 cluster member에서 성공적으로 DDL을 수행한 경우 DDL이 완료되며, 특정 cluster member에서 실패할 경우 모든 cluster member의 DDL 작업이 취소된다. Cluster member에서 장애가 발생할 경우 DDL을 수행할 수 없다. 이러한 과정을 통해 모든 cluster member들이 객체들에 대한 meta 정보를 동일하게 동기화한다.

DDL 동시 수행

Cluster system의 구성을 변경하는 cluster 객체에 대한 DDL과 SQL 객체에 대한 DDL은 동시에 수행할 수 없다. Cluster 객체에 대한 DDL과 SQL 객체에 대한 DDL의 동시 수행 가능 여부는 다음과 같다.

표 12-6 DDL 동시 수행 가능 여부

구분	Cluster 객체 DDL	SQL 객체 DDL
Cluster 객체 DDL	X	X
SQL 객체 DDL	X	O

위의 표에서와 같이 다음과 같은 DDL은 동시에 수행할 수 없다.

- Cluster 객체 DDL과 Cluster 객체 DDL
 - (X) CREATE CLUSTER GROUP g2 CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210;
 - (X) ALTER CLUSTER GROUP g1 ADD CLUSTER MEMBER g1n2 HOST '192.168.0.12' PORT 10120;
- Cluster 객체 DDL과 SQL 객체 DDL
 - (X) CREATE CLUSTER GROUP g2 CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210;
 - (X) CREATE TABLE t1 (c1 INTEGER);
- SQL 객체 DDL과 SQL 객체 DDL
 - (O) CREATE TABLE t1 (c1 INTEGER);
 - (O) CREATE TABLE t2 (a1 INTEGER);

Cluster의 SELECT 처리

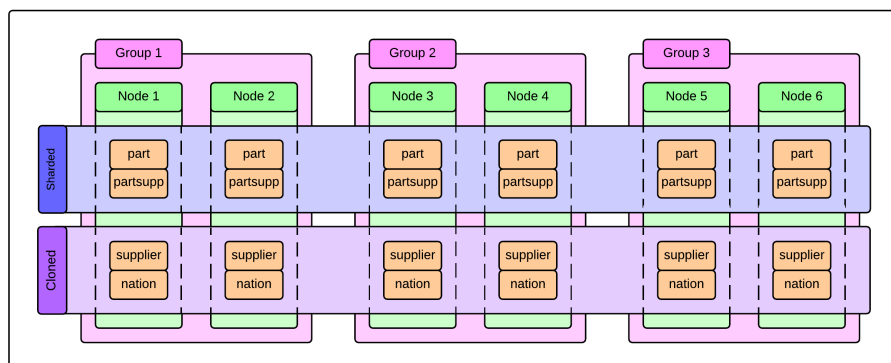
Cluster의 질의 처리

Cluster에서는 기본적으로 standalone와 동일하게 질의를 처리하지만 데이터가 local server 뿐만 아니라 remote server에도 존재할 경우 remote server로 질의 처리를 요청하고 그 결과를 취합한다는 차이가 있다.

Cluster 환경에는 sharded table과 cloned table (**14.5 Cluster Table과 Shard** 참조)이 있는데 각각의 테이블 데이터는 local server와 remote server에 저장된다. Sharded table의 데이터는 group에 분할 저장되고 동일 group의 member들에는 데이터가 복제되어 저장된다. Cloned table에는 모든 group과 member에 데이터가 복제되어 저장된다.

아래 그림은 3 x 2로 구성된 GOLDILOCKS의 cluster와 해당 cluster에 저장된 table들이다.

그림 2 3 x 2 cluster 구성과 table



다음은 위 그림의 table들을 생성하는 DDL 구문이다.

```
CREATE TABLE part
(
    p_partkey    INTEGER
, p_name       VARCHAR(55)
, p_brand      CHAR(10)
, p_type       VARCHAR(25)
, p_size       INTEGER
, p_retailprice NUMERIC(12,2)
, CONSTRAINT part_pk PRIMARY KEY( p_partkey ) INDEX part_pk_index
)
    SHARDING BY HASH(p_partkey)
    SHARD COUNT 3;
CREATE TABLE partsupp
(
    ps_partkey    INTEGER
, ps_suppkey     INTEGER
, ps_availqty    INTEGER
, ps_supplycost  NUMERIC(12,2)
, CONSTRAINT partsupp_pk PRIMARY KEY( ps_partkey, ps_suppkey ) INDEX partsupp_pk_index
)
    SHARDING BY HASH(ps_partkey)
    SHARD COUNT 3;
CREATE TABLE supplier
(
    s_suppkey    INTEGER
, s_name        CHAR(25)
, s_nationkey   INTEGER
, s_phone       CHAR(15)
, CONSTRAINT supplier_pk PRIMARY KEY( s_suppkey ) INDEX supplier_pk_index
) CLONED;
CREATE TABLE nation
(
    n_nationkey  INTEGER
, n_name        CHAR(25)
) CLONED;
```

위 그림에서 part table과 partsupp table은 sharded table로써 각 group별로 데이터가 분할되어 저장되어 있다. Supplier table은 cloned table로써 모든 node에 데이터가 복제되어 저장되어 있다.

GOLDILOCKS는 cluster 환경에서 table 형태 및 검색할 데이터의 위치에 따라 질의를 다르게 처리한다. Cluster의

질의처리는 single table에서 처리하는 것과 두 개 이상의 table에서 처리하는 것으로 나뉜다.

Single Table에서의 질의 처리

Single table에서의 질의 처리는 sharded table에서의 처리와 cloned table에서의 처리로 나누어진다. Sharded table에서 처리할 경우 n개의 group에 나뉘어 저장되므로 기본적으로 local server와 remote server 모두에 질의를 요청하고 이를 하나로 취합하여 결과 집합을 만든다. Remote server로 질의를 요청하고 결과를 받기 위하여 GOLDI LOCKS는 cluster access라는 access node를 사용한다. Cluster access는 local server와 remote server에 동시에 질의를 보내고 이에 대한 결과를 병렬로 수집하여 결과 집합을 만든다.

다음은 sharded table인 part table에서 질의를 처리하는 예이다.

```
gSQL> \explain plan
SELECT p_name, p_brand, p_type, cluster_group_id
FROM part;
```

P_NAME	P_BRAND	P_TYPE	CLUSTER_GROUP_ID
Part#3	Brand#2	STEEL	1
Part#2	Brand#1	NICKEL	2
Part#5	Brand#3	STEEL	2
Part#1	Brand#1	COPPER	3
Part#4	Brand#3	NICKEL	3

```
5 rows selected.
>>> start print plan
< Execution Plan >
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	CLUSTER ACCESS ("PART") [HASH SHARDING]	5
2	TABLE ACCESS ("PART") [HASH SHARDING]	1

```
1 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."P_NAME","_A1"."P_BRAND","_A1"."P_TYPE","_A1"."CLUSTER_GROUP_ID FROM "PUBLIC"."PART"@LOCAL "_A1"
2 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE
<<< end print plan
```

GOLDILOCKS는 sharded table에 대하여 **Cluster Domain**을 지정하거나 shard key의 equi 조건으로 특정 shard를 지정할 수 있다. Cluster domain을 사용하는 경우 명시된 domain에 위치한 데이터만 검색 대상이 되는데, 이 때 명시된 domain에 속한 node들에만 접근한다. 만약 cluster domain이 이것을 이용하여 local server에만 접근하면 될 경우 cluster access는 발생하지 않는다.

다음은 cluster domain을 이용하여 local server에만 접근하도록 했을 때 질의를 처리하는 예이다.

```
gSQL> \explain plan
SELECT p_name, p_brand, p_type, cluster_group_id
FROM part@g1;
```

```
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
```

```
-----
Part#3 Brand#2    STEEL                1
```

```
1 row selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

```
=====
```

```
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
```

```
-----
```

```
|    0  |  SELECT STATEMENT                                |
```

```
|
```

```
|    1  |  TABLE ACCESS ("PART"@G1") [HASH SHARDING]      |
```

```
1 |
```

```
=====
```

```
=====
```

```
1 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE
```

```
<<< end print plan
```

Shard key의 equi 조건을 이용하는 것은 table의 특정 shard를 가리키며 해당 shard key의 데이터가 위치한 domain에만 질의를 보낸다. Shard key의 equi 조건이 local server에 있는 shard를 가리키는 경우에는 cluster domain을 명시했을 때와는 달리 cluster access가 발생하지만 실제 실행할 때는 local server에만 접근한다.

다음은 shard key의 equi 조건을 이용하여 질의를 처리하는 예이다.

```
gSQL> \explain plan
SELECT p_name, p_brand, p_type, cluster_group_id
FROM part
WHERE p_partkey = 3;
```

```
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
```

```
-----
Part#3 Brand#2    STEEL                1
```

```
1 row selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

```
=====
```



```

|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
---
|    0  |  SELECT STATEMENT                                |
|
|    1  |  CLUSTER ACCESS ("PART") [HASH SHARDING]          |
1 |
|    2  |  INDEX ACCESS ("PART", "PART_PK_INDEX") [HASH SHARDING] | 1)
1 |
=====
===
1 - SQL : SELECT /*+ INDEX_ASC("_A1", "PART_PK_INDEX") */ "_A1"."P_PARTKEY", "_A1"."P_
NAME", "_A1"."P_BRAND", "_A1"."P_TYPE", "_A1".CLUSTER_GROUP_ID FROM "PUBLIC"."PART"@LOCAL "_A1"
WHERE "_A1"."P_PARTKEY" = ?
      BIND PARAMS : {0} IN
      REFERENCE SHARD KEY VALUE : (3)
2 - READ INDEX COLUMNS : P_PARTKEY
      READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
      MIN RANGE : P_PARTKEY = 3
      MAX RANGE : P_PARTKEY = 3
<<< end print plan

```

Sharded table과 달리 cloned table은 모든 node에 복제본을 가지고 있기 때문에 대부분의 local server에서 cloned table에 대한 질의를 처리할 수 있다. 다만, 새로운 group이나 member가 추가되는 경우 해당 group 또는 member에 cloned table에 대한 데이터가 없기 때문에 remote server로부터 데이터를 가져와야 하며, 이 때 cluster access가 발생한다.

다음은 cloned table인 supplier에 대한 질의를 처리하는 예이다.

```

gSQL> \explain plan
SELECT s_name, s_nation
FROM supplier;
S_NAME          S_NATION
-----
Supplier#1      FRANCE
Supplier#2      KOREA
Supplier#3      GERMANY
Supplier#4      UNITED STATES
Supplier#5      CANADA
5 rows selected.
>>> start print plan

```

< Execution Plan >

```
=====
====
|  IDX  |  NODE DESCRIPTION                               |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                               |
|
|    1  |  TABLE ACCESS ("SUPPLIER") [CLONED]           |
5 |
=====
====
      1 -  READ COLUMNS : S_NAME, S_NATION
<<< end print plan
```

조인 질의 처리

두 개 이상의 table에 대한 질의를 cluster로 처리할 수 있는지 여부는 해당 table들의 sharding 형태에 좌우된다. 해당 table들에 대한 조인 조건이 shard key에 대한 equi-join 조건인 경우 GOLDILOCKS는 cluster join의 join node를 사용하여 해당 join을 local server와 remote server에 병렬로 처리한다.

기본적으로 join을 처리하기 위한 데이터들이 논리적으로 같은 위치에 존재하는 경우에 join을 병렬 처리할 수 있는데, sharded table과 sharded table, sharded table과 cloned table, cloned table과 cloned table과 같은 조합이 있다.

Sharded table과 sharded table의 sharding 정책이 같고 shard key에 대한 equi 조건이 있을 경우 병렬처리 할 수 있다. 이는 shard의 배치가 동일한 상태에서 동일한 shard와의 equi-join이기 때문이다.

위 그림에서 part table의 shard key가 p_partkey이고, partsupp의 shard key가 ps_partkey라고 할 때, 두 table의 shard key에 대해 equi 조건을 갖는 질의는 다음과 같이 처리한다.

```
gSQL> \explain plan
SELECT p_name, ps_availqty
FROM part, partsupp
WHERE p_partkey = ps_partkey;
P_NAME PS_AVAILQTY
-----
Part#3      8895
Part#3      4969
Part#2      3956
Part#2      4069
Part#5      4651
```

```

Part#5      4093
Part#1      3325
Part#1      8076
Part#4      8539
Part#4      3025

```

10 rows selected.

```
>>> start print plan
```

```
< Execution Plan >
```

```

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    CLUSTER JOIN                                  |
10 |
|    2  |      HASH JOIN (INNER JOIN)                      |
2 |
|    3  |        TABLE ACCESS ("PARTSUPP") [HASH SHARDING] |
2 |
|    4  |          HASH JOIN INSTANT ACCESS                 |
2 |
|    5  |            TABLE ACCESS ("PART") [HASH SHARDING] |
1 |
=====
====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_HASH("_A1") FULL("_A2") USE_HASH
("_A2") */ "_A2"."P_NAME", "_A1"."PS_AVAILQTY" FROM "PUBLIC"."PARTSUPP"@LOCAL "_A1" INNER JOIN
"PUBLIC"."PART"@LOCAL "_A2" ON "_A2"."P_PARTKEY" = "_A1"."PS_PARTKEY"
2 - JOINED COLUMNS : PART.P_NAME, PARTSUPP.PS_AVAILQTY
3 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
4 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME
   READ COLUMNS : P_PARTKEY, P_NAME
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
5 - READ COLUMNS : P_PARTKEY, P_NAME
<<< end print plan

```

Sharded table과 cloned table의 경우 sharded table이 존재하는 group에 cloned table도 존재한다면 별도의 조

건없이 병렬 처리할 수 있다. 다음은 sharded table인 partsupp와 cloned table인 supplier를 병렬처리하는 예이다.

```
gSQL> \explain plan
```

```
SELECT s_name, ps_availqty
```

```
FROM supplier, partsupp
```

```
WHERE s_suppkey = ps_suppkey;
```

```
S_NAME                PS_AVAILQTY
```

```
-----
```

Supplier#1	8895
Supplier#4	4969
Supplier#5	3956
Supplier#2	4069
Supplier#1	4651
Supplier#4	4093
Supplier#3	3325
Supplier#2	8076
Supplier#3	8539
Supplier#5	3025

```
10 rows selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

```
====
```

IDX	NODE DESCRIPTION	

```
ROWS |
```

```
-----
```

0	SELECT STATEMENT	

```
|
```

1	CLUSTER JOIN	

```
10 |
```

2	HASH JOIN (INNER JOIN)	

```
2 |
```

3	TABLE ACCESS ("PARTSUPP") [HASH SHARDING]	

```
2 |
```

4	HASH JOIN INSTANT ACCESS	

```
2 |
```

5	TABLE ACCESS ("SUPPLIER") [CLONED]	

```
5 |
```

```
=====
```

```

1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_HASH("_A1") FULL("_A2") USE_HASH
("_A2") */ "_A2"."S_NAME", "_A1"."PS_AVAILQTY" FROM "PUBLIC"."PARTSUPP"@LOCAL "_A1" INNER JOIN
"PUBLIC"."SUPPLIER"@LOCAL "_A2" ON "_A2"."S_SUPPKEY" = "_A1"."PS_SUPPKEY"
2 - JOINED COLUMNS : SUPPLIER.S_NAME, PARTSUPP.PS_AVAILQTY
3 - READ COLUMNS : PS_SUPPKEY, PS_AVAILQTY
4 - INDEX COLUMNS : S_SUPPKEY
   TABLE COLUMNS : S_NAME
   READ COLUMNS : S_SUPPKEY, S_NAME
   HASH FILTER : S_SUPPKEY = {PS_SUPPKEY}
5 - READ COLUMNS : S_SUPPKEY, S_NAME
<<< end print plan

```

Cloned table과 cloned table이 모두 존재하는 node가 하나라도 있으면 병렬 처리할 수 있는데, 만약 local server에 있을 경우 remote server에 접근하지 않아도 되므로 cluster join을 사용하지 않는다.

다음은 local server에 두 개의 cloned table (supplier, nation)이 존재할 때 질의를 처리하는 예이다.

```
gSQL> \explain plan
```

```
SELECT s_name, n_name
```

```
FROM supplier, nation
```

```
WHERE s_nationkey = n_nationkey;
```

```

S_NAME                N_NAME
-----
Supplier#2            KOREA
Supplier#1            FRANCE
Supplier#3            GERMANY
Supplier#4            UNITED STATES
Supplier#5            CANADA
5 rows selected.

```

```
>>> start print plan
```

```
< Execution Plan >
```

```

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |  HASH JOIN (INNER JOIN)                          |
5 |
|    2  |  TABLE ACCESS ("NATION") [CLONED]                |

```

```

5 |
| 3 |      HASH JOIN INSTANT ACCESS      |
5 |
| 4 |      TABLE ACCESS ("SUPPLIER") [CLONED]      |
5 |
=====
====
1 - JOINED COLUMNS : SUPPLIER.S_NAME, NATION.N_NAME
2 - READ COLUMNS : N_NATIONKEY, N_NAME
3 - INDEX COLUMNS : S_NATIONKEY
   TABLE COLUMNS : S_NAME
   READ COLUMNS : S_NATIONKEY, S_NAME
   HASH FILTER : S_NATIONKEY = {N_NATIONKEY}
4 - READ COLUMNS : S_NAME, S_NATIONKEY
<<< end print plan

```

Cluster Domain

Cluster domain은 cluster 환경에서 제한된 server들에서만 데이터를 취합한다. 예를 들면 salary를 구간별로 나누어 구성한 sharded table에서 특정 구간의 직원들을 검색하려면 이에 해당하는 cluster group을 cluster domain으로 설정하여 다음과 같이 query할 수 있다.

- Salary 구간별로 sharded table을 구성

```

gSQL> CREATE TABLE t1( name VARCHAR(128), salary INTEGER )
      SHARDING BY RANGE( salary )
          SHARD s1 VALUES LESS THAN ( 200 )      AT CLUSTER GROUP G1,
          SHARD s2 VALUES LESS THAN ( 400 )      AT CLUSTER GROUP G2,
          SHARD s3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP G3;

```

Table created.

```
gSQL> INSERT INTO t1 VALUES ( 'A', 500 );
```

1 row created.

```
gSQL> INSERT INTO t1 VALUES ( 'B', 100 );
```

1 row created.

```
gSQL> INSERT INTO t1 VALUES ( 'C', 300 );
```

1 row created.

- 특정 salary 구간의 직원 검색

```
gSQL> SELECT name, salary FROM t1@G2;
NAME SALARY

```

```

-----
C          300
1 row selected.

```

Cluster domain은 **from clause**에 기술된 table이나 view를 대상으로 <cluster domain>을 참고하여 정의하는데 이 때 다음 중 하나를 선택할 수 있다.

- 사용자 질의를 수행하는 cluster member
- Offline table을 대상으로 사용자 질의를 수행하는 cluster member
- 모든 cluster group
- 하나의 cluster group
- 하나의 cluster member

Cluster member가 cluster domain으로 선택된 경우 해당 server에 접근하여 데이터를 취합한다. 만약 해당 server가 대상 테이블에 대한 data 분배를 가지지 않는 경우 검색 결과는 존재하지 않는다.

사용자가 접속한 server에서 offline 상태의 table 데이터를 조회하기 위해 "@LOCAL_OFFLINE" cluster domain을 지원한다. 만약 해당 table이 online 상태인 경우 에러가 발생한다.

Cluster group이 cluster domain으로 선택된 경우 해당 group 내에서 대상 테이블에 대한 data 분배를 가지며 통신이 가능한 하나의 server에 접근하여 데이터를 취합한다. 만약 접근 가능한 server가 없는 경우 검색 결과는 존재하지 않는다.

모든 cluster group이 cluster domain으로 선택된 경우 각각의 cluster group의 데이터를 수집하고 취합한 결과를 사용자에게 전달한다.

Cluster domain은 데이터 갱신의 대상을 정하기 위한 용도로 사용할 수 없다. 즉, <cluster domain>을 DML 대상 table에 적용할 수 없다.

DML 대상 table에 대한 cluster domain은 다음과 같이 syntax error를 발생시킨다.

```

gSQL> INSERT INTO T1@LOCAL VALUES ( 1 );
ERR-42000(16062): syntax error :
INSERT INTO T1@LOCAL VALUES ( 1 )
          *
ERROR at line 1:
gSQL> UPDATE T1@GLOBAL SET I1 = 1;
ERR-42000(40000): syntax error:
UPDATE T1@GLOBAL SET I1 = 1
.....^    ^
Error at line 1
gSQL> DELETE FROM T1@G1;
ERR-42000(40000): syntax error:
DELETE FROM T1@G1

```

```
.....^^
Error at line 1
```

SELECT FOR UPDATE의 변경 대상 table에 대한 cluster domain은 다음과 같이 syntax error를 발생시킨다.

- 데이터 취합 대상에 대한 cluster domain

```
gSQL> SELECT I1 FROM T1@LOCAL;
I1
--
1
1 row selected.
```

- 데이터 갱신 대상에 대한 cluster domain

```
gSQL> SELECT I1 FROM T1@LOCAL FOR UPDATE;
ERR-42000(16062): syntax error :
SELECT I1 FROM T1@LOCAL FOR UPDATE
                        *
ERROR at line 1:
```

Generated Query

Generated query는 사용자가 제공한 query를 처리할 때 다른 server의 데이터를 참조하거나 갱신하기 위해 생성된다.

```
SELECT c1 FROM t1;
```

위와 같은 사용자 query가 주어진 경우, 사용자 query를 받은 server는 관련된 모든 cluster group 으로부터 t1에 대한 데이터를 취합하기 위해서 다음과 같은 generated query를 생성한다.

```
SELECT c1 FROM t1@LOCAL;
```

다음은 generated query를 설명하기 위해 구성된 예제 테이블이다.

```
CREATE TABLE t_shard_1( shard_key INTEGER, c1 INTEGER )
    SHARDING BY RANGE( shard_key )
        SHARD s1 VALUES LESS THAN ( 200 )      AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( 400 )      AT CLUSTER GROUP G2,
        SHARD s3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP G3;
CREATE TABLE t_shard_2( shard_key INTEGER, c1 INTEGER )
    SHARDING BY RANGE( shard_key )
```



```

        SHARD s1 VALUES LESS THAN ( 300 )          AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( MAXVALUE )    AT CLUSTER GROUP G3;
CREATE TABLE t_clone_1( c1 INTEGER ) CLONED AT CLUSTER WIDE;
CREATE TABLE t_clone_2( c1 INTEGER ) CLONED AT CLUSTER GROUP G2, G3;

```

Generated Query 구성

Generated query는 plan node 정보를 바탕으로 query를 재구성한다. 다음과 같은 plan node를 generated query로 구성할 수 있다.

- Access node
 - Table access
 - Index access
 - Rowid access
- Join node
 - Nested loops join
 - Hash join
 - Merge join (현재 지원하지 않음)
- Access node에 대한 generated query를 구성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1;
>>> start print plan
< Execution Plan >
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |  CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
|    2  |  TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
=====
====
1 - SQL : SELECT /*+ FULL("_A1") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
2 - READ COLUMNS : NOTHING
<<< end print plan

```

- Join node에 대한 generated query를 구성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT * FROM t_shard_1, t_clone_1 WHERE t_shard_1.shard_key = t_clone_1.c1;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    CLUSTER JOIN                                  |
0 |
|    2  |      HASH JOIN (INNER JOIN)                      |
0 |
|    3  |        TABLE ACCESS ("T_CLONE_1") [CLONED]      |
0 |
|    4  |        HASH JOIN INSTANT ACCESS                  |
0 |
|    5  |          TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_HASH("_A1") FULL("_A2") USE_HASH
("_A2") */ "_A2"."SHARD_KEY", "_A2"."C1", "_A1"."C1" FROM "PUBLIC"."T_CLONE_1"@LOCAL "_A1" INNER
JOIN "PUBLIC"."T_SHARD_1"@LOCAL "_A2" ON "_A2"."SHARD_KEY" = "_A1"."C1"
2 - JOINED COLUMNS : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1, T_CLONE_1.C1
3 - READ COLUMNS : C1
4 - INDEX COLUMNS : SHARD_KEY
   TABLE COLUMNS : C1
   READ COLUMNS : SHARD_KEY, C1
   HASH FILTER : SHARD_KEY = {C1}
5 - READ COLUMNS : SHARD_KEY, C1
<<< end print plan

```

Generated query로 표현할 수 있는 plan node 중에서 비용을 고려하여 적합한 plan node를 기반으로 generated query를 구성한다.

위와 같이 두 테이블을 join하는 경우 t_shard_1과 t_clone_1에 대한 access node를 기반으로 generated query를

생성할 수도 있고, 두 테이블에 대한 join node를 기반으로 generated query를 생성할 수도 있다.

전자의 경우 SELECT 처리를 위해 두 개의 access node 기반 generated query를 수행하여야 하며, 후자의 경우 한 개의 join node 기반 generated query를 수행하여야 한다. 레코드 탐색 시간보다 통신 비용이 비싸다고 평가되는 환경에서는 cost optimizer가 후자의 실행 계획을 선택할 것이다.

Cluster의 DML 처리와 같이 사용자가 데이터 변경을 요청한 경우 데이터 변경 기능을 포함하는 generated query를 생성한다.

다음과 같은 **Generated Query 구성 제약 사항**에 따라 generated query 구성 내용의 범위를 정하거나 generated query 구성이 불가능하다고 판단할 수 있다.

Generated Query 구성 제약 사항

다음과 같은 경우에 generated query 구성에 제약을 받는다.

- Grouping
- Ordering
- Offset & limit
- Non-deterministic Expression 사용
- 서로 다른 Sharding 정책을 가지는 Shard 테이블 Join
- shard_key에 대한 Equi-join Condition이 없는 Shard 테이블 Join
- 부질의 (subquery)를 Unnest 할 수 없는 경우
- 다른 Sharding 정책을 가지는 부질의 (subquery)를 포함하는 경우

Grouping

노트

Grouping 정보는 generated query로 구성되지 않는다.

- Generated query는 grouping 정보를 포함하지 않는다.

```
gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 GROUP BY shard_key;
>>> start print plan
< Execution Plan >
=====
=====
|  IDX  |  NODE DESCRIPTION  |
ROWS |
```

```

-----
| 0 | SELECT STATEMENT |
|
| 1 | GROUP HASH INSTANT ACCESS |
0 |
| 2 | CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
| 3 | TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - GROUPING COLUMNS : SHARD_KEY
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
   "_A1"
3 - READ COLUMNS : SHARD_KEY
<<< end print plan

```

- Generated query는 grouping 정보를 포함하지 않는다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT DISTINCT c1 FROM t_shard_1;
>>> start print plan
< Execution Plan >
=====
====
| IDX | NODE DESCRIPTION |
ROWS |
-----
| 0 | SELECT STATEMENT |
|
| 1 | GROUP HASH INSTANT ACCESS |
0 |
| 2 | CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
| 3 | TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - GROUPING COLUMNS : C1
   READ COLUMNS : C1

```

```

2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
3 - READ COLUMNS : C1
<<< end print plan

```

Ordering

노트

Ordering 정보는 generated query로 구성되지 않는다.

- Generated query는 ordering 정보를 포함하지 않는다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 ORDER BY c1;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    SORT INSTANT ACCESS                            |
0 |
|    2  |    CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |    TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - SORT KEY : "T_SHARD_1.C1 ASC NULLS LAST"
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
3 - READ COLUMNS : C1
<<< end print plan

```

Offset & limit

[노트](#)

offset limit clause 정보는 generated query에 포함되지 않는다.

- Generated query는 offset 정보를 포함하지 않는다.

```
gSQL> \EXPLAIN PLAN ONLY
SELECT c1 FROM t_shard_1 OFFSET 1;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |  CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
|    2  |  TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
=====
====
1 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
2 - READ COLUMNS : C1
<<< end print plan
```

- Generated query는 limit 정보를 포함하지 않는다.

```
gSQL> \EXPLAIN PLAN ONLY
SELECT c1 FROM t_shard_1 LIMIT 1;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
```

```

-----
| 0 | SELECT STATEMENT |
|
| 1 | CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
| 2 | TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
2 - READ COLUMNS : C1
<<< end print plan

```

Non-deterministic Expression 사용

노트

Non-deterministic expression을 사용할 때 non-deterministic expression을 상수화할 수 없다면 generated query에 포함되지 않는다.

- Generated query는 sequence 관련 expression을 포함하지 않는다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT seq.nextval FROM t_shard_1;
>>> start print plan
< Execution Plan >

```

```

=====
====
| IDX | NODE DESCRIPTION |
ROWS |
-----
----
| 0 | SELECT STATEMENT |
|
| 1 | CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
| 2 | TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |

```

```
=====
====
1 - SQL : SELECT /*+ FULL("_A1") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
2 - READ COLUMNS : NOTHING
<<< end print plan
```

- Non-deterministic expression이 FILTER node에 의해 상수화된다.
- Generated query는 non-deterministic expression에 포함된다.

```
gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key = random( 1, 100 );
>>> start print plan
< Execution Plan >
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    FILTER                                          |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |      TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - READ COLUMNS : NOTHING
   NODE_EXPR : RANDOM(1,100)
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
"_A1" WHERE "_A1"."SHARD_KEY" = ?
   BIND_PARAMS : {0} IN
3 - READ COLUMNS : SHARD_KEY
   PHYSICAL_FILTER : SHARD_KEY = ?
<<< end print plan
```

- Non-deterministic expression은 상수화할 수 없다.
- Generated query는 non-deterministic expression을 포함하지 않는다.


```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key = random( c1, 100 );
>>> start print plan
< Execution Plan >
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    FILTER                                          |
0 |
|    2  |    CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |    TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1 - READ COLUMNS : NOTHING
    FILTER : T_SHARD_1.SHARD_KEY = RANDOM(T_SHARD_1.C1,100)
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY", "_A1"."C1" FROM "PUBLIC"."T_SHARD
_1"@LOCAL "_A1"
3 - READ COLUMNS : SHARD_KEY, C1
<<< end print plan

```

서로 다른 Sharding 정책을 가지는 Shard 테이블 Join

노트

서로 다른 sharding 정책을 가지는 shard 테이블 join에 대한 generated query를 생성할 수 없다.

- Join에 대한 generated query를 구성할 수 없다.
- Cluster access node에서 access와 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1, t_shard_2 WHERE t_shard_1.shard_key = t_shard_2.shard_key;
>>> start print plan

```

< Execution Plan >

```

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    HASH JOIN (INNER JOIN)                        |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |      TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    4  |      HASH JOIN INSTANT ACCESS                    |
0 |
|    5  |      CLUSTER ACCESS ("T_SHARD_2") [RANGE SHARDING] |
0 |
|    6  |      TABLE ACCESS ("T_SHARD_2") [RANGE SHARDING] |
0 |
=====
====
1 - JOINED COLUMNS : NOTHING
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
   "_A1"
3 - READ COLUMNS : SHARD_KEY
4 - INDEX COLUMNS : SHARD_KEY
   READ COLUMNS : SHARD_KEY
   HASH FILTER : {SHARD_KEY} = SHARD_KEY
5 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_2"@LOCAL
   "_A1"
6 - READ COLUMNS : SHARD_KEY
<<< end print plan

```

shard_key에 대한 Equi-join Condition이 없는 Shard 테이블 Join

노트

shard_key에 대한 equi-join condition이 없는 shard 테이블 join에 대한 generated query를 생성할 수 없다.

- Join에 대한 generated query를 구성할 수 없다.
- Cluster access node에서 access와 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1, t_shard_1;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |  NESTED LOOP JOIN (INNER JOIN)                  |
0 |
|    2  |  CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]   |
0 |
|    3  |  TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]   |
0 |
|    4  |  CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]   |
0 |
|    5  |  TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]   |
0 |
=====
====
1 - JOINED COLUMNS : NOTHING
2 - SQL : SELECT /*+ FULL("_A1") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
3 - READ COLUMNS : NOTHING
4 - SQL : SELECT /*+ FULL("_A1") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
5 - READ COLUMNS : NOTHING
<<< end print plan

```

- Join에 대한 generated query를 구성할 수 없다.
- Cluster access node에서 access와 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 A, t_shard_1 B WHERE A.c1 = B.c1;
>>> start print plan
< Execution Plan >

```

```

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    HASH JOIN (INNER JOIN)                        |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |      TABLE ACCESS ("T_SHARD_1" AS A) [RANGE SHARDING] |
0 |
|    4  |      HASH JOIN INSTANT ACCESS                    |
0 |
|    5  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    6  |      TABLE ACCESS ("T_SHARD_1" AS B) [RANGE SHARDING] |
0 |
=====
====
1 - JOINED COLUMNS : NOTHING
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
3 - READ COLUMNS : C1
4 - INDEX COLUMNS : C1
   READ COLUMNS : C1
   HASH FILTER : {C1} = C1
5 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1"
6 - READ COLUMNS : C1
<<< end print plan

```

- Join에 대한 generated query를 구성할 수 있다.
- Cluster join node에서 join과 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 A, t_shard_1 B WHERE A.shard_key = B.shard_key;
>>> start print plan
< Execution Plan >

```

```

=====
====

```

```

|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
|      0 |  SELECT STATEMENT                                |
|      |
|      1 |  CLUSTER JOIN                                    |
0 |
|      2 |  HASH JOIN (INNER JOIN)                          |
0 |
|      3 |  TABLE ACCESS ("T_SHARD_1" AS A) [RANGE SHARDING] |
0 |
|      4 |  HASH JOIN INSTANT ACCESS                        |
0 |
|      5 |  TABLE ACCESS ("T_SHARD_1" AS B) [RANGE SHARDING] |
0 |
=====
====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_HASH("_A1") FULL("_A2") USE_HASH
("_A2") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1" INNER JOIN "PUBLIC"."T_SHARD_1"@LOCAL "_
A2" ON "_A1"."SHARD_KEY" = "_A2"."SHARD_KEY"
2 - JOINED COLUMNS : NOTHING
3 - READ COLUMNS : SHARD_KEY
4 - INDEX COLUMNS : SHARD_KEY
   READ COLUMNS : SHARD_KEY
   HASH FILTER : {SHARD_KEY} = SHARD_KEY
5 - READ COLUMNS : SHARD_KEY
<<< end print plan

```

부질의 (subquery)를 Unnest 할 수 없는 경우

노트

상수화할 수 있는 부질의는 bind parameter 형태로 generated query 내에 포함된다.
상수화할 수 없는 부질의는 generated query 내에 포함될 수 없다.

부질의 unnest에 대한 자세한 내용은 **부질의 (Subquery)**를 참조한다.

- Join에 대한 generated query를 구성할 수 없다.
- Cluster access node에서 access와 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key IN ( SELECT /*+ NO_QUERY_TRANSFORMATION */ shard_key
FROM t_shard_1 );
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    FILTER                                          |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |      TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    4  |      SUB QUERY LIST                                |
0 |
|    5  |      SUB QUERY FUNCTION (MATERIALIZED)            |
0 |
|    6  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    7  |      TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
1  -  READ COLUMNS : NOTHING
      SUBQUERY FILTER : ( T_SHARD_1.SHARD_KEY ) IN ( T_SHARD_1.SHARD_KEY )
2  -  SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
      "_A1"
3  -  READ COLUMNS : SHARD_KEY
5  -  FUNCTION : ( SHARD_KEY ) IN ( SHARD_KEY )
6  -  SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
      "_A1"
7  -  READ COLUMNS : SHARD_KEY
<<< end print plan

```

- Generated query에 부질의를 포함할 수 없다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key = ( SELECT t_shard_1.c1 FROM dual );
>>> start print plan
< Execution Plan >
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    FILTER                                         |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |        TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    4  |          SUB QUERY LIST                          |
0 |
|    5  |            TABLE ACCESS ("DUAL") [CLONED]        |
0 |
=====
====
1 - READ COLUMNS : NOTHING
    SUBQUERY FILTER : T_SHARD_1.SHARD_KEY = {C1}
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY", "_A1"."C1" FROM "PUBLIC"."T_SHARD
_1"@LOCAL "_A1"
3 - READ COLUMNS : SHARD_KEY, C1
4 - READ COLUMNS : {C1}
5 - READ COLUMNS : NOTHING
<<< end print plan

```

- Generated query에 부질의를 포함한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key = ( SELECT 1 FROM dual );
>>> start print plan
< Execution Plan >
=====
====

```

```

|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |  CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
|    2  |  TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]    |
0 |
|    3  |  SUB QUERY LIST                                  |
0 |
|    4  |  TABLE ACCESS ("DUAL") [CLONED]                  |
0 |
=====
====
      1 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
"_A1" WHERE "_A1"."SHARD_KEY" = ?
      BIND PARAMS : {0} IN
      2 - READ COLUMNS : SHARD_KEY
      PHYSICAL FILTER : SHARD_KEY = 1
      3 - READ COLUMNS : 1
      4 - READ COLUMNS : NOTHING
<<< end print plan

```

다른 Sharding 정책을 가지는 부질의 (subquery)를 포함하는 경우

노트

서로 다른 sharding 정책을 가지는 shard 테이블 join과 유사한 상황으로써 부질의가 unnest 된 형태의 join으로 generated query를 구성할 수 없다.

부질의의 unnest에 대한 자세한 내용은 **부질의 (Subquery)**를 참조한다.

- Join에 대한 generated query를 구성할 수 없다.
- Cluster access node에서 access와 관련된 generated query를 생성한다.

```

gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key IN ( SELECT shard_key FROM t_shard_2 );
>>> start print plan

```


< Execution Plan >

```
=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    HASH JOIN (LEFT SEMI)                        |
0 |
|    2  |      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    3  |      TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    4  |      HASH JOIN INSTANT ACCESS (UNIQUE)           |
0 |
|    5  |      CLUSTER ACCESS ("T_SHARD_2") [RANGE SHARDING] |
0 |
|    6  |      TABLE ACCESS ("T_SHARD_2") [RANGE SHARDING] |
0 |
=====
====
1 - JOINED COLUMNS : NOTHING
2 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
"_A1"
3 - READ COLUMNS : SHARD_KEY
4 - INDEX COLUMNS : SHARD_KEY
   HASH FILTER : {SHARD_KEY} = SHARD_KEY
5 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_2"@LOCAL
"_A1"
6 - READ COLUMNS : SHARD_KEY
<<< end print plan
```

- Join에 대한 generated query를 구성한다.
- Cluster join node에서 join과 관련된 generated query를 생성한다.

gSQL> \EXPLAIN PLAN ONLY

SELECT 1 FROM t_shard_1 WHERE shard_key IN (SELECT shard_key FROM t_shard_1);

>>> start print plan

< Execution Plan >

```

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    CLUSTER JOIN                                  |
0 |
|    2  |      HASH JOIN (LEFT SEMI)                        |
0 |
|    3  |        TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
|    4  |          HASH JOIN INSTANT ACCESS (UNIQUE)          |
0 |
|    5  |            TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
=====
====
      1 - SQL : SELECT /*+ FULL("_A1") */ NULL FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1" WHERE
EXISTS( SELECT /*+ HASH_SJ FULL("_A2") */ TRUE FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A2" WHERE "_A
1"."SHARD_KEY" = "_A2"."SHARD_KEY" )
      2 - JOINED COLUMNS : NOTHING
      3 - READ COLUMNS : SHARD_KEY
      4 - INDEX COLUMNS : SHARD_KEY
          HASH FILTER : {SHARD_KEY} = SHARD_KEY
      5 - READ COLUMNS : SHARD_KEY
<<< end print plan

```

Generated Query 수행 대상 Server 선정

다음 정보들을 분석하여 generated query를 수행하기 위한 대상을 결정한다.

- 테이블 복제본 (replica) 배치 정책
- **Cluster Domain**
- Sharding key 검색 조건

Generated query 안에 포함된 각 테이블들을 대상으로 위에 나열한 정보들을 분석하고 이들에 공통적으로 포함된 s
erver를 찾아 generated query 수행 대상 server로 결정한다.

다음은 **Generated Query**를 설명하기 위해 정의된 테이블들의 복제본 (replica) 배치 정책을 요약한 것이다.

```
t_shard_1 (shard table) : at cluster group G1, G2, G3
t_shard_2 (shard table) : at cluster group G1, G3
t_clone_1 (cloned table) : at cluster group G1, G2, G3 (cluster wide)
t_clone_2 (cloned table) : at cluster group G2, G3
```

다음은 테이블 복제본 (replica) 배치 정책에 따라 수행 대상 server를 선정하는 예이다.

- 수행 대상 server: G1, G3 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_2;
```

- 수행 대상 server: G1, G2, G3 내 모든 server

```
gSQL> SELECT c1 FROM t_clone_1;
```

- 수행 대상 server: G1, G3 내 모든 server (공통으로 포함된 cluster group)

```
gSQL> SELECT c1 FROM t_shard_2, t_clone_1;
```

다음은 cluster domain 기술에 따라 수행 대상 server를 선택하는 예이다.

- 수행 대상 server: G1, G2, G3 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_1@GLOBAL;
```

- 수행 대상 server: 사용자 query를 요청 받은 server

```
gSQL> SELECT c1 FROM t_shard_1@LOCAL;
```

- 수행 대상 server: G2 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_1@G2;
```

- 수행 대상 server: G3N1 server

```
gSQL> SELECT c1 FROM t_shard_1@G3N1;
```

- 수행 대상 server: 대상 server 없음

```
gSQL> SELECT c1 FROM t_shard_1@G4;
```

다음은 sharding key 검색 조건에 따라 수행 대상 server를 선택하는 예이다.

- 수행 대상 server: G1, G2, G3 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_1;
```

- 수행 대상 server: G1, G2, G3 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_1 WHERE c1 = 1;
```

- 수행 대상 server: G3 내 모든 server

```
gSQL> SELECT c1 FROM t_shard_1 WHERE shard_key = 500;
```

- 수행 대상 server: 대상 server 없음

```
gSQL> SELECT c1 FROM t_shard_1 WHERE shard_key = 100 AND shard_key = 500;
```

Generated query를 실행할 때 데이터 갱신 여부에 따라 query 처리 단계에서의 generated query 수행 대상 server가 달라진다.

데이터를 갱신하는 Generated Query의 수행 대상 Server 선정

데이터를 갱신하기 위해 generated query를 수행할 때는 테이블 복제본 (replica) 배치 정책과 sharding key 검색 조건을 통해 수행 대상 cluster group을 분석한다. 수행 대상 cluster group 내에 속한 모든 server들이 generated query 수행 대상으로 선택된다.

데이터 갱신을 위한 사용자 query에는 **Cluster Domain**을 명시할 수 없다.

데이터를 갱신하기 위해 generated query를 수행할 때는 master server와 slave server로 나누어 수행 대상 server를 선정한다. 자세한 내용은 **Generated Query를 이용한 데이터 갱신**을 참조한다.

다음은 수행 대상 server를 선택하는 예이다.

- Generated query 수행 대상 server: G2 내의 모든 server
- t_shard_1: G2 내의 모든 server
- 테이블 복제본 (replica) 배치 정책: G1, G2, G3 내의 모든 server
- Sharding key 검색 조건: G2 내의 모든 server
- t_clone_1: G1, G2, G3 내의 모든 server
- 테이블 복제본 (replica) 배치 정책: G1, G2, G3 내의 모든 server

```
gSQL> \EXPLAIN PLAN ONLY
```

```
SELECT * FROM t_shard_1, t_clone_1 WHERE t_shard_1.shard_key = 300 FOR UPDATE;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

```
====
```

```
|  IDX  |  NODE DESCRIPTION
```

```
|
```

```

ROWS |
-----
----
| 0 | SELECT STATEMENT |
|
| 1 | CLUSTER JOIN |
0 |
| 2 | NESTED LOOP JOIN (INNER JOIN) |
0 |
| 3 | TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING] |
0 |
| 4 | TABLE ACCESS ("T_CLONE_1") [CLONED] |
0 |
=====
====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_NL("_A1") FULL("_A2") USE_NL("_A
2") */ "_A1"."SHARD_KEY", "_A1"."C1", "_A2"."C1", "_A1".ROWID, "_A2".ROWID FROM "PUBLIC"."T_SHARD
_1"@LOCAL "_A1" INNER JOIN "PUBLIC"."T_CLONE_1"@LOCAL "_A2" ON "_A1"."SHARD_KEY" = ?
      BIND PARAMS : {0} IN
      REFERENCE SHARD KEY VALUE (T_SHARD_1) : (300)
2 - JOINED COLUMNS : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1, T_CLONE_1.C1
3 - READ COLUMNS : SHARD_KEY, C1
      PHYSICAL FILTER : SHARD_KEY = 300
4 - READ COLUMNS : C1
<<< end print plan

```

데이터를 탐색하는 Generated Query 수행 대상 Server 선정

데이터를 탐색하기 위해 generated query를 수행할 때는 테이블 복제본 (replica) 배치 정책과 cluster domain, sharding key 검색 조건을 통해 공통된 server 들을 찾는다. 동일한 cluster group에 있는 다수의 server가 수행 대상이 되는 경우에는 수행 시의 네트워크 상태 등을 고려하여 cluster group 당 접근 가능한 하나의 server에 대해서만 generated query를 수행한다.

다음은 수행 대상 server를 선택하는 예이다.

- Generated query 수행 대상 server: G2 내의 한 개 server
- t_shard_1: G2 내의 모든 server
- 테이블 복제본 (replica) 배치 정책: G1, G2, G3 내의 모든 server
- Cluster domain: G2 내의 모든 server
- Sharding key 검색 조건: G2 내의 모든 server
- t_clone_1: G1, G2, G3 내의 모든 server
- 테이블 복제본(replica) 배치 정책: G1, G2, G3 내의 모든 server

- Cluster domain: G1, G2, G3 내의 모든 server

```

gSQL> \EXPLAIN PLAN ONLY
SELECT * FROM t_shard_1@G2, t_clone_1 WHERE t_shard_1.shard_key = 300;
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    CLUSTER JOIN                                  |
0 |
|    2  |      NESTED LOOP JOIN (INNER JOIN)               |
0 |
|    3  |        TABLE ACCESS ("T_SHARD_1"@G2N1") [RANGE SHARDING] |
0 |
|    4  |        TABLE ACCESS ("T_CLONE_1") [CLONED]        |
0 |
=====

====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_NL("_A1") FULL("_A2") USE_NL("_A2") */
   "_A1"."SHARD_KEY", "_A1"."C1", "_A2"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL "_A1" INNER
JOIN "PUBLIC"."T_CLONE_1"@LOCAL "_A2" ON "_A1"."SHARD_KEY" = ?
      BIND PARAMS : {0} IN
      REFERENCE SHARD KEY VALUE (T_SHARD_1) : (300)
2 - JOINED COLUMNS : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1, T_CLONE_1.C1
3 - READ COLUMNS : SHARD_KEY, C1
      PHYSICAL FILTER : SHARD_KEY = 300
4 - READ COLUMNS : C1
<<< end print plan

```

Cluster의 DML 처리

GOLDILOCKS에서는 사용자가 cluster system을 구성하는 모든 cluster member에서 DML을 수행할 수 있다.

Cluster 환경에서 데이터를 갱신하면 동일한 data 복제본 (replica)을 가지는 cluster member들의 데이터가 동일하

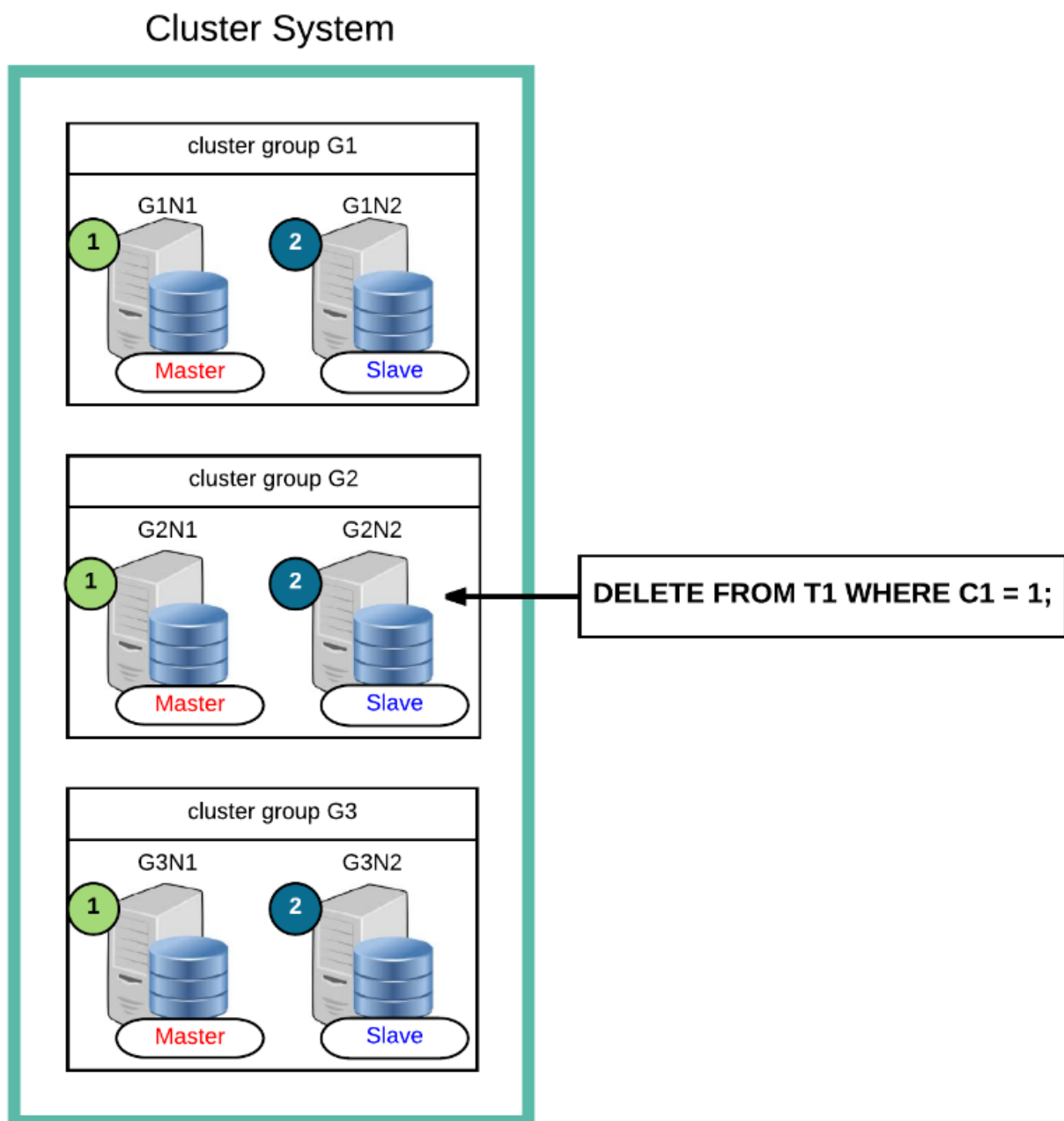
게 갱신된다.

다음은 데이터 갱신을 설명하기 위해 구성된 예제 테이블이다.

```
CREATE TABLE t1( shard_key INTEGER, c1 INTEGER )
  SHARDING BY RANGE( shard_key )
    SHARD s1 VALUES LESS THAN ( 200 )      AT CLUSTER GROUP G1,
    SHARD s2 VALUES LESS THAN ( 400 )      AT CLUSTER GROUP G2,
    SHARD s3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP G3;
```

Cluster 환경에서 DML은 아래 그림과 같은 절차로 수행된다.

그림 3 Cluster에서의 DML 처리



Cluster에서 DML을 처리하기 위해 각 group의 master server와 slave server를 정의한다.

각 Cluster Group의 Master Server 선정

Cluster 환경에서 데이터를 갱신할 때 각 cluster group에서 접근 가능한 cluster member 중에 member_id가 가장 작은 하나의 cluster member를 선택한다.

각 Cluster Group의 Slave Server 선정

Cluster 환경에서 데이터를 갱신할 때 각 cluster group에서 접근 가능한 cluster member 중에 master server를 제외한 나머지 cluster member들을 선택한다.

Cluster group과 cluster member의 구성 정보는 **DBA_CLUSTER**를 통해 확인할 수 있다.

```
gSQL> SELECT * FROM DBA_CLUSTER;
```

GROUP_ID	GROUP_NAME	MEMBER_ID	MEMBER_NAME	MEMBER_HOST	MEMBER_PORT
1	G1	1	G1N1	127.0.0.1	10110
1	G1	2	G1N2	127.0.0.1	10120
2	G2	3	G2N1	127.0.0.1	10210
2	G2	4	G2N2	127.0.0.1	10220
3	G3	5	G3N1	127.0.0.1	10310
3	G3	6	G3N2	127.0.0.1	10320

6 rows selected.

DML 수행

DML은 master server 반영 단계와 slave server 반영 단계로 나뉘어 차례대로 수행된다.

- Master server 반영 단계
 - 각 cluster group의 master server들의 데이터를 갱신한다.
- Slave server 반영 단계
 - 각 cluster group의 master server에 적용된 것과 동일하게 slave server들을 갱신한다.

GOLDILOCKS에서는 각 cluster group별로 master server와 slave server 간의 동기화를 유지하면서 데이터를 갱신하기 위해 다음 두 가지 방법을 지원한다.

- Generated Query를 이용한 데이터 갱신
- Rowid를 이용한 데이터 변경 (Rowid Pseudo Column)

Generated Query를 이용한 데이터 갱신

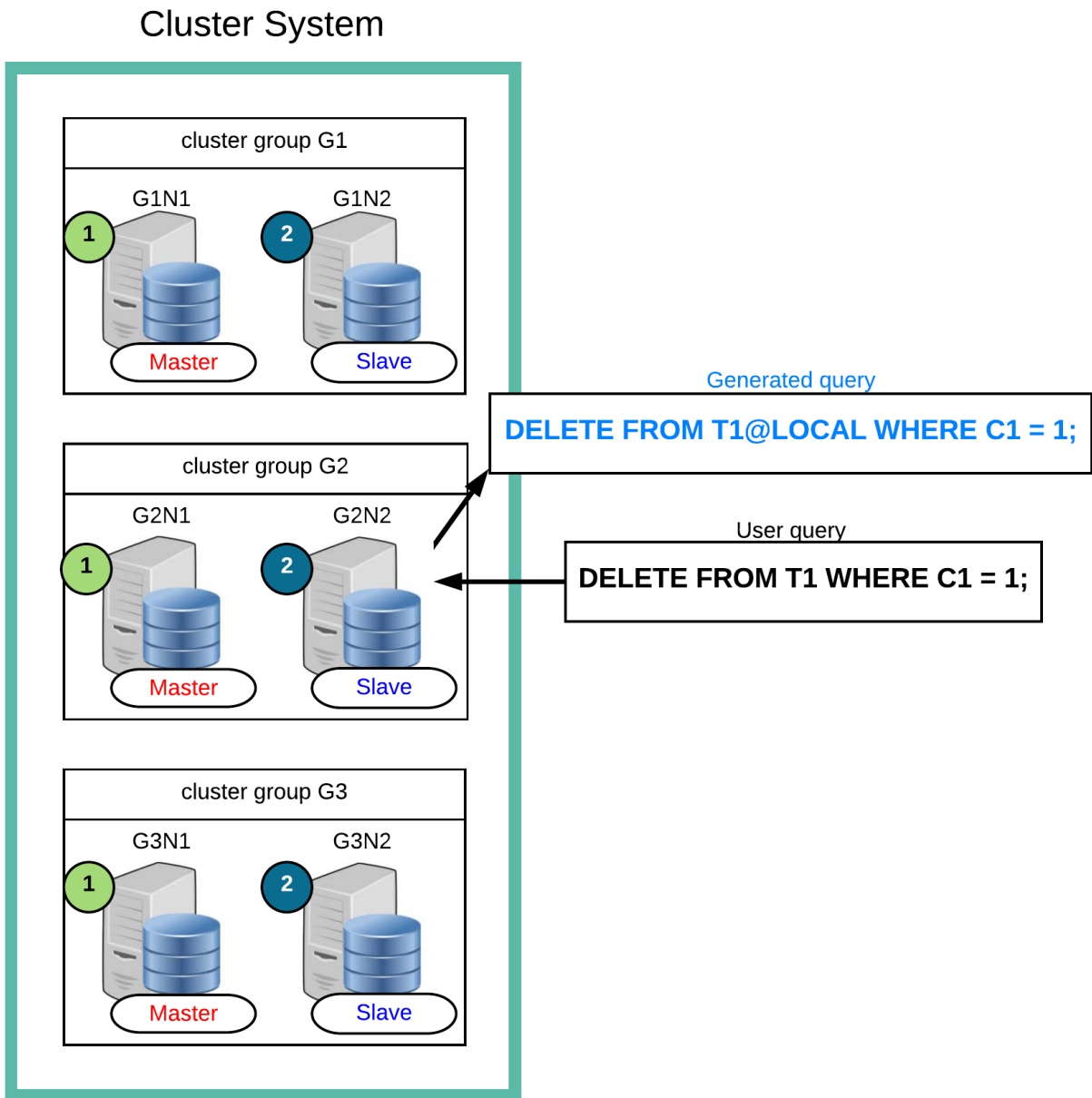
Generated query를 이용한 데이터 갱신은 사용자로부터 query를 받은 server가 내부적으로 생성한 generated query를 사용하여 각 server의 레코드들을 갱신하는 방법이다. 각 server에서 generated query를 이용해 수행된 결과가 동일함을 보장할 수 있는 경우에 한하여 지원된다.

자세한 내용은 **Generated Query**를 참조한다.

갱신된 레코드의 결과 반환 여부에 따라 master server와 slave server에 대한 generated query는 서로 다를 수 있다.

Generated query를 이용한 데이터 갱신은 아래 그림과 같은 절차로 수행된다.

그림 4 Generated query를 이용한 데이터 변경 (전체 cluster group 대상 변경)



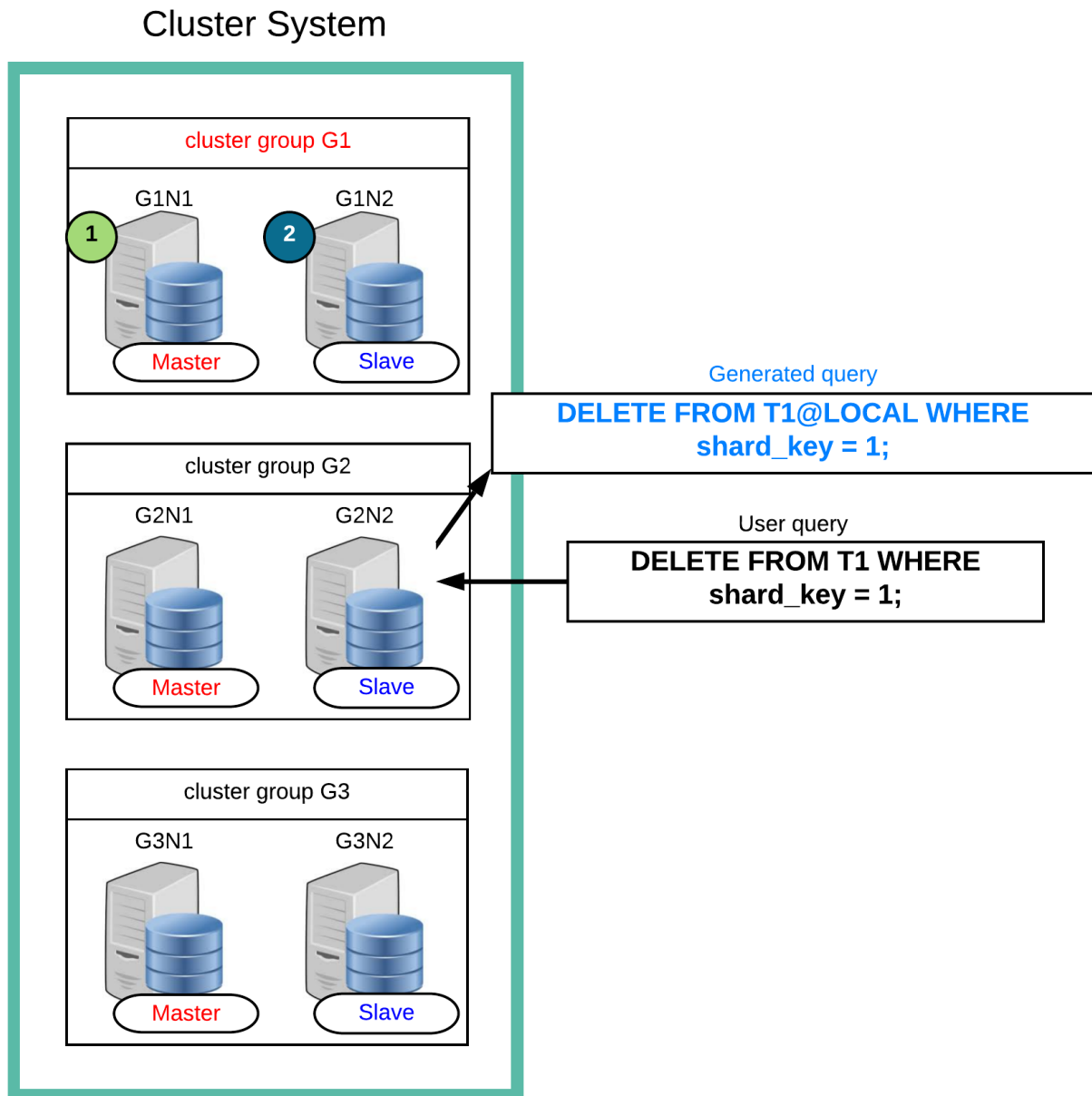
위의 그림과 같이 조건절에 갱신 대상 cluster group을 선택하기 위한 조건이 없는 경우, 모든 cluster group이 generated query를 이용한 데이터 갱신의 대상이 된다.

Generated query를 이용한 데이터 갱신은 다음과 같은 절차로 수행된다.

1. 각 master server에서 generated query 수행
2. 각 slave server에서 generated query 수행

조건절에 의해 특정 cluster group만 데이터 갱신 대상으로 정해지는 경우에는 아래 그림과 같은 절차로 수행된다.

그림 5 Generated query 이용한 데이터 변경 (특정 Cluster Group 대상 변경)



위의 그림과 같이 "shard_key = 1" 이라는 검색 조건을 사용하여 갱신 대상을 특정한 경우, sharding 정책에 의해 shard_key가 "1"에 해당하는 레코드가 cluster group "G1"에 있는 것을 알 수 있다. 따라서 "G1" cluster group의 레코드만 삭제한다.

특정 cluster group의 데이터만 갱신하는 경우에도 전체를 갱신하는 절차와 동일하게 master에 반영한 후에 slave들에 수행한다.

다음과 같은 경우에 한하여 generated query를 이용한 데이터 갱신이 지원된다.

- 한 cluster group의 각 cluster member에서 generated query를 수행한 결과가 서로 동일함을 보장할 수 있는 generated query를 생성할 수 있는 경우

- 이는 deterministic 하지 않은 구문 또는 함수들이 사용되지 않은 generated query를 구성한다는 것을 의미한다.
- Generated query를 구성할 때 데이터 참조 대상 server와 데이터 갱신 대상 server를 동일한 하나의 server로만 한정시킬 수 있는 경우
 - 이는 generated query를 수행하는 server에서 질의를 처리하는 도중에 다른 server에 접근할 필요가 없음을 의미한다.

Generated query를 이용한 데이터 갱신을 지원하는 사용자 query 유형은 다음과 같다.

- SELECT .. FOR UPDATE
- SELECT .. INTO .. FOR UPDATE
- DELETE FROM
- DELETE FROM name RETURNING
- DELETE FROM name RETURNING .. INTO
- UPDATE
- UPDATE name RETURNING
- UPDATE name RETURNING .. INTO

Generated query를 이용한 데이터 갱신은 Global Secondary Index가 구성되어 있지 않아도 수행할 수 있다.

Rowid를 이용한 데이터 변경 (Rowid Pseudo Column)

Rowid 정보를 이용한 데이터 갱신은 rowid를 사용하여 서로 다른 cluster member에 저장된 동일한 레코드들을 갱신하는 방법이다. 이 방법은 새로운 레코드를 추가하거나 **Generated Query를 이용한 데이터 갱신**이 불가능할 때 사용한다.

Cluster 환경에서 제공하는 rowid는 레코드에 부여된 논리적 식별 정보로서 레코드들 간의 동일성 여부를 판단하는 기준이 된다. 하나의 레코드가 서로 다른 여러 cluster member에 저장되는 경우, 모두 같은 rowid 값을 가지게 된다. Rowid 정보는 레코드를 생성할 때 부여되며, sharding key column에 대한 값을 갱신할 경우 새로운 값을 부여받기도 한다.

Rowid에 대한 자세한 내용은 **ROWID Pseudo Column**를 참조한다.

Rowid 정보를 이용한 데이터 갱신은 rowid 정보 수집 단계와 데이터 갱신 단계로 나누어 처리된다. 둘 이상의 레코드를 갱신할 때는 레코드 별로 rowid 정보 수집 단계와 데이터 갱신 단계를 반복 수행한다.

- Rowid 정보 수집 단계
 - 추가 또는 갱신 대상인 레코드들의 rowid 정보를 수집한다.
- 데이터 갱신 단계
 - 수집된 rowid에 해당하는 cluster group의 레코드를 갱신한다.
 - Master server의 레코드를 갱신한 후 slave server 레코드를 순차적으로 갱신한다.

Rowid 정보를 이용한 데이터 갱신은 **14.6 Global Secondary Index** 사용 여부에 따라 다음 두 가지 방법을 지원한다.

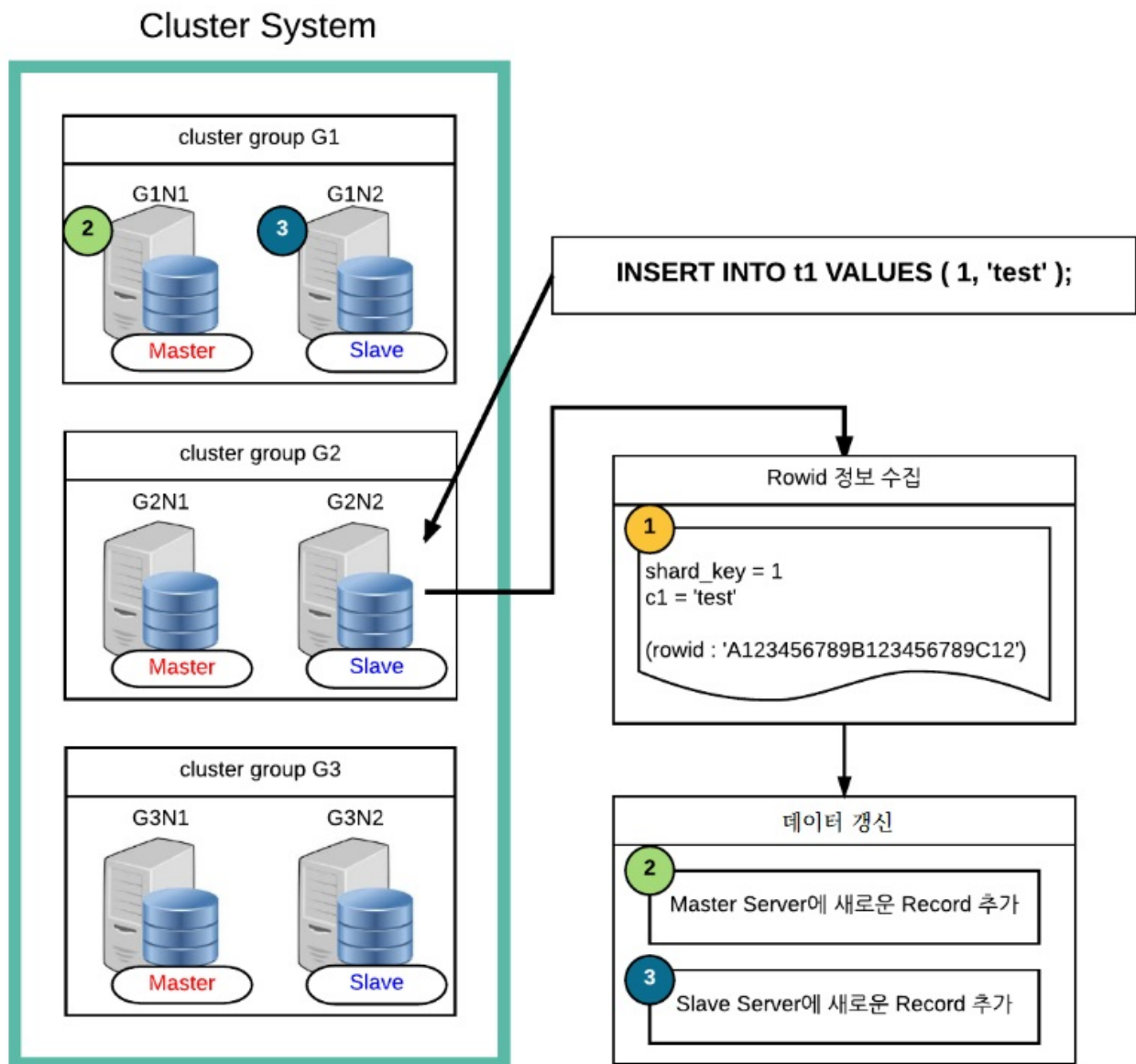
- Global Secondary Index를 사용하지 않는 rowid를 이용한 데이터 갱신
- Global Secondary Index를 사용하는 rowid를 이용한 데이터 갱신

Global Secondary Index를 사용하지 않는 rowid를 이용한 데이터 갱신

Global secondary index를 사용하지 않는 rowid를 이용한 데이터 갱신은 INSERT와 같이 기존 레코드를 갱신하지 않는 경우에 지원되는 방식이다.

Global secondary index를 사용하지 않는 rowid를 이용한 데이터 갱신은 아래 그림과 같은 절차로 수행된다.

그림 6 Global secondary index를 사용하지 않는 rowid를 이용한 데이터 갱신



레코드를 추가하는 경우에는 새로운 레코드들에 rowid를 부여하고 각 레코드들을 master server의 적합한 위치에 저장한다. 모든 레코드를 master server에 반영한 후 slave server에 반영한다.

위의 그림과 같이 "shard_key = 1" 이라는 값을 사용하여 sharding key column을 구성한 경우, sharding 정책에 의해 shard_key가 "1" 에 해당하는 레코드가 cluster group G1에 위치할 것을 알 수 있다. 따라서 G1 cluster grou

p에만 레코드를 추가한다.

Global secondary index를 사용하지 않는 rowid를 이용한 데이터 갱신은 다음과 같은 query 유형을 지원한다.

- INSERT INTO
- INSERT INTO name RETURNING
- INSERT INTO name RETURNING .. INTO

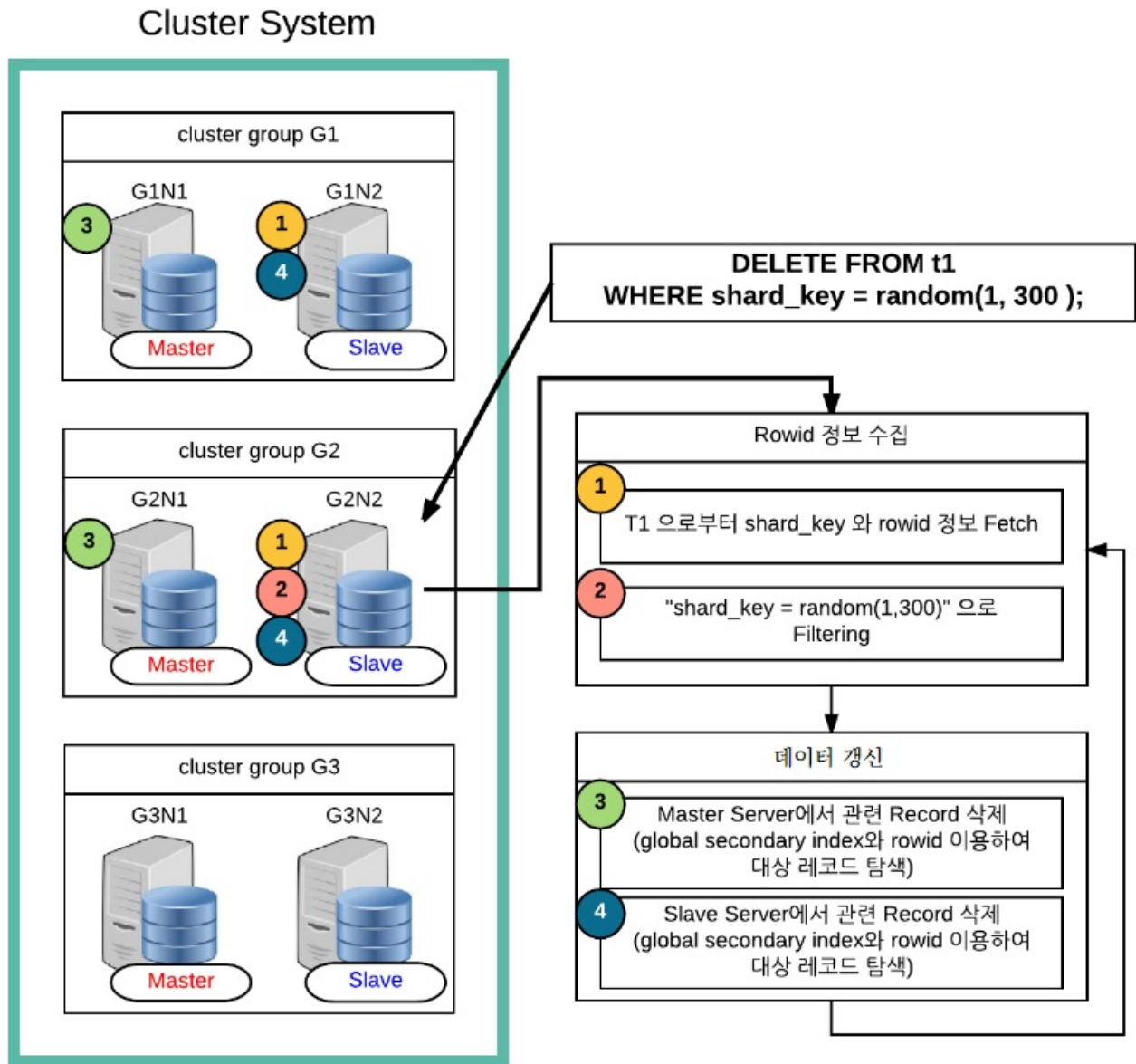
Global Secondary Index를 사용하는 rowid를 이용한 데이터 갱신

Global secondary index를 사용하는 rowid를 이용한 데이터 갱신은 기존 레코드를 갱신하는 경우에 지원된다.

기존 레코드를 갱신하고자 할 경우에는 rowid 정보를 이용하여 master server와 slave server에서 동일한 데이터가 똑같이 갱신되도록 보장해야 한다.

Global secondary index를 사용하는 rowid를 이용한 데이터 갱신은 아래 그림과 같은 절차로 수행된다.

그림 7 Global secondary index를 사용하는 rowid를 이용한 데이터 갱신



Rowid 정보 수집 단계에서는 사용자 query를 받은 server가 데이터 갱신 대상 레코드에 대한 정보를 수집한다. 위의 그림에서와 같이 평가할 때마다 결과값이 갱신될 수 있는 조건절이 사용된 경우, 사용자 query를 받은 server는 해당 조건절을 제외한 조건을 사용하여 rowid 정보와 제외된 조건절에서 참조된 column 정보를 취합한다. 결과를 취합하기 전에 제외된 탐색 조건으로 filtering을 하여 갱신 대상 레코드를 선정한다.

데이터 갱신 단계에서는 rowid 정보 수집 단계에서 선정된 레코드가 속한 cluster group을 대상으로 master server와 slave server의 데이터를 순차적으로 갱신한다. 위의 그림에서와 같이 shard_key는 "shard_key = random(1,300)" 조건에 의해 1부터 300까지의 범위를 가지고 sharding 정책에 의해 cluster group G1과 G2에 속하는 것을 알 수 있다. 따라서 cluster group G1과 G2에서만 레코드가 삭제된다.

데이터 갱신 단계에서 rowid 정보를 수집했던 server는 master server와 slave server에게 rowid 정보를 전달하여 데이터 갱신을 요청하고, 요청을 받은 server들은 전달받은 rowid 정보와 기존에 구축된 global secondary index를 이용하여 각 server에 저장된 갱신 대상 레코드를 찾아 데이터를 갱신한다.

Rowid 정보 수집 단계와 데이터 갱신 단계는 갱신 대상 레코드를 더 이상 찾을 수 없을 때까지 반복된다.

다음은 갱신 대상 레코드들간의 식별이 필요한 경우로써 generated query를 이용하여 데이터를 갱신할 수 없는 경우에 지원된다.

- Non-deterministic expression을 사용하는 경우
- Non-deterministic clause를 사용하는 경우
- 데이터 갱신을 위한 generated query를 구성할 수 없는 경우

다음은 rowid를 이용한 데이터 갱신을 지원하는 query 유형이다.

- SELECT .. FOR UPDATE
- SELECT .. INTO .. FOR UPDATE
- DELETE FROM
- DELETE FROM name RETURNING
- DELETE FROM name RETURNING .. INTO
- DELETE FROM name WHERE CURRENT OF cursor_name
- UPDATE
- UPDATE name RETURNING
- UPDATE name RETURNING .. INTO
- UPDATE name WHERE CURRENT OF cursor_name

Non-deterministic expression을 사용하는 경우

특정 expression으로 작성한 query를 수행할 때마다 동일한 결과가 나오는 것을 보장할 수 없는 expression을 non-deterministic expression 이라 한다.

Non-deterministic expression으로 작성한 generated query를 이용하여 데이터를 갱신할 경우 master server와 slave server에서 동일하게 갱신된다는 보장이 없다. 따라서 non-deterministic expression을 기술한 경우 rowid를 이용한 데이터 갱신을 지원한다.

- Non-deterministic expression에 해당하는 경우는 다음과 같다.
 - sequence
 - seq_name.currval, seq_name.nextval, next value for seq_name
 - currval(), nextval()
 - 각 expression 단위로 평가되어야 하는 built-in function
 - random()
 - clock_date(), clock_time(), clock_timestamp(), clock_localtime(), clock_localtimestamp()
 - psm
 - schema level function

다음은 non-deterministic expression을 사용하는 예이다.

- Sequence 사용


```
UPDATE t1 SET c1 = seq.nextval;
```

- Expression 단위로 평가되어야 하는 built-in function

```
DELETE FROM t1 WHERE c1 = random( 1, 100 );
```

- Schema level function

```
SELECT * FROM t1 WHERE shard_key = func( c1 );
```

Non-deterministic clause를 사용하는 경우

특정 clause로 작성한 query를 수행할 때마다 동일한 결과가 나오는 것을 보장할 수 없는 clause를 non-deterministic clause라고 한다.

Non-deterministic clause로 작성한 generated query를 이용하여 데이터를 갱신할 경우 master server와 slave server에 동일하게 갱신된다는 보장이 없다. 따라서 non-deterministic clause를 기술한 경우에는 rowid를 이용한 데이터 갱신을 지원한다.

〈offset limit clause〉 구문이 사용된 clause를 non-deterministic clause로 분류한다.

다음은 non-deterministic clause를 사용하는 예이다.

- OFFSET clause 사용

```
DELETE FROM t1 OFFSET 1;
```

- LIMIT clause 사용

```
UPDATE t1 SET c1 = c1 + 1 LIMIT 10;
```

데이터 갱신을 위한 generated query를 구성할 수 없는 경우

사용자로부터 주어진 데이터 갱신 query를 하나의 generated query로 구성할 수 없는 경우에 rowid를 이용한 데이터 변경을 지원한다. 자세한 내용은 **Generated Query 구성 제약 사항**을 참조한다.

다음은 generated query를 구성할 수 없는 경우의 예이다.

- 서로 다른 sharding 정책을 가진 테이블의 equi-join

```
SELECT * FROM t1, t2 WHERE t1.shard_id = t2.shard_id FOR UPDATE;
```

- 한 cluster group에 속한 t1 레코드를 탐색할 때 부질의 (subquery)의 t2 레코드를 탐색하기 위해 다른 cluster group에 접근하는 경우

```
DELETE FROM t1 WHERE t1.shard_id IN ( SELECT t2.shard_id FROM t2 );
```

13.

SQL Objects

본 장에서는 database를 구성하는 다음 객체들의 개념과 특징에 대해 설명한다.

- Authorization: User와 privilege
- Schema
- Tablespace
- Table
- Index
- Sequence
- View
- Synonym
- Stored procedure
- Stored function

13.1 Database

데이터베이스 관련 구문

자세한 내용은 다음 링크를 참조한다.

- 데이터베이스 구동: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**
- 백업 및 복구
 - **ALTER DATABASE BACKUP**
 - **ALTER DATABASE DELETE BACKUP**
 - **ALTER DATABASE RECOVER**
 - **ALTER DATABASE REGISTER**
 - **ALTER DATABASE RESTORE**
- 로그파일 생성, 제거, 변경
 - **ALTER SYSTEM CHECKPOINT**
 - **ALTER SYSTEM SWITCH LOGFILE**
 - **ALTER DATABASE ARCHIVELOG**
 - **ALTER DATABASE ADD LOGFILE**
 - **ALTER DATABASE DROP LOGFILE**
 - **ALTER DATABASE RENAME LOGFILE**
- 객체에 대한 주석: **COMMENT ON name IS**
- 시스템 통계 정보: **ANALYZE SYSTEM**

데이터베이스 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-1 데이터베이스 객체 관련 정보

스키마	View 이름	설명
DICTIONARY_SCHEMA	ALL_NONSCHEMA_COMMENTS	사용자가 접근 가능한 non-schema 객체의 주석 정보이다.
INFORMATION_SCHEMA	INFORMATION_SCHEMA_CATALOG_NAME	데이터베이스 이름 정보이다.
	SQL_FEATURES	GOLDILOCKS의 SQL 표준 호환성 정보이다.
	SQL_IMPLEMENTATION_INFO	GOLDILOCKS의 SQL 표준 호환성 정보이다.
	SQL_PACKAGES	GOLDILOCKS의 SQL 표준 호환성 정보이다.
	SQL_PARTS	GOLDILOCKS의 SQL 표준 호환성 정보이다.
	SQL_SIZING	GOLDILOCKS의 SQL 표준 호환성 정보이다.

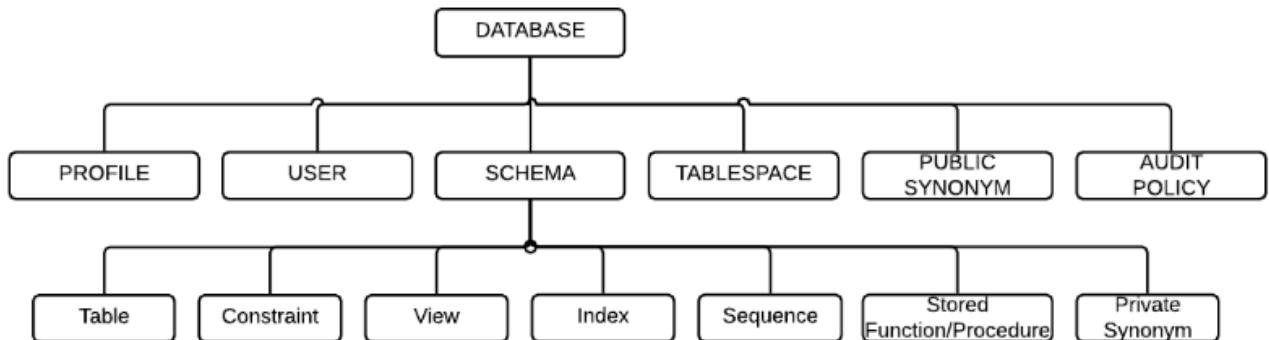
데이터베이스 구성 객체

데이터베이스를 구성하는 SQL 객체

데이터베이스는 다수의 SQL object로 구성된다.

데이터베이스에 포함되는 SQL object들은 schema에 포함되는지 여부에 따라 SQL schema object와 non-schema object로 구분된다.

그림 1 SQL objects



SQL schema object는 schema 내에 포함되는 객체로써 그 종류는 다음과 같다.

- TABLE: Column과 row로 구성되어 물리적 data를 저장하는 객체
- VIEW: 질의에 relation 이름을 부여하여 TABLE과 유사하게 접근하는 논리적 객체
- INDEX: 질의 성능을 향상시키기 위한 객체
- SEQUENCE: 번호를 생성하기 위한 객체
- CONSTRAINT: TABLE의 무결성을 유지하기 위한 객체
- SYNONYM: TABLE, VIEW, SEQUENCE, 기타 다른 SYNONYM에 대한 대체 이름
- STORED PROCEDURE: Procedure 형태의 persistent stored module 객체
- STORED FUNCTION: Function 형태의 persistent stored module 객체

SQL schema object는 schema 이름을 함께 명시해서 사용하거나 schema 이름을 생략하여 사용할 수도 있다. 생략할 경우의 schema 이름은 user의 schema path로 해석된다.

다음은 schema 이름을 명시하여 object를 생성하는 예이다.

```

gSQL> CREATE TABLE my_schema.lineitem ( id INTEGER );
gSQL> CREATE INDEX my_schema.my_index ON my_schema.lineitem ( id );
  
```

Non-schema object는 schema에 포함되지 않는 객체로써 그 종류는 다음과 같다.

- PROFILE: 비밀번호 관리 정책
- AUDIT POLICY: 감사 정책
- USER: 사용자

- SCHEMA: SQL schema object들의 논리적 위치
- TABLESPACE: SQL schema object들의 물리적 공간
- PUBLIC SYNONYM: 스키마 이름이 없는 TABLE, VIEW, SEQUENCE, 기타 다른 SYNONYM의 대체 이름

SQL 표준은 SCHEMA 객체의 개념과 구문을 명확하게 정의하고 있지만 USER와 DATABASE는 개념만 존재할 뿐 구문이 정의되어 있지 않다. TABLESPACE 객체는 SQL 표준에서 다루지 않는 객체이다. 즉, SQL 표준은 non-schema 객체의 관계를 명확하게 정의하지 않고 있다.

GOLDILOCKS는 USER, SCHEMA, TABLESPACE를 DATABASE 하위의 별도 객체로 정의하고 있는 반면에 다른 DBMS 업체들은 다음과 같이 non-schema 객체를 정의하고 있다.

- GOLDILOCKS
 - USER와 SCHEMA가 별도의 객체이다.
 - USER는 자신의 SCHEMA를 소유하지 않거나 여러 개의 SCHEMA를 소유할 수 있다.
 - USER : SCHEMA = 1 : N 관계이다.
- Oracle
 - USER와 SCHEMA를 유사한 개념으로 정의하고 있다.
 - USER : SCHEMA = 1 : 1 관계이다.
- DB2
 - USER가 DATABASE의 하위 객체가 아니다.
 - USER : SCHEMA = 1 : N 관계이다.
- Postgres
 - USER가 DATABASE의 하위 객체가 아니다.
 - USER : SCHEMA = 1 : N 관계이다.
- MySQL
 - DATABASE와 SCHEMA를 유사한 개념으로 정의하고 있다.
 - USER가 DATABASE (SCHEMA)의 하위 객체이다.

객체의 Name Space

데이터베이스 내의 객체는 식별 가능한 이름을 갖는다.

SQL schema object는 schema 내에서 중복되지 않은 이름을 갖는다.

다음 예와 같이 서로 다른 schema에 동일한 lineitem 테이블 객체를 생성할 수 있다.

```
gSQL> CREATE TABLE my_schema.lineitem ( id INTEGER );
gSQL> CREATE TABLE your_schema.lineitem ( name VARCHAR(128) );
```

SQL schema object는 하나의 schema 내에서 다음과 같은 name space를 갖는다.

- TABLE, VIEW, SEQUENCE, PRIVATE SYNONYM, STORED PROCEDURE, STORED FUNCTION
- INDEX
- CONSTRAINT

즉, 이름이 같은 table과 view는 생성할 수 없지만 이름이 같은 table과 index는 생성할 수 있다.

- Table과 view는 동일한 이름으로 생성할 수 없다.

```
gSQL> CREATE TABLE my_relation ( id INTEGER );
gSQL> CREATE VIEW my_relation ( name ) AS SELECT name FROM tmp_relation;
```

- Table과 index는 동일한 이름으로 생성할 수 있다.

```
gSQL> CREATE TABLE my_object ( id INTEGER );
gSQL> CREATE INDEX my_object ON my_table ( name );
```

Non-schema object는 database 내에서 식별 가능한 이름을 가지며 다음과 같은 name space를 갖는다.

- PROFILE
- AUDIT POLICY
- USER
- SCHEMA
- TABLESPACE

즉, 이름이 같은 USER는 생성할 수 없지만 이름이 같은 USER와 SCHEMA는 생성할 수 있다.

- my_name USER 객체와 my_name SCHEMA 객체가 생성된다.

```
gSQL> CREATE USER my_name IDENTIFIED BY my_name WITH SCHEMA my_name;
```

Built-in 객체

GOLDILOCKS는 database를 생성할 때 시스템 운영에 필요한 user, schema, tablespace 객체를 자동으로 생성한다.

Built-in User

Database를 생성할 때 자동으로 다음과 같은 계정을 생성한다. TEST user를 제외한 built-in 계정은 삭제할 수 없다.

- _SYSTEM 계정
 - 시스템 내부에서 사용하는 계정으로써 database를 생성할 때의 built-in 객체 소유자이며, 사용자가 객체를 생성할 때 객체의 소유자에게 권한을 부여하는 grantor가 된다.
- SYS 계정
 - Database를 관리하기 위한 user이다.
- TEST 계정

- 테스트 하기 위해 생성된 user로써 제거 가능한 객체이다.
- ADMIN 계정
 - Database를 관리하기 위한 role이다.
- SYSDBA 계정
 - Database를 구동/ 종료하기 위한 role이다.
- PUBLIC 계정
 - 모든 사용자를 의미하는 계정으로써 모든 사용자의 권한을 제어하기 위해 사용한다.

Built-in Schema

Database를 생성할 때 다음과 같은 스키마를 자동으로 생성한다. 모든 built-in 스키마는 제거할 수 없다.

- DEFINITION_SCHEMA
 - Database의 모든 객체 정보를 저장하고 있는 물리적 테이블들로 구성된다.
- FIXED_TABLE_SCHEMA
 - System의 자료 구조 정보를 테이블 형태로 보여주는 fixed table들로 구성된다.
- DICTIONARY_SCHEMA
 - Database의 객체 정보를 조회하기 위한 사용자 관점의 view들로 구성된다.
- INFORMATION_SCHEMA
 - Database의 객체 정보를 조회하기 위해 SQL 표준에서 정의한 view들로 구성된다.
- PERFORMANCE_VIEW_SCHEMA
 - System의 상태를 조회하기 위한 사용자 관점의 view들로 구성된다.
- PUBLIC
 - 모든 사용자가 객체를 생성할 수 있는 스키마로써 모든 사용자를 의미하는 PUBLIC 계정과는 다른 의미이다.

Built-in Tablespace

Database를 생성할 때 다음과 같은 테이블스페이스를 자동으로 생성한다. 모든 built-in 테이블스페이스는 제거할 수 없다.

- DICTIONARY_TBS
 - SQL 객체 정보를 관리하기 위한 dictionary table들을 저장한다.
- MEM_UNDO_TBS
 - Undo segment들과 transaction 정보들을 저장한다.
- MEM_DATA_TBS
 - 최초로 생성되는 data tablespace이다.
 - Data tablespace는 사용자가 생성한 table 등의 객체를 저장한다.
- MEM_TEMP_TBS
 - 최초로 생성되는 temporary tablespace이다.
 - Temporary tablespace는 질의 처리 중에 생성되는 sort나 hash와 같은 임시 객체를 저장한다.
- MEM_AUX_TBS

- System auxiliary tablespace이며 audit record와 같이 자동으로 생성되는 레코드를 저장한다.

Built-in Profile

Database를 생성할 때 다음과 같은 "DEFAULT" profile이 자동으로 생성된다. 자동 생성되는 "DEFAULT" profile의 password parameter 정보는 다음과 같다.

표 13-2 DEFAULT profile 구성

Parameter	Value
FAILED_LOGIN_ATTEMPTS	10
PASSWORD_LOCK_TIME	1
PASSWORD_LIFE_TIME	180
PASSWORD_GRACE_TIME	7
PASSWORD_REUSE_MAX	UNLIMITED
PASSWORD_REUSE_TIME	UNLIMITED
PASSWORD_VERIFY_FUNCTION	NULL

"DEFAULT" profile 기본값들의 특성은 다음과 같다.

- 계정 잠금
 - 10번 연속으로 login에 실패할 경우 (FAILED_LOGIN_ATTEMPTS) 1일 (PASSWORD_LOCK_TIME) 동안 계정을 잠근다.
- 비밀번호 만료
 - 180일 (PASSWORD_LIFE_TIME)이 경과한 후 7일 (PASSWORD_GRACE_TIME) 간의 유예 기간 후에 비밀번호가 만료된다.
- 비밀번호 재사용 가능 여부
 - 이전 비밀번호를 재사용할 수 있다.
- 비밀번호 복잡도 검사
 - 검사하지 않는다.

13.2 Profile

Profile 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Profile 생성: **CREATE PROFILE**
- Profile 제거: **DROP PROFILE**
- Profile 변경: **ALTER PROFILE**
- User에 profile 적용: **CREATE USER, ALTER USER**
- Password 이력 제거: **ALTER DATABASE CLEAR PASSWORD HISTORY**

표 13-3 Profile 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	DBA_PROFILES	모든 profile 정보
	DBA_USERS	사용자의 profile 정보

Profile 개념

GOLDILOCKS는 데이터베이스 보안을 위해 사용자 인증을 수행한다. 이 때 비밀번호가 사용되는데, 비밀번호는 도난, 위조, 잘못된 사용에 취약하므로 비밀번호 관리 정책이 필요하다.

Profile은 이와 같은 비밀번호 관리 정책 정보를 포함한다. DBA 또는 보안책임자는 profile을 사용자에게 할당하여 사용자의 역할에 적합한 비밀번호 관리 정책을 적용한다.

Profile 생성, 변경 및 할당

CREATE PROFILE 구문을 사용하여 profile을 생성한다.

```
CREATE PROFILE profile1 LIMIT
    FAILED_LOGIN_ATTEMPTS    10
    PASSWORD_LOCK_TIME       1
    PASSWORD_LIFE_TIME        180
    PASSWORD_GRACE_TIME       7
    PASSWORD_REUSE_MAX        UNLIMITED
    PASSWORD_REUSE_TIME       UNLIMITED
    PASSWORD_VERIFY_FUNCTION  NULL;
```

CREATE USER 또는 ALTER USER 구문을 사용하여 profile을 할당한다.

```
CREATE USER u1 IDENTIFIED BY u1 PROFILE profile1;
ALTER USER u2 PROFILE profile1;
```

ALTER PROFILE 구문을 사용하여 profile의 파라미터들을 변경한다.

```
ALTER PROFILE profile1 LIMIT
    PASSWORD_REUSE_MAX          3
    PASSWORD_REUSE_TIME        30;
```

사용자에게 profile을 할당하지 않으면 사용자는 비밀번호 생성 및 사용에 아무런 제약을 받지 않는다. 생성한 profile이나 DEFAULT profile을 사용자에게 할당하면 사용자는 비밀번호를 생성하거나 사용할 때 해당 profile의 비밀번호 정책을 따른다.

DEFAULT profile의 비밀번호 설정

사용자에게 DEFAULT profile을 할당하면 비밀번호는 다음과 같이 관리된다.

표 13-4 비밀번호의 DEFAULT profile

Parameter	Default setting	설명
FAILED_LOGIN_ATTEMPTS	10	허용된 연속 login 실패 횟수이다. 연속적으로 10 번을 초과하여 login에 실패할 경우, 계정이 잠긴다.
PASSWORD_LOCK_TIME	1	계정 잠금 기간이다. 허용된 연속 login 실패 횟수를 초과하는 경우, 1일 동안 계정이 잠긴다.
PASSWORD_LIFETIME	180	비밀번호 수명이다. 180일이 지나면 비밀번호가 만료된다.
PASSWORD_GRACE_TIME	7	비밀번호가 만료되었을 때, 비밀번호를 변경해야 하는 기간이다. 비밀번호가 만료된 첫 번째 로그인 이후로 7일 이내에 비밀번호를 변경해야 한다. 이 시기에 비밀번호를 변경하지 않으면 해당 비밀번호로 로그인 할 수 없게 된다.
PASSWORD_REUSE_MAX	UNLIMITED	비밀번호를 재사용할 수 없는 횟수이다. PASSWORD_REUSE_MAX와 PASSWORD_REUSE_TIME은 반드시 함께 설정되어야 하는데 두 값이 모두 UNLIMITED 인 경우, 비밀번호는 항상 재사용 가능하다.
PASSWORD_REUSE_TIME	UNLIMITED	비밀번호가 재사용될 수 없는 기간이다.

계정 잠금

연속적인 login 실패가 FAILED_LOGIN_ATTEMPTS에 명시된 횟수를 초과하는 경우, PASSWORD_LOCK_TIME에 명시된 기간만큼 계정이 잠긴다.

```
CREATE PROFILE profile1 LIMIT
    FAILED_LOGIN_ATTEMPTS    10
    PASSWORD_LOCK_TIME       1;
ALTER USER u1 PROFILE profile1;
```

u1 사용자가 연속적으로 10번 넘게 login에 실패하면, 1일동안 계정이 잠긴다. 그리고 1일이 지나면 자동으로 잠긴 계정이 해제된다.

PASSWORD_LOCK_TIME을 명시하지 않으면, default profile의 PASSWORD_LOCK_TIME에 명시된 값으로 간주된다.

PASSWORD_LOCK_TIME이 UNLIMITED면 잠긴 계정이 자동으로 해제되지 않기 때문에 다음 구문을 수행하여 계정을 해제해 주어야 한다.

```
ALTER USER u1 ACCOUNT UNLOCK;
```

Login에 성공하면 login 실패 횟수는 0으로 초기화 된다.

보안 책임자는 명시적으로 사용자 계정을 잠글 수 있는데, 이 경우에는 자동으로 해제되지 않기 때문에 보안 책임자가 잠근 계정을 해제해 주어야 한다.

```
ALTER USER u1 ACCOUNT LOCK;
ALTER USER u1 ACCOUNT UNLOCK;
```

비밀번호 수명

PASSWORD_LIFE_TIME은 비밀번호 수명으로써 이 기간이 지나면 비밀번호는 만료된다.

비밀번호가 만료되면, 사용자, DBA 또는 보안 담당자가 비밀번호를 변경해야만 한다.

```
CREATE PROFILE profile1 LIMIT
    PASSWORD_LIFE_TIME        180
    PASSWORD_GRACE_TIME       7;
ALTER USER u1 PROFILE profile1;
```

180일이 지난 후에 u1 사용자가 처음으로 login 하려고 시도할 때부터 grace time에 들어가게 된다.

Grace time인 7일 동안에는 사용자가 비밀번호를 변경할 때까지 사용자가 계정에 접근할 때마다 새로운 비밀번호를 입력하도록 상기시킨다.

Grace time인 7일이 경과된 후에도 비밀번호가 변경되지 않으면 새로운 비밀번호가 설정될 때까지 계정 접근이 거부된다.

CREATE USER 또는 ALTER USER 구문을 사용하여 비밀 번호를 만료시킬 수도 있다.

```
ALTER USER u1 PASSWORD EXPIRE;
```

비밀번호가 만료된 후 사용자가 접속하면 아래의 예와 같이 비밀번호 만료 에러 (ERR-28000(16312) the password has expired)가 발생하며, 새로운 비밀번호를 입력해야 한다.

```
% gsql u1 u1
ERR-28000(16312): the password has expired
Changing password for u1
New password:
Retype new password:
Connected to GOLDILOCKS Database.
gSQL>
```

비밀번호 재사용

동일한 비밀번호를 재사용하려면 PASSWORD_REUSE_MAX에 설정된 횟수만큼 다른 비밀번호로 변경해야 하고 PASSWORD_REUSE_TIME에 설정된 기간만큼 시간이 경과해야 한다.

```
CREATE PROFILE profile1 LIMIT
    PASSWORD_REUSE_MAX      2
    PASSWORD_REUSE_TIME     10;
ALTER USER u1 PROFILE profile1;
```

u1 사용자가 현재 비밀번호를 재사용하기 위해서는 비밀번호를 두 번 변경해야 하고 10일이 지나야 한다.

```
ALTER USER u1 IDENTIFIED BY u1 REPLACE u1;
ALTER USER u1 IDENTIFIED BY u1 REPLACE u1
*
ERROR at line 1:
ORA-28007: the password cannot be reused
ALTER USER u1 IDENTIFIED BY u2 REPLACE u1;
User altered.
ALTER USER u1 IDENTIFIED BY u3 REPLACE u2;
User altered.
```

- 10일 경과

```
ALTER USER u1 IDENTIFIED BY u1 REPLACE u3;
User altered.
```

반드시 두 조건을 모두 만족해야 재사용 가능하며, PASSWORD_REUSE_MAX, PASSWORD_REUSE_TIME 두 값 중 하나만 UNLIMITED인 경우에는 비밀번호를 재사용할 수 없다.
두 값이 모두 UNLIMITED면 항상 재사용 가능하다.

표 13-5 비밀번호 재사용

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	Password Reuse 여부
Integer value	Integer value	두 조건 모두 만족해야 재사용 가능
Integer value	UNLIMITED	재사용 불가
UNLIMITED	Integer value	재사용 불가
UNLIMITED	UNLIMITED	항상 재사용 가능

비밀번호 복잡도 검증

시스템을 공격하려는 침입자들이 쉽게 추측할 수 없을 만큼 복잡한 비밀번호인지 여부를 검증한다.
GOLDILOCKS에서 제공하는 비밀번호 복잡도 검증 방법은 다음과 같다.

표 13-6 비밀번호 복잡도 검증

복잡도 검증 방법	복잡도 검증 내용
KISA_VERIFY_FUNCTION	8 자리 이상 1 개 이상의 문자 1 개 이상의 숫자 1 개 이상의 특수문자
ORA12C_VERIFY_FUNCTION	8 자리 이상 1 개 이상의 문자 1 개 이상의 숫자 - Database name을 포함하면 안된다. - 사용자 이름 또는 거꾸로 된 사용자 이름을 포함하면 안된다. - goldilocks를 포함하면 안된다. - oracle을 포함하면 안된다. - 다음과 같이 단순한 비밀번호는 사용할 수 없다. - welcome1, database1, account1, user1234, password1, oracle123, computer1, abcdefg1, change_on_intall - 이전 비밀번호와 적어도 3 자리는 달라야 한다.
ORA12C_STRONG_VERIFY_FUNCTION	9 자리 이상 2 개 이상의 대문자 2 개 이상의 소문자 2 개 이상의 숫자 2 개 이상의 특수 문자 - 이전 비밀번호와 적어도 4 자리는 달라야 한다.
VERIFY_FUNCTION_11G	8 자리 이상 1 개 이상의 문자 1 개 이상의 숫자

복잡도 검증 방법	복잡도 검증 내용
	• 사용자 이름을 포함하면 안된다. • 이전 비밀번호와 적어도 3 자리는 달라야 한다.
VERIFY_FUNCTION	• 사용자 이름과 같아선 안된다. • 4 자리 이상 • 1 개 이상의 문자 • 1 개 이상의 숫자 • 1 개 이상의 특수문자 • 다음과 같이 단순한 비밀번호는 사용할 수 없다. ◦ welcome, database, account, user, password, oracle, computer, abcd • 이전 비밀번호와 적어도 3 자리는 달라야 한다.

Profile과 사용자 설정에 대한 자세한 내용은 **CREATE PROFILE** 구문과 **CREATE USER** 구문을 참조한다.

13.3 Audit Policy

Audit Policy 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Audit policy 생성: **CREATE AUDIT POLICY**
- Audit policy 제거: **DROP AUDIT POLICY**
- Audit policy 변경: **ALTER AUDIT POLICY**
- Audit policy 활성화: **AUDIT POLICY**
- Audit policy 비활성화: **NOAUDIT POLICY**
- Audit trail 삭제: **ALTER DATABASE CLEAR AUDIT TRAIL**

표 13-7 Audit policy 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	AUDIT_POLICIES	모든 audit policy 정보
	AUDIT_POLICY_OPTIONS	Audit policy 옵션 정보
	AUDIT_POLICY_ENABLED	Audit policy 활성화 정보

사용 예

다음과 같이 수행하려면 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

Audit Policy 생성

Audit policy 객체를 생성하기 위해 **CREATE AUDIT POLICY** 구문을 수행한다.

다음 구문은 u1.t1 테이블에 대한 DML을 감사하기 위해 audit_t1_dml 객체를 생성하는 예이다.

```
CREATE AUDIT POLICY audit_t1_dml
  ACTIONS INSERT ON u1.t1
    , DELETE ON u1.t1
    , UPDATE ON u1.t1
;
Audit policy created.
```

AUDIT_POLICY_OPTIONS view를 조회하여 audit policy의 옵션 정보를 확인한다.


```

SELECT policy_name
       , audit_option
       , object_schema
       , object_name
FROM   audit_policy_options
WHERE  policy_name = 'AUDIT_T1_DML'
ORDER BY audit_option
;

```

POLICY_NAME	AUDIT_OPTION	OBJECT_SCHEMA	OBJECT_NAME
AUDIT_T1_DML	DELETE	U1	T1
AUDIT_T1_DML	INSERT	U1	T1
AUDIT_T1_DML	UPDATE	U1	T1

3 rows selected.

Audit Policy 활성화

Audit policy를 활성화하려면 **AUDIT POLICY** 구문을 사용해야 한다.

다음은 u1, sys 이외의 사용자가 u1.t1 테이블에 대해 DML을 성공적으로 수행했을 경우, audit record를 남기도록 audit policy를 활성화하는 예이다.

```

AUDIT POLICY audit_t1_dml
  EXCEPT u1, sys
  WHENEVER SUCCESSFUL
;

```

활성화된 audit policy는 새로 생성되는 session에 적용되고 기존 session에는 영향을 주지 않는다.

Audit policy의 활성화 정보는 **AUDIT_POLICY_ENABLED** view를 조회하여 확인할 수 있다.

```

SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
ORDER BY user_name
;

```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
-----	-----	-----	-----	-----

```

AUDIT_T1_DML      EXCEPT      SYS      YES      NO
AUDIT_T1_DML      EXCEPT      U1       YES      NO
2 rows selected.

```

Audit policy를 활성화 한 뒤에 조건에 부합하는 action들이 audit record를 생성한다.

다음은 u2 사용자가 SQL 구문들을 모두 성공적으로 수행하는 예이다.

```

SELECT COUNT(*) FROM u1.t1;
INSERT INTO u1.t1 VALUES ( 1 );
UPDATE u1.t1 SET id = id + 1 WHERE id = 1;
DELETE u1.t1 WHERE id = 2;
COMMIT;

```

위의 예에서 INSERT, UPDATE, DELETE는 감사 대상 action이기 때문에 audit record를 생성하지만 SELECT, COMMIT은 감사 대상 action이 아니므로 audit record를 생성하지 않는다.

Audit Trail 조회

Audit record를 조회하기 위해 AUDIT_TRAIL view에 대해 SELECT ON DICTIONARY_SCHEMA.AUDIT_TRAIL 권한이 있어야 한다.

다음은 audit_t1_dml 감사 정책에 의해 생성된 audit trail을 조회하는 예이다.

```

SELECT policy_name
       , logon_username
       , action_name
       , object_schema
       , object_name
       , sql_text
FROM audit_trail
WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  LOGON_USERNAME ACTION_NAME OBJECT_SCHEMA OBJECT_NAME SQL_TEXT
-----
AUDIT_T1_DML U2          INSERT      U1          T1          INSERT INTO u1.t1 VALUES ( 1
)
AUDIT_T1_DML U2          UPDATE      U1          T1          UPDATE u1.t1 SET c1 = c1 + 1
WHERE c1 = 1
AUDIT_T1_DML U2          DELETE      U1          T1          DELETE u1.t1 WHERE c1 = 2
3 rows selected.

```

Audit Trail 소거

Audit policy를 활성화하면 audit trail의 크기가 계속 증가한다.

Audit trail을 삭제하려면 다음 구문을 수행해야 한다.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Audit trail을 저장하려면 다음 예와 같이 사용자 테이블에 저장한 후 삭제한다.

- 사용자 테이블 생성

```
CREATE TABLE my_audit_trail
AS SELECT *
FROM audit_trail
WITH NO DATA;
```

- 사용자 테이블에 저장한 후 삭제

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
ALTER DATABASE CLEAR AUDIT TRAIL;
```

저장된 audit trail과 현재 audit_trail을 함께 조회하고 싶은 경우 다음과 같이 view를 만들어 관리한다.

```
CREATE VIEW unified_audit_trail
AS
SELECT * FROM my_audit_trail
UNION ALL
SELECT * FROM dictionary_schema.audit_trail
;
```

Audit Policy 비활성화

다음 구문을 사용하여 audit policy를 비활성화한다.

```
NOAUDIT POLICY audit_t1_dml;
```

Audit policy 비활성화는 새로운 session에만 영향을 미치며, 기존 session의 활성화 정보에는 영향을 미치지 않는다.

Audit Policy 삭제

Audit policy 객체를 삭제하려면 다음 구문을 수행한다.

```
DROP AUDIT POLICY audit_t1_dml;
```

Audit policy 객체를 삭제하려면 비활성 상태이어야 하는데 객체를 삭제해도 기존 session에는 영향을 미치지 않는다.

Audit Policy 개념

Audit Trail

Audit Trail 조회

Audit record는 DICTIONARY_SCHEMA.AUDIT_TRAIL view를 사용하여 조회할 수 있다.

AUDIT_TRAIL view를 조회하기 위해서는 SELECT 권한이 있어야 한다.

```
GRANT SELECT ON DICTIONARY_SCHEMA.AUDIT_TRAIL TO user_name;
```

AUDIT_TRAIL view는 다음과 같은 정보를 갖는다.

표 13-8 Column 정보

Information	Column name	Description
Session information	MEMBER_NAME	Cluster member name
	SESSION_ID	Session identifier
	SESSION_SERIAL	Session serial number
	LOGON_USERNAME	Logon user name of the user whose actions were audited
	CURRENT_USERNAME	Effective user for the statement execution
	SERVER_PROCESS	Server process identifier for the session
Peer client information	CLIENT_PROGRAM_NAME	Client program used for session
	CLIENT_USERNAME	Client operating system user name for the session
	CLIENT_PROCESS	Client process identifier for the session
	CLIENT_HOST	Client host ip address for the session
	CLIENT_PORT	Client port number for the session
	CLIENT_TERMINAL	Client terminal name for the session
SQL information	TRANSACTION_ID	Transaction identifier
	SCN	System change number (SCN) string of the query at the time of the event
	GCN	Global change number (GCN) of the query at the time of the event
	DCN	Domain change number (DCN) of the query at the time of the event
	LCN	Local change number (LCN) of the query at the time of the event
	STMT_NO	Numeric number for each statement run in a session

	SQL_TEXT	SQL associated with the event
	SQL_BINDS	List of bind variables, if any, associated with SQL_TEXT
	RETURN_CODE	Error code generated by the action, zero if the action succeeded
	ERROR_MESSAGE	Error message generated by the action, null if the action succeeded
Event information	ENTRY_ID	Audit trail entry identifier in the session
	EVENT_TIMESTAMP	Timestamp of the creation of the audit trail entry in local time zone
	POLICY_NAME	Audit policy name that caused the current audit record
	PRIVILEGE_USED	Database privilege used to execute the action
	ACTION_NAME	Action name executed by the user
	OBJECT_TYPE	Object type of object affected by the action
	OBJECT_SCHEMA	Schema name of object affected by the action
	OBJECT_NAME	Object name of object affected by the action

Audit Trail 저장

AUDIT_TRAIL view를 구성하는 table들은 다음과 같다.

- AUDIT_TRAIL_SESSION
 - Session당 하나의 record를 기록한다.
 - Session information
 - Peer client information
- AUDIT_TRAIL_SQL
 - SQL당 하나의 record를 기록한다.
 - SQL information
- AUDIT_TRAIL_EVENT
 - Audit option당 하나의 record를 기록한다.
 - Audit event information

Audit trail을 구성하는 audit record들은 다수의 table로 분할되어 저장된다.

Table들의 schema는 DEFINITION_SCHEMA이며, MEM_AUX_TBS 테이블스페이스에 저장된다.

- DEFINITION_SCHEMA
 - Dictionary table들을 저장하고 있는 schema이다.
- MEM_AUX_TBS
 - System auxiliary tablespace
 - Database가 자동으로 생성하는 record를 저장하기 위한 tablespace이다.
 - 자동으로 생성하는 record가 서비스 운영에 영향을 주지 않도록 하기 위해 별도의 tablespace로 관리한다.

Audit Record 생성

Audit policy를 활성화하면 조건에 부합하는 action이 발생했을 때 audit record를 생성한다.

다수의 감사 조건에 부합하는 action이 발생할 경우, 하나 이상의 audit record를 생성한다.

- 다음과 같이 유사한 audit option을 나열한 경우 하나의 audit record를 생성한다.
 - Audit policy 정의

```
CREATE AUDIT POLICY p1
  PRIVILEGES INSERT ANY TABLE
  ACTIONS INSERT;
AUDIT POLICY p1;
```

- Audit action 수행

```
INSERT INTO other_user.t1 VALUES ( 1 );
```

- 다음과 같이 서로 다른 audit option을 나열한 경우, 두 개의 audit record를 생성한다.
 - Audit policy 정의

```
CREATE AUDIT POLICY p1
  ACTIONS SELECT ON u1.t1
    , SELECT ON u1.t2;
AUDIT POLICY p1;
```

- Audit action 수행

```
SELECT COUNT(*) FROM u1.t1 A, u1.t2 B WHERE A.id = B.id;
```

- 다음과 같이 동일한 action에 다수의 audit policy를 활성화한 경우, 두 개의 audit record를 생성한다.
 - Audit policy 정의

```
CREATE AUDIT POLICY p1
  PRIVILEGES INSERT ANY TABLE;
AUDIT POLICY p1;
CREATE AUDIT POLICY p2
  ACTIONS INSERT;
AUDIT POLICY p2;
```

- Audit action 수행

```
INSERT INTO other.t1 VALUES (1);
```

Audit Trail 삭제 (purge)

Audit trail을 삭제하려면 다음 구문을 수행해야 한다.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

과거의 audit record를 보관하려면 다음과 같이 사용자 테이블을 만들어 저장해야 한다.

```
CREATE TABLE my_audit_trail AS SELECT * FROM audit_trail WITH NO DATA;
```

이후 주기적으로 audit trail을 삭제하기 전에 INSERT .. SELECT 구문으로 저장한다.

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
ALTER DATABASE CLEAR AUDIT TRAIL;
```

만약 특정 기간 동안의 audit record를 보관하려면 EVENT_TIMESTAMP column을 사용하여 DELETE 구문을 수행해야 한다.

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
DELETE FROM my_audit_trail WHERE event_timestamp < ADD_MONTHS( sysdate, -3 );
ALTER DATABASE CLEAR AUDIT TRAIL;
```

과거의 audit record와 현재 audit record를 함께 조회하려면 다음과 같이 view를 만들어서 조회해야 한다.

```
CREATE VIEW audit_trail_view
AS SELECT * FROM dictionary_schema.audit_trail
   UNION ALL
   SELECT * FROM my_audit_trail;
SELECT * FROM audit_trail_view;
```

Audit Policy 설정

Audit policy에 포함할 수 있는 옵션은 다음과 같다.

- Privilege auditing
 - Database privilege를 사용하여 SQL 수행을 감사한다.
- Object action auditing
 - 특정 객체에 대한 SQL 수행을 감사한다.
- System action auditing
 - 모든 객체에 대한 SQL 수행을 감사한다.

다수의 audit policy를 생성하여 관리할 수도 있지만 적은 수의 audit policy로 다수의 audit option을 관리하는 것이 바람직하다.

활성화된 audit policy 정보들은 logon 시점에 session 정보로 구축되므로 audit policy 개수가 적을수록 부하가 줄어든다.

또한 다수의 audit policy를 활성화하면 하나의 SQL 문장에 대한 audit record 생성 여부를 판단하여 다수의 audit record를 생성하는 부하가 발생할 수 있다.

Logon 시점에 session 내에 구축된 audit policy 정보는 audit policy 제거, 변경, 활성화, 비활성화 등에 영향을 받지 않는다.

Audit policy 변경은 새로 logon 하는 session에만 적용된다.

Privilege Auditing

Privilege auditing은 database privilege를 이용해 SQL 문장을 성공적으로 수행한 경우에 이를 감사하기 위해 설정한다.

Database의 소유자인 sys 사용자에게 대해서는 privilege auditing에 따른 audit record를 생성하지 않는다.

Privilege auditing을 위해 나열할 수 있는 database privilege는 **V\$AUDITABLE_DB_PRIVILEGES** view 를 조회하여 확인할 수 있다.

```
SELECT privilege_name FROM v$auiditable_db_privileges;
PRIVILEGE_NAME
-----
ADMINISTRATION
ALTER DATABASE
ALTER SYSTEM
ACCESS CONTROL
CREATE USER
ALTER USER
DROP USER
CREATE TABLESPACE
ALTER TABLESPACE
DROP TABLESPACE
USAGE TABLESPACE
CREATE SCHEMA
DROP SCHEMA
ANALYZE ANY
CREATE ANY TABLE
ALTER ANY TABLE
DROP ANY TABLE
SELECT ANY TABLE
INSERT ANY TABLE
```



```

DELETE ANY TABLE
UPDATE ANY TABLE
LOCK ANY TABLE
CREATE ANY VIEW
DROP ANY VIEW
CREATE ANY SEQUENCE
ALTER ANY SEQUENCE
DROP ANY SEQUENCE
USAGE ANY SEQUENCE
CREATE ANY INDEX
ALTER ANY INDEX
DROP ANY INDEX
CREATE ANY SYNONYM
DROP ANY SYNONYM
CREATE PUBLIC SYNONYM
DROP PUBLIC SYNONYM
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
CREATE ANY PROCEDURE
ALTER ANY PROCEDURE
DROP ANY PROCEDURE
EXECUTE ANY PROCEDURE
AUDIT SYSTEM
43 rows selected.

```

다음은 SELECT ANY TABLE 권한을 가지고 있는 사용자 u1이 privilege auditing을 위해 audit policy를 활성화하는 예이다.

```

CREATE AUDIT POLICY p1
    PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;

```

사용자 u1이 다음 두 구문을 수행할 때 audit record 생성 여부는 다음과 같다.

- SELECT * FROM u1.t1
 - 객체의 소유자이기 때문에 SELECT ANY TABLE 권한을 사용하지 않는다.
 - Audit record가 없다.
- SELECT * FROM other.t1
 - SELECT ON other.t1 권한이 없으나 SELECT ANY TABLE 권한으로 성공한다.
 - Audit record를 생성한다.

Privilege auditing 정보를 확인하려면 다음과 같이 AUDIT_POLICY_OPTIONS view를 조회해야 한다.

```
SELECT audit_option
       , audit_option_type
       , object_schema
       , object_name
FROM audit_policy_options
WHERE policy_name = 'P1'
;
```

AUDIT_OPTION	AUDIT_OPTION_TYPE	OBJECT_SCHEMA	OBJECT_NAME
SELECT ANY TABLE	DATABASE PRIVILEGE	null	null

Object Action Auditing

특정 객체에 대해 수행하는 SQL을 감사한다.

각 객체 유형별로 감사할 수 있는 action은 다음과 같다.

표 13-9 객체 유형별 audit action

객체 유형	Actions
Table	ALTER, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE
View	ALTER, COMMENT, GRANT, SELECT
Sequence	ALTER, COMMENT, GRANT, SELECT
Stored function/ procedure	ALTER, COMMENT, EXECUTE, GRANT

예를 들어, 테이블 u1.t1에 대한 DML을 감사하려면 다음과 같이 audit policy를 생성한다.

```
CREATE AUDIT POLICY audit_t1_dml
ACTIONS INSERT ON u1.t1
       , DELETE ON u1.t1
       , UPDATE ON u1.t1
;
```

Object action auditing에 대한 정보는 다음과 같이 AUDIT_POLICY_OPTIONS view를 조회하여 확인한다.

```
SELECT audit_option
       , audit_option_type
       , object_schema
       , object_name
FROM audit_policy_options
```

```

WHERE policy_name = 'AUDIT_T1_DML'
ORDER BY audit_option
;

```

AUDIT_OPTION	AUDIT_OPTION_TYPE	OBJECT_SCHEMA	OBJECT_NAME
DELETE	OBJECT ACTION	U1	T1
INSERT	OBJECT ACTION	U1	T1
UPDATE	OBJECT ACTION	U1	T1

3 rows selected.

ALL ON schema.object와 같은 ALL 옵션은 해당 객체에 대해 정의할 수 있는 모든 audit action을 의미한다.

다음은 ALL 옵션과 다른 옵션을 함께 사용하는 예이다.

```

CREATE AUDIT POLICY p1
    ACTIONS ALL ON u1.seq1
    , ALTER ON u1.seq1
;

```

Audit policy created.

```

SELECT audit_option
    , audit_option_type
    , object_schema
    , object_name
FROM audit_policy_options
WHERE policy_name = 'P1'
ORDER BY audit_option
;

```

AUDIT_OPTION	AUDIT_OPTION_TYPE	OBJECT_SCHEMA	OBJECT_NAME
ALL	OBJECT ACTION	U1	SEQ1
ALTER	OBJECT ACTION	U1	SEQ1

2 rows selected.

다음과 같이 ALL 옵션을 삭제하면 모든 audit option이 삭제되는 것이 아니라 ALL 옵션만 삭제된다.

```

ALTER AUDIT POLICY p1
    DROP ACTIONS ALL ON u1.seq1;

```

Audit Policy altered.

```

SELECT audit_option
    , audit_option_type
    , object_schema
    , object_name

```

```

FROM audit_policy_options
WHERE policy_name = 'P1'
ORDER BY audit_option;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
-----
ALTER          OBJECT ACTION      U1          SEQ1
1 row selected.

```

Stored function이나 stored procedure의 EXECUTE 성공, 실패에 대한 감사는 실제 수행 시점에 수행 가능 여부만으로 결정된다.

- WHENEVER NOT SUCCESSFUL는 stored function/ procedure를 수행할 수 없을 때 감사 레코드를 생성한다.
- WHENEVER SUCCESSFUL은 stored function/procedure 내부의 SQL 구문 수행 중에 에러가 발생하더라도 감사 레코드를 생성한다.
- Stored function/ procedure 내부의 SQL 구문 실패에 대한 감사가 필요할 경우, 해당 SQL 구문을 감사 대상에 포함시켜야 한다.

다음은 stored function을 포함한 SELECT 구문을 수행하는 예이다.

```

SELECT others.func1( t1.c1 )
FROM t1;

```

권한 부족과 같은 오류로 인해 others.func1() 호출에 실패하는 경우는 WHENEVER NOT SUCCESSFUL에 해당하고, others.func1() 호출에 성공하면 stored function 내의 SQL 구문을 수행할 때 에러가 발생하더라도 WHENEVER SUCCESSFUL에 해당한다.

System Action Auditing

특정 객체와 관계없이 SQL 구문을 감사한다.

유효한 system action은 V\$AUDITABLE_SYSTEM_ACTIONS를 조회한다.

```

SELECT action_name FROM v$auditable_system_actions;
ACTION_NAME
-----
ALL
DDL
SELECT
INSERT
UPDATE
DELETE
EXECUTE
CREATE TABLE
DROP TABLE

```

ALTER TABLE
LOCK TABLE
TRUNCATE TABLE
ANALYZE TABLE
RENAME
CREATE INDEX
DROP INDEX
ALTER INDEX
CREATE SEQUENCE
DROP SEQUENCE
ALTER SEQUENCE
GRANT
REVOKE
CREATE SYNONYM
DROP SYNONYM
CREATE VIEW
DROP VIEW
ALTER VIEW
CREATE PROCEDURE
DROP PROCEDURE
ALTER PROCEDURE
CREATE FUNCTION
DROP FUNCTION
ALTER FUNCTION
COMMENT
ALTER DATABASE
CREATE PROFILE
DROP PROFILE
ALTER PROFILE
CREATE TABLESPACE
DROP TABLESPACE
ALTER TABLESPACE
CREATE USER
DROP USER
ALTER USER
CHANGE PASSWORD
CREATE SCHEMA
DROP SCHEMA
CREATE AUDIT POLICY
DROP AUDIT POLICY
ALTER AUDIT POLICY

```

AUDIT
NOAUDIT
ALTER SYSTEM
ALTER SESSION
ANALYZE SYSTEM
COMMIT
ROLLBACK
SAVEPOINT
LOGON
LOGOFF
SET SESSION
SET TRANSACTION
SET CONSTRAINTS
CREATE CLUSTER GROUP
DROP CLUSTER GROUP
ALTER CLUSTER GROUP
CREATE CLUSTER LOCATION
DROP CLUSTER LOCATION
ALTER CLUSTER LOCATION
69 rows selected.

```

각 SQL 구문에 해당하는 system action 이름은 V\$SQL_COMMAND view를 조회하여 확인한다.

```

SELECT command, audit_action FROM v$sql_command;

```

COMMAND	AUDIT_ACTION
ALTER AUDIT POLICY	ALTER AUDIT POLICY
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER	ALTER CLUSTER GROUP
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER	ALTER CLUSTER GROUP
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	ALTER DATABASE
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS	ALTER DATABASE
... 중략 ...	
UPDATE	UPDATE
UPDATE .. RETURNING query	UPDATE
UPDATE .. RETURNING .. INTO	UPDATE
UPDATE .. WHERE CURRENT OF cursor	UPDATE
CREATE CLUSTER LOCATION	CREATE CLUSTER LOCATION
ALTER CLUSTER LOCATION	ALTER CLUSTER LOCATION
DROP CLUSTER LOCATION	DROP CLUSTER LOCATION

```

170 rows selected.

```

다음은 system action을 포함하는 audit policy를 생성하고 audit option을 조회하는 예이다.

```
CREATE AUDIT POLICY p1
  ACTIONS SELECT
    , DROP TABLE
    , DROP USER
;
Audit Policy created.
SELECT audit_option
  , audit_option_type
  , object_schema
  , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
 ORDER BY audit_option
;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
-----
DROP TABLE   SYSTEM ACTION      null      null
DROP USER     SYSTEM ACTION      null      null
SELECT         SYSTEM ACTION      null      null
3 rows selected.
```

Useful Audit Policy

다음은 유용한 audit policy를 정의하는 예이다.

- Logon failure 감사

```
CREATE AUDIT POLICY AUDIT_LOGON_FAILURES
  ACTIONS LOGON
;
AUDIT POLICY AUDIT_LOGON_FAILURES
  WHENEVER NOT SUCCESSFUL
;
```

- Data Definition Language (DDL) 수행을 감사

```
CREATE AUDIT POLICY AUDIT_DDL
  ACTIONS DDL
;
AUDIT POLICY AUDIT_DDL
```

```

        WHENEVER SUCCESSFUL
    ;

```

- 파라미터 변경을 감사

```

CREATE AUDIT POLICY AUDIT_DATABASE_PARAMETER
    ACTIONS ALTER DATABASE
        , ALTER SYSTEM
;
AUDIT POLICY AUDIT_DATABASE_PARAMETER
    WHENEVER SUCCESSFUL
;

```

- 계정 변경을 감사

```

CREATE AUDIT POLICY AUDIT_ACCOUNT_MGMT
    ACTIONS CREATE USER
        , DROP USER
        , ALTER USER
        , CHANGE PASSWORD
        , GRANT
        , REVOKE
;
AUDIT POLICY AUDIT_ACCOUNT_MGMT
;

```

- Center for Internet Security (CIS) 권고 사항을 감사

```

CREATE AUDIT POLICY AUDIT_CIS_RECOMMENDATIONS
    PRIVILEGES ALTER SYSTEM
        , ALTER DATABASE
    ACTIONS CREATE USER
        , DROP USER
        , ALTER USER
        , CHANGE PASSWORD
        , GRANT
        , REVOKE
        , CREATE PROFILE
        , ALTER PROFILE
        , DROP PROFILE
        , CREATE SYNONYM
        , DROP SYNONYM

```



```

        , CREATE PROCEDURE
        , DROP PROCEDURE
        , ALTER PROCEDURE
;
AUDIT POLICY AUDIT_CIS_RECOMMENDATIONS
    WHENEVER SUCCESSFUL
;

```

Audit Policy 운영

Audit Policy 활성화

Audit policy 객체는 활성화되기 전까지는 감사를 수행하지 않는다.

감사를 수행하려면 다음과 같이 AUDIT POLICY 구문을 이용해 audit policy 객체를 활성화해야 한다.

```

CREATE AUDIT POLICY audit_t1_dml
    ACTIONS INSERT ON u1.t1
        , DELETE ON u1.t1
        , UPDATE ON u1.t1
;
Audit policy created.
AUDIT POLICY audit_t1_dml;
Audit succeeded.

```

Audit policy를 활성화하면 새로운 session에 대해서만 감사를 수행하며 기존 session에는 영향을 미치지 않는다.

AUDIT POLICY 구문을 사용하여 audit policy를 활성화할 때, BY 절 또는 EXCEPT 절을 이용하여 감사를 수행할 사용자를 명시하거나 WHENEVER 절을 이용해 audit action의 성공/ 실패를 감사할 수 있다.

- BY | EXCEPT
 - BY user_list: 감사를 수행할 사용자를 명시한다.
 - EXCEPT user_list: 지정한 사용자들을 제외한 모든 사용자를 감사한다.
 - 생략할 경우, 모든 사용자를 감사한다.
- WHENEVER
 - WHENEVER SUCCESSFUL: Audit action이 성공했을 때 audit record를 생성한다.
 - WHENEVER NOT SUCCESSFUL: Audit action이 실패했을 때 audit record를 생성한다.
 - 생략할 경우, audit action의 성공/ 실패와 무관하게 audit record를 생성한다.

Audit policy의 활성화 정보는 AUDIT_POLICY_ENABLED view를 조회하여 확인할 수 있다.

```

SELECT policy_name
      , enabled_opt
      , user_name
      , when_success
      , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;

```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
AUDIT_T1_DML	BY	ALL USERS	YES	YES

위의 예와 같이 감사 대상 사용자를 생략한 경우, 모든 사용자를 의미하는 ALL USERS를 출력한다.

BY와 EXCEPT 절을 사용할 때는 다음 사항에 유의해야 한다.

- 동일한 audit policy에 대해 BY 절과 EXCEPT 절을 함께 사용할 수 없다.

```

AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u2;
ERR-42000(16475): audit policy already applied with the BY clause

```

- 동일한 audit policy에 다수의 AUDIT POLICY BY 절을 사용할 경우, user들의 집합에 대해 활성화된다. 즉, 다음 두 예제는 동일한 의미이다.
 - 예제 1

```

AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml BY u2;
Audit succeeded.
SELECT policy_name
      , enabled_opt
      , user_name
      , when_success
      , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;

```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
AUDIT_T1_DML	BY	U1	YES	YES

```
AUDIT_T1_DML BY          U2          YES          YES
2 rows selected.
```

- 예제 2

```
AUDIT POLICY audit_t1_dml BY u1, u2;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT  USER_NAME  WHEN_SUCCESS  WHEN_FAILURE
-----
AUDIT_T1_DML BY          U1          YES          YES
AUDIT_T1_DML BY          U2          YES          YES
2 rows selected.
```

- 동일한 audit policy에 다수의 AUDIT POLICY EXCEPT 절을 사용할 경우, 마지막 AUDIT POLICY 구문만 유효하다. 즉, 다음 두 예제는 다른 의미이다.

- 예제 1

```
AUDIT POLICY audit_t1_dml EXCEPT u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u2;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT  USER_NAME  WHEN_SUCCESS  WHEN_FAILURE
-----
AUDIT_T1_DML EXCEPT    U2          YES          YES
1 row selected.
```

- 예제 2

```
AUDIT POLICY audit_t1_dml EXCEPT u1, u2;
```

Audit succeeded.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
AUDIT_T1_DML	EXCEPT	U1	YES	YES
AUDIT_T1_DML	EXCEPT	U2	YES	YES

2 rows selected.

- BY 절과 함께 사용하는 WHENEVER 절은 누적된다. 즉, 다음 두 예제는 동일한 의미이다.

- 예제 1

```
AUDIT POLICY audit_t1_dml BY u1 WHENEVER SUCCESSFUL;
```

Audit succeeded.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
AUDIT_T1_DML	BY	U1	YES	NO

1 row selected.

```
AUDIT POLICY audit_t1_dml BY u1 WHENEVER NOT SUCCESSFUL;
```

Audit succeeded.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
```

```

        , when_failure
    FROM audit_policy_enabled
    WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          U1        YES          YES
1 row selected.

```

◦ 예제 2

```

AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
    FROM audit_policy_enabled
    WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          U1        YES          YES
1 row selected.

```

- EXCEPT 절과 함께 사용하는 WHENEVER 절은 마지막만 유효하다. 즉, 다음 두 예제는 다른 의미이다.

◦ 예제 1

```

AUDIT POLICY audit_t1_dml EXCEPT u1 WHENEVER SUCCESSFUL;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u1 WHENEVER NOT SUCCESSFUL;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
    FROM audit_policy_enabled
    WHERE policy_name = 'AUDIT_T1_DML'
;

```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
AUDIT_T1_DML	EXCEPT	U1	NO	YES

1 row selected.

◦ 예제 2

```
AUDIT POLICY audit_t1_dml EXCEPT u1;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT  USER_NAME  WHEN_SUCCESS  WHEN_FAILURE
-----
AUDIT_T1_DML EXCEPT    U1          YES            YES
1 row selected.
```

Audit Policy 비활성화

Audit policy를 비활성화하기 위해서는 NOAUDIT POLICY 구문을 수행해야 한다.

NOAUDIT POLICY 구문은 새로운 session에만 적용되며 기존 session에는 영향을 미치지 않는다.

다음 질의를 사용하여 활성화된 정보가 없도록 설정하면 audit policy가 완전히 비활성화된다.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM   audit_policy_enabled
WHERE  policy_name = 'AUDIT_T1_DML'
;
no rows selected.
```

NOAUDIT POLICY 구문은 AUDIT POLICY 지정 방식에 따라 생성된 개별 활성화 정보를 삭제한다.

다음은 audit_t1_dml의 u1 사용자에게 대한 감사만 비활성화하는 예이다.

```

SELECT policy_name
      , enabled_opt
      , user_name
      , when_success
      , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          U1          YES          YES
AUDIT_T1_DML BY          SYS         YES          YES
2 rows selected.
NOAUDIT POLICY audit_t1_dml BY u1;
Noaudit succeeded.
SELECT policy_name
      , enabled_opt
      , user_name
      , when_success
      , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          SYS         YES          YES
1 row selected.

```

AUDIT POLICY name BY 절을 사용한 경우 NOAUDIT POLICY name BY 구문으로 비활성화해야 하고, AUDIT POLICY name EXCEPT 절을 사용한 경우 BY 절 없이 NOAUDIT POLICY name 구문으로 비활성화해야 한다.

AUDIT POLICY 구문의 각 옵션을 비활성화하려면 그 사용 방법에 따라 다음과 같이 NOAUDIT POLICY 구문을 사용해야 한다.

표 13-10 Audit policy 활성화/ 비활성화

유형	AUDIT POLICY 구문	NOAUDIT POLICY 구문
전체 사용자	AUDIT POLICY p1	NOAUDIT POLICY p1
BY를 사용	AUDIT POLICY p1 BY u1	NOAUDIT POLICY p1 BY u1
EXCEPT를 사용	AUDIT POLICY p1 EXCEPT u1	NOAUDIT POLICY p1

다음과 같이 모든 user를 활성화한 경우, NOAUDIT POLICY BY 절은 아무런 영향을 미치지 않는다.

```

AUDIT POLICY audit_t1_dml;
Audit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          ALL USERS YES          YES
1 row selected.
NOAUDIT POLICY audit_t1_dml BY u1;
Noaudit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
AUDIT_T1_DML BY          ALL USERS YES          YES
1 row selected.

```

하나 이상의 user들을 개별적으로 활성화한 경우, AUDIT POLICY 설정 방법에 따라 NOAUDIT POLICY 구문을 사용해야 한다.

- BY 절을 이용해 활성화한 경우
 - 다음은 BY 절을 이용하여 활성화하는 예이다.

```

AUDIT POLICY p1 WHENEVER NOT SUCCESSFUL;
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;

```

- 활성화 정보를 조회한 결과는 다음과 같다.


```

SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
P1          BY          ALL USERS NO          YES
P1          BY          U1          YES         YES
P1          BY          U2          YES         YES
3 rows selected.

```

- 다음은 NOAUDIT 구문을 수행했을 때의 활성화 정보이다.

```

NOAUDIT POLICY p1;
Noaudit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
-----
P1          BY          U1          YES         YES
P1          BY          U2          YES         YES
2 rows selected.

```

위의 예에서 ALL USERS에 대한 감사는 비활성화되었지만 사용자 u1, u2에 대한 감사는 여전히 활성화되어 있다.

다음과 같이 BY 옵션을 사용하여 NOAUDIT POLICY 구문을 다시 사용하면 audit policy p1은 완전히 비활성화된다.

```

NOAUDIT POLICY p1 BY u1, u2;
Noaudit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure

```

```
FROM audit_policy_enabled
WHERE policy_name = 'P1';
no rows selected.
```

- EXCEPT를 이용해 활성화한 경우
 - 다음은 audit policy를 사용하여 활성화하는 예이다.

```
AUDIT POLICY p1 EXCEPT u1, sys;
```

- 활성화 정보를 조회한 결과는 다음과 같다.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
P1	EXCEPT	U1	YES	YES
P1	EXCEPT	SYS	YES	YES

2 rows selected.

- AUDIT POLICY 구문과 달리 NOAUDIT POLICY 구문에는 EXCEPT option이 없고 다음과 같이 옵션 없이 구문을 수행한다.

```
NOAUDIT POLICY p1;
Noaudit succeeded.
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
no rows selected.
```

즉, EXCEPT 옵션을 사용하여 audit policy를 활성화한 경우 NOAUDIT POLICY 구문을 사용하여 개별 사용자를 다시 활성화할 수 없다.

13.4 Authorization

Authorization 관련 구문

자세한 내용은 다음 링크를 참조한다.

- User 생성: **CREATE USER**
- User 제거: **DROP USER**
- User 변경: **ALTER USER**
- 권한 부여: **GRANT privileges TO**
- 권한 제거: **REVOKE privileges FROM**

사용자 객체와 권한에 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-11 Authorization 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_COL_PRIVS	사용자가 접근 가능한 column에 대한 권한
	ALL_COL_PRIVS_MADE	사용자가 grantor인 column 권한
	ALL_COL_PRIVS_RECD	사용자가 grantee인 column 권한
	ALL_DB_PRIVS	사용자와 관련된 DB 권한
	ALL_DB_PRIVS_MADE	사용자가 grantor인 DB 권한
	ALL_DB_PRIVS_RECD	사용자가 grantee인 DB 권한
	ALL_PROC_PRIVS	사용자와 관련된 stored procedure/ function 권한
	ALL_PROC_PRIVS_MADE	사용자가 grantor인 stored procedure/ function 권한
	ALL_PROC_PRIVS_RECD	사용자가 grantee인 stored procedure/ function 권한
	ALL_SCHEMA_PRIVS	사용자가 접근 가능한 스키마에 대한 권한
	ALL_SCHEMA_PRIVS_MADE	사용자가 grantor인 스키마 권한
	ALL_SCHEMA_PRIVS_RECD	사용자가 grantee인 스키마 권한
	ALL_SEQ_PRIVS	사용자가 접근 가능한 시퀀스에 대한 권한
	ALL_SEQ_PRIVS_MADE	사용자가 grantor인 시퀀스 권한
	ALL_SEQ_PRIVS_RECD	사용자가 grantee인 시퀀스 권한
	ALL_TAB_PRIVS	사용자가 접근 가능한 테이블에 대한 권한
	ALL_TAB_PRIVS_MADE	사용자가 grantor인 테이블 권한
	ALL_TAB_PRIVS_RECD	사용자가 grantee인 테이블 권한
	ALL_TBS_PRIVS	사용자가 접근 가능한 테이블스페이스에 대한 권한
	ALL_TBS_PRIVS_MADE	사용자가 grantor인 테이블스페이스 권한
	ALL_TBS_PRIVS_RECD	사용자가 grantee인 테이블스페이스 권한

스키마	View	설명
	ALL_USERS	사용자가 접근 가능한 사용자 정보
	USER_COL_PRIVS	사용자가 소유한 column에 대한 권한 정보
	USER_COL_PRIVS_MADE	사용자가 소유한 column에 대한 권한 부여 정보
	USER_COL_PRIVS_RECD	사용자가 소유한 column에 대한 권한 획득 정보
	USER_PROC_PRIVS	사용자가 소유한 stored procedure/ function에 대한 권한 정보
	USER_PROC_PRIVS_MADE	사용자가 소유한 stored procedure/ function에 대한 권한 부여 정보
	USER_PROC_PRIVS_RECD	사용자가 소유한 stored procedure/ function에 대한 권한 획득 정보
	USER_SCHEMA_PRIVS	사용자가 소유한 스키마에 대한 권한 정보
	USER_SCHEMA_PRIVS_MADE	사용자가 소유한 스키마에 대한 권한 부여 정보
	USER_SCHEMA_PRIVS_RECD	사용자가 소유한 스키마에 대한 권한 획득 정보
	USER_SEQ_PRIVS	사용자가 소유한 시퀀스에 대한 권한 정보
	USER_SEQ_PRIVS_MADE	사용자가 소유한 시퀀스에 대한 권한 부여 정보
	USER_SEQ_PRIVS_RECD	사용자가 소유한 시퀀스에 대한 권한 획득 정보
	USER_TAB_PRIVS	사용자가 소유한 테이블에 대한 권한 정보
	USER_TAB_PRIVS_MADE	사용자가 소유한 테이블에 대한 권한 부여 정보
	USER_TAB_PRIVS_RECD	사용자가 소유한 테이블에 대한 권한 획득 정보
	USER_USERS	현재 사용자 정보
INFORMATION_SCHEMA	COLUMN_PRIVILEGES	사용자가 접근 가능한 column 권한 정보
	ROUTINE_PRIVILEGES	사용자가 접근 가능한 stored procedure/ function 권한 정보
	TABLE_PRIVILEGES	사용자가 접근 가능한 테이블 권한 정보
	USAGE_PRIVILEGES	사용자가 접근 가능한 시퀀스 권한 정보

User 개념

User 객체는 사용자가 수행할 수 있는 권한의 집합으로 구성된다. 특정 사용자가 SQL 문장을 수행하려면 관련 객체에 대해 해당 SQL 구문을 수행할 수 있는 권한이 있어야 한다.

예를 들어, **CREATE USER** 문장으로 생성한 user에는 어떠한 권한도 없고 database에 접속할 수 없으며 어떠한 SQL 문장도 수행할 수 없다. 해당 사용자가 접속하기 위해서는 database 객체에 session을 생성할 수 있는 **CREATE SESSION ON DATABASE** 권한이 있어야 한다. 즉, 다음 예와 같이 **CREATE USER** 구문을 실행하여 적절한 권한을 부여해야 한다.

```
CREATE USER u1 IDENTIFIED BY u1_password;
GRANT CREATE SESSION ON DATABASE TO u1;
COMMIT;
```

User 객체를 생성한 후에 부여할 권한에 대한 자세한 내용은 CREATE USER 구문의 **사용 예**와 GRANT privileges TO 구문을 참조한다.

객체의 생성과 권한

SQL Schema Object 생성

테이블과 같은 SQL schema object를 생성할 경우, 특정 사용자가 CREATE TABLE 구문을 수행하려면 테이블이 속할 상위 non-schema object에 대한 적절한 권한이 필요하다.

다음은 CREATE TABLE 구문의 예이다.

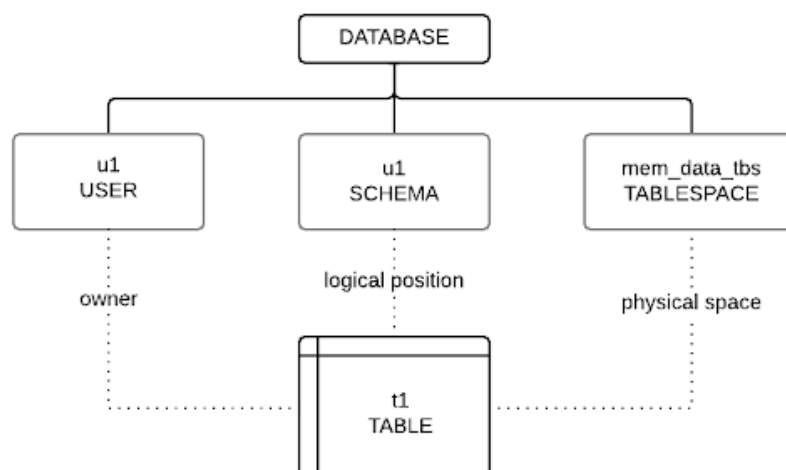
```
CREATE TABLE t1 ( id BIGINT, name VARCHAR(128) );
```

다음과 같은 의미로 해석될 경우

```
CREATE TABLE u1.t1 ( id BIGINT, name VARCHAR(128) ) TABLESPACE mem_data_tbs;
```

아래 그림에서와 같이 t1 테이블의 소유자는 u1 스키마의 소유자인 u1 계정이 되고 테이블이 포함될 논리적 위치는 u1 스키마가 되며 테이블이 저장될 물리적 공간은 mem_data_tbs 테이블스페이스가 된다.

그림 2 CREATE TABLE과 non-schema objects



이 때, 구문을 수행한 u1 사용자는 t1 테이블이 속할 스키마와 테이블스페이스에 대해 다음과 같은 권한이 필요하다. 논리적 위치인 u1 스키마에 포함될 테이블을 생성하려면 다음 권한 중 하나가 필요하다.

- u1 스키마에 테이블을 생성할 수 있는 권한
 - CREATE TABLE ON SCHEMA ON u1
- Database에 어떤 테이블이든 생성할 수 있는 권한
 - CREATE ANY TABLE ON DATABASE
 - 스키마 권한이 없더라도 u1 스키마의 상위 객체인 database에 대한 권한이 있다면 테이블을 생성할 수 있다.

물리적 저장공간인 mem_data_tbs 테이블스페이스에 테이블을 생성하려면 다음 권한 중 하나가 필요하다.

- mem_data_tbs 테이블스페이스에 객체를 생성할 수 있는 권한
 - CREATE OBJECT ON TABLESPACE mem_data_tbs
- Database에서 모든 테이블스페이스를 사용할 수 있는 권한
 - USAGE TABLESPACE ON DATABASE
 - 테이블스페이스 권한이 없더라도 mem_data_tbs 테이블스페이스의 상위 객체인 database에 대한 권한이 있다면 테이블을 생성할 수 있다.

테이블 객체의 소유자는 다음과 같이 결정된다.

- 테이블이 속한 스키마의 소유자
- 테이블이 속한 스키마가 PUBLIC인 경우, 구문을 수행한 사용자

테이블 객체의 소유자는 다음과 같이 테이블 구조를 변경하거나 테이블 데이터를 조작할 수 있는 권한을 갖는다.

- SELECT ON TABLE
 - 해당 테이블에 SELECT 구문을 수행할 수 있는 권한
- INSERT ON TABLE
 - 해당 테이블에 INSERT 구문을 수행할 수 있는 권한
- UPDATE ON TABLE
 - 해당 테이블에 UPDATE 구문을 수행할 수 있는 권한
- DELETE ON TABLE
 - 해당 테이블에 DELETE 구문을 수행할 수 있는 권한
- LOCK ON TABLE
 - 해당 테이블에 LOCK 구문을 수행할 수 있는 권한
- INDEX ON TABLE
 - 해당 테이블에 CREATE/ DROP/ ALTER INDEX 구문을 수행할 수 있는 권한
- ALTER ON TABLE
 - 해당 테이블에 ALTER TABLE 구문을 수행할 수 있는 권한

이 때, 부여받는 권한의 grantor (권한을 부여하는 자)는 내부적으로 사용되는 _SYSTEM 계정이고 grantee (권한을 부여받는 자)는 테이블의 소유자가 된다. 따라서 SYS 계정을 포함한 어떠한 사용자도 **REVOKE privileges FROM** 문

장을 이용해 소유자의 권한을 제거하거나 변경할 수 없다. 테이블 소유자가 부여받은 권한은 테이블이 제거될 때 함께 제거된다.

Non-schema Object 생성

테이블과 같은 SQL schema object를 생성할 때와 마찬가지로, user, schema, tablespace와 같은 non-schema object를 생성할 때도 상위 객체인 database에 대한 적절한 권한이 필요하다.

다음 구문을 수행하는 사용자는 각 구문별 상위 객체인 database 객체에 대해 다음과 같은 권한을 가져야 한다.

- CREATE USER 구문
 - CREATE USER ON DATABASE
 - Database에 CREATE USER 구문을 수행할 수 있는 권한
- CREATE TABLESPACE 구문
 - CREATE TABLESPACE ON DATABASE
 - Database에 CREATE TABLESPACE 구문을 수행할 수 있는 권한
- CREATE SCHEMA 구문
 - CREATE SCHEMA ON DATABASE
 - Database에 CREATE SCHEMA 구문을 수행할 수 있는 권한
- CREATE PUBLIC SYNONYM 구문
 - Database에 CREATE PUBLIC SYNONYM 구문을 수행할 수 있는 권한

SQL schema object를 생성할 때와는 달리 non-schema object의 소유자는 CREATE 구문을 수행한 사용자가 아니다. 각 객체별로 다음과 같은 특징이 있다.

- User 객체
 - 소유자가 존재하지 않는다.
 - User 객체를 제거하기 위해 DROP USER ON DATABASE 권한이 필요하다.
- Tablespace 객체
 - 소유자가 존재하지 않는다.
 - Tablespace 객체를 제거하기 위해 DROP TABLESPACE ON DATABASE 권한이 필요하다.
- Schema 객체
 - CREATE SCHEMA 구문에 지정된 소유자이다.
 - Schema 객체의 소유자가 제거할 수 있다.
- Public synonym 객체
 - 소유자가 존재하지 않는다.
 - Public synonym 객체를 제거하기 위해서는 DROP PUBLIC SYNONYM ON DATABASE 권한이 필요하다.

권한 (Privilege)

권한 부여

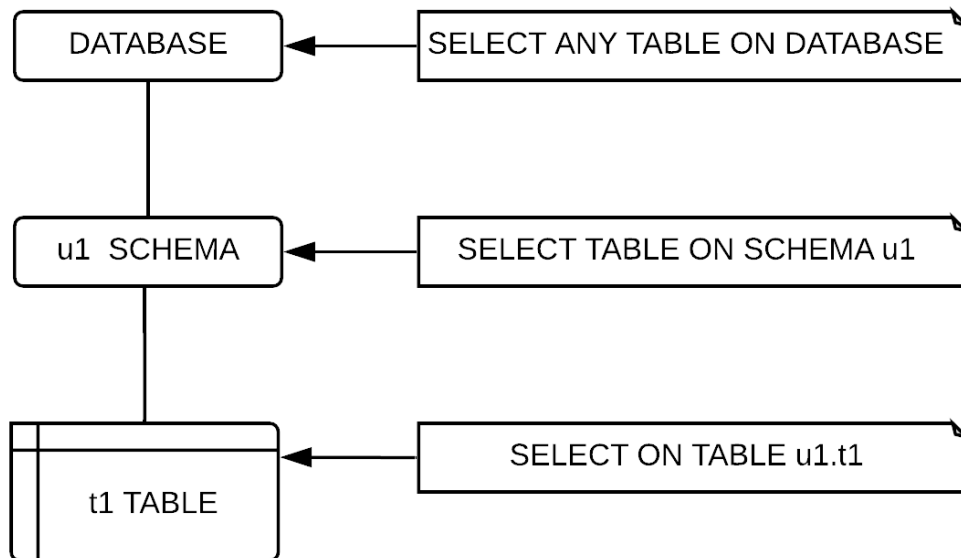
사용자가 테이블과 같은 SQL schema object의 소유자가 아닌 경우에 데이터를 추가 (INSERT), 삭제 (DELETE), 갱신 (UPDATE) 하거나 검색 (SELECT)하기 위해서는 **GRANT privileges TO** 구문을 통해 해당 테이블에 대한 적절한 권한을 부여받아야 한다.

예를 들어, 테이블의 소유자가 아닌 사용자가 다음과 같은 SELECT 구문을 수행하기 위해서는 다음 권한 중 하나가 필요하다. 아래 그림과 같이 테이블 t1에 대한 SELECT 권한이 없더라도, 상위 객체인 u1 스키마에 대한 SELECT 권한 또는 database에 대한 SELECT 권한이 있다면 u1.t1 테이블을 조회할 수 있다.

- Test 사용자가 수행한다.

```
SELECT id, name FROM u1.t1 WHERE id < 100;
```

그림 3 SELECT 구문을 수행하기 위한 privilege



- SELECT ON TABLE u1.t1
 - u1.t1 테이블에 대한 SELECT 권한
- SELECT ON SCHEMA u1
 - 상위 객체인 u1 스키마에 대한 SELECT 권한
 - u1 스키마에 속한 모든 테이블을 SELECT 할 수 있는 권한이다.
- SELECT ANY TABLE ON DATABASE
 - 차상위 객체인 database에 대한 SELECT 권한
 - Database의 모든 테이블을 SELECT 할 수 있는 권한이다.

다른 사용자에게 SELECT ON TABLE u1.t1 권한을 부여하려면 GRANT 구문을 수행하는 사용자가 다음 중 하나여야

한다.

- u1.t1 객체의 소유자
 - 객체의 소유자는 _SYSTEM 계정이 부여한 SELECT ON TABLE u1.t1 WITH GRANT OPTION 권한을 가진다.
- SELECT ON TABLE u1.t1 WITH GRANT OPTION을 가진 사용자
 - WITH GRANT OPTION은 부여받은 권한을 다른 사용자에게 다시 부여할 수 있는 옵션이다.
- ACCESS CONTROL ON DATABASE 권한을 가진 사용자
 - 해당 사용자는 모든 객체에 대한 모든 권한을 제어할 수 있다.

다음은 객체의 소유자가 다른 사용자에게 권한을 부여하는 예이다.

- u1.t1 테이블을 소유한 사용자가 GRANT 구문을 수행한다.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

권한의 종류는 **GRANT privileges TO** 구문을 참조한다. 각 SQL 구문을 수행하기 위한 권한은 **SQL References** 파트의 각 구문별 사용 범위 및 접근 권한 항목을 참조한다.

권한 철회

특정 사용자에게 부여된 권한은 **REVOKE privileges FROM** 구문을 사용하여 철회할 수 있다. 단, 객체의 소유자가 가진 권한은 객체가 제거되기 전까지 철회할 수 없다.

예를 들어, REVOKE 구문을 수행하여 test 사용자로부터 SELECT 권한을 철회할 수 있는 사용자는 다음과 같다. 사용자가 테이블의 소유자라 하더라도 자신이 부여하지 않은 권한은 철회할 수 없다.

```
REVOKE SELECT ON TABLE u1.t1 FROM test;
```

- 권한을 부여한 사용자
 - test 사용자에게 SELECT ON TABLE u1.t1 권한을 부여한 사용자
- ACCESS CONTROL ON DATABASE 권한을 가진 사용자
 - 해당 사용자는 모든 객체에 대한 모든 권한을 제어할 수 있다.

다음과 같이 여러 사용자로부터 동일한 권한을 부여받은 경우, test 사용자는 부여받은 모든 권한이 철회되기 전까지 SELECT 구문을 수행할 수 있다.

- 객체의 소유자 u1이 test 사용자에게 권한 부여

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- 객체의 소유자 u1이 사용자 u2에게 WITH GRANT OPTION 권한 부여

```
GRANT SELECT ON TABLE u1.t1 TO u2 WITH GRANT OPTION;
```

- 사용자 u2가 수행
- WITH GRANT OPTION 권한을 가진 u2가 test 사용자에게 권한 부여

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

위의 예에서, test 사용자는 grantor가 u1, u2 사용자인 두 개의 SELECT ON TABLE u1.t1 권한을 가지게 된다. 즉, 권한 정보는 {grantor, grantee, 객체 권한}으로 구성된다.

PUBLIC 계정

PUBLIC 계정은 모든 사용자를 의미하는 특수한 계정이다.

예를 들어, 다음과 같이 PUBLIC 계정에 SELECT 권한을 부여할 경우 모든 사용자가 u1.t1 테이블에 SELECT 구문을 수행할 수 있다.

```
GRANT SELECT ON TABLE u1.t1 TO PUBLIC;
```

즉, 별도로 SELECT 권한을 부여받지 않은 사용자라 하더라도 PUBLIC 계정에 부여된 SELECT 권한을 이용해 u1.t1 테이블에 대해 SELECT 구문을 수행할 수 있다.

PUBLIC 계정에 권한을 부여하는 것은 이미 존재하는 모든 사용자에게 해당 권한을 부여하는 것이 아니라 PUBLIC 계정 자체에 권한을 부여하는 것이다. 따라서 새로운 사용자를 생성하더라도 새로 생성된 사용자가 SELECT 구문을 수행할 수 있다.

마찬가지로 다음과 같이 PUBLIC 계정에 대한 권한을 철회할 때도 모든 사용자로부터 해당 권한을 철회하는 것이 아니며, 사용자에게 u1.t1 테이블에 대한 SELECT 권한이 있다면 SELECT 구문을 수행할 수 있다.

- test 사용자에게 권한 부여

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- PUBLIC 계정에 권한 부여

```
GRANT SELECT ON TABLE u1.t1 TO PUBLIC;
```

- PUBLIC 계정의 권한이 철회되었지만 test 사용자는 여전히 SELECT ON TABLE u1.t1에 대한 권한을 가지고 있다.

```
REVOKE SELECT ON TABLE u1.t1 FROM PUBLIC;
```

Column 권한

테이블의 특정 column에만 권한을 부여하여 다른 사용자의 DML이나 SELECT 구문 수행을 제어할 수 있다.

다음 예제 테이블을 참조한다.

```
CREATE TABLE u1.t1
(
  id      BIGINT,
  name    VARCHAR(128),
  addr    VARCHAR(1024),
  salary  NUMBER(20,0)
);
```

다른 사용자에게 u1.t1 테이블의 salary column 정보를 제외한 SELECT 권한을 부여할 경우 다음과 같이 권한을 부여할 column을 나열하여 **GRANT privileges TO** 구문을 수행한다. 권한을 부여받은 test 사용자는 salary column을 조회할 수 없다.

- test 사용자에게 SELECT column 권한을 부여한다.

```
GRANT SELECT( id, name, addr ) ON TABLE u1.t1 TO test;
```

다음과 같이 테이블 권한을 부여할 경우, 테이블에 속한 모든 column에 대한 권한을 자동으로 부여한다. Column에 대한 권한과 테이블에 대한 권한을 함께 부여할 경우 column에 대한 권한 정보는 동일한 정보로써 중복 관리되지 않는다.

- test 사용자에게 SELECT 권한을 부여한다.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- 다음과 같이 테이블의 모든 column에 대한 권한들을 자동으로 부여한다.

```
GRANT SELECT(id) ON TABLE u1.t1 TO test;
GRANT SELECT(name) ON TABLE u1.t1 TO test;
GRANT SELECT(addr) ON TABLE u1.t1 TO test;
GRANT SELECT(salary) ON TABLE u1.t1 TO test;
```

다음과 같이 **REVOKE privileges FROM** 구문을 수행하여 column에 대한 권한을 철회한다.

- test 사용자로부터 SELECT column 권한을 철회한다.

```
REVOKE SELECT( id, name, addr ) ON TABLE u1.t1 FROM test;
```

다음과 같이 테이블 권한을 철회하면 테이블에 속한 모든 column에 대한 권한도 자동으로 철회된다. 즉, column 권

한과 테이블 권한을 별도로 부여했다 하더라도 테이블에 대한 권한을 철회하면 column들에 대한 권한도 모두 철회된다.

- test 사용자로부터 SELECT 권한을 철회한다.
- 테이블의 모든 column에 대한 권한을 철회한다.

```
REVOKE SELECT ON TABLE u1.t1 FROM test;
```

다음과 같이 column 권한만 철회할 경우 테이블 권한이 존재한다면 그 테이블 권한을 사용하여 여전히 해당 구문을 수행할 수 있다. 따라서 특정 column에만 권한을 부여하려면 테이블 권한을 철회한 후 각 column에 권한을 부여해야 한다.

- test 사용자에게 SELECT 테이블 권한을 부여한다.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- test 사용자로부터 SELECT(salary) column 권한을 철회한다.
- test 사용자에게 SELECT 테이블 권한이 존재하여 u1.t1 테이블의 salary column을 조회할 수 있다.

```
REVOKE SELECT(salary) ON TABLE u1.t1 FROM test;
```

테이블 권한과 column 권한에 대한 자세한 내용은 **GRANT privileges TO** 구문을 참조한다.

13.5 Schema

Schema 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Schema 생성: **CREATE SCHEMA**
- Schema 제거: **DROP SCHEMA**

스키마 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-12 스키마 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_SCHEMAS	사용자가 접근 가능한 스키마 정보
	ALL_SCHEMA_PATH	사용자가 접근 가능한 스키마 경로
	USER_SCHEMAS	사용자가 소유한 스키마 정보
	USER_SCHEMA_PATH	사용자의 스키마 경로 정보
INFORMATION_SCHEMA	SCHEMATA	사용자가 접근 가능한 스키마 정보

Schema 개념

데이터베이스는 하나 이상의 스키마로 구성된다. 스키마는 테이블, 인덱스, view, 시퀀스와 같은 데이터를 다루는 객체들로 구성된다. 이와 같이 스키마에 속한 객체들을 SQL schema object라고 한다.

스키마는 OS의 디렉토리와 유사한 개념으로, 스키마와 테이블의 관계는 OS의 디렉토리와 파일의 관계와 비슷하다. 즉, 스키마는 SQL schema object들의 논리적 위치이며 이름을 구분하는 기준이 된다. 각 SQL schema object들은 스키마 내에서 중복되지 않는 이름을 가져야 하며, 스키마 내의 naming space는 다음과 같다.

- Table, view, sequence, private synonym, stored procedure, stored function
- Index
- Constraint

서로 다른 스키마 내에서는 동일한 이름의 테이블을 정의할 수 있고 이름이 동일한 서로 다른 테이블에 접근할 경우 스키마 이름을 함께 명시하여 구문을 수행할 수 있다.

```
gSQL> SELECT u1.t1.id, u1.t1.name, u2.t1.addr
      FROM u1.t1, u2.t1
      WHERE u1.t1.id = u2.t1.id;
```

SQL schema object의 이름에는 다음과 같이 스키마 이름을 명시할 수도 있고 생략할 수도 있다. 생략할 경우 스키마의 이름은 구문을 수행하는 사용자의 schema path에 의해 결정된다. Schema path에 대한 자세한 내용은 **Schema Path** 절을 참조한다.

- Schema 이름을 명시하는 경우

```
gSQL> SELECT u1.t1.id, u1.t1.name FROM u1.t1;
```

- Schema 이름을 명시하지 않는 경우

```
gSQL> SELECT t1.id, t1.name FROM t1;
```

User와 Schema

GOLDILOCKS에서는 사용자와 스키마가 1 : N의 관계를 갖는다. 즉, 사용자가 스키마를 소유하지 않거나, 여러 개의 스키마를 가질 수 있다.

SQL 표준은 user, schema, database 등과 같은 non-schema 객체들의 관계에 대해 명확히 정의하지 않고 있으며, 각 DBMS 들은 다음과 같이 상이하게 non-schema 객체간의 관계를 정의하고 있다.

- Oracle
 - user : schema = 1 : 1 관계이다.
- DB2
 - user : schema = 1 : N 관계이다.
- Postgres
 - user : schema = 1 : N 관계이다.
- MySQL
 - database : schema = 1 : 1 관계이다.
 - user는 schema(database)의 하위 객체이다.

GOLDILOCKS에서는 데이터베이스를 구축할 때 고객 시스템의 특성에 맞춰 아래 그림과 같이 다양하게 사용자와 스키마의 관계를 구성할 수 있다.

그림 4 User와 schema의 관계

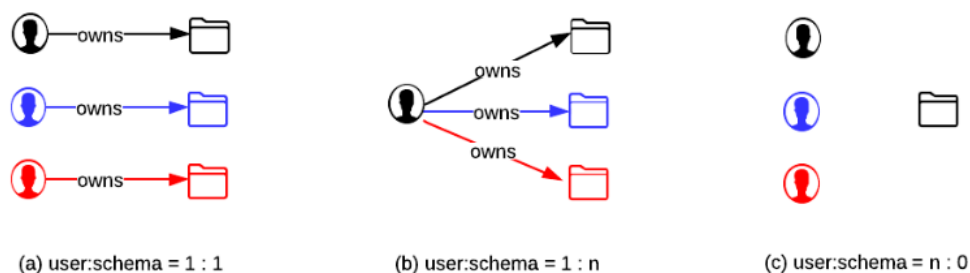


그림 (a)와 같이 사용자별로 스키마를 소유하도록 데이터베이스를 구성할 경우 다음과 같이 사용자와 스키마를 생성한다.

- u1 사용자를 생성할 때 u1 스키마를 함께 생성하여 소유하도록 한다.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITH SCHEMA;
```

- WITH SCHEMA 구문은 생략 가능하며 사용자와 동일한 이름의 스키마가 생성된다.

```
gSQL> CREATE USER u2 IDENTIFIED BY u2_password;
```

```
gSQL> CREATE USER u3 IDENTIFIED BY u3_password;
```

그림 (b)와 같이 하나의 사용자가 다수의 스키마를 소유하도록 데이터베이스를 구성할 경우 다음과 같이 사용자와 스키마를 생성한다.

- 스키마는 생성하지 않고 u1 사용자만 생성한다.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
```

- 다수의 스키마를 생성하고, 각 스키마의 소유자를 u1 사용자로 지정한다.

```
gSQL> CREATE SCHEMA s1 AUTHORIZATION u1;
```

```
gSQL> CREATE SCHEMA s2 AUTHORIZATION u1;
```

```
gSQL> CREATE SCHEMA s3 AUTHORIZATION u1;
```

그림 (c)와 같이 별도의 스키마를 생성하지 않고 다수의 사용자들이 하나의 스키마를 공유하도록 데이터베이스를 구성할 경우 다음과 같이 사용자를 생성한다. 다음 예의 경우, 모든 사용자들이 **PUBLIC 스키마**를 공유한다.

- 스키마를 생성하지 않고 사용자만 생성한다.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
```

```
gSQL> CREATE USER u2 IDENTIFIED BY u2_password WITHOUT SCHEMA;
```

```
gSQL> CREATE USER u3 IDENTIFIED BY u3_password WITHOUT SCHEMA;
```

사용자 생성과 스키마 생성에 대한 자세한 내용은 **CREATE USER** 구문과 **CREATE SCHEMA** 구문을 참조한다.

Schema Path

Schema path는 테이블과 같은 SQL schema 객체 이름을 스키마 이름없이 사용할 경우 해당 스키마 이름을 찾기 위한 경로이다. Schema path 개념은 Unix 시스템의 PATH 환경 변수와 유사하다. 즉, Unix 시스템에서 명령어를 수행할 때 PATH 경로에 지정된 순서대로 명령어를 검색하는 것과 유사한 개념이다.

사용자가 다수의 스키마를 소유하고 있고 다음과 같이 스키마 이름 없이 테이블을 생성하거나 조회할 때, 어떤 스키마에 테이블을 생성할지를 결정하는 기준이 schema path이다.

- s1 스키마에 t1 테이블을 생성한다.

```
gSQL> CREATE TABLE s1.t1 ( id INTEGER );
```

- s2 스키마에 t1 테이블을 생성한다.

```
gSQL> CREATE TABLE s2.t1 ( name VARCHAR(128) );
```

- 어떤 스키마에 t1 테이블이 생성되는가?

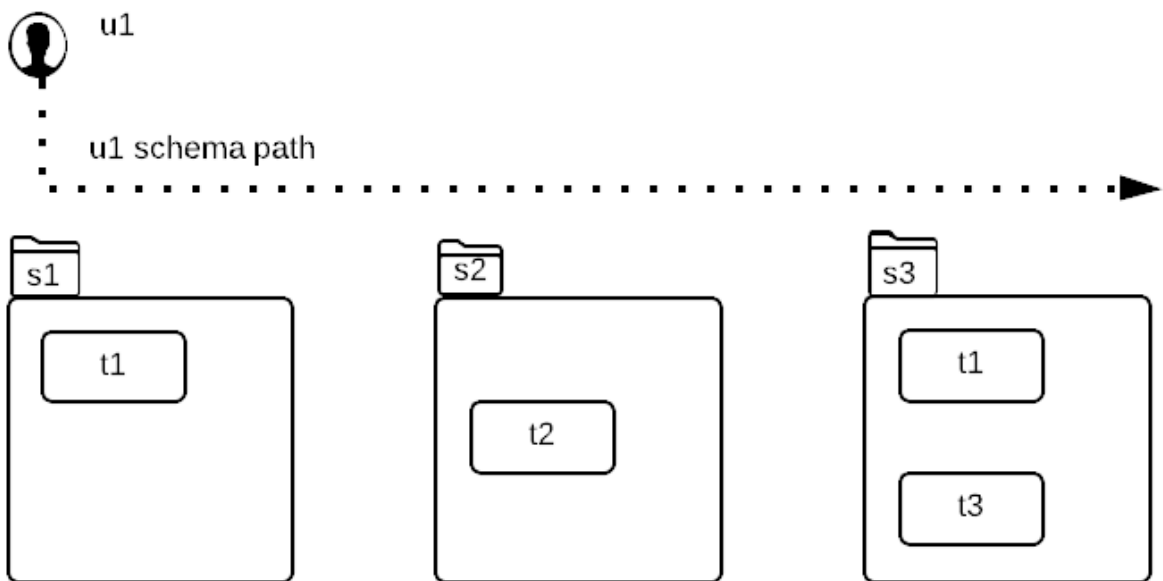
```
gSQL> CREATE TABLE t1 ( address VARCHAR(1024) );
```

- 어떤 스키마의 t1 테이블을 조회하는가?

```
gSQL> SELECT * FROM t1;
```

아래 그림은 사용자 u1의 schema path를 도식화한 예이다. 사용자 u1의 schema path는 {s1, s2, s3}의 순서로 지정되어 있으며, s1 스키마에는 t1 테이블, s2 스키마에는 t2 테이블, s3 스키마에는 t1과 t3 테이블이 존재한다.

그림 5 Schema path의 예



다음과 같이 SELECT 구문에 스키마 이름이 생략된 경우, schema path에 의해 스키마 이름이 해석된다.

- Schema path에 의해 s1.t1 테이블로 해석된다.


```
gSQL> SELECT * FROM t1;
gSQL> SELECT * FROM s1.t1;
```

- Schema path에 의해 s2.t2 테이블로 해석된다.

```
gSQL> SELECT * FROM t2;
gSQL> SELECT * FROM s2.t2;
```

- Schema path 에 의해 s3.t3 테이블로 해석된다.

```
gSQL> SELECT * FROM t3;
gSQL> SELECT * FROM s3.t3;
```

위의 예에서 s3.t1 테이블을 조회하기 위해 스키마 이름을 생략할 경우 schema path가 s1.t1 테이블을 결정하므로, 다음과 같이 스키마 이름을 명시하여야 한다.

- Schema path에 의해 s1.t1 테이블이 된다.

```
gSQL> SELECT * FROM t1;
```

- s3.t1 테이블을 조회하기 위해서는 s3 스키마를 명시해야 한다.

```
gSQL> SELECT * FROM s3.t1;
```

다음과 같이 스키마 이름이 생략된 CREATE TABLE 구문은 schema path의 첫 번째 스키마인 s1에 생성된다.

- 동일한 테이블 s1.t1이 존재하여 에러가 발생한다.

```
gSQL> CREATE TABLE t1 ( id INTEGER );
gSQL> CREATE TABLE s1.t1 ( id INTEGER );
```

- s2.t2 테이블이 존재하지만 서로 다른 스키마에 속한 s1.t2 테이블이 생성된다.

```
gSQL> CREATE TABLE t2 ( name VARCHAR(128) );
gSQL> CREATE TABLE s1.t2 ( name VARCHAR(128) );
```

- s3.t3 테이블이 존재하지만 서로 다른 스키마에 속한 s1.t3 테이블이 생성된다.

```
gSQL> CREATE TABLE t3 ( name VARCHAR(128) );
gSQL> CREATE TABLE s1.t3 ( name VARCHAR(128) );
```

현재 사용자가 객체를 생성할 때 스키마 이름을 생략할 경우에 사용되는 스키마 이름은 SQL 표준함수인 **CURRENT_SCHEMA**를 통해 조회할 수 있다.

```
gSQL> SELECT current_schema FROM dual;
CURRENT_SCHEMA
-----
S1
1 row selected.
```

현재 사용자의 schema path 정보는 DICTIONARY_SCHEMA 스키마의 ALL_SCHEMA_PATH view를 통해 다음과 같이 조회할 수 있다.

```
gSQL> SELECT * FROM all_schema_path;
AUTH_NAME  SCHEMA_NAME          SEARCH_ORDER
-----
U1          S1                    1
U1          S2                    2
U1          S3                    3
U1          PUBLIC                4
PUBLIC      DICTIONARY_SCHEMA     5
PUBLIC      INFORMATION_SCHEMA    6
PUBLIC      DEFINITION_SCHEMA     7
PUBLIC      PERFORMANCE_VIEW_SCHEMA 8
PUBLIC      FIXED_TABLE_SCHEMA    9
9 rows selected.
```

위의 예에서 u1 사용자의 schema path는 {s1, s2, s3, public}의 순서로 되어 있으며, PUBLIC 계정의 schema path는 {DICTIONARY_SCHEMA, INFORMATION_SCHEMA, DEFINITION_SCHEMA, PERFORMANCE_VIEW_SCHEMA, FIXED_TABLE_SCHEMA}의 순서로 되어 있다. 즉, 스키마 이름을 생략하고 임의의 t1 테이블을 조회할 경우 현재 사용자 u1의 schema path를 먼저 탐색하고, 존재하지 않을 경우 PUBLIC 계정의 schema path를 탐색한다.

특정 사용자의 schema path는 **ALTER USER** 구문을 사용하여 다음 예와 같이 변경할 수 있다. PUBLIC 계정의 schema path는 ALTER USER PUBLIC SCHEMA PATH 구문을 사용하여 변경한다. 사용자의 현재 schema path와 함께 추가적으로 schema도 변경할 경우 CURRENT PATH 절을 사용한다.

- u1 사용자의 schema path를 변경한다.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s3, s2, s1 );
User altered.
```

- PUBLIC 계정의 schema path를 변경한다.

```
gSQL> ALTER USER PUBLIC SCHEMA PATH ( s1, s2, s3 );
User altered.
```

- CURRENT PATH를 이용하여 현재 schema path를 포함한 schema path를 변경한다.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s4, CURRENT PATH );
User altered.
```

CREATE USER 구문을 사용하여 사용자를 생성할 때 schema path가 자동으로 결정되며, 이 후 **CREATE SCHEMA** 구문을 사용해 생성한 사용자의 스키마는 사용자의 schema path에 자동으로 포함되지 않으므로, 필요한 경우 **ALTER USER** 구문을 사용하여 schema path에 포함시켜야 한다. 자세한 내용은 각 구문을 참조한다.

PUBLIC 스키마

PUBLIC 스키마는 모든 사용자가 객체를 생성할 수 있는 공용 스키마이다. 다음 예와 같이 스키마를 소유하지 않은 사용자가 스키마 이름을 명시하지 않고 생성한 테이블의 스키마가 PUBLIC이 된다.

- 스키마를 소유하지 않는 사용자 u1을 생성한다.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
gSQL> GRANT CREATE SESSION TO u1;
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
```

- 사용자 u1이 테이블을 생성한다.

```
% gsql u1 u1_password
gSQL> CREATE TABLE t1 ( id INTEGER );
```

- 위 구문은 다음과 같은 의미이다.

```
gSQL> CREATE TABLE public.t1 ( id INTEGER );
```

위의 예에서 사용자 u1이 생성한 테이블 t1의 스키마는 PUBLIC이 된다. PUBLIC 스키마는 데이터베이스를 생성할 때 자동으로 생성되는 built-in 스키마이며, 모든 사용자가 스키마 내에 객체를 생성할 수 있도록 다음 구문과 동일한 의미의 권한이 부여된다.

```
gSQL> GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT
      ON SCHEMA PUBLIC
      TO PUBLIC;
```

즉, 공용 스키마를 의미하는 PUBLIC 스키마는 모든 사용자를 의미하는 PUBLIC 계정과는 다르다. 위의 구문에서 모든 사용자를 의미하는 PUBLIC 계정 (구문에서, TO PUBLIC)에는 PUBLIC 스키마 (구문에서, ON SCHEMA PUBLIC)에 객체를 생성할 수 있는 권한이 부여되어 있다. 단, 모든 사용자가 테이블을 생성하고 조작할 수 있지만 다른 사용자가 PUBLIC 스키마에 생성한 테이블을 조회하는 것과 같은 조작을 하기 위해서는 해당 테이블에 대한 적절한 권한을 부여받아야 한다.

다음은 GRANT 구문상에서 PUBLIC 계정과 PUBLIC 스키마를 사용하는 예이다.

- PUBLIC 계정에 u1.t1 테이블을 SELECT 할 수 있는 권한을 부여한다.
- 모든 사용자가 u1.t1 테이블을 조회할 수 있다.

```
gSQL> GRANT SELECT ON TABLE u1.t1 TO PUBLIC;
```

- PUBLIC 스키마 내에 속한 모든 테이블을 SELECT 할 수 있는 권한을 u1에 부여한다.
- u1 사용자는 PUBLIC 스키마 내의 모든 테이블을 조회할 수 있다.

```
gSQL> GRANT SELECT TABLE ON SCHEMA PUBLIC TO u1;
```

User와 Schema 활용 예

예를 들어, 한 명의 관리자와 다수의 개발자가 하나의 schema를 함께 이용하는 경우 다음과 같은 SQL 구문을 통해 제어할 수 있다.

다음은 스키마 our_schema를 관리하는 mgr_user 사용자와, our_schema를 이용해 application을 개발하는 다수의 사용자인 app_user1, app_user2, app_user3 를 생성하는 예이다.

다음과 같이 mgr_user 사용자와 our_schema 스키마를 생성한다.

- mgr_user와 our_schema를 생성한다.
- mgr_user가 our_schema의 소유자가 된다.

```
gSQL> CREATE USER mgr_user IDENTIFIED BY mgr_user WITH SCHEMA our_schema;
User created.
```

- mgr_user에게 적절한 권한을 부여한다.

```
gSQL> GRANT ALL PRIVILEGES ON DATABASE TO mgr_user;
Grant succeeded.
gSQL> COMMIT;
Commit complete.
```

다음과 같이 다수의 app_user 사용자를 생성한다.

app_user 사용자들은 our_schema에 대해 SELECT와 DML만 수행할 수 있도록 한다.

- 스키마를 소유하지 않는 다수의 app_user를 생성한다.

```
gSQL> CREATE USER app_user1 IDENTIFIED BY app_user1 WITHOUT SCHEMA;
User created.
```

```
gSQL> CREATE USER app_user2 IDENTIFIED BY app_user2 WITHOUT SCHEMA;
User created.
gSQL> CREATE USER app_user3 IDENTIFIED BY app_user3 WITHOUT SCHEMA;
User created.
gSQL> COMMIT;
Commit complete.
```

- 다수의 app_user에게 접속 권한을 부여한다.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO app_user1, app_user2, app_user3;
Grant succeeded.
gSQL> COMMIT;
Commit complete.
```

- 다수의 app_user에게 our_schema에 대한 읽기/ 쓰기 권한만 부여한다.

```
gSQL> GRANT SELECT TABLE, INSERT TABLE, UPDATE TABLE, DELETE TABLE ON SCHEMA our_schema TO app
_user1, app_user2, app_user3;
Grant succeeded.
gSQL> COMMIT;
Commit complete.
```

- 다수의 app_user의 SCHEMA_PATH를 our_schema로 지정한다.

```
gSQL> ALTER USER app_user1 SCHEMA PATH ( our_schema );
User altered.
gSQL> ALTER USER app_user2 SCHEMA PATH ( our_schema );
User altered.
gSQL> ALTER USER app_user3 SCHEMA PATH ( our_schema );
User altered.
gSQL> COMMIT;
Commit complete.
```

위와 같은 작업을 통해 mgr_user가 our_schema에 객체를 생성/ 제거/ 변경할 수 있는 DDL 권한들을 가지게 되는 반면에 다수의 app_user들은 our_schema에 포함된 테이블에 대해 읽기/ 쓰기 작업만 수행할 수 있다.

mgr_user는 다음 예와 같이 테이블을 생성하는 등의 관리 작업을 수행할 수 있다.

```
gSQL> \connect mgr_user mgr_user
gSQL> CREATE TABLE t1 ( c1 INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1 );
1 row created.
```

```
gSQL> INSERT INTO t1 VALUES (2), (3);
2 rows created.
gSQL> COMMIT;
Commit complete.
```

다수의 app_user는 다음과 같이 스키마 이름을 명시하지 않고 our_schema 내의 테이블들에 대해 읽기/쓰기는 할 수 있지만 객체를 제거하거나 생성할 수는 없다.

```
gSQL> \connect app_user1 app_user1
gSQL> SELECT * FROM t1;
C1
--
1
2
3
3 rows selected.
gSQL> INSERT INTO t1 VALUES (4);
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DROP TABLE t1;
ERR-42000(16208): insufficient privileges
```

13.6 Tablespace

Tablespace 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Tablespace 생성
 - CREATE TABLESPACE
 - CREATE MEMORY DATA TABLESPACE
 - CREATE MEMORY TEMPORARY TABLESPACE
- Tablespace 제거: DROP TABLESPACE
- Tablespace 변경
 - ALTER TABLESPACE
 - ALTER TABLESPACE name RENAME TO
 - ALTER TABLESPACE name BACKUP
 - ALTER TABLESPACE name [ONLINE|OFFLINE]
- Tablespace를 구성하는 파일의 추가, 제거, 변경
 - ALTER TABLESPACE name ADD [DATAFILE|MEMORY]
 - ALTER TABLESPACE name DROP [DATAFILE|MEMORY]
 - ALTER TABLESPACE name RENAME DATAFILE

테이블스페이스 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

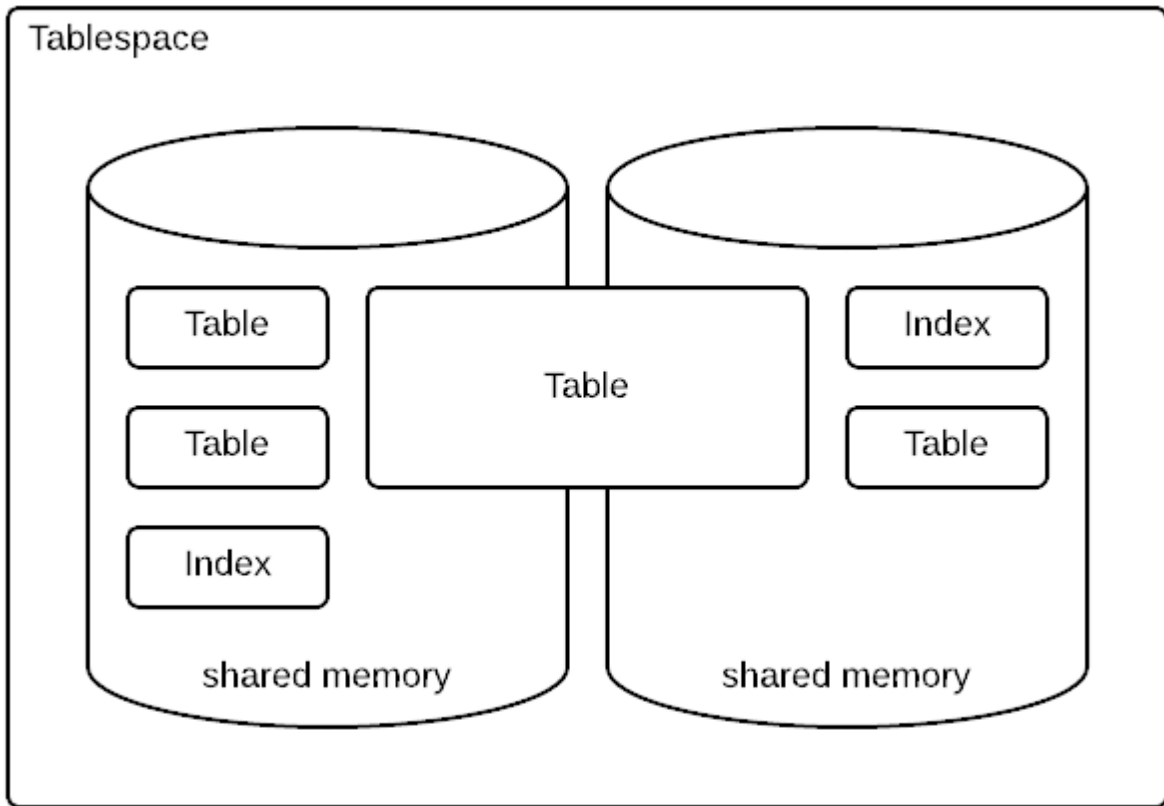
표 13-13 테이블스페이스 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	USER_TABLESPACES	사용자가 접근 가능한 테이블스페이스 정보

Tablespace 개념

테이블스페이스는 논리적 개념으로써 한 개 이상의 물리적 공유 메모리들로 구성되며, 테이블, 인덱스와 같은 데이터를 저장하기 위한 공간이다. 아래 그림에서와 같이 테이블스페이스에 저장되는 테이블, 인덱스와 같은 물리적 객체들은 여러 공유 메모리에 걸쳐 저장될 수 있으며, 공유 메모리를 추가하여 테이블스페이스를 확장할 수 있다.

그림 6 Tablespace 개념



테이블스페이스는 저장하는 data의 유형에 따라, 다음과 같이 세 종류로 구분할 수 있다.

- DATA TABLESPACE
 - 테이블, 로깅 인덱스 등 영속적으로 저장 관리되는 데이터를 저장하는 테이블스페이스
- TEMPORARY TABLESPACE
 - 로깅없는 인덱스, 질의 수행 중 생성되는 hash, sort 등의 휘발성 데이터를 저장 관리하는 테이블스페이스
- UNDO TABLESPACE
 - 트랜잭션을 철회하기 위해 데이터 갱신 정보를 저장 관리하는 테이블스페이스

DATA 테이블스페이스에 저장되는 테이블, 인덱스 (LOGGING) 등은 데이터를 영속적으로 관리하기 위한 redo log 등을 생성하는 반면, TEMPORARY 테이블스페이스에 저장되는 로깅없는 인덱스는 redo log를 생성하지 않는다. 로깅없는 인덱스의 경우, 변경 내용을 로깅하지 않는 반면, 시스템을 재구동할 때 테이블 데이터를 기준으로 재구축되므로 색인의 기능은 그대로 유지된다.

테이블스페이스는 SQL schema 객체가 저장되는 물리적 저장소이며 테이블, 인덱스 등을 생성할 때 특정 테이블스페이스를 지정할 수 있다. 즉, 하나의 테이블 및 테이블과 관련된 인덱스, 테이블의 제약 조건을 위해 생성되는 인덱스는 서로 다른 테이블스페이스에 저장할 수 있다. 객체를 생성할 때 테이블스페이스를 지정하는 방법은 다음 구문들을 참조한다.

- CREATE TABLE

- **CREATE INDEX**
- **ALTER TABLE name ADD CONSTRAINT**

테이블, 인덱스와 같은 객체를 생성할 때 테이블스페이스를 지정하지 않으면, 사용자에게 지정된 기본 테이블스페이스가 사용된다. 사용자의 기본 테이블스페이스를 설정하는 방법은 다음 구문들을 참조한다.

- **CREATE USER**
- **ALTER USER**

테이블스페이스에 대한 자세한 내용은 **테이블스페이스 관리**를 참조한다.

13.7 Table

테이블 관련 구문

테이블을 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- 테이블 생성
 - CREATE TABLE
 - CREATE TABLE AS SELECT
 - CREATE GLOBAL TEMPORARY TABLE
- 테이블 제거
 - DROP TABLE
 - TRUNCATE TABLE
- 테이블 변경
 - ALTER TABLE
 - ALTER TABLE name RENAME TO
 - ALTER TABLE name STORAGE
- 테이블의 column 추가, 제거, 변경
 - ALTER TABLE name ADD COLUMN
 - **16.42** ALTER TABLE name SET UNUSED COLUMN
 - ALTER TABLE name ALTER COLUMN
 - ALTER TABLE name RENAME COLUMN
- 테이블의 제약 조건 추가, 제거, 변경
 - ALTER TABLE name ADD CONSTRAINT
 - ALTER TABLE name DROP CONSTRAINT
 - ALTER TABLE name ALTER CONSTRAINT
- 테이블의 부가로그 추가, 제거
 - ALTER TABLE name ADD SUPPLEMENTAL LOG
 - ALTER TABLE name DROP SUPPLEMENTAL LOG
- 테이블 통계 정보: ANALYZE TABLE

테이블 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-14 테이블 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_ALL_TABLES	사용자가 접근 가능한 테이블 정보
	ALL_COL_COMMENTS	사용자가 접근 가능한 column의 주석 정보
	ALL_CONSTRAINTS	사용자가 접근 가능한 제약 조건 정보
	ALL_CONS_COLUMNS	사용자가 접근 가능한 제약 조건의 column 정보
	ALL_TABLES	사용자가 접근 가능한 테이블 정보
	ALL_TAB_COLS	사용자가 접근 가능한 column 정보
	ALL_TAB_COLUMNS	사용자가 접근 가능한 column 정보
	ALL_TAB_COMMENTS	사용자가 접근 가능한 테이블의 주석 정보
	ALL_TAB_IDENTITY_COLS	사용자가 접근 가능한 테이블의 identity column 정보
	USER_ALL_TABLES	사용자가 소유한 테이블의 정보
	USER_COL_COMMENTS	사용자가 소유한 column의 주석 정보
	USER_CONSTRAINTS	사용자가 소유한 제약 조건 정보
	USER_CONS_COLUMNS	사용자가 소유한 제약 조건의 column 정보
	USER_TABLES	사용자가 소유한 테이블 정보
	USER_TAB_COLS	사용자가 소유한 column 정보
	USER_TAB_COLUMNS	사용자가 소유한 column 정보
	USER_TAB_COMMENTS	사용자가 소유한 테이블의 주석 정보
	USER_TAB_IDENTITY_COLS	사용자가 소유한 테이블의 identity column 정보
INFORMATION_SCHEMA	COLUMNS	사용자가 접근 가능한 column 정보
	CONSTRAINT_COLUMN_USAGE	사용자가 접근 가능한 제약 조건의 column 정보
	CONSTRAINT_TABLE_USAGE	사용자가 접근 가능한 제약 조건의 테이블 정보
	KEY_COLUMN_USAGE	사용자가 접근 가능한 key 제약 조건의 column 정보
	TABLES	사용자가 접근 가능한 테이블 정보
	TABLE_CONSTRAINTS	사용자가 접근 가능한 제약 조건 정보

테이블 개념

테이블은 데이터베이스를 구성하는 가장 기본적인 객체이다. SQL 표준에서는 테이블을 base table이라고 하고 view는 viewed table이라고 한다.

테이블은 column과 row로 구성된다. 테이블은 다수의 row들로 구성되며, 각 row의 column 개수와 순서는 동일하다. 테이블은 한 개 이상의 column으로 구성되며 column에는 이름이 있지만, row에는 이름이 없다. Row의 순서가 데이터 추가 순서와 반드시 동일한 것은 아니다. 특정 row와 특정 column이 교차하는 위치의 데이터를 value라고 한다. Column은 동일한 데이터 타입을 가지는 value들의 집합이다.

테이블의 각 column은 테이블 내에서 다른 column들과 구별되는 고유한 이름을 가지며, value의 특성에 부합하는 데이터 타입을 가진다. 데이터 타입과 관련된 내용은 **Data Type** 절을 참조한다. 데이터 무결성을 위해 테이블에 제약 조건을 추가할 수 있으며, 제약 조건과 관련된 내용은 **CREATE TABLE**과 **ALTER TABLE name ADD CONSTRAINT**

구문을 참조한다. 테이블을 조회하는 질의들의 성능을 향상시키기 위해 인덱스를 생성할 수 있으며, 인덱스와 관련된 내용은 **Index** 절을 참조한다.

다음은 **CREATE TABLE** 구문을 이용하여 lineitem 테이블을 생성하는 예이다.

```
CREATE TABLE lineitem
(
    l_orderkey      INTEGER      NOT NULL
  , l_partkey      INTEGER      NOT NULL
  , l_suppkey      INTEGER      NOT NULL
  , l_linenum      INTEGER      NOT NULL
  , l_quantity     NUMERIC(12,2)
  , l_extendedprice NUMERIC(12,2)
  , l_discount     NUMERIC(12,2)
  , l_tax          NUMERIC(12,2)
  , l_returnflag   CHAR(1)      NOT NULL  DEFAULT 'F'
  , l_linestatus   CHAR(1)
  , l_shipdate     DATE
  , l_commitdate   DATE
  , l_receiptdate  DATE
  , PRIMARY KEY (l_orderkey, l_linenum) INDEX lineitem_pk_idx TABLESPACE mem_temp_tbs
) TABLESPACE mem_data_tbs;
```

위의 예에서 lineitem 테이블에는 다수의 column과 함께 제약 조건들이 정의되어 있다. 제약 조건으로 column을 정의할 때 l_orderkey, l_partkey, l_suppkey, l_linenum column 등에 NOT NULL 제약 조건을 정의하였고, 두 개의 column l_orderkey와 l_linenum를 조합하여 PRIMARY KEY 제약 조건을 정의하였다. Column 정의와 함께 기술하는 제약 조건을 in-line 제약 조건이라 하며, column 정의와 별도로 기술하는 제약 조건을 out-line 제약 조건이라 한다.

Column의 기본값을 지정하기 위해 l_returnflag column에 DEFAULT 절을 사용하여 'F' 값을 선언하였으며, PRIMARY KEY 제약과 함께 생성되는 인덱스의 이름을 lineitem_pk_idx라고 별도로 명명하고 해당 인덱스가 저장될 테이블스페이스를 mem_temp_tbs라고 지정하였다. 그리고 테이블이 저장될 물리적 저장소로 mem_data_tbs 테이블스페이스를 사용하고 있다.

다음은 **ALTER TABLE name ADD CONSTRAINT** 구문을 이용하여 테이블에 제약 조건을 추가하는 예이다.

```
ALTER TABLE lineitem
    ADD CONSTRAINT lineitem_unique_all_key
    UNIQUE( l_orderkey ASC, l_partkey DESC, l_suppkey DESC, l_linenum ASC);
```

위의 예에서 lineitem 테이블에 UNIQUE 제약 조건을 추가하였으며, 제약 조건을 위해 자동으로 생성될 인덱스에 대해 column의 정렬 순서 (ASC/ DESC)를 명시하고 있다.

다음은 **ALTER TABLE name ADD COLUMN** 구문을 이용하여 테이블에 새로운 column을 추가하는 예이다.

```
ALTER TABLE lineitem ADD COLUMN
(
    l_shipinstruct CHAR(25)
, l_shipmode      CHAR(10)
, l_comment       VARCHAR(44)
);
```

위의 예에서는 다수의 column을 추가하고 있다. Column을 추가할 때 in-line 제약 조건이나 기본값을 함께 명시할 수 있다.

다음은 **CREATE INDEX** 구문을 이용하여 테이블에 인덱스를 생성하는 예이다.

```
CREATE INDEX lineitem_idx_shipdate ON lineitem( l_shipdate ASC NULLS LAST );
```

위의 예에서는 질의 조건으로 자주 사용되는 l_shipdate column에 인덱스를 생성하며, column의 정렬 순서를 오름차순 (ASC)으로 지정하고 NULL 값이 존재할 경우 마지막에 위치하도록 지정한다.

테이블에 데이터를 추가/ 삭제/ 갱신하는 DML 구문에 대해서는 **Data Manipulation Language** 절을 참조하고, 테이블의 데이터를 조회하는 SELECT 구문에 대해서는 **Data Query Language** 절과 SELECT 구문을 참조한다.

Global Temporary Table

테이블의 정의는 모든 사용자가 공유하지만, 데이터는 각 세션별로 구분되어 사용되는 임시 테이블의 한 종류이다.

테이블의 정의는 CREATE GLOBAL TEMPORARY TABLE 명령을 실행할 때 생성되지만, 물리적 저장 공간 (segment)은 세션에서 해당 테이블에 처음으로 INSERT 명령을 수행할 때 세션에 종속된 상태로 생성된다. 세션이 종료되면 해당 세션에서 생성된 모든 global temporary table들에 할당되었던 저장 공간들이 해제된다. 생성할 때 옵션 지정 여부에 따라 COMMIT이나 ROLLBACK 할 때 남아있는 데이터를 TRUNCATE할지 여부를 지정할 수 있다.

Cluster 관련한 구문을 제외하고 일반 테이블이 제공하는 모든 DDL과 DML을 지원하며, DDL 명령은 현재 세션에서 사용 중인 global temporary table에 에러를 반환한다. 단, global temporary table에 대한 TRUNCATE TABLE 명령은 현재 세션에만 적용되기 때문에 다른 세션에서 사용 중이라고 하더라도 에러를 반환하지 않는다.

Global temporary table은 temporary tablespace에서만 정의할 수 있기 때문에 재시작 복구를 위한 로그(redo log)를 기록하지 않는다. 하지만, MVCC와 rollback을 위해 이전 값 (undo log)은 기록하는데, undo log가 기록되는 공간은 **TEMP_UNDO_ENABLED** 프로퍼티를 이용하여 system undo tablespace나 system temp tablespace 중에 선택할 수 있다.

TEMP_UNDO_ENABLED 값이 1이면 트랜잭션의 undo relation과 별개로 session의 temp undo relation에 undo log를 기록한다. 만약 트랜잭션이 global temporary table에 대한 DML만 수행하는 경우에는 트랜잭션 레코드와 commit 로그도 기록하지 않으므로 DML 성능이 향상된다.

세션에서 사용된 공간들을 해제하면 기본적으로 해당 tablespace에 반환되고, 이 후 다시 공간을 할당할 때 tablespa

ce로부터 할당되어 사용된다. Tablespace에서 공간을 할당하고 반환하는 과정은 다른 세션들과의 동시성 유지 및 공간 할당, 해제 비용이 크다. 따라서 **TEMP_SEGMENT_CACHE_SIZE** 프로퍼티를 통해 세션에서 사용 후 해제된 공간을 tablespace에 반환하지 않고 세션에서 재사용할 수 있다.

즉, **TEMP_SEGMENT_CACHE_SIZE**를 0 (기본값)으로 설정하면 사용 후에 해제되는 공간을 tablespace에 즉시 반환하고, 1보다 큰 값 (최대 4294967295)으로 설정하면 공간을 반환할 때 지정된 개수만큼의 공간만 session에서 재사용한다.

세션에서 더 이상 global temporary table을 사용하지 않는 경우 **ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL**; 구문을 사용하여 segment cache의 segment들을 한꺼번에 정리한다.

다음은 **CREATE GLOBAL TEMPORARY TABLE** 구문을 사용하여 global temporary table을 생성하는 예이다.

```
CREATE GLOBAL TEMPORARY TABLE SESSION_TABLE1(
    COL1    CHAR(10)
    ,COL2    VARCHAR2(20)
    ,COL3    NUMBER(10)
) ON COMMIT DELETE ROWS;
```

생성된 global temporary table들에 대한 정보는 일반 table들에 대한 정보를 조회하는 것과 동일한 방법으로 **DICTIONARY** 테이블들이나 view들에서 조회할 수 있다.

Cluster 테이블

Cluster 환경의 테이블에 대한 개념은 **Cluster Table**과 **Shard** 을 참조한다.

13.8 Index

Index 관련 구문

Index를 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Index 생성: **CREATE INDEX**
- Index 제거: **DROP INDEX**
- Index 변경: **ALTER INDEX**

인덱스 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-15 인덱스 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_INDEXES	사용자가 접근 가능한 인덱스 정보
	ALL_IND_COLUMNS	사용자가 접근 가능한 인덱스의 column 정보
	USER_INDEXES	사용자가 소유한 인덱스 정보
	USER_IND_COLUMNS	사용자가 소유한 인덱스의 column 정보

Index 개념

인덱스는 테이블 관련 객체로써 테이블을 조회할 때 데이터 접근 성능을 향상시키기 위한 객체이다. 각 인덱스는 테이블의 하나 이상의 column에 포함된 데이터를 이용한 key 값으로 구성되며, 테이블과는 분리된 객체이다. 데이터베이스는 인덱스를 생성할 때 자동으로 인덱스의 key 데이터를 구축하며, 테이블에 데이터를 추가/ 삭제/ 갱신할 때 인덱스의 key 데이터가 자동으로 관리된다.

다음은 질의의 예이다.

```
SELECT data FROM t1 WHERE id = 12345;
```

인덱스가 없는 경우, 테이블의 모든 row들을 검사하여 조건에 부합하는 결과를 찾는다. 만약 테이블이 다량의 row들로 구성된 반면, 조건에 부합하는 결과의 개수가 적으면 위 질의는 매우 비효율적인 응답 시간을 갖게 된다. 다음과 같이 **CREATE INDEX** 구문을 이용하여 id column에 인덱스를 생성할 경우, optimizer가 테이블 full scan과 index scan의 비용을 평가한 후 인덱스를 이용한 검색을 선택하여 질의 성능을 향상시킨다.

```
CREATE INDEX t1_idx_id ON t1(id);
```

인덱스를 생성할 때 두 개 이상의 column을 인덱스의 key로 사용할 수 있으며, 둘 이상의 key로 구성된 인덱스를 co

Composite index라고 한다. Composite index는 첫 번째 key를 기준으로 정렬하고, 첫 번째 key 값이 동일할 경우 두 번째 key 값을 기준으로 정렬한다. 이러한 방식으로 key의 개수만큼 정렬한다.

인덱스를 생성할 때 column을 오름차순 (ASC) 또는 내림차순 (DESC)으로 정렬하도록 지정할 수 있다. 그리고 NULL 값의 정렬 순서를 NULLS FIRST 또는 NULLS LAST로 지정할 수 있다. 다음 예제를 참조한다.

```
gSQL> CREATE TABLE t1 ( value INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES (1), (NULL), (3), (2), (NULL);
5 rows created.
gSQL> CREATE INDEX idx1 ON t1 ( value ASC NULLS LAST );
Index created.
gSQL> CREATE INDEX idx2 ON t1 ( value DESC NULLS FIRST );
Index created.
gSQL> SELECT /*+ INDEX(t1, idx1) */ * FROM t1;
VALUE
-----
      1
      2
      3
    null
    null
5 rows selected.
gSQL> SELECT /*+ INDEX(t1, idx2) */ * FROM t1;
VALUE
-----
    null
    null
      3
      2
      1
5 rows selected.
```

위의 예에서 idx1 인덱스는 정렬 순서를 오름차순 (ASC) NULLS LAST로 지정하였고, idx2 인덱스는 정렬 순서를 내림차순 (DESC) NULLS FIRST로 지정하였다. 동일한 질의로 테이블을 조회할 때 인덱스 힌트를 서로 다르게 주어 해당 인덱스를 이용한 모든 row 조회가 가능하도록 하였다. 인덱스 idx1을 사용한 결과는 오름차순으로 정렬되고 NULL 값들이 마지막에 위치하는 반면, 인덱스 idx2를 사용한 결과는 내림차순으로 정렬되고 NULL 값들이 처음에 위치한다.

UNIQUE 개념

인덱스를 생성할 때 UNIQUE 또는 non-unique 인덱스를 생성할 수 있으며, UNIQUE 인덱스를 생성할 때 key들이 UNIQUE 하지 않을 경우 에러가 발생한다.

UNIQUE 인덱스와 UNIQUE 제약 조건은 key 값으로 NULL 값을 허용한다.

NULL 값을 포함할 경우 UNIQUE 여부에 대한 truth 테이블은 다음과 같다. 즉, key가 하나일 경우 다수의 null 값을 가질 수 있다

표 13-16 두 값의 UNIQUE 여부

Value1	Value2	UNIQUE 여부
1	1	false
1	2	true
1	null	true
null	null	true

두 개 이상의 key로 구성되는 UNIQUE 인덱스 또는 UNIQUE 제약 조건은 전체 또는 일부 값이 null일 수 있는데, composite key에 null을 포함할 경우 UNIQUE에 대한 truth 테이블은 다음과 같다.

표 13-17 Composite key의 UNIQUE 여부

Row1	Row2	UNIQUE 여부
(1, 1)	(1, 1)	false
(1, 1)	(1, 2)	true
(1, null)	(1, null)	false
(1, null)	(2, null)	true
(null, null)	(null, null)	true

참고로 UNIQUE에 대한 정의는 SQL 표준에서 다음과 같이 변경되었다.

노트

- SQL1999까지의 UNIQUE 정의

If there are no two rows in T such that the value of each column in one row is non-null and is **equal to** the value of the corresponding column in the other row according to Subclause 8.2, “<comparison predicate>”, then the result of the <unique predicate> is true; otherwise, the result of the <unique predicate> is false.

- SQL2003 이후의 UNIQUE 정의

If there are no two rows in T such that the value of each column in one row is non-null and is **not distinct from** the value of the corresponding column in the other row, then the result of the $\langle \text{unique predicate} \rangle$ is True; otherwise, the result of the $\langle \text{unique predicate} \rangle$ is False.

GOLDILOCKS는 SQL2003 이후의 표준인 SQL2011을 따르며, SQL 표준의 UNIQUE는 다음 표와 같이 composite key의 UNIQUE 여부에 따라 정의된다.

표 13-18 SQL 표준 composite key의 UNIQUE 여부

Row1	Row2	~ SQL1999	SQL2003 ~
(1, 1)	(1, 1)	false	false
(1, 1)	(1, 2)	true	true
(1, null)	(1, null)	true	false
(1, null)	(2, null)	true	true
(null, null)	(null, null)	true	true

각 DBMS 벤더들은 UNIQUE를 정의할 때 다음과 같이 SQL 표준을 따르고 있다.

- SQL2003 이후의 UNIQUE 정의를 따르는 DBMS: Oracle, SQL Server
- SQL1999까지의 UNIQUE 정의를 따르는 DBMS: Postgres, MySQL

13.9 View

View 관련 구문

View를 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- View 생성: **CREATE VIEW**
- View 제거: **DROP VIEW**
- View 변경: **ALTER VIEW**

View 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-19 View 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_VIEWS	사용자가 접근 가능한 view 정보
	ALL_DEPENDENCIES	사용자가 접근 가능한 view와 관련된 객체정보
	USER_VIEWS	사용자가 소유한 view 정보
	USER_DEPENDENCIES	사용자가 소유한 view와 관련된 객체정보
INFORMATION_SCHEMA	VIEWS	사용자가 접근 가능한 view 정보
	VIEW_TABLE_USAGE	View를 생성할 때 사용한 테이블 정보
	VIEW_ROUTINE_USAGE	View를 생성할 때 사용한 stored function 정보

View 개념

테이블이 데이터를 저장하는 물리적 릴레이션인 반면에 view는 질의로 구성된 논리적 릴레이션이다. SQL 표준에서는 viewed table이라고 한다. View에 수행되는 질의는 테이블과 동일하게 사용할 수 있다.

View에는 다음과 같은 장점이 있다.

- 테이블의 일부 정보만 조회할 수 있도록 하여 데이터 접근을 제한할 수 있다.
- 복잡하고 자주 사용되는 질의를 하나의 view로 작성하여 질의의 복잡도를 낮출 수 있다.
- View의 column 이름 변경, 데이터 갱신 등을 통해 테이블과 다른 관점에서 데이터를 제공할 수 있다.
- View에 대한 응용 프로그램을 작성할 때 테이블 구조 변경 등에 영향을 받지 않도록 할 수 있다.

다음과 같이 **CREATE VIEW** 구문을 통해 작성된 view는 질의를 수행할 때 in-line view로 대체된다.

- View 생성

```
CREATE VIEW v1 ( v_id, v_sum )
AS
SELECT l_partkey, SUM( l_quantity )
FROM lineitem
GROUP BY l_partkey;
```

- View에 대한 질의

```
SELECT v_id, v_sum
FROM v1
WHERE v_sum > 1000;
```

- View에 대한 해석

```
SELECT v_id, v_sum
FROM ( SELECT l_partkey, SUM( l_quantity )
      FROM lineitem
      GROUP BY l_partkey
    ) v1 ( v_id, v_sum )
WHERE v_sum > 1000;
```

다음과 같이 SELECT 구문에 모든 column을 의미하는 asterisk(*)를 사용하여 v1 view를 생성한 경우, view가 접근하는 테이블 t1에 새로운 column addr가 추가된 후에도 v1 view에 대해 질의를 수행하면 추가된 column을 포함하여 모든 column을 조회할 수 있다.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- Asterisk (*)를 사용하여 view 생성

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1;
View created.
```

- View 조회

```
gSQL> SELECT * FROM v1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

- View가 참조하는 테이블에 column 추가

```
gSQL> ALTER TABLE t1 ADD COLUMN addr VARCHAR(1024) DEFAULT 'N/A';  
Table altered.
```

- Column 추가 후 view 조회

```
gSQL> SELECT * FROM v1;  
ID NAME   ADDR  
--  
1 leekmo N/A  
1 row selected.
```

그러나 위와 같이 asterisk (*)를 이용하여 view를 생성하면 테이블 구조를 변경할 때 응용 프로그램을 변경시킬 수 있어 바람직하지 않다.

13.10 Sequence

Sequence 관련 구문

Sequence를 생성, 제거, 변경, 사용하기 위한 구문은 다음과 같다.

- Sequence 생성: **CREATE SEQUENCE**
- Sequence 제거: **DROP SEQUENCE**
- Sequence 변경: **ALTER SEQUENCE**
- Sequence 사용: **NEXTVAL, CURRVAL**

시퀀스 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-20 시퀀스 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_SEQUENCES	사용자가 접근 가능한 시퀀스 정보
	USER_SEQUENCES	사용자가 소유한 시퀀스 정보
INFORMATION_SCHEMA	SEQUENCES	사용자가 접근 가능한 시퀀스 정보

Sequence 개념

시퀀스는 자동으로 순차 번호를 생성하는 객체이다. SQL 표준에서는 sequence generator라고 한다. 시퀀스는 unique key나 primary key를 자동으로 관리하기에 유용한 객체이며, 하나의 시퀀스를 여러 테이블에서 사용할 수 있다.

다음은 하나의 시퀀스 객체를 사용하여 id column에 해당하는 값을 자동으로 생성한 후 여러 테이블에서 사용하는 예이다.

```
gSQL> CREATE SEQUENCE seq;
Sequence created.
gSQL> INSERT INTO t1 (id, name) VALUES ( seq.NEXTVAL, 'leekmo' );
1 row created.
gSQL> INSERT INTO t2 (id, addr) VALUES ( seq.CURRVAL, 'Seoul, Korea' );
1 row created.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
1 row selected.
```

```
gSQL> SELECT * FROM t2;
ID ADDR
-- -----
1 Seoul, Korea
1 row selected.
```

위의 예에서 seq.NEXTVAL 함수를 이용하여 t1 테이블에 있는 id column의 다음 번호를 자동으로 생성하고, seq.CURRVAL 함수를 이용하여 동일한 값을 t2 테이블의 id column의 값으로 사용하였다. 시퀀스를 생성할 때 자동 생성되는 번호의 시작값, 증분값, 최소값, 최대값, cycle 여부, 캐쉬값 등을 지정할 수 있는데 자세한 내용은 **CREATE SEQUENCE** 구문을 참조한다.

시퀀스와 유사한 기능을 하는 identity column은 하나의 테이블에서 자동으로 번호를 생성하는 column이며 다음과 같이 사용할 수 있다.

```
gSQL> CREATE TABLE t1 ( id INTEGER GENERATED ALWAYS AS IDENTITY, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 (name) VALUES ( 'leekmo' );
1 row created.
gSQL> INSERT INTO t1 (name) VALUES ( 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 (name) VALUES ( 'xcom73' );
1 row created.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
2 mkkim
3 xcom73
3 rows selected.
```

위의 예에서 t1 테이블을 생성할 때 id column을 identity column으로 생성했고 INSERT 구문을 수행할 때 id column 값으로 identity column이 자동으로 생성한 값이 입력되었다. Identity column에 대한 자세한 내용은 **CREATE TABLE** 구문의 <identity column specification> 절을 참조한다.

시퀀스와 identity column은 순차 번호를 생성한다는 점에서 기능상 유사하지만 다음과 같은 차이가 있다.

- 시퀀스는 SQL schema 객체인 반면에 identity column은 테이블의 column 중 하나이다.
- 시퀀스는 여러 테이블에 걸쳐 사용할 수 있지만 identity column은 한 테이블에서만 사용할 수 있다.

시퀀스가 생성된 후에 **NEXTVAL** 또는 **CURRVAL** 함수를 이용하여 시퀀스 값을 사용할 수 있다. 시퀀스 값은 트랜잭션과 독립적으로 생성되며, 트랜잭션의 COMMIT 또는 ROLLBACK에 영향을 받지 않는다.

```

gSQL> INSERT INTO t1(id) VALUES( seq.NEXTVAL );
1 row created.
gSQL> SELECT id FROM t1;
ID
--
1
1 row selected.
gSQL> ROLLBACK;
Rollback complete.
gSQL> INSERT INTO t1(id) VALUES( seq.NEXTVAL );
1 row created.
gSQL> SELECT id FROM t1;
ID
--
2
1 row selected.

```

위 예제의 첫 번째 INSERT 구문에서 seq.NEXTVAL 함수는 1 값을 생성했고 이후 트랜잭션을 ROLLBACK 한 후 사용한 seq.NEXTVAL의 값으로써 트랜잭션과 무관하게 이후 값부터 다시 증가한 2를 생성했다.

시퀀스 값은 다음과 같은 위치에서만 사용할 수 있다.

- 최상위 SELECT 구문의 select list 값
 - SELECT seq.NEXTVAL FROM dual;
- INSERT .. SELECT 구문의 select list 값
 - INSERT INTO t1(id) SELECT seq.NEXTVAL FROM dual;
- INSERT .. VALUES 구문의 입력값
 - INSERT INTO t1(id) VALUES (seq.NEXTVAL);
- UPDATE 구문의 SET 값
 - UPDATE t1 SET id = seq.NEXTVAL;

시퀀스 값은 위에서 정의한 위치 이외에서는 사용할 수 없으며, subquery, aggregation 함수의 인자, WHERE, DISTINCT, GROUP BY, HAVING, ORDER BY 절 등에 위치할 수 없다.

Cluster Sequence

GOLDILOCKS를 cluster system으로 구성하여 사용할 때는 내부적으로 global 시퀀스 객체가 사용된다. Global 시퀀스 객체는 cluster system 전체에서 공용으로 사용할 시퀀스 값들의 pool을 생성해둔 후, 각 member 노드가 NEXTVAL을 호출할 때 cache 크기 만큼씩 할당해주는 방식이다. 즉, 만약 특정 노드가 20 개의 시퀀스 값들을 할당받았다면, 다른 노드들은 해당 값들의 다음 값부터 할당받을 수 있다. Member 노드는 global 시퀀스 객체로부터 할당받은 시퀀스 값들을 자신의 local cache에 적재해둔 후, 이를 모두 소진할 때까지 NEXTVAL 호출 결과로 반환한다.

Global 시퀀스 객체는 기존의 standalone database 용 시퀀스와 비교하여 다음과 같은 특징과 제약 사항을 가진다.

- ALTER SEQUENCE 구문에서 INCREMENT BY 옵션을 사용하여 부호를 변경할 수 없다. (크기는 변경할 수 있다.)
- CYCLE 옵션을 사용할 경우에는 전체 시퀀스 pool의 크기에 따라 member 노드 간에 중복된 값이 반환될 수 있다. 따라서 CYCLE 옵션이 필요한 경우에는 INCREMENT BY, CACHE SIZE, cluster member 노드 개수를 고려하여 시퀀스 pool 을 충분히 크게 생성해야 한다.
- 장애 상황이 아닌 경우라도 특정 member 노드를 기준으로 반환되는 시퀀스 값이 연속적이지 않을 수 있다. 물론, 하나의 member 노드만 NEXTVAL을 호출하는 경우에는 연속적인 시퀀스 값을 얻을 수 있다.
- NOCACHE일 경우에는 CACHE SIZE가 1이므로 local cache에 추가적인 시퀀스 값들을 적재해두지 않는다. 이 경우, NEXTVAL을 호출할 때마다 global 시퀀스 객체로부터 매번 시퀀스를 한 개씩 할당받게 되어, 네트워크 비용이 증가하므로 성능이 상당히 하락할 수 있다.
- 시퀀스를 생성, 변경, 삭제할 때는 기본적으로 AUTO COMMIT으로 동작한다.
- ALTER 구문을 사용하여 CACHE와 INCREMENT 크기를 변경할 경우, 모든 노드의 local cache에 적재된 시퀀스 값들이 모두 리셋된다. 즉, 이후에 NEXTVAL을 호출할 때 global 시퀀스 객체로부터 새로운 시퀀스 집합을 다시 할당받아야 한다.

13.11 Synonym

Synonym 관련 구문

Synonym을 생성, 제거하기 위한 구문은 다음과 같다.

- Synonym 생성: **CREATE SYNONYM**
- Synonym 제거: **DROP SYNONYM**

Synonym 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-21 Synonym 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_SYNONYMS	모든 synonym 정보
	USER_SYNONYMS	사용자가 소유한 synonym 정보

Synonym 개념

Synonym은 다음과 같은 객체에 대한 대체명이다.

- Table
- View
- Sequence
- Stored procedure
- Stored function
- 다른 synonym

SELECT, INSERT, UPDATE, DELETE, LOCK TABLE, GRANT, REVOKE, COMMENT 구문에 대체명으로 사용할 수 있다.

Synonym을 생성해서 사용하면 기본 객체의 스키마가 변경되더라도 응용 프로그램을 수정할 필요없이 synonym만 새로 정의하면 되기 때문에 매우 편리하다. 또한 객체의 실제 이름과 소유자를 감춤으로써 데이터베이스 보안성을 강화할 수 있고, 긴 객체 이름을 짧게 수정하여 사용성을 높일 수도 있다.

Synonym에는 private synonym과 public synonym이 있는데, private synonym은 스키마 객체이고 public synonym은 비스키마 객체이다.

아래 테이블을 가리키는 private synonym과 public synonym을 생성하고 사용하는 예제를 통해 그 개념을 설명한다.

```
gSQL> \CONNECT u1 u1
gSQL> CREATE TABLE u1.t1 (col1 INTEGER );
gSQL> INSERT INTO u1.t1 VALUES(1);
gSQL> COMMIT;
```

Private Synonym

Private synonym은 스키마 객체로써 이를 생성할 때 스키마 이름이 생략된 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

```
gSQL> \CONNECT u2 u2
gSQL> CREATE SYNONYM u2.syn1 FOR u1.t1;
Synonym created.
gSQL> SELECT * FROM u2.syn1;
ERR-42000(16254): lacks privilege (SELECT ON TABLE "U1"."T1")
```

Synonym은 대체명일 뿐이므로 synonym을 생성한 사용자라 하더라도 기본 객체 u1.t1에 대한 적절한 권한이 없으면 이를 사용할 수 없다.

```
gSQL> \CONNECT u1 u1
gSQL> GRANT SELECT ON TABLE u2.syn1 TO u2;
gSQL> \CONNECT u2 u2
gSQL> SELECT * FROM u2.syn1;
COL1
----
    1
1 row selected.
gSQL> SELECT * FROM u1.t1;
COL1
----
    1
1 row selected.
gSQL> DROP SYNONYM u2.syn1;
Synonym dropped.
```

위 예제에서 u2.syn1의 SELECT 권한을 u2에게 주었지만, 이는 u1.t1의 SELECT 권한을 u2에게 준 것과 동일하다. 따라서 synonym에 권한을 부여할 때는 주의해야 한다.

Public Synonym

Public synonym은 비스키마 객체로써 이를 생성하거나 삭제할 때는 스키마 이름을 명시할 수 없다.

```
gSQL> \CONNECT u2 u2
gSQL> CREATE PUBLIC SYNONYM pubSyn1 FOR u1.t1;
Synonym created.
gSQL> SELECT * FROM pubSyn1;
ERR-42000(16254): lacks privilege (SELECT ON TABLE "U1"."T1")
```

Public synonym의 소유자는 없고 모든 사용자가 사용할 수 있지만, 기본 객체에 대한 적절한 권한이 없으면 기본 객체에 접근할 수 없다.

```
gSQL> \CONNECT u1 u1
gSQL> GRANT SELECT ON TABLE pubSyn1 TO u2;
gSQL> \CONNECT u2 u2
gSQL> SELECT * FROM pubSyn1;
COL1
----
    1
1 row selected.
gSQL> SELECT * FROM u1.t1;
COL1
----
    1
1 row selected.
gSQL> DROP SYNONYM pubSyn1;
Synonym dropped.
```

13.12 Stored Procedure

Stored Procedure 관련 구문

Stored procedure를 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Stored procedure 생성: **CREATE PROCEDURE**
- Stored procedure 제거: **DROP PROCEDURE**
- Stored procedure 변경: **ALTER PROCEDURE**

Stored procedure 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-22 Stored procedure 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_ARGUMENTS	사용자가 접근 가능한 procedure, function의 argument 정보
	ALL_DEPENDENCIES	사용자가 접근 가능한 procedure, function과 관련된 객체 정보
	ALL PROCEDURES	사용자가 접근 가능한 procedure, function 객체 정보
	ALL_SOURCE	사용자가 접근 가능한 procedure, function의 source text 정보
	USER_ARGUMENTS	사용자가 소유한 procedure, function의 argument 정보
	USER_DEPENDENCIES	사용자가 소유한 procedure, function과 관련된 객체 정보
	USER PROCEDURES	사용자가 소유한 procedure, function 객체 정보
	USER_SOURCE	사용자가 소유한 procedure, function의 source text 정보
INFORMATION_SCHEMA	PARAMETERS	사용자가 접근 가능한 procedure, function의 argument 정보
	ROUTINES	사용자가 접근 가능한 procedure, function의 객체 정보
	ROUTINE_ROUTINE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 procedure, function 정보
	ROUTINE_SEQUENCE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 sequence 정보
	ROUTINE_TABLE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 table, view 정보

Stored Procedure 개념

Stored procedure는 procedure 형태의 persistent stored module 중 하나이며 다른 schema-level database 객체와 마찬가지로 schema 단위로 정의되고 관리된다. Procedure 형태이기 때문에 반환값은 정의되지 않으며, CALL 구문 또는 다른 stored procedure나 stored function 내에서 직접 호출하여 사용된다.

Stored procedure에 대한 자세한 내용은 **schema-level procedure**를 참조한다.

Stored procedure는 다음과 같이 사용된다.

```
CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
    V1 INTEGER;
BEGIN
    SELECT COUNT(*)
        INTO V1
        FROM T1
        WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
BEGIN
    PROC1( 2, 4 ); -- call schema-level procedure
END;
/
V1 = 3
Anonymous PL block executed.
```

13.13 Stored Function

Stored Function 관련 구문

Stored function을 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Stored function 생성: **CREATE FUNCTION**
- Stored function 제거: **DROP FUNCTION**
- Stored function 변경: **ALTER FUNCTION**

Stored function 객체와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 13-23 Stored function 객체 관련 정보

스키마	View	설명
DICTIONARY_SCHEMA	ALL_ARGUMENTS	사용자가 접근 가능한 procedure, function의 argument 정보
	ALL_DEPENDENCIES	사용자가 접근 가능한 procedure, function과 관계된 객체 정보
	ALL_PROCEDURES	사용자가 접근 가능한 procedure, function 객체 정보
	ALL_SOURCE	사용자가 접근 가능한 procedure, function의 source text 정보
	USER_ARGUMENTS	사용자가 소유한 procedure, function의 argument 정보
	USER_DEPENDENCIES	사용자가 소유한 procedure, function과 관계된 객체 정보
	USER_PROCEDURES	사용자가 소유한 procedure, function 객체 정보
	USER_SOURCE	사용자가 소유한 procedure, function의 source text 정보
INFORMATION_SCHEMA	PARAMETERS	사용자가 접근 가능한 procedure, function의 argument 정보
	ROUTINES	사용자가 접근 가능한 procedure, function의 객체 정보
	ROUTINE_ROUTINE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 procedure, function 정보
	ROUTINE_SEQUENCE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 sequence 정보
	ROUTINE_TABLE_USAGE	사용자가 접근 가능한 procedure, function이 참조하는 table, view 정보

Stored Function 개념

Stored function은 function 형태의 persistent stored module 중 하나이며 다른 schema-level database 객체와 마찬가지로 schema 단위로 정의되고 관리된다. 함수 형태이기 때문에 반환값을 정의해야 하며, CALL 구문 또는 다른 stored procedure나 stored function 내에서의 직접 호출 또는 일반 SQL 내부 표현식 내에서 호출하여 사용된다.

Stored function에 대한 자세한 내용은 **schema-level function**을 참조한다.

Stored function은 다음과 같이 사용된다.

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )  
RETURN INTEGER  
IS  
    V1 INTEGER;  
BEGIN  
    SELECT COUNT(*)  
        INTO V1  
        FROM T1  
        WHERE T1.I1 >= A1 AND T1.I1 <= A2;  
    RETURN V1;  
END;  
/  
Function created.  
gSQL> SELECT FUNC1( 2, 4 ) FROM DUAL;  
FUNC1( 2, 4 )  
-----  
                3  
1 row selected.
```


14.

Cluster Objects

14.1 Cluster System

Cluster System 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Cluster system 확장
 - **CREATE CLUSTER GROUP**
 - **ALTER CLUSTER GROUP name ADD MEMBER**
- Inactive cluster member 제어
 - **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**
 - **ALTER SYSTEM JOIN DATABASE**
- 데이터 재배치
 - **ALTER DATABASE REBALANCE**
 - **ALTER TABLE name REBALANCE**

Cluster system과 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-1 Cluster system 관련 정보

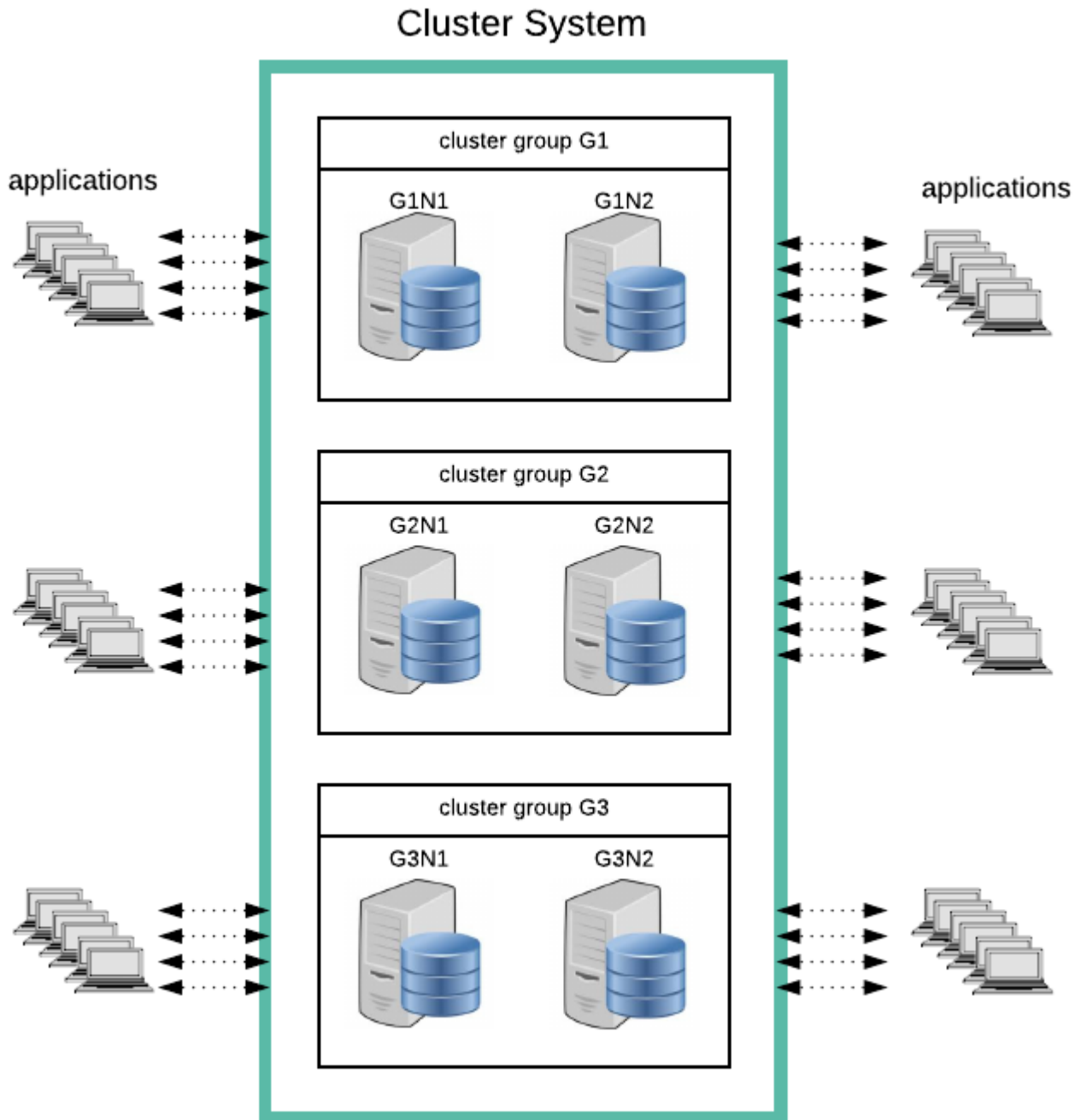
스키마	View 이름	설명
DICTIONARY_SCHEMA	DBA_CLUSTER	Cluster를 구성하는 cluster group, cluster member의 객체 정보
	DBA_CLUSTER_COMMENTS	Cluster group과 cluster member의 주석 정보
PERFORMANCE_VIEW_SCHEMA	V\$CLUSTER_MEMBER	Cluster member의 상태 정보

Cluster System 개념

GOLDILOCKS의 cluster system은 한 개 database의 data를 여러 대의 server에 분할 또는 복제하여 관리한다. Application들은 cluster system을 구성하는 모든 server에서 운영 가능하며, system 구성 방식이나 접속한 server에 관계없이 하나의 database를 사용하는 것과 동일하게 동작한다.

GOLDILOCKS cluster system은 하나 이상의 cluster group으로 구성되며, 하나의 cluster group은 하나 이상의 cluster member로 구성된다. 별도의 application server 또는 meta server를 필요로 하지 않으며, application들은 data server에 해당하는 cluster member에 접속하여 동작한다.

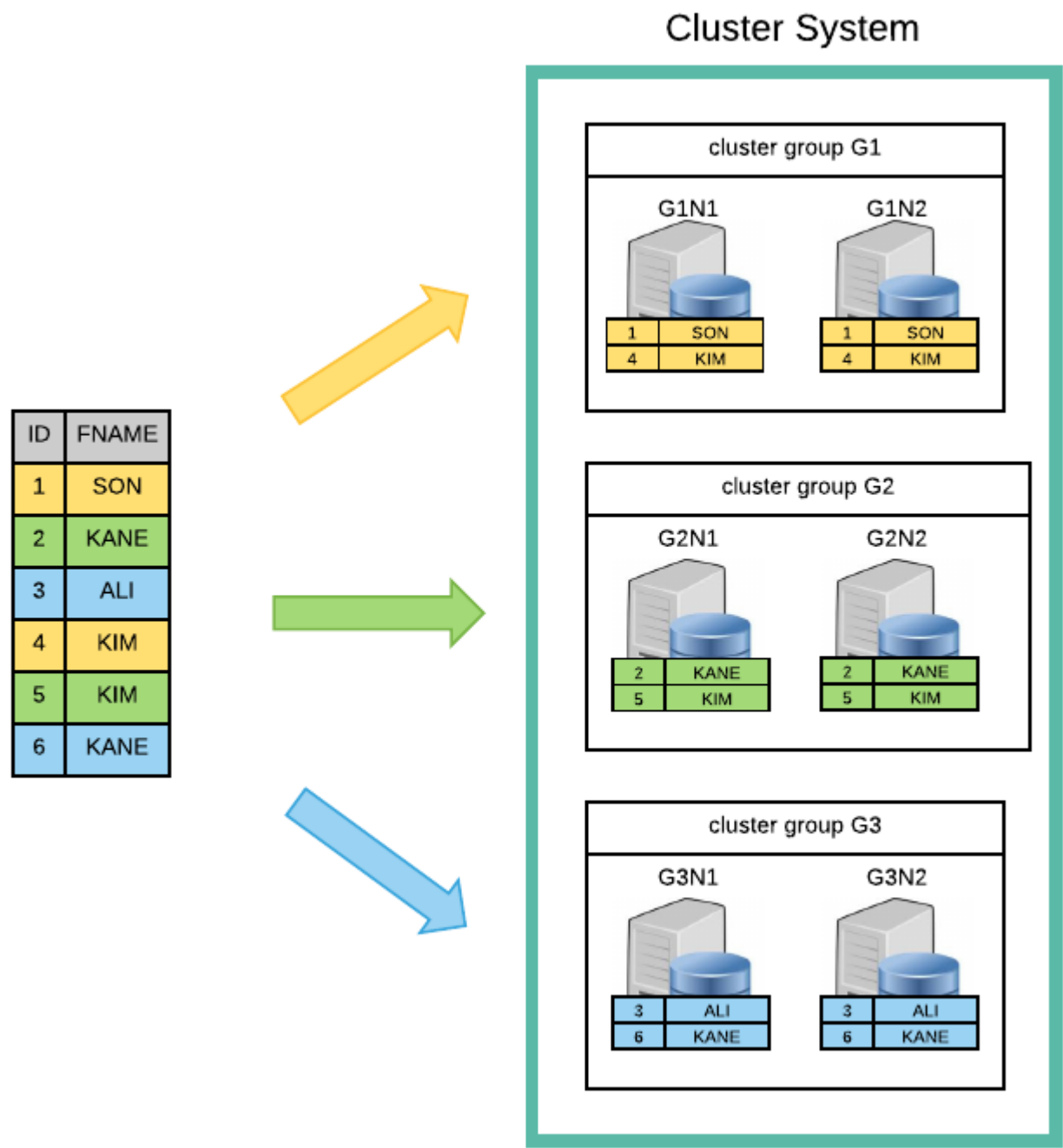
그림 1 3 x 2 cluster system



위의 그림은 세 개의 cluster group과 각 cluster group이 두 개의 cluster member를 구성하는 3 x 2 cluster system이다. 위의 그림에서 cluster system은 G1, G2, G3 cluster group으로 구성되어 있다. G1 cluster group은 G1N1, G1N2 cluster member로 구성되어 있으며, G2 cluster group은 G2N1과 G2N2로 구성되며, G3 cluster group은 G3N1, G3N2 cluster member로 구성되어 있다. Application들은 여섯 개의 cluster member 어디에나 접속할 수 있으며 하나의 database를 사용하는 것과 동일하게 동작한다.

테이블의 data는 각 cluster group에 분할 (sharding)되어 배치되며, cluster group 내의 cluster member 들은 복제본 (replica)을 동일하게 유지한다. 아래 그림은 3 x 2 cluster에 테이블의 data를 배치하는 개념을 표현한다.

그림 2 Cluster의 분할 및 복제 개념



테이블의 데이터는 사용자가 정의한 분할 전략에 따라 (위 그림에서는 ID column을 기준으로) 분할되어 각 cluster group에 배치된다. Cluster group에 배치된 data는 cluster group 내의 cluster member 들에 복제본을 유지한다.

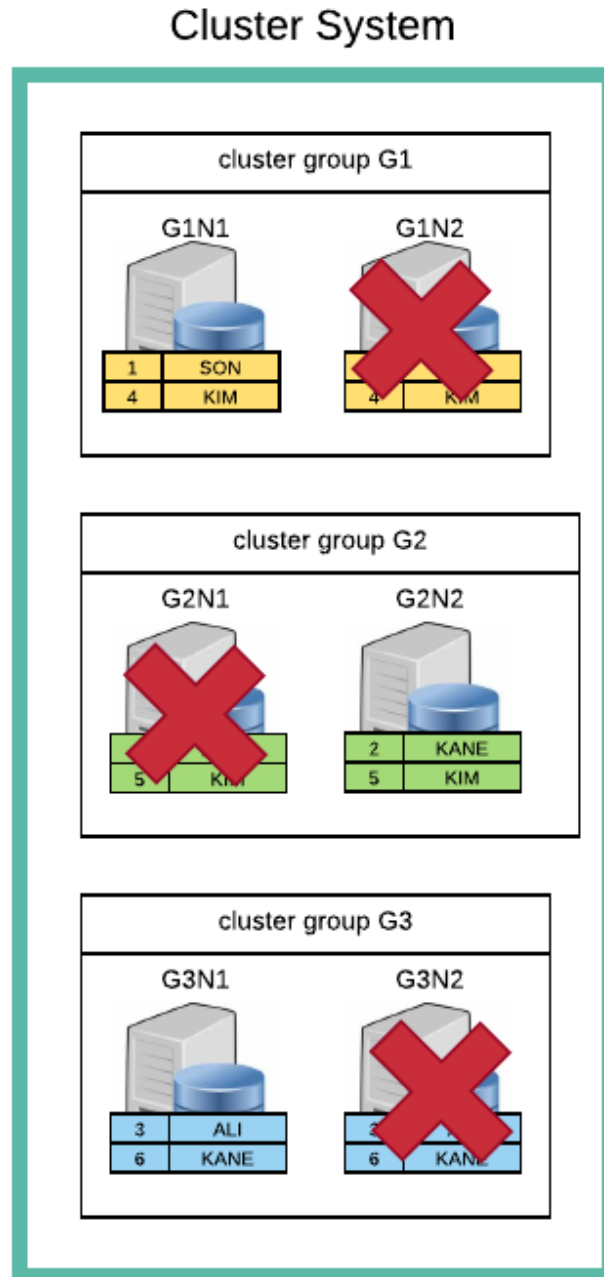
Cluster System의 가용성

Cluster는 특정 server가 고장나거나 네트워크가 단절되어도 계속 서비스할 수 있다. 각 cluster group을 구성하는 cluster member들은 데이터 복제본을 동일하게 유지하기 때문에 cluster member 한 개가 고장나더라도 서비스가 중

단되지 않는다. 즉, cluster group의 모든 cluster member들에 장애가 발생하여 data loss가 발생하지 않는 한 지속적으로 서비스할 수 있다.

아래 그림과 같은 3 x 2 cluster에서는 장비 세 대가 고장나더라도 계속 서비스할 수 있다.

그림 3 Cluster availability



위의 상황에서 G1N1, G2N2, G3N1에 추가적인 장애가 발생할 경우 data loss가 발생하여 서비스를 제공할 수 없으므로, 추가 장애가 발생하기 전에 장애가 발생한 장비를 cluster system에 참여시키거나, 새로운 cluster member를 추가해야 한다.

- **ALTER SYSTEM JOIN DATABASE**는 장애가 발생한 cluster member를 cluster system에 다시 참여시킬 때 사용되는 구문이다.

- **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**는 장애가 발생한 cluster member를 cluster system에서 제거하기 위해 사용되는 구문이다.
- **ALTER CLUSTER GROUP name ADD MEMBER**는 고가용성을 위해 cluster group에 cluster member를 추가할 때 사용되는 구문이다.
- **ALTER DATABASE REBALANCE**와 **ALTER TABLE name REBALANCE**는 생성한 cluster member에 데이터를 재배치하기 위해 사용되는 구문들이다.

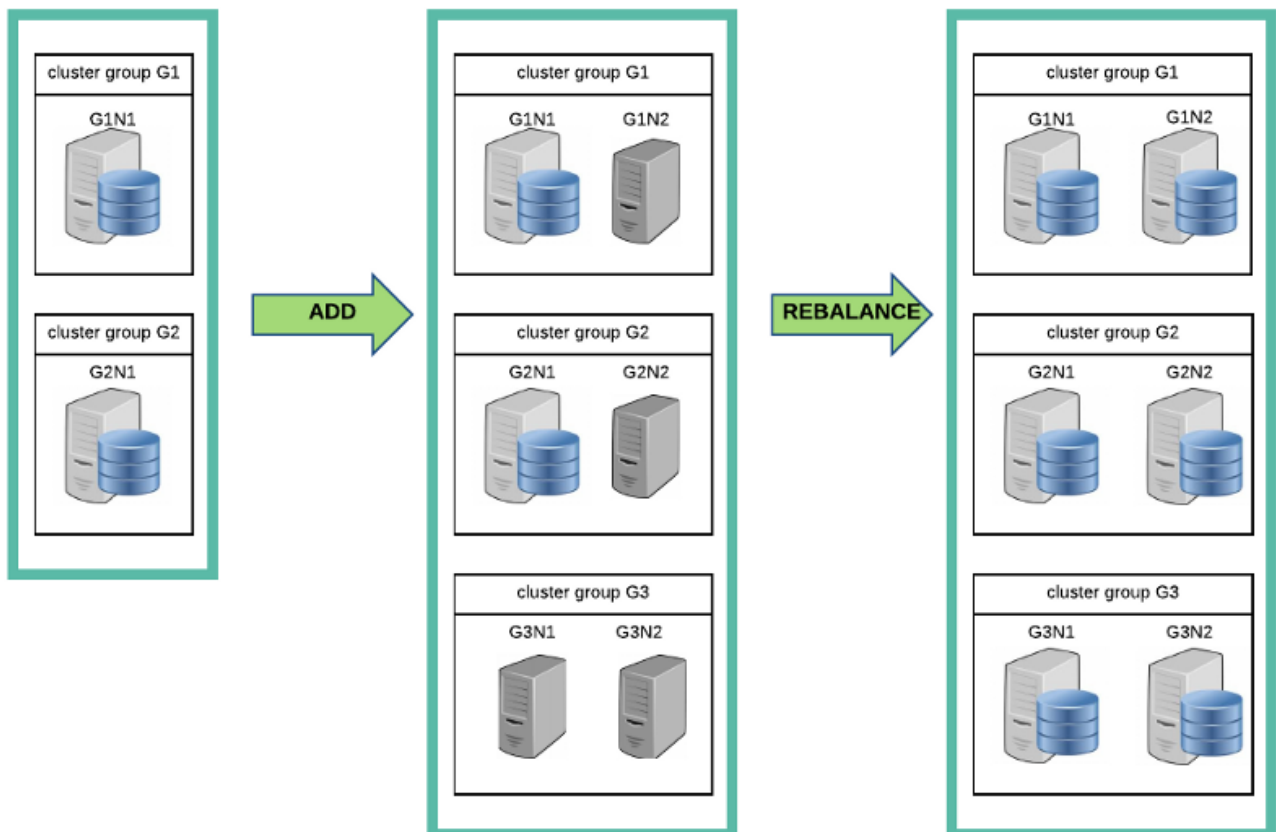
Cluster System 확장

서비스를 중단하지 않고 새로운 server를 추가하여 cluster를 확장할 수 있다.

Cluster는 cluster member 또는 cluster group을 추가하는 작업과 생성된 server에 data를 재분배하는 과정을 통해 확장된다.

다음은 2 x 1 cluster를 3 x 2 cluster로 확장하는 예이다.

그림 4 Cluster system 확장



Cluster를 확장하기 위해 다음 구문을 사용하여 cluster group과 cluster member를 추가한다.

- **ALTER CLUSTER GROUP name ADD MEMBER**

- **CREATE CLUSTER GROUP**

Cluster system에 새로운 cluster member를 추가하려면 cluster member와 cluster system의 tablespace가 동일해야 한다. 즉, cluster member에 cluster system의 모든 tablespace와 동일한 tablespace를 생성해야 한다.

다음은 2 x 1 cluster에 3 x 2 cluster의 cluster group과 cluster member를 추가하는 예이다. G1 group에 G1N2 member를 추가하고, G2 group에 G2N2 member를 추가한다. 그리고, G3N1, G3N2 member를 포함하는 G3 group을 새로 생성한다.

- G1 group에 G1N2 member를 추가한다.

```
gSQL>
ALTER CLUSTER GROUP G1
      ADD CLUSTER MEMBER G1N2 HOST '192.168.0.12' PORT 10120;
Cluster Group altered.
```

- G2 group에 G2N2 member를 추가한다.

```
gSQL>
ALTER CLUSTER GROUP G2
      ADD CLUSTER MEMBER G2N2 HOST '192.168.0.22' PORT 10220;
Cluster Group altered.
```

- G3 group을 생성한다.

```
gSQL>
CREATE CLUSTER GROUP G3
      CLUSTER MEMBER G3N1 HOST '192.168.0.31' PORT 10310,
      CLUSTER MEMBER G3N2 HOST '192.168.0.32' PORT 10320;
Cluster Group created.
```

Cluster system에 새로 추가한 cluster group과 cluster member는 SQL 객체들의 dictionary 정보를 모두 동기화하여 서비스할 수 있다. 그러나 추가된 cluster member에는 아직 데이터가 배치되어 있지 않기 때문에 가용성 증가나 부하 분산 효과는 기대할 수 없다. 이를 위해 생성된 cluster member에 데이터를 재배치해야 한다.

다음 구문을 차례대로 사용하여 데이터를 재배치한다.

- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

다음은 데이터베이스의 모든 테이블의 데이터를 재배치하는 예이다.

```
gSQL> ALTER DATABASE REBALANCE;  
Database altered.
```


14.2 Cluster Group

Cluster Group 관련 구문

Cluster group을 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Cluster group 생성: **CREATE CLUSTER GROUP**
- Cluster group 제거: **DROP CLUSTER GROUP**
- Cluster group 변경: **ALTER CLUSTER GROUP name ADD MEMBER**

Cluster group과 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-2 Cluster group 관련 정보

스키마	View 이름	설명
DICTIONARY_SCHEMA	DBA_CLUSTER	Cluster를 구성하는 cluster group, cluster member의 객체 정보
	DBA_CLUSTER_COMMENTS	Cluster group과 cluster member의 주석 정보

Cluster Group 개념

Cluster system을 운영하기 위해서는 최소 하나 이상의 cluster group을 생성해야 한다.

Cluster Group 생성

최초로 생성하는 cluster group은 자기 자신을 cluster member로 포함해야 한다. Cluster group을 생성하는 구문은 **CREATE CLUSTER GROUP**을 참조한다.

Cluster member는 data server를 의미하는 물리적 개념인 반면에 cluster group은 하나 이상의 cluster member들로 구성된 논리적 개념이다.

Cluster group의 구성 방식에 따라 cluster system의 고가용성 및 부하 분산 효과가 달라진다. Cluster group에 포함된 cluster member의 개수가 많을수록 가용성이 높아지고 cluster group의 개수가 많을수록 data가 분산되어 cluster system의 전체 처리량 (throughput)이 증가한다.

Cluster group 내의 모든 cluster member는 동일한 data를 복제하여 유지하므로 cluster group을 구성하는 모든 cluster member에 장애가 발생하지 않는 한 서비스를 지속할 수 있다. Cluster system의 가용성을 유지하기 위해서 각 cluster group은 두 개 이상의 cluster member로 구성하는 것이 바람직하다.

테이블의 데이터는 분할 정책 (sharding strategy)에 따라 분할되며, 배치 (shard placement) 정책에 따라 서로 다른 cluster group에 저장 관리된다. 서비스 특성에 따른 적절한 테이블 분할 정책과 배치 정책이 시스템의 전체 성능을 좌우한다. 각 트랜잭션과 질의들은 data를 저장하고 있는 cluster group들을 중심으로 처리되므로 참조하는 data들이 동일한 cluster group에 존재할 경우 성능이 향상된다.

Cluster Group 제거

Cluster system에 참여하여 서비스 중인 cluster group은 여러가지 이유로 제거될 수 있다. Cluster group의 제거는 **DROP CLUSTER GROUP**을 참조한다.

Cluster group을 제거하려면 해당 cluster group에 생성된 sharded table의 shard를 모두 다른 cluster group으로 이동시켜야 하는데 cluster-wide, group-specific table에 따라 별도로 이동시켜야 한다.

14.3 Cluster Member

Cluster Member 관련 구문

자세한 내용은 다음 링크를 참조한다.

- Cluster member 추가: **ALTER CLUSTER GROUP name ADD MEMBER**
- Cluster member 제거: **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**
- Cluster member 제어
 - **ALTER SYSTEM JOIN DATABASE**
 - **ALTER CLUSTER GROUP name OFFLINE MEMBER**
 - **ALTER DATABASE RESET LOCAL CLUSTER MEMBER**

Cluster member와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-3 Cluster member 관련 정보

스키마	View 이름	설명
DICTIONARY_SCHEMA	DBA_CLUSTER	Cluster를 구성하는 cluster group, cluster member의 객체 정보
	DBA_CLUSTER_COMMENTS	Cluster group과 cluster member의 주석 정보
PERFORMANCE_VIEW_SCHEMA	V\$CLUSTER_MEMBER	Cluster member의 상태 정보

Cluster Member 개념

Cluster member는 cluster system을 구성하는 하나의 server이며 cluster group 내의 cluster member들과 동일한 replica를 유지한다.

Cluster member는 cluster database의 일부 data를 저장하고 있는 data server이면서 application의 접속과 요청을 처리하는 application server이고 객체의 meta 정보를 복제하여 관리하고 있는 meta server이기도 하다. 즉, GOLDILOCKS cluster는 별도의 data server, application server, meta server들을 필요로 하지 않는다.

동일한 cluster group에 속한 cluster member들은 동일한 data 복제본을 가지고 있다. 따라서 특정 cluster member의 장애가 시스템 전체의 장애를 유발하지 않는다. 시스템의 고가용성을 위해 각 cluster group에 두 개 이상의 cluster member를 포함하는 것이 바람직하며 하나의 cluster group은 최대 32 개의 cluster member로 구성될 수 있다.

새로운 cluster member를 cluster group에 추가하려면 **ALTER CLUSTER GROUP name ADD MEMBER** 구문을 수행한다.

추가한 cluster member는 SQL 객체의 meta 정보를 cluster system과 동일하게 유지하기 때문에 application의 접속과 요청을 처리할 수 있다. 그러나 data는 배치되어 있지 않으므로 cluster member의 추가가 cluster group의 고가용성을 보장하지는 않는다.

Cluster member를 추가한 후에 데이터를 배치하기 위해서 다음 구문을 수행한다.

- 전체 data를 재배포하는 경우: **ALTER DATABASE REBALANCE**
- 일부 테이블만 재배포하는 경우: **ALTER TABLE name REBALANCE**

Cluster member에 장애가 발생해도 서비스 운영에는 지장이 없지만 DDL을 수행할 수 없다. 정상적인 서비스 운영을 위해서는 장애가 발생한 cluster member에 대한 조치를 취해야 한다.

장애가 발생한 cluster member를 cluster system에 다시 참여시키려면 해당 cluster member에 접속하여 LOCAL OPEN 단계로 구동한 후에 **ALTER SYSTEM JOIN DATABASE**를 수행한다.

일부 cluster member를 구동하지 않거나 네트워크 단절 상태에서 cluster system을 구동한 경우에도 해당 cluster member를 LOCAL OPEN 단계로 구동한 후에 **ALTER SYSTEM JOIN DATABASE** 구문을 수행한다.

만약 장애가 발생한 cluster member의 장비를 복구할 수 없는 경우에는 cluster system에서 **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS** 구문을 수행하여 장애가 발생한 cluster member들을 제거한다.

Cluster system에서 제거된 cluster member는 여전히 제거되기 이전의 정보를 가지고 있으며 다시 cluster system에 참여할 수 없다. Cluster member를 cluster system에 참여하기 이전의 상태로 초기화하기 위해서는 **ALTER DATABASE RESET LOCAL CLUSTER MEMBER** 구문을 수행한다.

위 구문은 tablespace의 정보를 그대로 유지하고 있기 때문에 cluster system에 새로운 cluster member로 추가될 때 tablespace 생성 시간을 단축할 수 있다는 점이 cluster member의 database를 새로 생성한 것과의 차이점이다.

14.4 Cluster Location

Cluster Location 관련 구문

Cluster location을 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Cluster location 생성: **CREATE CLUSTER LOCATION**
- Cluster location 제거: **DROP CLUSTER LOCATION**
- Cluster location 변경: **ALTER CLUSTER LOCATION**

Cluster location과 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-4 Cluster location 관련 정보

스키마	View 이름	설명
PERFORMANCE_VIEW_SCHEMA	V\$CLUSTER_LOCATION	Cluster location 정보

Cluster Location 개념

Cluster location은 cluster system에 등록된 각 cluster member들의 내부 cluster network들을 연결하기 위한 접속 정보이다. 각각의 cluster member들은 transaction의 처리 및 관리 정보 교환과 같은 다양한 protocol의 상호 송수신을 위해 cluster 전용 tcp network를 사용하는데, 이 때 접속하기 위해 사용하는 member name, host ip address, port를 묶어서 cluster location이라고 한다.

각 member 별로 고유한 cluster location 정보가 지정되어야 하며 만약 정보가 중복될 경우 cluster network 접속에 실패하여 cluster system이 정상적으로 작동하지 않는다.

Cluster location 정보는 cluster member를 추가하거나 제거할 때 자동으로 추가되거나 삭제되기 때문에 사용자가 직접적으로 location 정보 자체를 추가하거나 삭제해야 할 경우는 거의 없다. 다만 다음과 같은 경우에는 cluster location 관련 DDL 구문을 사용할 수 있다.

- Location control file이 삭제되어 cluster location 정보가 유실된 경우: **CREATE CLUSTER LOCATION**
- 이미 등록된 cluster member의 접속 정보가 변경된 경우, 즉 hardware가 변경되거나 접속 ip, port가 변경된 경우: **ALTER CLUSTER LOCATION**

14.5 Cluster Table과 Shard

Shard 관련 구문

Shard 정의와 재배포를 위한 구문은 다음과 같다.

- Shard 정의: **CREATE TABLE** 구문의 <table sharding strategy> 절
- Shard 재배포: **ALTER TABLE name REBALANCE**

Cluster table의 shard 관련 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-5 Cluster table과 shard 관련 정보

스키마	View 이름	설명
DICTIONARY_SCHEMA	ALL_CLUSTER_TABLES	사용자가 접근 가능한 cluster table 정보
	ALL_SHARD_KEY_COLUMNS	사용자가 접근 가능한 cluster table의 shard key column 정보
	ALL_TAB_PLACE	사용자가 접근 가능한 cluster table의 배치 정보
	ALL_TAB_SHARDS	사용자가 접근 가능한 cluster table의 shard 정보
	USER_CLUSTER_TABLES	사용자가 소유한 cluster table 정보
	USER_SHARD_KEY_COLUMNS	사용자가 소유한 cluster table의 shard key column 정보
	USER_TAB_PLACE	사용자가 소유한 cluster table의 배치 정보
	USER_TAB_SHARDS	사용자가 소유한 cluster table의 shard 정보

Cluster Table 유형

Cluster 환경에서 사용자가 생성하는 table은 다음 두 가지 중 하나이다.

- Cloned table: 테이블의 data를 동일하게 복제하여 관리한다.
- Sharded table: 테이블의 data를 수평으로 분할하여 관리한다.

Cloned table은 테이블 data를 모두 복제하여 관리하므로, 제품 목록, 공급자 목록 등과 같이 데이터의 갱신이 적고 데이터 양이 상대적으로 적은 테이블에 적합하다. Cloned table에 데이터를 추가, 삭제, 갱신할 경우 cloned table이 배치된 모든 cluster member에 동일하게 반영된다.

Sharded table은 거래 내역, 통화 내역 등과 같이 테이블의 데이터 양이 많아 분할이 필요한 경우에 적합하며, 다음과 같이 세 가지 분할 정책에 따라 분류된다.

- Hash sharded table

- 테이블 data를 sharding key의 해시 (hash)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.
- Range sharded table
 - 테이블 data를 sharding key의 범위 (range)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.
- List sharded table
 - 테이블의 data를 sharding key의 나열 (list)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.

Sharded table은 row들을 수평으로 분할하여 shard 단위로 관리하며, sharding 정책은 **CREATE TABLE** 구문의 **SHARDING BY** 절을 이용해 정의한다. Sharding 정책에 의해 분류된 row들의 집합을 shard라고 한다.

각 shard는 사용자가 정의한 배치 정책에 따라 cluster group에 배치되는데 **CREATE TABLE** 구문의 **AT CLUSTER WIDE**를 통해 자동으로 배치하거나, **AT CLUSTER GROUP** 절을 이용해 배치할 cluster group을 지정할 수도 있다. **AT CLUSTER WIDE**를 사용해 자동으로 배치할 경우, **CREATE CLUSTER GROUP** 구문을 사용하여 cluster group을 생성한 후에 **ALTER TABLE name REBALANCE** 구문을 수행하는 반면에, **AT CLUSTER GROUP**을 사용해 shard가 배치될 cluster group을 지정하는 경우 새롭게 추가된 cluster group에 shard가 배치되지 않는다.

다음은 3 x 2 cluster 환경에서 cluster table의 유형에 따라 테이블을 생성하고 data를 배치하는 개념을 설명하기 위한 예이다.

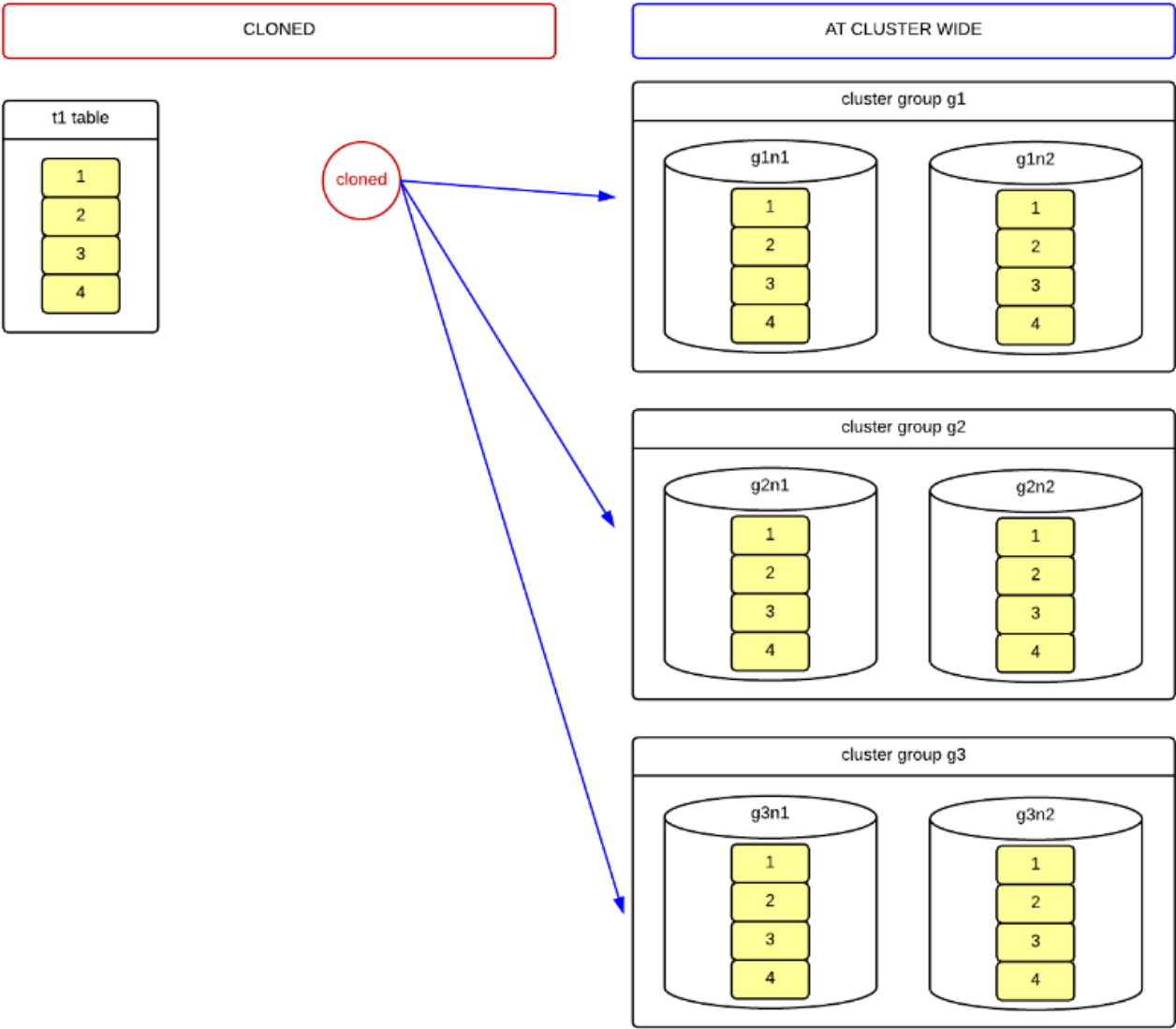
Cloned Table

Cloned table은 테이블의 모든 data를 동일하게 복제하여 관리한다.

다음은 cluster-wide cloned table을 생성하는 예이다. 모든 테이블 데이터를 3 x 2로 구성된 cluster member에 동일하게 복제하여 배치한다.

```
CREATE TABLE t1 ( id INTEGER )
  CLONED
  AT CLUSTER WIDE
;
```

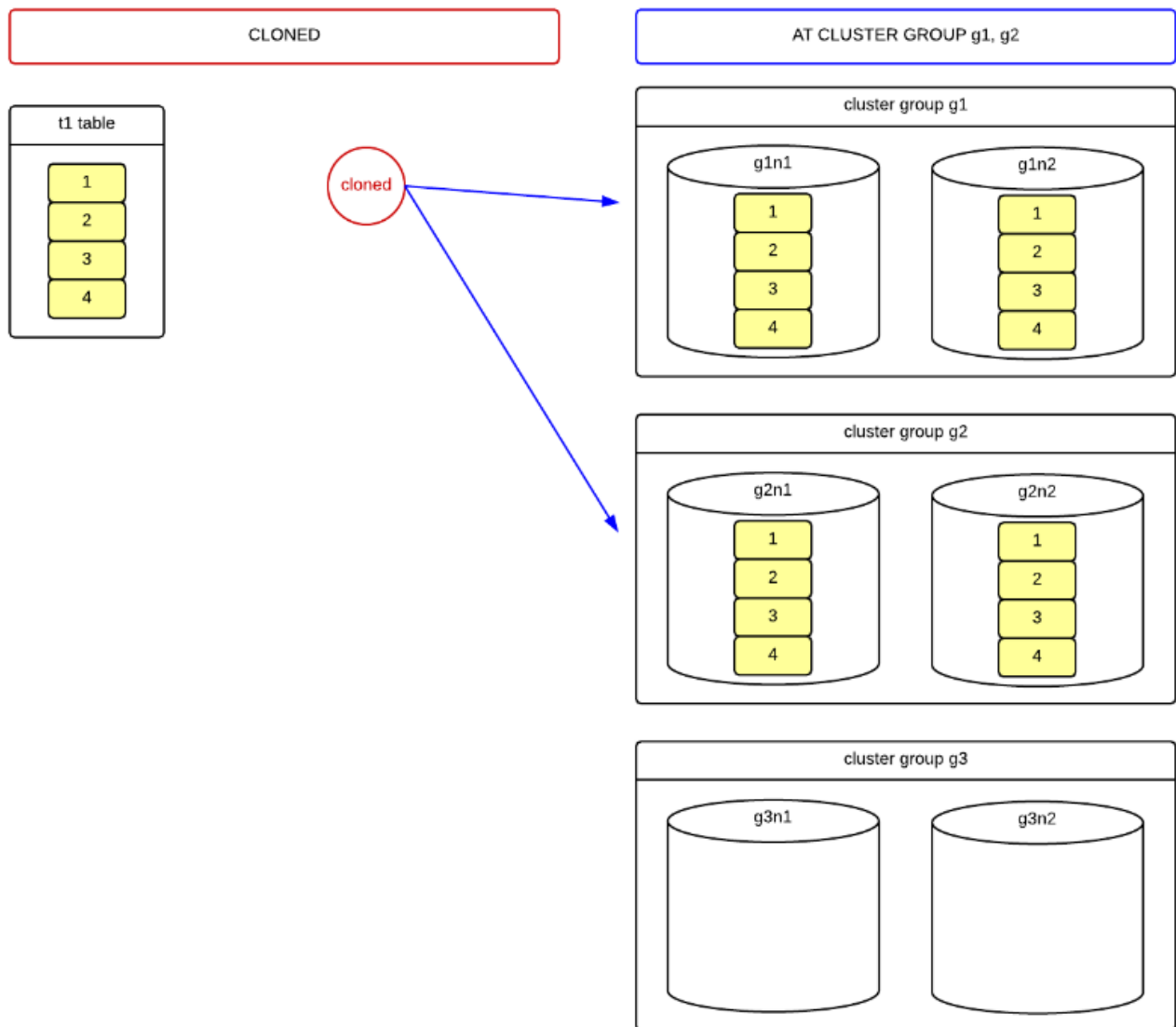
그림 5 Cluster-wide cloned table



다음은 group-specific cloned table을 생성하는 예이다. 테이블의 모든 data가 복제되어 관리되기는 하지만 복제된 테이블 data는 사용자가 지정한 g1과 g2 group의 cluster member에만 존재하고 g3 group에는 data가 존재하지 않는다.

```
CREATE TABLE t1 ( id INTEGER )  
  CLONED  
  AT CLUSTER GROUP g1, g2  
;
```


그림 6 Group-specific cloned table



Hash-sharded Table

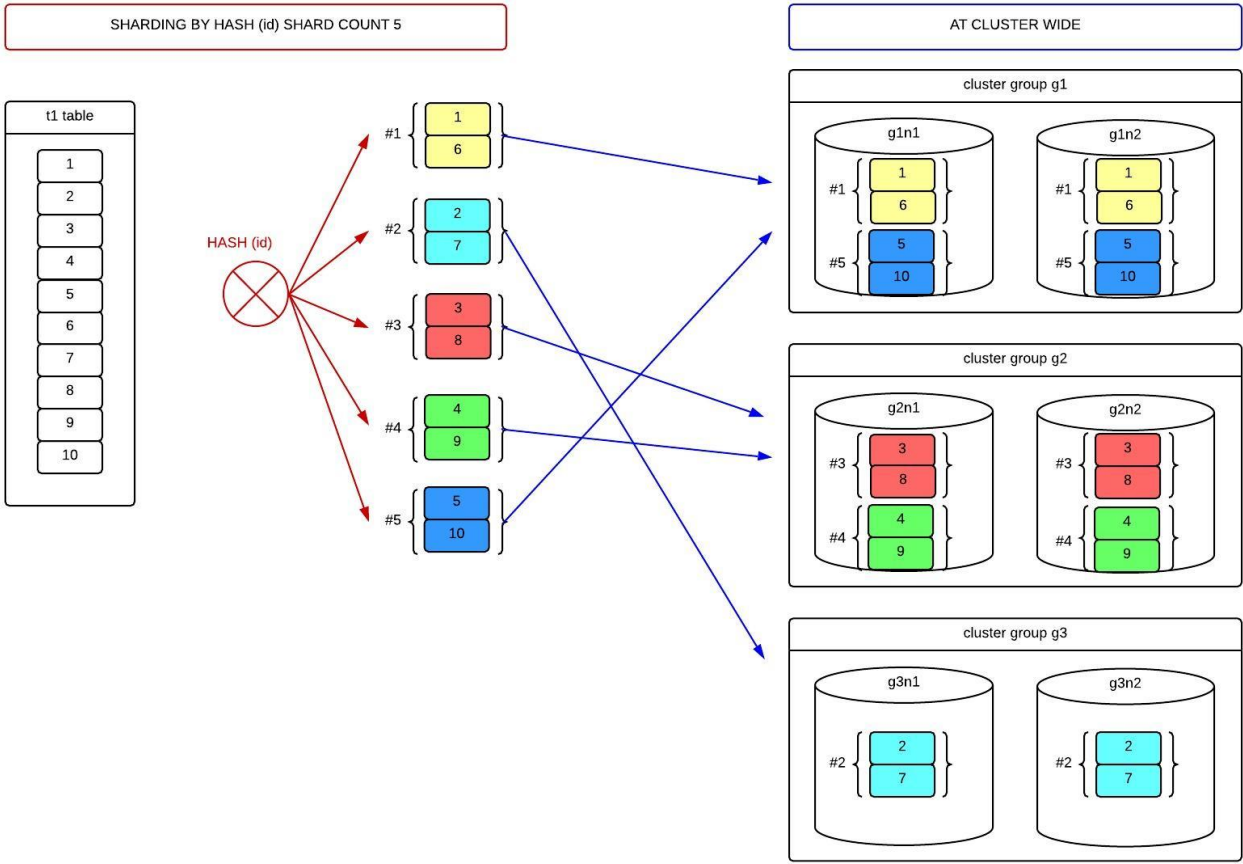
Hash-sharded table은 data를 sharding key의 해시 (hash)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.

다음은 cluster-wide hash-sharded table을 생성하는 예이다. 테이블에 data를 추가할 때 ID column값을 기준으로 해시값을 생성하고 이를 이용해 다섯 개의 shard 중 row를 배치할 shard를 결정한다. 각 shard는 자동으로 배치된다. ID column 값이 동일한 모든 row들은 동일한 shard에 포함되며 동일한 cluster group에 배치된다.

```
CREATE TABLE t1 ( id INTEGER )
  SHARDING BY HASH(id)
  SHARD COUNT 5
  AT CLUSTER WIDE
```

;

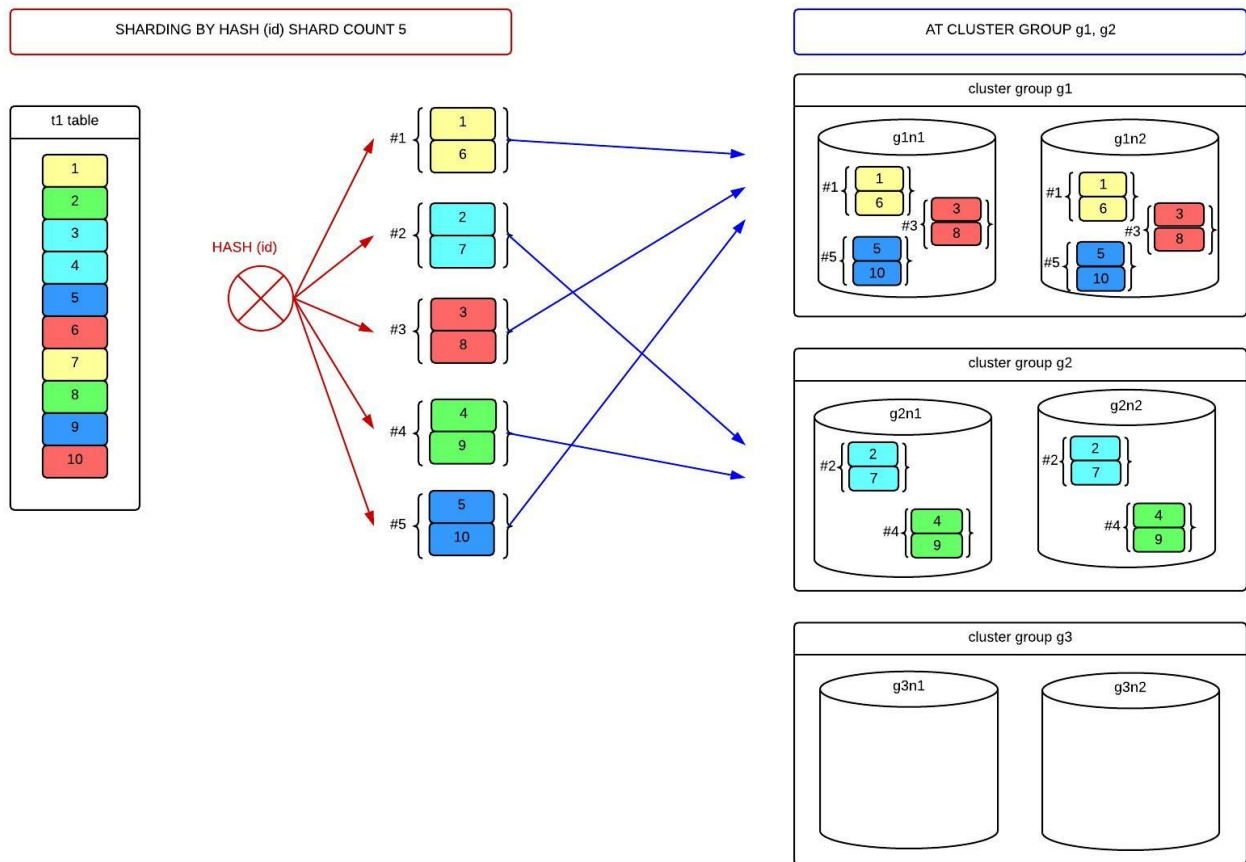
그림 7 Cluster-wide hash-sharded table



다음은 group-specific hash-sharded table을 생성하는 예이다. ID column의 해시값이 shard를 결정하지만 각 shard는 사용자가 지정한 g1, g2 cluster group에만 배치된다.

```
CREATE TABLE t1 ( id INTEGER )  
  SHARDING BY HASH(id)  
  SHARD COUNT 5  
  AT CLUSTER GROUP g1, g2  
;
```

그림 8 Group-specific hash-sharded table



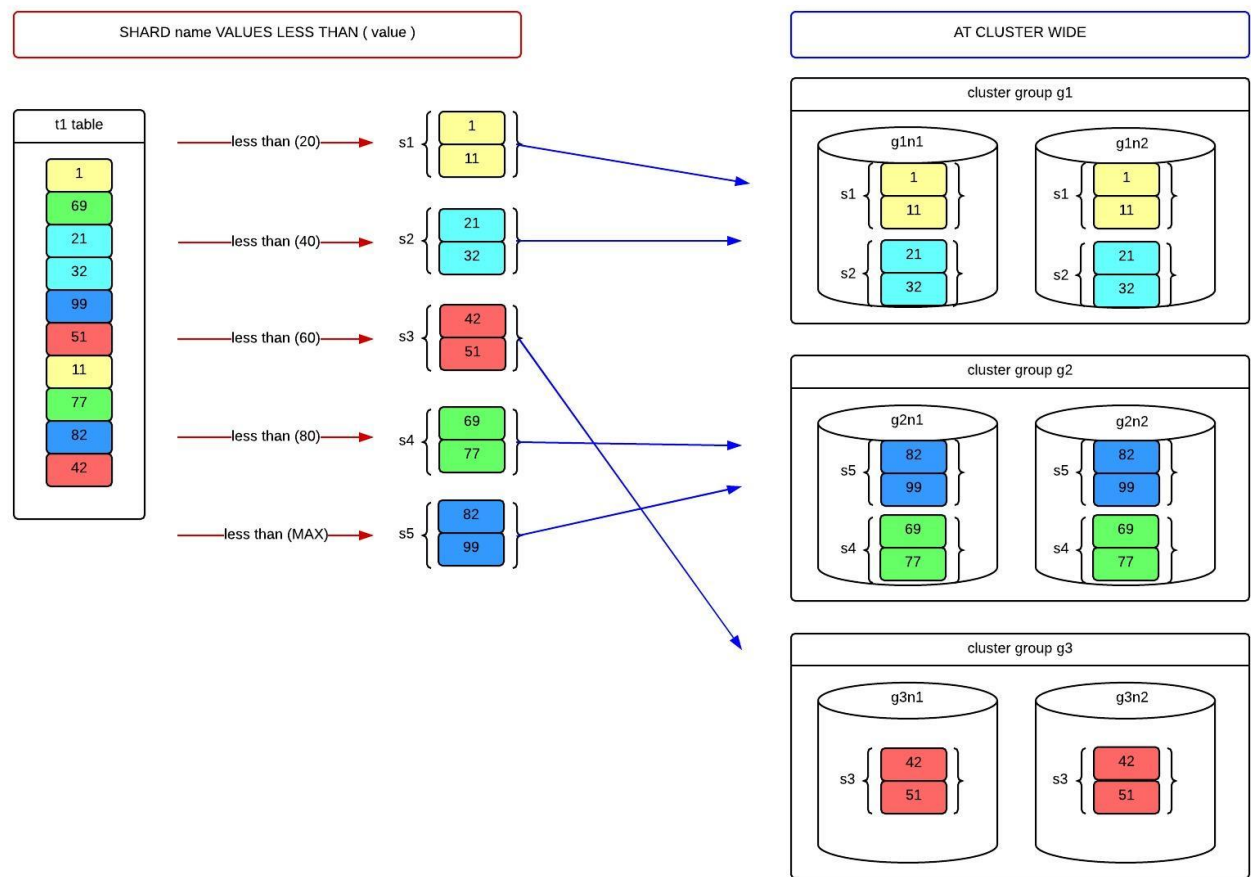
Range-sharded Table

Range-sharded table은 data를 sharding key의 범위 (range)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.

다음은 cluster-wide range-sharded table을 생성하는 예이다. 테이블에 data를 추가할 때 ID column의 범위값을 기준으로 다섯 개의 shard 중에 row를 배치할 shard를 결정한다. 각 shard는 자동으로 배치된다. 동일한 범위 내에 있는 ID column의 모든 row들은 동일한 shard에 포함되며 동일한 cluster group에 배치된다.

```
CREATE TABLE t1 ( id INTEGER )
  SHARDING BY RANGE(id)
  AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 20 ),
    SHARD s2 VALUES LESS THAN ( 40 ),
    SHARD s3 VALUES LESS THAN ( 60 ),
    SHARD s4 VALUES LESS THAN ( 80 ),
    SHARD s5 VALUES LESS THAN ( MAXVALUE )
;
```

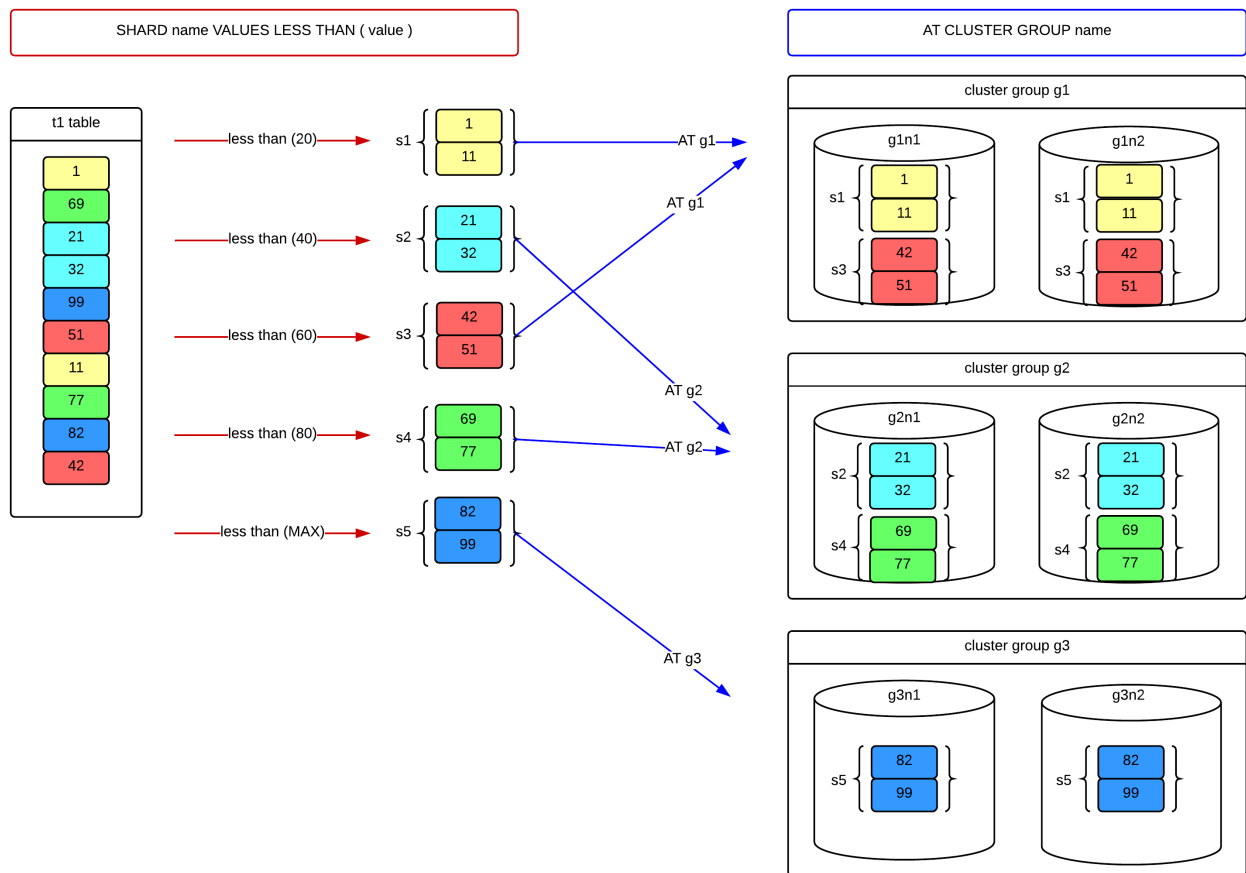
그림 9 Cluster-wide range-sharded table



다음은 group-specific range-sharded table을 생성하는 예이다. ID column의 범위값이 shard를 결정하지만 각 shard는 사용자가 지정한 cluster group에 배치된다.

```
CREATE TABLE t1 ( id INTEGER )
SHARDING BY RANGE(id)
  SHARD s1 VALUES LESS THAN ( 20 )      AT CLUSTER GROUP g1,
  SHARD s2 VALUES LESS THAN ( 40 )      AT CLUSTER GROUP g2,
  SHARD s3 VALUES LESS THAN ( 60 )      AT CLUSTER GROUP g1,
  SHARD s4 VALUES LESS THAN ( 80 )      AT CLUSTER GROUP g2,
  SHARD s5 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g3
;
```

그림 10 Group-specific range-sharded table



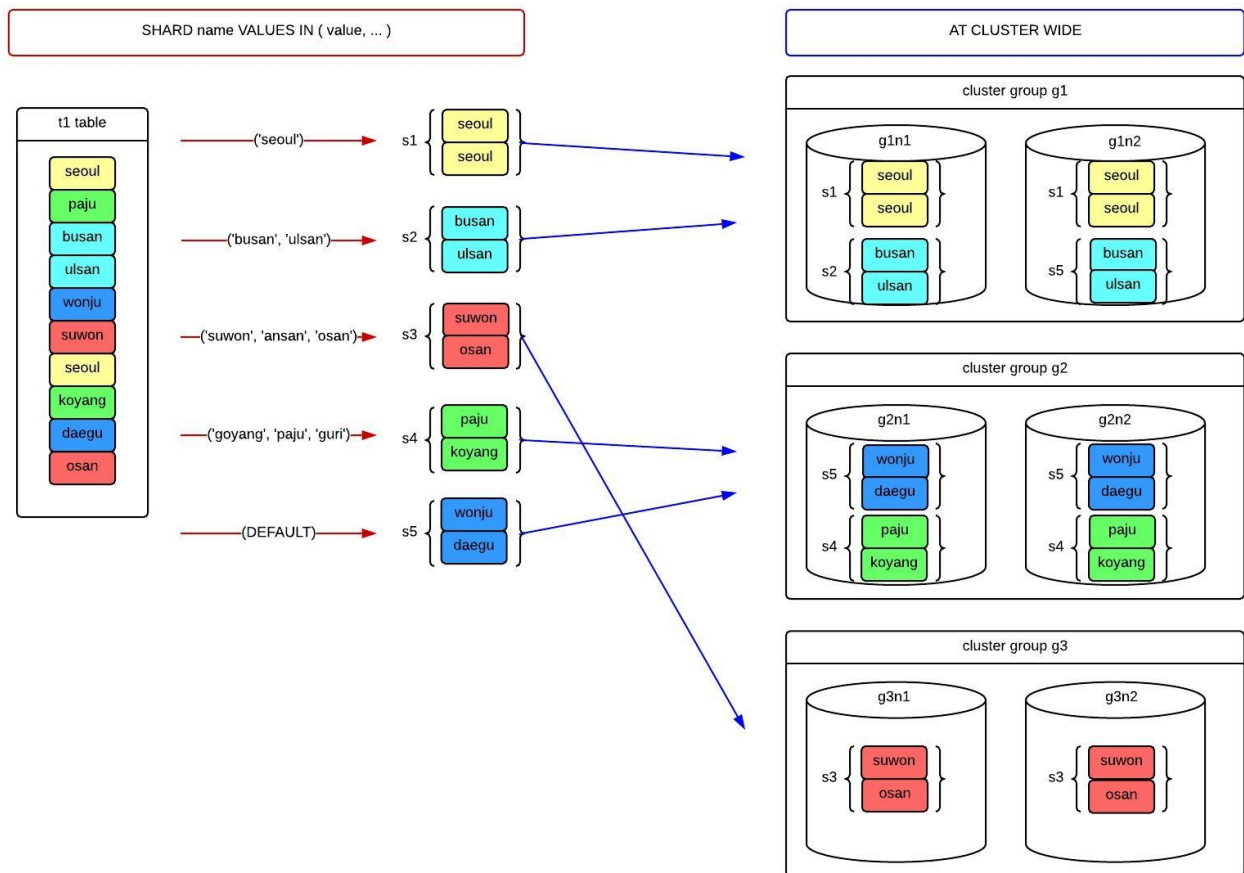
List-sharded Table

List-sharded table은 data를 sharding key의 나열 (list)값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.

다음은 cluster-wide list-sharded table을 생성하는 예이다. 테이블에 data를 추가할 때 CITY column의 값과 동일한 나열값을 가지는 shard에 row를 배치한다. 각 shard는 자동으로 배치된다.

```
CREATE TABLE t1 ( city VARCHAR(128) )
  SHARDING BY LIST (city)
  AT CLUSTER WIDE
  SHARD s1 VALUES IN ( 'seoul' ),
  SHARD s2 VALUES IN ( 'busan', 'ulsan' ),
  SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ),
  SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ),
  SHARD s5 VALUES IN ( DEFAULT )
;
```

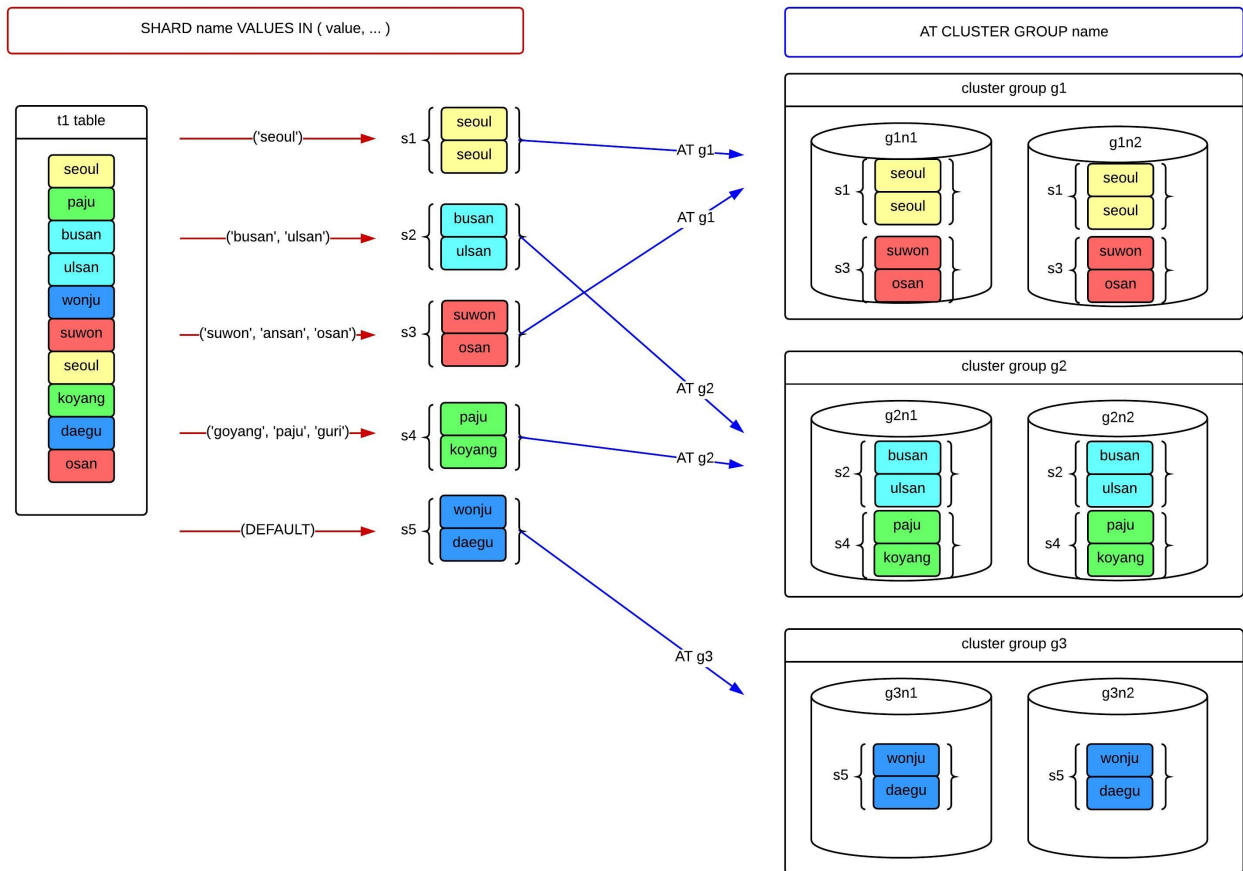
그림 11 Cluster-wide list-sharded table



다음은 group-specific list-sharded table을 생성하는 예이다. CITY column의 나열값이 shard를 결정하지만 각 shard는 사용자가 지정한 cluster group에 배치된다.

```
CREATE TABLE t1 ( city VARCHAR(128) )
SHARDING BY LIST (city)
  SHARD s1 VALUES IN ( 'seoul' )          AT CLUSTER GROUP g1,
  SHARD s2 VALUES IN ( 'busan', 'ulsan' )  AT CLUSTER GROUP g2,
  SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ) AT CLUSTER GROUP g1,
  SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ) AT CLUSTER GROUP g2,
  SHARD s5 VALUES IN ( DEFAULT )          AT CLUSTER GROUP g3
;
```

그림 12 Group-specific list-sharded table



Cluster Table 재배포

`ALTER TABLE name REBALANCE` 구문을 사용하여 cluster table의 데이터를 재배포한다.

Cluster table의 데이터는 다음과 같은 단위로 재배포된다.

- Cloned table: table 전체
- Sharded table: shard 단위

AT CLUSTER WIDE로 지정된 테이블인 경우, 생성된 cluster group에 shard가 자동으로 재배포되는 반면에 AT CLUSTER GROUP을 통해 shard가 배치될 cluster group을 지정한 경우에는 생성된 cluster group에 shard가 재배포되지 않는다.

- AT CLUSTER WIDE를 사용하여 자동으로 재배포할 수 있는 경우 새로운 cluster group, cluster member 에 데이터를 재배포할 수 있다.

```
CREATE TABLE region
```

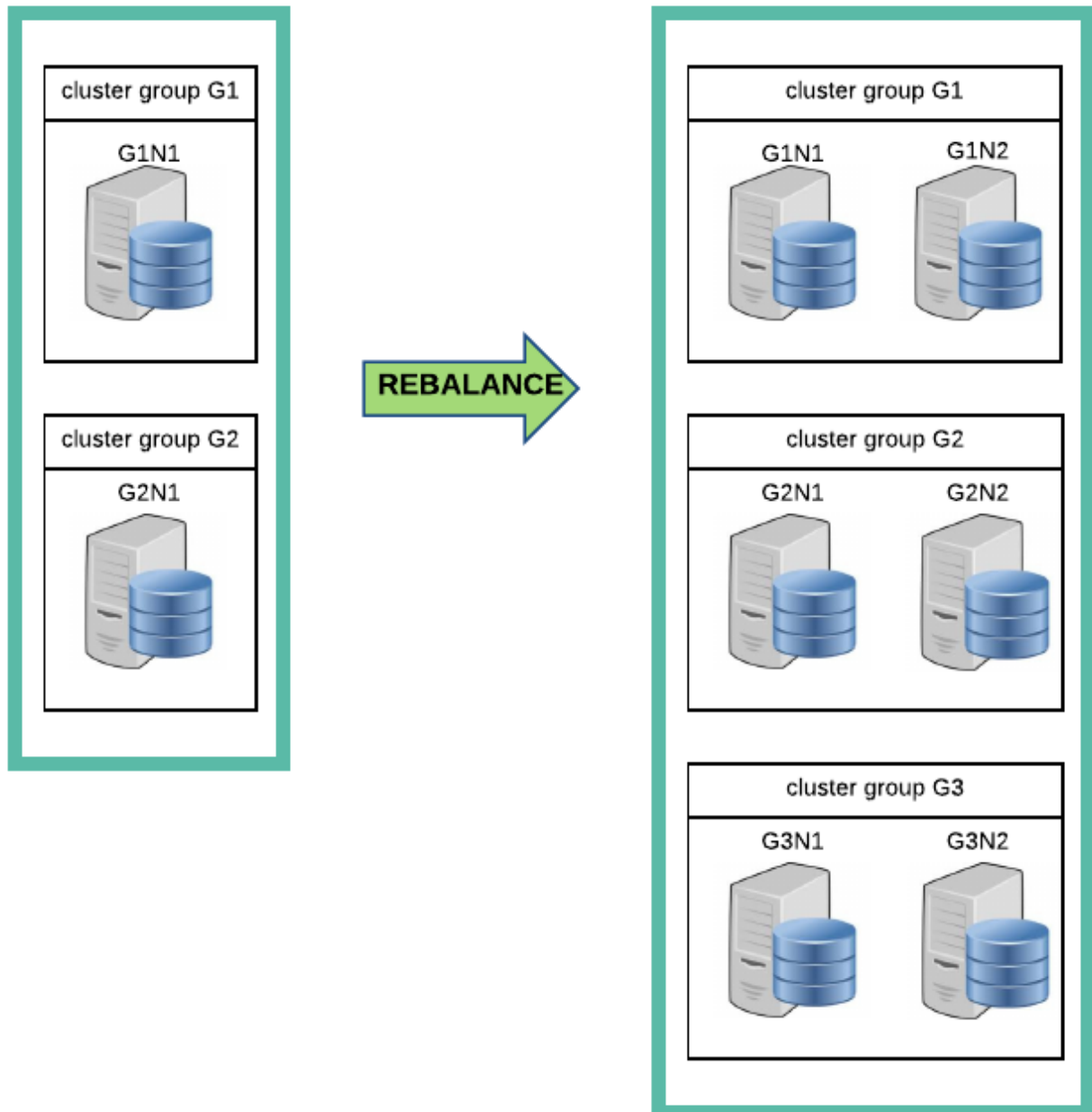
```
(
```

```

    r_regionkey    INTEGER
  , r_name        CHAR(25)
  , r_comment     VARCHAR(152)
)
CLONED
AT CLUSTER WIDE;

```

그림 13 AT CLUSTER WIDE로 정의한 region 테이블의 재배포



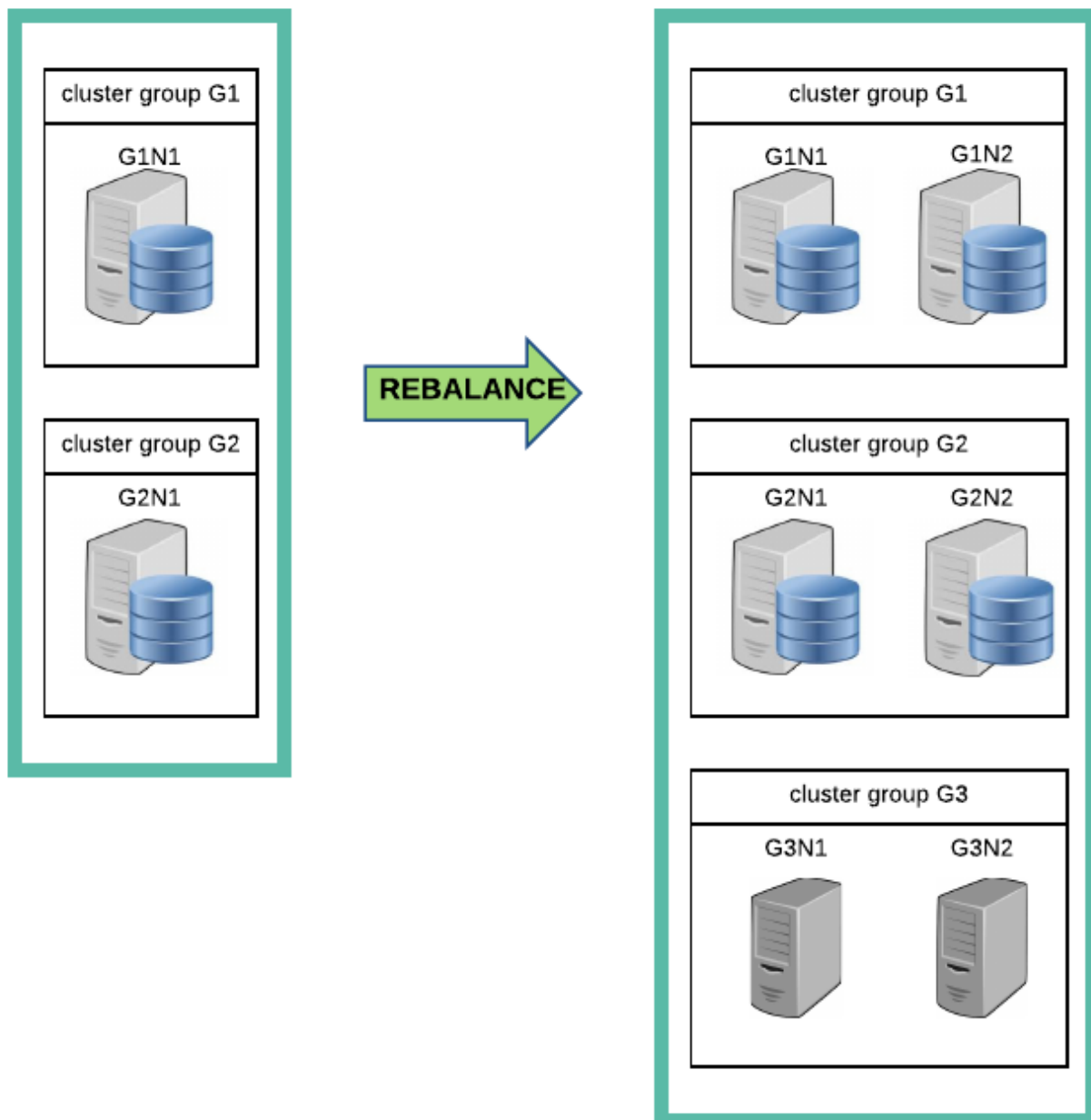
- AT CLUSTER GROUP을 사용하여 shard를 배치할 위치를 지정한 경우
 - 새로운 cluster group에는 데이터가 재배포 되지 않는다.
 - 지정한 cluster group에 추가된 새로운 cluster member에 데이터를 재배포할 수 있다.


```

CREATE TABLE nation
(
    n_nationkey    INTEGER
  , n_name        CHAR(25)
  , n_regionkey   INTEGER
  , n_comment     VARCHAR(152)
)
CLONED
AT CLUSTER GROUP g1, g2;

```

그림 14 AT CLUSTER GROUP으로 정의한 nation 테이블 재배포



테이블 배치 정보는 다음과 같은 view를 통해 조회할 수 있다.

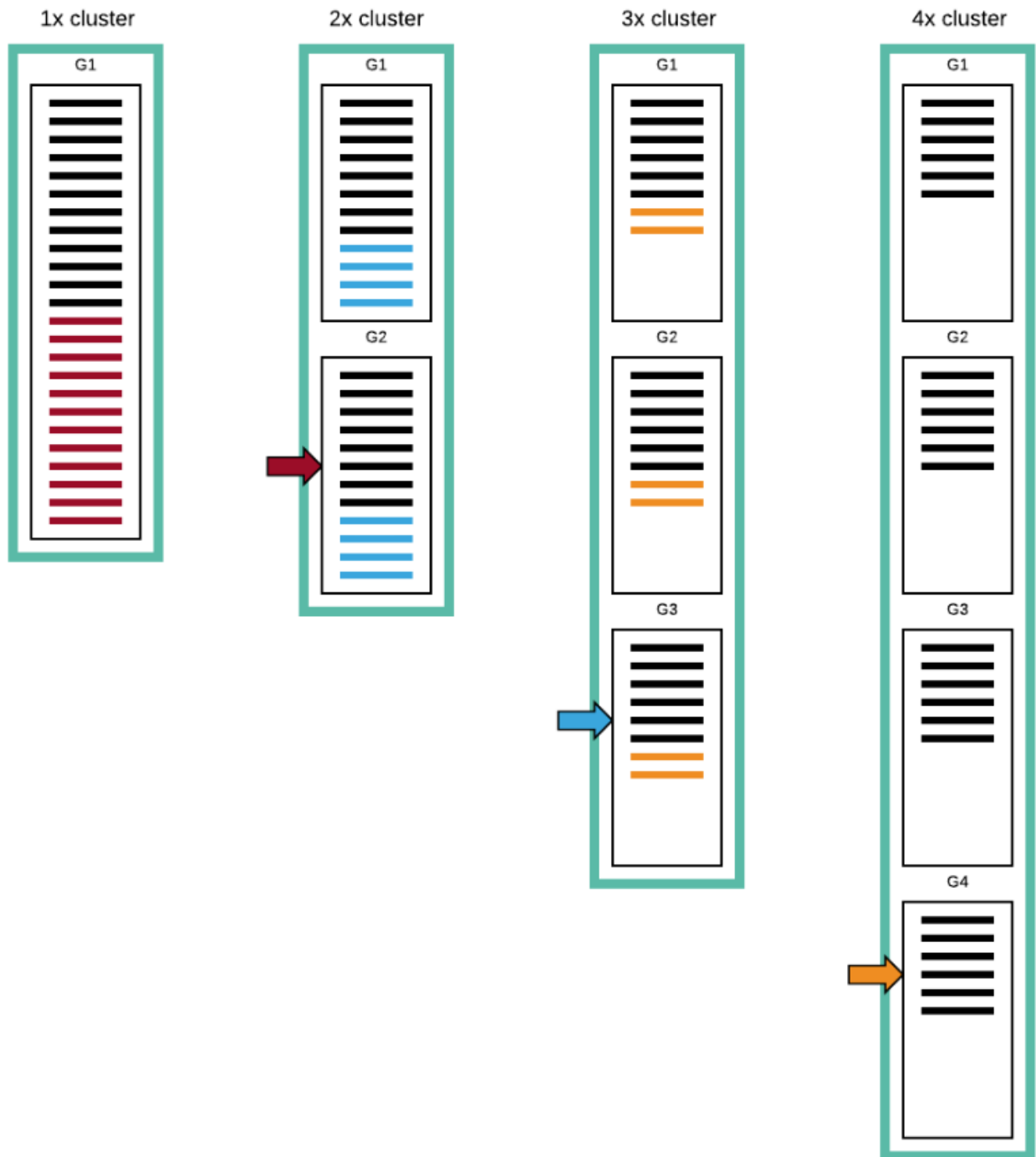
- **USER_TAB_PLACE**
- **ALL_TAB_PLACE**

```
gSQL>
SELECT group_name, member_name
FROM user_tab_place
WHERE table_name = 'REGION';
GROUP_NAME MEMBER_NAME
-----
G1          G1N1
G1          G1N2
G2          G2N1
G2          G2N2
G3          G3N1
G3          G3N2
6 rows selected.
```

Sharded table은 shard 단위로 재배포되며 group 증가에 따른 shard 재배포 개념은 다음과 같다

```
CREATE TABLE orders
(
    o_orderkey    INTEGER
  , o_custkey    INTEGER
  , o_orderstatus CHAR(1)
  , o_totalprice  NUMERIC(12,2)
  , o_orderdate   DATE
  , o_orderpriority CHAR(15)
  , o_clerk       CHAR(15)
  , o_shippriority INTEGER
  , o_comment     VARCHAR(79)
)
SHARDING BY HASH( o_orderkey )
SHARD COUNT 24
AT CLUSTER WIDE
;
```

그림 15 Cluster group 증가에 따른 shard 재배치



위의 예에서 orders 테이블의 data는 24 개의 shard로 분할되어 배치된다. Group이 한 개인 1x cluster에는 모든 shard가 하나의 group에 배치되고 group이 두 개인 2x cluster에는 각 group에 12 개의 shard들이 배치된다.

2x cluster에서 3x cluster로 확장한 경우, 기존 group에서 새로운 group으로 shard들이 이동하지만 각 group이 갖는 shard의 개수는 여덟 개로 동일하다. 3x cluster에서 4x cluster로 확장한 경우, G1, G2, G3 에서 각각 두 개의 shard들을 새로 생성된 G4 group으로 재배치한다.

즉, shard 재배치는 기존 group에서 새로운 group으로 shard가 이동할 때 그 움직임을 최소화하며, 각 group이 갖는 shard의 개수를 거의 동일하게 유지하면서 데이터를 골고루 배치한다.

Sharded table의 shard 배치 정보는 다음 예에서처럼 view들을 사용하여 조회할 수 있다.

- **USER_TAB_SHARDS**
- **ALL_TAB_SHARDS**

```
gSQL>
SELECT shard_name, group_name
   FROM user_tab_shards
  WHERE table_name = 'ORDERS';
SHARD_NAME    GROUP_NAME
-----
SHARD_000000  G1
SHARD_000001  G1
SHARD_000002  G1
SHARD_000003  G1
SHARD_000004  G1
SHARD_000005  G1
SHARD_000006  G1
SHARD_000007  G1
SHARD_000008  G3
SHARD_000009  G3
SHARD_000010  G3
SHARD_000011  G3
SHARD_000012  G2
SHARD_000013  G2
SHARD_000014  G2
SHARD_000015  G2
SHARD_000016  G2
SHARD_000017  G2
SHARD_000018  G2
SHARD_000019  G2
SHARD_000020  G3
SHARD_000021  G3
SHARD_000022  G3
SHARD_000023  G3
24 rows selected.
```

다음과 같이 동일한 <sharding strategy>를 가진 테이블들의 재배포가 완료되면 shard의 배치 결과는 동일하다. 서로 다른 테이블이라 하더라도 동일한 shard key를 갖는 row들은 동일한 shard와 동일한 group에 배치된다는 것이 보장된다.

- Table t1

- 2x cluster에 생성된다.
- CREATE TABLE t1 (c1 INTEGER) SHARDING BY (c1);
- 4x cluster에 재배치된다.
- ALTER TABLE t1 REBALANCE:
- Table t2
 - 3x cluster에 생성된다.
 - CREATE TABLE t2 (a1 INTEGER) SHARDING BY (a1);
 - 4x cluster에 재배치된다.
 - ALTER TABLE t2 REBALANCE:
- Table t3
 - 4x cluster에 생성된다.
 - CREATE TABLE t3 (i1 INTEGER) SHARDING BY (i1);

즉, 다음과 같은 query는 하나의 cluster member에만 접근하여 처리할 수 있다.

```
SELECT COUNT(*)
  FROM t1, t2, t3
 WHERE t1.c1 = t2.a1
       AND t2.a1 = t3.i1
       AND t1.c1 = 1;
```

14.6 Global Secondary Index

Global Secondary Index 관련 구문

Global secondary index를 생성, 제거, 변경하기 위한 구문은 다음과 같다.

- Global secondary index 생성: **ALTER TABLE name ADD GLOBAL SECONDARY INDEX**
- Global secondary index 제거: **ALTER TABLE name DROP GLOBAL SECONDARY INDEX**
- Global secondary index 변경: **ALTER TABLE name ALTER GLOBAL SECONDARY INDEX**

Global secondary index와 관련된 정보는 다음과 같은 view를 통해 조회할 수 있다.

표 14-6 Global secondary index 관련 정보

스키마	View 이름	설명
DICTIONARY_SCHEMA	ALL_CLUSTER_TABLES	사용자가 접근 가능한 테이블의 global secondary index 존재 여부
	ALL_GLOBAL_SECONDARY_INDEXES	사용자가 접근 가능한 global secondary index 객체 정보
	ALL_GSI_PLACE	사용자가 접근 가능한 global secondary index의 배치 정보
	USER_CLUSTER_TABLES	사용자가 소유한 테이블의 global secondary index 존재 여부
	USER_GLOBAL_SECONDARY_INDEXES	사용자가 소유한 global secondary index 객체 정보
	USER_GSI_PLACE	사용자가 소유한 global secondary index의 배치 정보

Global Secondary Index 개념

Global secondary index는 cluster 환경의 각 cluster member에서 테이블 레코드들의 Global Row Identifier (GRID) 값을 key로 구성한 B-tree 인덱스이다.

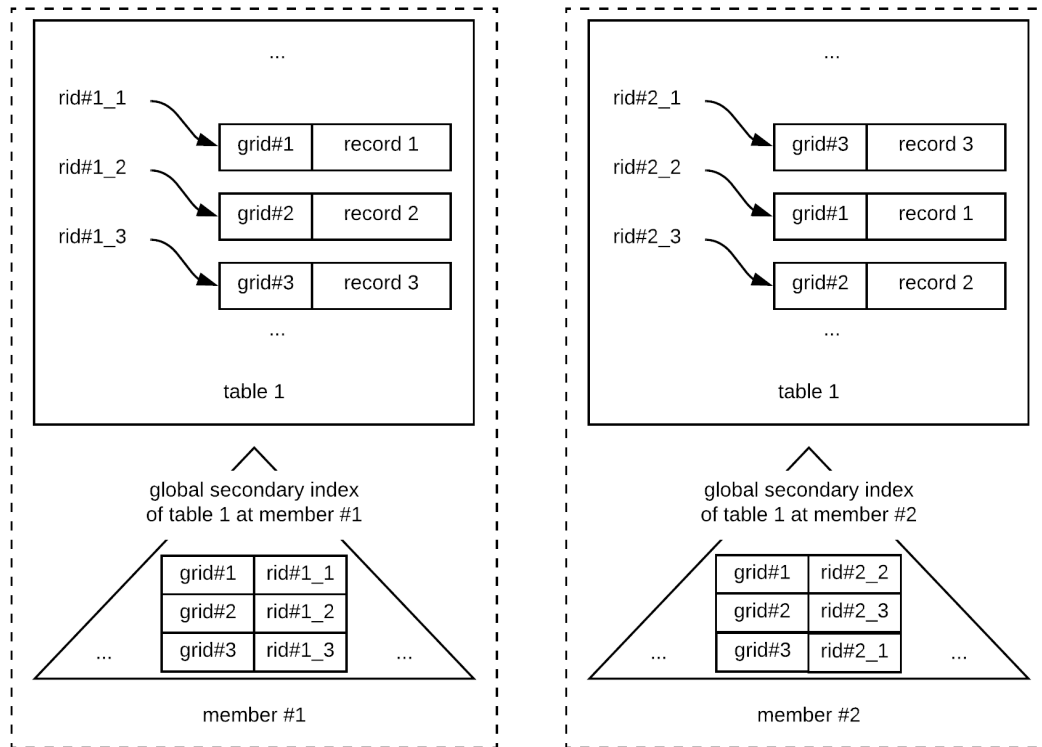
Cluster 환경에서 테이블은 group 내의 모든 member들에 복제되고, DML이나 select 시 동일한 레코드들이 반영되고 조회된다. GRID는 cluster member들에 있는 동일한 레코드들을 구분할 수 있는 고유한 값으로써 레코드가 최초로 삽입될 때 할당되어 group 내의 모든 member들에 전파되고 레코드와 함께 저장된다.

레코드의 GRID 값은 레코드가 갱신되더라도 변경되지 않고, shard key가 변경되어 다른 shard로 이동되더라도 변경되지 않는다.

Cluster 환경에서 테이블을 생성할 때 global secondary index는 생성할 수도 있고 생성하지 않을 수도 있는데 테이블을 생성한 후에 별도로 global secondary index를 생성하거나 삭제할 수 있다. 테이블에는 global secondary ind

ex가 없거나 있더라도 최대 한 개까지만 생성할 수 있다.

그림 16 Global secondary index 구조



테이블에 대한 non-deterministic 질의를 수행하기 위해서는 global secondary index가 반드시 필요한데 만약 테이블에 global secondary index가 없을 경우 non-deterministic 질의는 다음과 같이 실패한다.

```
gSQL> DELETE FROM T1 LIMIT 1;
```

```
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global  
secondary index expected
```

테이블에 global secondary index가 존재하는지 확인하려면 ALL_GSI_PLACE, DBA_GSI_PLACE, USER_GSI_PLACE와 같은 dictionary를 참조한다.

15.

SQL Tuning

15.1 SQL Tuning 개요

SQL tuning은 SQL 구문의 질의 성능을 향상시키기 위한 방법을 기술한다. GOLDILOCKS는 SQL tuning을 위한 hint, plan cache, SQL 실행 계획 출력 등을 제공한다.

SQL tuning은 질의에 대한 응답시간을 최소화하고, 질의 처리량을 개선하는 것을 목표로 하며 이를 위해 문제를 찾아내고 해결하는 방법을 기술한다.

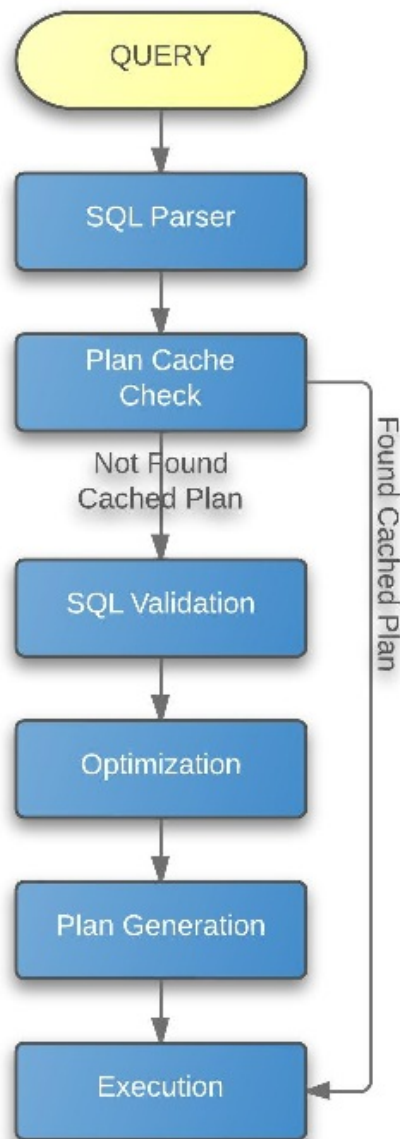
SQL tuning을 이해하기 위해서는 database, database 구조, SQL 구문, optimizer에 대한 지식이 필요하며, 본 장에서 다루는 내용은 이런 지식을 가지고 있다는 전제하에 설명한다.

15.2 SQL 처리 과정

GOLDILOCKS에서 SQL을 처리할 때 다음 그림과 같이 SQL parser, plan cache check, SQL validation, optimization, plan generation, execution 과정을 거친다.

Plan cache check 단계에서 plan cache에 동일한 질의 실행 계획이 저장되어 있으면 해당 실행 계획을 이용하여 execution을 수행한다.

그림 1 SQL 처리 과정



SQL Parser

SQL parser는 SQL 처리 과정의 첫 번째 단계로써 사용자가 입력한 SQL 구문이 문법적으로 올바른지 판단한다. 만약 SQL 구문이 문법적으로 올바르지 않다면 이 단계에서 에러로 처리된다.

다음은 문법적으로 올바르지 않은 SQL 구문의 예이다.

```
gSQL> SELECT * FORM t1;
ERR-42000(40000): syntax error
SELECT * FORM t1
.....^ ^
Error at line 1
```

SQL parser 단계에서는 parsing에 필요한 정보들을 수집하고 저장하며 이 단계가 완료되면 그 결과 SQL 구조가 parsing된다. 이 때 parsing 된 SQL 구조와 기타 정보들은 각각의 세션 영역에 저장되어 관리된다.

Plan Cache Check

SQL parser 단계가 끝나면 해당 질의와 동일한 질의가 plan cache에 저장되어 있는지 확인하는데 이 단계가 plan cache check이다.

Plan cache check에서는 SQL parser 단계가 끝난 질의 구문의 plan cache로부터 **Plan cache parameters**가 일치하는 plan을 찾는다.

해당 plan이 plan cache에 저장되어 있는 경우 이 plan을 가지고 와서 plan에서 접근하는 테이블이나 column과 같은 객체들의 유효성을 검사한다. 만약 모든 객체가 유효하면 해당 plan을 이용하여 execution 단계로 진행하며, 그렇지 않고 유효하지 않은 객체가 하나라도 존재한다면 SQL validation 단계로 진행한다.

Plan cache에서는 서로 다른 세션에 의해 발생한 plan들이 모두 공유되며, 다른 세션에 의해 저장된 plan도 참조할 수 있다.

Plan cache check 단계는 plan cache에 저장된 plan을 이용하여 SQL validation 단계부터 plan generation 단계까지의 과정을 생략함으로써 성능을 향상시킨다.

SQL Validation

SQL validation은 질의가 구문상 올바른지 판단하는 단계이다. 이 단계에서는 질의에 기술된 테이블이나 column 등과 같은 객체들이 존재하는지 여부와 참조 가능한 column인지와 같은 구문상 오류에 대하여 검사한다.

다음은 구문상 올바르지 않은 SQL 구문의 예이다.

```
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
          *
ERROR at line 1:
```

Optimization

Optimization은 SQL 구문에 대하여 다양한 실행 계획을 세우고 그 중 가장 좋은 plan을 선택하는 단계이다. Optimization 단계에서는 테이블에 대한 access 방법, join 순서, join 방법 등의 최적화 기법들에 대한 cost를 계산하며, 이를 기준으로 다양한 plan을 만든 후 각 plan의 cost를 비교하여 최적의 plan을 선택한다.

자세한 내용은 **Query Optimizer**을 참조한다.

Plan Generation

Plan generation은 optimization이 최종적으로 선택한 plan을 execution 단계에서 수행할 수 있는 실행 계획 형태로 생성하는 단계이다. 실행 계획은 여러 단계 노드들의 조합으로 구성되며, 각 단계의 노드들은 수행한 결과 집합을 상위 노드로 반환한다. 최종 단계의 노드는 SQL 구문의 최종 결과를 사용자에게 보낸다.

실행 계획은 tree 구조의 노드들로 구성되는데 tree 구조에는 다음과 같은 정보들이 포함된다.

- 각 테이블들의 access 방법
- 참조하는 테이블들의 순서
- 테이블들의 조인 연산에 대한 조인 방법
- 데이터 filter에 대한 정보
- 데이터 grouping 및 aggregation 정보
- 데이터 정렬에 대한 정보

다음은 plan generation으로 SQL 실행 계획을 생성하는 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT t1.i1, t1.i2, t2.i1, t2.i2
   FROM t1, t2
  WHERE t1.i1 = t2.i1
        AND t1.i2 = 1
   ORDER BY t1.i1;
no rows selected.
```

```
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0   |  SELECT STATEMENT                                |         |
|    1   |    SORT INSTANT ACCESS                           |         0 |
|    2   |      HASH JOIN (INNER JOIN)                      |         0 |
|    3   |        TABLE ACCESS ("T2")                     |         0 |
|    4   |          HASH JOIN INSTANT ACCESS                 |         0 |
|    5   |            TABLE ACCESS ("T1")                  |         0 |
=====

1 - SORT KEY : "T1.I1 ASC NULLS LAST"
    RECORD COLUMNS : I2, I1, I2
    READ COLUMNS : I1, I2, I1, I2
2 - JOINED COLUMNS : T1.I1, T1.I2, T2.I1, T2.I2
3 - READ COLUMNS : I1, I2
4 - INDEX COLUMNS : I1
    TABLE COLUMNS : I2
    READ COLUMNS : I1, I2
    HASH FILTER : I1 = {I1}
5 - READ COLUMNS : I1, I2
    PHYSICAL FILTER : I2 = 1

<<< end print plan
```

Plan Cache Registration

Plan cache를 사용하는 경우 **Plan Generation**에서 생성된 plan을 plan cache에 등록한다. **Plan cache parameters** 값들의 일치 여부를 기준으로 cache 내의 plan들을 구분한다.

표 15-1 Plan cache parameters

Parameter	설명
Query text	대소문자를 구분하는 query text
User information	User id
Cursor property	Fetch가 필요한 query의 cursor 속성
Bind parameter	Bind parameter 개수와 각 bind parameter의 IN/OUT 속성
Enable atomic	Atomic insertion 사용 여부
Enable hint error	Hint에 validation error 발생 여부

다음은 서로 다른 query text 값을 가지는 query들의 예이다.

```
"select i1 from t1"
"Select i1 from t1"
"select i1 from t1"
"SELECT I1 FROM T1"
```

노트

Plan에서 참조하고 있는 스키마 객체 (테이블, 인덱스, view, 시퀀스)가 commit 되지 않은 경우에는 plan이 등록되지 않는다.

Execution

Execution은 plan generation에 의해 생성되거나 plan cache check에서 선택된 실행 계획을 실제로 수행하여 결과를 반환하는 단계이다. Tree 구조의 실행 계획을 실제로 수행할 때는 왼쪽 최하위의 노드를 먼저 수행한 후에 상위 노드를 수행한다.

```
gSQL>
```

```
\EXPLAIN PLAN
```

```
SELECT t1.i1, t1.i2, t2.c1, t2.c2
```

```
FROM t1, t2
```

```
WHERE t1.i1 = t2.c1
```

```
AND t1.i2 = 1
```

```
ORDER BY t1.i1;
```

```
no rows selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	SORT INSTANT ACCESS	0
2	HASH JOIN (INNER JOIN)	0
3	TABLE ACCESS ("T2")	0
4	HASH JOIN INSTANT ACCESS	0
5	TABLE ACCESS ("T1")	0

```

1 - SORT KEY : "T1.I1 ASC NULLS LAST"
   RECORD COLUMNS : I2, C1, C2
   READ COLUMNS : I1, I2, C1, C2
2 - JOINED COLUMNS : T1.I1, T1.I2, T2.C1, T2.C2
3 - READ COLUMNS : C1, C2
4 - INDEX COLUMNS : I1
   TABLE COLUMNS : I2
   READ COLUMNS : I1, I2
   HASH FILTER : I1 = {C1}
5 - READ COLUMNS : I1, I2
   PHYSICAL FILTER : I2 = 1

```

```
<<< end print plan
```

위의 예제를 이용하여 실제 실행하는 과정은 다음과 같다.

1. IDX3에서 테이블 T2에 대한 table access를 이용하여 결과를 가져온다. 이 때 반환되는 결과 row에는 C1과 C2 column이 포함된다.
2. IDX5에서 테이블 T1에 대한 table access를 이용하여 결과를 가져온다. 이 때 "I2 = 1"이 physical filter로 처리된 결과를 반환하며, 반환되는 결과 row에는 I1과 I2 column이 포함된다.
3. IDX4에서 IDX5로부터 반환된 결과에 대해 hash join instant access를 생성하고 hash filter로 "I1 = {C1}"을 수행한다. 반환되는 결과 row에는 I1과 I2 column이 포함된다.
4. IDX4에서 hash filter를 이용하여 IDX2에서 IDX3의 수행 결과에 대해 hash join을 수행하고 결과로 반환되는 결과 row에는 T1 테이블의 I1과 I2 column 및 T2 테이블의 C1과 C2 column이 포함된다.
5. IDX1에서 IDX2의 수행 결과에 대하여 sort instant access를 생성하고 T1 테이블의 I1 column을 이용하여 정렬한다. 결과에는 T1 테이블의 I1과 I2 column 및 T2 테이블의 C1과 C2 column이 포함된다.
6. IDX0에서 IDX1의 수행 결과를 최종적으로 사용자에게 반환한다.

SQL 실행 계획에 대한 자세한 내용은 **SQL 실행 계획**을 참조한다.

15.3 Query Optimizer

개요

Query optimizer는 사용자의 SQL 구문을 가장 효율적으로 처리할 수 있는 실행 계획을 찾아내는 모듈이다. 이를 위해 query optimizer는 SQL 구문에 대하여 다양한 형태의 candidate plan을 생성하고 각 plan의 cost를 계산한다. 이렇게 계산한 cost들 중에 가장 cost가 작은 candidate plan를 최종 실행 계획으로 선택한다.

Cost를 계산하여 효율적인 plan을 찾는 과정을 cost-based optimization이라 하며, 이 과정에서 query optimizer는 각 노드가 반환하는 row의 개수, access paths, join methods와 같은 정보를 사용한다.

Query optimizer는 query transformations, 통계 정보를 이용해 parsing과 validation을 거친 SQL 구문의 cost를 계산하여 candidate plan들을 만들고, candidate plan들 중에서 가장 효율적인 (가장 cost가 작은) plan을 선택하여 최종 실행 계획을 작성한다.

Query transformations는 조건 구문을 view로 push하거나 subquery를 join 형태로 변환하는 것으로서 heuristic query transformation과 cost-based query transformation이 있다. Heuristic query transformation은 query를 변형하는 것이 그렇지 않은 것보다 항상 효율적인 경우에 transformation을 수행한다. Cost-based query transformation은 query를 변형하는 것이 항상 효율적인 것만은 아닐 경우에 변형하기 전과 후의 cost를 계산하여 효율적인 형태를 찾아 결정한다.

Cost를 계산할 때는 기본적으로 selectivity와 cardinality를 사용한다. Selectivity는 조건에 따라 결과 집합으로 반환될 row의 비율이고 cardinality는 각 노드에서 결과로 반환하는 row의 개수이다.

Query optimizer가 cost를 계산할 때는 selectivity와 cardinality를 기본으로 하여 table access, index access와 같은 access path에 대한 방법과 nested loops join, hash join과 같은 join method에 대한 방법, join ordering과 같은 방법들을 사용한다.

Query optimizer가 최종적으로 선택한 실행 계획은 explain plan 구문을 통해 확인할 수 있다. Explain plan 구문을 통해 출력된 실행 계획을 해석하기 위한 정보는 **SQL 실행 계획**을 참조한다.

Query Transformation

Query transformation은 SQL 구문을 보다 효율적으로 수행하기 위하여 filter를 push하거나 subquery를 unnest하는 식으로 SQL 구문을 변형한다. 이러한 query transformation에 대하여 GOLDBLOCKS는 다음과 같은 기법들을 사용한다.

- Simple view merging
- Filter push down
- Subquery unnesting

- Single table min/ max aggregation conversion
- Rewrite target on exists

Simple View Merging

Simple view를 상위 query block에 merge 한다.

Simple view merging을 하면 join ordering, join operation, access path를 결정할 때 더 많은 경우의 수를 적용할 수 있어 보다 최적화된 plan을 생성할 수 있다.

다음은 예제 질의이다.

```
gSQL> \explain plan
      select *
      from   ( select t1.col1 col1, t2.col1 col2
                from   t1, t2
                where  t1.col1 = t2.col1 )v1, t3
      where  v1.col1 = t3.col1;
```

t1, t2 테이블은 크고, t3 테이블은 작다고 가정한다.

Simple view merging이 되지 않으면 위 질의는 view 안의 join을 먼저 수행한 후 t3를 수행하게 되므로 중간 결과가 클 뿐만 아니라 많이 걸려지지도 않는다.

< Execution Plan >

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	NESTED LOOP JOIN (INNER JOIN)	1
2	VIEW (INLINE_VIEW AS V1)	5
3	NESTED LOOP JOIN (INNER JOIN)	5
4	INDEX ACCESS ("T2", "T2_COL1")	(5) 5
5	INDEX ACCESS ("T1", "T1_COL1")	(5) 5
6	INDEX ACCESS ("T3", "T3_COL3")	(1) 1

그러나 simple view merging이 되면 t1, t3를 먼저 수행하고, t2를 수행하기 때문에 중간 결과가 많이 걸려지므로 성능이 향상된다.

< Execution Plan >

IDX	NODE DESCRIPTION	ROWS

	0		SELECT STATEMENT			
	1		NESTED LOOP JOIN (INNER JOIN)			1
	2		NESTED LOOP JOIN (INNER JOIN)			1
	3		TABLE ACCESS ("T3")			1
	4		INDEX ACCESS ("T1", "T1_COL1")		(1)	1
	5		INDEX ACCESS ("T2", "T2_COL1")		(1)	1
=====						

Simple view merging이 가능한 view 조건은 다음과 같다.

- View 내부에 set 연산자가 존재하지 않아야 한다.
- View 내부에 DISTINCT가 없어야 한다.
- View 내부에 GROUP BY, HAVING, aggregate function이 존재하지 않아야 한다.
- View 내부에 full outer join이 없어야 한다.
- View 내부에 natural join이 없어야 한다.
- View 내부 SELECT list에 subquery를 포함하지 않아야 한다.
- View 내부에 LIMIT, OFFSET 구문이 존재하지 않아야 한다.
- View가 full outer join에 참여하지 않아야 한다.
- View가 left outer join의 오른쪽에 참여하고 있는 경우, view 내부의 from 절에 하나의 table만 존재해야 한다.
- View가 semi join의 오른쪽에 오면 안된다.

Filter Push Down

Filter push down은 <where clause>에 존재하는 filter들 중에서 FROM 절에 기술한 subquery (view)에 push 가능한 filter들을 push하는 기능이다. Filter push down이 subquery (view)에 push한 filter는 subquery (view)에서 index access 하는데 사용되거나 subquery (view)를 수행할 때 먼저 처리되는 filter로 작동하여 질의 처리 성능을 향상시킨다.

다음은 예제 질의이다.

```
SELECT l_linenum, l_quantity
FROM ( SELECT *
      FROM lineitem
      WHERE l_shipdate >= date '1996-01-01'
        AND l_shipdate <= date '1996-12-31' )
WHERE l_shipmode = 'AIR';
```

위 질의에서 최상위 노드 WHERE 절에 있는 filter (l_shipmode = 'AIR')는 FROM 절에 기술된 subquery로 push 되며 이는 다음 질의와 동일하다.

```

SELECT l_linenum, l_quantity
FROM ( SELECT *
        FROM lineitem
        WHERE l_shipdate >= date '1996-01-01'
              AND l_shipdate <= date '1996-12-31'
              AND l_shipmode = 'AIR' );

```

위와 같이 변형된 질의는 lineitem 테이블에 l_shipmode에 대한 index가 있을 경우 index access를 사용하여 성능을 향상시킬 수 있다.

Subquery Unnesting

Subquery unnesting은 조건절에 있는 subquery를 join 형태로 풀어내는 기능이다. Subquery unnesting을 사용하여 IN, NOT IN, EXISTS, NOT EXISTS 연산자와 ANY, ALL의 quantify 연산자들을 semi join 또는 anti-semi join으로 변형할 수 있고 이렇게 변형된 질의는 join을 효율적으로 처리하여 질의 처리 성능을 향상시킨다.

다음은 예제 질의이다.

```

SELECT ps_availqty
FROM partsupp
WHERE ps_partkey IN ( SELECT p_partkey
                      FROM part
                      WHERE p_type = 'STEEL' );

```

위 질의에서 IN 연산자는 ps_partkey = p_partkey 조건을 갖는 semi join으로 변형될 수 있고 이렇게 변형된 질의는 다음과 같이 실행 계획을 출력하여 확인할 수 있다.

```

gSQL>
\EXPLAIN PLAN
SELECT ps_availqty
FROM partsupp
WHERE ps_partkey IN ( SELECT p_partkey
                      FROM part
                      WHERE p_type = 'STEEL' );

PS_AVAILQTY
-----
      8895
      4969
      4651
      4093
4 rows selected.
>>> start print plan

```

< Execution Plan >

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |        |
|   1   |    NESTED LOOP JOIN (INVERTED LEFT SEMI)         |    4   |
|   2   |      SORT INSTANT ACCESS (UNIQUE)                |    2   |
|   3   |        TABLE ACCESS ("PART")                   |    2   |
|   4   |          INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") |    4   |
=====

1 - JOINED COLUMNS : PARTSUPP.PS_AVAILQTY
2 - SORT KEY : "PART.P_PARTKEY ASC NULLS LAST"
   READ COLUMNS : P_PARTKEY
3 - READ COLUMNS : P_PARTKEY, P_TYPE
   PHYSICAL FILTER : P_TYPE = 'STEEL'
4 - READ INDEX COLUMNS : PS_PARTKEY
   READ TABLE COLUMNS : PS_AVAILQTY
   MIN RANGE : PS_PARTKEY = {P_PARTKEY}
   MAX RANGE : PS_PARTKEY = {P_PARTKEY}

<<< end print plan

```

위에 출력된 실행 계획에서 IN 연산이 nested loop join을 이용한 inverted left semi join으로 변형되었다.

Single Table Min/ Max Aggregation Conversion

질의가 선택 목록에 min/ max aggregation을 가지고 있고 from 절에 single table이 있을 때, aggregation의 argument가 column이고 그 column을 첫 번째 key column으로 가지는 index가 존재하면 optimizer가 이 인덱스를 사용하여 결과를 반환한다.

이 기능을 이용하기 위해서는 다음의 조건을 만족해야 한다.

- 단일 테이블에 대한 질의이어야 하며, target에 MIN 또는 MAX 중 하나만 존재해야 한다.
- 질의에 OFFSET/ LIMIT 구문이 없어야 한다.
- Aggregation의 인자는 반드시 테이블의 column 하나만 존재해야 한다.
- Aggregation 대상 column이 index의 첫 번째 key인 index가 존재해야 한다.
- 사용자가 table access 또는 rowid access와 같은 힌트를 주지 않아야 한다.

Single table min/ max aggregation conversion은 모든 row를 읽지 않고 index의 처음 또는 끝의 데이터 하나만 읽어오는 방식으로 질의 처리 성능을 향상시킨다.

다음은 예제 질의이다.

```

SELECT p_name, p_brand, p_type
  FROM part
 WHERE p_size = ( SELECT MAX( p_size )
                  FROM part );

```

위 질의는 part 테이블에서 p_size가 가장 큰 값의 p_name, p_brand, p_type을 가져오는 질의로써 조건절에 있는 subquery에 MAX aggregation이 존재한다. 이는 p_size의 index access를 사용하여 ascending으로 정렬되어 있는 p_size 데이터 중에 마지막 한건만 가져오는 질의로 변형할 수 있으며, 이렇게 변형된 질의는 다음과 같이 실행 계획을 출력하여 확인할 수 있다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE p_size = ( SELECT MAX( p_size )
                  FROM part );
P_NAME P_BRAND  P_TYPE
-----
Part#3 Brand#2  STEEL
1 row selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    TABLE ACCESS ("PART")                        |         1 |
|    2  |      SUB QUERY LIST                              |         1 |
|    3  |        INDEX ACCESS ("PART, PART_SIZE")          |         1 |
=====
1 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE, P_SIZE
    PHYSICAL FILTER : P_SIZE = P_SIZE
2 - READ COLUMNS : P_SIZE
3 - READ INDEX COLUMNS : P_SIZE
    MAX RANGE : P_SIZE IS NOT NULL
<<< end print plan

```

위에 출력된 실행 계획에서 조건절의 subquery에 index access를 이용하여 max range의 값을 읽어온다는 것을 알 수 있다.

Rewrite Target on Exists

Rewrite target on exists는 exists 또는 not exists 연산에 존재하는 subquery에서 target절을 constant value로 변형하는 기능이다. Exists 또는 not exists 연산은 subquery의 결과 row가 존재하는지 여부를 판단하는 연산자로서 target의 개수나 target의 expression 처리 결과가 연산 결과에 영향을 미치지 않기 때문에 target절을 constant value로 변형하더라도 결과는 동일하다.

Rewrite target on exists는 target절의 불필요한 expression 처리를 줄여 질의 처리 성능을 향상시킨다.

다음은 예제 질의이다.

```
SELECT p_name, p_brand, p_type
  FROM part
 WHERE EXISTS( SELECT /*+ NO_QUERY_TRANSFORMATION */
                l_quantity, l_extendedprice * (1 - l_discount)
              FROM lineitem
              WHERE l_partkey = p_partkey
                AND l_quantity > 30 );
```

위 질의는 lineitem 테이블에서 l_partkey와 p_partkey가 동일한 row 중에 l_quantity가 30보다 큰 row가 존재하면 해당 p_partkey에 대한 part 테이블의 row를 결과로 반환한다. 조건절의 exists 연산자에는 l_quantity와 l_extendedprice * (1 - l_discount)라는 두 target이 존재하는데 이는 TRUE라는 constant value로 변형할 수 있고 이렇게 변형된 질의는 다음과 같이 실행 계획을 출력하여 확인할 수 있다. (참고로 **NO_QUERY_TRANSFORMATION** 힌트는 질의를 변형하지 않도록 하는 것이다.)

```
gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE EXISTS( SELECT /*+ NO_QUERY_TRANSFORMATION */
                l_quantity, l_extendedprice * (1 - l_discount)
              FROM lineitem
              WHERE l_partkey = p_partkey
                AND l_quantity > 30 );
P_NAME P_BRAND  P_TYPE
-----
Part#1 Brand#1  COPPER
Part#3 Brand#2  STEEL
Part#4 Brand#3  NICKEL
Part#5 Brand#3  STEEL
4 rows selected.
>>> start print plan
< Execution Plan >
```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |        |
|    1  |    SUB QUERY FILTER                              |        |
|    2  |      TABLE ACCESS ("PART")                      |        |
|    3  |        SUB QUERY LIST                            |        |
|    4  |          SUB QUERY FUNCTION                      |        |
|    5  |            TABLE ACCESS ("LINEITEM")            |        |
=====

1 - FILTER : EXISTS( ( TRUE ) )
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
4 - FUNCTION : EXISTS( ( TRUE ) )
5 - READ COLUMNS : L_PARTKEY, L_QUANTITY
    PHYSICAL FILTER : L_PARTKEY = {P_PARTKEY} AND L_QUANTITY > 30
<<< end print plan

```

위에 출력된 실행 계획에서 exists의 subquery target이 TRUE로 변형되었다는 것을 알 수 있다.

Access Paths

Access path는 단일 테이블에 access 하는 방법이다. 이러한 access 방법에는 table access, index access, rowid access, index concat 등이 있고 query optimizer는 각 access 방법별로 cost를 계산하여 그 중 가장 cost가 작은 access를 실행 계획으로 선택한다.

Table Access

Table access는 테이블을 검색할 때 index나 rowid를 사용하지 않고 저장된 테이블 그대로 scan하는 방식이다. 일반적으로 table access는 다른 access 방식보다 비용이 크기 때문에 다른 access 방식을 사용할 수 없는 경우나 사용자가 힌트로 table access를 요구한 경우에만 query optimizer가 이 방식을 선택한다.

다음과 같은 경우에 table access를 사용한다.

- Index가 존재하지 않는 경우
- Index에 존재하는 column에 대한 filter에서 column이 존재하는 쪽에 function이 존재하는 경우 (예: $i1 + 1 = 10$)
- Index의 첫 번째 column에 대한 조건이 없어서 테이블 접근 비용이 작은 경우 (예: index key가 $i1, i2$ 인 index에 $i2 = 3$ 조건이 주어진 경우)
- 테이블이 작아 index access보다 table access의 비용이 더 작은 경우
- 사용자가 테이블 접근 힌트를 준 경우 (예: FULL(t1) 힌트)
- Index에 대한 조건이 존재하지만 테이블에 접근할 필요가 있어서 cost를 계산한 결과 table access 비용이 더 작

은 경우

다음은 filter의 column을 포함하는 index가 없는 경우에 table access를 사용하는 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE p_size > 20;
P_NAME P_BRAND  P_TYPE
-----
Part#3 Brand#2   STEEL
1 row selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----
|    0  |  SELECT STATEMENT                                |        |
|    1  |  TABLE ACCESS ("PART")                          |        1 |
=====
1 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE, P_SIZE
    PHYSICAL FILTER : P_SIZE > 20
<<< end print plan
```

다음은 filter의 column을 포함하는 index가 있지만 사용자가 힌트로 table access를 명시했을 때의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT /*+ FULL(part) */
      p_name, p_brand, p_type
  FROM part
 WHERE p_partkey > 1;
P_NAME P_BRAND  P_TYPE
-----
Part#2 Brand#1   NICKEL
Part#3 Brand#2   STEEL
Part#4 Brand#3   NICKEL
Part#5 Brand#3   STEEL
4 rows selected.
>>> start print plan
< Execution Plan >
```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |        |
|    1  |  TABLE ACCESS ("PART")                          |        4 |
=====

1 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
    PHYSICAL FILTER : P_PARTKEY > 1
<<< end print plan

```

Index Access

Index access는 테이블을 검색할 때 index를 사용하여 scan하는 방식이다. Index를 이용할 수 있는 filter가 있을 경우 일반적으로 다른 access보다 index access가 더 효율적이다. Index access를 사용할 수 있는 경우, query optimizer가 cost를 계산하여 가장 비용이 작은 index access를 선택하며, 사용자가 index access를 힌트로 명시하는 경우, 사용자가 명시한 index들의 cost를 계산하여 가장 비용이 작은 index access를 선택한다.

다만 다음과 같은 경우에는 index access를 선택하지 않는다.

- Index에 해당하는 filter가 있지만 cost를 계산한 결과 table access와 같은 다른 access 비용보다 더 비쌀 경우
- 사용자가 index access 이외의 다른 access 힌트를 명시하고 해당 access가 가능한 경우 (예: FULL(t1) 힌트)
- 다른 access를 선택하게 되는 조건인 경우 (**Table Access**를 사용하는 경우를 참조한다.)

다음은 query optimizer가 선택한 index access를 사용하는 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE p_partkey = 1;
P_NAME P_BRAND  P_TYPE
-----
Part#1 Brand#1  COPPER
1 row selected.
>>> start print plan
< Execution Plan >

```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |        |
|    1  |  INDEX ACCESS ("PART, PART_PK_INDEX")            |        1 |
=====

```

```

1 - READ INDEX COLUMNS : P_PARTKEY
    READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
    MIN RANGE : P_PARTKEY = 1
    MAX RANGE : P_PARTKEY = 1
<<< end print plan

```

다음은 index access를 힌트로 명시했을 때의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT /*+ INDEX(part, part_size) */
    p_name, p_brand, p_type
FROM part
WHERE p_size > 10;
P_NAME P_BRAND    P_TYPE
-----
Part#4 Brand#3    NICKEL
Part#5 Brand#3    STEEL
Part#3 Brand#2    STEEL
3 rows selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----
|    0  |  SELECT STATEMENT                                |        |
|    1  |  INDEX ACCESS ("PART, PART_SIZE")                |        3 |
=====

1 - READ INDEX COLUMNS : P_SIZE
    READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
    MIN RANGE : P_SIZE > 10
    MAX RANGE : P_SIZE IS NOT NULL
<<< end print plan

```

Rowid Access

Rowid access는 테이블을 검색할 때 rowid를 사용하여 해당 페이지에 직접 접근하는 방식이다. Rowid access를 사용하기 위해서는 반드시 rowid에 대한 filter가 존재해야 한다. 일반적으로 rowid에 대한 조건이 존재하는 경우 query optimizer가 다른 access보다 우선적으로 rowid access를 선택한다.

다음은 rowid access를 사용하는 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE ROWID = 'AAAAAAAAADiACAAACCjAAA';
P_NAME P_BRAND   P_TYPE
-----
Part#1 Brand#1   COPPER
1 row selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |      ROWS |
-----
|    0  |  SELECT STATEMENT                                |           |
|    1  |  USER ROWID ACCESS ("PART")                     |          1 |
=====
1 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE
   ROWID ACCESS EXPR : ROWID = 'AAAAAAAAADiACAAACCjAAA'
<<< end print plan

```

Index Concat

Index concat은 filter에 or 구문이 존재하고 or에 의해 나뉘어진 filter들이 각각 index access할 수 있을 경우 각각의 index access 결과를 통합하여 하나의 결과로 만든다. 즉, index concat은 하위 노드에 다수의 index access를 갖는 concat 노드이다. Filter에 or 구문이 존재하는 경우 query optimizer가 index concat cost를 계산하고 다른 access보다 index concat의 비용이 작을 경우 이를 선택한다.

Index concat의 cost는 다음의 단계로 계산된다.

1. Or을 기준으로 재조정된 filter를 생성한다.
2. Or을 기준으로 분류한 각각의 filter들에 적용 가능한 index 중 최적의 index를 선택한다.
3. 선택한 index들의 결과를 취합할 때 중복을 제거하기 위한 concat의 cost를 계산한다.
4. 앞서 선택한 concat의 cost를 합산하여 index concat의 최종 cost를 결정한다.

Index concat node는 다음 단계로 수행된다.

1. Index concat node의 하위 노드들 중 첫 번째 노드를 수행한다.
2. 수행 결과 중 중복되는 것을 제거하기 위한 데이터를 concat 노드에 저장하고, 결과를 상위 노드로 보낸다.
3. 두 번째 노드부터는 concat 노드에서 중복여부를 판단하여 중복되지 않는 row들에서 중복을 제거하기 위한 데이터를 concat 노드에 저장하고, 결과를 상위 노드로 보낸다.

다음은 index concat을 사용하는 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT /*+ INDEX_COMBINE(part, part_size) */
      p_name, p_brand, p_type
FROM part
WHERE p_size = 1
      OR p_size = 21;
P_NAME P_BRAND    P_TYPE
-----
Part#2 Brand#1    NICKEL
Part#3 Brand#2    STEEL
2 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |      ROWS |
=====
|    0  |  SELECT STATEMENT                                |           |
|    1  |    CONCAT                                          |          2 |
|    2  |      INDEX ACCESS ("PART, PART_SIZE")             |          1 |
|    3  |      INDEX ACCESS ("PART, PART_SIZE")             |          1 |
=====

2 - READ INDEX COLUMNS : P_SIZE
   READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
   MIN RANGE : P_SIZE = 1
   MAX RANGE : P_SIZE = 1
3 - READ INDEX COLUMNS : P_SIZE
   READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
   MIN RANGE : P_SIZE = 21
   MAX RANGE : P_SIZE = 21
<<< end print plan
```

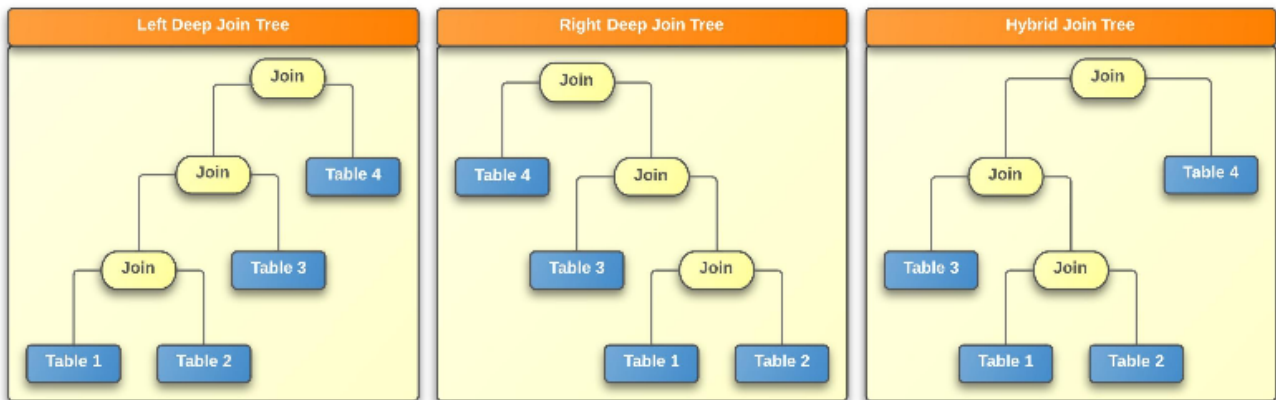
Join

Join은 두 테이블 (또는 view)의 결과 row들을 하나의 결과 row로 결합하는 과정이다. Join을 하는 과정에서 두 테이블 (또는 view)의 row들을 결합하는 조건이 있을 수 있는데 이를 join 조건이라고 한다. Join을 수행할 때 join 조건이 존재하지 않는 경우에는 한 쪽 테이블의 각 row를 다른쪽 테이블의 모든 row들과 결합한 결과 row들을 반환한다.

Join 처리과정은 일반적으로 tree 형태로 표현된다. Join의 tree에서 왼쪽에 놓여진 테이블을 outer node라고 하고 오른쪽에 놓여진 테이블을 inner node라고 하는데 일반적으로 join을 수행할 때 outer node의 row 하나를 읽어온 후 inner node에서 join 조건과 일치하는 row들을 읽어와서 결합한다.

FROM 절에 셋 이상의 테이블 (또는 view)들이 기술된 경우 두 테이블을 먼저 join한 후에 그 join 결과와 남아있는 다른 테이블을 join한다. 이 때 join tree의 outer node에만 join node가 존재하는 경우 left deep join tree라고 하고 inner node에만 존재하는 경우 right deep join tree라고 한다. 만약 join node가 join tree의 outer node와 inner node 모두에 존재하는 경우 hybrid join tree라고 한다. 다음 그림은 join tree의 종류를 나타낸다.

그림 2 Join tree의 종류



위 그림에서 세 종류의 join tree 모두 table 1, table 2, table 3, table 4의 순서로 join 되며 각 테이블의 데이터와 join 조건이 모두 같다면 세 종류의 join tree 모두 동일한 결과를 반환하지만 결과 row들의 순서는 서로 다를 수 있다.

Query optimizer는 다음 네 가지 사항을 고려하여 join cost를 계산한다.

- Join에 참여하는 각 테이블들의 access paths에 대한 cost
- Join 종류 (inner, outer 등)에 따른 cost
- 가능한 join methods들의 cost
- Join에 참여하는 두 테이블의 순서에 따른 cost

위에서 언급한 네 가지 사항에 따라 cost가 달라지며 query optimizer는 이 중 가장 비용이 작은 plan을 선택한다.

Join 종류

Cross Join

Cross join에는 join 조건이 없기 때문에 outer node의 각 row에 inner node의 모든 row를 결합하여 join 결과로 반환된다.

다음은 cross join의 예이다.

```
gSQL>
\EXPLAIN PLAN
```

```
SELECT s_name, c_name FROM supplier, customer;
```

S_NAME	C_NAME
Supplier#1	Customer#1
Supplier#1	Customer#2
Supplier#1	Customer#3
Supplier#1	Customer#4
Supplier#1	Customer#5
Supplier#2	Customer#1
Supplier#2	Customer#2
Supplier#2	Customer#3
Supplier#2	Customer#4
Supplier#2	Customer#5
Supplier#3	Customer#1
Supplier#3	Customer#2
Supplier#3	Customer#3
Supplier#3	Customer#4
Supplier#3	Customer#5
Supplier#4	Customer#1
Supplier#4	Customer#2
Supplier#4	Customer#3
Supplier#4	Customer#4
Supplier#4	Customer#5
Supplier#5	Customer#1
Supplier#5	Customer#2
Supplier#5	Customer#3
Supplier#5	Customer#4
Supplier#5	Customer#5

25 rows selected.

```
>>> start print plan
```

```
< Execution Plan >
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	25
1	NESTED LOOP JOIN (INNER JOIN)	25
2	TABLE ACCESS ("SUPPLIER")	5
3	TABLE ACCESS ("CUSTOMER")	5

```

1 - JOINED COLUMNS : SUPPLIER.S_NAME, CUSTOMER.C_NAME
2 - READ COLUMNS : S_NAME
3 - READ COLUMNS : C_NAME
<<< end print plan

```

Inner Join

Inner join에는 join 조건이 있다. Inner join은 outer node의 각 row에 join 조건을 만족하는 inner node row들만 결합하여 join 결과로 반환한다.

Join 조건은 두 테이블 (또는 view)의 column들 간의 연산자로 구성되어 있는데, Join 조건의 연산자가 =(equal)인 경우 equi-join이라 하고 그렇지 않은 경우 non-equi-join이라고 한다.

다음은 inner join 중 equi-join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, ps_availqty
  FROM part, partsupp
 WHERE p_partkey = ps_partkey;
P_NAME PS_AVAILQTY
-----

```

```

Part#1      3325
Part#1      8076
Part#2      3956
Part#2      4069
Part#3      8895
Part#3      4969
Part#4      8539
Part#4      3025
Part#5      4651
Part#5      4093

```

10 rows selected.

```
>>> start print plan
```

```
< Execution Plan >
```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |  HASH JOIN (INNER JOIN)                          |    10  |
|    2  |  TABLE ACCESS ("PARTSUPP")                      |    10  |
|    3  |  HASH JOIN INSTANT ACCESS                        |    10  |
|    4  |  TABLE ACCESS ("PART")                          |     5  |
=====

```



```

=====
1 - JOINED COLUMNS : PART.P_NAME, PARTSUPP.PS_AVAILQTY
2 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
3 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME
   READ COLUMNS : P_PARTKEY, P_NAME
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
4 - READ COLUMNS : P_PARTKEY, P_NAME
<<< end print plan

```

다음은 inner join 중 non-equi-join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT o_totalprice, l_extendedprice, l_discount
FROM orders, lineitem
WHERE o_orderkey = 1
AND l_shipdate > o_orderdate;
O_TOTALPRICE L_EXTENDEDPRICE L_DISCOUNT
-----
173665.47      21168.23      .04
173665.47      45983.16      .09
173665.47      13309.6       .1
173665.47      28955.64      .09
173665.47      22824.48      .1
5 rows selected.
>>> start print plan
< Execution Plan >

```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |        |
|   1   |    NESTED LOOP JOIN (INNER JOIN)                 |        |
|   2   |      INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")    |        |
|   3   |        TABLE ACCESS ("LINEITEM")                |        |
=====
1 - JOINED COLUMNS : ORDERS.O_TOTALPRICE, LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
2 - READ INDEX COLUMNS : O_ORDERKEY
   READ TABLE COLUMNS : O_TOTALPRICE, O_ORDERDATE
   MIN RANGE : O_ORDERKEY = 1
   MAX RANGE : O_ORDERKEY = 1

```

```

3 - READ COLUMNS : L_EXTENDEDPRICE, L_DISCOUNT, L_SHIPDATE
    PHYSICAL FILTER : L_SHIPDATE > {O_ORDERDATE}

```

```
<<< end print plan
```

Outer Join

Outer join에는 join 조건이 있어서 outer node의 각 row에 join 조건을 만족하는 inner node row들만 결합하여 결과로 반환한다. 만일 join 조건을 만족하는 row가 없으면 NULL 데이터만을 갖는 row와 결합한 row를 join 결과로 반환한다.

Outer join은 left, right, full과 같은 방향성을 갖는다. Left outer join의 경우 구문의 왼쪽에 존재하는 테이블이 outer node가 되고 right outer join의 경우 구문의 우측에 존재하는 테이블이 outer node가 된다. Full outer join은 left outer join 결과와 right outer join 결과들의 합집합 개념으로써 join 조건을 만족하는 row들을 결합한 row와 함께 join 조건을 만족하지 못하는 left node의 row들과 right node의 row들을 NULL 데이터만 갖는 row와 결합한 row들을 join 결과로 반환한다.

Right outer join은 구문의 양쪽에 있는 테이블의 위치를 바꾸어 left outer join으로 수행한 결과와 같다. 즉, "A LEFT OUTER JOIN B"와 "B RIGHT OUTER JOIN A"의 결과는 서로 같다. Query optimizer에서는 이를 기반으로 모든 right outer join을 left outer join으로 치환한 plan을 생성한다.

다음은 left outer join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, ps_availqty
FROM part LEFT OUTER JOIN partsupp
ON p_partkey = ps_partkey
AND ps_availqty > 5000;
P_NAME P_BRAND    PS_AVAILQTY
-----
Part#1 Brand#1      8076
Part#2 Brand#1      null
Part#3 Brand#2     8895
Part#4 Brand#3     8539
Part#5 Brand#3      null
5 rows selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----
|    0  |  SELECT STATEMENT                                |        |
|    1  |  HASH JOIN (LEFT OUTER JOIN)                     |        |

```

```

| 2 | TABLE ACCESS ("PART") | 5 |
| 3 | HASH JOIN INSTANT ACCESS | 5 |
| 4 | TABLE ACCESS ("PARTSUPP") | 3 |
=====
1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PARTSUPP.PS_AVAILQTY
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND
3 - INDEX COLUMNS : PS_PARTKEY
   TABLE COLUMNS : PS_AVAILQTY
   READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   HASH FILTER : {P_PARTKEY} = PS_PARTKEY
4 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   PHYSICAL FILTER : PS_AVAILQTY > 5000
<<< end print plan

```

다음은 full outer join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, ps_availqty
FROM part FULL OUTER JOIN partsupp
ON p_partkey = ps_partkey
AND ps_availqty > 3000
AND p_size < 20;
P_NAME P_BRAND PS_AVAILQTY
-----
Part#1 Brand#1 8076
Part#1 Brand#1 3325
Part#2 Brand#1 4069
Part#2 Brand#1 3956
Part#3 Brand#2 null
Part#4 Brand#3 3025
Part#4 Brand#3 8539
Part#5 Brand#3 4093
Part#5 Brand#3 4651
null null 4969
null null 8895
11 rows selected.
>>> start print plan
< Execution Plan >

```

```

=====
| IDX | NODE DESCRIPTION | ROWS |

```

```

-----
|  0 | SELECT STATEMENT                                |      |
|  1 |   HASH JOIN (FULL OUTER JOIN)                    |      | 11 |
|  2 |     TABLE ACCESS ("PART")                        |      |  5 |
|  3 |       HASH JOIN INSTANT ACCESS                     |      | 11 |
|  4 |         TABLE ACCESS ("PARTSUPP")                 |      | 10 |
=====
1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PARTSUPP.PS_AVAILQTY
   JOIN FILTER : {PS_AVAILQTY} > 3000 AND {P_SIZE} < 20
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_SIZE
3 - INDEX COLUMNS : PS_PARTKEY
   TABLE COLUMNS : PS_AVAILQTY
   READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   HASH FILTER : {P_PARTKEY} = PS_PARTKEY
4 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
<<< end print plan

```

Semi Join

Semi join은 join 조건을 만족하는 outer node의 row만을 결과로 반환하므로 join 조건이 반드시 있어야 한다. Semi join은 outer node의 각 row들에 대해 join 조건을 만족하는 inner node row들이 있을 경우 outer node의 row만을 join 결과로 반환한다.

Semi join은 SQL 구문으로 직접 기술할 수 없으며, query optimizer가 IN, EXISTS 연산자와 = ANY와 같은 ANY형 quantify 연산자들을 semi join 형태로 변환한다.

Semi join에는 inverted 형태가 있는데 이것은 nested loops join과 hash join에서 지원한다.

Nested loops join에서의 inverted semi join은 outer node에 join 조건에 대한 index가 존재한다. 따라서 inner node의 row들을 unique가 보장되는 형태로 읽어와서 outer node에서 join 조건을 만족하는 결과를 찾아 반환한다. 이 방법은 outer node의 row가 많고 join 조건에 대한 index가 존재하며, inner node의 row가 적은 경우에 사용할 수 있다. Inner node의 row들은 unique가 보장되어야 하는데 이를 위해 sort instant를 사용한다. Nested loops join을 이용한 inverted semi join은 적은 수의 inner node row를 이용하여 많은 수의 row가 존재하는 outer node에 index access한 join 결과를 반환한다. 따라서 일반적인 nested loops join을 이용한 semi join보다 성능이 향상된다.

Hash join에서의 inverted semi join은 outer node를 hash instant로 생성하고 inner node의 row들을 읽어서 hash instant에서 join 조건을 만족하는 row들 중에 아직 결과로 반환하지 않은 row들을 결과로써 반환한다. 이 방법은 outer node의 row가 적고 inner node의 row가 많은 경우에 사용할 수 있다. 이는 적은 개수를 갖는 outer node의 row들을 hash instant로 만들고 inner node에서 많은 수의 row를 읽어 hash instant를 탐색한 후 join 결과를 반환한다. 따라서 inner node에 hash instant를 만드는 hash join을 이용한 semi join보다 hash instant 생성비용이 감소하는 등 성능이 향상된다.

다음은 semi join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand
  FROM part
 WHERE p_partkey IN ( SELECT ps_partkey
                      FROM partsupp
                      WHERE ps_availqty > 5000 );

P_NAME P_BRAND
-----
Part#1 Brand#1
Part#3 Brand#2
Part#4 Brand#3
3 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    HASH JOIN (INVERTED LEFT SEMI)                 |    3    |
|   2   |      TABLE ACCESS ("PARTSUPP")                   |    3    |
|   3   |        HASH JOIN INSTANT ACCESS                   |    3    |
|   4   |          TABLE ACCESS ("PART")                   |    5    |
=====

1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND
2 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   PHYSICAL FILTER : PS_AVAILQTY > 5000
3 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME, P_BRAND
   READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
4 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND
<<< end print plan

```

Anti-semi Join

Anti-semi join은 outer node의 각 row들에 대하여 inner node에 join 조건을 만족하는 row가 하나도 없을 때 outer node의 row만을 join 결과로 반환한다. 따라서 join 조건이 반드시 존재해야 한다.

Anti-semi join은 SQL 구문으로 직접 기술할 수 없으며, query optimizer가 NOT IN, NOT EXISTS 연산자와 = ALL 과 같은 ALL형 quantify 연산자들을 anti-semi join 형태로 변환한다.

Semi-join과 달리 anti-semi join은 NULL 데이터가 존재하는 경우 NULL 데이터를 별도로 처리해야 한다. 이는 NUL

L 데이터의 비교연산 결과 TRUE나 FALSE가 아닌 UNKNOWN이 발생하기 때문인데 anti-semi join의 join 조건에 대해 NULL 데이터가 없는 것이 보장되는 경우 query optimizer는 anti-semi join을 수행하며, 그렇지 않은 경우 NULL 데이터를 고려하여 null-aware anti-semi join을 수행한다.

다음은 join 조건에 대해 NULL 데이터가 없음을 보장하는 anti-semi join의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand
  FROM part
 WHERE p_partkey NOT IN ( SELECT ps_partkey
                          FROM partsupp
                          WHERE ps_availqty > 5000 );

P_NAME P_BRAND
-----
Part#2 Brand#1
Part#5 Brand#3
2 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    HASH JOIN (LEFT ANTI SEMI)                     |        2 |
|   2   |      TABLE ACCESS ("PART")                       |         5 |
|   3   |        HASH JOIN INSTANT ACCESS (UNIQUE)           |         2 |
|   4   |          TABLE ACCESS ("PARTSUPP")                |         3 |
=====

1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND
3 - INDEX COLUMNS : PS_PARTKEY
   HASH FILTER : {P_PARTKEY} = PS_PARTKEY
4 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   PHYSICAL FILTER : PS_AVAILQTY > 5000
<<< end print plan
```

다음은 join 조건에 NULL 데이터가 없음을 보장하지 않는 anti-semi join의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand
```

```

FROM part
WHERE p_partkey NOT IN ( SELECT l_partkey
                        FROM lineitem
                        WHERE l_quantity > 30 );

```

```
P_NAME P_BRAND
```

```
-----
```

```
Part#2 Brand#1
```

```
1 row selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	HASH JOIN (LEFT ANTI SEMI NA)	1
2	TABLE ACCESS ("PART")	5
3	HASH JOIN INSTANT ACCESS (UNIQUE)	1
4	TABLE ACCESS ("LINEITEM")	5

```
=====
```

```
1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND
```

```
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND
```

```
3 - INDEX COLUMNS : L_PARTKEY
```

```
    HASH FILTER : {P_PARTKEY} = L_PARTKEY
```

```
4 - READ COLUMNS : L_PARTKEY, L_QUANTITY
```

```
    PHYSICAL FILTER : L_QUANTITY > 30
```

```
<<< end print plan
```

Join Method

Join method는 두 테이블 (또는 view)의 join 연산 방법으로써 nested loops join과 sort merge join, hash join이 있다. Query optimizer는 join 연산을 할 때 이 세 가지 join methods에 대한 cost를 계산하여 가장 적은 비용의 join 연산을 선택한다.

Nested Loops Join

Nested loops join은 가장 기본적인 join으로써 outer node와 inner node에 별도로 instant 등을 생성하지 않고 join을 수행한다. Query optimizer는 다음과 같은 경우에 nested loops join을 사용한다.

- Join 조건이 없는 경우
- Join 조건에 equi-join 조건이 없는 경우
- Inner node 쪽에 index와 같은 효율적인 access 방법이 존재하는 경우
- USE_NL 힌트를 명시한 경우

Nested loops join은 대개 join 조건에 따라 inner node가 index access를 사용하는 경우 성능이 향상되며, query optimizer가 cost를 계산한 결과 nested loops join이 좋은 경우 이를 선택한다.

Nested loops join에는 기본적인 형태 외에 instant를 사용하는 확장된 형태의 nested loops join이 있다. 이는 inner node 쪽에 sort instant를 생성하여 join 조건에 index access가 있는 것과 비슷한 효과를 주는데 inner node에 대해 sort instant를 생성하는 비용이 추가되긴 하지만 join 조건을 만족하는 inner node의 row를 검색하는 비용이 더 많이 감소되어 전체적인 join 비용이 감소하는 경우에 사용된다.

다음은 nested loops join의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT /*+ USE_NL(part, partsupp) */
      p_name, p_brand, p_type
FROM part, partsupp
WHERE p_partkey = ps_partkey
      AND ps_availqty > 5000;
P_NAME P_BRAND    P_TYPE
-----
Part#1 Brand#1    COPPER
Part#3 Brand#2    STEEL
Part#4 Brand#3    NICKEL
3 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    NESTED LOOP JOIN (INNER JOIN)                  |         3 |
|   2   |      TABLE ACCESS ("PART")                        |         5 |
|   3   |        INDEX ACCESS ("PARTSUPP", PARTSUPP_PK_INDEX)|         3 |
=====

1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
2 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
3 - READ INDEX COLUMNS : PS_PARTKEY
   READ TABLE COLUMNS : PS_AVAILQTY
   MIN RANGE : PS_PARTKEY = {P_PARTKEY}
   MAX RANGE : PS_PARTKEY = {P_PARTKEY}
   PHYSICAL TABLE FILTER : PS_AVAILQTY > 5000
<<< end print plan
```


다음은 inner node에 sort instant를 이용한 확장된 형태의 nested loops join의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT /*+ USE_INL(part, lineitem) */
      p_name, p_brand, p_type
FROM part, lineitem
WHERE p_partkey = l_partkey
      AND l_quantity > 30;
P_NAME P_BRAND  P_TYPE
-----
Part#4 Brand#3  NICKEL
Part#5 Brand#3  STEEL
Part#1 Brand#1  COPPER
Part#4 Brand#3  NICKEL
Part#3 Brand#2  STEEL
5 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    NESTED LOOP JOIN (INNER JOIN)                  |         |
|   2   |      TABLE ACCESS ("LINEITEM")                   |         |
|   3   |        SORT INSTANT ACCESS                         |         |
|   4   |          TABLE ACCESS ("PART")                   |         |
=====

1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
2 - READ COLUMNS : L_PARTKEY, L_QUANTITY
   PHYSICAL FILTER : L_QUANTITY > 30
3 - SORT KEY : "PART.P_PARTKEY ASC NULLS LAST"
   RECORD COLUMNS : P_NAME, P_BRAND, P_TYPE
   READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
   MIN RANGE : P_PARTKEY = {L_PARTKEY}
   MAX RANGE : P_PARTKEY = {L_PARTKEY}
4 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
<<< end print plan
```

Sort Merge Join

Sort merge join은 join 조건을 만족하는 column으로 outer node와 inner node의 row들을 정렬한 다음 순차적으로 비교하며 join 결과를 반환한다. Outer node와 inner node에 join 조건을 만족하는 column에 대한 index가 존재하고 이를 이용할 수 있다면 해당 index를 그대로 사용하고 그렇지 않은 경우 sort instant를 사용하여 row들을 정렬한다.

Sort merge join은 정렬된 데이터의 outer node와 inner node를 순차적으로 읽어가며 join 조건을 비교하여 결과를 내기 때문에 outer node의 row 각각에 대해 inner node의 전체 row 중 join 조건과 일치하는 row를 찾아야 하는 nested loops join보다 일반적으로 성능이 더 좋다. 그러나 outer node와 inner node에 sort instant를 생성하고 row들을 정렬하는 비용이 큰 경우 오히려 성능이 저하될 수 있어 query optimizer가 각각의 cost를 계산하여 sort merge join의 비용이 더 작은 경우에 선택한다.

Sort merge join을 위해서는 join 조건에 하나 이상의 equi-join 조건을 반드시 포함해야 하고 equi-join 조건에 존재하는 column들을 기준으로 정렬한다. Equi-join 조건의 column들이 모두 존재하는 index가 있는 경우 index key의 각 column들 정렬순서 (ascending, descending)를 sort merge join에 사용 가능한 경우에만 사용한다. 예를 들어 I1, I2 column이 sort merge join에 사용되고 I1은 ascending, I2는 descending으로 서로 다른 정렬순서를 갖는 index가 있을 경우, 이 index는 사용할 수 없다.

다음은 sort merge join의 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT /*+ USE_MERGE(part, partsupp) */
      p_name, p_brand, p_type
FROM part, partsupp
WHERE p_partkey = ps_partkey
      AND ps_availqty > 5000;
P_NAME P_BRAND P_TYPE
```

```
-----
Part#1 Brand#1 COPPER
Part#3 Brand#2 STEEL
Part#4 Brand#3 NICKEL
```

3 rows selected.

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----|-----|-----|
|    0  |  SELECT STATEMENT                                |         |
|    1  |  SORT MERGE JOIN (INNER JOIN) : EQUAL              |         3 |
|    2  |  INDEX ACCESS ("PART, PART_PK_INDEX")              |         5 |
|    3  |  SORT JOIN INSTANT ACCESS                          |         3 |
```

```

| 4 |          TABLE ACCESS ("PARTSUPP")          | 3 |
=====
1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
    MERGE FILTER : PART.P_PARTKEY = PARTSUPP.PS_PARTKEY
2 - READ INDEX COLUMNS : P_PARTKEY
    READ TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
3 - SORT KEY : "PARTSUPP.PS_PARTKEY ASC NULLS LAST"
    READ COLUMNS : PS_PARTKEY
4 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
    PHYSICAL FILTER : PS_AVAILQTY > 5000
<<< end print plan

```

Hash Join

Hash join은 inner node에 hash instant를 생성하여 join 연산을 수행한다. Inner node에만 hash instant를 생성하면 되고, hash를 사용하여 join 조건을 비교하기 때문에 성능이 향상된다.

Hash join은 join 조건에 반드시 하나 이상의 equi-join 조건을 포함해야 한다. Equi-join 조건에 존재하는 column들을 hash key로 하는 hash instant를 생성하는 비용이 발생하지만 outer node의 각 row에 대하여 join 조건을 수행할 때 hash key를 이용하여 빠르게 탐색할 수 있기 때문에 inner node 쪽에 join 조건에 대한 index가 없는 경우, 다른 join 방법을 쓸 때에 비해 성능이 향상된다.

다음은 hash join의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT /*+ USE_HASH(part, partsupp) */
    p_name, p_brand, p_type
FROM part, partsupp
WHERE p_partkey = ps_partkey
    AND ps_availqty > 5000;
P_NAME P_BRAND    P_TYPE
-----
Part#1 Brand#1    COPPER
Part#3 Brand#2    STEEL
Part#4 Brand#3    NICKEL
3 rows selected.
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----
|    0  |  SELECT STATEMENT                                |         |

```

```

| 1 | HASH JOIN (INNER JOIN) | 3 |
| 2 | TABLE ACCESS ("PARTSUPP") | 3 |
| 3 | HASH JOIN INSTANT ACCESS | 3 |
| 4 | TABLE ACCESS ("PART") | 5 |
=====
1 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
2 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   PHYSICAL FILTER : PS_AVAILQTY > 5000
3 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
   READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
4 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
<<< end print plan

```

Join Concat

Join concat의 join 조건에는 or 구문이 존재하고 or에 의해 나뉘어진 join 조건들을 각각 별도의 join으로 처리한 결과를 하나로 통합한다. Join concat은 or에 의해 나뉘어진 join 조건들 각각에 대해 가장 좋은 plan을 선택하고 join concat에서 중복을 제거한 후 하나의 결과로 반환한다.

Join concat은 하위 노드로 join node를 갖는다는 점을 제외하면 index concat과 동일하다.

다음은 join concat의 예이다.

```

gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
FROM part, partsupp
WHERE ( p_partkey = ps_partkey AND ps_availqty > 5000 )
      OR ( p_partkey = ps_partkey AND ps_supplycost > 900 );
P_NAME P_BRAND P_TYPE
-----
Part#1 Brand#1 COPPER
Part#3 Brand#2 STEEL
Part#4 Brand#3 NICKEL
Part#3 Brand#2 STEEL
Part#5 Brand#3 STEEL
5 rows selected.
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |

```

```

-----
| 0 | SELECT STATEMENT                                |      |
| 1 |   CONCAT                                          |      | 5 |
| 2 |     HASH JOIN (INNER JOIN)                      |      | 3 |
| 3 |       TABLE ACCESS ("PARTSUPP")                |      | 3 |
| 4 |       HASH JOIN INSTANT ACCESS                  |      | 3 |
| 5 |         TABLE ACCESS ("PART")                  |      | 5 |
| 6 |         HASH JOIN (INNER JOIN)                  |      | 3 |
| 7 |           TABLE ACCESS ("PARTSUPP")            |      | 3 |
| 8 |           HASH JOIN INSTANT ACCESS               |      | 3 |
| 9 |             TABLE ACCESS ("PART")              |      | 5 |
=====
2 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
3 - READ COLUMNS : PS_PARTKEY, PS_AVAILQTY
   PHYSICAL FILTER : PS_AVAILQTY > 5000
4 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
   READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
5 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
6 - JOINED COLUMNS : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
7 - READ COLUMNS : PS_PARTKEY, PS_SUPPLYCOST
   PHYSICAL FILTER : PS_SUPPLYCOST > 900
8 - INDEX COLUMNS : P_PARTKEY
   TABLE COLUMNS : P_NAME, P_BRAND, P_TYPE
   READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
   HASH FILTER : P_PARTKEY = {PS_PARTKEY}
9 - READ COLUMNS : P_PARTKEY, P_NAME, P_BRAND, P_TYPE
<<< end print plan

```

Cluster

Cluster는 local server와 remote server에 존재하는 데이터들을 수집하여 하나의 결과 집합으로 취합한다. Cluster에는 single table에 대한 데이터들을 취합하는 cluster access와 두 개 이상의 table들을 join한 결과 데이터들을 취합하는 cluster join이 있다.

Cluster Access

Cluster access는 cluster 환경에서 local server와 remote server에 존재하는 테이블 데이터들을 가져온다. Cluster access는 single table 데이터를 취합하고 그 하위에는 위 **Access Paths**에 해당하는 access들 중 하나가 올 수 있다.

Cluster access는 remote server로부터 데이터를 가져와야 할 때만 필요하다. 따라서 cluster access는 standalone에는 나타나지 않고 cluster 환경에서 local server에서만 데이터를 가져오면 되는 경우에도 나타나지 않는다.

다음은 cluster access를 사용하는 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part
 WHERE p_size > 20;
P_NAME P_BRAND    P_TYPE
-----
Part#3 Brand#2    STEEL
1 row selected.
>>> start print plan
< Execution Plan >

=====
====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
----
|    0  |  SELECT STATEMENT                                |
|
|    1  |    CLUSTER ACCESS ("PART") [HASH SHARDING]        |
1 |
|    2  |      TABLE ACCESS ("PART") [HASH SHARDING]        |
1 |
=====
====
1 - SQL : SELECT /*+ FULL("_A1") */ "_A1"."P_NAME", "_A1"."P_BRAND", "_A1"."P_TYPE", "_A
1"."P_SIZE" FROM "PUBLIC"."PART"@LOCAL "_A1" WHERE "_A1"."P_SIZE" > ?
      BIND PARAMS : {0} IN
2 - READ COLUMNS : P_NAME, P_BRAND, P_TYPE, P_SIZE
      PHYSICAL FILTER : P_SIZE > 20
<<< end print plan
```

Cluster Join

Cluster join은 cluster 환경에서 local server와 remote server에 존재하는 join 결과 데이터들을 가져온다. Cluster join은 두 개 이상의 table들을 join한 결과 데이터를 취합하며 하위에는 위 **Join Method**에 해당하는 method들 중 하나가 올 수 있다.

Cluster join은 remote server로부터 데이터를 가져와야 하는 경우에만 필요하다. 따라서 cluster join은 stand alone에는 나타나지 않고 cluster 환경에서 local server에서만 데이터를 가져오면 되는 경우에도 나타나지 않는다.

다음은 cluster join을 사용하는 예이다.

```
gSQL>
\EXPLAIN PLAN
SELECT c_name, o_totalprice
  FROM orders, customer
 WHERE o_custkey = c_custkey
       AND c_nation = 'KOREA';
C_NAME      O_TOTALPRICE
-----
Customer#1   173665.47
Customer#3   32151.78
2 rows selected.
>>> start print plan
< Execution Plan >
=====
=====
|  IDX  |  NODE DESCRIPTION                                |
ROWS |
-----
|    0  |  SELECT STATEMENT                                |
|      |  |
|    1  |  CLUSTER JOIN                                    |
2 |
|    2  |  NESTED LOOP JOIN (INNER JOIN)                   |
2 |
|    3  |  TABLE ACCESS ("CUSTOMER") [CLONED]              |
2 |
|    4  |  INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK") [HASH SHARDING] | (    2)
2 |
=====
=====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE FULL("_A1") USE_NL("_A1") INDEX_ASC("_A2", "
```

```

ORDERS_CUSTKEY_FK") USE_NL("_A2") */ "_A1"."C_NAME", "_A2"."O_TOTALPRICE" FROM "PUBLIC"."
CUSTOMER"@LOCAL "_A1" INNER JOIN "PUBLIC"."ORDERS"@LOCAL "_A2" ON "_A1"."C_NATION" = ? AND "_A
2"."O_CUSTKEY" = "_A1"."C_CUSTKEY"

      BIND PARAMS : {0} IN
2  -  JOINED COLUMNS : CUSTOMER.C_NAME, ORDERS.O_TOTALPRICE
3  -  READ COLUMNS : C_CUSTKEY, C_NAME, C_NATION
      PHYSICAL FILTER : C_NATION = 'KOREA'
4  -  READ INDEX COLUMNS : O_CUSTKEY
      READ TABLE COLUMNS : O_TOTALPRICE
      MIN RANGE : O_CUSTKEY = {C_CUSTKEY}
      MAX RANGE : O_CUSTKEY = {C_CUSTKEY}

<<< end print plan

```

통계 정보

Query optimizer는 통계 정보를 사용하여 cost를 계산한다. Query optimizer가 사용하는 통계 정보에는 테이블 통계 정보와 column 통계 정보, 인덱스 통계 정보 등이 있다.

- 테이블 통계 정보
 - Row 개수
- Column 통계 정보
 - 서로 다른 값의 개수
 - NULL 값의 개수
 - 값의 평균 길이
 - 최소값
 - 최대값
- 인덱스 통계 정보
 - 서로 다른 key의 개수

통계 정보를 구축하기 위해 **ANALYZE TABLE** 구문을 수행한다. 구축된 통계 정보는 데이터베이스에 저장되며 통계 정보를 재구축하기 전까지 동일한 통계 정보가 사용된다.

통계 정보가 구축되지 않은 테이블에 대해서는 카탈로그 정보와 질의 수행 시점의 페이지 정보를 이용하여 간단한 통계정보를 구축한 후 이용한다.

Optimizer 조정

일반적으로 query optimizer는 주어진 통계 정보를 이용하여 가장 효율적인 plan을 선택한다. 그러나 query optimizer가 선택한 plan보다 더 좋은 plan이 존재할 수 있으며, query optimizer가 이를 선택하지 못하는 경우 사용자가

해당 plan을 사용하도록 지정할 수 있다.

현재 GOLDBLOCKS의 query optimizer는 힌트를 제공하고 있으며, 적용 가능한 힌트가 기술된 경우 계산된 cost와 관계없이 사용자가 기술한 힌트를 우선 적용한다. 따라서 더 좋은 plan이 존재하는 경우 사용자가 힌트를 사용하여 plan을 강제로 변경할 수 있다.

힌트에 대한 자세한 내용은 **hint clause**를 참조한다.

15.4 SQL 실행 계획

개요

SQL 실행 계획은 query optimizer가 여러 candidate plan 중에 가장 좋은 plan으로 선택한 plan을 실제로 수행하기 위한 형태로 구성한 것이다. SQL 실행 계획은 실행을 위한 node 단위로 구분되며, 각 node는 자신이 수행해야 하는 filter를 갖는다.

SQL 실행 계획의 최상위 노드는 INSERT, DELETE, UPDATE, SELECT statement로 분류된다. 최상위 노드를 기준으로 각각의 노드가 tree 형태로 구성되어 있으며, 실제로는 왼쪽 최하위 노드부터 수행된다.

SQL 실행 계획의 tree에서는 다음 정보들을 확인할 수 있다.

- Statement에서 참조하는 테이블들의 순서
- 각각의 테이블에 대한 access path
- Join을 처리하는 node에서의 join method
- 정렬 또는 grouping, aggregation, filter 등의 정보

일반적으로 동일한 SQL 구문은 동일한 형태의 SQL 실행 계획을 갖지만 다음과 같은 경우에는 다른 형태의 SQL 실행 계획을 가질 수 있다.

- 서로 다른 schema에 동일한 이름의 테이블이 있는 상태에서 각각 다른 schema에 대한 질의를 수행한 경우
- 이전 질의를 수행한 후에 index 추가/삭제와 같은 schema 변경이 발생한 경우
- 이전 질의를 수행한 후에 데이터 추가/삭제 등으로 인해 통계 정보가 변경되었고 이를 이용해 query optimizer가 다른 plan을 더 좋은 plan으로 선택하여 그것이 수행된 경우

출력

구문

```
<explain plan> ::=
    \explain plan [ on | only ] <sql statement>
<sql statement> ::=
    <query expression>
    | <select for update statement>
    | <select statement: single row>
    | <insert statement>
    | <update statement: searched>
    | <delete statement: searched>
```

사용 범위 및 접근 권한

⟨explain plan⟩ 구문을 수행하려면 ⟨sql statement⟩ 구문에 대한 접근 권한을 가져야 한다.

구문 규칙 및 파라미터

⟨explain plan⟩

- \explain plan on
 - ⟨sql statement⟩에 대한 execution을 수행하여 질의 결과도 함께 출력한다.
- \explain plan only
 - ⟨sql statement⟩에 대한 execution을 수행하지 않는다.
- \explain plan
 - \explain plan on과 같다.

⟨sql statement⟩

⟨sql statement⟩는 실행 계획을 출력할 대상 질의이다.

설명

⟨sql statement⟩에 기술된 SELECT 및 DML 구문의 실행 계획을 출력한다.

사용 예

다음과 같이 ON과 함께 사용할 경우 SQL 구문을 수행하고 질의 결과를 실행 계획과 함께 출력한다.

```
gSQL> \explain plan on SELECT id, name FROM t1 ORDER BY 1;
ID NAME
-- -----
1 leekmo
2 jhkim
3 bsyou
3 rows selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION  |  ROWS  |
-----|-----|-----|
|    0  |  SELECT STATEMENT  |         |
|    1  |  SORT INSTANT ACCESS |         3 |
```

```

| 2 | TABLE ACCESS ("T1") | 3 |
=====
1 - SORT KEY : "T1.ID ASC NULLS LAST"
   RECORD COLUMNS : NAME
   READ COLUMNS : ID, NAME
2 - READ COLUMNS : ID, NAME
<<< end print plan

```

다음과 같이 ON이나 ONLY를 생략하였을 경우 SQL 구문을 수행하고 질의 결과를 실행 계획과 함께 출력한다.

```

gSQL> \explain plan SELECT id, name FROM t1 ORDER BY 1;
ID NAME
-- -----
1 leekmo
2 jhkim
3 bsyu
3 rows selected.
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |
-----|-----|-----
| 0 | SELECT STATEMENT |      |
| 1 | SORT INSTANT ACCESS |      3 |
| 2 | TABLE ACCESS ("T1") |      3 |
=====
1 - SORT KEY : "T1.ID ASC NULLS LAST"
   RECORD COLUMNS : NAME
   READ COLUMNS : ID, NAME
2 - READ COLUMNS : ID, NAME
<<< end print plan

```

다음과 같이 ONLY를 사용할 경우 SQL 구문을 수행하지 않고, 질의 결과 없이 실행 계획만 출력한다.

```

gSQL> \explain plan only SELECT id, name FROM t1 ORDER BY 1;
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |
-----|-----|-----
| 0 | SELECT STATEMENT |      |
| 1 | SORT INSTANT ACCESS |      0 |

```

```

| 2 | TABLE ACCESS ("T1") | 0 |
=====
1 - SORT KEY : "T1.ID ASC NULLS LAST"
   RECORD COLUMNS : NAME
   READ COLUMNS : ID, NAME
2 - READ COLUMNS : ID, NAME
<<< end print plan

```

읽기

SQL 실행 계획 구성

SQL 실행 계획은 query optimizer에 의해 결정되며 실제 질의 처리를 위한 노드로 구성되어 있다. 이러한 SQL 실행 계획은 EXPLAIN PLAN 명령을 사용하여 다음과 같이 출력할 수 있다.

```

gSQL> \explain plan SELECT id, name FROM t1;
ID NAME
-- -----
1 leekmo
2 jhkim
3 bsyou
3 rows selected.
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |
-----|-----|-----
| 0 | SELECT STATEMENT |      |
| 1 | TABLE ACCESS ("T1") | 3 |
=====
1 - READ COLUMNS : ID, NAME
<<< end print plan

```

위 예에서 SQL 실행 계획을 출력하기 위하여 SQL 구문 앞에 \explain plan 명령을 명시하였다. 해당 명령의 수행 결과가 >>> start print plan으로 시작되어 <<< end print plan으로 종료되는 사이에 실행 계획이 출력되었다.

SQL 실행 계획의 출력은 노드 이름과 수행된 row 수를 테이블 형태로 출력하는 execution plan node table과 각 execution plan node의 상세 정보를 출력하는 node information으로 나누어진다.

Execution Plan Node Table

테이블 t1의 id와 name을 검색하는 질의에 대한 **SQL 실행 계획 구성** 중 execution plan node table에 대한 부분은 다음과 같다.

< Execution Plan >

=====		
IDX	NODE DESCRIPTION	ROWS

0	SELECT STATEMENT	
1	TABLE ACCESS ("T1")	3
=====		

위 예에서 execution plan node table은 IDX와 NODE DESCRIPTION, ROWS의 column들로 구성되어 있고 각 노드는 0부터 시작되는 고유한 식별번호를 갖는다. 또한 각 노드들의 tree 형태 구성을 노드 이름 앞의 공백으로 구분할 수 있도록 되어 있다. 즉, 위 예에서 SELECT STATEMENT는 TABLE ACCESS를 child node로 갖는다.

Execution plan node table의 각 column에 대한 정보는 다음과 같다.

- IDX
 - 각 plan node에 부여된 식별자이다.
- NODE DESCRIPTION
 - Plan node의 이름이다.
 - 괄호 안의 내용은 plan node를 구분하는 부가적인 정보이다.
 - 들여쓰기로 표시된 plan node는 하위 plan node를 의미한다.
 - 하위 plan node부터 수행하여 상위 노드에 그 결과를 전달한다.
- ROWS
 - Plan node 수행에 따른 결과 레코드의 개수이다.

위 execution plan node table을 분석한 결과는 다음과 같다.

- T1 테이블에 TABLE ACCESS를 수행하여 세 개의 결과 record를 얻었고, SELECT STATEMENT를 입력하여 이를 전달한다.

Node Information

테이블 t1의 id와 name을 검색하는 질의에 대한 **SQL 실행 계획 구성** 중 node information에 대한 부분은 다음과 같다.

1 - READ COLUMNS : ID, NAME

Node information은 각 노드에 출력할 정보가 있을 때 출력되고 노드를 구분하기 위해 앞부분에 execution plan node table의 IDX를 출력한다. 뒷부분에는 출력 정보에 대한 분류명과 해당 분류에 대한 정보들이 나타나며, 여러 분류의 출력 정보가 필요한 경우 각 정보별로 하나의 줄에 출력한다.

위 node information을 분석한 결과는 다음과 같다.

- T1 테이블에 TABLE ACCESS를 수행할 때 ID와 NAME을 read column 대상으로 한다.

Execution Plan Node 분류

Execution plan node는 크게 <statement node>, <access node>, <join node>, <instant node>, <aggregation node>, <SET operator node>, <subquery node>, <filter node>, <cluster node>, <other node>로 분류되며, 각각의 노드는 다음 표와 같다.

표 15-2 Execution plan node의 분류

Node	참조
Statement node	DELETE STATEMENT
	INSERT STATEMENT
	SELECT STATEMENT
	UPDATE STATEMENT
Access node	INDEX ACCESS (table_name [AS alias], index_name)
	TABLE ACCESS (table_name [AS alias])
	USER ROWID ACCESS (table_name [AS alias])
Join node	HASH JOIN (join_method)
	NESTED LOOP JOIN (join_method)
	SORT MERGE JOIN (join_method) : EQUAL
Instant node	GROUP HASH INSTANT ACCESS
	HASH JOIN INSTANT ACCESS
	HASH JOIN INSTANT ACCESS (UNIQUE)
	SORT INSTANT ACCESS
	SORT INSTANT ACCESS (UNIQUE)
	SORT JOIN INSTANT ACCESS
	SORT JOIN INSTANT ACCESS (UNIQUE)
Aggregation node	HASH AGGREGATION
SET operator node	EXCEPT ALL
	EXCEPT DISTINCT
	INTERSECT ALL
	INTERSECT DISTINCT
	UNION ALL
	UNION DISTINCT
Subquery node	SUB QUERY FUNCTION
	SUB QUERY FUNCTION (MATERIALIZED)
	SUB QUERY LIST
Filter node	FILTER
Cluster node	CLUSTER ACCESS (table_name) [sharding_strategy]

Node	참조
	CLUSTER JOIN
Other node	CONCAT
	DELETE (table_name)
	INSERT (table_name)
	UPDATE (table_name)
	VIEW
	VIEW (view_name)
	GROUP

Node Information 분류

Node information은 크게 <column information>, <filter information>, <aggregation information>으로 분류되며, 각각의 node information은 다음 표와 같다.

표 15-3 Node information 분류

Node information	이름	설명
Column information	READ COLUMNS COLUMNS	현재 노드에서 결과로 보낼 column list
	INDEX COLUMNS READ INDEX COLUMNS	Index에서 참조할 column list
	TABLE COLUMNS READ TABLE COLUMNS	Table에서 참조할 column list
	JOINED COLUMNS	Join 결과로 보낼 column list
	SORT KEY	Sort instant에서 sort key로 사용할 column list
	RECORD COLUMNS	Sort instant에서 sort key를 제외한 나머지 column list
	GROUPING COLUMNS	Grouping key로 사용할 column list
Filter information	PHYSICAL FILTER	Type casting이 필요 없는 비교 연산에 대한 AND-filter
	LOGICAL FILTER	Physical filter를 제외한 AND-filter
	JOIN FILTER	Join 조건에 대한 filter
	WHERE FILTER	Join 연산으로 구성된 row에 적용할 filter
	HASH FILTER	Hash key를 이용하여 처리하는 filter
	PHYSICAL TABLE FILTER	Table filter 중 type casting이 필요 없는 비교 연산에 대한 AND-filter
	LOGICAL TABLE FILTER	Table filter 중 physical filter를 제외한 AND-filter
	MIN RANGE	Index의 min key range
	MAX RANGE	Index의 max key range
	PHYSICAL KEY FILTER	Index filter 중 type casting이 필요 없는 비교 연산에 대한 AND-filter
	LOGICAL KEY FILTER	Index filter 중 physical filter를 제외한 AND-filter

Node information	이름	설명
	MERGE FILTER	Sort merge join에서 merge join에 사용하는 filter
	SUBQUERY FILTER	Subquery를 처리하기 위한 subquery AND-filter
	FUNCTION	Subquery를 처리하기 위한 function expression
	ROWID ACCESS EXPR	Rowid와 관련된 filter
	NODE FILTER	처음 실행할 때 한 번만 평가하는 filter
	FILTER	매 row마다 평가하는 filter
Aggregation information	AGGREGATIONS	일반 aggregation list
	NESTED AGGREGATIONS	중첩 aggregation list
Other information	NODE EXPR	처음 한 번만 수행하는 expression
	SQL	Remote server에 보낼 SQL 질의 구문
	BIND PARAMS	위 SQL 질의 구문에서 사용된 bind parameter
	REFERENCE SHARD KEY VALUE	Table의 shard key로 지정된 column에 대한 '=' filter의 value 값

Execution Plan Node Reference

CLUSTER ACCESS (table_name) [sharding_strategy]

- Cluster 환경에서 local server와 remote server의 access node 결과를 취합한다.
- Remote server에서 결과를 받아야 하는 경우에만 생성된다.
- Node information
 - SQL: Remote server에 보낼 SQL 질의 구문
 - BIND PARAMS: 위 SQL 질의 구문에서 사용된 bind parameter
 - REFERENCE SHARD KEY VALUE: Table의 shard key로 지정된 column의 '=' filter value 값

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT l_orderkey
      FROM lineitem
      WHERE l_orderkey = 1
            AND l_linenumber = 2;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
=====
|  IDX  |  NODE DESCRIPTION                                     |
ROWS |
```

```

-----
|      0 | SELECT STATEMENT                                     |
|
|      1 | CLUSTER ACCESS ("LINEITEM") [HASH SHARDING]               |
0 |
|      2 | INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX") [HASH SHARDING] |
0 |
=====
=====
1 - SQL : SELECT /*+ INDEX_ASC("_A1", "LINEITEM_PK_INDEX") */ "_A1"."L_ORDERKEY", "_A
1"."L_LINENUMBER" FROM "PUBLIC"."LINEITEM"@LOCAL "_A1" WHERE "_A1"."L_ORDERKEY" = ? AND "_A
1"."L_LINENUMBER" = ?
      BIND PARAMS : {0} IN {1} IN
      REFERENCE SHARD KEY VALUE : (1, 2)
2 - READ INDEX COLUMNS : L_ORDERKEY, L_LINENUMBER
      MIN RANGE : L_ORDERKEY = 1 AND L_LINENUMBER = 2
      MAX RANGE : L_ORDERKEY = 1 AND L_LINENUMBER = 2
<<< end print plan

```

CLUSTER JOIN

- Cluster 환경에서 local server와 remote server의 join node 결과를 취합한다.
- Remote server로부터 결과를 받아야 하는 경우에만 생성된다.
- Node information
 - SQL: Remote server에 보낼 SQL 질의 구문
 - BIND PARAMS: 위 SQL 질의 구문에서 사용된 bind parameter
 - REFERENCE SHARD KEY VALUE: Table의 shard key로 지정된 column의 '=' filter value 값

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT c_name, o_totalprice
      FROM orders, customer
      WHERE o_custkey = c_custkey
            AND o_orderkey = 1;
2      3      4      5
>>> start print plan
< Execution Plan >
=====
=====
|  IDX  |  NODE DESCRIPTION                                     |
ROWS |

```

```

-----
| 0 | SELECT STATEMENT |
|
| 1 | CLUSTER JOIN |
0 |
| 2 | NESTED LOOP JOIN (INNER JOIN) |
0 |
| 3 | INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX") [HASH SHARDING] |
0 |
| 4 | INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX") [CLONED] |
0 |
=====
=====
1 - SQL : SELECT /*+ KEEP_JOINED_TABLE INDEX_ASC("_A1", "ORDERS_PK_INDEX") USE_NL("_A
1") INDEX_ASC("_A2", "CUSTOMER_PK_INDEX") USE_NL("_A2") */ "_A2"."C_NAME", "_A1"."O_TOTALPRICE"
FROM "PUBLIC"."ORDERS"@LOCAL "_A1" INNER JOIN "PUBLIC"."CUSTOMER"@LOCAL "_A2" ON "_A1"."O_
ORDERKEY" = ? AND "_A2"."C_CUSTKEY" = "_A1"."O_CUSTKEY"
      BIND PARAMS : {0} IN
      REFERENCE SHARD KEY VALUE (ORDERS) : (1)
2 - JOINED COLUMNS : CUSTOMER.C_NAME, ORDERS.O_TOTALPRICE
3 - READ INDEX COLUMNS : O_ORDERKEY
      READ TABLE COLUMNS : O_CUSTKEY, O_TOTALPRICE
      MIN RANGE : O_ORDERKEY = 1
      MAX RANGE : O_ORDERKEY = 1
4 - READ INDEX COLUMNS : C_CUSTKEY
      READ TABLE COLUMNS : C_NAME
      MIN RANGE : C_CUSTKEY = {O_CUSTKEY}
      MAX RANGE : C_CUSTKEY = {O_CUSTKEY}
<<< end print plan

```

CONCAT

- 하위 노드들의 결과를 병합한다.
- Index access 또는 join에 대한 where 절에 or 연산이 있는 경우 생성된다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT s_suppkey FROM supplier WHERE s_suppkey = 1 or s_suppkey = 2;
>>> start print plan
< Execution Plan >

```

```

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0   |  SELECT STATEMENT                                |         |
|    1   |    CONCAT                                         |         0 |
|    2   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
|    3   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
=====

      2 - READ INDEX COLUMNS : S_SUPPKEY
          MIN RANGE : S_SUPPKEY = 1
          MAX RANGE : S_SUPPKEY = 1
      3 - READ INDEX COLUMNS : S_SUPPKEY
          MIN RANGE : S_SUPPKEY = 2
          MAX RANGE : S_SUPPKEY = 2

<<< end print plan

```

DELETE STATEMENT

- DELETE 구문을 수행한다.
- **DELETE (table_name)** node를 하위 노드로 갖는다.

다음 예제를 참조한다.

```

gSQL> \explain plan only DELETE FROM supplier;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0   |  DELETE STATEMENT                                |         |
|    1   |    DELETE ("SUPPLIER")                             |         0 |
|    2   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
=====

      2 - READ INDEX COLUMNS : S_SUPPKEY

<<< end print plan

```

DELETE (table_name)

해당 table에 DELETE 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only DELETE FROM supplier;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

IDX	NODE DESCRIPTION	ROWS
0	DELETE STATEMENT	
1	DELETE ("SUPPLIER")	0
2	INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX")	0

```
=====
```

```
2 - READ INDEX COLUMNS : S_SUPPKEY
```

```
<<< end print plan
```

EXCEPT ALL

IDX 순서를 기준으로 하위 노드의 EXCEPT ALL 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
```

```
SELECT s_suppkey FROM supplier
```

```
EXCEPT ALL
```

```
SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
```

```
EXCEPT ALL
```

```
SELECT ps_suppkey FROM partsupp;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	EXCEPT-ALL	0
2	INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX")	0
3	TABLE ACCESS ("LINEITEM")	0
4	INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX")	0

```
=====
```

```
2 - READ INDEX COLUMNS : S_SUPPKEY
```

```
3 - READ COLUMNS : L_SUPPKEY
```

```
4 - READ INDEX COLUMNS : PS_SUPPKEY
```

```
<<< end print plan
```

EXCEPT DISTINCT

IDX 순서를 기준으로 하위 노드의 EXCEPT DISTINCT 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT s_suppkey FROM supplier
      EXCEPT DISTINCT
      SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
      EXCEPT DISTINCT
      SELECT ps_suppkey FROM partsupp;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    EXCEPT-DISTINCT                             |         0 |
|   2   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
|   3   |      TABLE ACCESS ("LINEITEM")                  |         0 |
|   4   |      INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") |         0 |
=====

      2 - READ INDEX COLUMNS : S_SUPPKEY
      3 - READ COLUMNS : L_SUPPKEY
      4 - READ INDEX COLUMNS : PS_SUPPKEY
<<< end print plan
```

FILTER

- 하위 subquery 결과에 대한 filter를 처리하거나 node expression, node filter를 처리한다.
- Node information
 - READ COLUMNS: Filter로부터 읽어들이 column list 이다.
 - NODE EXPR: 처음 한 번만 수행하는 expression 이다.
 - NODE FILTER: 처음 한 번만 평가하는 filter 이다.
 - FILTER: 매 row마다 평가하는 filter 이다.
 - SUBQUERY FILTER: Subquery AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT l_orderkey
      FROM lineitem
```

```

WHERE 1 = 1
AND l_linenumber = 1 + random(1,1)
AND l_quantity * l_discount >= 10 + random(1,1)
AND l_orderkey in
( SELECT /*+ NO_QUERY_TRANSFORMATION */ o_orderkey
  FROM orders );

```

```
>>> start print plan
```

```
< Execution Plan >
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	FILTER	0
2	TABLE ACCESS ("LINEITEM")	0
3	SUB QUERY LIST	0
4	SUB QUERY FUNCTION (MATERIALIZED)	0
5	INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")	0

```

1 - READ COLUMNS : LINEITEM.L_ORDERKEY
   NODE EXPR : 1 + RANDOM(1,1)
   NODE FILTER : 1 = 1
   FILTER : ( LINEITEM.L_QUANTITY * LINEITEM.L_DISCOUNT ) >= ( 10 + RANDOM(1,1) )
   SUBQUERY FILTER : ( LINEITEM.L_ORDERKEY ) IN ( ORDERS.O_ORDERKEY )
2 - READ COLUMNS : L_ORDERKEY, L_LINENUMBER
   PHYSICAL FILTER : L_LINENUMBER = ?
4 - FUNCTION : ( L_ORDERKEY ) IN ( O_ORDERKEY )
5 - READ INDEX COLUMNS : O_ORDERKEY

```

```
<<< end print plan
```

GROUP

- Materialization이 필요없는 grouping 연산을 수행한다.
- SELECT 구문에 GROUP BY 절이 오거나 DISTINCT 절이 오는 경우에 생성된다.
 - 단, grouping key에 대한 uniqueness 또는 sort order가 보장되는 경우에 한하여 지원된다.
 - 이 외의 경우에 대해서는 GROUP HASH INSTANT ACCESS를 통해 grouping을 지원한다.
- Node information
 - AGGREGATIONS
 - Grouping 중에 처리할 중첩되지 않은 aggregation list 이다.
 - Grouping을 수행한다.
 - GROUPING COLUMNS: Grouping key로 구성될 column list 이다.
 - RECORD COLUMNS: Grouping key는 아니지만 참조하기 위해 추가된 column list 이다.

- READ COLUMNS: Grouping 결과로 읽어들이 column list 이다.
- LOGICAL FILTER: 전체 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only SELECT /*+ INDEX( lineitem ) */ max( sum( l_quantity ) )
      FROM lineitem
      GROUP BY l_orderkey
      HAVING sum( l_quantity ) > 10 and mod( sum( l_quantity ), 10 ) = 1;2 3 4
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    HASH AGGREGATION                              |    0   |
|   2   |      GROUP                                        |    0   |
|   3   |        INDEX ACCESS("LINEITEM, LINEITEM_PK_INDEX")|    0   |
=====
1 - AGGREGATIONS : MAX( SUM( L_QUANTITY ) )
2 - AGGREGATIONS : SUM( L_QUANTITY )
   GROUPING COLUMNS : L_ORDERKEY
   RECORD COLUMNS : SUM( L_QUANTITY )
   READ COLUMNS : SUM( L_QUANTITY )
   LOGICAL FILTER : MOD(SUM( L_QUANTITY ),10) = 1 AND SUM( L_QUANTITY ) > 10
3 - READ INDEX COLUMNS : L_ORDERKEY
   READ TABLE COLUMNS : L_QUANTITY
<<< end print plan
```

GROUP HASH INSTANT ACCESS

- Grouping 연산을 수행한다.
- SELECT 구문에 GROUP BY 절이 오거나 DISTINCT 절이 오는 경우에 생성된다.
- Node information
 - AGGREGATIONS: Grouping 중에 처리할 중첩되지 않은 aggregation list 이다.
 - NESTED AGGREGATIONS
 - Grouping 중에 처리할 중첩된 aggregation list 이다.
 - Grouping을 수행한다.
 - GROUPING COLUMNS: Grouping key로 구성될 instant의 column list 이다.
 - RECORD COLUMNS: Grouping key는 아니지만 참조하기 위해 추가된 instant의 column list 이다.
 - READ COLUMNS: Instant로부터 읽어들이 column list 이다.
 - PHYSICAL FILTER: Type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.

- LOGICAL FILTER: 전체 FILTER에서 PHYSICAL FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only SELECT /*+ FULL( lineitem ) */ max( sum( l_quantity ) )
      FROM lineitem
      GROUP BY l_orderkey
      HAVING sum( l_quantity ) > 10 and mod( sum( l_quantity ), 10 ) = 1;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |      ROWS |
=====
|   0   |  SELECT STATEMENT                                |           |
|   1   |    GROUP HASH INSTANT ACCESS                      |          0 |
|   2   |      TABLE ACCESS ("LINEITEM")                  |          0 |
=====

1 - AGGREGATIONS : SUM( L_QUANTITY )
    NESTED AGGREGATIONS : MAX( SUM( L_QUANTITY ) )
    GROUPING COLUMNS : L_ORDERKEY
    RECORD COLUMNS : SUM( L_QUANTITY )
    READ COLUMNS : SUM( L_QUANTITY )
    PHYSICAL FILTER : SUM( L_QUANTITY ) > 10
    LOGICAL FILTER : MOD(SUM( L_QUANTITY ),10) = 1
2 - READ COLUMNS : L_ORDERKEY, L_QUANTITY
<<< end print plan
```

HASH AGGREGATION

- 하위 노드 결과에 대해 aggregation을 수행한다.
- 하위 노드에서 aggregation을 처리할 수 없는 경우 별도의 HASH AGGREGATION node가 생성된다.
- Node information
 - AGGREGATIONS: Node 내에서 처리할 중첩되지 않은 aggregation list 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only SELECT sum( l_quantity ) FROM orders, lineitem;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |      ROWS |
=====
```

```

| 0 | SELECT STATEMENT | |
| 1 | HASH AGGREGATION | 0 |
| 2 | NESTED LOOP JOIN (INNER JOIN) | 0 |
| 3 | TABLE ACCESS ("LINEITEM") | 0 |
| 4 | INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX") | 0 |
=====
1 - AGGREGATIONS : SUM( L_QUANTITY )
2 - JOINED COLUMNS : LINEITEM,L_QUANTITY
3 - READ COLUMNS : L_QUANTITY
4 - READ INDEX COLUMNS : NOTHING
<<< end print plan

```

HASH JOIN (join_method)

- 하위 두 노드에 대해 hash join을 수행한다.
- 하위 노드 중 inner node는 hash instant로 구성되어 있다.
- Outer node의 IDX가 inner node의 IDX보다 작다. (Outer node가 상위에 표시됨)
- join_method
 - INNER JOIN
 - LEFT OUTER JOIN
 - FULL OUTER JOIN
 - LEFT SEMI JOIN
 - LEFT SEMI JOIN NA
 - INVERTED LEFT SEMI JOIN
 - LEFT ANTI SEMI JOIN
- Node information
 - JOINED COLUMNS: Join 결과가 될 column list 이다.
 - JOIN FILTER: Join condition으로 적용될 filter 이다.
 - WHERE FILTER: Join 수행 후 적용될 filter 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT l_quantity
      FROM orders FULL OUTER JOIN lineitem
            ON o_orderkey = l_orderkey and o_orderdate <> l_shipdate
      WHERE o_orderkey > 1;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION  |  ROWS  |
=====

```

```

| 0 | SELECT STATEMENT | |
| 1 | HASH JOIN (FULL OUTER JOIN) | 0 |
| 2 | TABLE ACCESS ("ORDERS") | 0 |
| 3 | HASH JOIN INSTANT ACCESS | 0 |
| 4 | TABLE ACCESS ("LINEITEM") | 0 |
=====
1 - JOINED COLUMNS : ORDERS.O_ORDERKEY, LINEITEM.L_QUANTITY
   JOIN FILTER : {O_ORDERDATE} <> {L_SHIPDATE}
   WHERE FILTER : ORDERS.O_ORDERKEY > 1
2 - READ COLUMNS : O_ORDERKEY, O_ORDERDATE
3 - INDEX COLUMNS : L_ORDERKEY
   TABLE COLUMNS : L_SHIPDATE, L_QUANTITY
   READ COLUMNS : L_ORDERKEY, L_SHIPDATE, L_QUANTITY
   HASH FILTER : {O_ORDERKEY} = L_ORDERKEY
4 - READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_SHIPDATE
<<< end print plan

```

HASH JOIN INSTANT ACCESS

- Hash join의 inner node 이다.
- Hash join 조건 (equi-join)으로 사용되는 column expression을 기준으로 하위 노드 결과에 대한 hash instant를 구성한다.
- Node information
 - INDEX COLUMNS: Hash column이 되는 hash instant의 column list 이다.
 - TABLE COLUMNS: Hash column을 제외한 hash instant의 column list 이다.
 - READ COLUMNS: Hash instant로부터 읽어들이 column list 이다.
 - HASH FILTER: Hash join을 위한 equi-join condition 이다.
 - PHYSICAL TABLE FILTER: INDEX COLUMNS를 포함하지 않고 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL TABLE FILTER: 전체 TABLE FILTER에서 PHYSICAL TABLE FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT /*+ USE_HASH( orders, lineitem )
              ORDERING( orders, lineitem ) */ l_quantity
      FROM orders INNER JOIN lineitem
              ON o_orderkey = l_orderkey and l_suppkey > o_orderkey
              and l_quantity * 20 > o_totalprice;
>>> start print plan
< Execution Plan >
=====

```

```

|  IDX  |  NODE DESCRIPTION  |  ROWS  |
-----|-----|-----|
|   0   |  SELECT STATEMENT  |         |
|   1   |    HASH JOIN (INNER JOIN)    |         0 |
|   2   |      TABLE ACCESS ("ORDERS")    |         0 |
|   3   |    HASH JOIN INSTANT ACCESS    |         0 |
|   4   |      TABLE ACCESS ("LINEITEM")    |         0 |
=====
1 - JOINED COLUMNS : LINEITEM.L_QUANTITY
2 - READ COLUMNS : O_ORDERKEY, O_TOTALPRICE
3 - INDEX COLUMNS : L_ORDERKEY
   TABLE COLUMNS : L_SUPPKEY, L_QUANTITY
   READ COLUMNS : L_ORDERKEY, L_SUPPKEY, L_QUANTITY
   HASH FILTER : {O_ORDERKEY} = L_ORDERKEY
   PHYSICAL TABLE FILTER : L_SUPPKEY > {O_ORDERKEY}
   LOGICAL TABLE FILTER : ( L_QUANTITY * 20 ) > {O_TOTALPRICE}
4 - READ COLUMNS : L_ORDERKEY, L_SUPPKEY, L_QUANTITY
<<< end print plan

```

HASH JOIN INSTANT ACCESS (UNIQUE)

- Hash join 조건 (equi-join)으로 사용되는 column expression을 기준으로 하위 노드 결과에 대한 hash instant를 구성한다.
 - 단, 전체 hash column list를 기준으로 중복을 제거한 hash instant를 구성한다.
- Hash join method가 semi join 또는 anti semi join인 경우에 생성된다.
- Node information
 - INDEX COLUMNS: Hash column이 되는 hash instant의 column list 이다.
 - TABLE COLUMNS: Hash column을 제외한 hash instant의 column list 이다.
 - READ COLUMNS: Hash instant로부터 읽어들이 column list 이다.
 - HASH FILTER: Hash join을 위한 equi-join condition 이다.
 - PHYSICAL TABLE FILTER: INDEX COLUMNS이 아닌 column이 포함되어 있고 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL TABLE FILTER: 전체 TABLE FILTER에서 PHYSICAL TABLE FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT o_custkey
      FROM orders
     WHERE o_orderkey in
           ( SELECT l_orderkey
             FROM lineitem );

```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
```

IDX	NODE DESCRIPTION	ROWS
0	SELECT STATEMENT	
1	HASH JOIN (LEFT SEMI)	0
2	TABLE ACCESS ("ORDERS")	0
3	HASH JOIN INSTANT ACCESS (UNIQUE)	0
4	TABLE ACCESS ("LINEITEM")	0

```
=====
```

```
1 - JOINED COLUMNS : ORDERS.O_CUSTKEY
2 - READ COLUMNS : O_ORDERKEY, O_CUSTKEY
3 - INDEX COLUMNS : L_ORDERKEY
   HASH FILTER : {O_ORDERKEY} = L_ORDERKEY
4 - READ COLUMNS : L_ORDERKEY
```

```
<<< end print plan
```

INDEX ACCESS (table_name [AS alias], index_name)

- index_name의 index를 이용하여 table_name의 레코드를 탐색한다.
- Node information
 - READ INDEX COLUMNS: Index key column인 참조 column list 이다.
 - READ TABLE COLUMNS: Index key column을 제외한 참조 column list 이다.
 - AGGREGATIONS: Node 내에서 처리할 중첩되지 않은 aggregation list 이다.
 - MIN RANGE: Index의 min key range 이다.
 - MAX RANGE: Index의 max key range 이다.
 - PHYSICAL KEY FILTER: Key column 만을 대상으로 하고 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL KEY FILTER: 전체 KEY FILTER 중에 PHYSICAL KEY FILTER를 제외한 AND-filter 이다.
 - PHYSICAL TABLE FILTER: Index key column이 아닌 column을 포함하며 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL TABLE FILTER: 전체 TABLE FILTER에서 PHYSICAL TABLE FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
```

```
SELECT /*+ INDEX( lineitem ) */ count( l_orderkey )
FROM lineitem
WHERE l_orderkey > 100 and l_linenumber > 0
      and mod( l_orderkey, 2 ) = 0 and l_quantity > 0
      and l_discount * l_tax > 100;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    INDEX ACCESS ("LINEITEM, LINEITEM_PK_INDEX") |         0 |
=====
```

```
1 - READ INDEX COLUMNS : L_ORDERKEY, L_LINENUMBER
   READ TABLE COLUMNS : L_QUANTITY, L_DISCOUNT, L_TAX
   AGGREGATIONS : COUNT( L_ORDERKEY )
   MIN RANGE : L_ORDERKEY > 100
   MAX RANGE : L_ORDERKEY IS NOT NULL
   PHYSICAL KEY FILTER : L_LINENUMBER > 0
   LOGICAL KEY FILTER : MOD(L_ORDERKEY,2) = 0
   PHYSICAL TABLE FILTER : L_QUANTITY > 0
   LOGICAL TABLE FILTER : ( L_DISCOUNT * L_TAX ) > 100
```

```
<<< end print plan
```

INSERT STATEMENT

- INSERT 구문을 수행한다.
- **INSERT (table_name)** node를 하위 노드로 갖는다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
```

```
INSERT INTO supplier VALUES ( 123, 'jhkim', 'KOREA', '0123456789' );
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  INSERT STATEMENT                                |         |
|    1  |    INSERT ("SUPPLIER")                          |         0 |
=====
```

```
<<< end print plan
```

INSERT (table_name)

해당 table에 INSERT 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      INSERT INTO supplier VALUES ( 123, 'jhkim', 'KOREA', '0123456789' );
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  INSERT STATEMENT                                |         |
|    1  |    INSERT ("SUPPLIER")                           |         0 |
=====

<<< end print plan
```

INTERSECT ALL

하위 노드들에 대해 INTERSECT ALL 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT s_suppkey FROM supplier
      INTERSECT ALL
      SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
      INTERSECT ALL
      SELECT ps_suppkey FROM partsupp;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    INTERSECT-ALL                                |         0 |
|    2  |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
|    3  |      TABLE ACCESS ("LINEITEM")                  |         0 |
|    4  |      INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") |         0 |
=====

      2 - READ INDEX COLUMNS : S_SUPPKEY
      3 - READ COLUMNS : L_SUPPKEY
      4 - READ INDEX COLUMNS : PS_SUPPKEY

<<< end print plan
```

INTERSECT DISTINCT

하위 노드들에 대해 INTERSECT DISTINCT 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT s_suppkey FROM supplier
      INTERSECT DISTINCT
      SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
      INTERSECT DISTINCT
      SELECT ps_suppkey FROM partsupp;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    INTERSECT-DISTINCT                            |         0 |
|   2   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
|   3   |      TABLE ACCESS ("LINEITEM")                  |         0 |
|   4   |      INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") |         0 |
=====

      2 - READ INDEX COLUMNS : S_SUPPKEY
      3 - READ COLUMNS : L_SUPPKEY
      4 - READ INDEX COLUMNS : PS_SUPPKEY
<<< end print plan
```

NESTED LOOP JOIN (join_method)

- 하위 두 노드에 대해 nested loop join을 수행한다.
- Outer node의 IDX가 inner node의 IDX보다 작다. (Outer node가 상위에 표시됨)
- join_method
 - CROSS JOIN
 - INNER JOIN
 - LEFT OUTER JOIN
 - FULL OUTER JOIN
 - LEFT SEMI JOIN
 - INVERTED LEFT SEMI JOIN
 - LEFT ANTI SEMI JOIN
- Node information
 - JOINED COLUMNS: Join 결과가 될 column list 이다.

- JOIN FILTER: Join condition으로 적용될 filter 이다.
- WHERE FILTER: Join 수행 후 적용될 filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT /*+ USE_NL( orders, lineitem )
              ORDERING( orders, lineitem ) */ l_quantity
      FROM orders FULL OUTER JOIN lineitem
              ON o_orderkey = l_orderkey
      WHERE l_quantity > 10;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    NESTED LOOP JOIN (FULL OUTER JOIN)           |         0 |
|    2  |      TABLE ACCESS ("ORDERS")                   |         0 |
|    3  |      TABLE ACCESS ("LINEITEM")                 |         0 |
=====

1 - JOINED COLUMNS : LINEITEM.L_QUANTITY
   JOIN FILTER : {O_ORDERKEY} = {L_ORDERKEY}
   WHERE FILTER : LINEITEM.L_QUANTITY > 10
2 - READ COLUMNS : O_ORDERKEY, O_TOTALPRICE
3 - READ COLUMNS : L_ORDERKEY, L_SUPPKEY, L_QUANTITY
<<< end print plan
```

SELECT STATEMENT

SELECT 구문을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only SELECT s_suppkey FROM supplier;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
=====
```

```
1 - READ INDEX COLUMNS : S_SUPPKEY
```

```
<<< end print plan
```

SORT INSTANT ACCESS

- Sort key column을 기준으로 하는 하위 노드의 결과에 대한 sort instant를 구성한다.
- SELECT 구문에 ORDER BY 절을 사용할 때 또는 subquery를 실체화할 때 생성된다.
- Node information
 - SORT KEY: Sort key column이 되는 sort instant의 column list 이다.
 - RECORD COLUMNS: Sort key column을 제외한 sort instant의 column list 이다.
 - READ COLUMNS: Sort instant로부터 읽어들이 column list 이다.
 - MIN RANGE: Index의 min key range 이다.
 - MAX RANGE: Index의 max key range 이다.
 - PHYSICAL KEY FILTER: Key column만을 대상으로 하고 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL KEY FILTER: 전체 KEY FILTER 중에 PHYSICAL KEY FILTER를 제외한 AND-filter 이다.
 - PHYSICAL TABLE FILTER: Index key column이 아니고 column을 포함하며 type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL TABLE FILTER: 전체 TABLE FILTER에서 PHYSICAL TABLE FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT /*+ USE_NL( lineitem, partsupp )
              ORDERING( lineitem, partsupp ) */ l_orderkey
      FROM lineitem INNER JOIN
        ( SELECT /*+ NO_QUERY_TRANSFORMATION */
          ps_partkey, ps_suppkey, ps_availqty
        FROM partsupp OFFSET 1 )
      ON l_partkey = ps_partkey and l_suppkey = ps_suppkey
      and ps_suppkey > l_suppkey and ps_partkey + 1 > l_partkey
      and ps_availqty < l_quantity and ps_availqty * 10 > l_quantity;
```

```
>>> start print plan
```

```
< Execution Plan >
```

=====			=====		
	IDX		NODE DESCRIPTION		ROWS

	0		SELECT STATEMENT		
	1		NESTED LOOP JOIN (INNER JOIN)		0
	2		TABLE ACCESS ("LINEITEM")		0
	3		SORT INSTANT ACCESS		0
	4		VIEW		0

```

| 5 | TABLE ACCESS ("PARTSUPP") | 0 |
=====
1 - JOINED COLUMNS : LINEITEM.L_ORDERKEY
2 - READ COLUMNS : L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_QUANTITY
3 - SORT KEY : "PARTSUPP.PS_SUPPKEY ASC NULLS LAST",
              "PARTSUPP.PS_PARTKEY ASC NULLS LAST"
   RECORD COLUMNS : PS_AVAILQTY
   READ COLUMNS : PS_SUPPKEY, PS_PARTKEY, PS_AVAILQTY
   MIN RANGE : PS_SUPPKEY = {L_SUPPKEY}
               AND PS_SUPPKEY > {L_SUPPKEY}
   MAX RANGE : PS_SUPPKEY = {L_SUPPKEY}
   PHYSICAL KEY FILTER : PS_PARTKEY = {L_PARTKEY}
   LOGICAL KEY FILTER : ( PS_PARTKEY + 1 ) > {L_PARTKEY}
   PHYSICAL TABLE FILTER : PS_AVAILQTY < {L_QUANTITY}
   LOGICAL TABLE FILTER : ( PS_AVAILQTY * 10 ) > {L_QUANTITY}
4 - COLUMNS : PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY
5 - READ COLUMNS : PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY
<<< end print plan

```

SORT INSTANT ACCESS (UNIQUE)

- Sort key column을 기준으로 하는 하위 노드 결과에 대한 sort instant를 구성한다.
 - 단, 전체 sort key column list를 기준으로 중복을 제거한 sort instant를 구성한다.
- Join의 outer node로 subquery 결과를 실체화하는 경우에 해당 노드가 생성된다.
- Node information
 - SORT KEY: Sort key column이 되는 sort instant의 column list 이다.
 - READ COLUMNS: Sort instant로부터 읽어들이 column list 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT l_orderkey
      FROM lineitem
      WHERE EXISTS( SELECT /*+ NL_ISJ */ o_orderkey
                    FROM orders WHERE o_orderkey = l_orderkey );
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|  0    |  SELECT STATEMENT                                |         |
|  1    |  NESTED LOOP JOIN (INVERTED LEFT SEMI)           |         0 |

```

```

| 2 |      SORT INSTANT ACCESS (UNIQUE)      |      0 |
| 3 |      INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX") |      0 |
| 4 |      INDEX ACCESS ("LINEITEM, LINEITEM_PK_INDEX") |      0 |
=====
1 - JOINED COLUMNS : LINEITEM.L_ORDERKEY
2 - SORT KEY : "ORDERS.O_ORDERKEY ASC NULLS LAST"
   READ COLUMNS : O_ORDERKEY
3 - READ INDEX COLUMNS : O_ORDERKEY
4 - READ INDEX COLUMNS : L_ORDERKEY
   MIN RANGE : L_ORDERKEY = {O_ORDERKEY}
   MAX RANGE : L_ORDERKEY = {O_ORDERKEY}
<<< end print plan

```

SORT JOIN INSTANT ACCESS

- Sort merge join의 inner node이다.
- Sort merge join 조건 (equi-join)으로 사용되는 column expression을 기준으로 하위 노드 결과에 대한 sort instant를 구성한다.
- Node information
 - SORT KEY: Sort key column이 되는 sort instant의 column list 이다.
 - RECORD COLUMNS: Sort key column을 제외한 sort instant의 column list 이다.
 - READ COLUMNS: Sort instant로부터 읽어들일 column list 이다.
 - MIN RANGE: Sort instant의 min key range 이다.
 - MAX RANGE: Sort instant의 max key range 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT /*+ USE_MERGE( orders, lineitem )
              ORDERING( orders, lineitem ) */ l_quantity
      FROM orders INNER JOIN lineitem
              ON o_orderkey = l_orderkey
              and l_quantity * 20 > o_totalprice;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----|-----|-----|
|  0  |  SELECT STATEMENT                                |         |
|  1  |  SORT MERGE JOIN (INNER JOIN) : EQUAL            |         0 |
|  2  |  INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")        |         0 |
|  3  |  SORT JOIN INSTANT ACCESS                        |         0 |

```

```

| 4 | TABLE ACCESS ("LINEITEM") | 0 |
=====
1 - JOINED COLUMNS : LINEITEM.L_QUANTITY
    MERGE FILTER : ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY
    JOIN FILTER : ( {L_QUANTITY} * 20 ) > {O_TOTALPRICE}
2 - READ INDEX COLUMNS : O_ORDERKEY
    READ TABLE COLUMNS : O_TOTALPRICE
3 - SORT KEY : "LINEITEM.L_ORDERKEY ASC NULLS LAST"
    RECORD COLUMNS : L_QUANTITY
    READ COLUMNS : L_ORDERKEY, L_QUANTITY
    MIN RANGE : L_ORDERKEY >= {O_ORDERKEY}
    MAX RANGE : L_ORDERKEY IS NOT NULL
4 - READ COLUMNS : L_ORDERKEY, L_QUANTITY
<<< end print plan

```

SORT JOIN INSTANT ACCESS (UNIQUE)

- Sort merge join 조건 (equi-join)으로 사용되는 column expression을 기준으로 하위 노드의 결과에 대한 sort instant를 구성한다.
 - 단, 전체 sort key column list를 기준으로 중복을 제거한 sort instant를 구성한다.
- Node information
 - SORT KEY: Sort key column이 되는 sort instant의 column list 이다.
 - READ COLUMNS: Sort instant로부터 읽어들이 column list 이다.
 - MIN RANGE: Sort instant의 min key range 이다.
 - MAX RANGE: Sort instant의 max key range 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT o_custkey
      FROM orders
      WHERE o_orderkey in
            ( SELECT /*+ MERGE_SJ FULL( lineitem ) */ l_orderkey FROM lineitem );
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|  0  |  SELECT STATEMENT                                |         |
|  1  |  SORT MERGE JOIN (LEFT SEMI) : EQUAL              |         0 |
|  2  |  INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")         |         0 |
|  3  |  SORT JOIN INSTANT ACCESS (UNIQUE)                 |         0 |

```



```

| 0 | SELECT STATEMENT | |
| 1 |   SORT MERGE JOIN (FULL OUTER JOIN) : EQUAL | 0 |
| 2 |     INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX") | 0 |
| 3 |     SORT JOIN INSTANT ACCESS | 0 |
| 4 |       TABLE ACCESS ("LINEITEM") | 0 |
=====
1 - JOINED COLUMNS : ORDERS.O_ORDERKEY, LINEITEM.L_QUANTITY
   MERGE FILTER : ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY
   JOIN FILTER : {O_ORDERDATE} <> {L_SHIPDATE}
   WHERE FILTER : ORDERS.O_ORDERKEY > 1
2 - READ INDEX COLUMNS : O_ORDERKEY
   READ TABLE COLUMNS : O_ORDERDATE
3 - SORT KEY : "LINEITEM.L_ORDERKEY ASC NULLS LAST"
   RECORD COLUMNS : L_SHIPDATE, L_QUANTITY
   READ COLUMNS : L_ORDERKEY, L_SHIPDATE, L_QUANTITY
4 - READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_SHIPDATE
<<< end print plan

```

SUB QUERY FUNCTION

- Subquery를 포함하는 function을 처리한다.
- Node information
 - FUNCTION: Function expression 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT exists ( SELECT /*+ FULL( lineitem ) */ l_orderkey
      FROM lineitem
      WHERE o_orderkey = l_orderkey ) FROM orders;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION |  ROWS |
=====
|  0  |  SELECT STATEMENT |      |
|  1  |  INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX") |  0  |
|  2  |  SUB QUERY LIST |  0  |
|  3  |  SUB QUERY FUNCTION |  0  |
|  4  |  TABLE ACCESS ("LINEITEM") |  0  |
=====
1 - READ INDEX COLUMNS : O_ORDERKEY

```

```

3 - FUNCTION : EXISTS( ( L_ORDERKEY ) )
4 - READ COLUMNS : L_ORDERKEY
    PHYSICAL FILTER : {O_ORDERKEY} = L_ORDERKEY

```

```
<<< end print plan
```

SUB QUERY FUNCTION (MATERIALIZED)

- Subquery 수행 결과를 실체화한 후에 function을 처리한다.
- Node information
 - **FUNCTION**: Function expression 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT o_orderkey in
          ( SELECT /*+ FULL( lineitem ) */ l_orderkey FROM lineitem )
      FROM orders;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |  INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")         |    0   |
|   2   |  SUB QUERY LIST                                   |    0   |
|   3   |  SUB QUERY FUNCTION (MATERIALIZED)               |    0   |
|   4   |  TABLE ACCESS ("LINEITEM")                       |    0   |
=====
1 - READ INDEX COLUMNS : O_ORDERKEY
3 - FUNCTION : ( O_ORDERKEY ) IN ( L_ORDERKEY )
4 - READ COLUMNS : L_ORDERKEY
<<< end print plan

```

SUB QUERY LIST

- Subquery들을 수행하여 상위 노드로 결과를 전달한다.
- Outer column이 존재하는 경우 상수화하여 상위 노드로 전달한다.
 - 상위 노드에서 참조하는 하위 노드의 column을 outer column이라고 한다.
- Node information
 - **READ COLUMNS**: Outer column으로 구성할 column list 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT ( SELECT count(*) FROM lineitem ) = count(*) FROM orders;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")      |    0    |
|   2   |      SUB QUERY LIST                              |    0    |
|   3   |        INDEX ACCESS ("LINEITEM, LINEITEM_PK_INDEX") |    0    |
=====

1 - READ INDEX COLUMNS : NOTHING
   AGGREGATIONS : COUNT(*)
2 - READ COLUMNS : COUNT(*)
3 - READ INDEX COLUMNS : NOTHING
   AGGREGATIONS : COUNT(*)

<<< end print plan
```

TABLE ACCESS (table_name [AS alias])

- table_name의 레코드를 탐색한다.
- Node information
 - READ COLUMNS: 참조 column list 이다.
 - AGGREGATIONS: Node 내에서 처리할 중첩되지 않은 aggregation list 이다.
 - PHYSICAL FILTER: Type casting이 필요없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL FILTER: 전체 TABLE FILTER에서 PHYSICAL FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT /*+ FULL( lineitem ) */ count( l_orderkey )
      FROM lineitem
      WHERE l_orderkey > 0
            and l_quantity * l_extendedprice > 100;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
```

```

| 0 | SELECT STATEMENT | |
| 1 | TABLE ACCESS ("LINEITEM") | 0 |
=====
1 - READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_EXTENDEDPRICE
   AGGREGATIONS : COUNT( L_ORDERKEY )
   PHYSICAL FILTER : L_ORDERKEY > 0
   LOGICAL FILTER : ( L_QUANTITY * L_EXTENDEDPRICE ) > 100
<<< end print plan

```

UNION ALL

하위 노드들에 대한 UNION ALL 연산을 수행한다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT s_suppkey FROM supplier
      UNION ALL
      SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
      UNION ALL
      SELECT ps_suppkey FROM partsupp;
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |
=====
| 0 | SELECT STATEMENT | |
| 1 | UNION-ALL | 0 |
| 2 | INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") | 0 |
| 3 | TABLE ACCESS ("LINEITEM") | 0 |
| 4 | INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") | 0 |
=====
2 - READ INDEX COLUMNS : S_SUPPKEY
3 - READ COLUMNS : L_SUPPKEY
4 - READ INDEX COLUMNS : PS_SUPPKEY
<<< end print plan

```

UNION DISTINCT

하위 노드들에 대해 UNION DISTINCT 연산을 수행한다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT s_suppkey FROM supplier
      UNION DISTINCT
      SELECT /*+ FULL( lineitem) */ l_suppkey FROM lineitem
      UNION DISTINCT
      SELECT ps_suppkey FROM partsupp;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    UNION-DISTINCT                                |         0 |
|   2   |      INDEX ACCESS ("SUPPLIER, SUPPLIER_PK_INDEX") |         0 |
|   3   |      TABLE ACCESS ("LINEITEM")                  |         0 |
|   4   |      INDEX ACCESS ("PARTSUPP, PARTSUPP_PK_INDEX") |         0 |
=====
      2 - READ INDEX COLUMNS : S_SUPPKEY
      3 - READ COLUMNS : L_SUPPKEY
      4 - READ INDEX COLUMNS : PS_SUPPKEY
<<< end print plan

```

UPDATE STATEMENT

- UPDATE 구문을 수행한다.
- **UPDATE (table_name)** node를 하위 노드로 갖는다.

다음 예제를 참조한다.

```

gSQL> \explain plan only UPDATE lineitem SET l_quantity = l_quantity + 1;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  UPDATE STATEMENT                                |         |
|   1   |    UPDATE ("LINEITEM")                            |         0 |
|   2   |      TABLE ACCESS ("LINEITEM")                  |         0 |
=====
      2 - READ COLUMNS : L_QUANTITY
<<< end print plan

```

UPDATE (table_name)

해당 table에 UPDATE 연산을 수행한다.

다음 예제를 참조한다.

```
gSQL> \explain plan only UPDATE lineitem SET l_quantity = l_quantity + 1;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                               |  ROWS  |
-----|-----|-----|
|    0   |  UPDATE STATEMENT                               |         |
|    1   |    UPDATE ("LINEITEM")                         |         0 |
|    2   |      TABLE ACCESS ("LINEITEM")                 |         0 |
=====

      2 - READ COLUMNS : L_QUANTITY
<<< end print plan
```

USER ROWID ACCESS (table_name [AS alias])

- Rowid와 일치하는 하나의 레코드를 탐색한다.
- Node information
 - READ COLUMNS: 참조 column list 이다.
 - AGGREGATIONS: Node 내에서 처리할 중첩되지 않은 aggregation list 이다.
 - ROWID ACCESS EXPR: 찾고자 하는 rowid expression 이다.
 - PHYSICAL FILTER: Type casting이 필요 없는 비교 연산에 대한 AND-filter 이다.
 - LOGICAL FILTER: 전체 TABLE FILTER에서 PHYSICAL FILTER를 제외한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
      SELECT /*+ ROWID( T1 ) */ COUNT( l_orderkey )
      FROM lineitem
      WHERE rowid = null and l_orderkey > 0
             and l_quantity * l_extendedprice > 100;
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                               |  ROWS  |
-----|-----|-----|
|    0   |  SELECT STATEMENT                               |         |
|    1   |    USER ROWID ACCESS ("LINEITEM")               |         0 |
=====
```

```
=====
1 - READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_EXTENDEDPRICE
  AGGREGATIONS : COUNT( L_ORDERKEY )
  ROWID ACCESS EXPR : ROWID = NULL
  PHYSICAL FILTER : L_ORDERKEY > 0
  LOGICAL FILTER : ( L_QUANTITY * L_EXTENDEDPRICE ) > 100
```

```
<<< end print plan
```

VIEW

- Inline view로 구성된 하위 노드를 수행한다.
- Node information
 - COLUMNS: 참조 column list 이다.
 - FILTER: View 조회 결과에 대한 AND-filter 이다.

다음 예제를 참조한다.

```
gSQL> \explain plan only
SELECT /*+ ORDERED FULL( T1 ) */ count(*)
FROM lineitem AS T1,
     ( SELECT /*+ FULL( lineitem ) */ l_orderkey key FROM lineitem
       UNION ALL
       SELECT /*+ FULL( orders ) */ o_orderkey FROM orders )
WHERE key = 1 or key = 2;
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    HASH AGGREGATION                              |        0 |
|   2   |      NESTED LOOP JOIN (INNER JOIN)                |        0 |
|   3   |        TABLE ACCESS ("LINEITEM AS T1")           |        0 |
|   4   |          VIEW                                       |        0 |
|   5   |            UNION-ALL                              |        0 |
|   6   |              TABLE ACCESS ("LINEITEM")           |        0 |
|   7   |              TABLE ACCESS ("ORDERS")             |        0 |
=====
```

```
1 - AGGREGATIONS : COUNT(*)
2 - JOINED COLUMNS : NOTHING
3 - READ COLUMNS : NOTHING
4 - COLUMNS : L_ORDERKEY
```

```

        FILTER : LINEITEM.L_ORDERKEY = 1 OR LINEITEM.L_ORDERKEY = 2
    6 - READ COLUMNS : L_ORDERKEY
    7 - READ COLUMNS : O_ORDERKEY
<<< end print plan

```

VIEW (view_name)

- view_name을 가지는 view 또는 inline-view를 수행한다.
- Node information
 - COLUMNS: 참조 column list 이다.
 - FILTER: View 조회 결과에 대한 AND-filter 이다.

다음 예제를 참조한다.

```

gSQL> \explain plan only
      SELECT /*+ ORDERING( T1, V1 ) FULL( T1 ) */ count(*)
      FROM ( SELECT /*+ FULL( lineitem ) */ l_orderkey key FROM lineitem
            UNION ALL
            SELECT /*+ FULL( orders ) */ o_orderkey FROM orders ) V1,
            lineitem AS T1
      WHERE V1.key = 1 or V1.key = 2;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|   0   |  SELECT STATEMENT                                |         |
|   1   |    HASH AGGREGATION                              |         0 |
|   2   |      NESTED LOOP JOIN (INNER JOIN)                |         0 |
|   3   |        TABLE ACCESS ("LINEITEM AS T1")           |         0 |
|   4   |          VIEW ("V1")                               |         0 |
|   5   |            UNION-ALL                              |         0 |
|   6   |              TABLE ACCESS ("LINEITEM")            |         0 |
|   7   |                TABLE ACCESS ("ORDERS")             |         0 |
=====
    1 - AGGREGATIONS : COUNT(*)
    2 - JOINED COLUMNS : NOTHING
    3 - READ COLUMNS : NOTHING
    4 - COLUMNS : L_ORDERKEY
        FILTER : LINEITEM.L_ORDERKEY = 1 OR LINEITEM.L_ORDERKEY = 2
    6 - READ COLUMNS : L_ORDERKEY
    7 - READ COLUMNS : O_ORDERKEY

```

|| <<< end print plan

15.5 SQL Trace Log

개요

SQL trace log는 분석 가능한 사용자 질의 수행 정보를 기록하는 log이다. SQL trace log는 프로세스 ID와 세션 ID로 구분되어 각각 독립된 파일로 \$GOLDILOCKS_DATA/trc 디렉토리에 생성된다. SQL trace log는 사용자 질의, SQL 실행 계획, SQL 처리 과정별 수행시간 등과 같은 여러 정보로 구성되어 파일에 출력된다.

출력

SQL trace log를 출력하기 위해서는 ALTER SESSION 또는 ALTER SYSTEM 구문을 사용하여 TRACE_LOG_ID의 값을 설정해 주어야 한다. 설정값은 server property의 **TRACE_LOG_ID**를 참조한다.

Trace log는 성공한 SQL 질의와 실패한 SQL 질의 모두를 남길 수 있는데 이는 TRACE_LOG_ID의 flag 값을 조합해서 설정할 수 있다.

노트

SQL 구문의 parsing, validation, optimization 등을 거쳐 수행할 plan을 결정한 다음 수행되는 과정 (SQL 처리 과정 중 execution 과정)에서 실패한 SQL 질의를 실행에 실패한 SQL 질의라고 한다. 따라서 질의를 수행할 plan이 결정되기 전에 실패하는 질의에 대해서는 trace log를 남기지 않는다.

다음은 TRACE_LOG_ID 프로퍼티를 사용하여 SQL trace log를 출력하는 예이다.

- 성공한 SQL 구문과 실패한 SQL 구문을 출력한다.

```
gSQL> ALTER SESSION SET TRACE_LOG_ID = 110000;  
Session altered.
```

- 성공한 SQL 구문과 bind 값을 출력한다.

```
gSQL> ALTER SYSTEM SET TRACE_LOG_ID = 100010;  
System altered.
```

SQL trace log를 출력하는 프로퍼티인 TRACE_LOG_ID를 ALTER SESSION으로 설정하면 해당 세션에만 반영되어 동작하며, ALTER SYSTEM으로 설정하면 서버에 접속한 모든 프로세스의 모든 세션에 반영되어 동작한다. 따라서 현

재 세션의 SQL trace log를 보고 싶은 경우에는 ALTER SESSION을 사용하고 현재 수행되고 있는 다른 프로세스 및 다른 세션에 대한 SQL trace log를 보고 싶은 경우에는 ALTER SYSTEM을 사용한다.

주의

ALTER SYSTEM으로 설정된 상태에서 서버에 접속한 프로세스와 세션의 수가 많다면 SQL trace log의 파일도 그만큼 많이 생성되므로 사용에 주의하여야 한다.

SQL trace log 파일은 trc 디렉토리 아래에 생성되는데 파일명 규칙은 다음과 같다.

`opt_p[프로세스ID]_s[세션ID].trc`

파일명 앞에 opt 그 뒤에 p 식별자와 함께 프로세스ID, 그 뒤에 s 식별자와 함께 세션ID를 나열한다. _를 구분자로 하고 확장자는 trc이다. 만약 동일 프로세스의 동일 세션에서 생성된 내용이 SQL trace log 파일의 최대 크기보다 많은 경우 기존 파일은 파일명 뒤에 현재 시점의 시간을 추가한 파일명으로 변경하고, 현재 파일명으로 새로운 파일을 만들어 계속 기록한다.

다음은 SQL trace log 파일명의 예이다.

`opt_p17104_s12.trc`

출력 형식

SQL trace log는 크게 <SQL query string>, <Execution plan>, <Execution type>, <Bind param value>, <Time info>로 나누어 진다.

SQL Query String

사용자가 입력한 질의를 현재 시간과 성공여부, 질의 처리 시간을 포함하여 출력하며 출력형식은 다음과 같다.

`[현재 시간] [성공여부][질의 처리 시간] SQL 구문`

[현재 시간]은 날짜와 us 단위의 시간까지 출력하며, [성공여부]는 성공인 경우 'S', 실패인 경우 'F'로 출력한다. 질의 처리 시간은 us 단위의 시간으로 출력하며 SQL 구문은 사용자가 입력한 SQL 구문이다.

참고로 TRACE_LOG_TIME_DETAIL 프로퍼티를 ON으로 설정하지 않은 경우 질의 처리 시간이 10 ms 단위로 측정된다. 이 프로퍼티를 ON으로 설정하면 질의 처리 성능이 저하될 수 있으므로 주의한다.

Execution Plan

SQL 구문의 실행 계획을 출력한다. 이는 **SQL 실행 계획**과 거의 동일한 형태이며, execution plan node table에 total time column이 추가적으로 출력된다. Statement에 출력되는 total time은 질의 전체를 수행한 시간을 의미하며, 나머지 각 노드들에 출력되는 total time은 각각의 노드에서 수행한 시간을 의미한다. 이 때 total time은 10 ms 단위로 출력된다.

TRACE_LOG_TIME_DETAIL 프로퍼티를 사용하면 좀 더 자세한 시간을 출력할 수 있다. 단, 이 프로퍼티를 ON으로 설정하면 질의 처리 성능이 저하될 수 있으므로 주의한다.

Execution Type

SQL 구문의 실행 형태로서 직접 질의를 수행하는 경우 DIRECT EXECUTE를 출력하고, prepare를 사용하여 질의를 수행하는 경우 PREPARE EXECUTE를 출력한다.

Bind Param Value

SQL 구문에서 bind param value를 사용한 경우 해당 bind param value의 정보를 출력한다. 만약 SQL 구문에 bind param value를 사용하지 않은 경우 No Bind Param을 출력한다.

Time Info

SQL 처리과정의 각 단계별 실행 시간을 출력한다. Time info는 module, time, rate, call로 구분되어 출력되며, module은 parse, validate와 같은 단계명을 출력하고, time은 실제 수행시간을 출력하며, rate은 전체 수행시간에서 각 단계의 수행시간이 차지하는 비율을 출력한다. Call은 각 단계가 호출된 횟수를 출력한다.

Module은 parse, validate, code opt, optimizer, data opt, execute, fetch의 7단계와 total로 나뉘어 있다. Parse는 질의를 parsing하고 validate는 parsing된 질의에 대하여 validation을 수행한다. Code opt는 SQL optimizer를 수행하기 위한 전처리 단계이며, optimizer는 실제로 SQL optimizer를 수행한다. Data opt는 SQL 실행 계획을 실제로 수행하기 위해 준비하는 단계이고 execute는 SQL 실행 계획을 수행한다. Fetch는 SELECT 구문과 같이 질의 결과들을 수집하여 결과를 반환한다.

Plan cache가 사용될 경우 validate, code opt, optimizer 등은 호출되지 않을 수 있다. Time의 경우 10 ms 단위까지만 출력되기 때문에 10 ms 이하의 수행시간은 0으로 출력된다. 또한 rate의 경우 전체 수행시간 대비 각 단계 수행시간의 비율이기 때문에 total을 100%로 하여 각 단계별 수행시간을 나눈 비율을 출력하며, 이 때 각 단계의 수행시간이 0인 경우 0%로 출력된다.

TRACE_LOG_TIME_DETAIL 프로퍼티를 사용하면 좀 더 자세한 시간을 출력할 수 있다. 단, 이 프로퍼티를 ON으로 설정할 경우 질의 처리 성능이 저하될 수 있으므로 주의한다.

출력 예

다음은 bind param value가 존재하지 않는 형태의 SQL 구문이다.

```
SELECT O_TOTALPRICE, O_ORDERDATE, L_QUANTITY
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= DATE '1996-01-01'
AND L_SHIPMODE = 'AIR';
```

다음은 TRACE_LOG_ID를 101111로 설정한 다음 위의 SQL 구문을 수행할 경우 출력되는 SQL trace log이다.

```
[2017-05-25 12:27:59.199657] [S][0.000000] SELECT O_TOTALPRICE, O_ORDERDATE, L_QUANTITY
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= DATE '1996-01-01'
AND L_SHIPMODE = 'AIR'
```

< Execution Plan >

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS | Total Time |
=====
|   0   |  SELECT STATEMENT                                |       |             |
|   1   |    NESTED LOOP JOIN (INNER JOIN)                  |      1 | 0:00:00.00 |
|   2   |      TABLE ACCESS ("LINEITEM")                   |      2 | 0:00:00.00 |
|   3   |        INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX")   |      1 | 0:00:00.00 |
=====

1 - JOINED COLUMNS : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE, LINEITEM.L_QUANTITY
2 - READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_SHIPMODE
   PHYSICAL FILTER : L_SHIPMODE = 'AIR'
3 - READ INDEX COLUMNS : O_ORDERKEY
   READ TABLE COLUMNS : O_TOTALPRICE, O_ORDERDATE
   MIN RANGE : O_ORDERKEY = {L_ORDERKEY}
   MAX RANGE : O_ORDERKEY = {L_ORDERKEY}
   PHYSICAL TABLE FILTER : O_ORDERDATE >= CAST( '1996-01-01' AS DATE )
```

< Execution Type >

```
-----
DIRECT EXECUTE
< Bind Param Value >
```

```
-----
No Bind Param.
```

< Time Info >

=====			
Module	Time	Rate	Call

Parse	0:00:00.00	0.00 %	1
Validate	0:00:00.00	0.00 %	1
Code Opt	0:00:00.00	0.00 %	1
Optimizer	0:00:00.00	0.00 %	1
Data Opt	0:00:00.00	0.00 %	1
Execute	0:00:00.00	0.00 %	1
Fetch	0:00:00.00	0.00 %	1
Total	0:00:00.00	100.00 %	
=====			

다음은 bind param value가 존재하는 형태의 SQL 구문이다.

```
SELECT L_QUANTITY
FROM LINEITEM
WHERE L_SHIPMODE = :V1;
```

다음은 TRACE_LOG_ID를 101111로 설정한 다음 위의 SQL 구문을 수행한 경우 출력되는 SQL trace log이다.

```
[2017-05-25 12:27:59.200204] [S][0.000000] SELECT L_QUANTITY
FROM LINEITEM
WHERE L_SHIPMODE = :V1
```

< Execution Plan >

=====			
IDX	NODE DESCRIPTION	ROWS	Total Time

0	SELECT STATEMENT		0:00:00.00
1	TABLE ACCESS ("LINEITEM")	0	0:00:00.00
=====			

```
1 - READ COLUMNS : L_QUANTITY, L_SHIPMODE
    PHYSICAL FILTER : L_SHIPMODE = :V1
```

< Execution Type >

```
-----
DIRECT EXECUTE
```

< Bind Param Value >

```
-----
1 - :V1(IN, "AIR")
```

< Time Info >

```
=====
```

Module	Time	Rate	Call
Parse	0:00:00.00	0.00 %	1
Validate	0:00:00.00	0.00 %	1
Code Opt	0:00:00.00	0.00 %	1
Optimizer	0:00:00.00	0.00 %	1
Data Opt	0:00:00.00	0.00 %	1
Execute	0:00:00.00	0.00 %	1
Fetch	0:00:00.00	0.00 %	1
Total	0:00:00.00	100.00 %	

16.

SQL References

16.1 ALTER AUDIT POLICY

기능

Audit policy 객체에 감사 대상을 추가하거나 삭제한다.

구문

```

<alter audit policy statement> ::=
    ALTER AUDIT POLICY policy_name
    { <add_audit_option> | <drop_audit_option> }
    ;
<add_audit_option> ::=
    ADD { <privilege_audit_clause> | <action_audit_clause> | <privilege_audit_clause> <action
_audit_clause> }
<drop_audit_option> ::=
    DROP { <privilege_audit_clause> | <action_audit_clause> | <privilege_audit_clause> <
action_audit_clause> }
<privilege_audit_clause> ::=
    PRIVILEGES <database_privilege> [, ...]
<action_audit_clause> ::=
    ACTIONS { <object_action_audit> | <system_action_audit> } [, ...]
<object_action_audit> ::=
    ALL ON [schema_name.]object_name
    | <object_action> ON [schema_name.]object_name
<system_action_audit> ::=
    ALL
    | <system_action>

```

사용 범위 및 접근 권한

<alter audit policy statement> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

policy_name

변경할 audit policy 객체의 이름이다.

<add_audit_option>

Audit policy에 감사대상을 추가한다.

<drop_audit_option>

Audit policy 감사대상에서 삭제한다.

<privilege_audit_clause>

자세한 내용은 CREATE AUDIT POLICY 를 참조한다.

<action_audit_clause>

자세한 내용은 CREATE AUDIT POLICY 를 참조한다.

설명

이미 활성화된 audit policy를 변경할 수 있지만 기존 session에는 영향을 주지 않고 새로 생성되는 session에만 영향을 미친다.

노트

다음과 같이 ALL 옵션을 DROP 할 경우, 모든 action이 삭제되는 것이 아니라 해당 ALL 옵션만 삭제된다.

```
CREATE AUDIT POLICY p1
  ACTIONS ALL ON u1.t1,
          SELECT ON u1.t1;
ALTER AUDIT POLICY p1 DROP
  ACTIONS ALL ON u1.t1;
```

사용 예

다음은 audit policy에 새로운 audit option을 추가하는 예이다.

```
ALTER AUDIT POLICY policy_dml  
    ADD ACTIONS SELECT ON u1.t1;
```

다음은 audit policy에서 audit option을 삭제하는 예이다.

```
ALTER AUDIT POLICY policy_dml  
    DROP ACTIONS SELECT ON u1.t1;
```

호환성

SQL 표준에는 audit policy가 없다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - CREATE AUDIT POLICY
 - DROP AUDIT POLICY
 - ALTER AUDIT POLICY
- Audit policy 활성화/ 비활성화
 - AUDIT POLICY
 - NOAUDIT POLICY
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 삭제: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.2 ALTER CLUSTER GROUP name ADD MEMBER

기능

Cluster group에 cluster member를 추가한다.

구문

```

<alter cluster group add member statement> ::=
    ALTER CLUSTER GROUP group_name ADD
        <cluster member definition> [, ...]
    ;
<cluster member definition> ::=
    CLUSTER MEMBER member_name <connection attribute>
<connection attribute> ::
    HOST 'address' PORT port_no
  
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter cluster group add member statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

group_name

Cluster group의 이름이다.

<cluster member definition>

Cluster group에 포함될 cluster member를 정의한다.

Cluster group은 최대 32 개의 cluster member를 포함할 수 있다.

member_name

Cluster member의 이름이다.

Cluster member의 이름은 해당 member의 database를 생성할 때 정의한 member의 이름과 동일해야 한다.

동일한 cluster group, cluster member 이름이 존재하지 않아야 한다.

이름의 길이가 128 바이트보다 작아야 한다.

Cluster member의 start-up 단계가 OPEN이어야 한다.

<connection attribute>

Cluster member간 통신을 위한 연결 정보를 정의한다.

<connection attribute>는 해당 member의 database를 생성할 때 정의한 HOST, PORT와 동일해야 한다.

HOST와 PORT 조합은 cluster system 내에서 유일해야 한다.

- HOST 'address'는 ip v4 형식을 사용한다.
- PORT port_no는 1024 ~ 49151 범위의 값이어야 한다.

설명

<alter cluster group add member statement> 구문은 table들의 shard를 재배포하지 않는다.

추가된 cluster member에 shard를 재배포하려면 다음 구문을 수행해야 한다.

- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

사용 예

다음은 cluster group에 두 개의 cluster member를 추가하는 예이다.

```
gSQL>
ALTER CLUSTER GROUP g1 ADD
    CLUSTER MEMBER g1n3 HOST '192.168.0.13' PORT 10130,
    CLUSTER MEMBER g1n4 HOST '192.168.0.14' PORT 10140
;
Cluster Group altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE CLUSTER GROUP
- DROP CLUSTER GROUP
- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

16.3 ALTER CLUSTER GROUP name OFFLINE MEMBER

기능

Cluster group의 cluster member를 offline 상태로 변경한다.

구문

```
<alter cluster group offline member statement> ::=  
    ALTER CLUSTER GROUP group_name OFFLINE  
        <cluster member definition>  
    ;  
<cluster member definition> ::=  
    CLUSTER MEMBER member_name  
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter cluster group offline member statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATA BASE 권한이 있어야 한다.

구문 규칙 및 파라미터

group_name

Cluster group의 이름이다.

<cluster member definition>

Cluster group에 포함될 cluster member를 정의한다.

Cluster group은 최대 32개의 cluster member를 포함할 수 있다.

member_name

Cluster member의 이름이다.

Cluster member의 이름은 해당 member의 database를 생성할 때 정의한 member의 이름과 동일해야 한다.

동일한 cluster group, cluster member 이름이 존재하지 않아야 한다.

이름의 길이가 128 바이트보다 작아야 한다.

Cluster member의 start-up 단계가 OPEN이어야 한다.

설명

<alter cluster group offline member statement> 구문은 table들의 shard를 재배포하지 않는다.

사용 예

다음은 특정 cluster member를 offline 상태로 변경하는 예이다.

```
gSQL>
ALTER CLUSTER GROUP g1 OFFLINE
    CLUSTER MEMBER g1n3
;
Cluster Group altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE CLUSTER GROUP
- DROP CLUSTER GROUP
- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

16.4 ALTER CLUSTER LOCATION

기능

Cluster location 정보를 수정한다.

구문

```
<alter cluster location statement> ::=
    ALTER CLUSTER LOCATION member_name
    <cluster connection attribute>
    ;
<cluster connection attribute> ::
    HOST 'address' PORT port_no
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter cluster location statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

member_name

Cluster member의 이름이다.

등록된 cluster location 정보에 동일한 cluster member 이름이 존재해야 한다.

이름의 길이가 128 바이트보다 작아야 한다.

<cluster connection attribute>

Cluster member 간 통신을 위한 연결 정보를 정의한다.

HOST와 PORT 조합은 cluster system 내에서 유일해야 한다.

- HOST 'address'는 ip v4 형식으로 사용한다.
- PORT port_no는 1024 ~ 49151 범위의 값이어야 한다.

설명

만약 cluster location의 접속 정보가 변경될 경우에는 cluster member를 제거하거나 재생성할 필요없이 **ALTER CLUSTER LOCATION**을 이용하여 접속 정보를 변경할 수 있다.

사용 예

```
gSQL>
ALTER CLUSTER LOCATION g1n2
    HOST '192.168.0.12' PORT 10120,
;
Created
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- **CREATE CLUSTER LOCATION**
- **DROP CLUSTER LOCATION**

16.5 ALTER DATABASE ADD LOGFILE

기능

데이터베이스에 로그파일 그룹 또는 로그파일 멤버를 추가한다.

구문

```

<alter database add logfile statement> ::=
    <add logfile member statement>
  | <add logfile group statement>
  ;
<add logfile member statement> ::=
    ALTER DATABASE ADD LOGFILE MEMBER <add logfile clause> [, ...] TO
    <group clause>
<add logfile group statement> ::=
    ALTER DATABASE ADD LOGFILE <group clause> ( 'logfile_name' )
    <size clause> [ REUSE ]
<group clause> ::=
    GROUP integer
<add logfile clause> ::=
    'logfile_name' [ REUSE ]
<size clause> ::=
    integer [ M | G ]

```

사용 범위 및 접근 권한

<alter database add logfile statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database add logfile statement>

데이터베이스는 MOUNT 상태여야 한다.

<add logfile member statement>

기존의 로그파일 그룹에 로그 멤버를 추가한다.

- <add logfile clause>
 - 'logfile_name'은 로그파일 그룹에 추가할 로그파일 멤버의 파일이름이다.
 - 파일이 존재하지 않으면 새로운 파일을 생성한다.
 - 'logfile_name'의 길이는 1024 바이트보다 작아야 한다.
- <group clause>
 - 데이터베이스에 추가할 로그파일 그룹의 식별자를 지정한다.
 - integer는 존재하는 로그파일 그룹의 식별자여야 한다.
 - integer에 해당하는 식별자가 존재하지 않을 경우에는 에러가 발생한다.

<add logfile group statement>

새로운 로그파일 그룹을 추가한다.

- CURRENT 로그파일 그룹의 다음 그룹으로 추가된다.
- <group clause>
 - 데이터베이스에 추가할 로그파일 그룹의 식별자를 지정한다.
 - integer는 존재하지 않는 로그파일 그룹 식별자여야 한다.
 - integer에 해당하는 식별자가 존재할 경우 에러가 발생한다.
- <size clause>
 - 파일의 크기는 최소 10 MB ~ 최대 10 GB까지 지정할 수 있다.
 - 파일의 크기는 로그 버퍼 (redo log buffer)의 크기와 지연 로그 버퍼 (pending log buffer) 크기의 합보다 커야 한다.
- 만약 logfile_name이 이미 존재하고 REUSE 옵션을 사용한 경우, 로그 파일 그룹 내의 다른 멤버와 크기가 동일하다면 기존 로그파일을 재사용한다.

설명

새로운 로그파일 그룹과 로그 멤버가 추가되는 경우 controlfile에 저장되므로 향후 controlfile 손상에 대비해 controlfile을 백업하는 것을 권장한다.

사용 예

다음은 기존 로그파일인 GROUP 3에 두 개의 로그파일 멤버를 추가하는 예이다.

```
ALTER DATABASE ADD LOGFILE MEMBER 'logfile1.log', 'logfile2.log' TO GROUP 3;
```

다음은 로그파일 크기가 100 M이고 로그파일 이름이 'logfile1.log'인 새로운 로그파일 GROUP 4를 데이터베이스에 추가하는 예이다.

```
ALTER DATABASE ADD LOGFILE GROUP 4 ( 'logfile1.log' ) SIZE 100M;
```

노트

로그 그룹을 추가할 때는 한 개의 로그파일을 사용해야 하고 이미 존재하는 그룹에 다수의 로그 파일을 멤버로 추가할 수 있다.

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE DROP LOGFILE
- ALTER DATABASE RENAME LOGFILE

16.6 ALTER DATABASE ARCHIVELOG

기능

데이터베이스의 온라인 로그 파일 archive 설정을 변경한다.

구문

```
<alter database archivelog statement> ::=  
    ALTER DATABASE { ARCHIVELOG | NOARCHIVELOG }  
    ;
```

사용 범위 및 접근 권한

<alter database archivelog statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database archivelog statement>

- 데이터베이스가 MOUNT 상태여야 한다.
- ARCHIVELOG
 - 온라인 로그파일을 archive 한다.
- NOARCHIVELOG
 - 온라인 로그파일을 archive 하지 않는다.

설명

Database를 백업하고 백업을 이용해 복구 (media recovery)하려면 시스템이 ARCHIVELOG 모드로 운용되어야 한다.

사용 예

다음은 데이터베이스를 archive 모드로 설정하는 예이다.

```
ALTER DATABASE ARCHIVELOG;
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE BACKUP
- ALTER TABLESPACE name BACKUP

16.7 ALTER DATABASE BACKUP

기능

데이터베이스 전체 백업 (full backup)을 수행하기 위해 백업 상태를 'ACTIVE' 또는 'INACTIVE'로 설정한다. 그리고 데이터베이스 증분 백업 (incremental backup)과 제어 파일 (control file) 백업을 수행한다.

구문

```
<alter database backup statement> ::=
    <database begin backup statement>
  | <database end backup statement>
  | <database incremental backup statement>
  | <database controlfile backup statement>
  ;

<database begin backup statement> ::=
    ALTER DATABASE BEGIN BACKUP [ AT <domain name> ]
  ;

<database end backup statement> ::=
    ALTER DATABASE END BACKUP [ AT <domain name> ]
  ;

<database incremental backup statement> ::=
    ALTER DATABASE BACKUP INCREMENTAL
        <incremental backup option> [ AT <domain name> ]    ;

<incremental backup option> ::=
    LEVEL integer [ CUMULATIVE | DIFFERENTIAL ]

<database controlfile backup statement> ::=
    ALTER DATABASE BACKUP CONTROLFILE TO 'target_name'
        [ AT <domain name> ]    ;
```

사용 범위 및 접근 권한

<alter database backup statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<database begin backup clause>

데이터베이스를 전체 백업이 가능한 상태로 설정한다.

- 데이터베이스에서 생성되어 사용 중인 ONLINE 상태의 모든 테이블스페이스를 전체 백업이 가능한 상태로 설정한다.
- 데이터베이스는 OPEN 상태여야 하고 ARCHIVELOG 모드로 운영되어야 한다.
- BEGIN BACKUP이 시작된 이후에는 다음과 같이 데이터 파일에 쓰기를 요구하는 연산들은 수행할 수 없다.
 - SHUTDOWN NORMAL
 - OFFLINE/ DROP TABLESPACE
 - ADD/ DROP DATAFILE
- 전체 백업의 상태가 ACTIVE인 상태에서 인스턴스가 비정상 종료될 경우, 다시 시작할 때 미디어 복구를 요구할 수도 있다.

<database end backup clause>

데이터베이스를 전체 백업 불가능한 상태로 설정한다.

- 데이터베이스에서 생성되어 사용 중인 ONLINE 상태의 모든 테이블스페이스를 백업 불가능한 상태로 설정한다.
- 데이터베이스는 OPEN 상태여야 하고, ARCHIVELOG 모드로 운영되어야 한다.

<database incremental backup statement>

- 데이터베이스 증분 백업을 수행한다.
- 데이터베이스는 OPEN 상태여야 하고, ARCHIVELOG로 운영되어야 한다.

<incremental backup option>

- 'integer'는 0 ~ 4 까지 지정할 수 있다.
- LEVEL 0은 CUMULATIVE나 DIFFERENTIAL을 지정할 수 없다.
- CUMULATIVE | DIFFERENTIAL
 - CUMULATIVE
 - 'integer'가 n이면 가장 최근에 LEVEL 0 ~ LEVEL n-1까지 백업한 이후 변경된 모든 페이지들을 백업한다.
 - DIFFERENTIAL
 - 'integer'가 n이면 가장 최근에 LEVEL 0 ~ LEVEL n까지 백업한 이후 변경된 모든 페이지들을 백업한다.
 - 생략하면 DIFFERENTIAL이 기본으로 지정된다.

<database controlfile backup statement>

- 제어 파일 (controlfile)을 백업한다.
 - 'target_name'의 이름은 1024 바이트보다 작아야 한다.
 - 'target_name'이 이미 존재하는 경우 연산에 실패한다.
- 데이터베이스는 OPEN 상태여야 하고, ARCHIVELOG로 운영되고 있어야 한다.

노트

GOLDILOCKS에서 관리하는 'target_name' 길이는 최대 1024 바이트이지만, OS마다 최대 허용하는 파일 이름 길이가 다르기 때문에, 실제 생성가능한 'target_name'의 길이는 1024 바이트보다 작을 수 있다.

<domain name>

- 구문을 수행할 멤버나 그룹의 이름이다.
- 지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

Database의 datafile과 controlfile을 백업한다. BEGIN BACKUP을 수행한 후 OS의 파일 복사로 datafile들을 복사하고 나서 END BACKUP을 수행하여 database를 전체 백업한다. 한편, 증분 백업 파일은 하나의 구문으로 BACKUP _DIR_1 property에 설정된 경로에 생성한다.

사용 예

다음은 전체 백업 상태를 ACTIVE로 설정하는 예이다.

```
ALTER DATABASE BEGIN BACKUP;
```

다음은 전체 백업 상태를 INACTIVE로 설정하는 예이다.

```
ALTER DATABASE END BACKUP;
```

다음은 DIFFERENTIAL을 사용하여 LEVEL 1의 증분 백업을 만드는 예이다.

```
ALTER DATABASE BACKUP INCREMENTAL LEVEL 1 DIFFERENTIAL;
```

다음은 controlfile에 대해 'controlfile.bak' 백업 파일을 만드는 예이다. 절대 경로가 포함되지 않으면 LOG_DIR property에 설정된 경로에 백업 파일을 생성한다.

```
ALTER DATABASE BACKUP CONTROLFILE TO 'controlfile.bak';
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

16.8 ALTER DATABASE CLEAR AUDIT TRAIL

기능

Audit policy 적용으로 인해 누적된 audit record를 삭제 (purge)한다.

구문

```
<clear audit trail statement> ::=  
    ALTER DATABASE CLEAR AUDIT TRAIL  
    ;
```

사용 범위 및 접근 권한

<clear audit trail statement> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

설명

Audit policy를 활성화하면 시간이 지남에 따라 audit trail이 계속 커진다.

Audit trail을 구성하는 테이블들은 MEM_AUX_TBS 테이블스페이스에 저장되는데 audit trail이 계속 커지지 않도록 해야 한다.

Audit Trail 저장

필요에 따라 audit trail 을 저장하려면 다음과 같은 절차에 따라 저장한 후에 audit trail을 삭제해야 한다.

- 최초로 수행할 때

```
CREATE TABLE backup_audit_trail AS SELECT * FROM AUDIT_TRAIL;  
COMMIT;
```

- 반복해서 수행할 때

```
INSERT INTO backup_audit_trail SELECT * FROM AUDIT_TRAIL;
COMMIT;
```

- Audit trail을 삭제할 때

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

사용 예

다음 구문을 사용하여 audit trail을 삭제 (purge)한다.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

호환성

SQL 표준에는 audit policy가 없다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - CREATE AUDIT POLICY
 - DROP AUDIT POLICY
 - ALTER AUDIT POLICY
- Audit policy 활성화/ 비활성화
 - AUDIT POLICY
 - NOAUDIT POLICY
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 삭제: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.9 ALTER DATABASE CLEAR PASSWORD HISTORY

기능

Profile 적용에 따라 누적된 사용자의 비밀번호 변경 이력을 삭제한다.

구문

```
<clear password history statement> ::=  
    ALTER DATABASE CLEAR PASSWORD HISTORY  
    ;
```

사용 범위 및 접근 권한

<clear password history statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

설명

User에 profile을 적용할 때 PASSWORD_REUSE_MAX, PASSWORD_REUSE_TIME의 정책에 따라 사용자의 비밀번호 변경이력이 누적된다.

표 16-1 변경 이력 관리

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	변경 이력 관리
value	value	Value 범위 내의 변경 이력만 관리하고 범위를 벗어난 변경 이력은 자동으로 삭제한다.
value	UNLIMITED	모든 변경 이력을 검사해야 하므로 변경 이력을 누적할 뿐 제거하지 않는다.
UNLIMITED	value	모든 변경 이력을 검사해야 하므로 변경 이력을 누적할 뿐 제거하지 않는다.
UNLIMITED	UNLIMITED	변경 이력을 검사하지 않으므로 관리도 하지 않는다.

<clear password history statement> 구문은 누적된 사용자 비밀번호 변경 이력을 삭제한다.

사용 예

다음은 <clear password history statement> 구문을 수행하는 예이다.

```
gSQL> ALTER DATABASE CLEAR PASSWORD HISTORY;  
Database altered.  
gSQL> COMMIT;  
Commit complete.
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE PROFILE
- CREATE USER

16.10 ALTER DATABASE DELETE BACKUP

기능

증분 백업 (incremental backup)의 백업 정보와 백업 파일을 삭제한다. Database의 모든 증분 백업을 삭제하거나, 더 이상 쓸모 없는 백업 (obsolete backup)을 선택해서 삭제할 수 있다.

구문

```

<alter database delete backup statement> ::=
    ALTER DATABASE DELETE <delete backup list option>
        BACKUP LIST [ <including backup file option> ]
    ;
<delete backup list option> ::=
    OBSOLETE
    | ALL
<including backup file option> ::=
    INCLUDING BACKUP FILES
  
```

사용 범위 및 접근 권한

<alter database delete backup statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database delete backup statement>

데이터베이스가 MOUNT 또는 OPEN 상태여야 한다.

<delete backup list option>

기존 증분 백업들 중에서 삭제할 대상을 선정한다.

- OBSOLETE: 가장 최근의 데이터베이스 'LEVEL 0' 백업 이전에 백업한 데이터베이스 또는 테이블스페이스 백업본들을 삭제 대상으로 선정한다.
- ALL: 전체 증분 백업들을 삭제 대상으로 선정한다.

〈including backup file option〉

- 생략하면 백업 정보만 제어 파일에서 삭제한다.
- 백업 정보뿐만 아니라 백업 파일들도 함께 삭제한다.

설명

OBSOLETE 증분 백업 삭제는 가장 최근의 LEVEL 0 데이터베이스 백업 이전의 증분 백업을 삭제한다. 즉, LEVEL 0이 아닌 증분 백업을 수행할 때는 이전에 수행한 증분 백업을 포함하는 증분 백업이 있더라도 삭제하지 않는다. 왜냐하면 증분 백업을 이용한 불완전 복구를 수행할 때 사용될 수 있기 때문이다.

주의

증분 백업을 삭제할 때 백업 파일까지 함께 삭제하면 증분 백업 정보를 포함하는 백업된 controlfile을 이용하더라도 복구를 수행할 수 없으므로 주의해야 한다.

사용 예

다음은 기존 모든 증분 백업들의 백업정보와 백업파일들을 삭제하는 예이다.

```
ALTER DATABASE DELETE ALL BACKUP LIST INCLUDING BACKUP FILES;
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

16.11 ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS

기능

모든 inactive cluster member들을 제거한다.

구문

```
<alter database drop inactive members statement> ::=
    ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter database drop inactive cluster members statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

설명

모든 inactive cluster member들을 제거한다.

Cluster member의 inactive 상태는 cluster system과 연결되어 있지 않은 상태로서 다음과 같은 상황에서 발생한다.

- 운영 중인 cluster system에서 해당 cluster member에 장애가 발생한 경우
- 해당 cluster member를 구동하지 않고 cluster system의 start-up을 시도한 경우

단, cluster member를 제거할 때 table의 shard가 유실되는 경우에는 inactive cluster member를 제거할 수 없다.

<alter database drop inactive members statement> 구문은 모든 inactive cluster member들이 더 이상 cluster system에 포함될 수 없는 경우에 사용하는 것이 바람직하다.

사용 예

다음은 <alter database drop inactive members statement> 구문을 수행하는 예이다.

```
gSQL> ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS;  
Database altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 ALTER SYSTEM JOIN DATABASE 를 참조한다.

16.12 ALTER DATABASE DROP LOGFILE

기능

데이터베이스에 존재하는 로그파일 그룹이나 멤버를 제거한다.

구문

```

<alter database drop logfile statement> ::=
    <drop logfile group statement>
  | <drop logfile member statement>
  ;
<drop logfile group statement> ::=
    ALTER DATABASE DROP LOGFILE <group clause>
<group clause> ::=
    GROUP integer
<drop logfile member statement> ::=
    ALTER DATABASE DROP LOGFILE MEMBER <logfile_list>
<logfile_list> ::=
    'logfile_name'
  | <logfile_list> , 'logfile_name'

```

사용 범위 및 접근 권한

<alter database drop logfile statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database drop logfile statement>

데이터베이스는 MOUNT 상태여야 한다.

제거하려는 로그파일이 CURRENT 또는 ACTIVE 상태일 때는 에러가 발생한다.

제거한 후에 최소 네 개의 로그파일 그룹이 남아 있어야 한다.

〈drop logfile group statement〉

기존의 로그파일 그룹을 제거한다.

- 〈group clause〉
 - 제거할 로그파일 그룹을 지정한다.
 - integer는 존재하는 로그파일의 식별자여야 한다.
 - integer가 존재하지 않을 경우 에러가 발생한다.

〈drop logfile member statement〉

기존의 로그파일 멤버들을 제거한다.

- 〈logfile_list〉
 - 제거할 로그파일 멤버의 목록이다.
 - 'logfile_name'은 존재하는 이름이어야 한다.
 - 'logfile_name'이 존재하지 않을 경우 에러가 발생한다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 기존 로그파일인 GROUP 3을 제거하는 예이다.

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

다음은 기존 로그파일인 GROUP 3에서 'logfile1.log'와 'logfile2.log'를 제거하는 예이다.

```
ALTER DATABASE DROP LOGFILE MEMBER 'logfile1.log', 'logfile2.log';
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE RENAME LOGFILE

16.13 ALTER DATABASE MOVE SHARD

기능

특정 cluster group의 모든 table들의 shard를 다른 cluster group으로 재배치한다.

구문

```
<alter database move shard statement> ::=
    ALTER DATABASE MOVE SHARD FROM CLUSTER GROUP src_cluster_group
    TO CLUSTER GROUP dest_cluster_group [ ONLINE | OFFLINE ];
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter database move shard statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

src_cluster_group

테이블의 shard를 이동할 cluster group이다.

dest_cluster_group

테이블의 shard를 이동시킬 target cluster group이다.

[ONLINE | OFFLINE]

테이블의 shard를 재배치할 때 DML을 허용할지 여부를 결정한다.

- ONLINE

- INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE
 - INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

다음과 같은 구문을 사용하여 cluster member와 cluster group을 추가할 때 table들의 shard는 재배치하지 않는다.

- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

Cluster group과 cluster member를 추가할 때 재배치되지 않은 모든 테이블들의 shard를 재배치하려면 <alter database rebalance statement> 구문을 수행한다.

Shard를 재배치하지 않은 테이블들에 대해 <alter database move shard statement> 구문을 수행하는 것은 다음과 같은 의미이다.

```
ALTER TABLE t1 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
ALTER TABLE t2 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
ALTER TABLE t3 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
...
...
ALTER TABLE t_n MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
```

특정 테이블의 shard 재배치에 실패하더라도 <alter database move shard statement> 구문은 계속 진행되고 shard 재배치에 성공한 테이블을 rollback 하지 않는다.

따라서 에러에 대해 적절한 조치를 취한 후 <alter database move shard statement> 구문을 다시 수행하면 이미 shard 재배치에 성공한 테이블들은 재배치 대상에 포함되지 않으며, 재배치가 필요한 테이블들의 shard만 재배치한다.

사용 예

다음은 <alter database move shard statement> 구문을 수행하는 예이다.

```
gSQL> ALTER DATABASE MOVE SHARD FROM CLUSTER GROUP G1 TO CLUSTER GROUP G2;  
Database altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE name MOVE SHARD
- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

16.14 ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS

기능

모든 inactive cluster member들을 offline 상태로 변경한다. 즉, 해당 cluster member들에 대한 shard map을 offline 상태로 변경한다.

구문

```
<alter database offline inactive cluster members statement> ::=
    ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter database offline inactive cluster members statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

모든 inactive cluster member들을 offline 상태로 변경한다.

Cluster member의 inactive 상태는 cluster system과 연결되어 있지 않은 상태로서 다음과 같은 상황에서 발생한다.

- 운영 중인 cluster system에서 해당 cluster member에 장애가 발생한 경우
- 해당 cluster member를 구동하지 않고 cluster system의 start-up을 시도한 경우

설명

<alter database offline inactive members statement> 구문은 모든 inactive cluster member 들이 더 이상 cluster system에 포함될 수 없는 경우에 사용하는 것이 바람직하다.

Inactive cluster member가 cluster system에 참여할 수 있으면 **ALTER SYSTEM JOIN DATABASE** 구문을 수행하여 cluster system에 포함시킨다.

Offline 상태로 변경된 cluster member는 join 후에 다음 구문을 사용하여 online 상태로 다시 변경할 수 있다

- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

사용 예

```
gSQL> ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS;
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- **ALTER SYSTEM JOIN DATABASE**
- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

16.15 ALTER DATABASE REBALANCE

기능

모든 table들의 shard를 재배치한다.

구문

```
<alter database rebalance statement> ::=  
    ALTER DATABASE REBALANCE [ ONLINE | OFFLINE ];
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter database rebalance statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

[ONLINE | OFFLINE]

테이블의 shard를 재배치할 때 DML을 허용할지 여부를 결정한다.

- ONLINE
 - INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE
 - INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

다음과 같은 구문을 사용하여 cluster member, cluster group을 추가할 때 table들의 shard를 재배포하지 않는다.

- **CREATE CLUSTER GROUP**
- **ALTER CLUSTER GROUP name ADD MEMBER**

추가된 cluster group과 cluster member에 재배포하지 않은 모든 테이블들의 shard를 재배포하기 위해 <alter database rebalance statement> 구문을 수행한다.

Shard를 재배포하지 않은 테이블들에 대해 <alter database rebalance statement> 구문을 수행하는 것은 다음과 같은 의미이다.

```
ALTER TABLE t1 REBALANCE;
COMMIT;
ALTER TABLE t2 REBALANCE;
COMMIT;
ALTER TABLE t3 REBALANCE;
COMMIT;
...
...
ALTER TABLE t_n REBALANCE;
COMMIT;
```

특정 테이블의 shard 재배포에 실패하더라도 <alter database rebalance statement> 구문은 계속 진행되며 shard 재배포에 성공한 테이블을 rollback 하지 않는다.

따라서 에러에 대해 적절한 조치를 취한 후 <alter database rebalance statement> 구문을 재수행할 경우 이미 shard 재배포에 성공한 테이블들은 재배포 대상에 포함되지 않으며, 재배포가 필요한 테이블들의 shard만 재배포한다.

사용 예

다음은 <alter database rebalance statement> 구문을 수행하는 예이다.

```
gSQL> ALTER DATABASE REBALANCE;
Database altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **ALTER TABLE name REBALANCE**를 참조한다.

16.16 ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP

기능

특정 cluster group에 shard가 포함되지 않도록 모든 테이블의 shard를 재배포한다.

구문

```
<alter database rebalance exclude cluster group statement> ::=
    ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP cluster_group_name [ ONLINE | OFFLINE ];
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter database rebalance exclude cluster group statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

cluster_group_name

테이블들의 shard를 포함하지 않는 cluster group의 이름이다.

지정한 cluster group이 유일한 cluster group인 경우 구문을 수행할 수 없다.

[ONLINE | OFFLINE]

테이블의 shard를 재배포할 때 DML을 허용할지 여부를 결정한다.

- ONLINE
 - INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE

- INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

DROP CLUSTER GROUP 구문을 사용하여 cluster group을 제거하려면 해당 cluster group에 shard가 존재하지 않아야 한다.

해당 cluster group이 shard를 포함하지 않도록 하기 위해 <alter database rebalance exclude cluster group statement> 구문을 수행한다. 해당 cluster group의 shard를 포함한 테이블들에 대해 <alter database rebalance exclude cluster group statement> 구문을 수행하는 것은 다음과 같은 의미이다.

```
ALTER TABLE t1 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
ALTER TABLE t2 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
ALTER TABLE t3 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
...
...
ALTER TABLE t_n REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
```

저장 공간 부족 등으로 인해 <alter database rebalance exclude cluster group statement> 구문이 실패한 경우, shard 배제에 성공한 테이블은 rollback 하지 않는다.

따라서 에러에 대해 적절한 조치를 취한 후 <alter database rebalance exclude cluster group statement> 구문을 다시 수행하면 이미 shard 배제에 성공한 테이블들은 재배치 대상에 포함되지 않으며, 재배치가 필요한 테이블들의 shard만 배제하고 재배치한다.

사용 예

다음은 <alter database rebalance exclude cluster group statement> 구문을 수행하는 예이다.

```
gSQL> ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP g3;
Database altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- **DROP CLUSTER GROUP**
- **ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list**

16.17 ALTER DATABASE RENAME LOGFILE

기능

데이터베이스에서 로그파일의 이름을 수정한다.

구문

```
<alter database rename logfile statement> ::=
    ALTER DATABASE RENAME LOGFILE <logfile_list> TO <logfile_list>
    ;
<logfile_list> ::=
    'logfile_name'
    | <logfile_list> , 'logfile_name'
```

사용 범위 및 접근 권한

<alter database rename logfile statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database rename logfile statement>

- 데이터베이스는 MOUNT 상태여야 한다.
- FROM <logfile_list>
 - 데이터베이스에서 수정할 로그파일들의 이름 목록이다.
- TO <logfile_list>
 - 데이터베이스에서 수정될 로그파일들의 이름 목록이다.
 - <logfile_list>는 존재하는 파일이어야 한다.
 - 파일이 존재하지 않을 경우 에러가 발생한다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 기존 로그파일 'logfile.log'를 'newlogfile.log'로 변경하는 예이다.

```
ALTER DATABASE RENAME LOGFILE 'logfile.log' TO 'newlogfile.log';
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE DROP LOGFILE

16.18 ALTER DATABASE RECOVER

기능

온라인/ archive log file을 사용하여 데이터베이스 내의 전체 데이터파일 (datafile) 또는 일부 데이터파일을 복구한다.

구문

```

<alter database recover statement> ::=
    <complete database recover statement>
  | <datafile recover statement>
  | <complete tablespace recover statement>
  | <incomplete database recover statement>
  ;

<complete database recover statement> ::=
    ALTER DATABASE RECOVER

<datafile recover statement> ::=
    ALTER DATABASE RECOVER DATAFILE <datafile recovery clause>

<datafile recovery clause> ::=
    <datafile recovery object> [, ...]

<datafile recovery object> ::=
    'datafile_name' [<recovery using backup option>] [<recovery corruption option>]

<recovery using backup option> ::=
    USING BACKUP 'backup_datafile_name'

<recovery corruption option> ::=
    CORRUPTION

<complete tablespace recover statement> ::=
    ALTER DATABASE RECOVER TABLESPACE tablespace_name

<incomplete database recover statement> ::=
    <batch incomplete recovery statement>
  | <interactive incomplete recovery statement>
  ;

<batch incomplete recovery statement> ::=
    ALTER DATABASE RECOVER <until clause> [<using backup controlfile option>]

<until clause> ::=

```

```

        UNTIL CHANGE integer
<using backup controlfile option> ::=
    USING BACKUP CONTROLFILE
<interactive incomplete recovery statement> ::=
    ALTER DATABASE <incomplete recovery option> [<using backup controlfile option>]
<incomplete recovery option> ::=
    BEGIN INCOMPLETE RECOVERY
    | END INCOMPLETE RECOVERY
    | RECOVER 'logfile name'
    | RECOVER AUTOMATICALLY
    | RECOVER SUGGESTION
    ;

```

사용 범위 및 접근 권한

<alter database recover statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<complete database recover statement>

온라인 및 archive log file을 이용하여 데이터베이스의 데이터 파일들을 최신 상태로 복구한다.

- ONLINE 상태의 모든 테이블스페이스를 복구한다.
- 데이터베이스는 MOUNT 상태여야하고, ARCHIVELOG 모드여야 한다.
- 필요한 archive log file이 존재하지 않으면 실패한다.

<datafile recover statement>

Immediate option으로 offline 된 tablespace의 datafile, backup된 datafile 또는 backup 중 장애로 인해 archive logfile을 이용한 복구가 필요한 tablespace의 datafile들을 최신상태로 복구한다.

- Datafile은 MOUNT phase 또는 OPEN phase에서 복구할 수 있다.
- OPEN phase에서의 복구는 OFFLINE 상태의 tablespace의 datafile만 가능하고, MOUNT phase에서의 복구는 ONLINE/ OFFLINE 상태의 datafile 모두 가능하다.
- 필요한 archive logfile이 존재하지 않으면 복구에 실패한다.
- <datafile recovery clause>
 - 복구할 한 개 이상의 datafile object list를 기술한다.

- <datafile recovery object>
 - 복구할 datafile 이름과 복구 옵션을 설정한다.
- <recovery using backup option>
 - 복구할 datafile의 backup datafile 이름을 설정한다.
- <recovery corruption option>
 - 복구할 datafile에서 corrupt 된 page만 복구할지 여부를 설정한다.

<complete tablespace recover statement>

테이블스페이스의 데이터 파일들을 최신 상태로 복구한다.

- 테이블스페이스를 복구하려면 데이터베이스가 MOUNT 또는 OPEN 상태여야 한다.
- OPEN 상태에서의 복구는 OFFLINE 상태의 테이블스페이스만 가능하고, MOUNT 상태에서의 복구는 ONLINE/OFFLINE 상태의 테이블스페이스 모두에 가능하다.
- 필요한 archive log file이 존재하지 않으면 복구에 실패한다.
- 다음과 같은 경우에는 테이블스페이스 복구 연산이 필요하다.
 - IMMEDIATE로 OFFLINE 된 테이블스페이스
 - 백업된 데이터 파일을 이용해야 하는 경우
 - 전체 백업 중 장애가 발생한 경우

<incomplete database recover statement>

<batch incomplete database recover statement>

온라인 및 archive log file을 이용하여 데이터베이스의 데이터 파일들을 최신상태가 아닌 특정 시점까지 일괄 복구한다.

- ONLINE 상태의 모든 테이블스페이스를 복구한다.
- 데이터베이스는 MOUNT 상태여야하고, ARCHIVELOG 모드여야 한다.
- 불완전 복구될 시점 이후의 데이터가 존재하는 데이터 파일을 이용하면 실패한다.
- 불완전 복구가 완료되면 반드시 RESETLOGS로 데이터베이스를 OPEN 해야 한다.
- <until clause>
 - 불완전 복구될 특정 시점이다.
 - UNTIL CHANGE: 로그 단위로 불완전 복구 시점을 지정한다.
- <using backup controlfile>
 - Deprecated.

<interactive incomplete database recover statement>

온라인 및 archive log file을 이용하여 데이터베이스의 데이터 파일들을 최신상태가 아닌 특정 시점까지 사용자와 대화식으로 복구한다.

- ONLINE 상태의 모든 테이블스페이스를 복구한다.
- 데이터베이스는 MOUNT 상태여야하고, ARCHIVELOG 모드여야 한다.
- 불완전 복구될 시점 이후의 데이터가 존재하는 데이터파일을 이용하면 실패한다.
- 불완전 복구가 완료되면 반드시 RESETLOGS로 데이터베이스를 OPEN 해야 한다.
- <incomplete recovery option>
 - 로그 파일 단위로 대화식 불완전 복구를 수행하기 위한 옵션이다.
 - BEGIN INCOMPLETE RECOVERY: 불완전 복구를 시작한다.
 - END INCOMPLETE RECOVERY: 불완전 복구를 종료한다.
 - RECOVER 'logfile name': 복구를 수행할 로그 파일을 사용자가 직접 설정한다.
 - RECOVER AUTOMATICALLY: 복구 가능한 모든 archive log file을 복구한다.
 - RECOVER SUGGESTION: 시스템이 추천한 복구를 위해 필요한 archive log file을 복구한다.
- <using backup controlfile>
 - Deprecated

설명

데이터베이스 불완전 복구는 복구 완료 시점을 한 번에 찾아내기 어려우므로 여러 번 수행하여 원하는 복구 시점을 찾아야 한다. 그런데 불완전 복구가 완료된 후 RESETLOGS 옵션으로 데이터베이스를 기동하면 새로운 데이터베이스가 되기 때문에 archive log file과 온라인 redo log file에 대한 복사본을 만든 후에 불완전 복구를 여러 번 수행해야 한다.

사용 예

다음은 전체 데이터베이스를 완전 복구하는 예이다.

```
ALTER DATABASE RECOVER;
```

다음은 데이터 파일을 복구하는 예이다.

```
ALTER DATABASE RECOVER DATAFILE 'test.dbf';
```

다음은 테이블스페이스를 복구하는 예이다.

```
ALTER DATABASE RECOVER TABLESPACE test_tbs;
```

다음은 LSN이 11123인 지점까지 전체 데이터베이스를 불완전 복구하는 예이다.


```
ALTER DATABASE RECOVER UNTIL CHANGE 11123;
```

다음은 복구 가능한 archive log file까지만 복구하는 대화식 불완전 복구의 예이다.

```
ALTER DATABASE BEGIN INCOMPLETE RECOVERY;  
ALTER DATABASE RECOVER AUTOMATICALLY;  
ALTER DATABASE END INCOMPLETE RECOVERY;
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE BACKUP
- ALTER TABLESPACE name BACKUP
- ALTER SYSTEM {MOUNT | OPEN} DATABASE

16.19 ALTER DATABASE REGISTER

기능

복구 불가능한 세그먼트를 데이터베이스에 등록한다.

구문

```

<alter database register statement> ::=
    ALTER DATABASE REGISTER IRRECOVERALBE SEGMENT
        <segment physical identifier list>
    ;
<segment physical identifier list> ::=
    integer
    | <segment physical identifier list> , integer
  
```

사용 범위 및 접근 권한

<alter database register statement> 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter database register statement>

복구 불가능한 세그먼트를 데이터베이스에 등록한다. 해당 구문은 백업이 존재하지 않고 데이터베이스를 복구할 수 없는 경우, 세그먼트를 더 이상 사용하지 않는다는 가정하에 사용될 수 있다.

- 데이터베이스가 MOUNT 상태여야 한다.
- 등록된 세그먼트 식별자 목록은 재시작할 때 초기화된다.
- 서버 재시작에 성공하면 등록된 세그먼트들이 'UNUSABLE' 상태가 되는데 해당 세그먼트들은 반드시 삭제해야 한다.

<segment physical identifier list>

복구 불가능한 세그먼트의 식별자 목록이다.

- Integer: 8 바이트 정수형의 세그먼트 식별자

설명

서버를 비정상 종료하고 재시작할 때 데이터베이스를 복구하는데, 이 때 이전 서비스 단계에서 디스크에 반영되지 못한 페이지들을 복구하기 위해서 REDO 로그들을 이용해 페이지를 다시 수행한다.

REDO 연산을 수행하는 중에 예상하지 못한 실패가 발생한 경우, 이를 무시하고 복구하기 위해 사용될 수 있다.

사용 예

다음은 4028679323648을 식별자로 갖는 세그먼트 복구를 포기하는 예이다.

```
ALTER DATABASE REGISTER IRRECOVERABLE SEGMENT 4028679323648;
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

16.20 ALTER DATABASE RESET LOCAL CLUSTER MEMBER

기능

Local cluster member에서 tablespace 객체를 제외한 부분을 database 생성 시점으로 초기화한다.

구문

```
<alter database reset local cluster member statement> ::=
    ALTER DATABASE RESET LOCAL CLUSTER MEMBER
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

Start-up 과정 중 LOCAL OPEN 단계에서 수행할 수 있다.

<alter database reset local cluster member statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

설명

Local cluster member에서 tablespace 객체를 제외한 부분을 database 생성 시점으로 초기화한다.

Tablespace 객체를 제외하고 사용자가 생성한 모든 객체를 제거한다.

<alter database reset local cluster member statement> 구문은 inactive cluster member를 초기화하고, 새로운 cluster member를 cluster system에 참여시키기 위해 사용한다.

Cluster system과 연결이 끊긴 inactive cluster member 들은 다음과 같이 처리할 수 있다.

- Cluster system에 다시 참여할 수 있는 경우 JOIN 구문을 이용하여 참여시킨다.
 - **ALTER SYSTEM JOIN DATABASE**
- Cluster system에 다시 참여할 수 없는 경우, DROP 구문을 이용하여 cluster system에서 제외한다.

- **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**

이 때, cluster system에서 제외된 cluster member에 해당하는 장비는 다음 두 가지 방법으로 재사용할 수 있다.

- 방법 1: Local cluster member의 database를 다시 생성한다.
- 방법 2: <alter database reset local cluster member statement> 구문을 이용해 local cluster member를 초기화한다.

방법 2는 방법 1보다 tablespace를 재생성하는 비용을 줄일 수 있다.

사용 예

다음은 cluster system에서 제외된 local cluster member를 LOCAL OPEN 단계까지 구동한 후, <alter database reset local cluster member statement> 구문을 이용해 초기화하는 예이다.

```
gSQL> \startup nomount
Startup success
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN LOCAL DATABASE;
System altered.
gSQL> ALTER DATABASE RESET LOCAL CLUSTER MEMBER;
Database altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER SYSTEM JOIN DATABASE
- ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS

16.21 ALTER DATABASE RESTORE

기능

증분 백업을 이용하여 데이터베이스나 테이블스페이스 내의 데이터 파일들을 복원한다.

구문

```

<alter database restore statement> ::=
    <database restore statement>
  | <tablespace restore statement>
  | <controlfile restore statement>
  ;

<database restore statement> ::=
    ALTER DATABASE RESTORE [ <until clause> ]

<until clause> ::=
    UNTIL CHANGE integer

<tablespace restore statement> ::=
    ALTER DATABASE RESTORE TABLESPACE tablespace_name

<controlfile restore statement> ::=
    ALTER DATABASE RESTORE CONTROLFILE FROM 'file_name'

```

사용 범위 및 접근 권한

`alter database restore statement` 구문을 수행하려면 사용자에게 ALTER DATABASE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<database restore statement>

증분 백업을 사용하여 데이터베이스 내의 데이터 파일들을 복원한다.
데이터베이스가 MOUNT 상태여야 한다.

〈tablespace restore statement〉

증분 백업을 사용하여 테이블스페이스 내의 데이터 파일들을 복원한다.

- 데이터베이스가 MOUNT 또는 OPEN 상태여야 한다.
- OPEN 상태에서는 OFFLINE 상태의 테이블스페이스만 복원할 수 있고, MOUNT 상태에서는 ONLINE/ OFFLINE 상태의 테이블스페이스 모두 복원할 수 있다.

〈controlfile restore statement〉

'file_name'을 사용하여 제어파일을 복원한다.

- 데이터베이스가 NOMOUNT 상태여야 한다.
- 'file_name'은 절대 경로를 권장하지만, 만약 상대 경로를 기술한 경우에는 <GOLDILOCKS_HOME>/wal/'file_name'을 이용한다.

설명

전체 백업을 이용한 데이터 파일 복원은 OS 복사 명령으로 백업된 파일을 직접 데이터 파일 경로에 복사하는 방법이다. 증분 백업을 이용한 데이터 파일 복원은 삭제된 데이터 파일이나 이전 데이터 파일들만 복원한다.

사용 예

다음은 증분 백업을 사용하여 데이터베이스를 복원하는 예이다.

```
ALTER DATABASE RESTORE;
```

다음은 증분 백업을 사용하여 테이블스페이스를 복원하는 예이다.

```
ALTER DATABASE RESTORE TABLESPACE test_tbs;
```

다음은 LSN이 11123 보다 작은 증분 백업만 사용하여 데이터베이스를 복원하는 예이다.

```
ALTER DATABASE RESTORE UNTIL CHANGE 11123;
```

다음은 controlfile.bak을 사용하여 제어파일을 복원하는 예이다.

```
ALTER DATABASE RESTORE CONTROLFILE FROM 'controlfile.bak'
```

호환성

SQL 표준에서는 ALTER DATABASE 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER DATABASE BACKUP
- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

16.22 ALTER INDEX

기능

인덱스 정의를 변경한다.

구문

```
<alter index statement> ::=
    <alter index physical attribute statement>
  | <rename index statement>
  | <aging index statement>
  ;
```

사용 범위 및 접근 권한

<alter index statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 인덱스의 소유자
- 인덱스가 속한 테이블의 소유자
- 인덱스가 속한 테이블에 대해 CONTROL TABLE ON TABLE
- 인덱스가 속한 스키마에 대해 (ALTER INDEX 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY INDEX ON DATABASE

구문 규칙 및 파라미터

<alter index physical attribute statement>

인덱스의 물리적 속성을 변경한다.

자세한 내용은 ALTER INDEX name STORAGE 구문을 참조한다.

⟨rename index statement⟩

인덱스 이름을 변경한다.

자세한 내용은 `ALTER INDEX name RENAME TO` 구문을 참조한다.

⟨aging statement⟩

인덱스의 빈 페이지를 삭제한다.

자세한 내용은 `ALTER INDEX name AGING` 구문을 참조한다.

설명

각 세부 구문의 설명을 참조한다.

사용 예

각 세부 구문의 사용 예를 참조한다.

호환성

SQL 표준은 인덱스에 대한 개념을 다루지 않고 있다.

16.23 ALTER INDEX name AGING

기능

인덱스의 빈 페이지를 삭제한다.

구문

```
<aging index statement> ::=  
    ALTER INDEX index_name AGING  
    ;
```

사용 범위 및 접근 권한

<aging index statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 인덱스의 소유자
- 인덱스가 속한 테이블의 소유자
- 인덱스가 속한 테이블에 대해 CONTROL TABLE ON TABLE
- 인덱스가 속한 스키마에 대해 (ALTER INDEX 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY INDEX ON DATABASE

구문 규칙 및 파라미터

index_name

대상 인덱스의 이름이다.

설명

해당 구문은 인덱스 페이지들 중에 모든 키가 삭제된 페이지들을 세그먼트로 반납한다. Aging은 논리적 삭제와 물리적 삭제의 2단계로 진행된다. 논리적 삭제는 인덱스에서 페이지를 지칭하는 연결을 끊는 작업이며 페이지의 마지막

키를 삭제할 당시의 SCN이 시스템의 agable SCN보다 작을 때 수행된다. 이후 물리적 삭제가 이루어지는데 논리적으로 삭제할 때의 SCN이 시스템의 agable SCN보다 작을 때 수행된다.

주의

만약 시스템의 agable SCN이 증가하지 않으면 인덱스 AGING 구문이 성공하더라도 빈 페이지가 삭제되지 않을 수 있다.

사용 예

다음은 인덱스를 aging하는 예이다.

```
gSQL> select index_name, empty_blocks from user_indexes where index_name = 'T1X';
INDEX_NAME EMPTY_BLOCKS
-----
T1X          2
1 row selected.
gSQL> alter index t1x aging;
Index altered.
gSQL> select index_name, empty_blocks from user_indexes where index_name = 'T1X';
INDEX_NAME EMPTY_BLOCKS
-----
T1X          0
1 row selected.
```

호환성

SQL 표준은 인덱스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE INDEX
- ALTER INDEX

- DROP INDEX

16.24 ALTER INDEX name STORAGE

기능

인덱스의 물리적 속성을 변경한다.

구문

```

<alter index physical attribute statement> ::=
    ALTER INDEX index_name
    | <physical attribute clause>
    | [ STORAGE ( <segment attr clause> [...] ) ]
    ;

<physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer

<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>

<size clause> ::=
    integer [ K | M | G | T ]

```

사용 범위 및 접근 권한

<alter index physical attribute statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 인덱스의 소유자
- 인덱스가 속한 테이블의 소유자
- 인덱스가 속한 테이블에 대해 CONTROL TABLE ON TABLE
- 인덱스가 속한 스키마에 대해 (ALTER INDEX 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY INDEX ON DATABASE

구문 규칙 및 파라미터

index_name

대상 인덱스의 이름이다.

<physical attribute clause>

인덱스의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내에 key 삽입으로 인한 페이지 분할 빈도를 조절하기 위해 예약된 공간이다.
 - 인덱스 bottom-up 빌드 시에만 적용된다.
 - 0 에서 99 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 DEFAULT_INDEX_PCTFREE property에 설정된 값을 사용한다.
- INITRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 초기 트랜잭션의 개수이다.
 - 인덱스에 접근하는 사용자의 수가 적을 경우에는 INITRANS를 낮게 설정하고, 동시에 접근하는 사용자가 많을 경우에는 INITRANS를 높게 설정한다.
 - 필요한 경우 설정된 MAXTRANS까지 자동으로 늘어난다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 4 이다.
- MAXTRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 트랜잭션의 최대 개수이다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 8 이다.

<segment attr clause>

인덱스가 저장될 공간에 대한 정보를 기술한다.

- INITIAL integer
 - 정의
 - 인덱스를 생성할 때 초기에 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'INITIAL 100'은 실제 8192 bytes로 동작한다.)

- 이 크기 (TABLESPACE의 EXTENT 크기에 align 된 크기)는 MINEXTENTS의 크기와 같거나 커야 하며, MAXEXTENTS의 크기와 같거나 작아야 한다.
 - 인덱스 bottom-up 빌드 시에만 적용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- NEXT integer
 - 정의
 - 인덱스의 물리적 공간을 추가할 경우 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'NEXT 100'은 실제 8192 bytes로 동작한다.)
 - NEXT는 현재 인덱스가 사용할 수 있는 남은 공간의 크기 (MAXEXTENTS의 크기에서 현재 사용하고 있는 공간의 크기를 뺀 크기)에 따라 아래와 같이 동작한다.
 - 남은 공간의 크기가 0인 경우 더 이상 공간을 확장하지 못한다.
 - 남은 공간의 크기가 0보다 크고 NEXT보다 작은 경우 남은 공간의 크기만큼 할당한다.
 - 남은 공간의 크기가 NEXT 보다 클 경우 NEXT의 크기만큼 할당한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 인덱스가 속한 TABLESPACE의 EXTENT 하나 크기이다.
 - MINSIZE integer
 - 정의
 - 인덱스에서 유지해야할 최소 공간의 크기이다.
 - 이 값은 MAXSIZE의 값과 같거나 작아야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - EXTENT 두 개 크기보다 작을 경우 EXTENT 두 개 크기로 결정된다.
 - 생략할 경우, 기본값은 EXTENT 두 개 크기이다.
 - MAXSIZE integer
 - 정의
 - 인덱스에서 할당받을 수 있는 최대 공간의 크기이다.
 - 이 값은 MINSIZE의 값과 같거나 커야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 EXTENT 크기 * 2147483647 (INT32의 최대 양의 정수)이다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes
- M: Megabytes

- G: Gigabytes
- T: Terabytes

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 인덱스의 물리적 속성을 변경하는 예이다.

```
gSQL> ALTER INDEX idx_t1_id PCTFREE 10 INITRANS 4 MAXTRANS 8;  
Index altered.
```

호환성

SQL 표준은 인덱스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

16.25 ALTER INDEX name RENAME TO

기능

인덱스의 이름을 변경한다.

구문

```
<rename index statement> ::=  
    ALTER INDEX index_name  
        RENAME TO new_index_name  
    ;
```

사용 범위 및 접근 권한

<rename index statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 인덱스의 소유자
- 인덱스가 속한 테이블의 소유자
- 인덱스가 속한 테이블에 대해 CONTROL TABLE ON TABLE
- 인덱스가 속한 스키마에 대해 (ALTER INDEX 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY INDEX ON DATABASE

구문 규칙 및 파라미터

index_name

대상 인덱스의 이름이다.

스키마 이름을 기술할 수 없으며, 기존 인덱스와 동일한 스키마 이름을 갖는다.

new_index_name

새로운 인덱스의 이름이며 스키마 내에서 유일한 인덱스 이름이어야 한다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 인덱스의 이름을 변경하는 예이다.

```
gSQL> ALTER INDEX t1_idx1 RENAME TO idx_t1_id;  
Index altered.
```

호환성

SQL 표준은 인덱스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

16.26 ALTER PROFILE

기능

비밀번호 관리 방법을 변경한다.

구문

```

<alter profile statement> ::=
    ALTER PROFILE profile_name LIMIT
    { <password_parameters>, ... }
    ;

<password parameters> ::=
    FAILED_LOGIN_ATTEMPTS { integer | UNLIMITED | DEFAULT }
    | PASSWORD_LOCK_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_LIFE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_GRACE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_REUSE_MAX { integer | UNLIMITED | DEFAULT }
    | PASSWORD_REUSE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_VERIFY_FUNCTION { <verify_policy> | NULL | DEFAULT }

<verify_policy> ::=
    KISA_VERIFY_FUNCTION
    | ORA12C_VERIFY_FUNCTION
    | ORA12C_STRONG_VERIFY_FUNCTION
    | VERIFY_FUNCTION_11G
    | VERIFY_FUNCTION

<password_parameter_number_interval> ::=
    integer
    | integer / integer
  
```

사용 범위 및 접근 권한

<alter profile statement> 구문을 수행하려면 사용자에게 ALTER PROFILE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

profile_name

변경할 profile의 이름이다.

FAILED_LOGIN_ATTEMPTS

로그인 연속 실패 허용 횟수를 설정한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_LOCK_TIME

로그인에 연속적으로 실패한 후에 계정이 잠기는 기간 (day)을 설정한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_LIFE_TIME

비밀번호의 유효 기간 (day)을 설정한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_GRACE_TIME

PASSWORD_LIFE_TIME 이후에 로그인 할 때 비밀번호 만료를 유예하는 기간을 설정한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_REUSE_MAX

이전 비밀번호를 재사용하려 할 때 재사용할 수 없는 최근 비밀번호 개수를 명시한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_REUSE_TIME

이전 비밀번호를 재사용하기 위해 필요한 경과 기간을 명시한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

PASSWORD_VERIFY_FUNCTION

비밀번호 복잡도 검증 방법을 설정한다.

자세한 내용은 CREATE PROFILE 구문을 참조한다.

사용 예

다음은 계정 잠금을 제어하기 위해 profile을 변경하는 예이다.

```
gSQL> ALTER PROFILE prof1 LIMIT
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 3;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 만료를 제어하기 위해 profile을 변경하는 예이다

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_LIFE_TIME 90
        PASSWORD_GRACE_TIME 7;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 재사용 여부를 제어하기 위해 profile을 변경하는 예이다.

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_REUSE_MAX DEFAULT
        PASSWORD_REUSE_TIME DEFAULT;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 복잡도 검사를 제어하기 위해 profile을 변경하는 예이다.

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준은 profile에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 DROP PROFILE을 참조한다.

16.27 ALTER SEQUENCE

기능

시퀀스를 변경한다.

구문

```

<alter sequence generator statement> ::=
    ALTER SEQUENCE sequence_name <alter sequence generator options>
    ;
<alter sequence generator options> ::=
    <alter sequence generator option> [, ...]
<alter sequence generator option> ::=
    <alter sequence generator restart option>
    | <basic sequence generator option>
<alter sequence generator restart option> ::=
    RESTART [ WITH integer ]
<basic sequence generator option> ::=
    <sequence generator increment by option>
    | <sequence generator maxvalue option>
    | <sequence generator minvalue option>
    | <sequence generator cycle option>
    | <sequence generator cache option>
<sequence generator increment by option> ::=
    INCREMENT BY integer
<sequence generator maxvalue option> ::=
    MAXVALUE integer
    | (NO MAXVALUE | NOMAXVALUE)
<sequence generator minvalue option> ::=
    MINVALUE integer
    | (NO MINVALUE | NOMINVALUE)
<sequence generator cycle option> ::=
    CYCLE
    | (NO CYCLE | NOCYCLE)
<sequence generator cache option> ::=
    CACHE integer
  
```


| (NO CACHE | NOCACHE)

사용 범위 및 접근 권한

〈alter sequence generator statement〉 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 시퀀스의 소유자
- 시퀀스가 속한 스키마에 대해 (ALTER SEQUENCE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY SEQUENCE ON DATABASE

구문 규칙 및 파라미터

sequence_name

변경할 시퀀스의 이름이다.

schema_name.sequence_name과 같이 시퀀스가 속할 스키마를 정의할 수 있으며, schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

〈alter sequence generator restart option〉

시퀀스의 다음 값 (NEXT VALUE)을 설정한다.

단, CREATE SEQUENCE 구문에서 정의한 START WITH의 값은 변경하지 않는다.

- RESTART
 - 값을 명시하지 않을 경우, 〈sequence generator definition〉에서 정의한 START WITH의 값이 시퀀스의 다음 값으로 설정된다.
- RESTART WITH integer
 - integer 값을 시퀀스의 다음 값으로 설정한다.
 - integer 값은 MINVALUE와 MAXVALUE 사이의 값이어야 한다.

〈alter sequence generator restart option〉 절을 명시하지 않은 경우, 시퀀스의 현재값을 기준으로 시퀀스의 속성을 변경한다.

〈sequence generator increment by option〉

시퀀스 번호의 간격을 변경한다.

다음과 같은 제약과 특징을 갖는다.

- 양수 또는 음수를 사용할 수 있으며, 0은 사용할 수 없다.

- 간격의 절대값은 MINVALUE와 MAXVALUE의 차이보다 작아야 한다.
- 양수일 경우 오름차순 시퀀스가 되고 음수일 경우 내림차순 시퀀스가 된다.

〈sequence generator maxvalue option〉

시퀀스로 생성할 수 있는 최대값을 변경한다.

단, MAXVALUE의 값이 시퀀스의 현재값보다 작지 않아야 한다.

- MAXVALUE integer
 - 최대값의 범위는 64bit 정수의 최소값 (−9,223,372,036,854,775,808)에서 64bit 정수의 최대값(+9,223,372,036,854,775,807) 까지이다.
 - START WITH의 값과 같거나 크고, MINVALUE 값보다 커야 한다.
- NO MAXVALUE | NOMAXVALUE
 - 최대값을 다음과 같이 변경한다.
 - 오름차순 시퀀스일 경우, 64 bit 정수의 최대값 (+9,223,372,036,854,775,807)이 된다.
 - 내림차순 시퀀스일 경우, -1이 된다.
 - NO MAXVALUE (SQL 표준)와 NOMAXVALUE는 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.

〈sequence generator minvalue option〉

시퀀스로 생성할 수 있는 최소값을 변경한다.

단, MINVALUE의 값이 시퀀스의 현재값보다 크지 않아야 한다.

- MINVALUE integer
 - 최소값의 범위는 64 bit 정수의 최소값 (−9,223,372,036,854,775,808)에서 64 bit 정수의 최대값(+9,223,372,036,854,775,807) 까지이다.
 - START WITH의 값과 같거나 작고, MAXVALUE 값보다 작아야 한다.
- NO MINVALUE | NOMINVALUE
 - 최소값을 다음과 같이 변경한다.
 - 오름차순 시퀀스일 경우, 1이 된다.
 - 내림차순 시퀀스일 경우, 64 bit 정수의 최소값 (−9,223,372,036,854,775,808)이 된다.
 - NO MINVALUE (SQL 표준)와 NOMINVALUE는 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.

〈sequence generator cycle option〉

시퀀스의 값이 최대값 또는 최소값이 되었을 때, 계속 값을 생성할지 여부를 변경한다.

- CYCLE
 - 오름차순 시퀀스가 최대값이 되었을 때, 최소값부터 다시 생성한다.
 - 내림차순 시퀀스가 최소값이 되었을 때, 최대값부터 다시 생성한다.

- NO CYCLE | NOCYCLE
 - 최대값 또는 최소값이 되었을 때, 시퀀스 값을 생성할 수 없다.
 - NO CYCLE (SQL 표준)과 NOCYCLE은 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.

〈sequence generator cache option〉

시퀀스에 신속하게 접근하기 위해 메모리상에 미리 적재할 시퀀스 값의 개수를 정의한다.

Database를 재구동할 때, 메모리상에 적재한 시퀀스 값은 유실되고 적재한 이후의 값부터 시작된다.

- CACHE integer
 - CACHE 값은 2와 같거나 커야하고
 - CYCLE이 존재할 경우 CACHE 값이 CYCLE의 길이보다 크지 않아야 한다.
 - CYCLE의 길이: $\text{CEIL}(\text{MAXVALUE} - \text{MINVALUE}) / \text{ABS}(\text{INCREMENT})$
- NO CACHE | NOCACHE
 - 메모리 상에 시퀀스값을 미리 적재하지 않는다.

설명

CREATE SEQUENCE 구문에서 정의한 시퀀스 속성 중 START WITH는 변경할 수 없다.

START WITH 속성을 변경하려면 DROP SEQUENCE 구문을 수행한 후에 CREATE SEQUENCE 구문을 사용하여 다시 생성해야 한다.

사용 예

다음은 RESTART 옵션으로 시퀀스 값을 재시작하고 이를 사용하여 ID 값을 새로 부여하는 예이다.

```
gSQL> SELECT id, name FROM t1 ORDER BY 1;
ID NAME
---
10 leekmo
42 mkkim
51 jhkim
172 ehpark
4 rows selected.
gSQL> ALTER SEQUENCE seq1 RESTART;
Sequence altered.
gSQL> UPDATE t1 SET id = seq1.NEXTVAL;
4 rows updated.
```

```
gSQL> SELECT id, name FROM t1 ORDER BY 1;  
ID NAME  
-- -----  
1 leekmo  
2 mkkim  
3 jhkim  
4 ehpark  
4 rows selected.
```

호환성

SQL 표준에서는 CACHE/ NO CACHE 절을 정의하지 않고 있다.

표 16-2 SQL 표준 호환성

Feature ID	설명	지원 여부
T176	Sequence generator support	O
T177	Sequence generator support: simple restart option	O

참조

관련 내용은 다음을 참조한다.

- CREATE SEQUENCE
- DROP SEQUENCE

16.28 ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;

기능

세션에서 재사용하기 위해 catching 된 모든 공간들을 해당 tablespace로 반환한다.

구문

```
<alter session cleanup global temporary segment pool statement> ::=  
    ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL  
    ;
```

설명

수행된 세션에서 segment cache의 segment들만 cleanup한다.

사용 예

다음은 세션 segment cache를 cleanup하는 예이다.

```
gSQL> ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;  
Session altered.
```

호환성

SQL 표준에서는 global temporary table, global temporary index의 segment cache 개념을 정의하지 않고 있다.

참조

관련 내용은 **Global Temporary Table** 을 참조한다.

16.29 ALTER SESSION SET property_name

기능

세션의 프로퍼티 값을 설정한다.

구문

```
<alter session set statement> ::=
    ALTER SESSION SET <property name> { = <property value> | TO DEFAULT }
    ;
```

구문 규칙 및 파라미터

<property name>

설정할 프로퍼티 이름이다.

자세한 내용은 Database Administration 매뉴얼의 **Server Property** 장을 참조한다.

<property value>

설정할 프로퍼티 값이다.

TO DEFAULT

세션 프로퍼티 값을 시스템 프로퍼티 값으로 설정한다.

설명

각 property에 대한 자세한 설명은 Database Administration 매뉴얼의 **Server Property** 장을 참조한다.

사용 예

다음은 HINT_ERROR 속성을 설정하여 hint 구문에 오류가 있을 경우 에러가 발생하는 예이다.

```
gSQL> ALTER SESSION SET HINT_ERROR = ON;
Session altered.
gSQL> SELECT /*+ INDEX( t1, invalid_index ) */ name FROM t1 WHERE id = 1;
ERR-42000(16058): not applicable hint :
SELECT /*+ INDEX( t1, invalid_index ) */ name FROM t1 WHERE id = 1
          *
ERROR at line 1:
```

다음은 세션 프로퍼티 값을 시스템 프로퍼티의 값으로 설정하는 예이다.

```
gSQL> ALTER SESSION SET HINT_ERROR TO DEFAULT;
Session altered.
```

호환성

SQL 표준에서는 세션 프로퍼티 개념을 정의하지 않고 있다.

참조

관련 내용은 ALTER SESSION SET property_name을 참조한다.

16.30 ALTER SYSTEM CHECKPOINT

기능

CHECKPOINT를 수행한다.

구문

```
<alter system checkpoint statement> ::=
    ALTER SYSTEM CHECKPOINT
    [ AT <domain name> ]
    ;
```

사용 범위 및 접근 권한

<alter system checkpoint statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter system checkpoint statement>

CHECKPOINT는 commit된 트랜잭션들이 변경한 모든 데이터가 디스크에 기록되는 것을 보장하는 연산이다.

- 데이터베이스가 OPEN 상태여야 한다.
- 데이터베이스가 TDS 모드여야 한다.
- 전체 백업이 진행중일 때는 변경된 페이지가 데이터 파일에 기록되지 않고, REDO 로그와 제어파일만 디스크에 기록된다. 만약 이러한 상태에서 서버가 비정상 종료되는 경우에는 미디어 복구를 수행해야 한다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

체크포인트 (checkpoint) 연산은 commit 된 트랜잭션들이 변경한 모든 내용을 디스크에 기록함으로써 시스템 장애 시 신속한 복구를 가능하게 한다.

사용 예

다음은 CHECKPOINT를 수행하는 예이다.

```
ALTER SYSTEM CHECKPOINT;
```

호환성

SQL 표준에서는 CHECKPOINT 개념을 정의하지 않고 있다.

16.31 ALTER SYSTEM CLEANUP PLAN

기능

모든 SQL plan을 정리한다.

구문

```
<alter system cleanup plan statement> ::=  
    ALTER SYSTEM CLEANUP PLAN  
    [ AT <domain name> ]  
    ;
```

사용 범위 및 접근 권한

<alter system cleanup plan statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter system cleanup plan statement>

해당 구문 규칙이나 파라미터가 존재하지 않는다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

캐시되어 있는 모든 SQL plan을 정리한다.

단, V\$SQL_CACHE.REF_COUNT가 0보다 큰 plan (prepare된 statement에서 참조하는 plan)들은 정리 대상에서 제외한다.

사용 예

다음은 CLEANUP PLAN을 수행하는 예이다.

```
ALTER SYSTEM CLEANUP PLAN;
```

호환성

SQL 표준에서는 CLEANUP PLAN의 개념을 정의하지 않고 있다.

16.32 ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER MEMBER

기능

복구 불가능한 클러스터 멤버를 지정한다.

구문

```
<alter system irrecoverable cluster member statement> ::=  
    ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER <domain name>  
    ;
```

사용 범위 및 접근 권한

<alter system irrecoverable cluster member statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter system irrecoverable cluster member statement>

해당 구문 규칙이나 파라미터가 존재하지 않는다.

<domain name>

복구 불가능한 멤버 이름이다.

그룹 내의 모든 멤버들을 복구 불가능한 멤버로 지정할 수 없다.

설명

복구 불가능한 멤버로 인해 클러스터 재시작에 실패하는 경우 해당 멤버를 제외하고 시스템을 재시작하기 위해 사용한다. 시스템 재시작에 성공한 후에는 반드시 **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS** 구문을 이용해 해당 멤버를 삭제해야 한다.

사용 예

다음은 IRRECOVERABLE CLUSTER MEMBER를 수행하는 예이다.

```
gSQL> ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER g1n1;
```

호환성

SQL 표준에서는 IRRECOVERABLE CLUSTER MEMBER의 개념을 정의하지 않고 있다.

16.33 ALTER SYSTEM JOIN DATABASE

기능

비활성화된 특정 cluster member를 cluster system에 다시 포함시킨다.

구문

```
<alter system join database statement> ::=
    ALTER SYSTEM JOIN DATABASE
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter system join database statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

설명

Cluster member의 inactive 상태는 cluster system과 연결되어 있지 않은 상태로서 다음과 같은 상황에서 발생한다.

- 운영 중인 cluster system에서 해당 cluster member에 장애가 발생한 경우
- Cluster system에 포함된 cluster member를 구동하지 않고 cluster system start-up을 시도한 경우

특정 cluster member가 inactive인 상태에서 해당 member를 다음과 같은 절차를 거쳐 cluster system에 포함시킬 수 있다.

- 구동되지 않은 cluster member를 local open 단계로 구동한다.

```
$ gsql sys gliese --as sysdba --dsn=G3N2
gSQL> \startup
```

- <alter system join database statement> 구문을 사용하여 cluster system에 포함시킨다.

```
$ gsql sys gliese --as sysdba --dsn=G3N2
gSQL> ALTER SYSTEM JOIN DATABASE;
```

<alter system join database statement> 구문은 local open으로 구동된 inactive cluster member를 shutdown 하지 않고 cluster system에 참여시키려고 할 때 사용한다.

Inactive cluster member가 다시 cluster system에 참여하려면 cluster system과 inactive cluster member의 database 상태가 동일해야 한다.

Cluster system에서 database를 변경하는 트랜잭션이 완료된 후에는 inactive cluster member가 cluster system에 다시 참여할 수 없다.

다수의 inactive cluster member가 존재하는 경우, cluster system을 정상적으로 운영하기 위해 다음과 같은 순서로 inactive cluster member를 정리해야 한다.

1. Cluster system에 참여할 수 있는 inactive cluster member를 JOIN 한다.

```
gSQL> ALTER SYSTEM JOIN DATABASE;
```

2. Cluster system에 참여할 수 없는 inactive cluster member들을 DROP 한다.

```
gSQL> ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS;
```

<alter database drop inactive cluster member statement> 구문을 수행하면 모든 inactive cluster member가 cluster system에서 제거되므로, DROP 하기 전에 참여 가능한 모든 inactive cluster member를 cluster system에 포함시켜야 한다.

사용 예

```
gSQL> ALTER SYSTEM JOIN DATABASE;
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**를 참조한다.

16.34 ALTER SYSTEM {MOUNT | OPEN} DATABASE

기능

데이터베이스를 시스템에 마운트하거나 서비스 가능한 상태로 변경한다.

구문

```

<alter system database statement> ::=
    ALTER SYSTEM <alter system database clause>
    ;
<alter system database clause> ::=
    MOUNT DATABASE
    | OPEN [ <database_scope> ] DATABASE [ <open_database_option> ]
<open_database_option> ::=
    READ WRITE [ NORESETLOGS | RESETLOGS ]
    | READ ONLY
<database_scope> ::=
    LOCAL
    | GLOBAL

```

사용 범위 및 접근 권한

<alter system database statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

〈alter system database clause〉

- MOUNT DATABASE
 - 데이터베이스를 시스템에 마운트한다.
- OPEN DATABASE
 - 데이터베이스를 서비스 가능한 상태로 변경한다.

〈open database option〉

- READ ONLY/ READ WRITE
 - 읽기 쓰기 모드를 지정하여 데이터베이스를 구동한다.
 - 생략된 경우에는 READ WRITE로 구동된다.
- RESETLOGS/ NORESETLOGS
 - 데이터베이스를 복구한 이후에 온라인 redo log를 유지할지 선택한다.
 - NORESETLOGS는 기존 redo log를 유지하는 반면에 RESETLOGS는 이를 초기화한다.
 - 데이터베이스를 불완전 복구한 경우, 반드시 RESETLOGS를 지정해야 한다.
 - 생략된 경우에는 NORESETLOGS가 기본으로 지정된다.

〈database_scope〉

- LOCAL
 - LOCAL 영역 서버를 OPEN 단계로 구동한다.
- GLOBAL
 - GLOBAL 영역, 즉 전체 서버를 OPEN 단계로 구동한다.
- Cluster 환경에서 생략된 경우 GLOBAL로 구동 된다.

사용 예

다음은 읽기 전용으로 데이터베이스를 구동하는 예이다.

```
ALTER SYSTEM OPEN DATABASE READ ONLY;
```

다음은 읽기/ 쓰기 모드로 구동하고, online redo log를 초기화하는 예이다.

```
ALTER SYSTEM OPEN DATABASE READ WRITE RESETLOGS;
```

호환성

SQL 표준에서는 데이터베이스의 MOUNT 또는 OPEN에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 ALTER DATABASE RECOVER를 참조한다.

16.35 ALTER SYSTEM [KILL | DISCONNECT] SESSION

기능

세션을 종료한다.

구문

```
<alter system end session statement> ::=
    ALTER SYSTEM DISCONNECT SESSION [<member_position>,<session_id>,<serial#> [<disconnect_option>] [AT <domain_name>]
| ALTER SYSTEM KILL SESSION [<member_position>,<session_id>,<serial#> [AT <domain_name>] ;
<disconnect_option> ::=
    POST_TRANSACTION
| IMMEDIATE
```

사용 범위 및 접근 권한

<alter system end session statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<member_position>

Cluster 환경에서 disconnect/ kill 대상이 되는 세션의 member position 이다.

<session_id>

세션의 ID 이다.

<serial#>

세션의 SERIAL NUMBER이다.

<disconnect_option>

- POST_TRANSACTION: 트랜잭션 완료 후 세션을 종료한다.
- IMMEDIATE: 트랜잭션 완료를 기다리지 않고 바로 세션을 종료한다.

<disconnect_option>이 사용되지 않으면 IMMEDIATE로 작동한다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

DISCONNECT SESSION은 POST_TRANSACTION과 IMMEDIATE 옵션을 지정할 수 있으며, POST_TRANSACTION은 현재 실행되는 트랜잭션이 있을 경우 트랜잭션이 끝난 후에 세션을 종료한다. IMMEDIATE는 현재 수행 중인 트랜잭션을 바로 정리한 후에 세션을 종료한다.

KILL SESSION은 해당 세션의 프로세스는 존재하지 않지만 시스템에 남아있는 비정상 세션을 종료한다.

사용 예

```
gSQL> SELECT USER_NAME, SESSION_ID, SERIAL_NO, SESSION_STATUS, PROGRAM_NAME FROM V$SESSION
WHERE USER_NAME = 'TEST';
USER_NAME SESSION_ID SERIAL_NO SESSION_STATUS PROGRAM_NAME
-----
TEST          62          49 CONNECTED      gsql
TEST          65         109 CONNECTED      gsqlnet
TEST          66         130 CONNECTED      gsql
3 rows selected.
gSQL> ALTER SYSTEM DISCONNECT SESSION 65, 109;
System altered.
```

호환성

SQL 표준에서는 정의하지 않고 있다.

16.36 ALTER SYSTEM RECONNECT GLOBAL CONNECTION

기능

GLOBAL CONNECTION 형태로 접속한 세션에 재접속할지 여부를 설정한다.

구문

```
<alter system reconnect global connection statement> ::=  
    ALTER SYSTEM RECONNECT GLOBAL CONNECTION  
    ;
```

사용 범위 및 접근 권한

<alter system reconnect global connection statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

설명

GLOBAL CONNECTION 클라이언트의 재접속 여부는 최초 접속할 때 서버로부터 얻은 system 객체의 SCN과 현재 서버의 system 객체의 SCN을 비교하여 결정한다. 해당 구문은 system 객체의 SCN을 상승시켜 클라이언트의 재접속을 유도한다.

해당 구문을 수행한 즉시 클라이언트가 재접속하는 것은 아니다. 클라이언트가 서버에 명령어를 실행할 때 SCN 비교를 통해서 재접속하며 만약 클라이언트에서 모든 멤버로의 연결이 유효하다면 재접속을 시도하지 않는다.

사용 예

다음은 해당 구문을 수행하는 예이다.


```
gSQL> ALTER SYSTEM RECONNECT GLOBAL CONNECTION;  
System altered.
```

호환성

SQL 표준에서는 GLOBAL CONNECTION의 개념을 정의하지 않고 있다.

16.37 ALTER SYSTEM RESET property_name

기능

프로퍼티 파일에서 프로퍼티 값을 삭제한다.

구문

```
<alter system reset statement> ::=
    ALTER SYSTEM { RESET | UNSET } <property name>
        [ SCOPE = { FILE | SPFILE } ]
        [ AT <domain name> ]
    ;
```

사용 범위 및 접근 권한

<alter system reset statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

{ RESET | UNSET }

RESET과 UNSET은 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.

<property name>

삭제할 프로퍼티의 이름이다.

자세한 내용은 Database Administration 매뉴얼의 **Server Property** 장을 참조한다.

[SCOPE = { FILE | SPFILE }]

프로퍼티 파일에서 삭제하는 것이므로 SCOPE=FILE/SPFILE만 사용할 수 있다.

- SCOPE = FILE
 - FILE과 SPFILE은 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.
 - 프로퍼티를 FILE에서 삭제하고, 현재 상태에는 적용하지 않는다.
 - Database를 재구동할 때 변경 사항을 적용한다.

SCOPE 절을 명시하지 않을 경우, 기본값은 SCOPE = FILE 이다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

SCOPE=FILE/SPFILE을 사용하여 프로퍼티를 변경했을 경우, 갱신된 값이 프로퍼티 파일에 저장되고 데이터베이스를 재시작할 때 반영된다.

RESET 할 경우, 프로퍼티 파일에 저장된 해당 갱신값을 파일에서 제거하고 데이터베이스를 재시작할 때 default 값을 사용하도록 한다.

사용 예

다음은 SCOPE=FILE을 사용하여 프로퍼티를 변경하는 예이다.

```
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128 SCOPE=FILE;
System altered.
gSQL> alter system set DEFAULT_INDEX_LOGGING=YES scope=FILE;
System altered.
```

다음은 위에서 변경한 프로퍼티를 삭제하는 예이다.

```
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT SCOPE=FILE;
System altered.
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT SCOPE=SPFILE;
System altered.
```

```
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT;  
System altered.  
gSQL> ALTER SYSTEM RESET DEFAULT_INDEX_LOGGING;  
System altered.  
gSQL> ALTER SYSTEM UNSET DEFAULT_INDEX_LOGGING;  
System altered.
```

호환성

SQL 표준에서는 시스템 프로퍼티 개념을 정의하지 않고 있다.

참조

관련 내용은 `ALTER SYSTEM SET property_name`을 참조한다.

16.38 ALTER SYSTEM SET property_name

기능

시스템의 프로퍼티 값을 설정한다.

구문

```
<alter system set statement> ::=
    ALTER SYSTEM SET <property name> { = <property value> | TO DEFAULT }
        [ DEFERRED ]
        [ SCOPE = [ MEMORY | { FILE | SPFILE } | BOTH ] ]
    [AT <domain name>]
    ;
```

사용 범위 및 접근 권한

<alter system set statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<property name>

설정할 프로퍼티의 이름이다.

자세한 내용은 Database Administration 매뉴얼의 **Server Property** 장을 참조한다.

<property value>

설정할 프로퍼티의 값이다.

TO DEFAULT

시스템 프로퍼티 값을 시스템을 구동할 당시의 최초값으로 설정한다.

[DEFERRED]

변경된 프로퍼티를 적용할 시점을 정의한다.

- DEFERRED
 - 현재 SESSION에는 영향을 주지 않고, 새로 생성되는 SESSION에 적용된다.
 - 프로퍼티의 SYS_MODIFIABLE 속성값이 IMMEDIATE/ DEFERRED일 때 적용 가능하며, 반드시 명시해야 한다.
 - 프로퍼티의 SYS_MODIFIABLE 속성값이 FALSE인 경우 사용할 수 없다.

프로퍼티의 SYS_MODIFIABLE 속성값이 IMMEDIATE일 경우, DEFERRED를 명시하지 않으면 모든 SESSION에 바로 적용된다.

[SCOPE = [MEMORY | { FILE | SPFILE } | BOTH]]

시스템 프로퍼티 변경에 영향을 받는 범위를 정의한다.

- SCOPE = MEMORY
 - 변경 사항이 현재 상태에만 적용되며, database를 다시 구동할 경우 해당 값은 없어진다.
- SCOPE = FILE
 - FILE과 SPFILE은 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.
 - 변경 사항을 FILE에 저장하고 현재 상태에는 적용하지 않는다.
 - Database를 다시 구동할 때 변경을 적용한다.
- SCOPE = BOTH
 - 변경 사항을 FILE에 저장하고 현재 상태에도 적용한다.

SCOPE 절을 명시하지 않을 경우, 기본값은 SCOPE = MEMORY 이다.

프로퍼티의 SYS_MODIFIABLE 속성값이 FALSE인 경우에는 반드시 SCOPE=FILE/SPFILE이라고 명시해야 한다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

관련 내용은 Database Administration 매뉴얼의 **Server Property** 장을 참조한다.

사용 예

다음은 SYS_MODIFIABLE 속성이 DEFERRED인 프로퍼티를 변경하는 예이다.

```
gSQL> ALTER SYSTEM SET HINT_ERROR = ON;
ERR-22000(13019): Invalid property modify mode.(HINT_ERROR)
gSQL> ALTER SYSTEM SET HINT_ERROR = ON DEFERRED;
System altered.
```

다음은 SYS_MODIFIABLE 속성이 FALSE인 프로퍼티를 변경하는 예이다.

```
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128;
ERR-22000(13018): Specified property cannot be modified with this SCOPE option.(PROCESS_MAX_COUNT)
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128 SCOPE=FILE;
System altered.
```

다음은 변경된 속성을 session 접속 당시의 default 값으로 변경하는 예이다.

```
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE=0;
System altered.
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE TO DEFAULT;
System altered.
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE TO DEFAULT DEFERRED;
System altered.
```

호환성

SQL 표준에서는 시스템의 프로퍼티 개념을 정의하지 않고 있다.

참조

관련 내용은 **ALTER SYSTEM RESET property_name**을 참조한다.

16.39 ALTER SYSTEM SWITCH LOGFILE

기능

데이터베이스 내에 있는 CURRENT 상태의 로그파일을 ACTIVE 상태로 변경한다.

구문

```
<alter system switch logfile statement> ::=
    ALTER SYSTEM SWITCH LOGFILE
    [ AT <domain name> ]
    ;
```

사용 범위 및 접근 권한

<alter system switch logfile statement> 구문을 수행하려면 사용자에게 ALTER SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<alter system switch logfile statement>

데이터베이스가 MOUNT 또는 OPEN 상태여야 한다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

기본적으로 CURRENT 상태의 로그파일이 다 채워지면 자동으로 로그 스위치가 발생한다. 해당 구문은 특수한 상황에서 강제로 로그 스위치하고자 할 때 사용된다.

사용 예

```
ALTER SYSTEM SWITCH LOGFILE;
```

호환성

SQL 표준에서는 LOGFILE에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 ALTER SYSTEM {MOUNT | OPEN} DATABASE를 참조한다.

16.40 ALTER TABLE

기능

테이블 정의를 변경한다.

구문

```

<alter table statement> ::=
    <alter table physical attribute statement>
  | <rename table statement>
  | <add column definition>
  | <drop column definition>
  | <alter column definition>
  | <rename column statement>
  | <add table constraint definition>
  | <drop table constraint definition>
  | <alter table constraint definition>
  | <rename table constraint statement>
  | <add table supplemental log statement>
  | <drop table supplemental log statement>
  | <rebalance statement>
  | <move shard statement>
  | <split shard statement>
  | <rename shard statement>
  | <read { only | write } statement>
  ;

```

사용 범위 및 접근 권한

<alter table statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

〈alter table physical attribute statement〉

테이블의 물리적 속성을 변경한다.

자세한 내용은 ALTER TABLE name STORAGE 구문을 참조한다.

〈rename table statement〉

테이블 이름을 변경한다.

자세한 내용은 ALTER TABLE name RENAME TO 구문을 참조한다.

〈add column definition〉

테이블에 column을 추가한다.

자세한 내용은 ALTER TABLE name ADD COLUMN 구문을 참조한다.

〈drop column definition〉

테이블에서 column을 삭제한다.

자세한 내용은 ALTER TABLE name SET UNUSED COLUMN 구문을 참조한다.

〈alter column definition〉

테이블 column의 정의를 변경한다.

자세한 내용은 ALTER TABLE name ALTER COLUMN 구문을 참조한다.

〈rename column statement〉

테이블 column의 이름을 변경한다.

자세한 내용은 ALTER TABLE name RENAME COLUMN 구문을 참조한다.

〈add table constraint definition〉

테이블에 제약 조건을 추가한다.

자세한 내용은 ALTER TABLE name ADD CONSTRAINT 구문을 참조한다.

〈drop table constraint definition〉

테이블 제약 조건을 삭제한다.

자세한 내용은 ALTER TABLE name DROP CONSTRAINT 구문을 참조한다.

〈alter table constraint definition〉

테이블의 제약 조건을 변경한다.

자세한 내용은 ALTER TABLE name ALTER CONSTRAINT 구문을 참조한다.

〈rename table constraint statement〉

테이블 제약 조건의 이름을 변경한다.

자세한 내용은 ALTER TABLE name RENAME CONSTRAINT 구문을 참조한다.

〈add table supplemental log statement〉

테이블의 데이터가 변경되면 redo log에 부가 정보를 추가하도록 설정한다.

자세한 내용은 ALTER TABLE name ADD SUPPLEMENTAL LOG 구문을 참조한다.

〈drop table supplemental log statement〉

테이블의 데이터가 변경되면 redo log에 부가 정보를 추가하지 않도록 설정한다.

자세한 내용은 ALTER TABLE name DROP SUPPLEMENTAL LOG 구문을 참조한다.

〈rebalance statement〉

Cluster 환경에서 테이블의 shard를 재배포하거나 정합성이 깨진 shard를 동기화하여 정합성을 복구한다.

자세한 내용은 ALTER TABLE REBALANCE 구문을 참조한다.

〈move shard statement〉

Cluster 환경에서 테이블의 특정 shard를 특정 cluster group에 재배포한다.

자세한 내용은 ALTER TABLE MOVE SHARD 구문을 참조한다.

〈split shard statement〉

Cluster 환경에서 테이블의 특정 shard를 분산하여 특정 cluster group에 재배포한다.

자세한 내용은 ALTER TABLE SPLIT SHARD 구문을 참조한다.

⟨rename shard statement⟩

Cluster 환경에서 테이블의 특정 shard 이름을 변경한다.

자세한 내용은 **ALTER TABLE name RENAME SHARD** 구문을 참조한다.

⟨read { only | write } statement⟩

테이블에 READ (only | write)을 설정한다.

자세한 내용은 **ALTER TABLE name READ { ONLY | WRITE }** 구문을 참조한다.

설명

각 세부 구문의 설명을 참조한다.

사용 예

각 세부 구문의 사용 예를 참조한다.

호환성

SQL 표준에서는 다음과 같은 구문을 정의하지 않고 있다.

- ⟨alter table physical attribute statement⟩
- ⟨rename table statement⟩
- ⟨rename column statement⟩
- ⟨rename table constraint statement⟩
- ⟨add table supplemental log statement⟩
- ⟨drop table supplemental log statement⟩
- ⟨rebalance statement⟩
- ⟨move shard statement⟩
- ⟨split shard statement⟩
- ⟨rename shard statement⟩
- ⟨read { only | write } statement⟩

16.41 ALTER TABLE name ADD COLUMN

기능

테이블에 column을 추가한다.

구문

```
<add column definition> ::=
    ALTER TABLE table_name ADD [ COLUMN ] <column definition>
  | ALTER TABLE table_name ADD [ COLUMN ] ( <column definition> [, ...] )
  ;
```

사용 범위 및 접근 권한

<add column definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 테이블을 변경하려면 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- Column을 추가할 때 제약 조건을 함께 명시했다면, 다음과 같이 제약 조건을 생성할 수 있는 조건을 만족해야 한다.
 - 제약 조건이 생성될 스키마에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (ADD CONSTRAINT 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
 - 생성할 제약 조건이 key 제약 조건일 경우, 인덱스가 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
 - USAGE TABLESPACE ON DATABASE
- 해당 테이블의 소유자는 추가된 column에 대해 다음과 같은 권한을 갖는다.
 - 추가된 모든 column에 대한 권한
 - SELECT(columns) ON TABLE WITH GRANT OPTION
 - INSERT(columns) ON TABLE WITH GRANT OPTION

- UPDATE(columns) ON TABLE WITH GRANT OPTION
- REFERENCES(columns) ON TABLE WITH GRANT OPTION
- 함께 생성한 제약 조건에 대한 권한
 - 제약 조건의 소유자
 - 제약 조건과 함께 생성된 인덱스의 소유자

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

ADD [COLUMN]

COLUMN 예약어는 생략할 수 있다.

<column definition>

추가할 column을 정의한다.

자세한 내용은 CREATE TABLE 구문의 <column definition> 절을 참조한다.

테이블 내에 동일한 column 이름이 존재하지 않아야 한다.

Column을 정의할 때 DEFAULT 절을 명시할 경우, 모든 row의 기본값을 추가되는 column에 저장한다.

Column을 정의할 때 <identity column specification> 절을 명시한 경우, 모든 row 각각의 자동 생성값을 추가되는 column에 저장한다.

Column을 정의할 때 NOT NULL 제약 조건을 함께 명시한 경우, 테이블을 비우거나 DEFALUT 절 또는 <identity column specification> 절을 함께 기술해야 한다.

(<column definition> [, ...])

다수의 column을 추가한다.

괄호 내부에 다수의 <column definition>을 나열한다.

설명

추가되는 column은 기존 column들의 뒤에 위치한다.

DEFAULT 절이나 <identity column specification>을 명시한 경우, 수행시간은 테이블에 존재하는 row의 개수에 비례하여 증가한다.

사용 예

다음은 하나의 column을 추가하는 예이다.

```
gSQL> ALTER TABLE region ADD COLUMN r_new_comment VARCHAR(152);
Table altered.
```

다음은 다수의 column을 추가하는 예이다.

```
gSQL> ALTER TABLE partsupp ADD COLUMN (
    ps_retailprice NUMERIC(12,2),
    ps_acctbal NUMERIC(12,2), ps_mktsegment CHAR(10) );
Table altered.
```

다음은 identity column과 DEFAULT 절을 갖는 column을 추가하는 예이다.

```
gSQL> ALTER TABLE region ADD COLUMN (
    r_regionkey INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    r_comment VARCHAR(152) DEFAULT 'N/A' );
Table altered.
gSQL> SELECT r_regionkey, r_name, r_comment FROM region;
R_REGIONKEY R_NAME                                R_COMMENT
-----
          1 AFRICA                                N/A
          2 AMERICA                               N/A
          3 ASIA                                  N/A
          4 EUROPE                                N/A
          5 MIDDLE EAST                           N/A

5 rows selected.
```

다음은 지연가능한 제약 조건을 가지는 column을 추가하는 예이다.

```
gSQL> ALTER TABLE t1 ADD COLUMN ( id INTEGER CONSTRAINT t1_uk UNIQUE DEFERRABLE );  
Table altered.  
gSQL> COMMIT;  
Commit complete.
```

호환성

SQL 표준에서는 다수의 column definition 추가에 대해 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name SET UNUSED COLUMN
- ALTER TABLE name ALTER COLUMN
- ALTER TABLE name RENAME COLUMN

16.42 ALTER TABLE name SET UNUSED COLUMN

기능

테이블 column을 제거한다.

구문

```
<drop column definition> ::=
    ALTER TABLE table_name <drop column clause>
    ;
<drop column clause> ::=
    SET UNUSED [ COLUMN ] <column_name_list> [ <drop behavior> ]
<column name list> ::=
    column_name
    | ( column_name [, ...] )
<drop behavior> ::=
    RESTRICT
    | CASCADE
    | CASCADE CONSTRAINTS
```

사용 범위 및 접근 권한

<drop column definition> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

SET UNUSED [COLUMN]

해당 column들을 사용하지 않도록 설정한다.

column_name_list

한 개 이상의 삭제될 column 이름이다.

- 예: ALTER TABLE t1 SET UNUSED COLUMN c1
- 예: ALTER TABLE t1 SET UNUSED COLUMN (c1, c2)

column_name

삭제할 column의 이름이다.

해당 column을 이용하는 제약 조건과 인덱스도 함께 삭제한다.

drop behavior

생략할 경우, 기본값은 RESTRICT 이다.

현재는 RESTRICT/ CASCADE가 동일하게 작동한다.

설명

SET UNUSED COLUMN은 data를 물리적으로 제거하지 않으므로 row의 개수에 관계없이 일정한 성능을 보장한다.

사용 예

다음은 해당 column을 사용하지 않도록 설정하는 예이다.

```
gSQL> ALTER TABLE t1 SET UNUSED COLUMN ( addr );  
Table altered.
```

호환성

SQL 표준에서는 다음과 같은 절들을 정의하지 않고 있다.

- SET UNUSED
- CASCADE CONSTRAINTS
- 다수의 column 나열

표 16-3 SQL 표준 호환성

Feature ID	설명	지원 여부
F033	ALTER TABLE statement: DROP COLUMN clause	X

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name ADD COLUMN
- ALTER TABLE name ALTER COLUMN
- ALTER TABLE name RENAME COLUMN

16.43 ALTER TABLE name ALTER COLUMN

기능

Column의 정의를 변경한다.

구문

```

<alter column definition> ::=
    ALTER TABLE table_name
        ALTER [ COLUMN ] column_name <alter column action>
<alter column action> ::=
    <set column default clause>
    | <drop column default clause>
    | <set column not null clause>
    | <drop column not null clause>
    | <alter column data type clause>
    | <alter identity column specification>
    | <drop identity property clause>
    ;
<set column default clause> ::=
    SET DEFAULT <default option>
<drop column default clause> ::=
    DROP DEFAULT
<set column not null clause> ::=
    SET [ CONSTRAINT constraint_name ] NOT NULL [ <constraint characteristics> ]
<constraint characteristics> ::=
    [ NOT ] DEFERRABLE [ <constraint check time> ]
    | <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
    INITIALLY DEFERRED
    | INITIALLY IMMEDIATE
<drop column not null clause> ::=
    DROP NOT NULL
<alter column data type clause> ::=
    SET DATA TYPE <data type>
<alter identity column specification> ::=

```

```

    <set identity column generation clause> [ <alter identity column option> ... ]
  | <alter identity column option> ...
<set identity column generation clause> ::=
    SET GENERATED { ALWAYS | BY DEFAULT }
<alter identity column option> ::=
    <alter sequence generator restart option>
  | [ SET ] <basic sequence generator option>
<alter sequence generator restart option> ::=
    RESTART [ WITH integer ]
<basic sequence generator option> ::=
    <sequence generator increment by option>
  | <sequence generator maxvalue option>
  | <sequence generator minvalue option>
  | <sequence generator cycle option>
  | <sequence generator cache option>
<drop identity property clause> ::=
    DROP IDENTITY

```

사용 범위 및 접근 권한

<alter column definition> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

ALTER [COLUMN]

COLUMN 예약어는 생략할 수 있다.

column_name

변경할 column의 이름이다.

〈set column default clause〉

Column의 기본값을 설정한다.

identity column이 아니어야 한다.

이후에 수행되는 INSERT 구문 등에서 DEFAULT 절을 사용할 경우 설정된 기본값이 사용된다.

DEFAULT expression의 데이터 타입은 column의 데이터 타입과 호환 가능해야 한다.

타입이 호환되지 않거나 공간이 부족한 경우 INSERT, UPDATE 구문에서 DEFAULT를 사용할 때 에러가 발생한다.

자세한 설명은 CREATE TABLE 구문의 〈default clause〉 절을 참조한다.

〈drop column default clause〉

Column의 기본값을 제거한다.

identity column이 아니어야 한다.

기본값을 제거하면 INSERT 구문 등에서 DEFAULT 절을 사용할 때 NULL 값으로 설정된다.

〈set column not null clause〉

- SET [CONSTRAINT constraint_name] NOT NULL [〈constraint characteristics〉]
 - Column에 NOT NULL 제약 조건을 설정한다.
 - Column의 값으로 NULL 값을 허용하지 않는다.
 - 해당 column에 NULL 값이 존재하지 않아야 한다.
- [CONSTRAINT constraint_name]을 생략할 경우 자동으로 제약 조건 이름을 부여한다.
- 〈constraint characteristics〉를 생략할 경우, NOT DEFERRABLE INITIALLY IMMEDIATE 속성을 갖는다.
- identity column은 DEFERRABLE 속성을 가질 수 없다.

자연 가능한 제약 조건에 대한 자세한 내용은 SET CONSTRAINTS 구문의 설명을 참조한다.

〈drop column not null clause〉

- DROP NOT NULL
 - Column의 NOT NULL 제약 조건을 제거한다.

<alter column data type clause>

- SET DATA TYPE <data type>
 - Column의 데이터 타입을 변경한다.

노트

SET DATA TYPE 구문은 자동으로 commit 되는 DDL 구문이다.

동일한 계열간에 타입을 변경할 수 있는데 이 때 다음 조건을 만족해야 한다.

표 16-4 character string type 변환

from W to	CHAR(n)	VARCHAR(n)	LONG VARCHAR
CHAR(m)	X	X	X
VARCHAR(m)	X	n >= m	X
LONG VARCHAR	X	X	O

char length unit을 변경할 경우 다음과 같은 조건을 만족해야 한다.

표 16-5 character length unit 변환

from W to	OCTETS	CHARACTERS
OCTETS	O	O
CHARACTERS	X	O

표 16-6 binary string type 변환

from W to	BINARY(n)	VARBINARY(n)	LONG VARBINARY
BINARY(m)	X	X	X
VARBINARY(m)	X	n >= m	X
LONG VARBINARY	X	X	O

표 16-7 numeric type

from W to	SMALLINT	INTEGER	BIGINT	NUMERIC	NUMERIC(q)	NUMERIC(q, t)	NUMERIC(q)	NUMBER(q, t)	REAL	DOUBLE PRECISION	FLOAT	FLOAT(q)	NUMBER
						q >= 5		q >= 5					

from W to	SM ALLI NT	INTEGER	BIGINT	NU MER IC	NU MER IC(q)	NU MER IC(q, t)	NU MBE R(q)	NUMBER (q,t)	REA L	DOU BLE P RECIS ION	FLO AT	FLOAT(q)	NU MBE R
SMAL LINT	O	O	O	O	$q \geq 5$	$t \geq 0$ $(q-t) \geq 5$	$q \geq 5$	$t \geq 0$ $(q-t) \geq 5$	O	O	O	$ddc(q) \geq 5$	O
INTE GER	X	O	O	O	$q \geq 10$	$t \geq 0$ $(q-t) \geq 10$	$q \geq 10$	$q \geq 10$ $t \geq 0$ $(q-t) \geq 10$	X	O	O	$ddc(q) \geq 10$	O
BIGIN T	X	X	O	O	$q \geq 19$	$t \geq 0$ $(q-t) \geq 19$	$q \geq 19$	$q \geq 19$ $t \geq 0$ $(q-t) \geq 19$	X	X	O	$ddc(q) \geq 19$	O
NUM ERIC	X	X	X	O	$q == 38$	$q == 38$ $t == 0$	$q == 38$	$q == 38$ $t == 0$	X	X	O	$ddc(q) == 38$	O
NUM ERIC(p)	$5 \geq p$	$10 \geq p$	$19 \geq p$	O	$q \geq p$	$t \geq 0$ $(q-t) \geq p$	$q \geq p$	$q \geq p$ $t \geq 0$ $(q-t) \geq p$	$8 \geq p$	$16 \geq p$	O	$ddc(q) \geq p$	O
NUM ERIC(p,s)	$5 \geq p$ $0 \geq s$ $5 \geq (p-s)$	$10 \geq p$ $0 \geq s$ $10 \geq (p-s)$	$19 \geq p$ $0 \geq s$ $9 \geq (p-s)$	$38 \geq p$ $0 \geq s$ $38 \geq (p-s)$	$q \geq p$ $0 \geq s$ $q \geq (p-s)$	$t \geq s$ $(q-t) \geq (p-s)$	$q \geq p$ $0 \geq s$ $q \geq (p-s)$	$q \geq p$ $t \geq s$ $(q-t) \geq (p-s)$	$8 \geq p$	$16 \geq p$	O	$ddc(q) \geq p$	O
NUM BER(p)	$5 \geq p$	$10 \geq p$	$19 \geq p$	O	$q \geq p$	$t \geq 0$ $(q-t) \geq p$	$q \geq p$	$q \geq p$ $t \geq 0$ $(q-t) \geq p$	$8 \geq p$	$16 \geq p$	O	$ddc(q) \geq p$	O

from to	SM ALLI NT	INTEGER	BIGINT	NUM ERIC	NUM ERIC(q)	NUM ERIC(q, t)	NUM BER(q)	NUMBER (q,t)	REAL	DOUBLE PRECIS ION	FLO AT	FLOAT(q)	NUM BER
						) >= p							
NUM BER(p, s)	5 >= p 0 >= s 5 >= (p-s)	10 >= p 0 >= s 10 >= (p-s)	19 >= p 0 >= s 19 >= (p-s)	38 >= p 0 >= s 3 8 >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	8 >= p	16 >= p	O	ddc(q) >= p	O
REAL	X	X	X	X	X	X	X	X	O	O	O	ddc(q) >= 8	O
DOU BLE P RECIS ION	X	X	X	X	X	X	X	X	X	O	O	ddc(q) >= 16	O
FLOA T	X	X	X	X	X	X	X	X	X	X	O	ddc(q) == 38	O
FLOA T(p)	X	X	X	X	X	X	X	X	8 >= ddc(p)	16 >= ddc(p)	O	ddc(q) >= ddc(p)	O
NUM BER	X	X	X	X	X	X	X	X	X	-	O	ddc(q) == 38	O

FLOAT(p)에 대한 ddc (decimal digit count) 값은 다음과 같다.

표 16-8 ddc (decimal digit count) 값

FLOAT(p)	ddc(p)
1 ~ 3	1
4 ~ 6	2
7 ~ 9	3
10 ~ 13	4
14 ~ 16	5
17 ~ 19	6
20 ~ 23	7
24 ~ 26	8
27 ~ 29	9
30 ~ 33	10
34 ~ 36	11

FLOAT(p)	ddc(p)
37 ~ 39	12
40 ~ 43	13
44 ~ 46	14
47 ~ 49	15
50 ~ 53	16
54 ~ 56	17
57 ~ 59	18
60 ~ 63	19
64 ~ 66	20
67 ~ 69	21
70 ~ 73	22
74 ~ 76	23
77 ~ 79	24
80 ~ 83	25
84 ~ 86	26
87 ~ 89	27
90 ~ 93	28
94 ~ 96	29
97 ~ 99	30
100 ~ 103	31
104 ~ 106	32
107 ~ 109	33
110 ~ 113	34
114 ~ 116	35
117 ~ 119	36
120 ~ 123	37
124 ~ 126	38

참고로 모든 numeric type은 동일한 구조로 관리되며, 각 numeric type은 다음과 같은 NUMBER(p,s) 표현식과 동일하다.

표 16-9 Numeric type들의 NUMBER 표현식

Numeric type	NUMBER(p,s) 표현식
SMALLINT	NUMBER(5,0)
INTEGER	NUMBER(10,0)
BIGINT	NUMBER(19,0)
NUMERIC	NUMBER(38,0)
NUMERIC(p)	NUMBER(p,0)
NUMERIC(p,s)	NUMBER(p,s)
NUMBER(p)	NUMBER(p,0)

Numeric type	NUMBER(p,s) 표현식
NUMBER(p,s)	NUMBER(p,s)
REAL	NUMBER(8,N/A) <= FLOAT(24)
DOUBLE PRECISION	NUMBER(16,N/A) <= FLOAT(53)
FLOAT	NUMBER(38,N/A) <= FLOAT(126)
FLOAT(p)	NUMBER(ddc(p), N/A)
NUMBER	NUMBER(38, N/A)

Native 숫자형은 C 언어의 숫자형 타입과 동일한 데이터 타입이며 서로 다른 타입으로 변환할 수 없다.

표 16-10 Native 숫자형의 변환

from $\mathbb{W}$ to	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBLE
NATIVE_SMALLINT	O	X	X	X	X
NATIVE_INTEGER	X	O	X	X	X
NATIVE_BIGINT	X	X	O	X	X
NATIVE_REAL	X	X	X	O	X
NATIVE_DOUBLE	X	X	X	X	O

표 16-11 Boolean type의 변환

from $\mathbb{W}$ to	BOOLEAN
BOOLEAN	O

표 16-12 Date/ time type의 변환 (TZ: WITH TIME ZONE)

from $\mathbb{W}$ to	DATE	TIME	TIME(g)	TIME TZ	TIME(g) TZ	TIMESTAMP	TIMESTAMP P(g)	TIMESTAMP P TZ	TIMESTAMP(g) TZ
DATE	O	X	X	X	X	X	X	X	X
TIME	X	O	g >= 6	X	X	X	X	X	X
TIME(f)	X	6 >= f	g >= f	X	X	X	X	X	X
TIME TZ	X	X	X	O	g >= 6	X	X	X	X
TIME(f) TZ	X	X	X	6 >= f	g >= f	X	X	X	X
TIMESTAMP	X	X	X	X	X	O	g >= 6	X	X
TIMESTAMP(f)	X	X	X	X	X	6 >= f	g >= f	X	X
TIMESTAMP TZ	X	X	X	X	X	X	X	O	g >= 6
TIMESTAMP(g) TZ	X	X	X	X	X	X	X	6 >= f	g >= f

from $\mathbb{W}$ to	DATE	TIME	TIME(g)	TIME TZ	TIME(g) TZ	TIMESTAMP	TIMESTAMP(g)	TIMESTAMP TZ	TIMESTAMP(g) TZ
f) TZ									

표 16-13 INTERVAL YEAR TO MONTH 계열의 type 변환 (p,q 가 생략된 경우 2)

from $\mathbb{W}$ to	YEAR(q)	MONTH(q)	YEAR(q) TO MONTH
YEAR(p)	$q \geq p$	X	X
MONTH(p)	X	$q \geq p$	X
YEAR(p) TO MONTH	X	X	$q \geq p$

표 16-14 INTERVAL DAY TO TIME 계열의 type 변환 (p,q 가 생략된 경우 2) (f,g 가 생략된 경우 6)

from $\mathbb{W}$ to	DAY(q)	HOUR(q)	MINUTE(q)	SECOND(q, g)	DAY(q) TO HOUR	DAY(q) TO MINUTE	DAY(q) TO SECOND(g)	HOUR(q) TO MINUTE	HOUR(q) TO SECOND(g)	MINUTE(q) TO SECOND(g)
DAY(p)	$q \geq p$	X	X	X	X	X	X	X	X	X
HOUR(p)	X	$q \geq p$	X	X	X	X	X	X	X	X
MINUTE(p)	X	X	$q \geq p$	X	X	X	X	X	X	X
SECOND(p, f)	X	X	X	$q \geq p$ $g \geq f$	X	X	X	X	X	X
DAY(p) TO HOUR	X	X	X	X	$q \geq p$	X	X	X	X	X
DAY(p) TO MINUTE	X	X	X	X	X	$q \geq p$	X	X	X	X
DAY(p) TO SECOND(f)	X	X	X	X	X	X	$q \geq p$ $g \geq f$	X	X	X
HOUR(p) TO MINUTE	X	X	X	X	X	X	X	$q \geq p$	X	X
HOUR(p) TO SECOND(f)	X	X	X	X	X	X	X	X	$q \geq p$ $g \geq f$	X
MINUTE(p) TO SECOND(f)	X	X	X	X	X	X	X	X	X	$q \geq p$ $g \geq f$

표 16-15 ROWID type

from W to	ROWID
ROWID	O

〈alter identity column specification〉

Column의 identity 속성을 변경한다.

Column은 identity column 이어야 한다.

- SET GENERATED [ALWAYS | BY DEFAULT]
 - identity column의 생성 방식을 변경한다.
 - 자세한 내용은 CREATE TABLE 구문의 〈identity column specification〉을 참조한다.
- 〈alter sequence generator restart option〉
 - identity column의 다음 값 (NEXT VALUE)을 변경한다.
 - 자세한 내용은 ALTER SEQUENCE 구문의 〈alter sequence generator restart option〉 절을 참조한다.
- 〈basic sequence generator option〉
 - identity column의 속성을 변경한다.
 - SQL 표준에서는 SET 〈basic sequence generator option〉의 형태로 기술하도록 정의하고 있으나 생략 가능하다.
 - 자세한 내용은 ALTER SEQUENCE 구문을 참조한다.

〈drop identity property clause〉

Column의 identity 속성을 제거한다.

Column은 identity column 이어야 한다.

설명

SET NOT NULL 절의 null 검사 수행시간은 테이블의 row 개수에 비례한다.

다음과 같은 column은 NULL 값을 허용하지 않는다. 즉, DROP NOT NULL 절을 수행하더라도 다음 조건 중 하나를 만족할 경우 NULL 값을 허용하지 않는다.

- NOT NULL 제약 조건이 있는 column
- Column이 primary key 제약 조건에 포함되는 경우
- Column이 identity column인 경우

SET DEFAULT 절을 이용한 기본값 변경과 〈alter identity column specification〉 절을 이용한 identity 속성의 변경은 이후에 수행되는 INSERT 또는 UPDATE 구문에 적용된다.

사용 예

다음은 column에 DEFAULT 속성을 설정하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment SET DEFAULT 'N/A';  
Table altered.
```

다음은 column의 DEFAULT 속성을 제거하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment DROP DEFAULT;  
Table altered.
```

다음은 column에 NOT NULL 제약 조건을 설정하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey SET NOT NULL;  
Table altered.
```

다음은 column의 NOT NULL 제약 조건을 제거하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey DROP NOT NULL;  
Table altered.
```

다음은 column의 data type 크기를 확장하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment SET DATA TYPE VARCHAR(512);  
Table altered.
```

다음은 identity column의 다음 값을 다시 시작하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey RESTART;  
Table altered.
```

다음은 column의 identity 속성을 제거하는 예이다.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey DROP IDENTITY;  
Table altered.
```


호환성

표 16-16 SQL 표준 호환성

Feature ID	설명	지원 여부
F381	Extended schema manipulation	X
F382	Alter column data type	O
F383	Set column not null clause	O
F384	Drop identity property value	O
F385	Drop column generation expression clause	X
F386	Set identity column generation clause	O
S043	Enhanced reference types	X
T174	Identity columns	O
T178	Identity columns: simple restart option	O

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name ADD COLUMN
- ALTER TABLE name SET UNUSED COLUMN
- ALTER TABLE name RENAME COLUMN

16.44 ALTER TABLE name RENAME COLUMN

기능

테이블 column의 이름을 변경한다.

구문

```
<rename column statement> ::=  
    ALTER TABLE table_name  
        RENAME COLUMN old_column_name TO new_column_name  
    ;
```

사용 범위 및 접근 권한

<rename column statement> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있으며, schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

old_column_name

변경할 column의 기존 이름이다.

new_column_name

변경할 column의 새로운 이름이다.

테이블 내에 동일한 column 이름이 존재하지 않아야 한다.

설명

Column 이름이 변경되더라도 이전에 해당 column을 기준으로 생성된 index, constraint 등의 객체를 변경할 필요는 없다.

사용 예

다음은 두 개의 column 이름인 col_1과 col_2를 서로 바꾸는 예이다.

```
gSQL> ALTER TABLE t1 RENAME COLUMN col_1 TO col_temp;
Table altered.
gSQL> ALTER TABLE t1 RENAME COLUMN col_2 TO col_1;
Table altered.
gSQL> ALTER TABLE t1 RENAME COLUMN col_temp TO col_2;
Table altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준에서는 <rename column statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name ADD COLUMN
- ALTER TABLE name SET UNUSED COLUMN
- ALTER TABLE name ALTER COLUMN

16.45 ALTER TABLE name ADD CONSTRAINT

기능

테이블 제약 조건을 추가한다.

구문

```
<add table constraint definition> ::=
    ALTER TABLE table_name
        ADD <table constraint definition>
    ;
```

사용 범위 및 접근 권한

<add table constraint definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 제약 조건을 생성할 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- 제약 조건이 생성될 스키마에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (ADD CONSTRAINT 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- 생성할 제약 조건이 key 제약 조건일 경우, 인덱스가 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
 - USAGE TABLESPACE ON DATABASE
- 생성한 제약 조건의 소유자는 다음과 같이 결정된다.
 - 제약 조건이 속한 스키마의 소유자
 - 제약 조건이 속한 스키마가 PUBLIC 일 경우, 구문을 수행한 사용자

노트

Cluster system에서 PRIMARY KEY와 UNIQUE 제약 조건은 모든 sharding key를 포함해야 한다.

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<table constraint definition>

추가할 제약 조건을 정의한다.

NOT NULL 제약 조건은 ALTER TABLE .. ADD CONSTRAINT 구문으로 추가할 수 없으며, 다음 예와 같이 **ALTER TABLE name ALTER COLUMN** 구문을 이용해 정의할 수 있다.

```
ALTER TABLE t1 ALTER COLUMN c1 SET NOT NULL;
```

자세한 내용은 CREATE TABLE 구문의 <table constraint definition> 절을 참조한다.

설명

Primary key, unique key와 같은 key 제약을 추가할 때 이를 위한 index가 자동으로 생성된다.

사용 예

다음은 테이블에 primary key 제약 조건을 추가하는 예이다.

```
gSQL> ALTER TABLE t1 ADD PRIMARY KEY ( id );
Table altered.
```

다음은 테이블에 primary key 제약 조건을 추가할 때 제약 조건 이름을 명시하는 예이다.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_pk PRIMARY KEY ( id );  
Table altered.
```

다음은 지연 가능한 제약 조건을 추가하는 예이다.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE ( id ) DEFERRABLE INITIALLY DEFERRED;  
Table altered.  
gSQL> COMMIT;  
Commit complete.
```

호환성

표 16-17 SQL 표준 호환성

Feature ID	설명	지원 여부
F381	Extended schema manipulation	O

참조

관련 내용은 다음을 참조한다.

- CREATE TABLE
- CREATE INDEX
- ALTER TABLE
- ALTER TABLE name DROP CONSTRAINT

16.46 ALTER TABLE name DROP CONSTRAINT

기능

테이블 제약 조건을 제거한다.

구문

```
<drop table constraint definition> ::=
    ALTER TABLE table_name
        DROP <constraint object>
        [ <drop behavior> ]
    ;
<constraint object> ::=
    CONSTRAINT constraint_name
    | PRIMARY KEY
    | UNIQUE ( column_name [, ...] )
<drop behavior> ::=
    RESTRICT
    | CASCADE
    | CASCADE CONSTRAINTS
```

사용 범위 및 접근 권한

<drop table constraint definition> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 제약 조건의 소유자
- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있으며, schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

CONSTRAINT constraint_name

제거할 제약 조건의 이름이다.

PRIMARY KEY

테이블의 primary key 제약 조건이다.

UNIQUE(column_name [, ...])

Column들에 대한 unique 제약 조건이다.

<drop behavior>

생략할 경우, 기본값은 RESTRICT 이다.

현재는 RESTRICT/ CASCADE가 동일하게 작동한다.

설명

제약 조건의 이름을 사용하지 않고 NOT NULL 제약 조건을 제거하려고 할 경우 ALTER TABLE name ALTER COLUMN 구문의 <drop column not null clause> 절을 이용한다.

사용 예

다음은 테이블의 primary key 제약 조건을 제거하는 예이다.

```
gSQL> ALTER TABLE t1 DROP PRIMARY KEY;  
Table altered.
```


다음은 제약 조건 이름을 명시하여 테이블의 제약 조건을 제거하는 예이다.

```
gSQL> ALTER TABLE t1 DROP CONSTRAINT t1_pk;
Table altered.
```

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

- DROP PRIMARY KEY
- DROP UNIQUE (column_name [, ...])
- CASCADE CONSTRAINTS

표 16-18 SQL 표준 호환성

Feature ID	설명	지원 여부
F381	Extended schema manipulation	O

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name ADD CONSTRAINT
- DROP INDEX

16.47 ALTER TABLE name ALTER CONSTRAINT

기능

테이블 제약 조건의 특성을 변경한다.

구문

```

<alter table constraint definition> ::=
    ALTER TABLE table_name
        ALTER <constraint object> <constraint characteristics>
    ;
<constraint object> ::=
    CONSTRAINT constraint_name
    | PRIMARY KEY
    | UNIQUE ( column_name [, ...] )
<constraint characteristics> ::=
    [ NOT ] DEFERRABLE [ <constraint check time> ]
    | <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
    INITIALLY DEFERRED
    | INITIALLY IMMEDIATE

```

사용 범위 및 접근 권한

<alter table constraint definition> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

노트

Cluster는 지연 가능한 제약 조건을 지원하지 않는다.

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있으며, schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<constraint object>

변경할 제약 조건은 다음과 같이 지정할 수 있다.

- CONSTRAINT constraint_name
 - 변경할 제약 조건의 이름이다.
- PRIMARY KEY
 - 테이블의 PRIMARY KEY 제약 조건이다.
- UNIQUE(column [,...])
 - Column 목록에 부합되는 UNIQUE 제약 조건이다.

DEFERRABLE | NOT DEFERRABLE

제약 조건의 지연 가능 여부를 변경한다.

- DEFERRABLE
 - 제약 조건을 지연가능하도록 변경한다.
- NOT DEFERRABLE
 - 제약 조건을 지연가능하지 않도록 변경한다.

INITIALLY IMMEDIATE | INITIALLY DEFERRED

제약 조건의 검사시점 초기값을 변경한다.

- INITIALLY IMMEDIATE
 - DML을 수행할 때 제약 조건을 검사한다.
- INITIALLY DEFERRED
 - COMMIT을 수행할 때 제약 조건을 검사한다.

NOT DEFERRABLE로 정의된 제약 조건은 INITIALLY DEFERRED로 변경할 수 없다.

설명

지연가능한 제약 조건에 대한 자세한 내용은 **SET CONSTRAINTS** 구문을 참조한다.

사용 예

다음은 t1_uk 제약 조건을 지연가능하게 하고 검사시점을 DEFERRED로 설정하는 예이다.

```
gSQL> ALTER TABLE t1 ALTER CONSTRAINT t1_uk DEFERRABLE INITIALLY DEFERRED;
Table altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

- ALTER PRIMARY KEY 절
- ALTER UNIQUE(column [,...]) 절

표 16-19 SQL 표준 호환성

Feature ID	설명	지원 여부
F492	Optional table constraint enforcement	X

16.48 ALTER TABLE name RENAME CONSTRAINT

기능

테이블 제약 조건의 이름을 변경한다.

구문

```
<rename table constraint statement> ::=
    ALTER TABLE table_name
        RENAME <constraint object> TO new_constraint_name
    ;
<constraint object> ::=
    CONSTRAINT constraint_name
    | PRIMARY KEY
    | UNIQUE ( column_name [, ...] )
```

사용 범위 및 접근 권한

<rename table constraint statement> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<constraint object>

변경할 제약 조건의 기존 이름은 다음과 같이 지정할 수 있다.

- CONSTRAINT constraint_name
 - 변경할 제약 조건의 이름이다.
- PRIMARY KEY
 - 테이블의 PRIMARY KEY 제약 조건이다.
- UNIQUE(column [,...])
 - Column 목록에 부합되는 UNIQUE 제약 조건이다.

new_column_name

변경할 제약 조건의 새로운 이름이다.

설명

Primary key, unique key와 같이 key 제약 조건으로 자동 생성된 index의 이름은 변경되지 않는다. Index 이름은 **ALTER INDEX name RENAME TO** 구문을 사용하여 변경해야 한다.

사용 예

다음은 테이블의 primary key 제약 조건 이름을 변경하는 예이다.

```
gSQL> ALTER TABLE t1 RENAME PRIMARY KEY TO pk_t1;
Table altered.
```

다음은 제약 조건 이름을 명시하여 테이블의 제약 조건 이름을 변경하는 예이다.

```
gSQL> ALTER TABLE t1 RENAME CONSTRAINT pk_t1 TO t1_pk;
Table altered.
```

호환성

SQL 표준에서는 <rename table constraint statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name ADD CONSTRAINT
- ALTER TABLE name DROP CONSTRAINT
- ALTER TABLE name ALTER CONSTRAINT

16.49 ALTER TABLE name ADD GLOBAL SECONDARY INDEX

기능

테이블에 global secondary index를 생성한다.

구문

```

<alter table add global secondary index definition> ::=
    ALTER TABLE table_name
        ADD GLOBAL SECONDARY INDEX
            [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    ;
<index attributes> ::=
    <physical attribute clause>
    | STORAGE ( <segment attr clause> [...] )
    | <logging clause>
    | <parallel clause>
<physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer
<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>
<size clause> ::=
    integer [ K | M | G | T ]
<logging clause> ::=
    LOGGING
    | NOLOGGING
<parallel clause> ::=
    NOPARALLEL
    | PARALLEL [ integer ]

```


사용 범위 및 접근 권한

<alter table add global secondary index definition> 구문은 cluster system에서 정의할 수 있으며 사용자는 다음 조건들을 만족해야 한다.

- 인덱스를 생성할 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- 인덱스가 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
 - USAGE TABLESPACE ON DATABASE

구문 규칙 및 파라미터

table_name

인덱스를 생성할 테이블 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있으며, schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<physical attribute clause>

인덱스의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내 key 삽입으로 인한 페이지 분할 빈도를 조절하기 위해 예약된 공간이다.
 - 인덱스 bottom-up 빌드 시에만 적용된다.
 - 0에서 99까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 DEFAULT_INDEX_PCTFREE property에 설정된 값을 사용한다.
- INITRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 초기 트랜잭션의 개수이다.
 - 인덱스에 접근하는 사용자의 수가 적을 경우에는 INITRANS를 낮게 설정하고, 동시에 접근하는 사용자가 많을 경우에는 INITRANS를 높게 설정한다.
 - 필요한 경우 설정된 MAXTRANS까지 자동으로 늘어난다.
 - 1 부터 32 까지의 값을 사용할 수 있다.

- 생략할 경우, 기본값은 4 이다.
- MAXTRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 트랜잭션의 최대 개수이다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 8 이다.

〈segment attr clause〉

인덱스가 저장될 공간에 대한 정보를 기술한다.

- INITIAL integer
 - 정의
 - 인덱스를 생성할 때 초기에 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'INITIAL 100'은 실제 8192 bytes로 작동한다.)
 - 이 크기 (TABLESPACE의 EXTENT 크기에 align 된 크기)는 MINEXTENTS의 크기와 같거나 커야 하며, MAXEXTENTS의 크기와 같거나 작아야 한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- NEXT integer
 - 정의
 - 인덱스의 물리적 공간을 추가할 경우 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT크기가 8192 bytes일 경우 'NEXT 100'은 실제 8192 bytes로 작동한다.)
 - NEXT는 현재 인덱스가 사용할 수 있는 남은 공간의 크기 (MAXEXTENTS의 크기에서 현재 사용하고 있는 공간의 크기를 뺀 크기)에 따라 다음과 같이 작동한다.
 - 남은 공간의 크기가 0인 경우 더 이상 공간을 확장하지 못한다.
 - 남은 공간의 크기가 0보다 크고 NEXT보다 작은 경우 남은 공간의 크기만큼 할당한다.
 - 남은 공간의 크기가 NEXT 보다 클 경우 NEXT의 크기만큼 할당한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 인덱스가 속한 TABLESPACE의 EXTENT 하나 크기이다.
- MINSIZE integer
 - 정의
 - 인덱스에서 유지해야 할 최소 공간의 크기이다.
 - 이 값은 MAXSIZE의 값과 같거나 작아야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - EXTENT 두 개 크기보다 작을 경우 EXTENT 두 개 크기로 결정된다.

- 생략할 경우, 기본값은 EXTENT 두 개 크기이다.
- MAXSIZE integer
 - 정의
 - 인덱스에서 할당받을 수 있는 최대 공간의 크기이다.
 - 이 값은 MINSIZE의 값과 같거나 커야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 EXTENT 크기 * 2147483647 (INT32의 최대 양의 정수)이다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

LOGGING | NOLOGGING

인덱스의 리두 로깅 여부를 명시한다.

명시하지 않을 경우, 기본값은 NOLOGGING 이다.

NOPARALLEL | PARALLEL [integer]

인덱스 구축과정에 사용될 thread의 개수를 지정한다.

- NOPARALLEL
 - 병렬로 인덱스를 구축하지 않는다.
- PARALLEL [integer]
 - 병렬로 인덱스를 구축한다.
 - integer가 생략되거나 0으로 지정된 경우에는 프로퍼티 (INDEX_BUILD_PARALLEL_FACTOR)를 따른다.
 - integer는 0부터 사용할 수 있으며 최대값은 16이다.
 - 만약 프로퍼티의 값이 0인 경우에는 최적값을 시스템이 결정한다.
- 명시하지 않을 경우, 기본값은 PARALLEL이다.

TABLESPACE tablespace_name

인덱스가 저장될 tablespace의 이름을 지정한다.

- tablespace_name을 지정할 경우

- LOGGING 인덱스의 tablespace_name은 data tablespace여야 하며
- NOLOGGING 인덱스의 tablespace_name은 temporary tablespace 또는 nologging tablespace여야 한다.
- TABLESPACE 절을 생략할 경우,
 - USER의 INDEX TABLESPACE tablespace_name을 지정한 경우
 - 정의한 테이블스페이스를 사용한다.
 - USER의 INDEX TABLESPACE가 NULL인 경우
 - LOGGING 인덱스는 사용자의 기본 데이터 테이블스페이스를 사용하고,
 - NOLOGGING 인덱스는 사용자의 기본 임시 테이블스페이스를 사용한다.

설명

Non-deterministic 질의에는 global secondary index가 반드시 필요하다. LOGGING 인덱스와 NOLOGGING 인덱스에는 다음과 같은 trade-off가 있다.

- LOGGING 인덱스
 - 장점: 시스템을 구동할 때 로그를 이용해 인덱스가 자동으로 복구되므로 별도의 구축과정이 없다.
 - 단점: 관련 row를 변경할 때 인덱스 변경 내용을 로그로 기록하여 디스크 IO를 유발한다.
- NOLOGGING 인덱스
 - 장점: 관련 row를 변경할 때 인덱스 변경 내용에 대해 디스크 IO가 발생하지 않는다.
 - 단점: 시스템을 구동할 때 인덱스에 대한 로그 정보가 없어 자동으로 인덱스를 재구축한다.

사용 예

다음은 테이블 T1에 global secondary index를 추가하는 예이다.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX;
Table altered.
gSQL> COMMIT;
Commit complete.
```

다음은 logging option으로 테이블 T1의 tablespace USER_DATA_TBS에 global secondary index를 생성하는 예이다.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX LOGGING TABLESPACE USER_DATA_TBS;
Table altered.
gSQL> COMMIT;
Commit complete.
```

다음은 nologging option으로 테이블 T1의 tablespace USER_TEMP_TBS에 global secondary index를 생성하는 예이다.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX NOLOGGING TABLESPACE USER_TEMP_TBS;  
Table altered.  
gSQL> COMMIT;  
Commit complete.
```

호환성

SQL 표준에서는 global secondary index에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE name DROP GLOBAL SECONDARY INDEX
- ALTER TABLE name ALTER GLOBAL SECONDARY INDEX
- CREATE TABLE

16.50 ALTER TABLE name DROP GLOBAL SECONDARY INDEX

기능

테이블에서 global secondary index를 제거한다.

구문

```
<alter table drop global secondary index definition> ::=  
    ALTER TABLE table_name  
        DROP GLOBAL SECONDARY INDEX  
    ;
```

사용 범위 및 접근 권한

<alter table drop global secondary index definition> 구문은 cluster system에서 정의할 수 있으며 사용자는 다음 조건들을 만족해야 한다.

- 인덱스를 제거할 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

인덱스를 제거할 테이블 이름이다.

설명

Non-deterministic 질의를 하려면 global secondary index가 반드시 필요하다.

사용 예

테이블 T1에서 global secondary index를 제거한다.

```
gSQL> ALTER TABLE T1 DROP GLOBAL SECONDARY INDEX;  
Table altered.  
gSQL> COMMIT;  
Commit complete.
```

호환성

SQL 표준에서는 global secondary index에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE name ADD GLOBAL SECONDARY INDEX
- ALTER TABLE name ALTER GLOBAL SECONDARY INDEX

16.51 ALTER TABLE name ALTER GLOBAL SECONDARY INDEX

기능

테이블에서 global secondary index의 물리적 속성을 변경한다.

구문

```

<alter table alter global secondary index storage statement> ::=
    ALTER TABLE table_name ALTER GLOBAL SECONDARY INDEX
        <physical attribute clause>
        | [ STORAGE ( <segment attr clause> [...] ) ]
    ;

<physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer

<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>

<size clause> ::=
    integer [ K | M | G | T ]

```

사용 범위 및 접근 권한

<alter table alter global secondary index storage statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

인덱스를 생성할 테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<physical attribute clause>

인덱스의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내 key 삽입으로 인한 페이지 분할 빈도를 조절하기 위해 예약된 공간이다.
 - 인덱스 bottom-up 빌드 시에만 적용된다.
 - 0 에서 99 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 DEFAULT_INDEX_PCTFREE property에 설정된 값을 사용한다.
- INITRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 초기 트랜잭션의 개수이다.
 - 인덱스에 접근하는 사용자의 수가 적을 경우에는 INITRANS를 낮게 설정하고, 동시에 접근하는 사용자가 많을 경우에는 INITRANS를 높게 설정한다.
 - 필요한 경우 설정된 MAXTRANS까지 자동으로 늘어난다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 4 이다.
- MAXTRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 트랜잭션의 최대 개수이다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 8 이다.

<segment attr clause>

인덱스가 저장될 공간에 대한 정보를 기술한다.

- INITIAL integer
 - 정의
 - 인덱스를 생성할 때 초기에 할당할 물리적 공간의 크기를 기술한다.

- 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'INITIAL 100'은 실제 8192 bytes로 작동한다.)
 - 이 크기 (TABLESPACE의 EXTENT 크기에 align 된 크기)는 MINEXTENTS의 크기와 같거나 커야 하며, MAXEXTENTS의 크기와 같거나 작아야 한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- NEXT integer
 - 정의
 - 인덱스의 물리적 공간을 추가할 경우 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT크기가 8192 bytes일 경우 'NEXT 100'은 실제 8192 bytes로 작동한다.)
 - NEXT는 현재 인덱스가 사용할 수 있는 남은 공간의 크기 (MAXEXTENTS의 크기에서 현재 사용하고 있는 공간의 크기를 뺀 크기)에 따라 다음과 같이 작동한다.
 - 남은 공간의 크기가 0인 경우 더 이상 공간을 확장하지 못한다.
 - 남은 공간의 크기가 0보다 크고 NEXT보다 작은 경우 남은 공간의 크기만큼 할당한다.
 - 남은 공간의 크기가 NEXT 보다 클 경우 NEXT의 크기만큼 할당한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 인덱스가 속한 TABLESPACE의 EXTENT 하나 크기이다.
 - MINSIZE integer
 - 정의
 - 인덱스에서 유지해야할 최소 공간의 크기이다.
 - 이 값은 MAXSIZE의 값과 같거나 작아야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - EXTENT 두 개 크기보다 작을 경우 EXTENT 두 개 크기로 결정된다.
 - 생략할 경우, 기본값은 EXTENT 두개 크기이다.
 - MAXSIZE integer
 - 정의
 - 인덱스에서 할당받을 수 있는 최대 공간의 크기이다.
 - 이 값은 MINSIZE의 값과 같거나 커야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 EXTENT 크기 * 2147483647 (INT32의 최대 양의 정수)이다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes

- M: Megabytes
- G: Gigabytes
- T: Terabytes

설명

Non-deterministic 질의를 하려면 global secondary index가 반드시 필요하다.

사용 예

테이블 T1의 global secondary index가 사용할 최대 크기를 100 Mbyte로 변경한다.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX STORAGE( MAXSIZE 100M );
Table altered.
gSQL> COMMIT;
Commit complete.
```

테이블 T1의 global secondary index가 사용할 INITRANS, MAXTRANS 값을 2와 4로 변경한다.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX INITRANS 2 MAXTRANS 4;
Table altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준에서는 global secondary index에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE name ADD GLOBAL SECONDARY INDEX
- ALTER TABLE name DROP GLOBAL SECONDARY INDEX

16.52 ALTER TABLE name MOVE SHARD

기능

테이블의 특정 shard 또는 특정 cluster group의 전체 shard를 특정 cluster group에 재배포한다.

구문

```
<alter table move shard statement> ::=
    ALTER TABLE table_name MOVE SHARD
        { shard_name_list | FROM CLUSTER GROUP src_cluster_group }
    TO CLUSTER GROUP dest_cluster_group [ ONLINE | OFFLINE ]
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter table move shard statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

해당 테이블이 cluster group specific인 경우에만 구문을 수행할 수 있다.

shard_name_list

재배치할 shard name 목록이다.

해당 테이블에 존재하지 않는 shard인 경우 구문을 수행할 수 없다.

src_cluster_group

재배치할 특정 cluster group의 이름이다.

dest_cluster_group

테이블의 shard를 배치할 target cluster group의 이름이다.

해당 테이블의 shard가 지정한 cluster group에 이미 존재할 경우 구문을 수행할 수 없다.

[ONLINE | OFFLINE]

테이블의 shard를 재배치할 때 DML을 허용할지 여부를 결정한다.

- ONLINE
 - INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE
 - INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

테이블의 특정 shard를 특정 cluster group에서 다른 cluster group으로 재배치한다.

특정 cluster group을 제거하려면 테이블의 shard를 재배치한 후에 **DROP CLUSTER GROUP** 구문을 수행해야 한다.

모든 테이블의 shard를 특정 cluster group에서 다른 cluster group으로 이동시키는 경우, ALTER DATABASE MOVE SHARD FROM CLUSTER GROUP TO CLUSTER GROUP 구문을 수행한다.

사용 예

다음은 <alter table move shard statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t1 MOVE SHARD shard1, shard2 TO CLUSTER GROUP g3;
Table altered.
```

```
gSQL> ALTER TABLE t1 MOVE SHARD FROM CLUSTER GROUP g1 TO CLUSTER GROUP g3;  
Table altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **ALTER DATABASE MOVE SHARD**를 참조한다.

16.53 ALTER TABLE name REBALANCE

기능

테이블의 shard를 재배치한다.

구문

```
<alter table rebalance statement> ::=
    ALTER TABLE table_name REBALANCE [ ONLINE | OFFLINE ]
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter table rebalance statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

[ONLINE | OFFLINE]

테이블의 shard를 재배치할 때 DML을 허용할지 여부를 결정한다

- ONLINE

- INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE
 - INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

다음과 같은 구문을 통해 cluster member, cluster group을 추가할 때 table들의 shard는 재배치하지 않는다.

- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

추가된 cluster group과 cluster member의 테이블 shard를 재배치하려면 <alter table rebalance statement> 구문을 수행한다. 테이블의 shard가 이미 재배치된 경우, 별도의 재배치 작업 없이 실행에 성공한다.

모든 테이블들의 shard를 재배치하려면 **ALTER DATABASE REBALANCE** 구문을 수행한다.

사용 예

다음은 <alter table rebalance statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t1 REBALANCE;
Table altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

16.54 ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list

기능

특정 cluster group에 shard를 포함하지 않도록 테이블의 shard를 재배포한다.

구문

```
<alter table rebalance exclude cluster group statement> ::=
    ALTER TABLE table_name REBALANCE
        EXCLUDE CLUSTER GROUP cluster_group_list [ ONLINE | OFFLINE ]
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter table rebalance exclude cluster group statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

해당 테이블이 cluster-wide인 경우에만 구문을 수행할 수 있다.

cluster_group_list

테이블의 shard를 포함하지 않는 cluster group의 list이다.

재배치에서 제외될 cluster group이 cluster 전체 group인 경우 구문을 수행할 수 없다.

[ONLINE | OFFLINE]

테이블의 shard를 재배치할 때 DML을 허용할지 여부를 결정한다.

- ONLINE
 - INSERT, UPDATE, DELETE를 허용한다.
- OFFLINE
 - INSERT, UPDATE, DELETE를 허용하지 않는다.
- 생략할 경우, 기본값은 ONLINE 이다.

설명

특정 cluster group을 배제하고 테이블의 shard를 재배치한다.

해당 cluster group에 테이블의 shard가 존재하지 않을 경우, 별도의 재배치 작업없이 성공한다.

테이블의 shard가 위치한 cluster group을 기준으로 shard를 재배치한다.

특정 cluster group을 제거하려면 테이블의 shard를 재배치한 후 **DROP CLUSTER GROUP** 구문을 수행해야 한다.

모든 테이블들에서 cluster group을 배제하고 shard를 재배치하고자 할 경우, **ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP**을 수행한다.

사용 예

다음은 <alter table rebalance exclude cluster group statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t1 REBALANCE EXCLUDE CLUSTER GROUP g3;
Table altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

16.55 ALTER TABLE name SPLIT SHARD

기능

Cluster 환경에서 테이블의 특정 shard를 split하여 재배포한다.

구문

```

<alter table split shard statement> ::=
    ALTER TABLE table_name SPLIT SHARD source_shard_name
        INTO ( <split shard placement> [, ...] )
    ;
<split shard placement> ::=
    <split shard bound def> AT CLUSTER GROUP dest_group_name
<split shard bound def> ::=
    <split list shard def>
    | <split range shard def>
<split list shard def> :=
    SHARD dest_shard_name VALUES IN ( <split list value clause> )
<split list value clause> :=
    <split list value> [, ...]
<split list value> :=
    constant
    | NULL
<split range shard def> :=
    SHARD dest_shard_name VALUES LESS THAN ( <split range value clause> )
<split range value clause> :=
    <split range value> [, ...]
<split range value> :=
    constant

```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter table split shard statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

해당 테이블이 cluster group specific이고 list shard 또는 range shard인 경우에만 구문을 수행할 수 있다.

source_shard_name

Split할 원본 shard 이름이다.

해당 테이블에 shard가 존재하지 않을 경우 구문을 수행할 수 없다.

<split shard placement>

원본 shard를 split하여 재배치할 대상 shard를 정의한다.

<split shard bound def>

Split 될 대상 shard의 bound를 정의한다.

다음 두 가지 bound def 중 하나로 정의할 수 있다.

- <split list shard def>
- <split range shard def>

<split list shard def>

List shard를 위한 split shard bound를 정의한다.

- dest_shard_name
 - 대상 shard 이름이다.
- <split list value clause>
 - <split list value>는 상수값이어야 한다.
 - <split list value>는 NULL 값을 사용할 수 없다.

- <split list value>는 DEFAULT를 사용할 수 없다.
- S1 : (1, 11, 21, 31, NULL) SPLIT SHARD S1 INTO (<split list value clause> ..)
 - (O) SHARD S11 VALUES IN (1)
 - (O) SHARD S11 VALUES IN (1, NULL)
 - (O) SHARD S11 VALUES IN (1, 11, 21, 31)
 - (X) SHARD S11 VALUES IN (2)
 - (X) SHARD S11 VALUES IN (DEFAULT)
 - (X) SHARD S11 VALUES IN (1, 11, 21, 31, NULL)

<split range shard def>

Range shard를 위한 split shard bound를 정의한다.

- <split range value clause>
 - <split list value>는 상수값이어야 한다.
 - <split list value>는 NULL 값을 사용할 수 없다.
 - <split list value>는 MAXVALUE를 사용할 수 없다.
 - S1 : (100, 100), S2 : (50, 50) SPLIT SHARD S1 INTO (<split range value clause> ..)
 - (O) SHARD S11 VALUES IN (50, 100)
 - (O) SHARD S11 VALUES IN (100, 50)
 - (O) SHARD S11 VALUES IN (60, 60)
 - (X) SHARD S11 VALUES IN (50, NULL)
 - (X) SHARD S11 VALUES IN (50, 50)
 - (X) SHARD S11 VALUES IN (100, 100)
 - (X) SHARD S11 VALUES IN (100, 110)
 - (X) SHARD S11 VALUES IN (MAXVALUE, 100)

dest_group_name

Split 된 shard가 배치될 cluster group의 이름이다.

설명

특정 테이블의 특정 shard를 분산하여 임의의 cluster group에 배치한다.

특정 shard에 해당하는 레코드가 많거나 특정 group member에 부하가 편중될 때 shard를 분산하여 레코드와 부하를 분산하기 위해 사용된다.

사용 예

다음은 <alter table split shard statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t1 SPLIT SHARD shard1 INTO ( SHARD shard11 VALUES IN ( 11 ) AT CLUSTER GROUP G2 );  
Table altered.  
gSQL> ALTER TABLE t1 SPLIT SHARD shard1 INTO ( SHARD shard11 VALUES LESS THAN ( 11 ) AT CLUSTER GROUP G2 );  
Table altered.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE name REBALANCE
- ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list
- ALTER TABLE name MOVE SHARD

16.56 ALTER TABLE name RENAME SHARD

기능

Cluster 환경에서 테이블의 특정 shard의 이름을 변경한다.

구문

```
<alter table rename shard statement> ::=
    ALTER TABLE table_name
        RENAME SHARD shard_name TO new_shard_name
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<alter table rename shard statement> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

shard_name

변경할 shard의 기존 이름이다.

Shard가 해당 테이블에 존재하지 않을 경우 구문을 수행할 수 없다.

new_shard_name

변경할 shard의 새 이름이다.

테이블 내에 동일한 shard 이름이 존재하지 않아야 한다.

설명

Hash, range, list 테이블의 특정 shard의 이름을 변경한다. Cloned 테이블에 대해서는 해당 구문을 수행할 수 없다.

사용 예

다음은 <alter table rename shard statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t_range RENAME SHARD r_01 TO r_new_01;
Table altered.
gSQL> ALTER TABLE t_list RENAME SHARD l_01 TO l_new_01;
Table altered.
gSQL> ALTER TABLE t_hash RENAME SHARD shard_000000 TO h_new_00;
Table altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLE
- ALTER TABLE name MOVE SHARD
- ALTER TABLE name SPLIT SHARD
- ALTER TABLE name REBALANCE

16.57 ALTER TABLE name READ { ONLY | WRITE }

기능

테이블에 READ { ONLY | WRITE }을 설정한다.

구문

```
<alter table read { only | write } statement> ::=  
    ALTER TABLE table_name  
        READ { ONLY | WRITE }  
    ;
```

사용 범위 및 접근 권한

<alter table read { only | write } statement> 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

테이블 속성을 READ { ONLY | WRITE } 으로 지정한다.

READ ONLY로 지정하면 SELECT .. FOR UPDATE 구문, 테이블의 데이터가 변경되는 DML 또는 DDL 구문들을 사용할 수 없다. 반면, 테이블의 데이터가 변경이 되지 않는 DDL 구문은 허용한다.

노트

READ ONLY로 지정할 때 허용하지 않는 SQL 구문

- INSERT, UPDATE, DELETE
- TRUNCATE
- SELECT .. FOR UPDATE
- ALTER TABLE RENAME/DROP COLUMN
- ALTER TABLE SET COLUMN UNUSED

READ ONLY로 지정할 때 허용하는 SQL 구문

- SELECT
- CREATE/ALTER/DROP INDEX
- ALTER TABLE ADD/ALTER COLUMN
- ALTER TABLE ADD/ALTER/RENAME/DROP CONSTRAINT
- ALTER TABLE for physical property changes
- ALTER TABLE DROP UNUSED COLUMNS
- ALTER TABLE RENAME TO
- DROP TABLE
- ALTER TABLE ADD/DROP SUPPLEMENTAL LOG
- LOCK TABLE

사용 예

다음은 <alter table read { only | write } statement> 구문을 수행하는 예이다.

```
gSQL> ALTER TABLE t1 READ ONLY;  
Table altered.  
gSQL> ALTER TABLE t1 READ WRITE;  
Table altered.
```

호환성

SQL 표준에서는 <alter table read { only | write } statement> 구문을 정의하지 있지 않다.

참조

관련 내용은 **ALTER TABLE**을 참조한다.

16.58 ALTER TABLE name RENAME TO

기능

테이블의 이름을 변경한다.

구문

```
<rename table statement> ::=  
    ALTER TABLE table_name  
        RENAME TO new_table_name  
    ;
```

사용 범위 및 접근 권한

<rename table statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

테이블의 기존 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

new_table_name

테이블의 새 이름이다.

스키마 내에 동일한 테이블 이름이 존재하지 않아야 한다.

설명

테이블 이름이 변경되더라도 이를 참조하는 index constraint 등의 객체는 변경할 필요없다.

사용 예

다음은 두 테이블 t1, t2의 이름을 서로 바꾸는 예이다.

```
gSQL> ALTER TABLE t1 RENAME TO t_temp;
Table altered.
gSQL> ALTER TABLE t2 RENAME TO t1;
Table altered.
gSQL> ALTER TABLE t_temp RENAME TO t2;
Table altered.
gSQL> COMMIT;
Commit complete.
```

호환성

SQL 표준에서는 <rename table statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 **ALTER TABLE**을 참조한다.

16.59 ALTER TABLE name STORAGE

기능

테이블의 물리적 속성을 변경한다.

구문

```

<alter table physical attribute statement> ::=
    ALTER TABLE table_name
        [ <physical attribute clause> ]
    | [ STORAGE ( <segment attr clause> [...] ) ]
    ;

<physical attribute clause> ::=
    PCTFREE integer
    | PCTUSED integer
    | INITTRANS integer
    | MAXTRANS integer

<segment attr clause> ::=
    NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>

<size clause> ::=
    integer [ K | M | G | T ]
  
```

사용 범위 및 접근 권한

<alter table physical attribute statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

〈physical attribute clause〉

테이블을 구성하는 page의 물리적 속성을 변경한다.

이미 할당된 page에는 적용되지 않으며 새로 할당받는 page에 적용된다.

자세한 설명은 CREATE TABLE 구문의 〈table physical attribute clause〉 절을 참조한다.

〈segment attr clause〉

세그먼트를 구성하는 extent의 물리적 속성을 변경한다.

이미 할당된 extent에는 적용되지 않으며, 새로 할당받는 extent에 적용된다.

- MAXSIZE integer
 - 할당될 수 있는 세그먼트 공간의 크기를 변경한다.
 - 이미 할당되어 있는 공간보다 작은 크기를 지정하면, MAXSIZE가 현재 할당되어 있는 크기로 변경된다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 테이블의 물리적 속성을 변경하는 예이다.

```
gSQL> ALTER TABLE t1 PCTFREE 10 PCTUSED 40 STORAGE ( NEXT 10M MAXSIZE 100M );
Table altered.
```


호환성

SQL 표준에서는 테이블의 물리적 속성에 대하여 정의하지 않고 있다.

참조

관련 내용은 **ALTER TABLE**을 참조한다.

16.60 ALTER TABLE name ADD SUPPLEMENTAL LOG

기능

테이블의 데이터가 변경될 때 테이블에 primary key가 있으면 redo log에 primary key 값을 추가하도록 설정한다.

구문

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

사용 범위 및 접근 권한

<add table supplemental log statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

테이블에 primary key가 존재하지 않더라도 구문을 수행할 수 있다.

설명

해당 TABLE에 UPDATE/ DELETE를 수행할 때 SUPPLEMENTAL LOG를 추가로 기록하도록 한다. 기록된 SUPPLEMENTAL LOG는 CDC와 같은 tool 또는 로그를 분석할 때 사용된다.

모든 TABLE의 SUPPLEMENTAL LOG를 기록하려면 *SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY = YES* 로 설정한다.

사용 예

다음은 테이블의 data를 변경할 때 redo log에 primary key 값을 추가하도록 설정하는 예이다.

```
gSQL> ALTER TABLE t1 ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS;  
Table altered.
```

호환성

SQL 표준에서는 <add table supplemental log statement>를 다루지 않는다.

참조

관련 내용은 ALTER TABLE name DROP SUPPLEMENTAL LOG를 참조한다.

16.61 ALTER TABLE name DROP SUPPLEMENTAL LOG

기능

테이블의 데이터가 변경될 때 redo log에 primary key 정보를 남기지 않도록 설정한다.

구문

```
<add table supplemental log statement> ::=
ALTER TABLE table_name
    DROP SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
;
```

사용 범위 및 접근 권한

<drop table supplemental log statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

변경할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

ALTER TABLE name ADD SUPPLEMENTAL LOG 구문을 사용해 설정된 상태여야 한다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 테이블의 데이터가 변경되었을 때 redo log에 primary key 정보를 남기지 않도록 설정하는 예이다.

```
gSQL> ALTER TABLE t1 DROP SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS;  
Table altered.
```

호환성

SQL 표준에서는 <drop table supplemental log statement>를 다루지 않는다.

16.62 ALTER TABLESPACE

기능

테이블스페이스의 정의를 변경한다.

구문

```
<alter tablespace statement> ::=  
    <rename tablespace statement>  
  | <backup tablespace statement>  
  | <on/offline tablespace statement>  
  | <add file statement>  
  | <drop file statement>  
  | <rename datafile statement>  
  ;
```

사용 범위 및 접근 권한

<alter tablespace statement> 구문을 수행하려면 사용자에게 database에 대한 ALTER TABLESPACE 권한이 있어야 한다.

구문 규칙 및 파라미터

<rename tablespace statement>

테이블스페이스의 이름을 변경한다.

자세한 내용은 ALTER TABLESPACE name RENAME TO 구문을 참조한다.

<backup tablespace statement>

테이블스페이스를 백업한다.

자세한 내용은 ALTER TABLESPACE name BACKUP 구문을 참조한다.

〈on-offline tablespace statement〉

테이블스페이스의 모든 파일을 online 또는 offline으로 변경한다.

자세한 내용은 ALTER TABLESPACE name [ONLINE|OFFLINE] 구문을 참조한다.

〈add file statement〉

테이블스페이스에 파일을 추가한다.

자세한 내용은 ALTER TABLESPACE name ADD [DATAFILE|MEMORY] 구문을 참조한다.

〈drop file statement〉

테이블스페이스의 파일을 제거한다.

자세한 내용은 ALTER TABLESPACE name DROP [DATAFILE|MEMORY] 구문을 참조한다.

〈rename datafile statement〉

데이터 테이블스페이스의 데이터 파일 이름을 변경한다.

자세한 내용은 ALTER TABLESPACE name RENAME DATAFILE 구문을 참조한다.

설명

ALTER TABLESPACE 구문은 다른 Data Definition Language (DDL)과 달리 ROLLBACK 할 수 없고 구문을 수행한 transaction이 자동으로 COMMIT 된다.

사용 예

각 세부 구문의 사용 예를 참조한다.

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- `CREATE TABLESPACE`
- `DROP TABLESPACE`

16.63 ALTER TABLESPACE name RENAME TO

기능

테이블스페이스의 이름을 변경한다.

구문

```
<rename tablespace statement> ::=  
    ALTER TABLESPACE tablespace_name RENAME TO <new_tablespace_name>  
    ;
```

사용 범위 및 접근 권한

<rename space statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

tablespace_name

기존 테이블스페이스의 이름이다.

- Built-in 테이블스페이스의 이름은 변경할 수 없다.
- OFFLINE 테이블스페이스의 이름은 변경할 수 없다.

new_tablespace_name

새로운 테이블스페이스의 이름이다.

설명

Tablespace 이름이 변경되더라도 기존에 이미 해당 tablespace에 생성된 table, index 등은 변경할 필요없다.

사용 예

다음은 테이블스페이스의 이름을 변경하는 예이다.

```
gSQL> ALTER TABLESPACE space1 RENAME TO space2;  
Tablespace altered.
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 **ALTER TABLESPACE**를 참조한다.

16.64 ALTER TABLESPACE name BACKUP

기능

테이블스페이스를 backup 하기 위해 backup이 가능한 상태와 불가능한 상태로 전환한다.

구문

```

<backup tablespace statement> ::=
    <tablespace begin backup statement>
  | <tablespace end backup statement>
  | <tablespace incremental backup statement>
  ;

<tablespace begin backup statement> ::=
    ALTER TABLESPACE tablespace_name BEGIN BACKUP [AT <domain_name>]
  ;

<tablespace end backup statement> ::=
    ALTER TABLESPACE tablespace_name END BACKUP [AT <domain_name>]
  ;

<tablespace incremental backup statement> ::=
    ALTER TABLESPACE tablespace_name
      BACKUP INCREMENTAL <incremental backup option> [AT <domain_name>]    ;

<incremental backup option> ::=
    LEVEL integer [ CUMULATIVE | DIFFERENTIAL ]
  
```

사용 범위 및 접근 권한

<backup space statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<tablespace begin backup statement>

테이블스페이스를 백업 가능한 상태로 설정한다.

- 생성되어 사용 중인 테이블스페이스를 백업 가능한 상태로 설정한다.
- OFFLINE/ temporary 테이블스페이스의 backup 상태는 전환할 수 없다.

tablespace_name

Backup 상태를 전환할 테이블스페이스의 이름이다.

<tablespace end backup statement>

테이블스페이스를 백업이 불가능한 상태로 설정한다.

<tablespace incremental backup statement>

테이블스페이스의 증분 백업을 수행한다.

데이터베이스가 OPEN 상태이고, ARCHIVELOG로 운영되어야 한다.

<incremental backup option>

- 'integer'는 0 ~ 4 까지 지정할 수 있다.
- 'LEVEL 0'는 CUMULATIVE나 DIFFERENTIAL을 지정할 수 없다.
- CUMULATIVE | DIFFERENTIAL
 - CUMULATIVE
 - 'integer'가 n이면 가장 최근에 'LEVEL 0' ~ 'LEVEL n-1'까지 백업한 이후 변경된 모든 페이지들을 백업한다.
 - DIFFERENTIAL
 - 'integer'가 n이면 가장 최근에 'LEVEL 0' ~ 'LEVEL n'까지 백업한 이후 변경된 모든 페이지들을 백업한다.
 - 생략되면 DIFFERENTIAL이 기본으로 지정된다.

<domain_name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

테이블스페이스에 생성된 datafile을 백업한다. 테이블스페이스 전체를 백업하려면 BEGIN BACKUP을 수행한 후 OS의 파일 복사로 datafile들을 복사하고 나서 END BACKUP을 수행한다. 한편, 증분 백업 파일은 하나의 구문으로 BACKUP_DIR_1 property에 설정된 경로에 생성한다.

사용 예

다음은 DICTIONARY_TBS 테이블스페이스의 전체 백업 상태를 'ACTIVE'로 설정하는 예이다.

```
ALTER TABLESPACE DICTIONARY_TBS BEGIN BACKUP;
```

다음은 DICTIONARY_TBS 테이블스페이스의 전체 백업 상태를 'INACTIVE'로 설정하는 예이다.

```
ALTER TABLESPACE DICTIONARY_TBS END BACKUP;
```

다음은 DICTIONARY_TBS 테이블스페이스의 LEVEL 0 증분 백업을 만드는 예이다.

```
ALTER TABLESPACE DICTIONARY_TBS BACKUP INCREMENTAL LEVEL 0;
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE
- ALTER TABLESPACE name [ONLINE|OFFLINE]

16.65 ALTER TABLESPACE name [ONLINE|OFFLINE]

기능

테이블스페이스 상태를 변경한다.

구문

```
<on/off tablespace statement> ::=
    ALTER TABLESPACE tablespace_name { ONLINE | OFFLINE [ NORMAL | IMMEDIATE ] }
    [ AT <domain name> ]
    ;
```

사용 범위 및 접근 권한

<on/off tablespace statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

ONLINE

OFFLINE 상태의 테이블스페이스를 ONLINE으로 변경한다.

OFFLINE NORMAL

ONLINE 상태의 테이블스페이스를 OFFLINE으로 변경한다.

OFFLINE으로 변경된 테이블스페이스는 일관된 (consistent) 상태이기 때문에 ONLINE 상태로 변경할 때 미디어 복구할 필요없다.

노트

MOUNT 단계에서는 OFFLINE NORMAL을 사용할 수 없다.

(단, 이전 인스턴스가 \SHUTDOWN NORMAL에 의해서 종료된 경우에는 가능하다.)

OFFLINE IMMEDIATE

ONLINE 상태의 테이블스페이스를 OFFLINE으로 변경한다.

OFFLINE으로 변경된 테이블스페이스는 비일관적인 (inconsistent) 상태이기 때문에, ONLINE 상태로 변경할 때 미디어 복구해야 한다.

노트

SYSTEM 테이블스페이스는 OFFLINE으로 변경할 수 없다.

OFFLINE IMMEDIATE는 미디어 복구를 필요로 하기 때문에 archive log mode에서만 수행할 수 있다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 테이블스페이스를 OFFLINE으로 설정하는 예이다.

```
gSQL> ALTER TABLESPACE space1 OFFLINE;
Tablespace altered.
```

다음은 MOUNT 단계에서 테이블스페이스에 OFFLINE NORMAL을 수행하여 실패하는 예이다.

```
gSQL> ALTER TABLESPACE space1 OFFLINE;
ERR-42000(16290): OFFLINE NORMAL is only allowed if the database is in OPEN phase :
ALTER TABLESPACE space1 OFFLINE
          *
ERROR at line 1:
gSQL> ALTER TABLESPACE space1 OFFLINE NORMAL;
ERR-42000(16290): OFFLINE NORMAL is only allowed if the database is in OPEN phase :
ALTER TABLESPACE space1 OFFLINE NORMAL
          *
ERROR at line 1:
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE
- ALTER TABLESPACE name BACKUP

16.66 ALTER TABLESPACE name ADD [DATAFILE|MEMORY]

기능

테이블스페이스의 공간을 확장한다.

구문

```

<add space statement> ::=
    ALTER TABLESPACE tablespace_name ADD <space specification>
        [AT <domain name>]
    ;
<space specification> ::=
    MEMORY <memory clause> [, ...]
    | DATAFILE <add datafile clause> [, ...]
<size clause> ::=
    integer [ K | M | G | T ]
<memory clause>
    'memory_name' { SIZE <size clause> }
<add datafile clause> ::=
    'filename'
        { SIZE <size clause> | REUSE | SIZE <size clause> REUSE }

```

사용 범위 및 접근 권한

<add space statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

tablespace_name

변경할 테이블스페이스의 이름이다.

<file specification>

테이블스페이스의 유형에 따라 다음과 같은 구문을 사용해야 한다.

- 메모리 데이터 테이블스페이스
 - DATAFILE <add datafile clause>
- 메모리 임시 테이블스페이스
 - MEMORY <memory clause>

<add datafile clause>

추가할 메모리 데이터 파일을 정의한다.

- 'filename'
 - 데이터를 저장 관리할 파일의 이름이다.
 - 메모리 데이터에 대한 체크포인트 이미지를 저장할 공간이다.
 - filename은 새로운 파일이거나 이미 존재하는 파일일 수 있다.
 - filename의 길이는 1024 바이트보다 작아야 한다.
- SIZE <size clause>
 - 새로운 파일일 경우 SIZE 절을 이용해 초기 크기를 지정한다.
 - 파일이 이미 존재할 경우 예러가 발생한다.
 - 파일의 크기는 최소 1 M ~ 최대 30 G 까지 지정할 수 있다.
- REUSE
 - 이미 존재하는 파일일 경우 REUSE 절을 이용한다.
 - 파일이 존재하지 않을 경우 새로운 파일을 생성한다.
 - 새로 생성되는 파일의 사이즈는
 - 데이터 테이블스페이스의 경우 MEMORY_DATA_TABLESPACE_SIZE 프로퍼티에 의해 결정되고
 - 임시 테이블스페이스의 경우 MEMORY_TEMP_TABLESPACE_SIZE 프로퍼티에 의해 결정된다.
- SIZE <size clause> REUSE
 - SIZE 절과 REUSE 절을 모두 명시할 경우 filename의 존재 여부에 따라 다음과 같이 작동한다.
 - 새로운 filename일 경우 SIZE 절을 이용해 초기 파일 크기를 지정한다.
 - 이미 존재하는 filename일 경우 기존 파일을 이용하여 SIZE 절의 값으로 크기를 조정한다.

〈memory clause〉

- 〈size clause〉
 - 추가할 메모리를 정의한다.

자세한 내용은 CREATE MEMORY TEMPORARY TABLESPACE 구문의 〈memory clause〉를 참조한다.

〈domain name〉

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 테이블스페이스에 data file을 추가하는 예이다.

```
gSQL> ALTER TABLESPACE space1 ADD DATAFILE 'test_file_a2.dbf' SIZE 10M REUSE;  
Tablespace altered.
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE MEMORY DATA TABLESPACE
- CREATE MEMORY TEMPORARY TABLESPACE
- ALTER TABLESPACE

16.67 ALTER TABLESPACE name DROP [DATAFILE|MEMORY]

기능

테이블스페이스의 공간을 축소한다.

구문

```
<drop space statement> ::=  
    ALTER TABLESPACE tablespace_name DROP <file specification>  
    [ AT <domain name> ]  
    ;  
<file specification> ::=  
    DATAFILE 'filename'  
    | MEMORY 'memory_name'
```

사용 범위 및 접근 권한

<drop space statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

tablespace_name

변경할 테이블스페이스의 이름이다.

<file specification>

테이블스페이스의 유형에 따라 다음과 같은 구문을 사용해야 한다.

- 메모리 데이터 테이블스페이스

- DATAFILE 'filename'
- 메모리 임시 테이블스페이스
 - MEMORY 'memory_name'

노트

오프라인 테이블스페이스의 파일은 삭제할 수 없다.
테이블스페이스의 첫 번째 파일은 삭제할 수 없다.
한 번이라도 사용된 적이 있는 데이터 파일은 삭제할 수 없다.

<domain name>

구문을 수행하는 멤버 및 그룹의 이름이다.
지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 테이블스페이스의 파일을 제거하는 예이다.

```
gSQL> ALTER TABLESPACE space1 DROP DATAFILE 'test_file_f2.dbf';
Tablespace altered.
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE
- ALTER TABLESPACE name ADD [DATAFILE|MEMORY]
- ALTER TABLESPACE name RENAME DATAFILE

16.68 ALTER TABLESPACE name RENAME DATAFILE

기능

테이블스페이스를 구성하는 데이터 파일의 이름을 변경한다.

구문

```
<rename datafile statement> ::=  
    ALTER TABLESPACE tablespace_name RENAME DATAFILE <filename_list> TO <filename_list>  
    ;  
<filename_list> ::=  
    'filename' [ AT <domain name> ] [, ...]
```

사용 범위 및 접근 권한

<rename datafile statement> 구문을 수행하려면 사용자에게 ALTER TABLESPACE ON DATABASE 권한이 있어야 한다.

노트

TDS 모드이면서 데이터베이스가 OPEN인 상태에서는 온라인 테이블스페이스 파일을 변경할 수 없다. (임시 메모리 테이블스페이스는 제외된다.)
변경한 후에도 파일은 반드시 존재해야 한다.

구문 규칙 및 파라미터

tablespace_name

변경할 테이블스페이스의 이름이다.

'filename'

메모리 임시 테이블스페이스는 'memory_name'을 의미하며, 그 외의 테이블스페이스 종류는 'filename'을 의미한다.

<domain name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

설명

테이블스페이스 상태에 따라 연산 가능 여부가 결정된다.

- OFFLINE: MOUNT 단계나 OPEN 단계에서 수행 가능하다.
- ONLINE: MOUNT 단계에서만 수행 가능하다.

사용 예

다음은 'test.dbf'를 'test1.dbf'로 변경하는 예이다.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'test.dbf' TO 'test1.dbf';  
Tablespace altered.
```

호환성

SQL 표준에서는 테이블스페이스 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- ALTER TABLESPACE
- ALTER TABLESPACE name ADD [DATAFILE|MEMORY]
- ALTER TABLESPACE name DROP [DATAFILE|MEMORY]

16.69 ALTER USER

기능

데이터베이스 사용자 정의를 변경한다.

구문

```

<alter user statement> ::=
    ALTER USER user_identifier <alter user action>
  | ALTER USER PUBLIC <alter schema path>
  ;

<alter user action> ::=
    <alter password>
  | <alter profile>
  | <password expire>
  | <account lock>
  | <alter default tablespace>
  | <alter temporary tablespace>
  | <alter index tablespace>
  | <alter schema path>

<alter password> ::=
    IDENTIFIED BY new_password [ REPLACE old_password ]

<alter profile> ::=
    PROFILE { profile_name | DEFAULT | NULL }

<password expire> ::=
    PASSWORD EXPIRE

<account lock> ::=
    ACCOUNT { LOCK | UNLOCK }

<alter default tablespace> ::=
    DEFAULT TABLESPACE tablespace_name

<alter temporary tablespace> ::=
    TEMPORARY TABLESPACE tablespace_name

<alter index tablespace> ::=
    INDEX TABLESPACE { tablespace_name | NULL }

<alter schema path> ::=
    SCHEMA PATH ( { schema_name | CURRENT PATH } [, ...] )

```

사용 범위 및 접근 권한

〈alter user statement〉 구문을 수행하려면 사용자에게 ALTER USER ON DATABASE 권한이 있어야 한다. 단, 〈alter password〉는 사용자가 user_identifier와 동일할 경우에 권한 없이 수행할 수 있다.

구문 규칙 및 파라미터

user_identifier

변경할 사용자의 이름이다.

〈alter password〉

사용자의 password를 변경한다.

- IDENTIFIED BY new_password
 - 새로운 password로 암호화되어 저장된다.
 - Password의 길이는 128 byte보다 작아야 한다.
 - Password는 대소문자를 구별한다.
- REPLACE old_password
 - ALTER USER ON DATABASE 권한이 있을 경우에는 생략 가능하다.
 - ALTER USER ON DATABASE 권한이 없을 경우에는 생략할 수 없다.
 - 사용자와 user_identifier가 동일해야 한다.

〈alter profile〉

비밀번호 관리 정책 profile을 변경한다.

- PROFILE profile_name
 - 사용자가 생성한 profile_name을 할당한다.
- PROFILE DEFAULT
 - 기본 profile인 "DEFAULT"를 할당한다.
- PROFILE NULL
 - Profile을 할당하지 않는다.

〈password expire〉

사용자의 비밀번호를 만료시킨다.

〈account lock〉

- ACCOUNT LOCK
 - 사용자 계정을 잠근다.
- ACCOUNT UNLOCK
 - 계정 잠금을 해제한다.

〈alter default tablespace〉

사용자의 기본 tablespace를 변경한다.

tablespace_name은 data tablespace여야 한다.

〈alter temporary tablespace〉

사용자의 temporary tablespace를 변경한다.

tablespace_name은 temporary tablespace여야 한다.

〈alter index tablespace〉

사용자의 index tablespace를 변경한다.

- INDEX TABLESPACE tablespace_name을 지정한다.
 - Data tablespace를 지정한 경우, LOGGING 인덱스여야 한다.
 - Temporary tablespace를 지정한 경우, NOLOGGING 인덱스여야 한다.
- INDEX TABLESPACE NULL
 - Index tablespace를 지정하지 않는다.

〈alter schema path〉

사용자의 스키마 접근 경로를 변경한다.

사용자의 SQL 구문에 schema가 명시되지 않았을 경우 스키마 접근 경로는 객체의 naming resolution을 위한 스키마 순서에 따라 결정된다.

스키마 이름이 기존에 스키마 접근 경로에 나열된 스키마 이름과 동일할 경우에는 추가적으로 반영되지 않는다.

다음은 `ALTER USER u1 SCHEMA PATH ( u1, s2, public );` 구문을 수행했을 때 schema에 존재하는 객체의 예이다.

스키마 이름	u1	s2	public
-	t1	-	t1
-	-	t2	-
-	-	-	t3

다음과 같이 u1 사용자가 수행하는 schema가 명시되지 않은 객체의 이름은 SCHEMA PATH에 의해 다음과 같이 해석된다.

- CREATE 구문
 - CREATE TABLE t1 (c1 INTEGER);
 - 예러: CREATE TABLE u1.t1 (c1 INTEGER);
 - CREATE TABLE t2 (c1 INTEGER);
 - 수행: CREATE TABLE u1.t2 (c1 INTEGER);
- SELECT 구문
 - SELECT * FROM t1;
 - 수행: SELECT * FROM u1.t1;
 - SELECT * FROM t2;
 - 수행: SELECT * FROM s2.t2;
 - SELECT * FROM t3;
 - 수행: SELECT * FROM public.t3;

CURRENT PATH

현재 사용자의 schema path이다.

다음 예와 같이 CURRENT PATH를 이용해 기존의 schema path를 유지하면서 새로운 schema path를 추가할 수 있다.

- u1의 현재 schema path
 - (u1, public)
- 구문 수행
 - ALTER USER u1 SCHEMA PATH (s1, CURRENT PATH, s2);
- u1의 schema path는 다음과 같이 변경된다.
 - (s1, u1, public, s2)

ALTER USER PUBLIC <alter schema path>

PUBLIC 계정의 schema path를 변경한다.

PUBLIC 계정의 schema path는 모든 사용자의 schema path에 포함된다.

PUBLIC 계정에 최초로 부여된 schema path는 다음과 같다.

- DICTIONARY_SCHEMA
- INFORMATION_SCHEMA
- DEFINITION_SCHEMA
- PERFORMANCE_VIEW_SCHEMA
- FIXED_TABLE_SCHEMA

설명

각 구문별 사용 규칙을 참조한다.

사용 예

다음은 사용자의 password를 변경하는 예이다.

```
gSQL> ALTER USER u1 IDENTIFIED BY new_password;  
User altered.
```

다음의 사용자에게 profile을 할당하는 예이다.

```
gSQL> ALTER USER u1 PROFILE prof1;  
User altered.  
gSQL> COMMIT;  
Commit complete.
```

다음은 사용자의 profile을 제거하는 예이다.

```
gSQL> ALTER USER u1 PROFILE NULL;  
User altered.  
gSQL> COMMIT;  
Commit complete.
```

다음은 사용자의 비밀번호를 만료시키는 예이다.

```
gSQL> ALTER USER u1 PASSWORD EXPIRE;  
User altered.  
gSQL> COMMIT;  
Commit complete.
```

다음은 사용자의 계정잠금을 해제하는 예이다.

```
gSQL> ALTER USER u1 ACCOUNT UNLOCK;  
User altered.  
gSQL> COMMIT;  
Commit complete.
```

다음은 사용자의 DEFAULT TABLESPACE를 변경하는 예이다.

```
gSQL> ALTER USER u1 DEFAULT TABLESPACE mem_data_tbs;  
User altered.
```

다음은 사용자의 TEMPORARY TABLESPACE를 변경하는 예이다.

```
gSQL> ALTER USER u1 TEMPORARY TABLESPACE mem_temp_tbs;  
User altered.
```

다음은 사용자의 INDEX TABLESPACE를 변경하는 예이다.

```
gSQL> ALTER USER u1 INDEX TABLESPACE mem_temp_tbs;  
User altered.
```

다음은 사용자의 schema path를 변경하는 예이다.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s1, CURRENT PATH );  
User altered.
```

호환성

SQL 표준에서 user의 개념은 다루고 있지만 user의 생성, 변경 및 삭제와 관련된 SQL 구문은 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE USER
- DROP USER

16.70 ALTER VIEW

기능

View 정의를 변경한다.

구문

```
<alter view statement> ::=  
    ALTER VIEW view_name <alter view action>  
    ;  
<alter view action> ::=  
    COMPILE
```

사용 범위 및 접근 권한

<alter view statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 view에 대해 (ALTER 또는 CONTROL TABLE) ON TABLE
- View가 속한 스키마에 대해 (ALTER TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY TABLE ON DATABASE

구문 규칙 및 파라미터

view_name

변경할 view의 이름이다.

schema_name.view_name과 같이 view가 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

COMPILE

View를 다시 컴파일한다.

View column에 부여한 COMMENT는 초기화된다.

설명

View가 참조하는 테이블이나 view가 변경되거나 삭제되면 해당 view도 영향을 받는다.

이런 정보는 INFORMATION_SCHEMA.VIEWS를 통해 조회할 수 있다.

- IS_COMPILED column
 - TRUE: View가 정상적으로 생성되었다.
 - FALSE: 에러가 존재하는 상태에서 FORCE 옵션으로 view가 생성되었다.
- IS_AFFECTED column
 - TRUE : View가 참조하는 테이블 또는 view가 변경되었다.
 - FALSE: View가 생성되고 COMPILE 된 후에 view가 참조하는 테이블이나 view가 변경되지 않았다.

사용 예

다음은 view가 참조하는 table 변경에 영향을 받은 view를 COMPILE 하는 예이다.

```
gSQL> SELECT TABLE_NAME, IS_AFFECTED
        FROM INFORMATION_SCHEMA.VIEWS
        WHERE TABLE_SCHEMA = 'PUBLIC'
          AND TABLE_NAME = 'V1';
TABLE_NAME IS_AFFECTED
-----
V1          TRUE
1 row selected.
gSQL> ALTER VIEW v1 COMPILE;
View altered.
COMMIT;
Commit complete.
gSQL> SELECT TABLE_NAME, IS_AFFECTED
        FROM INFORMATION_SCHEMA.VIEWS
        WHERE TABLE_SCHEMA = 'PUBLIC'
          AND TABLE_NAME = 'V1';
TABLE_NAME IS_AFFECTED
-----
```

```
V1          FALSE
1 row selected.
```

호환성

SQL 표준에서는 <alter view statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE VIEW
- DROP VIEW

16.71 ANALYZE SYSTEM

기능

시스템의 통계 정보를 제어한다.

구문

```
<analyze system statement> ::=
    ANALYZE SYSTEM [ <analyze action> ]
    ;
<analyze action> ::=
    COMPUTE STATISTICS
    | DELETE STATISTICS
```

사용 범위 및 접근 권한

<analyze system statement> 구문을 수행하려면 사용자에게 ANALYZE ANY ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

<analyze action>

생략할 경우 기본값은 COMPUTE STATISTICS 이다.

COMPUTE STATISTICS

시스템과 관련된 다음과 같은 통계 정보를 구축한다.

- CPU_OPS (Operations Per Second)
 - CPU가 초당 처리할 수 있는 operation의 개수이다.
- NETWORK_IOPS (IO operations Per Second)
 - Cluster인 경우에 유효하다.

- 초당 처리할 수 있는 network IO 횟수이다.

DELETE STATISTICS

시스템 통계 정보를 삭제한다.

설명

구축한 시스템 통계 정보는 질의 처리를 위한 최적화 과정의 비용을 계산하기 위해 사용한다.

사용 예

다음은 <analyze system statement> 구문을 사용하여 시스템 통계 정보를 구축하는 예이다.

```
gSQL> ANALYZE SYSTEM COMPUTE STATISTICS;  
analyzed.
```

다음은 구축된 시스템 통계 정보를 조회하는 예이다.

```
gSQL>  
SELECT * FROM DBA_STAT_SYSTEM;  
CPU_OPS NETWORK_IOPS NETWORK_BUFSIZE LAST_ANALYZED  
-----  
53000412          2914          65536 2017-03-30 16:49:42.200000  
1 row selected.
```

호환성

SQL 표준에서는 통계 정보에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **ANALYZE TABLE**을 참조한다.

16.72 ANALYZE TABLE

기능

테이블의 통계 정보를 제어한다.

구문

```

<analyze table statement> ::=
    ANALYZE TABLE table_name
    [ <parallel clause> ]
    [ <analyze action> ]
    ;

<parallel clause> ::=
    NOPARALLEL
    | PARALLEL [thread_count]

<analyze action> ::=
    COMPUTE STATISTICS [ <for_clause> ]
    | ESTIMATE STATISTICS <sample_clause> [ <for_clause> ]
    | DELETE STATISTICS

<sample_clause>
    SAMPLE row_count ROWS
    | SAMPLE percentage PERCENT

<for_clause>
    FOR ALL COLUMNS
    | FOR ALL INDEXED COLUMNS
    | FOR COLUMNS column_name [, ...]
    | FOR ALL INDEXES
    | FOR INDEXES index_name [, ...]
  
```

사용 범위 및 접근 권한

<analyze table statement> 구문을 수행하려면 사용자에게 ANALYZE ANY ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

table_name

테이블 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

<parallel clause>

분석 과정에서 사용할 thread 개수를 지정한다.

명시하지 않을 경우, 기본값은 PARALLEL 이다.

- NOPARALLEL
 - 병렬로 분석하지 않는다.
- PARALLEL [thread_count]
 - 병렬로 분석한다.
 - thread_count 값은 0 부터 사용할 수 있으며 최대값은 64 이다.
 - thread_count 값이 0이거나 생략된 경우 시스템의 CPU 개수에 의해 결정된다.

<analyze action>

생략할 경우 기본값은 COMPUTE STATISTICS 이다.

COMPUTE STATISTICS

전수 검사를 통해 테이블과 관련된 다음과 같은 통계 정보를 구축한다.

- Table 통계정보
 - Row count
- 각 column의 통계 정보
 - 서로 다른 값의 개수
 - NULL 값의 개수
 - 값의 평균 길이
 - 최소값
 - 최대값
- Index 통계정보
 - 서로 다른 key의 개수

Column의 data type에 따라 구축하는 통계 정보는 다음과 같다.

표 16-20 Data type에 따라 구축된 통계 정보

Data type	NUM_DISTINCT	NUM_NULLS	AVG_LENGTH	MIN/MAX
BOOLEAN	O	O	O	X
NATIVE_SMALLINT	O	O	O	O
NATIVE_INTEGER	O	O	O	O
NATIVE_BIGINT	O	O	O	O
NATIVE_REAL	O	O	O	O
NATIVE_DOUBLE	O	O	O	O
NUMBER	O	O	O	O
NUMERIC	O	O	O	O
FLOAT	O	O	O	O
CHAR(n)	O	O	O	64 bytes 이하인 경우에 구축된다.
VARCHAR(n)	O	O	O	64 bytes 이하인 경우에 구축된다.
LONG VARCHAR	X	X	X	X
BINARY	O	O	O	X
VARBINARY	O	O	O	X
LONG VARBINARY	X	X	X	X
DATE	O	O	O	O
TIME	O	O	O	O
TIMESTAMP	O	O	O	O
INTERVAL	O	O	O	O
ROWID	O	O	O	X

ESTIMATE STATISTICS <sample_clause>

지정한 <sample_clause>만큼의 샘플을 사용하여 column과 index의 통계 정보를 구축한다.

- SAMPLE row_count ROWS
 - 지정한 row 개수만큼 샘플을 사용한다.
 - row_count는 0보다 큰 양의 정수이다.
- SAMPLE percentage PERCENT
 - 지정한 비율만큼 샘플을 사용한다.
 - Percentage는 1 ~ 99 범위의 양의 정수이다.

샘플링 row의 개수가 MIN_SAMPLE_ROW_COUNT 프로퍼티 값보다 작을 경우 프로퍼티 값을 따른다.

〈for_clause〉

생략할 경우, 통계정보 구축이 가능한 모든 column과 모든 인덱스의 통계 정보를 구축한다.

FOR ALL COLUMNS

통계 정보 구축이 가능한 모든 column의 통계 정보를 구축한다.

인덱스 통계 정보는 구축하지 않는다.

FOR ALL INDEXED COLUMNS

인덱스에 포함된 모든 column의 통계 정보를 구축한다.

그 외 column의 통계 정보는 구축하지 않는다.

인덱스 통계 정보는 구축하지 않는다.

FOR COLUMNS column_name [, ...]

나열한 column의 통계 정보를 구축한다.

기술했지 않은 column의 통계 정보는 구축하지 않는다.

인덱스 통계 정보는 구축하지 않는다.

FOR ALL INDEXES

모든 인덱스의 통계 정보를 구축한다.

Column 통계 정보는 구축하지 않는다.

FOR INDEXES index_name [, ...]

나열한 인덱스의 통계 정보를 구축한다.

기술했지 않은 인덱스의 통계 정보는 구축하지 않는다.

Column 통계 정보는 구축하지 않는다.

DELETE STATISTICS

테이블의 통계 정보를 제거한다.

설명

테이블 통계 정보는 질의 최적화의 정확도에 영향을 미치는 매우 중요한 정보이다.

통계 정보 구축시간은 테이블의 데이터 양에 비례하므로, 데이터 양이 많을 경우 샘플링을 사용하여 통계 정보를 구축하거나 질의에 영향을 미치는 주요 정보에 대해서만 통계 정보를 구축하는 것이 바람직하다.

- 다음은 샘플링을 사용하여 통계 정보를 구축하는 예이다.

```
ANALYZE TABLE lineitem ESTIMATE STATISTICS SAMPLE 10 PERCENT;
```

- 다음은 주요 column과 인덱스에 대해서만 통계 정보를 구축하는 예이다.

```
ANALYZE TABLE lineitem COMPUTE STATISTICS FOR ALL INDEXED COLUMNS;
ANALYZE TABLE lineitem COMPUTE STATISTICS FOR ALL INDEXES;
```

사용 예

다음은 전수 검사를 통해 통계 정보를 구축하는 예이다.

```
gSQL> ANALYZE TABLE orders;
Table analyzed.
```

다음은 구축된 테이블 통계 정보를 조회하는 예이다.

```
gSQL>
SELECT
    TABLE_NAME
    , NUM_ROWS
FROM
    DICTIONARY_SCHEMA.USER_TABLES
WHERE
    TABLE_SCHEMA = 'PUBLIC'
    AND TABLE_NAME = 'ORDERS'
;
TABLE_NAME NUM_ROWS
-----
ORDERS      1500000
1 row selected.
gSQL>
```

```
SELECT
    TABLE_NAME
    , COLUMN_NAME
    , NUM_DISTINCT
    , NUM_NULLS
    , LOW_VALUE
    , HIGH_VALUE
FROM
    DICTIONARY_SCHEMA.USER_TAB_COLUMNS
WHERE
    TABLE_SCHEMA = 'PUBLIC'
    AND TABLE_NAME = 'ORDERS'
;
```

TABLE_NAME	COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	LOW_VALUE	HIGH_VALUE
ORDERS	O_ORDERKEY	1500000	0 1		6000000
ORDERS	O_CUSTKEY	99996	0 1		149999
ORDERS	O_ORDERSTATUS	3	0 F		P
ORDERS	O_TOTALPRICE	1464556	0 857.71		555285.16
ORDERS	O_ORDERDATE	2406	0 1992-01-01 00:00:00		1998-08-02 00:00:00
ORDERS	O_ORDERPRIORITY	5	0 1-URGENT		5-LOW
ORDERS	O_CLERK	1000	0 Clerk#0000000001		Clerk#0000001000
ORDERS	O_SHIPPRIORITY	1	0 0		0
ORDERS	O_COMMENT	1482071	0 null		null

9 rows selected.

gSQL>

```
SELECT
    TABLE_NAME
    , INDEX_NAME
    , DISTINCT_KEYS
FROM
    DICTIONARY_SCHEMA.USER_INDEXES
WHERE
    TABLE_SCHEMA = 'PUBLIC'
    AND TABLE_NAME = 'ORDERS'
;
```

TABLE_NAME	INDEX_NAME	DISTINCT_KEYS
ORDERS	ORDERS_PK_INDEX	1500000
ORDERS	ORDERS_CUSTKEY_FK	99996

2 rows selected.

호환성

SQL 표준에서는 통계 정보에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **ANALYZE SYSTEM**을 참조한다.

16.73 AUDIT POLICY

기능

Audit policy를 활성화한다.

구문

```

<audit policy statement> ::=
    AUDIT POLICY policy_name
    [ <specified_user_option> ]
    [ <specified_success_option> ]
    ;
<specified_user_option> ::=
    BY user_name [, ...]
    | EXCEPT user_name [, ...]
<specified_success_option> ::=
    WHENEVER SUCCESSFUL
    | WHENEVER NOT SUCCESSFUL

```

사용 범위 및 접근 권한

<audit policy statement> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

policy_name

활성화할 audit policy 객체의 이름이다.

활성화 된 audit policy는 기존 session에 영향을 미치지 않으며 새로 생성되는 session에만 영향을 준다.

<specified_user_option>

감사를 수행할 사용자를 명시한다.

생략할 경우 모든 사용자에게 대해 감사를 수행한다.

동일한 audit policy에 대해 BY 절과 EXCEPT 절을 함께 사용할 수 없다.

- BY user_list: 감사를 수행할 사용자를 특정할 경우 BY 절을 사용한다.
- EXCEPT user_list: 특정 사용자를 배제하고 다른 사용자들을 감사할 경우 EXCEPT 절을 사용한다.

<specified_success_option>

- WHENEVER SUCCESSFUL
 - Action이 성공했을 때 audit record가 생성된다.
- WHENEVER NOT SUCCESSFUL
 - Action이 실패했을 때 audit record가 생성된다.
- 생략할 경우 성공할 경우와 실패할 경우 모두 audit record를 생성한다.

설명

Audit policy를 활성화하면 기존 session에는 영향을 미치지 않으며 새로 생성되는 session에 대해 감사를 시작한다.

Audit Record 조회

감사 조건에 부합할 경우 audit record를 생성하는데 다음과 같이 DICTIONARY_SCHEMA.AUDIT_TRAIL view를 통해 조회할 수 있다.

```
SELECT logon_user
       , event_timestamp
       , action_name
       , object_name
       , sql_text
FROM   audit_trail
WHERE  policy_name = 'P1'
;
```

일반 사용자가 AUDIT_TRAIL을 조회하려면 SELECT 권한을 부여받아야 한다.

```
GRANT SELECT ON DICTIONARY_SCHEMA.AUDIT_TRAIL TO user_name;
```

Audit Policy 정보 조회

Audit policy 객체 정보는 `DICTIONARY_SCHEMA.AUDIT_POLICY_OPTIONS` view를 통해 조회할 수 있다.

```
SELECT policy_name
       , audit_option
       , object_schema
       , object_name
FROM audit_policy_options
;
```

Audit policy 객체들의 활성화 정보는 `DICTIONARY_SCHEMA.AUDIT_POLICY_ENABLED` view를 통해 조회할 수 있다.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
;
```

BY 절과 EXCEPT 절 사용 시 유의사항

동일한 audit policy에 대해 다수의 `AUDIT POLICY BY` 절을 사용할 경우, `user`들의 집합을 활성화한다. 즉, 다음 두 예는 동일한 의미이다.

- 예제 1: `u1`, `u2`에 대해 `p1` audit policy를 활성화한다.

```
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;
```

- 예제 2: `u1`, `u2`에 대해 `p1` audit policy를 활성화한다.

```
AUDIT POLICY p1 BY u1, u2;
```

동일한 audit policy에 대해 다수의 `AUDIT POLICY EXCEPT` 절을 사용할 경우, 마지막 `AUDIT POLICY` 구문만 유효하다.

즉, 다음 두 예는 다른 의미이다.

- 예제 1: 마지막 구문만 유효하며 `u2`를 배제하고 `p1` audit policy를 활성화한다.

```
AUDIT POLICY p1 EXCEPT u1;
```

```
AUDIT POLICY p1 EXCEPT u2;
```

- 예제 2: u1과 u2를 배제하고 p1 audit policy를 활성화한다.

```
AUDIT POLICY p1 EXCEPT u1, u2;
```

같은 audit policy에 대해 BY와 EXCEPT를 함께 사용할 수 없다.

- BY 절로 audit policy가 활성화된 경우 이후에 BY 절만 사용할 수 있다.

```
AUDIT POLICY p1 BY u1;
```

- Error

```
AUDIT POLICY p1 EXCEPT u2;
```

- EXCEPT 절로 audit policy가 활성화 된 경우 이후에 EXCEPT 절만 사용할 수 있다.

```
AUDIT POLICY p1 EXCEPT u1;
```

- Error: by all users에 해당한다.

```
AUDIT POLICY p1;
```

BY로 활성화된 audit policy를 EXCEPT로 전환하거나 EXCEPT로 활성화된 audit policy를 BY로 전환하려면 먼저 활성화된 audit policy를 비활성화 한 후에 전환해야 한다.

다음 예와 같이 NOAUDIT POLICY 구문으로 비활성화한다.

- AUDIT POLICY p1 BY u1, u2;
 - NOAUDIT POLICY p1 BY u1, u2;
- AUDIT POLICY p1;
 - NOAUDIT POLICY p1;
- AUDIT POLICY p1 EXCEPT u1, u2;
 - NOAUDIT POLICY p1;
 - NOAUDIT POLICY 구문에는 EXCEPT option이 없다.

BY 절과 함께 사용하는 WHENEVER 절은 누적된다.

다음 두 예는 동일한 의미이다.

- 예제 1: 성공/ 실패와 관계없이 audit record를 생성한다.

```
AUDIT POLICY p1 BY u1 WHENEVER SUCCESSFUL;
AUDIT POLICY p1 BY u1 WHENEVER NOT SUCCESSFUL;
```

- 예제 2: 성공/ 실패와 관계없이 audit record를 생성한다.

```
AUDIT POLICY p1 BY u1;
```

WHENEVER 절이 EXCEPT 절과 함께 사용되면 마지막 WHENEVER 절만 유효하다.

다음 두 예는 서로 다른 의미이다.

- 예제 1: 실패할 경우 audit record를 생성한다.

```
AUDIT POLICY p1 EXCEPT u1 WHENEVER SUCCESSFUL;
AUDIT POLICY p1 EXCEPT u1 WHENEVER NOT SUCCESSFUL;
```

- 예제 2: 성공/ 실패와 관계없이 audit record를 생성한다.

```
AUDIT POLICY p1 EXCEPT u1;
```

사용 예

다음은 모든 사용자에게 대해 audit policy를 활성화하는 예이다.

```
AUDIT POLICY table_pol;
```

다음 질의를 통해 활성화 정보를 확인할 수 있다.

```
SELECT policy_name
       , enabled_opt
       , user_name
FROM   audit_policy_enabled
WHERE  policy_name = 'TABLE_POL';
```

POLICY_NAME	ENABLED_OPT	USER_NAME
TABLE_POL	BY	ALL USERS

다음은 특정 사용자들을 한정하여 audit policy를 활성화하는 예이다.

```
AUDIT POLICY dml_pol BY u1, u2;
```


다음은 특정 사용자를 배제하여 audit policy를 활성화하는 예이다.

```
AUDIT POLICY read_seq_pol EXCEPT sys;
```

다음은 특정 사용자가 SQL 구문에 실패한 경우를 감사하는 예이다.

```
AUDIT POLICY delete_pol BY u1 WHENEVER NOT SUCCESSFUL;
```

호환성

SQL 표준에는 audit policy가 없다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - CREATE AUDIT POLICY
 - DROP AUDIT POLICY
 - ALTER AUDIT POLICY
- Audit policy 활성화/ 비활성화
 - AUDIT POLICY
 - NOAUDIT POLICY
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 제거: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.74 CLOSE cursor_name

기능

커서를 닫는다.

구문

```
<close statement> ::=  
    CLOSE cursor_name  
    ;
```

구문 규칙 및 파라미터

cursor_name

커서가 open 되어 있어야 한다.

세션 내에서 **DECLARE cursor_name** 구문으로 선언된 커서이어야 한다.

설명

Cursor는 session 내에 존재하는 객체이고 서로 다른 session의 cursor에 영향을 주지 않는다.

사용 예

다음은 interactive SQL tool (gsql)을 사용하여 커서를 DECLARE, OPEN, FETCH, CLOSE 하는 예이다.

```
gsQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;  
Cursor declared.  
gsQL> OPEN cur1;  
Cursor is open.  
gsQL> \var v_id    INTEGER
```

```
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
-----
      1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
-----
      2 data_2
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

호환성

표 16-21 SQL 표준 호환성

Feature ID	설명	지원 여부
B031	Basic dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- DECLARE cursor_name
- OPEN cursor_name
- FETCH cursor_name

16.75 COMMENT ON name IS

기능

객체에 대한 설명을 dictionary에 저장한다.

구문

```
<comment statement> ::=
    COMMENT ON <comment object> IS 'comment string'
    ;
<comment object> ::=
    CLUSTER GROUP group_name
    | CLUSTER MEMBER member_name
    | DATABASE
    | PROFILE profile_name
    | AUDIT POLICY policy_name
    | AUTHORIZATION user_name
    | TABLESPACE tablespace_name
    | SCHEMA schema_name
    | TABLE [schema_name].table_name
    | COLUMN [schema_name].table_name.column_name
    | INDEX [schema_name].index_name
    | SEQUENCE [schema_name].sequence_name
    | CONSTRAINT [schema_name].constraint_name
    | PROCEDURE [schema_name].procedure_name
```

사용 범위 및 접근 권한

<comment statement> 구문을 수행하려면 각 객체에 대하여 다음과 같이 권한을 변경해야 한다.

- CLUSTER GROUP
 - ALTER SYSTEM ON DATABASE
- CLUSTER MEMBER
 - ALTER SYSTEM ON DATABASE

- DATABASE
 - ALTER DATABASE ON DATABASE
- PROFILE
 - ALTER PROFILE ON DATABASE
- AUDIT POLICY
 - AUDIT SYSTEM ON DATABASE
- AUTHORIZATION (user)
 - ALTER USER ON DATABASE
- AUTHORIZATION (role)
 - ALTER ROLE ON DATABASE
- TABLESPACE
 - ALTER TABLESPACE ON DATABASE
- SCHEMA: 다음 권한 중 하나가 있어야 한다.
 - 스키마의 소유자
 - 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER SCHEMA ON DATABASE
- TABLE: 다음 권한 중 하나가 있어야 한다.
 - 테이블의 소유자
 - 테이블에 대해 CONTROL TABLE ON TABLE
 - 테이블이 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- COLUMN: 다음 권한 중 하나가 있어야 한다.
 - Column이 속한 테이블의 소유자
 - Column이 속한 테이블에 대해 CONTROL TABLE ON TABLE
 - Column이 속한 테이블의 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- INDEX: 다음 권한 중 하나가 있어야 한다.
 - 인덱스의 소유자
 - 인덱스가 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER ANY INDEX ON DATABASE
- SEQUENCE
 - 시퀀스의 소유자
 - 시퀀스가 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER ANY SEQUENCE ON DATABASE
- CONSTRAINT
 - 제약 조건의 소유자
 - 제약 조건이 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- PROCEDURE
 - Stored procedure/ function의 소유자
 - Stored procedure/ function이 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA

- ALTER ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

<comment object>

설명을 저장할 대상 객체로써 다음과 같은 database 객체에 대한 comment를 저장할 수 있다.

- Cluster object
 - CLUSTER GROUP
 - CLUSTER MEMBER
- Non-schema object
 - DATABASE
 - PROFILE
 - AUDIT POLICY
 - AUTHORIZATION (User or Role)
 - TABLESPACE
 - SCHEMA
- Schema object
 - TABLE 또는 VIEW
 - COLUMN
 - INDEX
 - SEQUENCE
 - CONSTRAINT
 - PROCEDURE 또는 FUNCTION

Schema object의 경우 schema_name을 기술하지 않으면 구문을 수행하는 사용자의 **Schema Path**에 의해 스키마 이름이 결정된다.

```
COMMENT ON TABLE test_table IS 'test comment';
→ COMMENT ON TABLE user_default_schema.test_table IS 'test comment';
```

'comment string'

저장할 comment 문장을 기술한다.

Comment를 삭제하려면 다음과 같이 empty string ('')을 사용한다.

```
COMMENT ON TABLE test_table IS '';
```

Comment string의 길이는 1024 bytes를 초과할 수 없다.

설명

다음 dictionary view의 COMMENTS column으로부터 객체 유형별 정보를 확인할 수 있다.

- Cluster objects
 - CLUSTER GROUP & CLUSTER MEMBER
 - DICTIONARY_SCHEMA.DBA_CLUSTER_COMMENTS view
- Non-schema objects
 - DATABASE
 - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
 - PROFILE
 - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
 - AUDIT POLICY
 - DICTIONARY_SCHEMA.AUDIT_POLICIES view
 - USER
 - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
 - TABLESPACE
 - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
 - SCHEMA
 - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
- SQL schema objects
 - TABLE
 - DICTIONARY_SCHEMA.USER_TAB_COMMENTS view
 - DICTIONARY_SCHEMA.ALL_TAB_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_TAB_COMMENTS view
 - VIEW
 - DICTIONARY_SCHEMA.USER_TAB_COMMENTS view
 - DICTIONARY_SCHEMA.ALL_TAB_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_TAB_COMMENTS view
 - COLUMN
 - DICTIONARY_SCHEMA.USER_COL_COMMENTS view
 - DICTIONARY_SCHEMA.ALL_COL_COMMENTS view
 - DICTIONARY_SCHEMA.DBA_COL_COMMENTS view
 - INDEX
 - DICTIONARY_SCHEMA.USER_INDEXES view
 - DICTIONARY_SCHEMA.ALL_INDEXES view

- `DICTIONARY_SCHEMA.DBA_INDEXES` view
- `SEQUENCE`
 - `DICTIONARY_SCHEMA.USER_SEQUENCES` view
 - `DICTIONARY_SCHEMA.ALL_SEQUENCES` view
 - `DICTIONARY_SCHEMA.DBA_SEQUENCES` view
- `CONSTRAINT`
 - `DICTIONARY_SCHEMA.USER_CONSTRAINTS` view
 - `DICTIONARY_SCHEMA.ALL_CONSTRAINTS` view
 - `DICTIONARY_SCHEMA.DBA_CONSTRAINTS` view
- `PROCEDURE & FUNCTION`
 - `DICTIONARY_SCHEMA.USER_PROCEDURES` view
 - `DICTIONARY_SCHEMA.ALL_PROCEDURES` view
 - `DICTIONARY_SCHEMA.DBA_PROCEDURES` view

각 view에 대한 자세한 내용은 **DICTIONARY_SCHEMA**를 참조한다.

사용 예

다음은 테이블에 주석을 작성하는 예이다.

```
gSQL> COMMENT ON TABLE t1 IS 'test comment on table t1';
Comment created.
```

다음은 column에 주석을 작성하는 예이다.

```
gSQL> COMMENT ON COLUMN t1.id IS 'test comment on column t1.id';
Comment created.
```

다음은 스키마에 주석을 작성하는 예이다.

```
gSQL> COMMENT ON SCHEMA s1 IS 'test comment on schema s1';
Comment created.
```

호환성

SQL 표준에는 <comment statement>가 없다.

16.76 COMMIT

기능

현재 트랜잭션을 종료하고 변경된 모든 내용을 영속화한다.

구문

```

<commit statement> ::=
    COMMIT [ WORK ]
        [ [ <commit comment clause> ] [ <commit write clause> ] |
          [ <commit force clause> ] [ <commit comment clause> ] ]
    ;
<commit comment clause> ::=
    COMMENT 'comment_string'
<commit write clause> ::=
    WRITE [ WAIT | NOWAIT ]
<commit force clause> ::=
    FORCE 'xid_string'

```

구문 규칙 및 파라미터

WORK

동작에 영향을 미치지 않는 예약어이다.

<commit comment clause>

- COMMENT 'comment_string'
 - 트랜잭션을 commit 할 때 트랜잭션에 주석을 지정한다.

<commit write clause>

Commit 연산으로 생성된 redo log가 redo log file에 기록될 때까지 기다릴지 여부를 결정한다.

- WAIT
 - Commit 연산으로 생성된 redo log가 redo log file에 기록될 때까지 기다린 후 연산을 종료한다.
- NOWAIT
 - Commit 연산으로 생성된 redo log가 redo log buffer에 기록되면 연산을 종료한다.
- 지정되어 있지 않을 경우, 프로퍼티를 따른다.

〈commit force clause〉

분산 트랜잭션을 수동으로 commit 할 때 사용한다.

- FORCE 'xid_string'
 - 'xid_string'에 해당하는 분산 트랜잭션을 commit 한다.
 - 'xid_string'은 'format_id.transaction_id.branch_id'로 구성된다.

설명

COMMIT 구문은 트랜잭션 내에서 수행된 다음 구문들을 완료한다.

- Data Manipulation Language (DML) 구문
 - 데이터를 변경하는 INSERT, UPDATE, DELETE 등의 구문
- Data Definition Language (DDL) 구문
 - 객체의 구조 및 정의를 변경하는 CREATE, DROP, ALTER, TRUNCATE, GRANT, REVOKE 등의 구문

예외적으로, DDL 중에 OS 자원을 다루거나 DATA TYPE을 변경하는 다음 구문들은 자동으로 COMMIT 된다.

- CREATE TABLESPACE
- DROP TABLESPACE
- ALTER TABLESPACE
- ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE: 〈alter column data type clause〉

COMMIT을 수행하면 WITHOUT HOLD 옵션으로 열린 커서는 자동으로 닫힌다. 커서에 대한 자세한 내용은 다음의 커서 관련 구문을 참조한다.

- DECLARE cursor_name
- OPEN cursor_name

트랜잭션이 지연된 (DEFERRED) 제약 조건을 위반하면 COMMIT 구문의 수행은 실패하고 트랜잭션은 ROLLBACK 된다. 지연된 제약 조건에 대한 자세한 내용은 SET CONSTRAINTS 구문의 설명을 참조한다.

사용 예

다음은 INSERT 구문을 수행한 후에 COMMIT을 수행하는 예이다.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );  
1 row created.  
gSQL> COMMIT WORK COMMENT 'INSERT T1';  
Commit complete.
```

호환성

표 16-22 SQL 표준 호환성

Feature ID	설명	지원 여부
T261	Chained transactions	X

참조

관련 내용은 다음을 참조한다.

- ROLLBACK
- SAVEPOINT savepoint_specifier

16.77 CREATE AUDIT POLICY

기능

Audit policy 객체를 생성한다.

생성한 audit policy 객체를 활성화하려면 AUDIT POLICY 구문을 수행하여야 한다.

구문

```

<audit policy definition> ::=
    CREATE AUDIT POLICY policy_name
    { <privilege_audit_clause> | <action_audit_clause> | <privilege_audit_clause> <action_
audit_clause> }
    ;
<privilege_audit_clause> ::=
    PRIVILEGES <database_privilege> [, ...]
<action_audit_clause> ::=
    ACTIONS { <object_action_audit> | <system_action_audit> } [, ...]
<object_action_audit> ::=
    ALL ON [schema_name.]object_name
    | <object_action> ON [schema_name.]object_name
<system_action_audit> ::=
    ALL
    | DDL
    | <system_action>

```

사용 범위 및 접근 권한

<audit policy definition> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

policy_name

생성할 audit policy의 이름이다.

<privilege_audit_clause>

권한 감사는 database privilege를 이용해 SQL 구문을 성공적으로 수행한 경우를 감사한다.

특정 사용자가 database privilege를 이용해 SQL 구문을 수행하는 것을 감사할 수 있으며, database의 소유자인 SYS 사용자에게 대해서는 권한 감사 기록을 남기지 않는다.

다음은 u1 사용자에게 SELECT ANY TABLE 권한을 부여하고 audit policy를 활성화하는 예이다.

```
CREATE AUDIT POLICY p1
    PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;
```

사용자 u1이 다음과 같은 SQL 구문을 수행할 경우 권한 감사가 다르게 동작한다.

- SELECT * FROM u1.t1;
 - u1.t1 테이블의 소유자 권한으로 SQL 구문을 수행하여 audit record를 생성하지 않는다.
- SELECT * FROM u2.t1;
 - SELECT ANY TABLE 권한으로 SQL 구문을 수행하여 audit record를 생성한다.

권한 감사에 기술할 수 있는 <database_privilege>는 다음 질의로 조회할 수 있다.

```
SELECT PRIVILEGE_NAME FROM V$AUDITABLE_DB_PRIVILEGES;
```

<action_audit_clause>

특정 객체에 대한 action과 database 전체에 대한 action을 감사한다.

<object_action_audit>

ALL ON object_name

object_name에 해당하는 객체를 나열할 수 있는 모든 action을 의미한다.

각 객체 유형별로 감사할 수 있는 audit action은 다음 표와 같다.

표 16-23 객체별 audit action

Object type	Action
Table	ALTER, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE
View	ALTER, COMMENT, GRANT, SELECT
Sequence	ALTER, COMMENT, GRANT, SELECT
Stored function/ procedure	ALTER, COMMENT, EXECUTE, GRANT

<object_action> ON object_name

특정 object에 대한 개별 action들은 다음과 같이 ON 절을 명시하여 하나씩 나열한다.

```
CREATE AUDIT POLICY p1
    ACTIONS INSERT ON u1.t1
        , DELETE ON u1.t1
        , UPDATE ON u1.t1
;
```

EXECUTE action 유의 사항

Stored function이나 stored procedure의 EXECUTE action 성공, 실패 여부에 대한 감사는 실제 수행 시점의 수행 가능 여부만으로 판단한다.

- WHENEVER NOT SUCCESSFUL의 경우, stored function/ procedure를 수행할 수 없을 경우에 감사 레코드를 생성한다.
- WHENEVER SUCCESSFUL의 경우, stored function/ procedure 내부의 SQL 구문을 수행하는 중에 에러가 발생하더라도 감사 레코드를 생성한다.
- Stored function/ procedure 내부의 SQL 구문 실패에 대한 감사가 필요할 경우, 해당 SQL 구문을 감사 대상에 포함해야 한다.

<system_action_audit>

특정 객체와 관계없이 database에 발생하는 system action을 감사한다.

- <system_action>

유효한 system action은 다음 질의로 조회할 수 있다.

```
SELECT ACTION_NAME FROM V$AUDITABLE_SYSTEM_ACTIONS;
```

- ALL

모든 system action을 의미한다.

- DDL

모든 Data Definition Language (DDL) 구문을 의미한다.

설명

Audit policy 객체는 감사할 대상들을 정의한 객체이다.

Audit policy를 활성화하려면 AUDIT POLICY 구문을 수행해야 한다.

다수의 audit policy를 정의하고 활성화할 수 있지만, 제한된 개수의 audit policy를 유지하는 것이 바람직하다.

여러 개의 작은 policy 조각들을 묶어 소수의 policy group으로 만드는 것이 바람직하다.

생성한 audit policy 객체의 옵션 정보는 다음과 같이 AUDIT_POLICY_OPTIONS view를 통해 조회할 수 있다.

```
SELECT audit_option
       , audit_option_type
       , object_schema
       , object_name
FROM audit_policy_options
WHERE policy_name = 'P1'
;
```

AUDIT_OPTION	AUDIT_OPTION_TYPE	OBJECT_SCHEMA	OBJECT_NAME
DELETE	OBJECT ACTION	U1	T1
INSERT	OBJECT ACTION	U1	T1
UPDATE	OBJECT ACTION	U1	T1

Audit Record 생성

여러 감사 조건에 부합하는 action이 발생할 경우, 한 개 이상의 audit record를 생성한다.

다음과 같이 유사한 audit option을 나열한 경우 하나의 audit record를 생성한다.

- Audit policy 정의

```
CREATE AUDIT POLICY p1
  PRIVILEGES SELECT ANY TABLE
  ACTIONS SELECT;
AUDIT POLICY p1;
```

- Audit action 수행

```
SELECT * FROM other_user.t1;
```

다음과 같이 서로 다른 audit option을 나열한 경우 두 개의 audit record를 생성한다.

- Audit policy 정의

```
CREATE AUDIT POLICY p1
    ACTIONS SELECT ON u1.t1
        , SELECT ON u2.t2;
AUDIT POLICY p1;
```

- Audit action 수행

```
SELECT COUNT(*) FROM u1.t1 A, u2.t2 B WHERE A.id = B.id;
```

다음과 같이 동일한 action에 대해 여러 audit policy를 활성화한 경우, 두 개의 audit record를 생성한다.

- Audit policy 정의

```
CREATE AUDIT POLICY p1
    PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;
CREATE AUDIT POLICY p2
    ACTIONS SELECT;
AUDIT POLICY p2;
```

- Audit action 수행

```
SELECT * FROM other.t1;
```

사용 예

다음은 권한을 감사하는 audit policy를 정의하는 예이다.

```
CREATE AUDIT POLICY policy_table
    PRIVILEGES CREATE ANY TABLE
        , DROP ANY TABLE
;
```


다음은 객체에 대한 action을 감사하는 audit policy를 정의하는 예이다.

```
CREATE AUDIT POLICY policy_dml
    ACTIONS INSERT ON u1.t1
        , DELETE ON u1.t1
        , UPDATE ON u1.t1
        , ALL ON u1.t2
;
```

다음은 system action을 감사하는 audit policy를 정의하는 예이다.

```
CREATE AUDIT POLICY policy_drop
    ACTIONS DROP TABLE, TRUNCATE TABLE
;
```

다음은 위의 예를 모두 합친 audit policy를 정의한 예이다.

```
CREATE AUDIT POLICY policy_group
    PRIVILEGES CREATE ANY TABLE
        , DROP ANY TABLE
    ACTIONS INSERT ON u1.t1
        , DELETE ON u1.t1
        , UPDATE ON u1.t1
        , ALL ON u1.t2
        , DROP TABLE
        , TRUNCATE TABLE
;
```

호환성

SQL 표준에는 audit policy가 없다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - CREATE AUDIT POLICY
 - DROP AUDIT POLICY

- **ALTER AUDIT POLICY**
- Audit policy 활성화/ 비활성화
 - **AUDIT POLICY**
 - **NOAUDIT POLICY**
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 제거: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.78 CREATE CLUSTER GROUP

기능

Cluster system에 참여할 cluster group을 생성한다.

구문

```
<cluster group definition> ::=
    CREATE CLUSTER GROUP group_name
        <cluster member definition> [, ...]
    ;
<cluster member definition> ::=
    CLUSTER MEMBER member_name <connection attribute>
<connection attribute> ::
    HOST 'address' PORT port_no
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<cluster group definition> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

group_name

Cluster group의 이름이다.

동일한 cluster group, cluster member 이름이 존재하지 않아야 한다.

이름의 길이는 128 바이트보다 작아야 한다.

<cluster member definition>

Cluster group에 포함될 cluster member를 정의한다.

Cluster group은 cluster member를 최대 32 개까지 포함할 수 있다.

Cluster system에 최초로 생성하는 cluster group에는 cluster member를 한 개만 정의할 수 있고 자기 자신을 cluster member로 포함해야 한다.

member_name

Cluster member의 이름이다.

Cluster member 이름은 해당 member의 database를 생성할 때 정의한 member 이름과 동일해야 한다.

동일한 cluster group, cluster member 이름이 존재하지 않아야 한다.

이름의 길이는 128 바이트보다 작아야 한다.

Cluster member의 start-up 단계는 OPEN 단계여야 한다.

<connection attribute>

Cluster member간 통신을 위한 연결 정보를 정의한다.

<connection attribute>는 해당 member의 database를 생성할 때 정의한 HOST, PORT와 동일해야 한다.

HOST와 PORT 조합은 cluster system 내에서 유일해야 한다.

- HOST address는 ip v4 형식으로 사용한다.
- PORT port_no는 1024 ~ 49151 범위의 값이어야 한다.

설명

<cluster group definition> 구문은 table들의 shard를 재배치하지 않는다.

추가된 cluster group에 shard를 재배치하려면 다음 구문을 수행해야 한다.

- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

사용 예

다음은 두 개의 cluster member로 구성된 cluster group을 생성하는 예이다.

```
gSQL>
CREATE CLUSTER GROUP g1
    CLUSTER MEMBER g1n1 HOST '192.168.0.11' PORT 10110
;
Cluster Group created.
gSQL>
ALTER CLUSTER GROUP g1
    ADD CLUSTER MEMBER g1n2 HOST '192.168.0.12' PORT 10120
;
Cluster Group altered.
gSQL>
CREATE CLUSTER GROUP g2
    CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210,
    CLUSTER MEMBER g2n2 HOST '192.168.0.22' PORT 10220
;
Cluster Group created.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

16.79 CREATE CLUSTER LOCATION

기능

Cluster member의 접속 정보를 생성한다.

구문

```
<cluster location definition> ::=
    CREATE CLUSTER LOCATION member_name
    <cluster connection attribute>
    ;
<cluster connection attribute> ::
    HOST 'address' PORT port_no
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<cluster location definition> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

member_name

Cluster member의 이름이다.

등록된 cluster location 정보에 동일한 cluster member 이름이 존재하지 않아야 한다.

이름의 길이는 128 바이트보다 작아야 한다.

<cluster connection attribute>

Cluster member간 통신을 위한 연결 정보를 정의한다.

HOST와 PORT 조합은 cluster system 내에서 유일해야 한다.

- HOST address는 ip v4 형식으로 사용한다.
- PORT port_no는 1024 ~ 49151 범위의 값이어야 한다.

설명

기본적으로 cluster location 정보는 cluster group을 생성하거나 cluster member를 추가할 때 제공되는 접속 정보를 이용하여 자동으로 생성된다. 생성된 정보는 cluster member와 group을 삭제할 때 함께 삭제된다.

만약 cluster location의 접속 정보가 변경되면 cluster member를 삭제하거나 다시 생성할 필요없이 **ALTER CLUSTER LOCATION**을 이용하여 접속 정보를 변경할 수 있다.

사용 예

```
gSQL>
CREATE CLUSTER LOCATION g1n2
    HOST '192.168.0.12' PORT 10120,
;
Created
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 **DROP CLUSTER LOCATION**을 참조한다.

16.80 CREATE INDEX

기능

인덱스를 생성한다.

구문

```

<index definition> ::=
    CREATE [ UNIQUE ] INDEX index_name
        ON table_name ( <index column element> [, ...] )
        [ <index attributes> [...] ]
        [ TABLESPACE tablespace_name ]
    ;
<index column element> ::=
    column_name [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
<index attributes> ::=
    <physical attribute clause>
    | STORAGE ( <segment attr clause> [...] )
    | <logging clause>
    | <parallel clause>
<physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer
<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>
<size clause> ::=
    integer [ K | M | G | T ]
<logging clause> ::=
    LOGGING
    | NOLOGGING
<parallel clause> ::=
    NOPARALLEL

```


| PARALLEL [integer]

사용 범위 및 접근 권한

<index definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 인덱스를 생성할 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (INDEX 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 CONTROL SCHEMA ON SCHEMA
 - CREATE ANY INDEX ON DATABASE
- 인덱스가 생성될 스키마에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (CREATE INDEX 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY INDEX ON DATABASE
- 인덱스가 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
 - USAGE TABLESPACE ON DATABASE
- 인덱스의 소유자는 다음과 같이 결정된다.
 - 인덱스가 속한 스키마의 소유자
 - 인덱스가 속한 스키마가 PUBLIC인 경우, 구문을 수행한 사용자

노트

Cluster에서 unique index는 모든 sharding key를 포함해야 한다.

구문 규칙 및 파라미터

UNIQUE

인덱스를 구성하는 column들에 중복 값을 허용하지 않는다.

index_name

생성할 인덱스의 이름이며, 스키마 내에서 유일해야 한다.

스키마 이름을 생략할 경우, 참조하는 테이블이 속한 스키마에 인덱스가 생성된다.

인덱스 이름의 길이는 128 바이트보다 작아야 한다.

table_name

인덱스를 생성할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

column_name

인덱스 key로 사용할 column의 이름이다.

하나 이상의 column을 정의해야 하는데 최대 32 개의 column을 인덱스 key로 사용할 수 있다.

구현 내용에 따라 다음과 같은 제약이 발생할 수 있다.

- 인덱스에 포함되는 column의 데이터 타입이 LONG CHARACTER VARYING, LONG BINARY VARYING 일 경우 인덱스를 생성할 수 없다.
- Column들의 precision 합계가 page size의 1/2 보다 작아야 생성할 수 있다.

ASC | DESC

Column의 정렬 순서를 명시한다.

- ASC: 오름차순으로 정렬한다.
- DESC: 내림차순으로 정렬한다.
- 명시하지 않을 경우, 기본값은 ASC이다.

NULLS FIRST | NULLS LAST

NULL 값의 정렬 순서를 명시한다.

- NULLS FIRST: NULL이 아닌 값들보다 앞에 위치한다.
- NULLS LAS : NULL이 아닌 값들보다 뒤에 위치한다.
- 명시하지 않을 경우, 기본값은 NULLS LAST 이다.

<physical attribute clause>

인덱스의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내 key 삽입으로 인한 페이지 분할 빈도를 조절하기 위해 예약된 공간이다.

- 인덱스 bottom-up 빌드 시에만 적용된다.
- 0 에서 99 까지의 값을 사용할 수 있다.
- 생략할 경우, DEFAULT_INDEX_PCTFREE property에 설정된 값을 사용한다.
- INITRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 초기 트랜잭션의 개수이다.
 - 인덱스에 접근하는 사용자의 수가 적을 경우에는 INITRANS를 낮게 설정하고, 동시에 접근하는 사용자가 많을 경우에는 INITRANS를 높게 설정한다.
 - 필요한 경우 설정된 MAXTRANS까지 자동으로 늘어난다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 4 이다.
- MAXTRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 트랜잭션의 최대 개수이다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 8 이다.

<segment attr clause>

인덱스가 저장될 공간에 대한 정보를 기술한다.

- INITIAL integer
 - 정의
 - 인덱스를 생성할 때 초기에 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'INITIAL 100'은 실제 8192 bytes로 작동한다.)
 - 이 크기 (TABLESPACE의 EXTENT 크기에 align 된 크기)는 MINEXTENTS의 크기와 같거나 커야 하며, MAXEXTENTS의 크기와 같거나 작아야 한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- NEXT integer
 - 정의
 - 인덱스의 물리적 공간을 추가할 경우 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT크기가 8192 bytes일 경우 'NEXT 100'은 실제 8192 bytes로 작동한다.)
 - NEXT는 현재 인덱스가 사용할 수 있는 남은 공간의 크기 (MAXEXTENTS의 크기에서 현재 사용하고 있는 공간의 크기를 뺀 크기)에 따라 다음과 같이 작동한다.
 - 남은 공간의 크기가 0인 경우 더 이상 공간을 확장하지 못한다.
 - 남은 공간의 크기가 0보다 크고 NEXT보다 작은 경우 남은 공간의 크기만큼 할당한다.
 - 남은 공간의 크기가 NEXT 보다 클 경우 NEXT의 크기만큼 할당한다.

- 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
- 생략할 경우, 기본값은 인덱스가 속한 TABLESPACE의 EXTENT 하나 크기이다.
- MINSIZE integer
 - 정의
 - 인덱스에서 유지해야할 최소 공간의 크기이다.
 - 이 값은 MAXSIZE의 값과 같거나 작아야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - EXTENT 두 개 크기보다 작을 경우 EXTENT 두 개 크기로 결정된다.
 - 생략할 경우, 기본값은 EXTENT 두 개 크기이다.
- MAXSIZE integer
 - 정의
 - 인덱스에서 할당받을 수 있는 최대 공간의 크기이다.
 - 이 값은 MINSIZE의 값과 같거나 커야 한다.
 - 이 크기는 인덱스가 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 EXTENT 크기 * 2147483647 (INT32의 최대 양의 정수)이다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

LOGGING | NOLOGGING

인덱스의 리드 로깅 여부를 명시한다.

명시하지 않을 경우, 기본값은 NOLOGGING이다.

NOPARALLEL | PARALLEL [integer]

인덱스 구축과정에서 사용될 thread 개수를 지정한다.

- NOPARALLEL
 - 인덱스를 병렬로 구축하지 않는다.
- PARALLEL [integer]
 - 인덱스를 병렬로 구축한다.

- integer가 생략되거나 0으로 지정된 경우에는 프로퍼티 (INDEX_BUILD_PARALLEL_FACTOR)를 따른다.
- integer는 0 부터 사용할 수 있으며 최대값은 16이다.
- 만약 프로퍼티의 값이 0인 경우에는 시스템이 최적값을 결정한다.
- 명시하지 않을 경우, 기본값은 PARALLEL이다.

TABLESPACE tablespace_name

인덱스가 저장될 tablespace의 이름을 지정한다.

- tablespace_name을 지정할 경우
 - LOGGING 인덱스의 tablespace_name은 data tablespace여야 하며
 - NOLOGGING 인덱스의 tablespace_name은 temporary tablespace 또는 nologging tablespace여야 한다.
- TABLESPACE 절을 생략할 경우,
 - USER의 INDEX TABLESPACE tablespace_name을 지정한 경우
 - 정의한 테이블스페이스를 사용한다.
 - USER의 INDEX TABLESPACE가 NULL인 경우
 - LOGGING 인덱스는 사용자의 기본 데이터 테이블스페이스를 사용하고,
 - NOLOGGING 인덱스는 사용자의 기본 임시 테이블스페이스를 사용한다.

설명

LOGGING 인덱스와 NOLOGGING 인덱스에는 다음과 같은 trade-off가 있다.

- LOGGING 인덱스
 - 장점: 시스템을 구동할 때 로그를 이용해 인덱스가 자동으로 복구되므로 별도의 구축과정이 없다.
 - 단점: 관련 row를 변경할 때 인덱스 변경 내용을 로그로 기록하여 디스크 IO를 유발한다.
- NOLOGGING 인덱스
 - 장점: 관련 row를 변경할 때 인덱스 변경 내용에 대한 디스크 IO가 발생하지 않는다.
 - 단점: 시스템을 구동할 때 인덱스에 대한 로그 정보가 없어 자동으로 인덱스를 재구축한다.

사용 예

다음은 unique index를 생성하는 예이다.

```
gSQL> CREATE UNIQUE INDEX idx_t1_id ON t1( id );
Index created.
```

다음은 다수의 column에 대해 인덱스를 생성하는 예이다.

```
gSQL> CREATE INDEX idx_t1_id_name ON t1( id, name );
Index created.
```

다음은 인덱스 column의 정렬 순서를 지정하는 예이다.

```
gSQL> CREATE INDEX idx_t1_dept_id ON t1( dept_id DESC );
Index created.
```

다음은 인덱스 column의 NULL 값 정렬 순서를 지정하는 예이다.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name NULLS FIRST );
Index created.
```

다음은 인덱스가 저장될 공간에 대한 정보를 설정하는 예이다.

```
gSQL> CREATE INDEX idx_t1_id ON t1( id )
        STORAGE ( INITIAL 10M NEXT 1M MINSIZE 10M MAXSIZE 100M );
Index created.
```

다음은 인덱스에 대해 리두 로깅을 생성하도록 하는 예이다.

```
gSQL> CREATE INDEX idx_t1_id ON t1( id ) LOGGING;
Index created.
```

다음은 인덱스를 병렬로 생성하도록 하는 예이다.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name ) PARALLEL;
Index created.
```

다음은 인덱스를 생성할 때 테이블스페이스를 지정하는 예이다.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name ) NOLOGGING TABLESPACE mem_temp_tbs;
Index created.
```

호환성

SQL 표준에서는 인덱스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 **DROP INDEX**를 참조한다.

16.81 CREATE PROFILE

기능

Profile을 생성하는 구문으로써 password 관리 방법을 설정할 수 있다.

User에게 profile을 할당하면 profile에 정의된 방법으로 user의 password를 관리한다.

구문

```

<profile definition> ::=
    CREATE PROFILE profile_name LIMIT
    { <password_parameters>, ... }
    ;

<password parameters> ::=
    FAILED_LOGIN_ATTEMPTS { integer | UNLIMITED | DEFAULT }
    | PASSWORD_LOCK_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_LIFE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_GRACE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_REUSE_MAX { integer | UNLIMITED | DEFAULT }
    | PASSWORD_REUSE_TIME { password_parameter_number_interval | UNLIMITED | DEFAULT }
    | PASSWORD_VERIFY_FUNCTION { <verify_policy> | NULL | DEFAULT }

<verify_policy> ::=
    KISA_VERIFY_FUNCTION
    | ORA12C_VERIFY_FUNCTION
    | ORA12C_STRONG_VERIFY_FUNCTION
    | VERIFY_FUNCTION_11G
    | VERIFY_FUNCTION

<password_parameter_number_interval> ::=
    integer
    | integer / integer
  
```

사용 범위 및 접근 권한

<profile definition> 구문을 수행하려면 사용자에게 CREATE PROFILE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

profile_name

생성할 profile의 이름을 명시한다.

password_parameters

비밀번호 관리를 위한 parameter들을 설정한다.

- 다음과 같은 parameter를 설정할 수 있다.
 - FAILED_LOGIN_ATTEMPTS
 - PASSWORD_LOCK_TIME
 - PASSWORD_LIFE_TIME
 - PASSWORD_GRACE_TIME
 - PASSWORD_REUSE_MAX
 - PASSWORD_REUSE_TIME
 - PASSWORD_VERIFY_FUNCTION

생략한 parameter는 "DEFAULT" profile의 정책을 따른다.

FAILED_LOGIN_ATTEMPTS

연속적인 로그인 실패 가능 횟수를 설정한다.

명시된 횟수를 넘어서면 계정이 잠긴다.

- FAILED_LOGIN_ATTEMPTS integer
 - 값의 범위는 0보다 큰 양의 정수여야 한다.
- FAILED_LOGIN_ATTEMPTS UNLIMITED
 - 로그인 실패로 인해 계정이 잠기지 않는다.
- FAILED_LOGIN_ATTEMPTS DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

PASSWORD_LOCK_TIME

연속적인 login 실패 후 계정이 잠기는 기간 (day)을 설정한다.

- PASSWORD_LOCK_TIME constant_expression
 - 잠금이 지속되는 기간 (day)이다.
 - 기본 단위는 일 (day)이다.
 - 테스트하기 위해 시 (n/24), 분 (n/1440), 초 (n/86400)를 명시할 수 있다.

- 값의 범위는 1초 (1/86400) ~ 100000 일이다.
- PASSWORD_LOCK_TIME UNLIMITED
 - 계정이 잠길 경우 ALTER USER user_name ACCOUNT UNLOCK 구문을 수행하기 전까지 잠금이 해제되지 않는다.
- PASSWORD_LOCK_TIME DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

PASSWORD_LIFE_TIME

비밀번호의 유효 기간 (day)을 설정한다.

- PASSWORD_LIFE_TIME constant_expression
 - 비밀번호의 유효 기간 (day)이다.
 - 기본 단위는 일 (day)이다.
 - 테스트하기 위해 시 (n/24), 분 (n/1440), 초 (n/86400)를 명시할 수 있다.
 - 값의 범위는 1초 (1/86400) ~ 100000 일이다.
- PASSWORD_LIFE_TIME UNLIMITED
 - 비밀번호의 만료 기간이 없다.
- PASSWORD_LIFE_TIME DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

PASSWORD_GRACE_TIME

PASSWORD_LIFE_TIME 이후에 login 했을 때 비밀번호 만료를 유예하는 기간을 설정한다.

- PASSWORD_GRACE_TIME constant_expression
 - 비밀번호 만료 유예 기간 (day)이다.
 - 기본 단위는 일 (day)이다.
 - 테스트 하기 위해 시 (n/24), 분 (n/1440), 초 (n/86400)를 명시할 수 있다.
 - 값의 범위는 1초 (1/86400) ~ 100000 일이다.
- PASSWORD_GRACE_TIME UNLIMITED
 - 비밀번호 만료를 계속 유예한다.
- PASSWORD_GRACE_TIME DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

비밀번호 유효기간이 지난 후 처음으로 login 하려고 시도할 때부터 PASSWORD_GRACE_TIME이 시작되고, 이 기간동안 비밀번호를 변경하지 않으면 비밀번호가 만료된다.

PASSWORD_REUSE_MAX

이전 비밀번호를 재사용하려 할 때 재사용할 수 없는 최근 비밀번호의 개수를 설정한다.

PASSWORD_REUSE_MAX는 PASSWORD_REUSE_TIME과 함께 사용해야 한다.

- PASSWORD_REUSE_MAX integer
 - 값의 범위는 0보다 큰 양의 정수여야 한다.
- PASSWORD_REUSE_MAX UNLIMITED
 - PASSWORD_REUSE_TIME이 UNLIMITED인 경우, 이전 비밀번호 모두를 재사용할 수 있다.
 - PASSWORD_REUSE_TIME이 UNLIMITED가 아닌 경우, 이전 비밀번호 중 어떤 것도 재사용할 수 없다.
- PASSWORD_REUSE_MAX DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

PASSWORD_REUSE_TIME

이전 비밀번호를 재사용하려 할 때, 해당 비밀번호를 재사용할 수 없는 기간을 설정한다.

PASSWORD_REUSE_TIME은 PASSWORD_REUSE_MAX와 함께 사용해야 한다.

- PASSWORD_REUSE_TIME constant_expression
 - 해당 비밀번호를 재사용할 수 없는 기간 (day)이다.
 - 기본 단위는 일 (day)이다.
 - 테스트 하기 위해 시 (n/24), 분 (n/1440), 초 (n/86400)를 명시할 수 있다.
 - 값의 범위는 1초 (1/86400) ~ 100000 일이다.
- PASSWORD_REUSE_TIME UNLIMITED
 - PASSWORD_REUSE_MAX가 UNLIMITED인 경우, 이전 비밀번호 모두를 재사용할 수 있다.
 - PASSWORD_REUSE_MAX가 UNLIMITED가 아닌 경우, 이전 비밀번호 중 어떤 것도 재사용할 수 없다.
- PASSWORD_REUSE_TIME DEFAULT
 - "DEFAULT" profile의 정책을 따른다.

PASSWORD_VERIFY_FUNCTION

비밀번호 복잡도 검증 방법을 설정한다.

- PASSWORD_VERIFY_FUNCTION null
 - 비밀번호 복잡도를 검증하지 않는다.
- PASSWORD_VERIFY_FUNCTION DEFAULT
 - "DEFAULT" profile의 정책을 따른다.
- PASSWORD_VERIFY_FUNCTION <verify policy>
 - 다음과 같이 비밀번호 복잡도 검증 방법을 지정할 수 있다.
 - KISA_VERIFY_FUNCTION
 - ORA12C_VERIFY_FUNCTION

- ORA12C_STRONG_VERIFY_FUNCTION
- VERIFY_FUNCTION_11G
- VERIFY_FUNCTION

KISA_VERIFY_FUNCTION

Korea Internet & Security Agency (KISA)의 비밀번호 검증 방법이다.

- 8 글자 이상
- 1 개 이상의 문자
- 1 개 이상의 숫자
- 1 개 이상의 특수 문자

ORA12C_VERIFY_FUNCTION

Oracle의 ORA12C_VERIFY_FUNCTION 비밀번호 검증 방법이다.

- 8 글자 이상
- 1 개 이상의 문자
- 1 개 이상의 숫자
- database name을 포함하면 안된다.
- 사용자 이름 또는 거꾸로 된 사용자 이름을 포함하면 안된다.
- goldilocks를 포함하면 안된다.
- oracle을 포함하면 안된다.
- 다음과 같이 단순한 비밀번호는 사용할 수 없다.
 - welcome1, database1, account1, user1234, password1, oracle123, computer1, abcdefg1, change_on_install
- 이전 비밀번호와 적어도 3 글자는 달라야 한다.

ORA12C_STRONG_VERIFY_FUNCTION

Oracle의 ORA12C_STRONG_VERIFY_FUNCTION 비밀번호 검증 방법이다.

- 9 글자 이상
- 2 개 이상의 대문자
- 2 개 이상의 소문자
- 2 개 이상의 숫자
- 2 개 이상의 특수 문자
- 이전 비밀번호와 적어도 4 글자는 달라야 한다.

VERIFY_FUNCTION_11G

Oracle의 VERIFY_FUNCTION_11G 비밀번호 검증 방법이다.

- 8 글자 이상

- 1 개 이상의 문자
- 1 개 이상의 숫자
- 사용자 이름을 포함하면 안된다.
- 이전 비밀번호와 적어도 3 글자는 달라야 한다.

VERIFY_FUNCTION

Oracle의 VERIFY_FUNCTION 비밀번호 검증 방법이다.

- 사용자 이름과 같으면 안된다.
- 4 글자 이상
- 1 개 이상의 문자
- 1 개 이상의 숫자
- 1 개 이상의 특수문자
- 다음과 같이 단순한 비밀번호는 사용할 수 없다.
 - welcome, database, account, user, password, oracle, computer, abcd
- 이전 비밀번호와 적어도 3 글자는 달라야 한다.

설명

계정 잠금

계정 잠금에 영향을 주는 parameter는 다음과 같다.

- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME

예를 들어 다음과 같은 profile과 user를 생성할 경우

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30;
ALTER USER u1 PROFILE prof;
```

u1 사용자의 login 실패횟수가 네 번을 초과할 경우 30일 동안 계정이 잠긴다.

그리고 30일이 지나면 계정 잠금이 해제된다.

PASSWORD_LOCK_TIME이 UNLIMITED면, ALTER USER 구문을 사용하여 계정을 명시적으로 잠금 해제해주어야 한다.

```
ALTER USER user1 ACCOUNT UNLOCK;
```

비밀번호 만료

비밀번호 만료에 영향을 주는 parameter는 다음과 같다.

- PASSWORD_LIFE_TIME
- PASSWORD_GRACE_TIME

비밀번호는 다음과 같은 순서로 만료된다.

1. 비밀번호 설정

- 비밀번호가 변경된 순간부터 PASSWORD_LIFE_TIME만큼 경과된 기간이 비밀번호 만료 시점으로 설정된다.
- 비밀번호가 만료된 상태는 OPEN이며, 정상적으로 login 할 수 있다.

2. 만료 시점 이후에 login할 경우

- Login에는 성공하지만 비밀번호의 만료 상태가 EXPIRED (GRACE)가 되며 다음과 같은 warning이 발생한다.
 - ERR-28000(16310): The password will expire in n days
 - ERR-28000(16311): The password will expire soon
 - SQL 표준에서는 password expire 개념을 다루지 않고 있다.
 - 28000은 authentication warning 또는 error의 SQL 표준 상태코드이며, (16310, 16311)은 GOLDILOCKS error code이다.
- Login한 순간부터 PASSWORD_GRACE_TIME만큼 경과된 기간이 비밀번호의 만료 시점으로 재설정된다.

3. 유예기간 이후에 login할 경우

- 비밀번호 만료 상태가 EXPIRED 되어 login 할 수 없으며 다음과 같은 error가 발생한다.
 - ERR-28000(16312): The password has expired
 - SQL 표준에서는 password expire 개념을 다루지 않고 있다.
 - 28000은 authentication warning 또는 error의 SQL 표준 상태코드이며, (16312)는 GOLDILOCKS error code이다.
 - Program을 사용하여 password 재입력을 제어하려면 16312 값의 GOLDILOCKS internal error code를 사용해야 한다.

표 16-24 비밀번호 만료 상태 전이

단계	시점	Login 성공 여부	계정 상태
1	비밀번호 변경	Success	OPEN
2	PASSWORD_LIFE_TIME 경과	Success with warning	EXPIRED(GRACE)
3	PASSWORD_GRACE_TIME 경과	Error	EXPIRED

다음 예제를 참조한다.

```
CREATE PROFILE prof LIMIT
  PASSWORD_LIFE_TIME 90
  PASSWORD_GRACE_TIME 3;
ALTER USER u1 PROFILE prof;
```

위 예에서 사용자 u1은 90일이 지난 후 login에 성공하지만 3일 안에 비밀번호가 만료된다는 경고 메시지를 받는다.

3일 안에 비밀번호를 변경하지 않으면 비밀번호는 만료된다.

비밀번호가 만료되면, login 할 때 새로운 비밀번호를 입력하라는 메시지를 받고 계정 접근이 거부된다.

비밀번호 재사용 가능 여부

비밀번호 재사용 가능 여부에 영향을 주는 parameter는 다음과 같다.

- PASSWORD_REUSE_MAX
- PASSWORD_REUSE_TIME

두 parameter의 비밀번호 재사용 가능 여부는 다음 표와 같다.

표 16-25 비밀번호 재사용 가능 조건

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	재사용 가능 조건
value	value	PASSWORD_REUSE_TIME과 PASSWORD_REUSE_MAX 조건을 만족해야 한다.
value	UNLIMITED	항상 불가
UNLIMITED	value	항상 불가
UNLIMITED	UNLIMITED	항상 가능

다음과 같은 profile을 생성한 경우

```
CREATE PROFILE prof LIMIT
  PASSWORD_REUSE_MAX 5
  PASSWORD_REUSE_TIME 3;
```

최근 다섯 개 비밀번호와 최근 3일 이내에 변경한 비밀번호는 재사용할 수 없다.

사용자 u1의 비밀번호 변경 이력이 다음과 같을 경우 현재 비밀번호가 P#_000007이고, 현재 날짜가 2015-08-08 이면 기존 비밀번호의 재사용 가능 여부는 다음과 같다.

표 16-26 재사용 가능 여부 예

password	password_date	재사용 가능 여부
P#_000001	2015-08-01	가능
P#_000002	2015-08-02	가능
P#_000003	2015-08-03	REUSE_MAX 위반

password	password_date	재사용 가능 여부
P#_000004	2015-08-04	REUSE_MAX 위반
P#_000005	2015-08-05	REUSE_MAX, REUSE_TIME 위반
P#_000006	2015-08-06	REUSE_MAX, REUSE_TIME 위반
P#_000007	2015-08-07	REUSE_MAX, REUSE_TIME 위반

비밀번호 재사용 가능 여부를 검사하기 위해 누적된 비밀번호 변경 이력은 다음 구문을 사용하여 삭제할 수 있다.

```
ALTER DATABASE CLEAR PASSWORD HISTORY;
```

DEFAULT profile

Database를 생성할 때 다음과 같은 "DEFAULT" profile을 자동으로 생성한다. 생성하는 "DEFAULT" profile의 password parameter 정보는 다음과 같다.

표 16-27 DEFAULT profile의 구성

Parameter	Value
FAILED_LOGIN_ATTEMPTS	10
PASSWORD_LOCK_TIME	1
PASSWORD_LIFE_TIME	180
PASSWORD_GRACE_TIME	7
PASSWORD_REUSE_MAX	UNLIMITED
PASSWORD_REUSE_TIME	UNLIMITED
PASSWORD_VERIFY_FUNCTION	NULL

"DEFAULT" profile의 기본값들은 다음과 같은 특성을 갖는다.

- 계정 잠금
 - 10 (FAILED_LOGIN_ATTEMPTS) 번 연속으로 login에 실패할 경우 1일 (PASSWORD_LOCK_TIME) 동안 계정을 잠근다.
- 비밀번호 만료
 - 180일 (PASSWORD_LIFE_TIME)이 경과된 후에 7일 (PASSWORD_GRACE_TIME)의 유예 기간이 지나면 비밀번호가 만료된다.
- 비밀번호 재사용 가능 여부
 - 이전 비밀번호를 재사용할 수 있다.
- 비밀번호 복잡도 검사
 - 검사하지 않는다.

DEFAULT profile은 삭제할 수 없고 다음 구문으로 변경은 가능하다.

ALTER PROFILE DEFAULT LIMIT ...

사용 예

다음은 계정 잠금을 제어하는 profile을 생성하는 예이다. 세 번 연속 login에 실패할 경우 3 일동안 계정을 잠근다.

```
gSQL> CREATE PROFILE prof1 LIMIT
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 3;
Profile created.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 만료를 제어하는 profile을 생성하는 예이다. 비밀번호의 유효기간은 90 일이며 7 일간의 유예기간을 갖는다.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_LIFE_TIME 90
        PASSWORD_GRACE_TIME 7;
Profile created.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 재사용 여부를 제어하는 profile을 생성하는 예이다. 다음 예에서는 비밀번호를 변경할 때 이전 비밀번호를 검사하지 않는다.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_REUSE_MAX DEFAULT
        PASSWORD_REUSE_TIME DEFAULT;
Profile created.
gSQL> COMMIT;
Commit complete.
```

다음은 비밀번호 복잡도 검사를 제어하는 profile을 생성하는 예이다.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
Profile created.
gSQL> COMMIT;
Commit complete.
```

다음은 모든 parameter를 설정하여 profile을 생성하는 예이다.

```
gSQL> CREATE PROFILE prof1 LIMIT
      FAILED_LOGIN_ATTEMPTS 3
      PASSWORD_LOCK_TIME 3
      PASSWORD_LIFE_TIME 90
      PASSWORD_GRACE_TIME 7
      PASSWORD_REUSE_MAX DEFAULT
      PASSWORD_REUSE_TIME DEFAULT
      PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
```

Profile created.

```
gSQL> COMMIT;
```

Commit complete.

호환성

SQL 표준은 profile에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP PROFILE
- ALTER PROFILE
- CREATE USER
- ALTER USER
- ALTER DATABASE CLEAR PASSWORD HISTORY

16.82 CREATE SCHEMA

기능

스키마를 정의한다.

구문

```

<schema definition> ::=
    CREATE SCHEMA <schema name clause>
        [ <schema element> [...] ]
    ;
<schema name clause> ::=
    schema_name
    | AUTHORIZATION user_identifier
    | schema_name AUTHORIZATION user_identifier
<schema element> ::=
    <table definition>
    | <view definition>
    | <index definition>
    | <sequence generator definition>
    | <grant privilege statement>
    | <comment statement>

```

사용 범위 및 접근 권한

<schema definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 스키마를 생성하려면 CREATE SCHEMA ON DATABASE 권한이 있어야 한다.
- <schema element>가 존재할 경우, 각 <schema element> 구문을 수행하기 위한 권한이 있어야 한다.
자세한 내용은 다음 각 구문의 *사용 범위 및 접근 권한*을 참조한다.
 - CREATE TABLE
 - CREATE VIEW
 - CREATE INDEX
 - CREATE SEQUENCE

- **GRANT privileges TO**
 - **COMMENT ON name IS**
 - user_identifier에 해당하는 사용자는 생성한 스키마에 대해 다음과 같은 권한을 갖는다.
 - 생성한 schema_name 스키마의 소유자
 - <schema element> 절로 생성된 객체의 소유자
 - 생성한 스키마에 별도의 권한을 부여하지 않으므로 객체를 생성하려면 적절한 스키마 권한을 부여받아야 한다. 스키마 권한의 종류에 대한 내용은 GRANT privileges TO 구문의 <schema privilege>를 참조한다.
- 사용 예**는 CREATE USER 구문을 참조한다.

구문 규칙 및 파라미터

schema_name

생성할 스키마의 이름이다.

Database 내에 동일한 스키마 이름이 존재하지 않아야 한다.

스키마 이름의 길이는 128 바이트보다 작아야 한다.

AUTHORIZATION user_identifier

스키마 이름을 생략할 경우, user_identifier와 동일한 이름의 스키마를 생성한다.

AUTHORIZATION을 지정하지 않을 경우, 구문을 수행한 사용자의 user_identifier가 사용된다.

schema_name AUTHORIZATION user_identifier

생성할 스키마 이름과 스키마의 소유자를 지정한다.

소유자는 role이나 PUBLIC이 될 수 없다.

<schema element>

스키마를 생성할 때 스키마 내에 함께 생성할 객체를 정의한다.

schema_element는 나열된 순서대로 실행되며, comma (,) 없이 공백으로만 구분한다.

생성하는 스키마와 이름이 다른 스키마에는 객체를 정의할 수 없다.

- <grant privilege statement> 구문은 다음 <privilege>에 대해서만 기술할 수 있다.
 - <schema privilege>
 - <table privilege>
 - <sequence privilege>

- <comment statement> 구문은 다음 객체에 대해서만 기술할 수 있다.
 - SCHEMA schema_name
 - TABLE [schema_name].table_name
 - COLUMN [schema_name].table_name.column_name
 - INDEX [schema_name].index_name
 - SEQUENCE [schema_name].sequence_name
 - CONSTRAINT [schema_name].constraint_name

설명

스키마는 table, view, index, sequence, constraint와 같은 SQL schema 객체들을 논리적으로 분류하는 객체이다.

GOLDILOCKS에서 user와 schema의 관계는 1 : N이다. 즉, user가 소유한 schema가 존재하지 않거나 user가 다수의 schema를 소유할 수 있다.

SQL 표준에서는 user, schema, database와 같은 non-schema 객체들의 관계를 명확히 정의하고 있지 않으며, 각 DBMS들은 다음과 같이 non-schema 객체간의 관계를 상이하게 정의하고 있다.

노트

DBMS에서 user와 schema의 관계

- Oracle
 - User : schema = 1 : 1
- DB2
 - User : schema = 1 : N
- Postgres
 - User : schema = 1 : N
- MySQL

- Database : schema = 1 : 1
- User는 database (schema)의 하위 객체이다.

사용 예

다음은 schema를 생성하는 예이다.

```
gSQL> CREATE SCHEMA s1;
Schema created.
```

다음은 schema를 생성하고 schema의 소유자를 지정하는 예이다.

```
gSQL> CREATE SCHEMA s1 AUTHORIZATION test;
Schema created.
```

다음은 schema와 schema에 속한 객체들을 함께 생성하는 예이다.

```
gSQL> CREATE SCHEMA s1
      CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
      CREATE INDEX idx_t1_id ON t1 ( id )
      COMMENT ON TABLE t1 IS 'comment on s1.t1'
;
Schema created.
```

호환성

표 16-28 SQL 표준 호환성

Feature ID	설명	지원 여부
S071	SQL paths in function and type name resolution	X
F461	Named character sets	X
F171	Multiple schemas per user	O
T332	Extended roles	X

참조

관련 내용은 다음을 참조한다.

- DROP SCHEMA
- CREATE USER
- CREATE TABLE
- CREATE VIEW
- CREATE INDEX
- CREATE SEQUENCE
- GRANT privileges TO
- COMMENT ON name IS

16.83 CREATE SEQUENCE

기능

시퀀스를 생성한다.

구문

```

<sequence generator definition> ::=
    CREATE SEQUENCE [schema_name.] sequence_name
        [ <sequence generator option> [, ...] ]
    ;

<sequence generator option> ::=
    <sequence generator start with option>
    | <basic sequence generator option>

<sequence generator start with option> ::=
    START WITH integer

<basic sequence generator option> ::=
    <sequence generator increment by option>
    | <sequence generator maxvalue option>
    | <sequence generator minvalue option>
    | <sequence generator cycle option>
    | <sequence generator cache option>

<sequence generator increment by option> ::=
    INCREMENT BY integer

<sequence generator maxvalue option> ::=
    MAXVALUE integer
    | (NO MAXVALUE | NOMAXVALUE)

<sequence generator minvalue option> ::=
    MINVALUE integer
    | (NO MINVALUE | NOMINVALUE)

<sequence generator cycle option> ::=
    CYCLE
    | (NO CYCLE | NOCYCLE)

<sequence generator cache option> ::=
    CACHE integer
    | (NO CACHE | NOCACHE)

```


사용 범위 및 접근 권한

〈sequence generator definition〉 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 시퀀스가 속한 스키마에 대해 (CREATE SEQUENCE 또는 CONTROL SCHEMA) ON SCHEMA
- CREATE ANY SEQUENCE ON DATABASE

시퀀스의 소유자는 다음과 같이 결정된다.

- 시퀀스가 속한 스키마의 소유자
- 시퀀스가 속한 스키마가 PUBLIC 인 경우, 구문을 수행한 사용자

시퀀스 소유자는 USAGE ON SEQUENCE WITH GRANT OPTION 권한을 갖는다.

생성한 시퀀스를 사용하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 시퀀스에 대해 USAGE ON SEQUENCE
- 시퀀스가 속한 스키마에 대해 (USAGE SEQUENCE 또는 CONTROL SCHEMA) ON SCHEMA
- USAGE ANY SEQUENCE ON DATABASE

구문 규칙 및 파라미터

sequence_name

생성할 시퀀스의 이름이며 스키마 내에서 유일한 이름이어야 한다.

schema_name.sequence_name과 같이 시퀀스가 속할 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

시퀀스 이름의 길이는 128 바이트보다 작아야 한다.

〈sequence generator option〉

〈sequence generator option〉을 사용하지 않을 경우 다음 두 문장은 같은 의미를 갖는다.

- CREATE SEQUENCE test_seq;
- CREATE SEQUENCE test_seq START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE NO CYCLE CACHE 20;

〈sequence generator start with option〉

첫 번째로 생성할 시퀀스 번호를 정의한다.

오름차순인지 내림차순인지에 따라 다음과 같은 특징을 갖는다.

- 오름차순 시퀀스일 경우 (INCREMENT BY 양수)
 - 최소값보다 큰 시퀀스 값으로 시작하고자 할 경우에 사용한다.

- START WITH 절을 생략할 경우, 기본값은 최소값 (MINVALUE value)이 된다.
- 내림차순 시퀀스일 경우 (INCREMENT BY 음수)
 - 최대값보다 작은 시퀀스 값으로 시작하고자 할 경우에 사용한다.
 - START WITH 절을 생략할 경우, 기본값은 최대값 (MAXVALUE value)이 된다.

〈sequence generator increment by option〉

시퀀스 번호의 간격을 정의한다.

다음과 같은 제약 및 특징을 갖는다.

- 양수 또는 음수를 사용할 수 있으며, 0은 사용할 수 없다.
- 간격의 절대값은 MINVALUE와 MAXVALUE의 차이보다 작아야 한다.
- 양수일 경우 오름차순 시퀀스가 생성되며 음수일 경우 내림차순 시퀀스가 생성된다.
- INCREMENT BY 절을 생략할 경우, 기본값은 양수 1 이다.

〈sequence generator maxvalue option〉

시퀀스로 생성할 수 있는 최대값을 정의한다.

- MAXVALUE integer
 - 최대값의 범위는 64 bit 정수의 최소값 (−9,223,372,036,854,775,808)에서 64 bit 정수의 최대값(+9,223,372,036,854,775,807) 사이인데
 - START WITH의 값과 같거나 크고, MINVALUE 값보다 커야 한다.
- NO MAXVALUE | NOMAXVALUE
 - 최대값은 다음과 같이 정의한다.
 - 오름차순 시퀀스일 경우, 64 bit 정수의 최대값 (+9,223,372,036,854,775,807)이다.
 - 내림차순 시퀀스일 경우, -1 이다.
 - NO MAXVALUE (SQL 표준)와 NOMAXVALUE는 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.
- MAXVALUE 와 NO MAXVALUE를 명시하지 않을 경우, 기본값은 NO MAXVALUE 이다.

〈sequence generator minvalue option〉

시퀀스로 생성할 수 있는 최소값을 정의한다.

- MINVALUE integer
 - 최소값의 범위는 64 bit 정수의 최소값 (−9,223,372,036,854,775,808)에서 64bit 정수의 최대값(+9,223,372,036,854,775,807) 사이인데
 - START WITH의 값과 같거나 작고, MAXVALUE 값보다 작아야 한다.
- NO MINVALUE | NOMINVALUE
 - 최소값은 다음과 같이 정의한다.
 - 오름차순 시퀀스일 경우, 1 이다.

- 내림차순 시퀀스일 경우, 64 bit 정수의 최소값 ($-9,223,372,036,854,775,808$) 이다.
- NO MINVALUE (SQL 표준)와 NOMINVALUE는 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.
- MINVALUE와 NO MINVALUE를 명시하지 않을 경우, 기본값은 NO MINVALUE 이다.

〈sequence generator cycle option〉

시퀀스의 값이 최대값 또는 최소값이 되었을 때, 계속 값을 생성할지 여부를 명시한다.

- CYCLE
 - 오름차순 시퀀스가 최대값이 되었을 때, 최소값부터 다시 생성한다.
 - 내림차순 시퀀스가 최소값이 되었을 때, 최대값부터 다시 생성한다.
- NO CYCLE | NOCYCLE
 - 최대값 또는 최소값이 되었을 때, 시퀀스 값을 생성할 수 없다.
 - NO CYCLE (SQL 표준)과 NOCYCLE은 동일한 의미의 예약어이므로 어떤 것을 사용해도 무방하다.
- CYCLE과 NO CYCLE을 명시하지 않을 경우, 기본값은 NO CYCLE 이다.

〈sequence generator cache option〉

시퀀스에 빠르게 접근하기 위해 메모리상에 미리 적재할 시퀀스 값의 개수를 정의한다.

Database를 재구동할 때 메모리상에 적재한 시퀀스 값은 유실되며 적재한 이후의 값부터 시작된다.

- CACHE integer
 - CACHE 값은 2와 같거나 커야 하며,
 - CYCLE이 존재할 경우 CACHE 값은 CYCLE의 길이보다 크지 않아야 한다.
 - CYCLE의 길이: $\text{CEIL}(\text{MAXVALUE} - \text{MINVALUE}) / \text{ABS}(\text{INCREMENT})$
- NO CACHE | NOCACHE
 - 메모리 상에 시퀀스값을 미리 적재하지 않는다.
- CACHE/ NO CACHE를 명시하지 않을 경우, 기본값은 CACHE 20 이다.

설명

생성한 시퀀스 객체의 시퀀스 값은 **NEXTVAL** 함수와 **CURRVAL** 함수를 이용하여 사용할 수 있다.

시퀀스 값은 트랜잭션 속성을 가지지 않으며, 시퀀스 함수를 사용한 SQL 구문에서 에러가 발생하거나 명시적인 ROLLBACK을 수행하더라도 시퀀스 값은 가장 최신 값을 유지한다.

CURRVAL 함수의 경우, session에서 가장 최근에 호출한 NEXTVAL 값을 반환한다.

이러한 특성을 이용하면 NEXTVAL을 이용하여 한 번 얻은 시퀀스 값을 다른 SQL 문장에 계속 사용할 수 있다. 단, session에서 NEXTVAL을 호출하지 않은 경우에 CURRVAL를 사용하면 에러가 발생한다.

사용 예

다음과 같이 시퀀스 옵션을 정의하지 않은 seq1 객체는 seq2 객체와 동일한 의미의 오름차순 시퀀스이다.

```
gSQL> CREATE SEQUENCE seq1;
Sequence created.
gSQL> CREATE SEQUENCE seq2 START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE NO CYCLE CACHE
20;
Sequence created.
```

다음은 홀수값을 생성하는 시퀀스이다.

```
gSQL> CREATE SEQUENCE seq1 START WITH 1 INCREMENT BY 2;
Sequence created.
```

다음은 0 부터 시작하여 1000 까지 반복적으로 짝수를 생성하는 시퀀스를 생성하는 예이다.

```
gSQL> CREATE SEQUENCE seq1 START WITH 0 MINVALUE 0 MAXVALUE 1000 INCREMENT BY 2 CYCLE;
Sequence created.
```

다음은 -1 부터 시작하는 내림차순 시퀀스를 생성하는 예이다.

```
gSQL> CREATE SEQUENCE seq1 INCREMENT BY -1;
Sequence created.
```

호환성

SQL 표준에서는 <sequence generator cache option> 절을 정의하지 않고 있다.

표 16-29 SQL 표준 호환성

Feature ID	설명	지원 여부
T176	Sequence generator support	O

참조

관련 내용은 다음을 참조한다.

- DROP SEQUENCE

- ALTER SEQUENCE
- NEXTVAL
- CURRVAL

16.84 CREATE SYNONYM

기능

Synonym을 생성한다. Synonym은 테이블, view, 시퀀스, 또다른 synonym의 대체 이름으로써 이들 대신 다음 구문에서 사용될 수 있다.

- DML: SELECT, INSERT, UPDATE, DELETE, LOCK TABLE, CALL
- DDL: GRANT, REVOKE, COMMENT

구문

```
<table definition> ::=
    CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema_name.]synonym_name
    FOR [schema_name.]object_name
    ;
```

사용 범위 및 접근 권한

<synonym definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- PUBLIC을 명시하여 public synonym을 생성하려면 CREATE PUBLIC SYNONYM ON DATABASE 권한이 있어야 한다.
- Public synonym의 소유자는 PUBLIC이며, 생성한 사용자는 아무런 권한을 갖지 않는다.
- Private synonym을 생성하려면 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (CREATE SYNONYM 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY SYNONYM ON DATABASE
- Private synonym의 소유자는 다음과 같이 결정된다.
 - Private synonym이 속한 스키마의 소유자
 - Private synonym이 속한 스키마가 PUBLIC인 경우, 구문을 수행한 사용자
- Synonym을 생성했어도 기본 객체에 대한 권한이 없으면, 해당 synonym을 사용한 구문을 실행할 수 없다.
- 또한 synonym에 권한을 승인하면 synonym이 지칭하는 기본 객체에 대한 권한이 부여되므로 권한을 승인할 때

주의해야 한다.

구문 규칙 및 파라미터

[OR REPLACE]

이미 synonym이 존재할 경우, 기존의 synonym을 대체한다.

[PUBLIC]

Public synonym을 만들기 위해 명시한다.

이 절을 생략하면 private synonym이 생성된다.

synonym_name

생성할 synonym의 이름이며, 스키마 내에서 유일한 이름이어야 한다.

schema_name.synonym_name과 같이 synonym이 소속할 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

Synonym 이름의 길이는 128 바이트보다 작아야 한다.

Public synonym은 non-schema 객체이다. 따라서 PUBLIC을 명시하여 public synonym을 생성할 때는 스키마 이름을 명시할 수 없다.

object_name

schema_name.object_name과 같이 객체가 소속된 스키마를 명시할 수 있으며, schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

object_name을 명시할 수 있는 객체 타입은 다음과 같다.

- Table
- View
- Sequence
- 또 다른 synonym

대상 객체의 존재 여부, cycle check, 권한 검사 등은 synonym을 사용한 구문을 수행할 때 실행된다.

설명

Synonym은 테이블, view, 시퀀스, 다른 synonym의 대체 이름이다.

Synonym을 생성해서 사용하면 기본 객체가 변경되더라도 응용 프로그램 수정 없이 synonym만 재정의 해서 사용하면 되기 때문에 매우 편리하다. 또한 객체의 실제 이름과 스키마를 숨김처리해서 데이터베이스 보안을 개선할 수도 있고, 객체의 긴 이름을 사용하기 쉬운 짧은 이름으로 변경하여 사용성을 높일 수도 있다.

Synonym은 말 그대로 대체 이름이기 때문에, 이를 생성했다고 해서 synonym을 이용하여 해당 객체에 접근할 수는 없다. 해당 객체에 대한 적절한 권한이 있어야만 접근할 수 있다.

Synonym을 사용하여 구문을 수행할 때 객체는 다음과 같은 순서로 접근한다.

1. 해당 이름의 테이블을 찾는다.
2. 테이블이 없을 경우, 해당 이름의 private synonym을 찾는다.
3. Private synonym이 없을 경우, 해당 이름의 public synonym을 찾는다.

```
gSQL> CREATE PUBLIC SYNONYM syn1 FOR u1.t1;
Synonym created.
gSQL> CREATE PUBLIC SYNONYM syn2 FOR syn1;
Synonym created.
gSQL> SELECT * FROM syn2;
```

위 SELECT 구문 예제에서 객체 접근 순서는 다음과 같다.

1. syn2 테이블을 검색하였으나 해당 테이블이 없다.
2. syn2 private synonym을 검색하였으나 해당 synonym이 없다.
3. syn2 public synonym을 검색하여 해당 synonym을 찾았다.
 - 가. syn1 테이블을 검색하였으나 해당 테이블이 없다.
 - 나. syn1 private synonym을 검색하였으나 해당 synonym이 없다.
 - 다. syn1 public synonym을 검색하여 해당 synonym을 찾았다.
 - ㄱ. u1.t1 테이블을 검색하여 찾았다.

사용 예

다음은 private synonym을 생성하는 예이다.

```
gSQL> CREATE SYNONYM MyEmp FOR branch.Employee;
Synonym created.
gSQL> SELECT * FROM MyEmp;
```


다음은 public synonym을 생성하는 예이다.

```
gSQL> CREATE PUBLIC SYNONYM MainEmp FOR main.Employee;  
Synonym created.  
gSQL> SELECT * FROM MainEmp;
```

호환성

SQL 표준에서는 CREATE SYNONYM 구문을 정의하지 않고 있다.

참조

관련 내용은 DROP SYNONYM을 참조한다.

16.85 CREATE TABLE

기능

테이블을 정의한다.

구문

```

<table definition> ::=
    CREATE TABLE table_name
        ( <table element> [, ...] )
        [ <table sharding strategy> ]
        [ <table attribute clause> [...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
    ;
<table element> ::=
    <column definition>
    | <table constraint definition>
<column definition> ::=
    column_name <data type>
        [ <default clause> | <identity column specification> ]
        [ <column constraint definition> ]
<data type> ::=
    <character string type>
    | <binary string type>
    | <numeric type>
    | <boolean type>
    | <datetime type>
    | <interval type>
<character string type> ::=
    CHARACTER [ ( integer [ <character length units> ] ) ]
    | CHAR [ ( integer [ <character length units> ] ) ]
    | CHARACTER VARYING ( integer [ <character length units> ] )
    | CHAR VARYING ( integer [ <character length units> ] )
    | VARCHAR ( integer [ <character length units> ] )
    | CHARACTER LONG VARYING

```

| LONG VARCHAR

<character length units> ::=

CHARACTERS

| CHAR

| OCTETS

| BYTE

<binary string type> ::=

BINARY [(length)]

| BINARY VARYING (length)

| VARBINARY (length)

| LONG BINARY VARYING

| LONG VARBINARY

<numeric type> ::=

<exact numeric type>

| <approximate numeric type>

| <native numeric type>

<exact numeric type> ::=

NUMERIC [(precision [, scale])]

| SMALLINT

| INTEGER

| INT

| BIGINT

<approximate numeric type> ::=

| FLOAT [(precision)]

| REAL

| DOUBLE PRECISION

<native numeric type> ::=

NATIVE_SMALLINT

| NATIVE_INTEGER

| NATIVE_BIGINT

| NATIVE_REAL

| NATIVE_DOUBLE

<boolean type> ::=

BOOLEAN

<datetime type> ::=

DATE

| TIME [(time_precision)] [WITH TIME ZONE | WITHOUT TIME ZONE]

| TIMESTAMP [(timestamp_precision)] [WITH TIME ZONE | WITHOUT TIME ZONE]

<interval type> ::=

INTERVAL <interval qualifier>

```

<interval qualifier> ::=
    <non-second primary datetime field> [ ( interval_leading_field_precision ) ]
        TO { <non-second primary datetime field> | SECOND [ ( interval_fractional_seconds_
precision ) ] }
    | <non-second primary datetime field> [ ( interval_leading_field_precision ) ]
    | SECOND [ ( interval_leading_field_precision [, interval_fractional_seconds_precision ] )
]
<non-second primary datetime field> ::=
    YEAR
    | MONTH
    | DAY
    | HOUR
    | MINUTE
<default clause> ::=
    DEFAULT <default option>
<default option> ::=
    constant
    | NULL
    | expression
<identity column specification> ::=
    GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY
    [ ( <common sequence generator option> [, ...] ) ]
<common sequence generator option> ::=
    START WITH integer_constant
    | <basic sequence generator option>
<basic sequence generator option> ::=
    INCREMENT BY integer_constant
    | { MAXVALUE integer_constant | NO MAXVALUE }
    | { MINVALUE integer_constant | NO MINVALUE }
    | { CYCLE | NO CYCLE }
    | { CACHE integer_constant | NO CACHE }
<column constraint definition> ::=
    [ CONSTRAINT constraint_name ] <column constraint> [ <constraint characteristics> ]
<column constraint> ::=
    NOT NULL
    | { UNIQUE | PRIMARY KEY } [ <index name clause> [ <index attributes> ] [ TABLESPACE index_
tablename ] ]
<index name clause> ::=
    INDEX index_name
<index attributes> ::=
    <index physical attribute clause>

```

```

    | STORAGE ( <segment attr clause> [...] )
    | <logging clause>
<table constraint definition> ::=
    [ CONSTRAINT constraint_name ] <table constraint> [ <constraint characteristics> ]
<table constraint> ::=
    <unique constraint definition> [ <index name clause> [ <index attributes> ] [ TABLESPACE
index_tablespace_name ] ]
<unique constraint definition> ::=
    { UNIQUE | PRIMARY KEY } ( <key column element> [, ...] )
<key column element> ::=
    column_name [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
<table sharding strategy> ::=
    <cloned strategy>
    | <hash sharding strategy>
    | <range sharding strategy>
    | <list sharding strategy>
<cloned strategy> ::=
    CLONED [ <clone placement> ]
<clone placement> ::=
    AT CLUSTER WIDE
    | AT CLUSTER GROUP group_list
<hash sharding strategy> ::=
    SHARDING BY [HASH] ( column_list )
    [ <hash shard count> ]
    [ <hash shard placement> ]
<hash shard count> ::=
    SHARD COUNT integer
<hash shard placement> ::=
    AT CLUSTER WIDE
    | AT CLUSTER GROUP group_list
<range sharding strategy> ::=
    SHARDING BY RANGE ( column_list )
    { <cluster-wide range shard placement> | <group-specific range shard placement> }
<cluster-wide range shard placement> ::=
    AT CLUSTER WIDE
    <range shard definition> [, ...]
<group-specific range shard placement> ::=
    <group-specific range shard definition> [, ...]
<group-specific range shard definition> ::=
    <range shard definition> AT CLUSTER GROUP group_name
<range shard definition> ::=

```

```

    SHARD range_name VALUES LESS THAN ( <range value clause> )
<range value clause> ::=
    <range value> [, ...]
<range value> ::=
    constant
    | MAXVALUE
<list sharding strategy> ::=
    SHARDING BY LIST ( column_name )
    { <cluster-wide list shard placement> | <group-specific list shard placement> }
<cluster-wide list shard placement> ::=
    AT CLUSTER WIDE
    <list shard definition> [, ...]
<group-specific list shard placement> ::=
    <group-specific list shard definition> [, ...]
<group-specific list shard definition> ::=
    <list shard definition> AT CLUSTER GROUP group_name
<list shard definition> ::=
    SHARD shard_name VALUES IN ( <list value clause> )
<list value clause> ::=
    <list value> [, ...]
<list value> ::=
    constant
    | NULL
    | DEFAULT
<table attribute clause> ::=
    [ <table physical attribute clause> ]
    | [ STORAGE ( <segment attr clause> [...] ) ]
<table physical attribute clause> ::=
    PCTFREE integer
    | PCTUSED integer
    | INITTRANS integer
    | MAXTRANS integer
<index physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer
<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>

```

```

<size clause> ::=
    integer [ K | M | G | T ]
<logging clause> ::=
    LOGGING
    | NOLOGGING
<constraint characteristics> ::=
    [ NOT ] DEFERRABLE [ <constraint check time> ]
    | <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
    INITIALLY DEFERRED
    | INITIALLY IMMEDIATE
<table global secondary index clause> ::=
    WITH GLOBAL SECONDARY INDEX [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    | WITHOUT GLOBAL SECONDARY INDEX

```

사용 범위 및 접근 권한

Database가 standalone 인지 아니면 cluster 인지에 따라 다음과 같은 차이가 있다

- Standalone
 - <table sharding strategy>를 정의할 수 없다.
 - <table global secondary index clause>를 정의할 수 없다.
- Cluster
 - UNIQUE, PRIMARY KEY 제약 조건을 정의할 때 모든 sharding key를 포함해야 한다.
 - 지연가능한 제약 조건을 정의할 수 없다.

<table definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 테이블이 생성될 스키마에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (CREATE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY TABLE ON DATABASE
- 테이블이 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
 - USAGE TABLESPACE ON DATABASE
- 함께 생성한 제약 조건이 있을 경우, 제약 조건이 생성될 스키마에 대해 다음 권한 중 하나가 있어야 한다.
 - 해당 스키마에 대해 (ADD CONSTRAINT 또는 CONTROL SCHEMA) ON SCHEMA
 - ALTER ANY TABLE ON DATABASE
- 함께 생성한 제약 조건이 key 제약 조건일 경우, 인덱스가 생성될 테이블스페이스에 대해 다음 권한 중 하나가 있

어야 한다.

- 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
- USAGE TABLESPACE ON DATABASE
- 테이블의 소유자는 다음과 같이 결정된다.
 - 테이블이 속한 스키마의 소유자
 - 테이블이 속한 스키마가 PUBLIC일 경우, 구문을 수행한 사용자
- 테이블 소유자는 생성한 테이블에 대해 다음과 같은 권한을 갖는다.
 - 해당 테이블에 대한 권한
 - SELECT ON TABLE WITH GRANT OPTION
 - INSERT ON TABLE WITH GRANT OPTION
 - UPDATE ON TABLE WITH GRANT OPTION
 - DELETE ON TABLE WITH GRANT OPTION
 - TRIGGER ON TABLE WITH GRANT OPTION
 - REFERENCES ON TABLE WITH GRANT OPTION
 - LOCK ON TABLE WITH GRANT OPTION
 - INDEX ON TABLE WITH GRANT OPTION
 - ALTER ON TABLE WITH GRANT OPTION
 - 해당 테이블의 모든 column에 대한 권한
 - SELECT(columns) ON TABLE WITH GRANT OPTION
 - INSERT(columns) ON TABLE WITH GRANT OPTION
 - UPDATE(columns) ON TABLE WITH GRANT OPTION
 - REFERENCES(columns) ON TABLE WITH GRANT OPTION
 - 함께 생성한 제약 조건에 대한 권한
 - 제약 조건의 소유자
 - 제약 조건과 함께 생성된 인덱스의 소유자

〈table sharding strategy〉 구문은 cluster system에서 사용할 수 있다.

구문 규칙 및 파라미터

table_name

생성할 테이블의 이름이며, 스키마 내에서 유일한 이름이어야 한다.

schema_name.table_name과 같이 테이블이 속할 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

테이블 이름의 길이는 128 바이트보다 작아야 한다.

〈column definition〉

테이블을 구성할 column을 정의한다.

테이블은 하나 이상의 column에 대한 정의를 포함해야 한다.

Column의 데이터 타입, 기본값, 자동 생성 값, 제약 조건 등을 기술할 수 있다.

column_name

테이블을 구성할 column의 이름으로 각 column은 테이블 내에서 유일한 이름을 가져야 한다.

Column 이름의 길이는 128 바이트보다 작아야 한다.

〈data type〉

- 〈character string type〉
- 〈binary string type〉
- 〈numeric type〉
- 〈exact numeric type〉
- 〈approximate numeric type〉
- 〈boolean type〉
- 〈datetime type〉
- 〈interval type〉
- 〈interval qualifier〉
- 〈non-second primary datetime field〉

Column의 데이터 타입을 정의한다.

자동 생성 값을 갖는 (〈identity column specification〉) column을 정의할 경우, SMALLINT, INTEGER, BIGINT 타입 중 하나의 데이터 타입을 사용해야 한다.

데이터 타입과 관련한 자세한 내용은 **Data Type** 정의를 참조한다.

〈character length units〉

Character 타입의 문자 하나당 길이 단위를 지정한다.

- CHARACTERS/ CHAR는 문자 하나의 최대 byte만큼을 문자 하나의 길이로 지정한다. 따라서 한글과 같은 multi-bytes 문자 하나의 길이도 1로 처리한다.
- OCTETS/ BYTE는 1 byte를 문자 하나의 길이로 지정한다. 따라서 한글과 같은 multi-bytes 문자 하나의 길이는 multi-bytes로 처리한다.
- 생략할 경우 database를 생성할 때 사용된 CHAR_LENGTH_UNITS 속성값을 따른다.

SQL 표준의 기본값은 CHARACTERS이다.

노트

다른 DBMS의 char length unit 기본값은 다음과 같다.

- Oracle, DB2: OCTETS
- MS-SQL, MySQL, PostgreSQL: CHARACTERS

[<default clause> | <identity column specification>]

Column의 기본값을 명시한다.

<default clause>와 <identity column specification>은 함께 사용할 수 없다.

모두 생략할 경우, 기본값은 NULL이다.

<default clause>

DEFAULT 절은 INSERT, UPDATE와 같은 구문에 DEFAULT가 명시되거나 해당 column 이름이 생략될 경우에 사용할 기본값을 정의한다.

- DEFAULT 절이 사용되는 경우
 - 예: CREATE TABLE t1 (id INTEGER, name VARCHAR(32) DEFAULT 'anonymous');
 - Column이 생략된 경우
 - INSERT INTO t1(id) VALUES (1);
 - INSERT INTO t1(id) SELECT id FROM other_table;
 - DEFAULT를 명시한 경우
 - INSERT INTO t1 DEFAULT VALUES;
 - INSERT INTO t1 VALUES (2, DEFAULT);
 - UPDATE t1 SET name = DEFAULT;

DEFAULT expression의 데이터 타입은 column의 데이터 타입과 호환 가능해야 한다.

타입이 호환되지 않거나 공간이 부족한 경우 INSERT, UPDATE 구문에서 DEFAULT를 사용할 때 다음과 같은 예러가 발생한다.

- CREATE TABLE t1 (user_name VARCHAR(1) DEFAULT CURRENT_USER);
- 예) error: INSERT INTO t1 VALUES (DEFAULT);
- 예) error: UPDATE t1 SET user_name = DEFAULT;

DEFAULT expression은 모든 built-in 함수를 사용할 수 있지만 다음은 사용할 수 없다.

- 논리연산자 (AND, OR, NOT), 비교 연산자 (=, >, ...)
- Stored function

- Column 이름
- Subquery expression

〈identity column specification〉

자동 생성값을 갖는 column을 정의한다.

테이블은 하나의 identity column만 가질 수 있다.

NOT NULL 제약 조건을 명시하지 않아도 identity column은 not nullable column이 된다.

〈identity column specification〉 절은 DEFAULT 절과 함께 기술할 수 없다.

〈identity column specification〉 절은 DEFAULT 절과 마찬가지로 INSERT, UPDATE 구문에서 DEFAULT를 명시하거나 해당 column 이름이 생략될 경우에 사용할 기본값을 정의한다.

생성 방식은 다음과 같이 정의된다.

- GENERATED BY DEFAULT AS IDENTITY

사용자가 값을 지정한 경우 이를 적용하나, DEFAULT 절과 같이 기본값이 사용되어야 하는 경우 자동값을 생성한다.

- CREATE TABLE t1 (id INTEGER GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(32));
- (O) INSERT INTO t1 VALUES (12345, 'GOLDILOCKS');
 - 사용자가 지정한 값(12345) 를 입력한다.
- (O) INSERT INTO t1(name) VALUES ('GOLDILOCKS');
 - id column에 자동 생성값이 입력된다.
- (O) INSERT INTO t1(id, name) SELECT other_id, other_name FROM other_table;
 - 사용자가 지정한 값을 입력한다.
- (O) INSERT INTO t1(name) SELECT other_name FROM other_table;
 - id column에 자동 생성값이 입력된다.
- (O) UPDATE t1 SET id = 10000 WHERE id = 12345;
 - 사용자가 지정한 값을 입력한다.
- (O) UPDATE t1 SET id = DEFAULT WHERE id = 12345;
 - id column에 자동 생성값이 입력된다.

- GENERATED ALWAYS AS IDENTITY

사용자가 값을 지정할 수 없으며, DEFAULT 절과 같이 기본값을 생성할 수 있어야 한다.

- CREATE TABLE t1 (id INTEGER GENERATED ALWAYS AS IDENTITY, name VARCHAR(32));
- (X) INSERT INTO t1 VALUES (12345, 'GOLDILOCKS');
 - 에러, 사용자가 값을 지정할 수 없다.
- (O) INSERT INTO t1(name) VALUES ('GOLDILOCKS');
 - id column에 자동 생성값이 입력된다.
- (X) INSERT INTO t1(id, name) SELECT other_id, other_name FROM other_table;
 - 에러, 사용자가 값을 지정할 수 없다.
- (O) INSERT INTO t1(name) SELECT other_name FROM other_table;

- id column에 자동 생성값이 입력된다.
- (X) UPDATE t1 SET id = 10000 WHERE id = 12345;
 - 에러, 사용자가 값을 지정할 수 없다.
- (O) UPDATE t1 SET id = DEFAULT WHERE id = 12345;
 - id column에 자동 생성값이 입력된다.

identity column 생성 옵션인 <common sequence generator option>과 <basic sequence generator option>에 대한 자세한 내용은 CREATE SEQUENCE 구문을 참조한다.

<column constraint definition>

Column에 대해 다음과 같은 제약 조건을 정의한다.

- NOT NULL 제약 조건
- UNIQUE 제약 조건
- PRIMARY KEY 제약 조건

constraint_name

제약 조건의 이름이며 생략 가능하다.

constraint_name을 생략할 경우 다음과 같은 형태로 제약 조건 이름을 자동으로 설정한다. 자동 생성하는 이름이 중복될 경우, constraint_name을 명시적으로 부여해야 한다.

- NOT NULL 제약 조건
 - "table_name" + "_" + "NOT_NULL" + "_" + "column_name"
- UNIQUE 제약 조건
 - "table_name" + "_" + "UNIQUE" + "_" + "column_name"
- PRIMARY KEY 제약 조건
 - "table_name" + "_" + "PRIMARY_KEY"

제약 조건의 이름은 128 바이트보다 작아야 한다.

NOT NULL 제약 조건

Column 값으로 NULL 값을 허용하지 않는다.

UNIQUE 제약 조건

Column 값으로 동일한 값을 허용하지 않는다.

단, NULL 값은 허용한다.

PRIMARY KEY 제약 조건

Column 값으로 NULL 값이나 동일한 값을 허용하지 않는다.

하나의 테이블에 하나의 PRIMARY KEY 제약 조건을 정의할 수 있다.

<index name clause>

UNIQUE 제약 조건, PRIMARY KEY 제약 조건을 정의할 때 생성되는 인덱스의 이름을 정의한다.

- INDEX index_name
 - 제약 조건을 위한 인덱스의 이름을 정의한다.
 - 스키마 이름과 함께 사용할 수 없으며, 제약 조건과 동일한 스키마에 생성된다.

UNIQUE 제약 조건, PRIMARY KEY 제약 조건을 정의할 때 INDEX 절을 생략할 경우에는 제약 조건에 부합하는 인덱스를 자동으로 생성한다.

자동 생성되는 인덱스 이름으로는 "constraint_name" + "_INDEX"가 부여된다.

- <index attributes>
 - 생성할 인덱스의 물리적 속성을 지정한다.
 - 자세한 내용은 CREATE INDEX 구문을 참조한다.
- TABLESPACE index_tablespace_name
 - 인덱스를 생성할 tablespace를 지정한다.
 - 자세한 내용은 CREATE INDEX을 참조한다.

<table constraint definition>

- <unique constraint definition>
- 테이블을 정의할 때의 제약 조건은 구문 내의 위치에 따라 두 가지 방법으로 나뉜다.
 - Column을 정의할 때 column 제약 정의 (<column constraint definition>)를 사용하여 하나의 column에 대한 제약 조건을 기술할 수 있다.
 - 이에 반해, 테이블 제약 정의 (<table constraint definition>)는 column 정의와 별도로 기술할 수 있으며, 하나 이상의 column에 대한 제약 조건을 기술할 수 있다.

테이블 제약 정의는 column 제약 정의와 비교하여 다음과 같은 구문상의 차이가 있다.

- NOT NULL 제약
 - 테이블 제약 정의를 통해 명시할 수 없다.
- Column 제약 정의와 달리 column을 명시해야 한다.
 - UNIQUE 제약 <unique constraint definition>
 - UNIQUE (column_name [, ...])
 - PRIMARY KEY 제약 <unique constraint definition>

- PRIMARY KEY (column_name [, ...])

key column element

Key 대상이 되는 column을 지정한다.

- Column name
 - Key를 생성할 column 이름이다.
- ASC | DESC
 - ASC: 오름차순으로 정렬한다.
 - DESC: 내림차순으로 정렬한다.
 - 명시하지 않을 경우, 기본값은 ASC 이다.
- NULLS FIRST | NULLS LAST
 - NULLS FIRST: NULL이 아닌 값들보다 앞에 위치한다.
 - NULLS LAST: NULL이 아닌 값들보다 뒤에 위치한다.
 - 명시하지 않을 경우, 기본값은 NULLS LAST 이다.

<table sharding strategy>

테이블의 sharding 정책을 정의한다.

다음과 같은 네 가지 정책 중 하나로 정의할 수 있다.

- <cloned strategy>
- <hash sharding strategy>
- <range sharding strategy>
- <list sharding strategy>

생략할 경우 **DEFAULT_SHARDING** 프로퍼티 값에 의해 결정된다.

- **DEFAULT_SHARDING** 값이 0 인 경우
 - <cloned strategy>
- **DEFAULT_SHARDING** 값이 1 인 경우
 - <hash sharding strategy>

<cloned strategy>

테이블의 모든 data를 복제한다.

<clone placement>

Clone의 배치 정책을 정의한다.

- AT CLUSTER WIDE
 - Cluster system의 모든 cluster group의 모든 cluster member에 clone을 배치한다.
 - Cluster group과 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 clone을 재배포할 수 있다.
- AT CLUSTER GROUP group_list
 - 지정한 cluster group들의 모든 cluster member에 clone을 배치한다.
 - 지정된 cluster group에 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 clone을 재배포할 수 있다.
 - Cluster group 추가는 clone의 재배포에 영향을 주지 않는다.
- 생략할 경우, 기본값은 AT CLUSTER WIDE이다.

<hash sharding strategy>

테이블 data를 sharding key의 hash 값을 기준으로 shard 분할한다.

SHARDING BY [HASH] (column_list)

Hash sharding을 위한 sharding key를 정의한다.

- Column을 32 개까지 나열할 수 있다.
- 동일한 column은 사용할 수 없다.
- LONG VARCHAR, LONG VARBINARY 타입의 column은 사용할 수 없다.

<hash shard count>

분할할 hash shard의 개수를 정의한다.

Shard의 개수는 1부터 512까지 정의할 수 있다.

생략할 경우 기본값은 24이다.

<hash shard placement>

Hash shard의 배치 정책을 정의한다.

- AT CLUSTER WIDE
 - Cluster system의 모든 cluster group의 모든 cluster member에 shard들을 배치한다.
 - Cluster group과 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 shard들을 재배포할 수 있다.
- AT CLUSTER GROUP group_list

- 지정한 cluster group들의 모든 cluster member에 hash shard들을 배치한다.
- group_list의 개수는 <hash shard count>의 값과 같거나 작아야 한다.
- range shard, list shard와 달리 hash shard는 특정 shard가 배치될 cluster group을 지정할 수 없으며, system이 자동으로 shard들을 배치할 cluster group을 결정한다.
- 지정한 cluster group에 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 shard를 재배포할 수 있다.
- Cluster group 추가는 hash shard의 재배포에 영향을 주지 않는다.
- 생략할 경우, 기본값은 AT CLUSTER WIDE 이다.

<range sharding strategy>

테이블의 data를 sharding key의 범위값을 기준으로 shard 분할한다.

SHARDING BY RANGE (column_list)

Range sharding을 위한 sharding key를 정의한다.

- Column을 32 개까지 나열할 수 있다.
- 동일한 column은 사용할 수 없다.
- LONG VARCHAR, LONG VARBINARY 타입의 column은 사용할 수 없다.

<cluster-wide range shard placement>

Range shard들을 cluster system의 모든 cluster group으로 자동으로 배치한다.

<range shard definition>을 기술하기 전에 AT CLUSTER WIDE 구문을 기술한다.

Cluster group과 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 자동으로 shard 들을 재배포할 수 있다.

- Range sharded table을 생성한다.
- 기존 cluster group인 g1, g2, g3에 여섯 개의 shard들을 배치한다.

```
CREATE TABLE t1
(
  id    INTEGER,
  name  VARCHAR(32)
)
SHARDING BY RANGE (id)
  AT CLUSTER WIDE
  SHARD s1 VALUES LESS THAN ( 200000 ),
  SHARD s2 VALUES LESS THAN ( 400000 ),
  SHARD s3 VALUES LESS THAN ( 500000 ),
  SHARD s4 VALUES LESS THAN ( 600000 ),
```



```
SHARD s5 VALUES LESS THAN ( 800000 ),
SHARD s6 VALUES LESS THAN ( MAXVALUE )
;
```

- Cluster group을 추가한다.

```
CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Range shard를 재배치한다.
- 추가된 g4를 포함하여 cluster group g1, g2, g3, g4에 여섯 개의 shard들을 재배치한다.

```
ALTER TABLE t1 REBALANCE;
```

<group-specific range shard placement>

Range shard들을 지정한 cluster group에 배치한다.

<range shard definition>과 함께 해당 shard를 배치할 AT CLUSTER GROUP group_name 구문을 기술한다.

지정한 cluster group에 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 자동으로 shard들을 재배치할 수 있다.

Cluster group 추가는 range shard의 재배치에 영향을 주지 않는다.

- Range sharded table을 생성한다.
- 각 range shard들을 지정한 cluster group에 배치한다.

```
CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY RANGE (id)
    SHARD s1 VALUES LESS THAN ( 200000 )    AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( 400000 )    AT CLUSTER GROUP g2,
    SHARD s3 VALUES LESS THAN ( 500000 )    AT CLUSTER GROUP g3,
    SHARD s4 VALUES LESS THAN ( 600000 )    AT CLUSTER GROUP g2,
    SHARD s5 VALUES LESS THAN ( 800000 )    AT CLUSTER GROUP g3,
    SHARD s6 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g1
;
```

- Cluster group을 추가한다.

```
CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Range shard를 재배치한다.
- 새로 생성한 cluster group g4 에는 shard가 배치되지 않는다

```
ALTER TABLE t1 REBALANCE;
```

〈range shard definition〉

SHARD range_name은 테이블 내에서 유일해야 한다.

최대 512 개의 〈range shard definition〉을 정의할 수 있다.

나열된 〈range shard definition〉은 〈range value clause〉의 순서로 정렬되며 서로 다른 〈range value clause〉를 사용해야 한다.

모든 값을 MAXVALUE로 정의한 〈range shard definition〉을 MAX shard라 한다.

MAX shard는 반드시 존재해야 하며, 하나만 존재해야 한다.

- MAX shard를 포함해야 한다.

```
gSQL>
CREATE TABLE t1
(
    id INTEGER,
    name VARCHAR(32)
)
SHARDING BY RANGE (id)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 100000 ),
    SHARD s2 VALUES LESS THAN ( 200000 ),
    SHARD s3 VALUES LESS THAN ( MAXVALUE )
;
```

Table created.

- MAX shard를 포함하지 않는 경우 error가 발생한다.

```
gSQL>
CREATE TABLE t1
(
    id INTEGER,
```

```

    name VARCHAR(32)
)
SHARDING BY RANGE (id)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 100000 ),
    SHARD s2 VALUES LESS THAN ( 200000 ),
    SHARD s3 VALUES LESS THAN ( 300000 )
;
ERR-42000(16377): MAX shard not defined :
    SHARD s3 VALUES LESS THAN ( 300000 )
*
ERROR at line 10:

```

〈range value clause〉

〈range value〉는 상수값이거나 최대값을 의미하는 MAXVALUE여야 한다.

NULL 값은 〈range value〉로 사용할 수 없다.

- (O) SHARD s1 VALUES LESS THAN (1)
- (O) SHARD s2 VALUES LESS THAN (1 + 1)
- (O) SHARD s3 VALUES LESS THAN (MAXVALUE)
- (X) SHARD s4 VALUES LESS THAN (SYSDATE)
- (X) SHARD s5 VALUES LESS THAN (NULL)

MAXVALUE는 다른 값보다 항상 큰 값을 의미하며 null 값을 포함한다.

Sharding key가 여러 개인 경우 MAXVALUE 이후에는 MAXVALUE만 지정할 수 있다.

- (O) SHARD s1 VALUES LESS THAN (100, MAXVALUE)
- (X) SHARD s2 VALUES LESS THAN (MAXVALUE, 100)
- (O) SHARD s3 VALUES LESS THAN (MAXVALUE, MAXVALUE)

다수의 column을 사용하여 sharding key를 정의한 경우 다음 SHARD s3와 같이 모든 값을 MAXVALUE 로 나열한 MAX shard가 반드시 하나만 존재해야 한다.

```

CREATE TABLE t1
(
    id INTEGER,
    name VARCHAR(32)
)
SHARDING BY RANGE (id, name)
    AT CLUSTER WIDE

```

```

SHARD s1 VALUES LESS THAN ( 100000, MAXVALUE ),
SHARD s2 VALUES LESS THAN ( 200000, 200000 ),
SHARD s3 VALUES LESS THAN ( MAXVALUE, MAXVALUE )
;

```

〈list sharding strategy〉

테이블 data를 sharding key의 나열값을 기준으로 shard 분할한다.

SHARDING BY LIST (column_name)

List sharding을 위한 sharding key를 정의한다.

- 하나의 column만 사용할 수 있다.
- LONG VARCHAR, LONG VARBINARY 타입의 column을 사용할 수 없다.

〈cluster-wide list shard placement〉

Cluster system의 모든 cluster group에 list shard들을 자동으로 배치한다.

〈list shard definition〉을 기술하기 전에 AT CLUSTER WIDE 구문을 기술한다.

Cluster group과 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 자동으로 shard들을 재배포할 수 있다.

- List sharded table을 생성한다.
- 기존 cluster group인 g1, g2, g3에 다섯 개의 shard들을 배치한다.

```

CREATE TABLE city
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY LIST (name)
    AT CLUSTER WIDE
    SHARD s1 VALUES IN ( 'SEOUL' ),
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ),
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' ),
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' ),
    SHARD s5 VALUES IN ( DEFAULT )
;

```

- Cluster group을 추가한다.

```
CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- List shard를 재배치한다.
- 추가된 g4를 포함하여 g1, g2, g3, g4 cluster group에 다섯 개의 shard들을 재배치한다.

```
ALTER TABLE t1 REBALANCE;
```

〈group-specific list shard placement〉

List shard들을 지정한 cluster group에 배치한다.

〈list shard definition〉과 함께 해당 shard를 배치할 AT CLUSTER GROUP group_name 구문을 기술한다.

지정한 cluster group에 cluster member를 추가할 때 **ALTER TABLE name REBALANCE** 구문을 사용하여 자동으로 shard들을 재배치할 수 있다.

Cluster group 추가는 list shard의 재배치에 영향을 주지 않는다.

- List sharded table을 생성한다.
- 각 list shard를 지정한 cluster group에 배치한다.

```
CREATE TABLE city
(
    id    INTEGER,
    name  VARCHAR(32)
)
SHARDING BY LIST (name)
    SHARD s1 VALUES IN ( 'SEOUL' )           AT CLUSTER GROUP g1,
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ) AT CLUSTER GROUP g2,
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' )   AT CLUSTER GROUP g3,
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' )      AT CLUSTER GROUP g2,
    SHARD s5 VALUES IN ( DEFAULT )              AT CLUSTER GROUP g1
;
```

- Cluster group을 추가한다.

```
CREATE CLUSTER GROUP g4
    CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- List shard를 재배치한다.
- 추가한 cluster group g4에 shard가 배치되지 않는다.

```
ALTER TABLE t1 REBALANCE;
```

〈list shard definition〉

LIST list_name 은 테이블 내에서 유일해야 한다.

최대 512개의 〈list shard definition〉을 정의할 수 있다.

나열된 〈list shard definition〉의 모든 〈list value〉 값이 서로 달라야 한다.

DFFAULT는 나열된 모든 〈list value〉를 제외한 나머지 값이다.

DEFAULT는 다른 값과 함께 지정할 수 없다.

DEFAULT를 포함하는 shard를 DEFAULT shard라고 한다.

DEFAULT shard는 반드시 존재해야 하며, 하나만 존재해야 한다.

- DEFAULT shard를 포함해야 한다.

```
gSQL>
CREATE TABLE t1
(
    category INTEGER,
    name      VARCHAR(32)
)
SHARDING BY LIST (category)
    AT CLUSTER WIDE
    SHARD s1 VALUES IN ( 1, 3, 5, 7 ),
    SHARD s2 VALUES IN ( 2, 4, 6, 8 ),
    SHARD s3 VALUES IN ( DEFAULT )
;
Table created.
```

- DEFAULT shard를 포함하지 않는 경우 error가 발생한다.

```
gSQL>
CREATE TABLE t1
(
    category INTEGER,
    name      VARCHAR(32)
)
SHARDING BY LIST (category)
    AT CLUSTER WIDE
    SHARD s1 VALUES IN ( 1, 3, 5, 7 ),
    SHARD s2 VALUES IN ( 2, 4, 6, 8 ),
```

```

    SHARD s3 VALUES IN ( 9, 10 )
;
ERR-42000(16385): DEFAULT shard not defined :
    SHARD s3 VALUES IN ( 9, 10 )
*
ERROR at line 10:

```

〈list value clause〉

〈list value〉는 상수값이어야 한다.

NULL 값이나 DEFAULT를 〈list value〉로 사용할 수 있다.

DEFAULT는 다른 값과 함께 지정할 수 없다.

- (O) SHARD s1 VALUES IN (1, 1 + 1, 3, 4)
- (O) SHARD s2 VALUES IN (5, 6, 7, NULL)
- (O) SHARD s3 VALUES IN (DEFAULT)
- (X) SHARD s4 VALUES IN (DEFAULT, 8, 9, 10)
- (X) SHARD s5 VALUES IN (current_timestamp, systimestamp)
- (X) SHARD s6 VALUES IN (c1, c2)

〈table physical attribute clause〉

테이블의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내에서 row를 수정하거나 업데이트 할 때 행 크기가 증가될 것에 대비하여 예약된 공간이다.
 - 초기에는 이 공간을 제외하고 입력된다.
 - PCTFREE가 부족하면 데이터를 수정하거나 업데이트 할 때 행 이전 (ROW MIGRATION)이 발생한다.
 - 0 에서 99 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 10이다.
- PCTUSED integer
 - 정의
 - 새로운 row가 페이지에 추가되기 전에 row 데이터와 오버헤드에 대해 사용될 수 있는 페이지의 최소 퍼센트이다.
 - 즉, 기존 데이터의 수정이나 삭제 등으로 인해 PCTUSED보다 값이 작아지면 이 페이지들에 한하여 입력이 가능하다.
 - 0 에서 99 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 40 이다.

- INITRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 초기 트랜잭션의 개수이다.
 - 인덱스에 접근하는 사용자의 수가 적을 경우에는 INITRANS를 낮게 설정하고, 동시에 접근하는 사용자가 많을 경우에는 INITRANS를 높게 설정한다.
 - 필요한 경우 설정된 MAXTRANS까지 자동으로 늘어난다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 4 다.
- MAXTRANS integer
 - 정의
 - 페이지에 동시 접근할 수 있는 트랜잭션의 최대 개수이다.
 - 1 부터 32 까지의 값을 사용할 수 있다.
 - 생략할 경우, 기본값은 8 이다.

〈index physical attribute clause〉

인덱스의 물리적 속성 정보를 정의한다.

- PCTFREE integer
 - 정의
 - 페이지 내 key 삽입으로 인한 페이지 분할 빈도를 조절하기 위해 예약된 공간이다.
 - 인덱스 bottom-up 빌드 시에만 적용된다.
 - 0 에서 99 까지의 값을 사용할 수 있다.
 - 생략할 경우, DEFAULT_INDEX_PCTFREE property에 설정된 값을 사용한다.
- INITRANS integer
 - 〈table physical attribute clause〉의 INITRANS와 동일하다.
- MAXTRANS integer
 - 〈table physical attribute clause〉의 MAXTRANS와 동일하다.

〈segment attr clause〉

테이블이 저장될 공간에 대한 정보를 기술한다.

- INITIAL integer
 - 정의
 - 테이블을 생성할 때 초기에 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT 크기가 8192 bytes일 경우 'INITIAL 100'은 실제 8192 bytes로 작동한다.)
 - 이 크기 (TABLESPACE의 EXTENT 크기에 align 된 크기)는 MINEXTENTS의 크기와 같거나 커야 하며, MAXEXTENTS의 크기와 같거나 작아야 한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.

- 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- NEXT integer
 - 정의
 - 테이블의 물리적 공간을 추가할 경우 할당할 물리적 공간의 크기를 기술한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다. (예: EXT크기가 8192 bytes일 경우 'NEXT 100'은 실제 8192 bytes로 작동한다.)
 - NEXT는 현재 테이블이 사용할 수 있는 남은 공간의 크기 (MAXEXTENTS의 크기에서 현재 사용하고 있는 공간의 크기를 뺀 크기)에 따라 아래와 같이 작동한다.
 - 남은 공간의 크기가 0인 경우 더 이상 공간을 확장하지 못한다.
 - 남은 공간의 크기가 0보다 크고 NEXT보다 작은 경우 남은 공간의 크기만큼 할당한다.
 - 남은 공간의 크기가 NEXT 보다 클 경우 NEXT의 크기만큼 할당한다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 테이블이 속한 TABLESPACE의 EXTENT 하나 크기이다.
- MINSIZE integer
 - 정의
 - 테이블에서 유지해야할 최소 공간의 크기이다.
 - 이 값은 MAXSIZE의 값과 같거나 작아야 한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - EXTENT 두 개의 크기보다 작을 경우 EXTENT 두 개 크기로 결정된다.
 - 생략할 경우, 기본값은 EXTENT 두 개 크기이다.
- MAXSIZE integer
 - 정의
 - 테이블에서 할당받을 수 있는 최대 공간의 크기이다.
 - 이 값은 MINSIZE의 값과 같거나 커야 한다.
 - 이 크기는 테이블이 속하는 TABLESPACE의 EXTENT 크기에 align 되어 사용된다.
 - 최소값은 1이고 최대값은 시스템 환경에 따라 다르다.
 - 생략할 경우, 기본값은 EXTENT 크기 * 2147483647 (INT32의 최대 양의 정수)이다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

TABLESPACE tablespace_name

테이블이 저장될 tablespace의 이름을 지정한다.

TABLESPACE 절을 생략할 경우, 구문을 수행하는 사용자의 기본 tablespace_name을 사용한다.

LOGGING | NOLOGGING

인덱스의 리두 로깅 수행 여부를 명시한다.

명시하지 않을 경우, 기본값은 NOLOGGING이다.

TABLESPACE index_tablespace_name

인덱스가 저장될 tablespace의 이름을 지정한다.

TABLESPACE 절을 생략할 경우, LOGGING 인덱스는 사용자의 기본 데이터 테이블스페이스를 사용하고, NOLOGGING 인덱스는 사용자의 기본 임시 테이블스페이스를 사용한다.

<constraint characteristics>

제약 조건의 특성을 정의한다.

제약 조건을 정의할 때 다음과 같은 특성들을 설정할 수 있다.

- 제약 조건의 지연가능성 (DEFERRABLE | NOT DEFERRABLE)
- 제약 조건의 검사시점 (<constraint check time>)

<constraint characteristics>를 생략할 경우, NOT DEFERRABLE INITIALLY IMMEDIATE로 설정한다.

DEFERRABLE | NOT DEFERRABLE

제약 조건을 DML을 수행할 때 검사하지 않고, COMMIT을 수행할 때 검사할 수 있게 지연시킬 수 있는지 여부를 설정한다.

지연 가능한 제약 조건의 검사시점은 SET CONSTRAINTS 구문으로 제어한다.

- NOT DEFERRABLE
 - 검사 시점을 지연시킬 수 없으며, INSERT, DELETE, UPDATE 구문을 수행할 때 제약 조건을 검사한다.
- DEFERRABLE
 - 검사 시점을 SET CONSTRAINTS 구문으로 제어할 수 있다.
 - SET CONSTRAINTS constraint_name IMMEDIATE
 - DML을 수행할 때 제약 조건을 검사한다.
 - SET CONSTRAINTS constraint_name DEFERRED
 - COMMIT을 수행할 때 제약 조건을 검사한다.
- 명시하지 않을 경우 기본값은 <constraint check time>에 따라 결정된다.

- INITIALLY IMMEDIATE를 명시한 경우, NOT DEFERRABLE 이다.
- INITIALLY DEFERRED를 명시한 경우, DEFERRABLE 이다.
- <constraint check time>을 명시하지 않은 경우, NOT DEFERRABLE 이다.

<constraint check time>

지연 가능한 (DEFERRABLE) 제약 조건일 경우, 검사 시점의 초기값을 설정한다.

- INITIALLY IMMEDIATE
 - DML을 수행할 때 제약 조건을 검사한다.
- INITIALLY DEFERRED
 - COMMIT을 수행할 때 제약 조건을 검사한다.
 - NOT DEFERRABLE과 함께 사용할 수 없다.
- 명시하지 않을 경우, 기본값은 INITIALLY IMMEDIATE 이다.

지연 가능한 제약 조건에 대한 자세한 내용은 SET CONSTRAINTS 구문을 참조한다.

<table global secondary index clause>

테이블의 global secondary index를 정의한다.

- WITH GLOBAL SECONDARY INDEX [<index attributes> [...]] [TABLESPACE tablespace_name]
 - Cluster system 환경에서 테이블을 생성할 때 global secondary index를 생성한다.
 - <index attribute>
 - Global secondary index의 index attribute를 설정한다.
 - TABLESPACE tablespace_name
 - Global secondary index를 생성할 tablespace를 지정한다.
- WITHOUT GLOBAL SECONDARY INDEX
 - Cluster system 환경에서 테이블을 생성할 때 global secondary index를 생성하지 않도록 한다.

설명

제약 조건의 특성

GOLDILOCKS는 key 제약 조건을 생성할 때 uniqueness 검사를 하기 위해 자동으로 index를 생성한다.

다음과 같은 column은 NULL 값을 허용하지 않는다.

- NOT NULL 제약 조건이 있는 column
- Primary key 제약 조건에 포함되는 column
- Identity column

Cluster Table

Cluster 환경에서 테이블은 다음 중 하나의 sharding 정책으로 데이터를 관리한다.

- Cloned table
 - 테이블의 모든 data를 복제하여 cluster system에 배치한다.
- Hash sharded table
 - 테이블의 data를 sharding key의 해쉬 (hash) 값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.
- Range sharded table
 - 테이블의 data를 sharding key의 범위 (range) 값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.
- List sharded table
 - 테이블의 data를 sharding key의 나열 (list) 값을 기준으로 여러 개의 shard로 분할하여 cluster system에 배치한다.

테이블을 생성할 때 다음을 고려하여 sharding 정책을 결정한다. Cluster system 상에서 운영되는 테이블들은 그 특성에 따라 code table과 fact table로 구분할 수 있다.

- Code table
 - 제품 목록, 공급자 목록 등과 같이 데이터의 변경이 적으며 데이터 양이 적은 테이블
 - Fact table과 함께 자주 참조되는 테이블
- Fact table
 - 거래 내역, 통화 내역 등과 같이 데이터의 변경이 많으며 데이터 양이 많은 테이블
 - 데이터 양이 많아 sharding이 필요한 테이블

Code table에는 <cloned strategy>가 바람직하며, fact table의 경우 테이블의 접근 패턴에 따라 <table sharding strategy>를 결정해야 한다.

사용 예

다음은 일반 테이블을 생성하는 예이다.

```
gSQL> CREATE TABLE region
(
    r_regionkey    INTEGER
  , r_name        CHAR(25)
  , r_comment     VARCHAR(152)
);
Table created.
```

다음은 테이블 생성할 때 column에 제약 조건을 기술하는 예이다.

```
gSQL> CREATE TABLE supplier
(
    s_suppkey    INTEGER PRIMARY KEY
  , s_name      CHAR(25) NOT NULL
  , s_address   VARCHAR(40)
  , s_nationkey INTEGER
  , s_phone     CHAR(15)
  , s_acctbal   NUMERIC(12,2)
  , s_comment   VARCHAR(101)
);
Table created.
```

다음은 테이블 생성할 때 여러 column을 포함하는 제약 조건을 기술하는 예이다.

```
gSQL> CREATE TABLE partsupp
(
    ps_partkey   INTEGER
  , ps_suppkey   INTEGER
  , ps_availqty  INTEGER
  , ps_supplycost NUMERIC(12,2)
  , ps_comment   VARCHAR(199)
  , CONSTRAINT ps_unique_key UNIQUE(ps_partkey, ps_suppkey)
);
Table created.
```

다음은 테이블을 생성할 때 지연 가능 여부를 포함한 제약 조건을 기술하는 예이다.

```
gSQL> CREATE TABLE t1
(
    id      NUMBER          PRIMARY KEY
                                NOT DEFERRABLE INITIALLY IMMEDIATE
  , name    VARCHAR(128)    CONSTRAINT t1_nn NOT NULL
                                DEFERRABLE INITIALLY IMMEDIATE
  , addr    VARCHAR(1024)
  , CONSTRAINT t1_uk UNIQUE ( id, name )
                                DEFERRABLE INITIALLY DEFERRED
);
Table created.
gSQL> COMMIT;
Commit complete.
```

다음은 테이블을 생성할 때 자동 생성값과 기본값을 갖는 column들을 기술하는 예이다.

```
CREATE TABLE customer
(
    c_custkey    INTEGER    GENERATED BY DEFAULT AS IDENTITY
  , c_name      VARCHAR(25)
  , c_address   VARCHAR(40) DEFAULT 'N/A'
  , c_nationkey INTEGER
  , c_phone     CHAR(15)
  , c_acctbal   NUMERIC(12,2)
  , c_mktsegment CHAR(10)
  , c_comment   VARCHAR(117)
);
Table created.
```

다음은 테이블을 생성할 때 저장될 tablespace를 지정하는 예이다.

```
gSQL> CREATE TABLE lineitem
(
    l_orderkey    INTEGER
  , l_partkey     INTEGER
  , l_suppkey     INTEGER
  , l_linenumber  INTEGER
  , l_quantity    NUMERIC(12,2)
  , l_extendedprice NUMERIC(12,2)
  , l_discount    NUMERIC(12,2)
  , l_tax         NUMERIC(12,2)
  , l_returnflag  CHAR(1)
  , l_linestatus  CHAR(1)
  , l_shipdate    DATE
  , l_commitdate  DATE
  , l_receiptdate DATE
  , l_shipinstruct CHAR(25)
  , l_shipmode    CHAR(10)
  , l_comment     VARCHAR(44)
  , PRIMARY KEY (l_orderkey, l_linenumber) INDEX lineitem_pk_idx TABLESPACE mem_temp_tbs
) TABLESPACE mem_data_tbs;
Table created.
```

다음은 cluster-wide cloned table을 정의하는 예이다. 테이블의 data를 cluster system 전체에 복제하여 배치한다.

```

gSQL>
CREATE TABLE region
(
    r_regionkey    INTEGER
  , r_name        CHAR(25)
  , r_comment     VARCHAR(152)
)
CLONED
AT CLUSTER WIDE
;
Table created.

```

다음은 group-specific cloned table을 정의하는 예이다. 테이블의 데이터는 사용자가 지정한 g1, g2 cluster group에 복제하여 배치한다.

```

gSQL>
CREATE TABLE region
(
    r_regionkey    INTEGER
  , r_name        CHAR(25)
  , r_comment     VARCHAR(152)
)
CLONED
AT CLUSTER GROUP g1, g2
;
Table created.

```

다음은 cluster-wide hash sharded table을 정의하는 예이다. 테이블의 데이터가 ps_partkey column의 hash 값에 의해 24개의 shard로 분할되며, 각 shard는 cluster system 전체에 자동으로 배치된다.

```

gSQL>
CREATE TABLE partsupp
(
    ps_partkey     INTEGER
  , ps_suppkey     INTEGER
  , ps_availqty    INTEGER
  , ps_supplycost  NUMERIC(12,2)
  , ps_comment     VARCHAR(199)
)
SHARDING BY HASH ( ps_partkey )
SHARD COUNT 24

```

```

AT CLUSTER WIDE
;
Table created.

```

다음은 group-specific hash sharded table을 정의하는 예이다. 테이블의 데이터가 ps_partkey column의 hash 값에 의해 24개의 shard로 분할되며, 각 shard는 지정한 cluster group g2, g3에 자동으로 배치된다.

```

gSQL>
CREATE TABLE partsupp
(
    ps_partkey    INTEGER
  , ps_suppkey    INTEGER
  , ps_availqty   INTEGER
  , ps_supplycost NUMERIC(12,2)
  , ps_comment    VARCHAR(199)
)
SHARDING BY HASH ( ps_partkey )
SHARD COUNT 24
AT CLUSTER GROUP g2, g3
;
Table created.

```

다음은 cluster-wide range sharded table을 정의하는 예이다. 테이블 데이터가 D_ID column의 range 값을 기준으로 여덟 개의 shard로 분할되고, 각 shard가 cluster system 전체에 자동으로 배치된다.

```

gSQL>
CREATE TABLE DISTRICT (
    D_ID          INTEGER,
    D_W_ID        INTEGER,
    D_NAME        VARCHAR(10),
    D_STREET_1    VARCHAR(20),
    D_STREET_2    VARCHAR(20),
    D_CITY        VARCHAR(20),
    D_STATE       CHAR(2),
    D_ZIP         CHAR(9),
    D_TAX         NUMERIC(4,4),
    D_YTD         NUMERIC(15,2),
    D_NEXT_O_ID   INTEGER,
    PRIMARY KEY (D_W_ID, D_ID) INDEX DISTRICT_PK_IDX
)
SHARDING BY RANGE (D_ID)
AT CLUSTER WIDE

```



```

SHARD s1 VALUES LESS THAN ( 100 ),
SHARD s2 VALUES LESS THAN ( 200 ),
SHARD s3 VALUES LESS THAN ( 300 ),
SHARD s4 VALUES LESS THAN ( 400 ),
SHARD s5 VALUES LESS THAN ( 500 ),
SHARD s6 VALUES LESS THAN ( 600 ),
SHARD s7 VALUES LESS THAN ( 700 ),
SHARD s8 VALUES LESS THAN ( MAXVALUE );

```

```
;
```

Table created.

다음은 group-specific range sharded table을 정의하는 예이다. 테이블 데이터는 NO_D_ID column의 range 값을 기준으로 세 개의 range 값으로 분할되고, s1 shard는 g1 cluster group에, s2 shard는 g2 cluster group에 그리고 s3 shard는 g3 cluster group에 각각 지정되어 배치된다.

```
gSQL>
```

```
CREATE TABLE NEW_ORDER
```

```
(
```

```
    NO_O_ID INTEGER,
```

```
    NO_D_ID INTEGER,
```

```
    NO_W_ID INTEGER,
```

```
    PRIMARY KEY(NO_W_ID, NO_D_ID, NO_O_ID) INDEX NEW_ORDER_PK_IDX
```

```
)
```

```
    SHARDING BY RANGE (NO_D_ID)
```

```
    SHARD s1 VALUES LESS THAN ( 5 )          AT CLUSTER GROUP g1,
```

```
    SHARD s2 VALUES LESS THAN ( 8 )          AT CLUSTER GROUP g2,
```

```
    SHARD s3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g3
```

```
;
```

Table created.

다음은 cluster-wide list sharded table을 정의하는 예이다. List shard가 city column을 기준으로 다섯 개로 분할 되고, 각 shard는 cluster system 전체에 자동으로 배치된다.

```
gSQL>
```

```
CREATE TABLE t1
```

```
(
```

```
    id    INTEGER
```

```
, name VARCHAR(32)
```

```
, city VARCHAR(128)
```

```
)
```

```
    SHARDING BY LIST (city)
```

```
    AT CLUSTER WIDE
```

```

SHARD s1 VALUES IN ( 'seoul' ),
SHARD s2 VALUES IN ( 'busan', 'ulsan' ),
SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ),
SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ),
SHARD s5 VALUES IN ( DEFAULT )
;

```

Table created.

다음은 group-specific list sharded table을 정의하는 예이다. List shard가 city column을 기준으로 다섯 개로 분할 되고, 각 shard는 지정된 cluster group에 배치된다.

```

gSQL>
CREATE TABLE t1
(
    id    INTEGER
  , name VARCHAR(32)
  , city VARCHAR(128)
)
SHARDING BY LIST (city)
    SHARD s1 VALUES IN ( 'seoul' )           AT CLUSTER GROUP g1,
    SHARD s2 VALUES IN ( 'busan', 'ulsan' )   AT CLUSTER GROUP g2,
    SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ) AT CLUSTER GROUP g1,
    SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ) AT CLUSTER GROUP g2,
    SHARD s5 VALUES IN ( DEFAULT )           AT CLUSTER GROUP g3
;

```

Table created.

Global secondary index 없이 테이블 T1을 생성한다.

```

gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) ) WITHOUT GLOBAL SECONDARY INDEX;
Table created.

```

테이블 T1을 생성한 후에 테이블 T1의 global secondary index를 생성한다.

```

gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) ) WITH GLOBAL SECONDARY INDEX;
Table created.

```

테이블 T1을 생성한 후에 테이블 T1의 global secondary index를 tablespace 'USER_DATA_TBS'에 logging option으로 생성한다.

```

gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )
      WITH GLOBAL SECONDARY INDEX

```

```
LOGGING TABLESPACE USER_DATA_TBS;
```

Table created.

테이블 T1을 생성한 후에 테이블 T1의 global secondary index를 tablespace 'USER_TEMP_TBS'에 nologging option으로 생성한다.

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )
      WITH GLOBAL SECONDARY INDEX
      NOLOGGING TABLESPACE USER_TEMP_TBS;
```

Table created.

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

- TABLESPACE 절, <physical attribute clause> 절 등의 물리적 개념
- SQL 표준은 DEFAULT 절에 연산을 사용할 수 없다.

표 16-30 SQL 표준 호환성

Feature ID	설명	지원 여부
T171	LIKE clause in table definition	X
F531	Temporary tables	X
S051	Create table of type	X
S043	Enhanced reference types	X
S081	Subtables	X
T173	Extended LIKE clause in table definition	X
T180	System-versioned tables	X
F692	Extended collation support	X
T174	Identity columns	O
T175	Generated columns	X
S071	SQL paths in function and type name resolution	X
F321	User authorization	O
T322	Extended roles	X
F762	CURRENT_CATALOG	O
F763	CURRENT_SCHEMA	O

참조

관련 내용은 다음을 참조한다.

- DROP TABLE
- ALTER TABLE
- CREATE TABLESPACE
- CREATE SCHEMA
- CREATE INDEX
- CREATE SEQUENCE
- SET CONSTRAINTS
- ALTER TABLE name ADD GLOBAL SECONDARY INDEX

16.86 CREATE TABLE AS SELECT

기능

질의 결과로부터 새로운 테이블을 생성한다.

구문

```

<table definition: AS query expression> ::=
    CREATE TABLE table_name
        [ ( column_name [, ...] ) ]
        [ <table sharding strategy> ]
        [ <table attribute clause> [, ...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
        AS <query expression> [ WITH [ NO ] DATA ]
    ;

<table sharding strategy> ::=
    <cloned strategy>
    | <hash sharding strategy>
    | <range sharding strategy>
    | <list sharding strategy>

<cloned strategy> ::=
    CLONED [ <clone placement> ]

<clone placement> ::=
    AT CLUSTER WIDE
    | AT CLUSTER GROUP group_list

<hash sharding strategy> ::=
    SHARDING BY [HASH] ( column_list )
    [ <hash shard count> ]
    [ <hash shard placement> ]

<hash shard count> ::=
    SHARD COUNT integer

<hash shard placement> ::=
    AT CLUSTER WIDE
    | AT CLUSTER GROUP group_list

<range sharding strategy> ::=

```

```

    SHARDING BY RANGE ( column_list )
    { <cluster-wide range shard placement> | <group-specific range shard placement> }
<cluster-wide range shard placement> ::=
    AT CLUSTER WIDE
    <range shard definition> [, ...]
<group-specific range shard placement> ::=
    <group-specific range shard definition> [, ...]
<group-specific range shard definition> ::=
    <range shard definition> AT CLUSTER GROUP group_name
<range shard definition> ::=
    SHARD range_name VALUES LESS THAN ( <range value clause> )
<range value clause> ::=
    <range value> [, ...]
<range value> ::=
    constant
    | MAXVALUE
<list sharding strategy> ::=
    SHARDING BY LIST ( column_name )
    { <cluster-wide list shard placement> | <group-specific list shard placement> }
<cluster-wide list shard placement> ::=
    AT CLUSTER WIDE
    <list shard definition> [, ...]
<group-specific list shard placement> ::=
    <group-specific list shard definition> [, ...]
<group-specific list shard definition> ::=
    <list shard definition> AT CLUSTER GROUP group_name
<list shard definition> ::=
    SHARD shard_name VALUES IN ( <list value clause> )
<list value clause> ::=
    <list value> [, ...]
<list value> ::=
    constant
    | NULL
    | DEFAULT
<table attribute clause> ::=
    [ <table physical attribute clause> ]
    | [ STORAGE ( <segment attr clause> [...] ) ]
<table physical attribute clause> ::=
    PCTFREE integer
    | PCTUSED integer
    | INITRANS integer

```

```

    | MAXTRANS integer
<index physical attribute clause> ::=
    PCTFREE integer
    | INITTRANS integer
    | MAXTRANS integer
<segment attr clause> ::=
    INITIAL <size_clause>
    | NEXT <size_clause>
    | MINSIZE <size_clause>
    | MAXSIZE <size_clause>
<size clause> ::=
    integer [ K | M | G | T ]
<table global secondary index clause> ::=
    WITH GLOBAL SECONDARY INDEX [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    | WITHOUT GLOBAL SECONDARY INDEX

```

사용 범위 및 접근 권한

<table definition:AS query expression> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 테이블 생성 권한
 - CREATE TABLE 구문의 접근 권한을 참조한다.
- SELECT 접근 권한
 - SELECT 구문의 접근 권한을 참조한다.

구문 규칙 및 파라미터

table_name

생성할 테이블의 이름이다.

자세한 내용은 **table_name** 구문을 참조한다.

column_name_list

테이블을 구성할 column의 이름으로써 테이블 내에서 유일한 이름이어야 하며, column의 개수는 SELECT 절의 결과 column 개수와 동일해야 한다.

명시하지 않을 경우, <query expression>의 SELECT 절의 column 이름을 사용한다.

단, SELECT절에 column이 아닌 expression (function, operation, subquery 등)이 오면 alias 또는 column name을 명시해야 한다.

Column 이름의 길이는 128 바이트보다 작아야 한다.

WITH [NO] DATA

WITH DATA가 명시된 경우, SELECT 절의 결과가 생성될 테이블에 INSERT 된다.

WITH NO DATA가 명시된 경우, SELECT 절의 결과가 생성될 테이블에 INSERT 되지 않는다.

명시하지 않을 경우, WITH DATA를 명시한 것과 동일하게 작동한다.

other syntax

이 외의 구문 규칙은 CREATE TABLE 구문의 syntax를 참조한다.

설명

CREATE TABLE AS SELECT 구문을 수행할 때 SELECT list에 NOT NULL 제약 조건이 있는 column이 명시된 경우, 새로운 테이블에도 NOT NULL 제약 조건이 생성된다. 단, 자연 가능한 NOT NULL 제약 조건인 경우, 새로운 테이블에는 NOT NULL 제약 조건을 생성하지 않는다.

그러나 명시적으로 NOT NULL 제약 조건을 생성한 것이 아니라, primary key, identity column과 같이 NOT NULL 속성을 가지고 있는 경우에는 새로운 테이블에 NOT NULL 제약 조건을 생성하지 않는다.

사용 예

다음은 CREATE TABLE AS SELECT 구문을 실행하는 예이다.

```
gSQL> CREATE TABLE recent_orders
      AS SELECT order_id, order_item, order_date
      FROM orders
      WHERE order_date >= '2015-03-03'
```

Table created.

다음은 column 이름을 명시하는 예이다.

```
gSQL> CREATE TABLE recent_orders ( order_id, order_item, order_date )
      AS SELECT order_id, order_item, order_date
      FROM orders
```



```
WHERE order_date >= '2015-03-03'
```

Table created.

다음은 SELECT list에 함수가 있는 경우의 예이다.

```
gSQL> CREATE TABLE recent_orders ( order_date, order_count )
      AS SELECT order_date, COUNT(*)
      FROM orders
      WHERE order_date >= '2015-03-03'
      GROUP BY order_date;
```

Table created.

다음은 WITH DATA가 있는 경우의 예이다.

```
gSQL>CREATE TABLE orders
(
  order_id  NUMBER
, order_item VARCHAR(128)
, order_date DATE
);
gSQL> COMMIT;
gSQL> INSERT INTO orders VALUES ( 1, 'Pen', '2010-01-01' );
gSQL> INSERT INTO orders VALUES ( 2, 'Book', '2015-03-03' );
gSQL> COMMIT;
gSQL> CREATE TABLE recent_orders
      AS SELECT order_id, order_item, order_date
      FROM orders
      WHERE order_date >= '2015-03-03'
      WITH DATA;
```

Table created.

```
gSQL> SELECT COUNT(*) FROM recent_orders;
```

```
COUNT(*)
```

```
-----
```

```
1
```

1 row selected.

다음은 WITH NO DATA가 있는 경우의 예이다.

```
gSQL>CREATE TABLE orders
(
  order_id  NUMBER
, order_item VARCHAR(128)
```

```

, order_date DATE
);
gSQL> COMMIT;
gSQL> INSERT INTO orders VALUES ( 1, 'Pen', '2010-01-01' );
gSQL> INSERT INTO orders VALUES ( 2, 'Book', '2015-03-03' );
gSQL> COMMIT;
gSQL> CREATE TABLE recent_orders
        AS SELECT order_id, order_item, order_date
        FROM orders
        WHERE order_date >= '2015-03-03'
        WITH NO DATA;
Table created.
gSQL> SELECT COUNT(*) FROM recent_orders;
COUNT(*)
-----
          0
1 row selected.

```

호환성

CREATE TABLE AS SELECT 구문은 SQL 표준을 따른다. 단, 다음은 표준에서 확장된 것이다.

- 표준에서 SELECT 절 밖의 괄호는 필수이다. 그러나 GOLDBLOCKS에서는 선택 사항이다.
- 표준에서 WITH [NO] DATA 절은 필수이다. 그러나 GOLDBLOCKS에서는 선택 사항이다.
- GOLDBLOCKS의 tablespace 개념은 표준에 없는 확장 개념이다.

표 16-31 SQL 표준 호환성

Feature ID	설명	지원 여부
T172	AS subquery clause in table definition	O

참조

관련 내용은 다음을 참조한다.

- CREATE TABLE
- SELECT

16.87 CREATE GLOBAL TEMPORARY TABLE

기능

새로운 global temporary table을 생성한다.

구문

```
<global temporary table definition> ::=
    CREATE GLOBAL TEMPORARY TABLE table_name
        ( <table element> [, ...] )
        [ <table commit action clause> ]
        [ TABLESPACE tablespace_name ]
    ;

<global temporary table definition: AS query expression> ::=
    CREATE GLOBAL TEMPORARY TABLE table_name
        [ TABLESPACE tablespace_name ]
        AS <query expression> [ WITH [ NO ] DATA ]
    ;

<table commit action clause> ::=
    ON COMMIT { PRESERVE | DELETE } ROWS
```

노트

<table element>의 정의는 <table_definition>의 정의와 동일하다. 자세한 내용은 **CREATE TABLE** 을 참조한다.

사용 범위 및 접근 권한

<global temporary table definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- 테이블 생성 권한
 - CREATE TABLE** 구문의 접근 권한을 참조한다.
- SELECT 접근 권한

- SELECT 구문의 접근 권한을 참조한다.

구문 규칙 및 파라미터

table_name

생성할 테이블의 이름이다.

자세한 내용은 `table_name` 구문을 참조한다.

other syntax

이 외의 구문 규칙은 CREATE TABLE 및 CREATE TABLE AS SELECT 구문의 syntax를 참조한다.

설명

GLOBAL TEMPORARY TABLE은 한 트랜잭션이나 세션이 실행되는 동안 유지될 데이터를 보관하는 용도로 사용하는 임시 테이블이다.

개발자가 응용 프로그램을 개발할 때 연산 중간 데이터를 잠시 저장하는 변수와 같은 용도로 사용된다.

Global temporary table의 특징은 다음과 같다.

- Global temporary table의 정의는 모든 세션에서 볼 수 있다.
- Global temporary table을 정의할 때는 물리적 공간이 할당되지 않고, 처음으로 insert 할 때 해당 세션에 종속된 실제 공간 (segment)이 할당된다.
- Global temporary table의 데이터는 insert 한 세션이나 트랜잭션에서만 볼 수 있다.
- Global temporary table의 데이터가 저장되는 tablespace는 다음과 같이 결정된다.

Tablespace 명시 여부	Table이 생성되는 tablespace
Tablespace를 명시한다.	명시된 tablespace에 생성된다.
Tablespace를 명시하지 않는다.	현재 세션 사용자의 default temporary tablespace에 생성된다.

- Global temporary table에 대한 인덱스는 대응하는 테이블과 같은 세션에 종속되며 지속기간 역시 해당 테이블과 같다.
- Global temporary table에 대한 view를 정의할 수 있다.
- Global temporary table에 대해 테이블의 cluster 관련 특성을 기술하는 <table sharding strategy> 구문이나 <table global secondary index clause>구문을 지정할 수 없다.
- Global temporary table에 대해 테이블의 물리적 특징을 기술하는 <table attribute clause>나 인덱스의 물리적 특징을 기술하는 <index attribute clause>를 지정할 수 없다.
- Global temporary table을 base table로 하여 생성되는 index들에는 인덱스의 물리적 특징을 기술하는 <inde

x attribute clause>를 지정할 수 없다.

- <table commit action clause>에 의해 transaction이 종료될 경우, 남아있는 데이터의 처리 방법을 결정할 수 있다.

Table commit action	설명
ON COMMIT PRESERVE ROWS	COMMIT 되거나 ROLLBACK 되어도 테이블에 남아있는 데이터를 그대로 유지한다.
ON COMMIT DELETE ROWS (default)	COMMIT 되거나 ROLLBACK 하는 시점에 테이블에 남아있는 데이터를 모두 삭제한다 (TRUNCATE).

- 일반 테이블에 대한 대부분의 DDL을 지원한다. (ALTER, TRUNCATE 포함)
 - CLUSTER 관련 구문 (SHARD 및 global secondary index 관련 구문 등)은 지원하지 않는다.
 - 자신의 세션이나 다른 세션에서 현재 사용 중인 global temporary table에 대한 DDL은 오류를 발생시킨다.
 - 사용 중인 모든 세션에서 TRUNCATE TABLE이나 COMMIT 등으로 사용 중인 segment들을 모두 제거한 후에 DDL이 가능해진다.
- 일반 테이블에 대한 모든 DML과 select 구문을 지원한다.
- Global temporary table에 대한 모든 변경 (DML)은 redo log를 남기지 않는다.
- Global temporary table에 대한 모든 변경 (DML)은 undo log를 남기며, TEMP_UNDO_ENABLED 프로퍼티에 따라 undo log의 위치가 결정된다.

TEMP_UNDO_ENABLED 값	설명
TRUE	Database system의 default temporary tablespace에 undo log가 기록된다.
FALSE	Database system의 undo tablespace에 undo log가 기록된다.

- Global temporary table에 대한 TRUNCATE 명령은 해당 세션의 segment만 truncate 한다.
- 세션이 종료되면 모든 segment들이 TRUNCATE 된 후에 반환된다.

사용 예

다음은 CREATE GLOBAL TEMPORARY TABLE 구문을 실행하는 예이다.

```
gSQL> CREATE GLOBAL TEMPORARY TABLE SESSION_TABLE1(
        COL1      CHAR(10)
        ,COL2      VARCHAR2(20)
        ,COL3      NUMBER(10)
    ) ON COMMIT DELETE ROWS;
Table created.
```

다음은 CREATE GLOBAL TEMPORARY TABLE ... AS SELECT 구문을 실행하는 예이다.

```
gSQL> CREATE GLOBAL TEMPORARY TABLE SESSION_TABLE2
ON COMMIT PRESERVE ROWS
AS SELECT *
FROM EMPLOYEES;
```

Table created.

호환성

CREATE GLOBAL TEMPORARY TABLE과 CREATE GLOBAL TEMPORARY TABLE AS SELECT 구문은 SQL 표준의 <table definition> 정의를 따른다. 단, 다음은 표준에서 확장된 것이다.

- 표준에서 SELECT 절 밖의 괄호는 필수이다. 그러나 GOLDBLOCKS에서는 선택 사항이다.
- 표준에서 WITH [NO] DATA 절은 필수이다. 그러나 GOLDBLOCKS에서는 선택 사항이다.
- GOLDBLOCKS의 tablespace 개념은 표준에 없는 확장 개념이다.

표 16-32 SQL 표준 호환성

Feature ID	설명	지원 여부
T171	LIKE clause in table definition	X
T172	AS subquery clause in table definition	O
F531	Temporary tables	X
S051	Create table of type	X
S043	Enhanced reference types	X
S081	Subtables	X
T173	Extended LIKE clause in table definition	X
T180	System-versioned tables	X
F692	Extended collation support	X
T174	Identity columns	O
T175	Generated columns	X
S071	SQL paths in function and type name resolution	X
F321	User authorization	O
T322	Extended roles	X
F762	CURRENT_CATALOG	O
F763	CURRENT_SCHEMA	O

참조

관련 내용은 다음을 참조한다.

- CREATE TABLE
- CREATE TABLE AS SELECT

16.88 CREATE TABLESPACE

기능

테이블스페이스를 생성한다.

구문

```
<create tablespace statement> ::=
    <memory data tablespace statement>
  | <memory temporary tablespace statement>
  ;
```

사용 범위 및 접근 권한

<create tablespace statement> 구문을 수행하려면 사용자에게 CREATE TABLESPACE ON DATABASE 권한이 있어야 한다.

구문을 수행한 사용자는 생성한 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE 권한을 갖는다.

생성한 테이블스페이스에 객체를 생성하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
- USAGE TABLESPACE ON DATABASE

구문 규칙 및 파라미터

<memory data tablespace statement>

- [MEMORY] TEMPORARY

질의 처리 과정에서 생성되는 중간 결과 등의 임시 객체나 no logging 인덱스를 저장할 메모리 임시 테이블스페이스이다.

MEMORY 예약어는 생략할 수 있다.

〈memory data tablespace clause〉

메모리 데이터의 테이블스페이스를 정의한다.

자세한 내용은 CREATE MEMORY DATA TABLESPACE 구문을 참조한다.

〈memory temporary tablespace definition〉

메모리의 임시 테이블스페이스를 정의한다.

자세한 내용은 CREATE MEMORY TEMPORARY TABLESPACE 구문을 참조한다.

설명

각 세부 구문의 설명을 참조한다.

사용 예

각 세부 구문의 사용 예를 참조한다.

호환성

SQL 표준은 tablespace에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP TABLESPACE
- ALTER TABLESPACE

16.89 CREATE MEMORY DATA TABLESPACE

기능

메모리 데이터의 테이블스페이스를 정의한다.

구문

```

<memory data tablespace statement> ::=
    CREATE [ MEMORY ] [ DATA ] TABLESPACE tablespace_name
        DATAFILE <memory datafile clause> [, ...]
        [ <data tablespace management clause> [, ...] ]
<memory datafile clause> ::=
    'filename'
        [ SIZE <size clause> | REUSE | SIZE <size clause> REUSE ]
        [ AT <domain_name> ]
<size clause> ::=
    integer [ K | M | G | T ]
<data tablespace management clause> ::=
    { ONLINE | OFFLINE }
    | EXTSIZE <size clause>
  
```

사용 범위 및 접근 권한

<memory data tablespace definition> 구문을 수행하려면 사용자에게 CREATE TABLESPACE ON DATABASE 권한이 있어야 한다.

구문을 수행한 사용자는 생성한 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE 권한을 갖는다.

생성한 테이블스페이스에 객체를 생성하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
- USAGE TABLESPACE ON DATABASE

구문 규칙 및 파라미터

[MEMORY] [DATA]

테이블, 인덱스 등 영구적인 객체를 저장할 메모리 테이블스페이스이다.
MEMORY와 DATA 예약어는 생략할 수 있다.

tablespace_name

생성할 테이블스페이스의 이름이다.
테이블스페이스 이름의 길이는 128 바이트보다 작아야 한다.

<memory datafile clause>

- 'filename'
 - 데이터를 저장 관리할 파일의 이름이다.
 - 메모리 데이터에 대한 체크포인트 이미지를 저장할 공간이다.
 - filename은 새로운 파일이거나 이미 존재하는 파일이다.
 - filename의 길이는 1024 바이트보다 작아야 한다.
- SIZE <size clause>
 - 새로운 파일일 경우 SIZE 절을 이용해 초기 크기를 지정한다.
 - 파일이 존재할 경우 에러가 발생한다.
 - 파일의 크기는 최소 1 M ~ 최대 30 G 까지 지정할 수 있다.
- REUSE
 - 이미 존재하는 파일일 경우 REUSE 절을 이용한다.
 - 파일이 존재하지 않을 경우 새로운 파일을 생성한다.
 - 새로 생성되는 파일의 크기는
 - 데이터 테이블스페이스의 경우 MEMORY_DATA_TABLESPACE_SIZE 프로퍼티에 의해 결정되고
 - 임시 테이블스페이스의 경우 MEMORY_TEMP_TABLESPACE_SIZE 프로퍼티에 의해 결정된다.
- SIZE <size clause> REUSE
 - SIZE 절과 REUSE 절을 모두 명시할 경우 filename의 존재 여부에 따라 다음과 같이 작동한다.
 - 새로운 filename일 경우에는 SIZE 절을 이용해 초기 파일 크기를 지정한다.
 - 이미 존재하는 filename일 경우에는 기존 파일을 이용하여 SIZE 절의 값으로 크기를 조정한다.

〈size clause〉

파일의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

〈domain_name〉

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

ONLINE | OFFLINE

테이블스페이스 ONLINE/ OFFLINE 여부를 설정한다.

- ONLINE은 테이블스페이스를 생성하는 즉시 사용할 수 있는 상태이다.
- OFFLINE은 사용 불가능한 상태이므로 ONLINE 상태로 변경한 후에 사용할 수 있다.

EXTSIZE 〈size clause〉

테이블스페이스의 extent 크기를 지정한다.

- Extent 크기는 바이트 단위로 기술하며, 다섯 가지 (64 K, 128 K, 256 K, 512 K, 1 M) 중 하나가 선택된다.
- 만약 extent 크기가 64 K ~ 128 K 사이의 값으로 지정되면 128 K로 설정되고, 1 M 이상으로 지정되면 1 M로 설정된다.

설명

Data tablespace는 table, index (with LOGGING option) 등의 SQL schema 객체를 저장할 물리적 공간을 제공하는 객체이다.

사용 예

다음은 memory data tablespace를 생성하는 예이다.

```
gSQL> CREATE TABLESPACE space1 DATAFILE 'test_file_1.dbf' SIZE 10M REUSE;  
Tablespace created.
```

다음은 다수의 data file로 구성된 tablespace를 생성하는 예이다.

```
gSQL> CREATE TABLESPACE space1  
        DATAFILE 'test_file_3_1.dbf' SIZE 10M REUSE,  
                'test_file_3_2.dbf' SIZE 10M REUSE;  
Tablespace created.
```

호환성

SQL 표준은 테이블스페이스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP TABLESPACE
- ALTER TABLESPACE

16.90 CREATE MEMORY TEMPORARY TABLESPACE

기능

메모리의 임시 테이블스페이스를 정의한다.

구문

```

<memory temporary tablespace statement> ::=
    CREATE [ MEMORY ] TEMPORARY TABLESPACE tablespace_name
        MEMORY <memory clause> [, ...]
        <temporary tablespace management clause>
<memory clause>
    'memory_name' { SIZE <size clause> } [ AT <domain_name> ]
<temporary tablespace management clause> ::=
    EXTSIZE <size clause>

```

사용 범위 및 접근 권한

<memory temporary tablespace definition> 구문을 수행하려면 사용자에게 CREATE TABLESPACE ON DATABASE 권한이 있어야 한다.

구문을 수행한 사용자는 생성한 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE 권한을 갖는다.

생성한 테이블스페이스에 객체를 생성하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블스페이스에 대해 CREATE OBJECT ON TABLESPACE
- USAGE TABLESPACE ON DATABASE

구문 규칙 및 파라미터

[MEMORY] TEMPORARY

질의 처리 과정에서 생성되는 중간 결과 등의 임시 객체나 no logging 인덱스를 저장할 메모리 임시 테이블스페이스이다.

MEMORY 예약어는 생략할 수 있다.

tablespace_name

생성할 테이블스페이스의 이름이다.

테이블스페이스 이름의 길이는 128 바이트보다 작아야 한다.

<memory clause>

- 'memory_name'
 - 임시 데이터를 저장할 메모리 이름이다.
 - memory_name은 해당 테이블스페이스 내에서 유일해야 한다.
 - memory_name의 길이는 1024 바이트보다 작아야 한다.
- SIZE <size clause>
 - 초기 크기를 지정한다.
 - 최소 1 M ~ 최대 30 G 까지 지정할 수 있다.

<size clause>

공유 메모리 공간의 바이트 크기를 명시한다. (단위를 기술하지 않을 경우 bytes이다.)

임시 메모리 데이터의 경우 이미지를 파일로 관리하지 않는다.

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

<domain_name>

구문을 수행할 멤버나 그룹의 이름이다.

지정하지 않은 경우에는 모든 그룹에 수행된다.

EXTSIZE <size clause>

테이블스페이스의 extent 크기를 지정한다.

- Extent 크기는 바이트 단위로 기술하며, 다섯 가지 (64 K, 128 K, 256 K, 512 K, 1 M) 중 하나가 선택된다.
- 만약 extent 크기가 64 K ~ 128 K 사이의 값으로 지정되면 128 K로 설정되고, 1 M 이상으로 지정되면 1 M로 설정된다.

설명

Temporary tablespace는 index (with NOLOGGING option) 등의 SQL schema 객체와, 질의를 처리할 때 sorting, hashing 하기 위한 중간 결과를 저장하는 물리적 공간을 제공하는 객체이다.

사용 예

다음은 temporary tablespace를 생성하는 예이다.

```
gSQL> CREATE TEMPORARY TABLESPACE temp_space1 MEMORY 'test_memory_1' SIZE 10M;
Tablespace created.
```

다음은 다수의 메모리 공간을 갖는 temporary tablespace를 생성하는 예이다.

```
gSQL> CREATE TEMPORARY TABLESPACE temp_space1
      MEMORY 'test_memory_3_1' SIZE 10M,
      'test_memory_3_2' SIZE 10M;
Tablespace created.
```

호환성

SQL 표준은 테이블스페이스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP TABLESPACE

- ALTER TABLESPACE

16.91 CREATE USER

기능

데이터베이스 사용자를 정의한다.

구문

```

<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ PROFILE { profile_name | DEFAULT | NULL } ]
    [ PASSWORD EXPIRE ]
    [ ACCOUNT { LOCK | UNLOCK } ]
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE { tablespace_name | NULL } ]
    [ <schema clause> ]
    ;
<schema clause> ::=
    WITH SCHEMA [ schema_name ]
    | WITHOUT SCHEMA
  
```

사용 범위 및 접근 권한

<user definition> 구문을 수행하려면 사용자에게 CREATE USER ON DATABASE 권한이 있어야 한다.

생성한 user_identifier 사용자는 <schema clause>로 생성한 스키마의 소유자라는 권한을 갖는다.

노트

생성된 user_identifier에는 별도의 권한이 부여되지 않는다.

user_identifier 사용자가 접속해서 SQL 구문을 수행하려면 적절한 권한을 부여받아야 한다.

구문 규칙 및 파라미터

user_identifier

생성할 user의 이름이다.

동일한 사용자 이름 (user_identifier)이나 역할 이름 (role_name)이 존재하지 않아야 한다.

user_identifier의 길이는 128 byte 보다 작아야 한다.

password

생성할 user의 password로써 암호화되어 저장된다.

password의 길이는 128 byte보다 작아야 한다.

password는 영문자로 시작해야 하고 영문자, 숫자, underscore(_), \$를 포함할 수 있다.

그 외의 특수문자를 사용하려면 double-quotation (")으로 묶어야 한다.

PROFILE { profile_name | DEFAULT | NULL }

비밀번호 관리 정책의 profile을 할당한다.

- PROFILE profile_name
 - 사용자가 생성한 profile_name을 할당한다.
- PROFILE DEFAULT
 - 기본 profile "DEFAULT"를 할당한다.
- PROFILE NULL
 - Profile을 할당하지 않는다.

PROFILE 절을 생략할 경우, PROFILE NULL과 동일하며 profile이 적용되지 않는다.

비밀번호 관리 정책에 대한 자세한 내용은 CREATE PROFILE을 참조한다.

PASSWORD EXPIRE

사용자의 비밀번호 유효기간을 만료시킨다.

사용자가 login 하기 전에 강제로 비밀번호를 변경하도록 하기 위해 사용한다.

ACCOUNT { LOCK | UNLOCK }

- ACCOUNT LOCK
 - 사용자 계정을 잠근다.
- ACCOUNT UNLOCK
 - 계정 잠금을 해제한다.

DEFAULT TABLESPACE tablespace_name

User가 생성하는 테이블, 인덱스 (LOGGING) 등의 객체가 저장될 기본 TABLESPACE를 지정한다.

DEFAULT TABLESPACE 절을 생략할 경우, DATABASE를 생성할 때 정의한 default data tablespace (MEM_DATA_TBS)가 지정된다.

TEMPORARY TABLESPACE tablespace_name

User가 생성하는 임시 테이블, 인덱스 (NO LOGGING), 질의 처리 과정에서 생성되는 중간 결과들을 저장할 TABLESPACE를 지정한다.

TEMPORARY TABLESPACE 절을 생략할 경우, DATABASE를 생성할 때 정의한 default temporary tablespace (MEM_TEMP_TBS)가 지정된다.

INDEX TABLESPACE { tablespace_name | NULL }

User가 생성하는 인덱스 객체가 저장되는 기본 TABLESPACE를 지정한다.

- INDEX TABLESPACE tablespace_name 지정
 - Data tablespace를 지정할 경우, LOGGING 인덱스여야 한다.
 - Temporary tablespace를 지정할 경우, NOLOGGING 인덱스여야 한다.
- INDEX TABLESPACE NULL
 - Index tablespace를 지정하지 않는다.

INDEX TABLESPACE 절을 생략할 경우, INDEX TABLESPACE NULL 이다.

<schema clause>

User가 기본적으로 사용할 스키마를 생성한다.

Database 내에 동일한 스키마 이름이 존재하지 않아야 한다.

- WITH SCHEMA [schema_name]
 - schema_name을 부여하지 않을 경우, user_identifier와 동일한 이름의 스키마가 생성된다.
 - User의 SCHEMA PATH는 다음과 같이 설정된다.
 - schema_name, PUBLIC
- WITHOUT SCHEMA
 - User가 소유할 스키마를 생성하지 않는다.
 - User의 SCHEMA PATH는 PUBLIC으로 설정된다.

<schema clause>를 명시하지 않을 경우, 기본값은 WITH SCHEMA이고 user_identifier와 동일한 이름의 스키마가 생성된다.

사용자가 소유할 스키마는 **CREATE SCHEMA** 구문을 사용하여 추가로 생성할 수 있다.

설명

User는 권한의 집합으로 구성된 authorization 객체이다.

최초로 <user definition> 구문을 수행할 때 어떠한 권한도 부여받지 않은 user가 생성되고 다음과 같이 적절한 권한을 부여해야 한다.

GOLDILOCKS에서 user와 schema의 관계는 1 : N 이다.

즉, user가 소유한 schema가 존재하지 않을 수도 있고 다수의 schema를 소유할 수도 있다.

SQL 표준은 user, schema, database 등의 non-schema 객체들의 관계에 대해 명확히 정의하지 않고 있다. 반면, 각 DBMS 들은 다음과 같이 non-schema 객체 간의 관계를 상이하게 정의하고 있다.

노트

DBMS별 user와 schema 관계

- Oracle
 - User : schema = 1 : 1의 관계이다.
- DB2
 - OS user와 동일하다.
 - User를 생성하고 삭제하는 별도의 SQL 구문이 없다.
- Postgres
 - User : schema = 1 : N의 관계이다.
- MySQL
 - Database : schema = 1 : 1의 관계이다.
 - User는 database (schema)의 하위 객체이다.

사용 예

사용자를 생성하고 생성한 사용자가 객체를 생성하고 데이터를 조작하도록 하려면 다음과 같이 권한을 부여해야 한다.

다음은 사용자를 생성하고 그 사용자에게 권한을 부여하는 예이다.

- Create a user.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password
        DEFAULT TABLESPACE mem_data_tbs
        TEMPORARY TABLESPACE mem_temp_tbs
        INDEX TABLESPACE NULL;
```

User created.

```
gSQL> COMMIT;
```

Commit complete.

- Grant database privileges.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO u1;
```

Grant succeeded.

```
COMMIT;
```

Commit complete.

- Grant schema privileges.

```
GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT
        ON SCHEMA u1 TO u1;
```

Grant succeeded.

```
COMMIT;
```

Commit complete.

- Grant tablespace privileges.

```
GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
```

Grant succeeded.

```
GRANT CREATE OBJECT ON TABLESPACE mem_temp_tbs TO u1;
```

Grant succeeded.

```
COMMIT;
```

Commit complete.

다음은 사용자가 객체를 생성하는 예이다.

- It needs CREATE SESSION ON DATABASE.

```
gSQL> \connect u1 u1_password
```

- It needs CREATE TABLE ON SCHEMA u1.
- It needs CREATE OBJECT ON TABLESPACE mem_data_tbs.

```
gSQL> CREATE TABLE u1.t1 ( c1 INTEGER, c2 INTEGER ) TABLESPACE mem_data_tbs;  
Table created.  
gSQL> COMMIT;
```

- It needs CREATE INDEX ON SCHEMA u1.
- It needs CREATE OBJECT ON TABLESPACE mem_temp_tbs.

```
gSQL> CREATE INDEX u1.idx ON t1 (c2) TABLESPACE mem_temp_tbs;  
Index created.  
gSQL> COMMIT;
```

- It needs ADD CONSTRAINT ON SCHEMA u1.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT u1.t1_pk PRIMARY KEY (c1) ;  
Table altered.  
gSQL> COMMIT;
```

- It needs CREATE SEQUENCE ON SCHEMA u1.

```
gSQL> CREATE SEQUENCE u1.seq;  
Sequence created.  
gSQL> COMMIT;  
gSQL> INSERT INTO u1.t1 VALUES ( u1.seq.NEXTVAL, u1.seq.NEXTVAL );  
1 row created  
gSQL> COMMIT;
```

호환성

SQL 표준에서는 user 개념은 다루고 있지만 user 생성 및 삭제와 관련된 SQL 구문은 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- DROP USER
- ALTER USER
- CREATE SCHEMA

16.92 CREATE VIEW

기능

View를 정의한다.

구문

```
<view definition> ::=
    CREATE [ OR REPLACE ] [ FORCE | NO FORCE ]
        VIEW view_name [ ( column_name [, ...] ) ]
        AS <query expression>
    ;
```

사용 범위 및 접근 권한

<view definition> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- View를 생성하기 위해 다음 권한 중 하나가 있어야 한다.
 - View가 속한 스키마에 대해 (CREATE VIEW 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY VIEW ON DATABASE
- OR REPLACE 절을 사용할 때 이미 view가 존재할 경우, 기존 view를 제거할 수 있는 다음 권한 중 하나가 필요하다.
 - 해당 view의 소유자
 - 해당 view에 대해 CONTROL TABLE ON TABLE
 - View가 속한 스키마에 대해 (DROP VIEW 또는 CONTROL SCHEMA) ON SCHEMA
 - DROP ANY VIEW ON DATABASE
- <query expression> 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE
- 생성한 view의 소유자는 다음과 같이 결정된다.

- View가 속한 스키마의 소유자
- View가 속한 스키마가 PUBLIC인 경우, 구문을 수행한 사용자
- View의 소유자는 생성한 view에 대해 다음과 같은 권한을 갖는다.
 - SELECT ON TABLE WITH GRANT OPTION
 - INSERT ON TABLE WITH GRANT OPTION
 - UPDATE ON TABLE WITH GRANT OPTION
 - DELETE ON TABLE WITH GRANT OPTION
 - TRIGGER ON TABLE
 - LOCK ON TABLE WITH GRANT OPTION
 - ALTER ON TABLE WITH GRANT OPTION

구문 규칙 및 파라미터

[OR REPLACE]

이미 존재하는 view가 있을 경우, 기존의 view를 대체한다.

[FORCE | NO FORCE]

- FORCE
 - <query expression>의 유효성 여부에 관계없이 view를 생성한다.
- NO FORCE
 - <query expression>이 유효할 경우 view를 생성한다.
- 기본값은 NO FORCE 이다

view_name

생성할 view의 이름이며, 스키마 내에서 유일한 이름이어야 한다.

schema_name.view_name과 같이 view가 속할 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

View 이름의 길이는 128 바이트보다 작아야 한다.

[(column_name [, ...])]

View를 구성할 column의 이름을 정의한다.

각 column의 이름은 view 내에서 고유한 이름이어야 한다.

Column의 개수는 SELECT 절의 결과 column 개수와 동일해야 한다.

Column 이름의 리스트를 생략할 경우, <query expression>의 SELECT 절의 column 이름을 사용한다.

AS <query expression>

View를 생성하는 SELECT 절의이다.

<query expression>에는 다음과 같은 변수를 포함할 수 없다.

- Host parameter
- SQL parameter
- Dynamic parameter
- Embedded variable
- SEQUENCE 객체

설명

View는 질의에 이름을 부여한 객체로서 table과 유사한 방식으로 사용할 수 있다.

View를 포함하는 질의를 수행할 때 해당 view는 view 정의에 포함된 질의로 해석된다. 예를 들어, 다음과 같이 view가 참조하는 table이 변경되면 view 정의에 포함된 asterisk (*) 등은 변경된 table 정보에 따라 자동으로 재해석된다.

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1;
gSQL> COMMIT;
gSQL> SELECT * FROM v1;
ID NAME
-- -----
1 leekmo
2 mkkim
3 egonspace
3 rows selected.
gSQL> ALTER TABLE t1 ADD COLUMN ( dept_id INTEGER, addr VARCHAR(1024) );
gSQL> COMMIT;
gSQL> select * from v1;
ID NAME      DEPT_ID ADDR
-- -----
1 leekmo      null null
2 mkkim       null null
3 egonspace   null null
3 rows selected.
```

FORCE 옵션을 사용하여 질의에 에러가 존재하는 상태로 view를 생성하거나, view가 참조하는 테이블이나 view가 변경 또는 제거되었을 경우 해당 view에 영향을 미친다.

이런 정보는 INFORMATION_SCHEMA.VIEWS 정보로부터 조회할 수 있다.

- IS_COMPILED column
 - TRUE: View가 정상적으로 생성되었다.
 - FALSE: FORCE 옵션을 사용하여 에러가 존재하는 상태에서 view가 생성되었다.
- IS_AFFECTED column
 - TRUE: View가 참조하는 테이블 또는 view가 변경되었다.
 - FALSE: View를 생성하고 COMPILE한 후에 view가 참조하는 테이블이나 view가 변경되지 않았다.

View의 최대 생성 개수와 view 내부에 생성 가능한 최대 column 개수에는 제한이 없으므로 저장 공간에 문제가 없는 한 계속 생성할 수 있다.

사용 예

다음은 view를 생성하는 예이다.

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1 WHERE dept_id = 101;
View created.
```

다음은 view를 정의하면서 column 이름을 정의하는 예이다.

```
gSQL> CREATE VIEW v1 ( v_id, v_name )
      AS SELECT id, name FROM t1 WHERE dept_id = 101;
View created.
```

다음은 기존 view가 있을 경우 REPLACE 옵션을 사용하여 이를 제거하고 새로 view를 생성하는 예이다.

```
gSQL> CREATE OR REPLACE VIEW v1(id, name)
      AS SELECT id, name FROM t1;
View created.
```

다음은 view가 참조하는 객체가 존재하지 않더라도 FORCE 옵션을 사용하여 해당 view를 강제로 생성하는 예이다.

```
gSQL> CREATE FORCE VIEW v1
      AS SELECT * FROM t1 WHERE dept_id = 101;
ERR-01000(16243): Warning: View created with compilation errors
ERR-42000(16040): table or view does not exist :
      AS SELECT * FROM t1 WHERE dept_id = 101
      *
ERROR at line 2:
View created.
```

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

- [OR REPLACE] 절
- [FORCE | NO FORCE] 절

표 16-33 SQL 표준 호환성

Feature ID	설명	지원 여부
T131	Recursive query	X
F751	View CHECK enhancements	X
S043	Enhanced reference types	X
T111	Updatable joins, unions, and columns	X
F852	Top-level <order by clause> in views	O
F864	Top-level <result offset clause> in views	O
F859	Top-level <fetch first clause> in views	O
S081	Subtables	X

참조

관련 내용은 다음을 참조한다.

- DROP VIEW
- ALTER VIEW
- SELECT

16.93 DECLARE cursor_name

기능

커서를 선언한다.

구문

```

<declare cursor> ::=
    DECLARE cursor_name <cursor properties> { FOR | IS } <cursor specification>
    ;
<cursor properties> ::=
    [ <cursor sensitivity> ] [ <cursor scrollability> ] CURSOR [ <cursor holdability> ]
    | [ <odbc cursor type> ] CURSOR [ <cursor holdability> ]
<cursor sensitivity> ::=
    INSENSITIVE
    | SENSITIVE
    | ASENSITIVE
<cursor scrollability> ::=
    NO SCROLL
    | SCROLL
<cursor holdability> ::=
    WITH HOLD
    | WITHOUT HOLD
<odbc cursor type> ::=
    STATIC
    | KEYSET
<cursor specification> ::=
    statement_name
    | <cursor query> [ <updatability clause> ]
<cursor query> ::=
    <select statement>
    | <insert returning query statement>
    | <update returning query statement>
    | <delete returning query statement>
<updatability clause> ::=
    FOR READ ONLY

```

```

| FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
| WAIT
| WAIT second
| NOWAIT

```

사용 범위 및 접근 권한

statement_name을 사용한 동적 커서 (dynamic cursor)는 embedded SQL에서 사용할 수 있다.

<cursor query>의 유형에 따라 적절한 접근 권한을 가져야 한다.

접근 권한에 대한 자세한 내용은 다음을 참조한다.

- SELECT 구문의 접근 권한
- SELECT .. FOR UPDATE 구문의 접근 권한
- INSERT INTO name RETURNING 구문의 접근 권한
- UPDATE name RETURNING 구문의 접근 권한
- DELETE FROM name RETURNING 구문의 접근 권한

구문 규칙 및 파라미터

cursor_name

선언할 커서의 이름이다.

하나의 session 내에서 고유한 이름이어야 한다.

커서 이름의 길이는 128 바이트보다 작아야 한다.

{ FOR | IS }

SQL 표준에서는 구문 키워드로 FOR나 IS 중에 하나를 사용한다.

<cursor properties>

커서의 속성을 정의한다.

- <cursor sensitivity>를 명시하지 않은 경우, 기본값은 INSENSITIVE 이다.
- <cursor scrollability>를 명시하지 않은 경우, 기본값은 NO SCROLL 이다.
- <cursor holdability>를 명시하지 않은 경우, <cursor updatability>가 기본값을 결정한다.

updatable query

Cursor 속성 중에 SENSITIVE나 FOR UPDATE를 사용하려면 cursor의 query가 base table의 row 변화를 식별하거나 row에 lock을 획득할 수 있는 updatable query 여야 한다.

updatable query는 다음 조건을 모두 만족해야 한다.

- 최상위 query에 DISTINCT가 존재하지 않아야 한다.
 - (X) SELECT DISTINCT * FROM t1;
- 최상위 query에 GROUP BY, HAVING, aggregation function이 존재하지 않아야 한다.
 - (X) SELECT MAX(c1) FROM t1;
- Returning query가 존재하지 않아야 한다.
 - (X) DELETE FROM t1 RETURNING c1;
- Set 연산자가 존재하지 않아야 한다.
 - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- FROM 절에 나열된 table 들에 하나 이상의 updatable column이 존재해야 한다.
 - Join에 포함되는 테이블 중 cross join에 해당하지 않는 테이블의 column은 updatable column이 아니다.
 - FULL OUTER JOIN은 cross join이 아니다.
 - NATURAL JOIN은 cross join이 아니다.
 - INNER JOIN에 USING 구문이 사용되면 cross join이 아니다.
 - 다음과 같은 table들의 column은 updatable column이 아니다.
 - Dictionary table, fixed table, performance view
 - View의 column은 updatable table이 아니다.

<cursor sensitivity>

커서를 운용할 때 query 결과에 영향을 미치는 다음과 같은 데이터 변화를 볼 수 있는지 여부를 설정한다.

- INSENSITIVE
 - 커서 운용 중에 변경된 데이터 내용을 감지할 수 없다.
- SENSITIVE
 - <cursor query>가 updatable query여야 한다.
 - 커서와 동일한 트랜잭션에서 변경 (UPDATE), 삭제 (DELETE)된 데이터를 감지한다.
 - 다른 트랜잭션의 COMMIT을 통해 변경 (UPDATE), 삭제 (DELETE)된 데이터를 감지한다.
- ASENSITIVE
 - <cursor query>의 유형에 따라 INSENSITIVE인지 SENSITIVE인지가 결정된다.
 - updatable query인 경우, SENSITIVE이다.
 - updatable query가 아닌 경우, INSENSITIVE이다.
- 명시하지 않을 경우, 기본값은 INSENSITIVE이다.

〈cursor scrollability〉

Cursor의 result set을 순차적 또는 비순차적으로 fetch 할 수 있는지 여부를 명시한다.

- NO SCROLL
 - 순차적 FETCH (FETCH NEXT)만 가능하다.
- SCROLL
 - 비순차적 FETCH가 가능하다.
- 명시하지 않을 경우, 기본값은 NO SCROLL이다.

〈cursor holdability〉

Cursor를 OPEN하고 트랜잭션을 commit 한 후에도 cursor가 유지되는지 여부를 설정한다.

- WITH HOLD
 - 트랜잭션을 COMMIT 해도 cursor가 유지된다.
 - FOR UPDATE 구문과 함께 사용할 수 없다.
 - **INSERT INTO name RETURNING** 구문과 함께 사용할 수 없다.
 - **UPDATE name RETURNING** 구문과 함께 사용할 수 없다.
 - **DELETE FROM name RETURNING** 구문과 함께 사용할 수 없다.
- WITHOUT HOLD
 - 트랜잭션을 COMMIT/ ROLLBACK하면 cursor를 닫는다.
- Rollback과 cursor
 - 트랜잭션을 rollback 할 경우, 트랜잭션에 포함된 cursor를 닫는다.
 - Savepoint까지 rollback하면 savepoint 이후에 생성된 cursor를 닫는다.
- 명시하지 않을 경우, 〈cursor holdability〉의 기본값은 〈cursor updatability〉에 따라 결정된다.
 - FOR READ ONLY이거나 〈cursor updatability〉를 명시하지 않은 경우, 기본값은 WITH HOLD 이다.
 - FOR UPDATE 구문과 함께 사용할 경우, 기본값은 WITHOUT HOLD 이다.

〈odbc cursor type〉

ODBC 표준의 cursor 유형으로 SCROLL 속성을 갖는다.

- STATIC CURSOR
 - SQL 표준의 INSENSITIVE SCROLL과 동일하다.
 - 비순차적 FETCH가 가능하다.
 - ODBC 표준의 static scroll cursor이다.
- KEYSET CURSOR
 - SQL 표준의 ASENSITIVE SCROLL과 동일하다.
 - ODBC 표준의 keyset-driven scroll cursor이다.

- Sensitivity 속성은 다음과 같은 특성에 따라 결정된다.

표 16-34 FOR [UPDATE / READ ONLY] 구문과 query 유형에 따른 sensitivity

Updatability	Query 유형	Sensitivity
FOR UPDATE	Updatable query	SENSITIVE
FOR UPDATE	Non-updatable query	Query error
FOR READ ONLY	Any query	INSENSITIVE
N/A	Updatable query	SENSITIVE
N/A	Non-updatable query	INSENSITIVE

<cursor specification>

Cursor의 대상이 되는 query를 정의한다.

statement_name을 사용할 경우, query가 정해지지 않은 동적 커서 (dynamic cursor)가 선언되고, <cursor query>를 사용할 경우, query가 정해진 고정 커서 (standing cursor)가 선언된다.

statement_name

Cursor가 참조할 statement_name이며 embedded SQL에서 사용할 수 있다.

statement_name은 <declare cursor> 구문을 수행하기 전에 존재해야 하며, statement_name이 참조하는 SQL 문장은 PREPARE statement_name 구문이 준비한 query여야 한다.

Query가 아닐 경우 OPEN cursor_name 구문을 수행할 때 error가 발생한다.

<cursor query>

Cursor에서 사용할 수 있는 query 유형은 다음 각 구문을 참조한다.

- SELECT
- SELECT .. FOR UPDATE
- INSERT INTO name RETURNING
- UPDATE name RETURNING
- DELETE FROM name RETURNING

<updatability clause>

Cursor를 이용해 row를 변경할지 여부를 명시한다.

- FOR READ ONLY
 - 읽기 전용 커서를 선언한다.
- FOR UPDATE

- 쓰기 가능한 커서를 선언한다.
- 커서를 open 할 때 해당 트랜잭션이 종료될 때까지, 다른 트랜잭션이 변경할 수 없도록 해당 row 들에 대한 x lock을 획득한다.
- WITH HOLD 구문과 함께 사용할 수 없다.
- <cursor query>가 updatable query여야 한다.
- 명시하지 않을 경우, 기본값은 FOR READ ONLY이다.

FOR UPDATE OF ...

커서를 OPEN 할 때 lock 획득과 관련된 column을 나열한다.

- FOR UPDATE OF 구문에 나열된 column은
 - <select statement>의 FROM 절에 나열된 table의 갱신 가능한 column이어야 한다.
 - 나열된 column의 table에 대한 lock을 획득한다.
- FOR UPDATE만 사용하는 경우에는
 - <select statement>의 FROM 절에 나열된 table들의 모든 갱신 가능한 column을 나열한 것과 동일한 의미이다.
 - 모든 column의 table에 대한 lock을 획득한다.

<lock wait mode>

FOR UPDATE 구문과 함께 사용하며, lock 획득 방법을 지정한다.

- WAIT
 - 커서를 OPEN 할 때 질의 결과의 모든 row들에 대한 lock을 획득한다.
 - Lock을 획득할 수 있을 때까지 대기한다.
- WAIT second
 - 커서를 OPEN 할 때 질의 결과의 모든 row 들에 대한 lock을 획득한다.
 - 지정된 시간 안에 lock을 획득하지 못하면 에러가 발생한다.
 - 초 단위이며 0 ~ 10000000000 까지의 값을 사용할 수 있다.
- NOWAIT
 - 커서를 OPEN 할 때 질의 결과의 모든 row들에 대한 lock을 획득한다.
 - 즉시 lock을 획득하지 못하면 에러가 발생한다.
- 명시하지 않을 경우, 기본값은 WAIT 이다.

설명

Query에 대한 속성을 제어할 때 DECLARE CURSOR 구문과 OPEN, FETCH, CLOSE 구문을 사용할 경우, 서버의 커서를 제어하기 때문에 ODBC statement나 JDBC statement를 이용하여 cursor를 사용하는 경우보다 성능상 부하가 걸린다.

Query를 수행하기 전에 ODBC statement와 JDBC statement를 사용하여 cursor 속성을 제어할 수 있으며, DECLARE CURSOR 구문을 통한 SQL cursor의 속성 제어 방법과 이에 대응하는 ODBC 표준과 JDBC 표준의 cursor 속성 제어 방법은 다음과 같다.

표 16-35 ODBC/ JDBC의 커서 속성 제어

Property 분류	GOLDILOCKS cursor property	ODBC 표준의 cursor 속성 설정	JDBC 표준의 cursor 속성 설정
Sensitivity	INSENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_SENSITIVITY, SQL_INSENSITIVE, len)	설정할 수 없음
	SENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_SENSITIVITY, SQL_SENSITIVE, len)	설정할 수 없음
	ASENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_SENSITIVITY, SQL_UNSPECIFIED, len)	설정할 수 없음
Scrollability	NO SCROLL	SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_SCROLLABLE, SQL_NONSCROLLABLE, len)	설정할 수 없음
	SCROLL	SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_SCROLLABLE, SQL_SCROLLABLE, len)	설정할 수 없음
Holdability	WITHOUT HOLD	설정할 수 없음	java.sql.Connection::prepareStatement(query, type, conc, ResultSet.CLOSE_CURSORS_AT_COMMIT)
	WITH HOLD	설정할 수 없음	java.sql.Connection::prepareStatement(query, type, conc, ResultSet.HOLD_CURSORS_OVER_COMMIT)

ODBC cursor type에 대응되는 SQL cursor 선언은 다음과 같다.

표 16-36 ODBC 커서 type에 대응되는 SQL 커서 선언

ODBC cursor type	SQL cursor 선언
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_FORWARD_ONLY, len)	NO SCROLL CURSOR
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_STATIC, len)	STATIC CURSOR
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN, len)	KEYSET CURSOR

JDBC cursor type에 대응되는 SQL cursor 선언은 다음과 같다.

표 16-37 JDBC 커서 type에 대응되는 SQL 커서 선언

JDBC cursor type	SQL cursor 선언
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_FORWARD_ONLY, c onc, hold)	INSENSITIVE NO SCROLL CUR SOR
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIV E, conc, hold)	INSENSITIVE SCROLL CURSO R
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_SCROLL_SENSITIVE, conc, hold)	SENSITIVE SCROLL CURSOR

사용 예

다음은 interactive SQL (gsq)을 사용하여 cursor를 선언하고 사용하는 예이다.

```
gsqSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gsqSQL> OPEN cur1;
Cursor is open.
gsqSQL> \var v_id    INTEGER
gsqSQL> \var v_data VARCHAR(128)
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
1 data_1
1 row fetched.
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
2 data_2
1 row fetched.
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
3 data_3
1 row fetched.
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
4 data_4
1 row fetched.
```

```

gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.

```

다음은 KEYSET 커서를 선언하고 순차적으로 검색한 후, UPDATE, DELETE 구문에 대한 transaction이 완료된 후에 이를 역방향으로 검색하는 예이다.

```

gSQL> DECLARE cur_keyset KEYSET CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur_keyset;
Cursor is open.
gSQL> \var v_id    INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
----
1 data_1
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
----
2 data_2
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
----
3 data_3
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
----
4 data_4
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA

```

```

-----
      5 data_5
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
no rows fetched.
gSQL> UPDATE t1 SET data = 'new data_2' WHERE id = 2;
1 row updated.
gSQL> COMMIT;
Commit complete.
gSQL> DELETE FROM t1 WHERE id = 4;
1 row deleted.
gSQL> COMMIT;
Commit complete.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
-----
      5 data_5
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
-----
      3 data_3
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
-----
      2 new data_2
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
-----
      1 data_1
1 row fetched.
gSQL> CLOSE cur_keyset;
Cursor closed.

```

다음은 SCROLL 커서를 선언하고 fetch orientation을 통해 커서를 사용하는 예이다.

```
gSQL> DECLARE cur_scroll SCROLL CURSOR FOR SELECT id, data FROM t1;
```

Cursor declared.

```
gSQL> OPEN cur_scroll;
```

Cursor is open.

```
gSQL> \var v_id INTEGER
```

```
gSQL> \var v_data VARCHAR(128)
```

```
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

5 data_5

1 row fetched.

```
gSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

4 data_4

1 row fetched.

```
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

1 data_1

1 row fetched.

```
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

3 data_3

1 row fetched.

```
gSQL> FETCH RELATIVE -1 cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

2 data_2

1 row fetched.

```
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
```

V_ID V_DATA

3 data_3

1 row fetched.

```
gSQL> CLOSE cur_scroll;
```

Cursor closed.

호환성

〈declare cursor〉 구문은 SQL 표준과 비교하여 다음과 같은 차이가 있다.

- SQL 표준의 〈cursor sensitivity〉 기본값은 **ASENSITIVE**이지만, GOLDILOCKS의 기본값은 **INSENSITIVE**이다.
- SQL 표준에서는 다음과 같은 〈odbc cursor type〉을 다루지 않고 있다.
 - **STATIC CURSOR**
 - **KEYSET CURSOR**
- SQL 표준의 〈cursor holdability〉 기본값은 **WITHOUT HOLD**이지만, GOLDILOCKS의 기본값은 〈cursor updatability〉에 따라 다르다.
- SQL 표준에서는 〈cursor query〉로 〈select statement〉만 사용할 수 있지만, GOLDILOCKS는 다음과 같은 **returning query**를 사용할 수 있다.
 - **INSERT INTO name RETURNING**
 - **UPDATE name RETURNING**
 - **DELETE FROM name RETURNING**
- SQL 표준의 〈cursor updatability〉 기본값은 〈select statement〉에 따라 결정되지만, GOLDILOCKS의 기본값은 **FOR READ ONLY**이다.
- SQL 표준에는 〈lock wait mode〉 구문이 존재하지 않는다.

표 16-38 SQL 표준 호환성

Feature ID	설명	지원 여부
F831	Full cursor update	O
T231	Sensitive cursors	O
F791	Insensitive cursors	O
F431	Read-only scrollable cursors	O
T471	Result sets return value	X
T551	Optional key words for default syntax	O
T111	Updatable joins, unions, and columns	X
B031	Basic dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- **OPEN cursor_name**
- **FETCH cursor_name**
- **CLOSE cursor_name**
- **PREPARE statement_name**
- **SELECT**

- SELECT .. FOR UPDATE
- INSERT INTO name RETURNING
- UPDATE name RETURNING
- DELETE FROM name RETURNING

16.94 DELETE FROM

기능

테이블의 row들을 삭제한다.

구문

```

<delete statement: searched> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

```

사용 범위 및 접근 권한

<delete statement: searched> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 테이블에 대해 (DELETE 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (DELETE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- DELETE ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 삭제할 대상 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

[AS alias_name]

table_name의 alias 이다.

WHERE <search condition>

WHERE 조건을 만족하는 row를 삭제한다.

WHERE 조건을 명시하지 않은 경우, 모든 row를 삭제한다.

WHERE 조건에 대한 자세한 내용은 SELECT 구문의 **where clause**를 참조한다.

<result offset clause>

질의 결과 중 건너될 row의 개수를 명시한다.

자세한 내용은 SELECT 구문의 **offset limit clause**를 참조한다.

<fetch limit clause>

Fetch 할 row의 개수를 명시하는 구문으로써 다음 두 가지 방법이 사용된다.

- <fetch first clause>
 - Fetch 할 row의 개수를 명시한다.
 - 자세한 내용은 SELECT 구문의 <fetch first clause>를 참조한다.
- <limit clause>
 - Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너될 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
 - 자세한 내용은 SELECT 구문의 <limit clause>를 참조한다.

설명

DELETE 관련 구문들의 차이점

- **DELETE FROM**
 - 조건에 부합하는 다수의 row를 삭제한다.
 - 예: DELETE FROM t1 WHERE c1 = 0;
- **DELETE FROM name WHERE CURRENT OF cursor_name**
 - Cursor가 현재 가리키는 row를 삭제한다.
 - 예: DELETE FROM t1 WHERE CURRENT OF cursor;
- **DELETE FROM name RETURNING**
 - 조건을 만족하는 다수의 row를 삭제하며, SELECT 구문과 동일한 방식(SQLFetch() 등의 API)으로 삭제한 row들을 검색할 수 있다.
 - 예: DELETE FROM t1 WHERE c1 = 0 RETURNING c2;
- **DELETE FROM name RETURNING .. INTO**
 - 한 건 이하의 row를 삭제할 수 있으며, 삭제한 row가 한 건일 경우 RETURNING INTO 절의 호스트 변수에 값을 얻어온다.
 - 예: DELETE FROM t1 WHERE c1 = 0 RETURNING c2 INTO :v1;

사용 예

다음은 DELETE 구문의 예이다.

```
gSQL> DELETE FROM t1 WHERE id > 3;
2 rows deleted.
```

다음은 <result offset clause>와 <fetch first clause>를 이용하여 조건을 만족하는 row들 중 일부 row들을 (두 건) 건너뛰고 일부 row들만 (두 건) 삭제하는 예이다.

```
gSQL> DELETE FROM t1 OFFSET 2 FETCH 2;
2 rows deleted.
gSQL> SELECT * FROM t1 ORDER BY 1;
ID DATA
-- -----
1 data_1
2 data_2
5 data_5
3 rows selected.
```

호환성

SQL 표준은 DELETE 구문에서 다음 절을 정의하지 않고 있다.

- <result offset clause>
- <fetch limit clause>

표 16-39 SQL 표준 호환성

Feature ID	설명	지원 여부
F781	Self-referencing operations	X
T111	Updatable joins, unions, and columns	X

참조

관련 내용은 다음을 참조한다.

- DELETE FROM name WHERE CURRENT OF cursor_name
- DELETE FROM name RETURNING
- DELETE FROM name RETURNING .. INTO
- SELECT

16.95 DELETE FROM name RETURNING

기능

테이블의 row들을 삭제하고, 삭제한 row들을 검색한다.

구문

```

<delete returning query statement> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning clause>
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
<returning clause> ::=
    { RETURN | RETURNING } { * | { <value expression> [ [AS] alias_name ] } [, ...] }

```

사용 범위 및 접근 권한

<delete returning query statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- DELETE 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - 테이블에 대해 (DELETE 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (DELETE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - DELETE ANY TABLE ON DATABASE

- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 삭제할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias 이다.

WHERE <search condition>

WHERE 조건을 만족하는 row를 삭제한다.
자세한 내용은 DELETE FROM 구문을 참조한다.

<result offset clause>

질의 결과 중에 건너될 row의 개수를 명시한다.
자세한 내용은 DELETE FROM 구문을 참조한다.

<fetch first clause>

Fetch 할 row의 개수를 명시한다.
자세한 내용은 DELETE FROM 구문을 참조한다.

<limit clause>

Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너될 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
자세한 내용은 DELETE FROM 구문을 참조한다.

〈returning clause〉

삭제된 row들을 결과 집합으로 하고, 이들 중에서 검색할 column을 기술한다.

- RETURNING 절은 DELETE 구문으로 삭제된 row들을 result set으로 하는 결과를 반환한다.
- 〈value expression〉
 - SELECT 구문의 〈select list〉와 동일하지만 aggregation 등을 사용할 수 없다.
- [[AS] alias_name]
 - AS 절을 이용해 value expression의 이름을 지정할 수 있다.

RETURN과 RETURNING은 동일한 의미의 키워드이다.

설명

자세한 내용은 DELETE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 조건을 만족하는 row들을 삭제하고 삭제한 row들을 검색하는 예이다.

```
gSQL> DELETE FROM t1 WHERE id > 3 RETURNING *;
ID DATA
-- -----
4 data_4
5 data_5
2 rows deleted.
```

다음은 RETURNING 절에 연산을 사용하여 삭제한 row들의 정보를 조회하는 예이다.

```
gSQL> DELETE FROM t1
      WHERE id > 3
      RETURNING 'ID: ' || id || ', DATA: ' || data AS id_data;
ID_DATA
-----
ID: 4, DATA: data_4
ID: 5, DATA: data_5
2 rows deleted.
```


호환성

SQL 표준에는 <delete returning query statement> 구문이 존재하지 않는다.

참조

관련 내용은 다음을 참조한다.

- DELETE FROM
- DELETE FROM name WHERE CURRENT OF cursor_name
- DELETE FROM name RETURNING .. INTO
- SELECT

16.96 DELETE FROM name RETURNING .. INTO

기능

테이블에서 row 하나를 삭제하고, 삭제한 row의 값을 호스트 변수에 얻어온다.

구문

```
<delete returning query statement> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning into clause>
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
<returning into clause> ::=
    { RETURN | RETURNING } { * | { <value expression> [ [AS] alias_name ] } [, ...] } INTO
    variable_name [, ...]
```

사용 범위 및 접근 권한

<delete returning into statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- DELETE 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - 테이블에 대해 (DELETE 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (DELETE TABLE 또는 CONTROL SCHEMA) ON SCHEMA

- DELETE ANY TABLE ON DATABASE
- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 삭제할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias이다.

WHERE <search condition>

WHERE 조건을 만족하는 row를 삭제한다.

자세한 내용은 DELETE FROM 구문을 참조한다.

<result offset clause>

질의 결과 중 건너뛴 row의 개수를 명시한다.

자세한 내용은 DELETE FROM 구문을 참조한다.

<fetch first clause>

Fetch 할 row의 개수를 명시한다.

자세한 내용은 DELETE FROM 구문을 참조한다.

<limit clause>

Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너뛴 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.

자세한 내용은 DELETE FROM 구문을 참조한다.

〈returning into clause〉

- RETURNING .. AS ..
 - DELETE FROM name RETURNING 구문의 returning clause를 참조한다.
- INTO variable_name [, ...]
 - INTO 절에 기술된 변수의 개수는 RETURNING 절에 기술된 expression의 개수와 동일해야 한다.

설명

삭제할 row가 하나 이하여야 한다.

둘 이상의 row가 삭제되면 에러가 발생한다.

자세한 내용은 DELETE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 interactive SQL (gsq)에서 row를 삭제하고 삭제된 row의 값을 호스트 변수에 얻어오는 예이다.

```
gsqSQL> \var v_id    INTEGER
gsqSQL> \var v_data  VARCHAR(128)
gsqSQL> DELETE FROM t1 WHERE id = 3 RETURNING id, data INTO :v_id, :v_data;
V_ID V_DATA
----
3 data_3
1 row deleted.
```

호환성

SQL 표준에는 〈delete returning into statement〉 구문이 존재하지 않는다.

참조

관련 내용은 다음을 참조한다.

- DELETE FROM

- DELETE FROM name WHERE CURRENT OF cursor_name
- DELETE FROM name RETURNING
- SELECT

16.97 DELETE FROM name WHERE CURRENT OF cursor_name

기능

커서가 가리키는 row 하나를 삭제한다.

구문

```
<delete statement: positioned> ::=  
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]  
        WHERE CURRENT OF cursor_name  
    ;
```

사용 범위 및 접근 권한

<delete statement: positioned> 구문을 수행하려면 사용자에게 **16.94 DELETE FROM** 구문을 수행할 수 있는 권한이 있어야 한다.

구문 규칙 및 파라미터

table_name

Row를 삭제할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias 이다.

cursor_name

cursor_name에 해당하는 커서는 다음 조건을 만족해야 한다.

- OPEN 된 커서여야 한다. (**OPEN cursor_name**을 참조한다.)
- 커서를 이용해 FETCH한 row가 존재해야 한다. (**FETCH cursor_name**을 참조한다.)
- 커서를 위해 사용된 질의가 table_name을 식별할 수 있어야 한다. (**DECLARE cursor_name**을 참조한다.)
- table_name에 대해 갱신할 수 있는 커서여야 한다. (**DECLARE cursor_name**을 참조한다.)

설명

자세한 내용은 DELETE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 interactive SQL (gsq)에서 FOR UPDATE 커서를 선언하고 그 커서를 이용해 row를 삭제하는 예이다.

```
gsqSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1 FOR UPDATE;
Cursor declared.
gsqSQL> OPEN cur1;
Cursor is open.
gsqSQL> \var v_id    INTEGER
gsqSQL> \var v_data VARCHAR(128)
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
1 data_1
1 row fetched.
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
2 data_2
1 row fetched.
gsqSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
1 row deleted.
gsqSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
3 data_3
```

```
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
4 data_4
1 row fetched.
gSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
1 row deleted.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
gSQL> SELECT id, data FROM t1 ORDER BY 1;
ID DATA
--
1 data_1
3 data_3
5 data_5
3 rows selected.
```

호환성

표 16-40 SQL 표준 호환성

Feature ID	설명	지원 여부
S111	ONLY in query expressions	X
B031	Basic dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- DECLARE cursor_name
- OPEN cursor_name

- FETCH cursor_name
- DELETE FROM
- DELETE FROM name RETURNING
- DELETE FROM name RETURNING .. INTO

16.98 DROP AUDIT POLICY

기능

Audit policy를 제거한다.

구문

```
<drop audit policy statement> ::=  
    DROP AUDIT POLICY [ IF EXISTS ] policy_name  
    ;
```

사용 범위 및 접근 권한

<drop audit policy statement> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

IF EXISTS

policy_name이 존재하지 않더라도 에러가 발생하지 않는다.

policy_name

제거할 audit policy 객체의 이름이다.

설명

이미 활성화된 audit policy 객체는 제거할 수 없다. 이 경우, NOAUDIT POLICY 구문을 이용해 audit policy를 비활성화해야 한다.

사용 예

다음은 audit policy를 제거하는 예이다.

```
DROP AUDIT POLICY policy_table;
```

호환성

SQL 표준에는 audit policy가 존재하지 않는다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - **CREATE AUDIT POLICY**
 - **DROP AUDIT POLICY**
 - **ALTER AUDIT POLICY**
- Audit policy 활성화/ 비활성화
 - **AUDIT POLICY**
 - **NOAUDIT POLICY**
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 제거: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.99 DROP CLUSTER GROUP

기능

Cluster group을 cluster system에서 제거한다.

구문

```
<drop cluster group statement> ::=  
    DROP CLUSTER GROUP [IF EXISTS] group_name  
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<drop cluster group statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

[IF EXISTS]

Cluster group이 존재하지 않더라도 에러가 발생하지 않는다.

group_name

Cluster group의 이름이다.

Shard가 존재하지 않는 cluster group을 제거할 수 있다.

설명

Cluster group을 제거하더라도 data loss가 발생하지 않는 경우에 해당 cluster group을 제거할 수 있다.

사용 예

다음은 cluster group을 제거하는 예이다.

```
gSQL> DROP CLUSTER GROUP g3;  
Cluster Group dropped.
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 CREATE CLUSTER GROUP을 참조한다.

16.100 DROP CLUSTER LOCATION

기능

Cluster member의 접속 정보를 삭제한다.

구문

```
<drop cluster location statement> ::=  
    DROP CLUSTER LOCATION member_name  
    ;
```

사용 범위 및 접근 권한

Cluster system에서 수행할 수 있다.

<drop cluster location statement> 구문을 수행하려면 사용자에게 ADMINISTRATION ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

member_name

Cluster member의 이름이다.

등록된 cluster location 정보에 동일한 cluster member 이름이 존재해야 한다.

이름의 길이는 128 바이트보다 작아야 한다.

설명

기본적으로 cluster location 정보는 cluster group을 생성할 때나 cluster member를 추가할 때 제공되는 접속 정보를 이용하여 자동으로 생성된다. 생성된 정보는 cluster member나 group을 삭제할 때 함께 삭제된다.

만약 cluster location의 접속 정보가 변경되면 cluster member를 삭제하거나 재생성할 필요없이 **ALTER CLUSTER LOCATION**을 이용하여 접속 정보를 변경할 수 있다.

사용 예

```
gSQL>  
DROP CLUSTER LOCATION g1n2  
;  
Created
```

호환성

SQL 표준에서는 cluster에 대한 개념을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE CLUSTER LOCATION
- ALTER CLUSTER LOCATION

16.101 DROP INDEX

기능

인덱스를 제거한다.

구문

```
<drop index statement> ::=  
    DROP INDEX [ IF EXISTS ] index_name  
    ;
```

사용 범위 및 접근 권한

<drop index statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 인덱스의 소유자
- 인덱스가 속한 테이블의 소유자
- 인덱스가 속한 테이블에 대해 CONTROL TABLE ON TABLE
- 인덱스가 속한 스키마에 대해 (DROP INDEX 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY INDEX ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

인덱스가 존재하지 않더라도 에러가 발생하지 않는다.

index_name

삭제할 인덱스의 이름이다.

schema_name.index_name과 같이 인덱스가 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

UNIQUE 제약 조건, PRIMARY KEY 제약 조건을 위해 생성한 인덱스는 제거할 수 없다.

위 제약 조건을 위해 생성된 인덱스를 제거하려면 **ALTER TABLE name DROP CONSTRAINT** 구문을 사용하여 관련된 제약 조건을 삭제해야 한다.

설명

DROP INDEX와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 인덱스를 삭제하는 예이다.

```
gSQL> DROP INDEX idx_t1_id;  
Index dropped.
```

다음은 IF EXISTS 구문을 사용하여 인덱스가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP INDEX IF EXISTS not_exist_index;  
Index dropped.
```

호환성

SQL 표준에서는 인덱스에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE INDEX
- DROP TABLE
- ALTER TABLE name DROP CONSTRAINT

16.102 DROP PROFILE

기능

Profile을 삭제한다.

구문

```
<drop profile statement> ::=  
    DROP PROFILE [ IF EXISTS ] profile_name [ CASCADE ] ;
```

사용 범위 및 접근 권한

<drop profile statement> 구문을 수행하려면 사용자에게 DROP PROFILE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

IF EXISTS

Profile이 존재하지 않더라도 에러가 발생하지 않는다.

profile_name

삭제할 profile의 이름을 명시한다.

DEFAULT profile은 삭제할 수 없다.

CASCADE

이미 할당받은 사용자들이 존재하는 경우, profile을 삭제하기 위해 반드시 이 절을 명시해야 한다.

삭제할 profile을 할당받은 사용자들의 profile은 DEFAULT profile로 변경한다.

사용 예

다음은 CASCADE 구문을 사용하여 profile을 삭제하는 예이다.

```
gSQL> DROP PROFILE prof CASCADE;  
Profile dropped.  
gSQL> COMMIT;  
Commit complete.
```

호환성

SQL 표준에서는 profile에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE PROFILE
- ALTER PROFILE

16.103 DROP SCHEMA

기능

스키마를 제거한다.

구문

```
<drop schema statement> ::=  
    DROP SCHEMA [ IF EXISTS ] schema_name  
        [ <drop behavior> ]  
    ;  
<drop behavior> ::=  
    RESTRICT  
    | CASCADE
```

사용 범위 및 접근 권한

<drop schema statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 스키마의 소유자
- 해당 스키마에 대해 CONTROL SCHEMA ON SCHEMA
- DROP SCHEMA ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

스키마가 존재하지 않더라도 에러가 발생하지 않는다.

schema_name

제거할 스키마의 이름이다.

단, database를 생성할 때 자동으로 생성되는 DICTIONARY_SCHEMA, INFORMATION_SCHEMA, PUBLIC과 같은 built-in 스키마는 제거할 수 없다.

<drop behavior>

- RESTRICT
 - Schema 내에 존재하는 객체가 없어야 한다.
- CASCADE
 - Schema 내의 모든 객체를 함께 제거한다.
- 생략할 경우, 기본값은 RESTRICT이다.

설명

DROP SCHEMA와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 schema와 schema 내에 존재하는 모든 객체를 함께 제거하는 예이다.

```
gSQL> DROP SCHEMA s1 CASCADE;
Schema dropped.
```

다음은 IF EXISTS 구문을 사용하여 schema가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP SCHEMA IF EXISTS not_exist_schema;
Schema dropped.
```

호환성

SQL 표준에서는 IF EXISTS 절을 정의하지 않고 있다.

표 16-41 SQL 표준 호환성

Feature ID	설명	지원 여부
F032	CASCADE drop behavior	O

Feature ID	설명	지원 여부
F381	Extended schema manipulation	O

참조

관련 내용은 CREATE SCHEMA를 참조한다.

16.104 DROP SEQUENCE

기능

시퀀스를 제거한다.

구문

```
<drop sequence generator statement> ::=  
    DROP SEQUENCE [ IF EXISTS ] [schema_name.] sequence_name  
    ;
```

사용 범위 및 접근 권한

<drop sequence generator statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 시퀀스의 소유자
- 시퀀스가 속한 스키마에 대해 (DROP SEQUENCE 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY SEQUENCE ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

시퀀스가 존재하지 않더라도 에러가 발생하지 않는다.

sequence_name

제거할 시퀀스의 이름이다.

schema_name.sequence_name과 같이 시퀀스가 속할 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

DROP SEQUENCE와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 sequence를 제거하는 예이다.

```
gSQL> DROP SEQUENCE seq1;
Sequence dropped.
```

다음은 IF EXISTS 구문을 사용하여 시퀀스가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP SEQUENCE invalid_sequence;
ERR-42000(16044): sequence does not exist :
DROP SEQUENCE invalid_sequence
      *
ERROR at line 1:
gSQL> DROP SEQUENCE IF EXISTS invalid_sequence;
Sequence dropped.
```

호환성

SQL 표준에서는 IF EXISTS 절을 정의하지 않고 있다.

표 16-42 SQL 표준 호환성

Feature ID	설명	지원 여부
T176	Sequence generator support	O

참조

관련 내용은 다음을 참조한다.

- CREATE SEQUENCE
- ALTER SEQUENCE

16.105 DROP SYNONYM

기능

Synonym을 제거한다.

구문

```
<drop synonym statement> ::=  
    DROP [ PUBLIC ] SYNONYM [ IF EXISTS ] [ schema_name. ] synonym_name  
    ;
```

사용 범위 및 접근 권한

PUBLIC을 명시하여 public synonym을 제거하려면 DROP PUBLIC SYNONYM ON DATABASE 권한이 있어야 한다.

Private synonym을 제거하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 synonym의 소유자
- Synonym이 속한 스키마에 대해 (DROP SYNONYM 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY SYNONYM ON DATABASE

구문 규칙 및 파라미터

[PUBLIC]

Public synonym을 제거하고자 할 때 명시한다.

이 절을 생략하면 private synonym이 제거된다.

IF EXISTS

Synonym이 존재하지 않더라도 에러가 발생하지 않는다.

synonym_name

제거할 synonym의 이름이다.

schema_name.synonym_name과 같이 synonym이 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

PUBLIC을 명시한 경우, 스키마 이름을 명시할 수 없다.

설명

DROP SYNONYM과 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 private synonym을 제거하는 예이다.

```
gSQL> DROP SYNONYM MyEmp;  
Synonym dropped.
```

다음은 public synonym을 제거하는 예이다.

```
gSQL> DROP PUBLIC SYNONYM MainEmp;  
Synonym dropped.
```

호환성

SQL 표준에서는 DROP SYNONYM 구문을 정의하지 않고 있다.

참조

관련 내용은 CREATE SYNONYM을 참조한다.

16.106 DROP TABLE

기능

테이블을 제거한다.

구문

```
<drop table statement> ::=  
    DROP TABLE [ IF EXISTS ] table_name  
    [ <drop behavior> ]  
    ;  
  
<drop behavior> ::=  
    RESTRICT  
    | CASCADE  
    | CASCADE CONSTRAINTS
```

사용 범위 및 접근 권한

<drop table statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 테이블의 소유자
- 해당 테이블에 대해 CONTROL TABLE ON TABLE
- 테이블이 속한 스키마에 대해 (DROP TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY TABLE ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

테이블이 존재하지 않더라도 에러가 발생하지 않는다.

table_name

제거할 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 소속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

Database를 생성할 때 자동으로 생성되는 다음과 같은 테이블들은 삭제할 수 없다.

- DEFINITION_SCHEMA 스키마의 테이블들
- FIXED_TABLE_SCHEMA 스키마의 테이블들

테이블에 생성된 제약 조건과 인덱스도 함께 제거한다.

drop behavior

현재는 RESTRICT/ CASCADE가 동일하게 동작한다.

생략할 경우, 기본값은 RESTRICT 이다.

설명

DROP TABLE과 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 일반 테이블을 제거하는 예이다.

```
gSQL> DROP TABLE region;
Table dropped.
```

다음과 같이 IF EXISTS 구문을 사용하면 테이블이 존재하지 않더라도 에러가 발생하지 않는다.

```
gSQL> DROP TABLE IF EXISTS invalid_table;
Table dropped.
```

다음은 DROP 된 테이블을 ROLLBACK 하는 예이다.

```
gSQL> SELECT r_regionkey, r_name FROM region;
R_REGIONKEY R_NAME
```

```

0 AFRICA
1 AMERICA
2 ASIA
3 EUROPE
4 MIDDLE EAST
5 rows selected.
gSQL> DROP TABLE region;
Table dropped.
gSQL> SELECT r_regionkey, r_name FROM region;
ERR-42000(16040): table or view does not exist :
SELECT r_regionkey, r_name FROM region
                                *

ERROR at line 1:
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT r_regionkey, r_name FROM region;
R_REGIONKEY R_NAME
-----
0 AFRICA
1 AMERICA
2 ASIA
3 EUROPE
4 MIDDLE EAST
5 rows selected.
```

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

- IF EXISTS
- CASCADE CONSTRAINTS

표 16-43 SQL 표준 호환성

Feature ID	설명	지원 여부
F032	CASCADE drop behavior	O

참조

관련 내용은 `CREATE TABLE`을 참조한다.

16.107 DROP TABLESPACE

기능

테이블스페이스를 제거한다.

구문

```
<drop tablespace statement> ::=  
    DROP TABLESPACE [ IF EXISTS ] tablespace_name  
        [ INCLUDING CONTENTS ]  
        [ { AND | KEEP } DATAFILES ]  
        [ <drop behavior> ]  
    ;  
<drop behavior> ::=  
    RESTRICT  
    | CASCADE  
    | CASCADE CONSTRAINTS
```

사용 범위 및 접근 권한

<drop tablespace definition> 구문을 수행하려면 사용자에게 DROP TABLESPACE ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

IF EXISTS

테이블스페이스가 존재하지 않더라도 에러가 발생하지 않는다.

tablespace_name

제거할 테이블스페이스의 이름이다.

Database를 생성할 때 구축되는 다음과 같은 시스템 테이블스페이스는 제거할 수 없다.

- `DICTIONARY_TBS` : system tablespace for dictionary management
- `MEM_UNDO_TBS` : system tablespace for default undo tablespace
- `MEM_DATA_TBS` : system tablespace for default user data tablespace
- `MEM_TEMP_TBS` : system tablespace for default temporary tablespace

노트

`tablespace_name`이 사용자들의 default tablespace로 사용되고 있었다면 tablespace가 제거된 후에는 객체를 위한 공간을 할당받을 수 없다. 따라서 tablespace를 제거한 후에는 **ALTER USER** 구문을 사용하여 사용자들의 default tablespace를 변경해 주어야 한다.

INCLUDING CONTENTS

테이블스페이스에 속하는 객체 (table, index, key constraints)를 삭제한다. 테이블스페이스에 속하는 table을 참조하는 index와 key constraints가 테이블스페이스 외부에 존재할 경우에는 이들도 함께 삭제한다.

INCLUDING CONTENTS 구문을 사용하지 않을 경우에는 테이블스페이스에 속하는 객체가 없어야 한다.

[{ AND | KEEP } DATAFILES]

테이블스페이스를 구성하는 데이터 파일들을 함께 삭제할지 여부를 지정한다.

Memory temporary tablespace에는 데이터 파일이 존재하지 않으므로, 해당 절은 무시된다.

- `AND DATAFILES`
 - 데이터 파일들을 함께 삭제한다.
- `KEEP DATAFILES`
 - 데이터 파일을 삭제하지 않고 남겨둔다.
- 명시하지 않을 경우, 기본값은 `KEEP DATAFILES` 이다.

drop behavior

현재는 `RESTRICT/ CASCADE`가 동일하게 동작한다.

생략할 경우, 기본값은 `RESTRICT` 이다.

설명

DROP TABLESPACE 구문은 다른 Data Definition Language (DDL)과 달리 ROLLBACK 할 수 없으며, 구문을 수행한 transaction이 자동으로 COMMIT 된다.

사용 예

다음은 tablespace와 함께 tablespace에 존재하는 모든 객체와 tablespace를 구성하는 모든 data file을 삭제하는 예이다.

```
gSQL> DROP TABLESPACE space1 INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;  
Tablespace dropped.
```

다음은 IF EXISTS 구문을 사용하여 tablespace가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP TABLESPACE IF EXISTS not_exist_tablespace;  
Tablespace dropped.
```

호환성

SQL 표준은 tablespace에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- CREATE MEMORY DATA TABLESPACE
- CREATE MEMORY TEMPORARY TABLESPACE
- ALTER TABLESPACE

16.108 DROP USER

기능

데이터베이스 사용자를 제거한다.

구문

```
<drop user statement> ::=  
    DROP USER [ IF EXISTS ] user_identifier [ <drop behavior> ]  
    ;  
<drop behavior> ::=  
    RESTRICT  
    | CASCADE
```

사용 범위 및 접근 권한

<drop user statement> 구문을 수행하려면 사용자에게 DROP USER ON DATABASE 권한이 있어야 한다.

노트

user_identifier가 소유한 스키마가 존재하지 않아야 한다.
스키마 제거에 대한 자세한 내용은 **DROP SCHEMA** 구문을 참조한다.

구문 규칙 및 파라미터

IF EXISTS

사용자가 존재하지 않더라도 에러가 발생하지 않는다.

user_identifier

제거할 데이터베이스 사용자의 이름이다.

단, database를 생성할 때 자동으로 생성되는 "SYS" 등과 같은 사용자는 제거할 수 없다.

다음과 같이 user_identifier가 생성했으나, 소유자가 아닌 객체는 제거하지 않는다.

- Role
- Tablespace

<drop behavior>

- RESTRICT
 - User가 소유한 다음과 같은 SQL schema object가 존재하지 않아야 한다.
 - Table, view
 - Index
 - Sequence
 - Table constraint
- CASCADE
 - User가 소유한 다음과 같은 SQL schema object를 모두 제거한다.
 - Table, view
 - Index
 - Sequence
 - Table constraint
- 생략할 경우, 기본값은 RESTRICT 이다.

노트

DBMS에서 user와 schema의 관계

- Oracle
 - User : schema = 1 : 1
 - CASCADE 할 때 schema도 함께 제거한다.
- DB2
 - OS user와 동일하다.
 - User를 생성하고 제거하는 별도의 SQL 구문이 없다.

- Postgres
 - User : schema = 1 : N
 - CASCADE 옵션이 없으며 user가 소유한 모든 객체와 다른 사용자에게 부여한 모든 권한을 제거해야 user를 제거할 수 있다.
- MySQL
 - Database : schema = 1 : 1
 - User는 database (schema)의 하위 객체이며, CASCADE 옵션이 존재하지 않는다.

설명

GOLDILOCKS에서 user와 schema의 관계는 1 : N 이다.

즉, user가 schema를 소유하지 않을 수도 있고, 다수의 schema를 소유할 수도 있다.

User 객체를 제거하려면 user가 소유한 모든 schema를 제거해야 한다.

사용 예

다음은 user가 소유한 모든 schema를 모두 제거한 후 해당 user를 제거하는 예이다.

```
gSQL> DROP SCHEMA u1 CASCADE;
Schema dropped.
gSQL> DROP USER u1 CASCADE;
User dropped.
```

다음은 IF EXISTS 구문을 사용하여 user가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP USER IF EXISTS not_exist_user;
User dropped.
```

호환성

SQL 표준에서는 user의 개념은 다루고 있지만 user의 생성 및 제거와 관련된 SQL 구문은 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- [CREATE USER](#)
- [ALTER USER](#)
- [DROP SCHEMA](#)

16.109 DROP VIEW

기능

View를 제거한다.

구문

```
<drop view statement> ::=  
    DROP VIEW [ IF EXISTS ] view_name  
    ;
```

사용 범위 및 접근 권한

<drop view statement> 구문을 수행하려면 사용자는 다음 권한 중 하나가 있어야 한다.

- 해당 view의 소유자
- 해당 view에 대해 CONTROL TABLE ON TABLE
- View가 속한 스키마에 대해 (DROP VIEW 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY VIEW ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

View가 존재하지 않더라도 에러가 발생하지 않는다.

view_name

제거할 view의 이름이다.

schema_name.view_name과 같이 테이블이 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

DROP VIEW와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 view를 제거하는 예이다.

```
gSQL> DROP VIEW v1;  
View dropped.
```

다음은 IF EXISTS 구문을 사용하여 view가 존재하지 않더라도 에러가 발생하지 않도록 하는 예이다.

```
gSQL> DROP VIEW IF EXISTS not_exist_view;  
View dropped.
```

호환성

SQL 표준에서는 IF EXISTS 절을 정의하지 않고 있다.

표 16-44 SQL 표준 호환성

Feature ID	설명	지원 여부
F032	CASCADE drop behavior	X

참조

관련 내용은 다음을 참조한다.

- CREATE VIEW
- ALTER VIEW

16.110 EXECUTE statement_name

기능

준비된 statement를 수행한다.

구문

```

<execute statement> ::=
    EXECUTE statement_name [ <parameter using clause> ] [ <result into clause> ]
    ;
<parameter using clause> ::=
    <using parameter arguments>
<using parameter arguments> ::=
    USING variable_name [, ...]
<result into clause> ::=
    <into result arguments>
<into result arguments> ::=
    INTO variable_name [, ...]

```

사용 범위 및 접근 권한

Embedded SQL에서 사용할 수 있다.

Dynamic SQL 구문의 종류에 부합하는 수행 권한이 있어야 한다.

구문 규칙 및 파라미터

statement_name

준비된 statement의 이름이다.

PREPARE statement_name 구문을 사용하여 statement_name을 준비해야 한다.

statement_name이 참조하는 dynamic SQL 문장이 dynamic parameter를 포함하고 있는 경우, <parameter using clause>를 명시해야 한다.


```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 > ?';
    EXEC SQL EXECUTE stmt1 USING :sValue;
    ...
}
```

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT COUNT(*) INTO :v1 FROM t1';
    EXEC SQL EXECUTE stmt1 USING :sValue;
    ...
}
```

statement_name이 참조하는 dynamic SQL 문장이 query이거나 결과가 존재하는 stored function일 경우, <result into clause>를 명시해야 한다.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT COUNT(*) FROM t1';
    EXEC SQL EXECUTE stmt1 INTO :sValue;
    ...
}
```

질의가 여러 건인 경우 정상적으로 수행되지만 결과는 최초 한 건만 얻을 수 있다.
여러 건의 결과를 얻기 위해서는 다음과 같은 커서 관련 구문을 사용해야 한다.

- DECLARE cursor_name
- OPEN cursor_name
- FETCH cursor_name
- CLOSE cursor_name

질의 결과가 없을 경우, NO DATA로 완료된다.

[<parameter using clause>] [<result into clause>]

<parameter using clause>와 <result into clause>는 순서에 관계없이 기술할 수 있지만 중복해서 기술하지 않아야 한다.

〈parameter using clause〉

statement_name이 참조하는 dynamic SQL 문장에 parameter가 존재할 경우, parameter에 대한 정보를 〈using parameter arguments〉 절로 명시한다.

〈using parameter arguments〉

〈using parameter arguments〉 구문이 사용될 경우, variable_name의 개수는 statement_name이 참조하는 dynamic SQL 문장에 포함된 parameter의 개수와 동일해야 한다.

나열된 variable_name은 기술된 순서대로 dynamic parameter 순서에 대응된다.

```
{
  ...
  EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 IN ( ?, ?, ? )';
  EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2, :sValue3;
  ...
}
```

〈result into clause〉

statement_name이 참조하는 dynamic SQL 문장이 query일 경우, 결과 column에 대한 정보를 〈into result arguments〉 절로 명시한다.

결과값이 null인 경우, INDICATOR를 명시하지 않으면 [DATA EXCEPTION, NULL VALUE, NO INDICATOR PARAMETER] 에러가 발생한다.

〈into result arguments〉

〈into result arguments〉 구문이 사용될 경우, variable_name의 개수는 statement_name이 참조하는 dynamic SQL 문장의 결과 column 개수와 동일해야 한다.

나열된 variable_name은 기술된 순서대로 dynamic parameter 순서에 대응된다.

```
{
  ...
  EXEC SQL PREPARE stmt1 FROM 'SELECT MIN(salary), MAX(salary), AVG(salary) FROM employee';
  EXEC SQL EXECUTE stmt1 INTO :sMinValue, :sMaxValue, :sAvgValue;
  ...
}
```

설명

statement_name은 embedded SQL 소스 코드에서 precompiler에게 statement를 알려주는 식별자로써, host variable이 아니기 때문에 별도의 type이나 선언이 필요하지 않다. EXECUTE statement_name 구문은 PREPARE statement_name 구문 뒤에 쓰여야 한다.

자세한 내용은 **Embedded Dynamic SQL**을 참조한다.

사용 예

다음은 EXECUTE statement_name이 embedded SQL 소스 코드에서 사용되는 예이다.

```
{
    ...
    sprintf( sUpdateSql, "UPDATE EMP SET sal = sal * :v1 WHERE JOB = 'SALES'");
    EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sRatio = 1.1;
    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}
```

EXECUTE statement_name이 사용된 전체 소스 코드는 **Dynamic Embedded SQL Example Program**에서 확인할 수 있다.

호환성

표 16-45 SQL 표준 호환성

Feature ID	설명	지원 여부
B031	Basic Dynamic SQL	O

Feature ID	설명	지원 여부
B032	Extended dynamic SQL	X

참조

관련 내용은 다음을 참조한다.

- `PREPARE statement_name`
- `DECLARE cursor_name`
- `OPEN cursor_name`
- `FETCH cursor_name`
- `CLOSE cursor_name`
- Embedded Dynamic SQL

16.111 EXECUTE IMMEDIATE 'sql_string'

기능

프로그램 작성 시점에 정의되지 않았던 dynamic SQL 문장을 수행한다.

구문

```

<execute immediate statement> ::=
    EXECUTE IMMEDIATE <SQL statement variable>
    ;
<SQL statement variable> ::=
    variable_name
    | 'sql statement'
    | "sql statement"
    | sql statement

```

사용 범위 및 접근 권한

Embedded SQL에서 사용할 수 있다.

Dynamic SQL 구문의 종류에 부합하는 수행 권한이 있어야 한다.

구문 규칙 및 파라미터

<SQL statement variable>

<SQL statement variable>이 참조하는 dynamic SQL 문장은 host variable (:var)이나 parameter marker (?)를 사용할 수 없다.

다음과 같은 네 가지 유형의 <SQL statement variable>을 사용할 수 있다.

- variable_name: SQL이 저장된 변수
- 'sql statement': Single quote (')로 묶인 SQL 문장
- "sql statement": Double quote (")로 묶인 SQL 문장
- sql statement: Quote 없는 SQL 문장

Single-quoted string 내에 문자열 data를 표현하려면 다음과 같이 single quote (')를 두 번 기술해야 한다.

```
{
    ...
    EXEC SQL EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''literal data'' )';
    ...
}
```

SQL 문장이 질의 결과를 갖는 query인 경우, 수행에는 성공하지만 그 결과는 얻을 수 없다.

variable_name

variable_name에 대응되는 type은 character string이어야 한다.

variable_name에 정의된 dynamic SQL 문장은 유효한 문장이어야 한다.

sql statement

sql statement에 정의된 dynamic SQL 문장은 유효한 문장이어야 한다.

설명

EXECUTE IMMEDIATE 'sql_string' 구문은 dynamic embedded SQL 응용 프로그램에서 host variable이 없는 non-query SQL에 사용될 수 있다. 별도의 준비과정이 필요하지 않기 때문에, DDL이나 DML 등을 일회성으로 수행하기에 적합하다.

자세한 내용은 **Embedded Dynamic SQL**을 참조한다.

사용 예

다음은 EXECUTE IMMEDIATE 'sql_string'이 embedded SQL 소스 코드 내에서 사용되는 예이다.

```
{
    ...
    sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
        "SELECT *\n"
        "FROM EMP\n"
        "WHERE JOB = 'RND'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
```

```

    {
        goto fail_exit;
    }
    ...
}

```

EXECUTE IMMEDIATE 'sql_string'이 사용된 전체 소스 코드는 **Dynamic Embedded SQL Example Program**에서 확인할 수 있다.

호환성

표 16-46 SQL 표준 호환성

Feature ID	설명	지원 여부
B031	Basic Dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- PREPARE statement_name
- EXECUTE statement_name
- Embedded Dynamic SQL

16.112 FETCH cursor_name

기능

커서를 결과 집합의 특정 row에 위치시키고, 해당 row의 값을 호스트 변수에 얻어온다.

구문

```

<fetch statement> ::=
    FETCH [ <fetch orientation> ] [ FROM ] cursor_name
        <result into clause>
    ;
<fetch orientation> ::=
    NEXT
    | PRIOR
    | FIRST
    | LAST
    | CURRENT
    | ABSOLUTE position
    | RELATIVE position
<result into clause> ::=
    <into result arguments>
<into result arguments> ::=
    INTO variable_name [, ...]
  
```

구문 규칙 및 파라미터

[FROM] cursor_name

세션 내에서 open 된 커서이어야 한다.
FROM은 생략할 수 있다.

<fetch orientation>

FETCH NEXT 이외의 <fetch orientation>을 사용하려면 scrollable cursor를 사용해야 한다.
<fetch orientation>을 생략할 경우, 기본값은 NEXT이다.

Open 된 커서는 결과 집합에 대해 아래 그림과 같은 커서 위치 정보를 갖는다.

그림 1 커서의 위치 정보

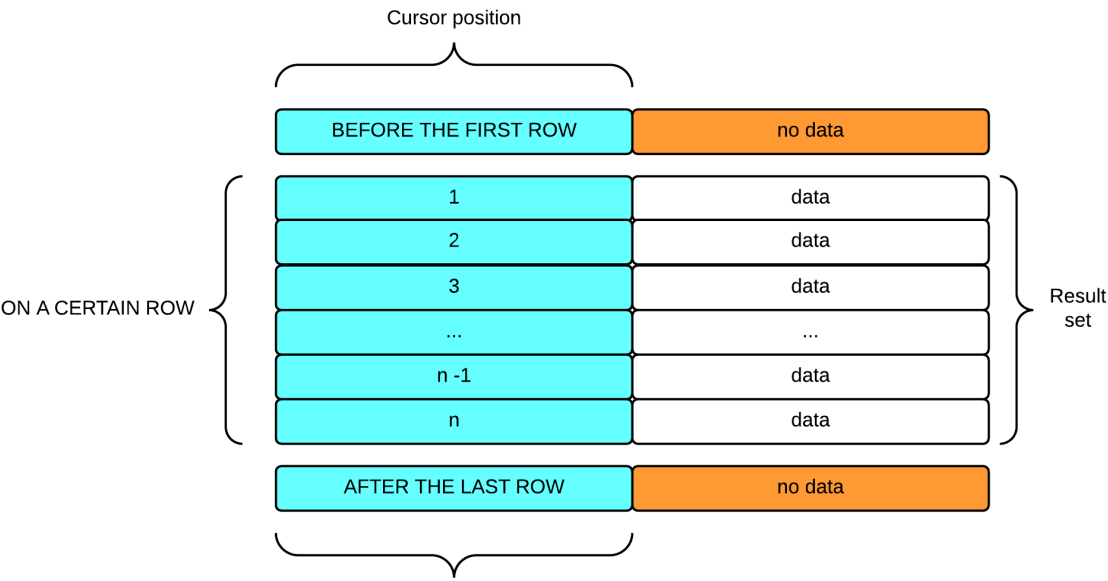


표 16-47 커서의 위치

커서의 위치	설명
BEFORE THE FIRST ROW	결과 집합의 첫 번째 row의 이전 위치에 있는 상태로써 OPEN 시점의 위치도 이에 해당한다.
ON A CERTAIN ROW	FETCH를 통해 결과 집합의 특정 row에 위치한 상태이다.
AFTER THE LAST ROW	결과 집합의 마지막 row 이후의 위치에 있는 상태이다.

현재 커서의 위치를 기준으로 각 <fetch orientation>은 다음과 같이 동작한다.

- NEXT: 현재 위치의 다음 row를 검색한다.
- PRIOR: 현재 위치의 이전 row를 검색한다.
- FIRST: 결과 집합의 첫 번째 row를 검색한다.
- LAST: 결과 집합의 마지막 row를 검색한다.
- CURRENT: 현재 위치의 row를 검색한다.
- ABSOLUTE position
 - 결과 집합에서 position의 위치에 해당하는 row를 검색한다.
 - Position 값이 음수일 경우 AFTER THE LAST ROW로부터 이전의 위치에 해당하는 row를 검색한다.
- RELATIVE position
 - 현재 위치에서 position만큼 떨어진 위치에 해당하는 row를 검색한다.

〈result into clause〉

〈into result arguments〉를 사용하여 결과 column을 획득할 변수 정보를 기술한다.

결과값이 null인 경우, INDICATOR를 명시하지 않으면 [DATA EXCEPTION, NULL VALUE, NO INDICATOR PARAMETER] 에러가 발생한다.

〈into result arguments〉

INTO 절에 기술된 변수의 개수는 커서의 결과 집합의 column 개수와 동일해야 한다.

설명

FETCH를 수행한 후에 커서 위치가 BEFORE THE FIRST ROW 거나 AFTER THE LAST LOW 인 경우, 〈fetch orientation〉에 입력된 위치값에 관계없이 동일한 위치에 자리한다.

사용 예

다음은 interactive SQL (gsq)에서 SCROLL 커서를 선언하고, 다양한 〈fetch orientation〉의 동작을 보여주는 예이다.

```
gsqSQL> DECLARE cur_scroll SCROLL CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gsqSQL> OPEN cur_scroll;
Cursor is open.
gsqSQL> \var v_id    INTEGER
gsqSQL> \var v_data VARCHAR(128)
gsqSQL> FETCH NEXT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
1 data_1
1 row fetched.
gsqSQL> FETCH NEXT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
2 data_2
1 row fetched.
gsqSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
```

```

-----
      1 data_1
1 row fetched.
gSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      1 data_1
1 row fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      1 data_1
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      5 data_5
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      5 data_5
1 row fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      1 data_1
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      1 data_1
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
-----

      5 data_5
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;

```

```

V_ID V_DATA
----
      5 data_5
1 row fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      3 data_3
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      3 data_3
1 row fetched.
gSQL> FETCH ABSOLUTE 1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      1 data_1
1 row fetched.
gSQL> FETCH ABSOLUTE -1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      5 data_5
1 row fetched.
gSQL> FETCH ABSOLUTE 6 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH ABSOLUTE -6 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      3 data_3
1 row fetched.
gSQL> FETCH ABSOLUTE -3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
      3 data_3
1 row fetched.
gSQL> FETCH RELATIVE 1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----

```

```

    4 data_4
1 row fetched.
gSQL> FETCH RELATIVE -1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
    3 data_3
1 row fetched.
gSQL> FETCH RELATIVE 5 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH RELATIVE -5 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
----
    1 data_1
1 row fetched.
gSQL> CLOSE cur_scroll;
Cursor closed.
```

호환성

SQL 표준에서는 <fetch orientation> 중에 CURRENT를 정의하지 않고 있다.

표 16-48 SQL 표준 호환성

Feature ID	설명	지원 여부
F431	Read-only scrollable cursors	O
B031	Basic dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- DECLARE cursor_name
- OPEN cursor_name
- CLOSE cursor_name

16.113 GRANT privileges TO

기능

사용자에게 권한을 부여한다.

구문

```

<grant privilege statement> ::=
    GRANT <privilege> TO <grantee> [, ...]
        [ WITH GRANT OPTION ]
    ;

<grantee> ::=
    PUBLIC
    | user_identifier
    ;

<privilege> ::=
    <database privilege>
    | <tablespace privilege>
    | <schema privilege>
    | <table privilege>
    | <sequence privilege>
    | <procedure privilege>

<database privilege> ::=
    ALL [ PRIVILEGES ] [ON DATABASE]
    | <database action> [, ...] [ON DATABASE]

<database action> ::=
    ADMINISTRATION
    | ANALYZE ANY
    | ALTER DATABASE
    | ALTER SYSTEM
    | AUDIT SYSTEM
    | ACCESS CONTROL
    | CREATE SESSION
    | CREATE PROFILE
    | ALTER PROFILE

```

```

| DROP PROFILE
| CREATE USER
| ALTER USER
| DROP USER
| CREATE ROLE
| ALTER ROLE
| DROP ROLE
| CREATE TABLESPACE
| ALTER TABLESPACE
| DROP TABLESPACE
| USAGE TABLESPACE
| CREATE SCHEMA
| ALTER SCHEMA
| DROP SCHEMA
| CREATE PUBLIC SYNONYM
| DROP PUBLIC SYNONYM
| CREATE ANY TABLE
| ALTER ANY TABLE
| DROP ANY TABLE
| SELECT ANY TABLE
| INSERT ANY TABLE
| DELETE ANY TABLE
| UPDATE ANY TABLE
| LOCK ANY TABLE
| CREATE ANY VIEW
| DROP ANY VIEW
| CREATE ANY SEQUENCE
| ALTER ANY SEQUENCE
| DROP ANY SEQUENCE
| USAGE ANY SEQUENCE
| CREATE ANY INDEX
| ALTER ANY INDEX
| DROP ANY INDEX
| CREATE ANY SYNONYM
| DROP ANY SYNONYM
| CREATE ANY PROCEDURE
| ALTER ANY PROCEDURE
| DROP ANY PROCEDURE
| EXECUTE ANY PROCEDURE

```

<tablespace privilege> ::=

```
ALL [ PRIVILEGES ] ON TABLESPACE tablespace_name
```

```

    | <tablespace action> [, ...] ON TABLESPACE tablespace_name
<tablespace action> ::=
    CREATE OBJECT
<schema privilege> ::=
    ALL [ PRIVILEGES ] ON SCHEMA schema_name
    | <schema action> [, ...] [ON SCHEMA schema_name]
<schema action> ::=
    CONTROL SCHEMA
    | CREATE TABLE
    | ALTER TABLE
    | DROP TABLE
    | SELECT TABLE
    | INSERT TABLE
    | DELETE TABLE
    | UPDATE TABLE
    | LOCK TABLE
    | CREATE VIEW
    | DROP VIEW
    | CREATE SEQUENCE
    | ALTER SEQUENCE
    | DROP SEQUENCE
    | USAGE SEQUENCE
    | CREATE INDEX
    | ALTER INDEX
    | DROP INDEX
    | ADD CONSTRAINT
    | CREATE SYNONYM
    | DROP SYNONYM
    | CREATE PROCEDURE
    | ALTER PROCEDURE
    | DROP PROCEDURE
    | EXECUTE PROCEDURE
<table privilege> ::=
    ALL [ PRIVILEGES ] ON [TABLE] table_name
    | { <table action> | <column action> } [, ...] ON [TABLE] table_name
<table action> ::=
    CONTROL TABLE
    | SELECT
    | INSERT
    | UPDATE
    | DELETE

```



```

| REFERENCES
| LOCK
| INDEX
| ALTER
<column action> ::=
    SELECT ( column_name [, ...] )
| INSERT ( column_name [, ...] )
| UPDATE ( column_name [, ...] )
| REFERENCES ( column_name [, ...] )
<sequence privilege> ::=
    ALL [ PRIVILEGES ] ON SEQUENCE sequence_name
| <sequence action> ON SEQUENCE sequence_name
<sequence action> ::=
    USAGE
<procedure privilege> ::=
    ALL [ PRIVILEGES ] ON PROCEDURE procedure_name
| <procedure action> ON PROCEDURE procedure_name
<procedure action> ::=
    EXECUTE

```

구문 규칙 및 파라미터

<grantee>

권한을 부여받을 사용자이다.

- user_identifier
 - 해당 사용자에게 권한을 부여한다
- PUBLIC
 - 모든 사용자를 의미하는 authorization 객체이다.

WITH GRANT OPTION

Grantee (권한을 부여받은 사용자)가 다른 사용자에게 해당 권한을 부여할 수 있도록 한다.

다음과 같이 동일한 <privilege>에 대한 권한을 부여할 때 WITH GRANT OPTION은 계속 유지된다.

- GRANT SELECT ON t1 TO u1 WITH GRANT OPTION;
- GRANT SELECT ON t1 TO u1;

<privilege>

Grantee (권한을 부여받는 사용자)에게 부여할 권한이다.

Grantor (구문을 수행하는 사용자)는 다음 조건 중 하나를 만족해야 한다.

- Grantor가 WITH GRANT OPTION을 사용하여 해당 <privilege>를 소유한다.
 - Grantor는 구문을 수행하는 사용자가 된다.
- Grantor가 ACCESS CONTROL ON DATABASE 권한을 소유한다.
 - Grantor는 객체의 소유자가 된다.
 - <database privilege>: _SYSTEM 계정
 - <tablespace privilege>: _SYSTEM 계정
 - <schema privilege>: _SYSTEM 계정
 - <table privilege>: table의 소유자
 - <sequence privilege>: sequence의 소유자
 - <procedure privilege>: procedure/ function의 소유자

<database privilege>

데이터베이스 객체에 대한 권한이다.

[ON DATABASE] 구문은 생략할 수 있다.

database privilege로 정의할 수 있는 database action은 다음과 같다.

- ALL [PRIVILEGES] [ON DATABASE]
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 DATABASE에 대해 소유한 모든 권한이다.

표 16-49 Database privilege

<database action>	설명
ADMINISTRATION	서버 구동, 종료 권한
ALTER DATABASE	ALTER DATABASE 구문을 수행할 수 있는 권한
ALTER SYSTEM	ALTER SYSTEM 구문을 수행할 수 있는 권한
AUDIT SYSTEM	Audit policy를 제어할 수 있는 권한
ACCESS CONTROL	모든 권한을 제어할 수 있는 권한
CREATE SESSION	Database에 접속할 수 있는 권한
CREATE PROFILE	Database에 profile을 생성할 수 있는 권한
ALTER PROFILE	Database의 모든 profile을 변경할 수 있는 권한
DROP PROFILE	Database의 모든 profile을 제거할 수 있는 권한
CREATE USER	Database에 user를 생성할 수 있는 권한
ALTER USER	Database의 모든 user를 변경할 수 있는 권한
DROP USER	Database의 모든 user를 제거할 수 있는 권한
CREATE ROLE	Database에 role을 생성할 수 있는 권한

〈database action〉	설명
ALTER ROLE	Database의 모든 role을 변경할 수 있는 권한
DROP ROLE	Database의 모든 role을 제거할 수 있는 권한
CREATE TABLESPACE	Database에 tablespace를 생성할 수 있는 권한
ALTER TABLESPACE	Database의 모든 tablespace를 변경할 수 있는 권한
DROP TABLESPACE	Database의 모든 tablespace를 제거할 수 있는 권한
USAGE TABLESPACE	Database의 모든 tablespace를 사용할 수 있는 권한
CREATE SCHEMA	Database에 스키마를 생성할 수 있는 권한
ALTER SCHEMA	Database의 모든 스키마를 변경할 수 있는 권한
DROP SCHEMA	Database의 모든 스키마를 제거할 수 있는 권한
CREATE PUBLIC SYNONYM	Database에 PUBLIC SYNONYM을 생성할 수 있는 권한
DROP PUBLIC SYNONYM	Database의 모든 PUBLIC SYNONYM을 제거할 수 있는 권한
CREATE ANY TABLE	Database의 모든 스키마에 테이블을 생성할 수 있는 권한
ALTER ANY TABLE	Database의 모든 테이블을 변경할 수 있는 권한
DROP ANY TABLE	Database의 모든 테이블을 제거할 수 있는 권한
SELECT ANY TABLE	Database의 모든 테이블의 row를 검색할 수 있는 권한
INSERT ANY TABLE	Database의 모든 테이블에 row를 생성할 수 있는 권한
DELETE ANY TABLE	Database의 모든 테이블의 row를 삭제할 수 있는 권한
UPDATE ANY TABLE	Database의 모든 테이블의 row를 갱신할 수 있는 권한
LOCK ANY TABLE	Database의 모든 테이블에 LOCK 구문을 수행할 수 있는 권한
CREATE ANY VIEW	Database의 모든 스키마에 view를 생성할 수 있는 권한
DROP ANY VIEW	Database의 모든 view를 제거할 수 있는 권한
CREATE ANY SEQUENCE	Database의 모든 스키마에 시퀀스를 생성할 수 있는 권한
ALTER ANY SEQUENCE	Database의 모든 시퀀스를 변경할 수 있는 권한
DROP ANY SEQUENCE	Database의 모든 시퀀스를 제거할 수 있는 권한
USAGE ANY SEQUENCE	Database의 모든 시퀀스를 사용할 수 있는 권한
CREATE ANY INDEX	Database의 모든 스키마에 인덱스를 생성할 수 있는 권한
ALTER ANY INDEX	Database의 모든 인덱스를 변경할 수 있는 권한
DROP ANY INDEX	Database의 모든 인덱스를 제거할 수 있는 권한
CREATE ANY SYNONYM	Database의 모든 synonym을 생성할 수 있는 권한
DROP ANY SYNONYM	Database의 모든 synonym을 제거할 수 있는 권한
CREATE ANY PROCEDURE	Database의 모든 스키마에 procedure/ function을 생성할 수 있는 권한
ALTER ANY PROCEDURE	Database의 모든 procedure/ function을 변경할 수 있는 권한
DROP ANY PROCEDURE	Database의 모든 procedure/ function을 제거할 수 있는 권한
EXECUTE ANY PROCEDURE	Database의 모든 procedure/ function을 수행할 수 있는 권한

〈tablespace privilege〉

테이블스페이스 객체에 대한 권한이다.

tablespace privilege로 정의할 수 있는 tablespace action은 다음과 같다.

- ALL [PRIVILEGES] ON TABLESPACE tablespace_name
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 해당 TABLESPACE에 대해 소유한 모든 권한이다.

표 16-50 Tablespace privilege

〈tablespace action〉	설명
CREATE OBJECT	Tablespace에 객체를 생성할 수 있는 권한

〈schema privilege〉

스키마 객체에 대한 권한이다.

- [ON SCHEMA schema_name] 생략 여부
 - 〈grantee〉가 다수일 경우 [ON SCHEMA schema_name] 구문을 생략할 수 없다.
 - ALL [PRIVILEGES]를 사용할 경우 [ON SCHEMA schema_name] 구문을 생략할 수 없다.
 - [ON SCHEMA schema_name] 구문을 생략할 경우 〈grantee〉는 하나의 user_identifier만 사용할 수 있으며, 〈grantee〉의 스키마 검색 경로 중 첫 번째 스키마에 대한 권한을 부여한다.

schema privilege로 정의할 수 있는 schema action은 다음과 같다.

- ALL [PRIVILEGES] ON SCHEMA schema_name
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 해당 SCHEMA에 대해 소유한 모든 권한이다.

표 16-51 Schema privilege

〈schema action〉	설명
CONTROL SCHEMA	해당 스키마에 대한 모든 권한
CREATE TABLE	스키마에 테이블을 생성할 수 있는 권한
ALTER TABLE	스키마의 모든 테이블을 변경할 수 있는 권한
DROP TABLE	스키마의 모든 테이블을 제거할 수 있는 권한
SELECT TABLE	스키마의 모든 테이블의 row를 검색할 수 있는 권한
INSERT TABLE	스키마의 모든 테이블의 row를 생성할 수 있는 권한
DELETE TABLE	스키마의 모든 테이블의 row를 삭제할 수 있는 권한
UPDATE TABLE	스키마의 모든 테이블의 row를 갱신할 수 있는 권한
LOCK TABLE	스키마의 모든 테이블에 LOCK 구문을 수행할 수 있는 권한
CREATE VIEW	스키마에 view를 생성할 수 있는 권한
DROP VIEW	스키마의 모든 view를 제거할 수 있는 권한
CREATE SEQUENCE	스키마에 시퀀스를 생성할 수 있는 권한
ALTER SEQUENCE	스키마의 모든 시퀀스를 변경할 수 있는 권한
DROP SEQUENCE	스키마의 모든 시퀀스를 제거할 수 있는 권한
USAGE SEQUENCE	스키마의 모든 시퀀스를 사용할 수 있는 권한
CREATE INDEX	스키마에 인덱스를 생성할 수 있는 권한

〈schema action〉	설명
ALTER INDEX	스키마의 모든 인덱스를 변경할 수 있는 권한
DROP INDEX	스키마의 모든 인덱스를 제거할 수 있는 권한
ADD CONSTRAINT	스키마에 제약 조건을 생성할 수 있는 권한
CREATE SYNONYM	스키마에 synonym을 생성할 수 있는 권한
DROP SYNONYM	스키마의 모든 synonym을 제거할 수 있는 권한
CREATE PROCEDURE	스키마에 procedure/ function을 생성할 수 있는 권한
ALTER PROCEDURE	스키마의 모든 procedure/ function을 변경할 수 있는 권한
DROP PROCEDURE	스키마의 모든 procedure/ function을 제거할 수 있는 권한
EXECUTE PROCEDURE	스키마의 모든 procedure/ function을 수행할 수 있는 권한

〈table privilege〉

테이블 또는 view 객체에 대한 권한이다.

[TABLE] 구문은 생략할 수 있다.

table privilege로 정의할 수 있는 table action은 다음과 같다.

- ALL [PRIVILEGES] ON [TABLE] table_name
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 해당 TABLE에 대해 소유한 모든 권한이다.

표 16-52 Table privilege

〈table action〉	설명
CONTROL TABLE	해당 테이블에 대한 모든 권한
SELECT	테이블의 row를 검색할 수 있는 권한
INSERT	테이블의 row를 생성할 수 있는 권한
UPDATE	테이블의 row를 갱신할 수 있는 권한
DELETE	테이블의 row를 삭제할 수 있는 권한
REFERENCES	해당 테이블을 참조하는 참조 제약 조건을 생성할 수 있는 권한
LOCK	테이블에 LOCK 구문을 수행할 수 있는 권한
INDEX	테이블에 인덱스를 생성할 수 있는 권한
ALTER	테이블을 변경할 수 있는 권한

SELECT, INSERT, UPDATE, REFERENCES의 경우, 테이블의 모든 column에 추가적으로 권한을 부여한다.

table privilege로 정의할 수 있는 column action은 다음과 같다. 단, column action은 base table에만 적용된다.

표 16-53 Column privilege

〈column action〉	설명
SELECT (columns)	해당 column들을 검색할 수 있는 권한
INSERT (columns)	해당 column들을 포함한 row를 생성할 수 있는 권한

〈column action〉	설명
UPDATE (columns)	해당 column들을 갱신할 수 있는 권한
REFERENCES (columns)	해당 column들을 참조하는 참조 제약 조건을 생성할 수 있는 권한

〈sequence privilege〉

시퀀스 객체에 대한 권한이다.

sequence privilege로 정의할 수 있는 sequence action은 다음과 같다.

- ALL [PRIVILEGES] ON SEQUENCE sequence_name
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 해당 SEQUENCE에 대해 소유한 모든 권한이다.

표 16-54 Sequence privilege

〈sequence action〉	설명
USAGE	시퀀스를 사용할 수 있는 권한

〈procedure privilege〉

Procedure/ function 객체에 대한 권한이다.

procedure privilege로 정의할 수 있는 action은 다음과 같다.

- ALL [PRIVILEGES] ON PROCEDURE procedure_name
 - Grantor (구문을 수행하는 사용자)가 WITH GRANT OPTION을 사용하여 해당 procedure/ function에 대해 소유한 모든 권한이다.

표 16-55 Procedure privilege

〈procedure action〉	설명
EXECUTE	Procedure/ function을 실행할 수 있는 권한

설명

GRANT privilege와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

Table, sequence 등과 같은 SQL schema object를 생성한 owner는 해당 객체에 대한 권한을 별도로 부여받지 않더라도 일정한 권한을 가진다.

이에 대한 자세한 설명은 다음과 같은 CREATE 구문을 참조한다.

- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- ALTER TABLE name ADD COLUMN
- CREATE FUNCTION
- CREATE PROCEDURE

Schema, tablespace 등과 같은 non-schema object를 생성한 owner에는 해당 객체에 대한 어떤 권한도 자동으로 부여되지 않으므로 별도의 권한을 부여받아야 한다.

자세한 설명은 다음과 같은 CREATE 구문을 참조한다.

- CREATE SCHEMA
- CREATE TABLESPACE
- CREATE USER

사용 예

다음은 user u1에 SELECT ON TABLE t1 권한을 부여하는 예이다.

```
gSQL> GRANT SELECT ON t1 TO u1;
Grant succeeded.
```

다음은 모든 사용자를 의미하는 PUBLIC 계정에 SELECT ON TABLE t1 권한을 부여하는 예이다.

```
gSQL> GRANT SELECT ON t1 TO PUBLIC;
Grant succeeded.
```

다음은 user u1이 WITH GRANT OPTION을 사용하여 다른 user에게 해당 권한을 부여하는 예이다.

```
gSQL> GRANT SELECT ON t1 TO u1 WITH GRANT OPTION;
Grant succeeded.
```

다음은 구문을 수행하는 사용자가 WITH GRANT OPTION을 사용하여 TABLE t1 객체에 대해 소유한 모든 권한을 user u1에게 부여하는 예이다.

```
gSQL> GRANT ALL PRIVILEGES ON TABLE t1 TO u1;
Grant succeeded.
```

다음은 database에 접속할 수 있는 CREATE SESSION ON DATABASE 권한을 부여하는 예이다.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO u1;
Grant succeeded.
```

다음은 SCHEMA s1에 table, view, index, sequence, constraint 객체를 생성할 수 있는 다수의 권한을 user u1에게 부여하는 예이다.

```
gSQL> GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT ON SCHEMA
s1 TO u1;
Grant succeeded.
```

다음은 TABLESPACE mem_data_tbs에 객체를 생성할 수 있는 권한을 user u1에 부여하는 예이다.

```
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
Grant succeeded.
```

다음은 TABLE t1의 일부 column을 조회할 수 있는 권한을 user u1에 부여하는 예이다.

```
gSQL> GRANT SELECT( id, name ) ON TABLE t1 TO u1;
Grant succeeded.
```

다음은 SEQUENCE seq1에 대해 NEXTVAL(), CURRVAL() 함수를 사용할 수 있는 권한을 user u1에 부여하는 예이다.

```
gSQL> GRANT USAGE ON SEQUENCE seq1 TO u1;
Grant succeeded.
```

호환성

SQL 표준에서는 다음 privilege 들을 정의하지 않고 있다.

- <database privilege>
- <tablespace privilege>
- <schema privilege>

표 16-56 SQL 표준 호환성

Feature ID	설명	지원 여부
S023	Basic structured types	X
S024	Enhanced structured types	X
S081	Subtables	X
T211	Basic trigger capability	X

Feature ID	설명	지원 여부
T281	SELECT privilege with column granularity	O
T332	Extended roles	X
F731	INSERT column privileges	O

참조

관련 내용은 다음을 참조한다.

- REVOKE privileges FROM
- CREATE USER
- DROP USER
- ALTER USER

16.114 INSERT INTO

기능

테이블에 새로운 row들을 생성한다.

구문

```

<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
    ;
<insert source> ::=
    <values clause>
    | <from subquery>
    | <from default>
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
  
```

사용 범위 및 접근 권한

<insert statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- INSERT 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Insert 대상이 되는 모든 column에 대해 INSERT(columns) ON TABLE
 - 테이블에 대해 (INSERT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (INSERT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - INSERT ANY TABLE ON DATABASE
- <from subquery>를 사용할 경우, 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE

- 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 생성할 대상 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

[(column_name [, ...])]

테이블의 column 이름이다.

Column 리스트는 생략할 수 있다.

Column의 개수와 <insert source> 값의 개수는 동일해야 하며, 생략된 column에는 DEFAULT 값을 할당한다.

<values clause>

대응하는 column에 할당할 값의 리스트이다.

- <value expression>
 - 대응하는 column에 할당할 값이나 연산식이다.
- DEFAULT
 - 대응하는 column의 값은 CREATE TABLE 구문을 통해 정의한 기본값을 사용한다.
 - 정의하지 않았을 경우 NULL 값이 할당된다.

다음과 같이 다수의 row를 생성할 수 있다.

```
INSERT INTO table_name VALUES ( 1, 'A' ), ( 2, 'B' ), ( 3, 'C' )
```

<from subquery>

Row들을 생성할 질의이다.

자세한 내용은 SELECT 구문의 query expression 절을 참조한다.

DEFAULT VALUES

모든 column들을 기본값으로 채운다.

DEFAULT VALUES 절은 다음과 같은 의미이다.

VALUES (DEFAULT, DEFAULT, ..., DEFAULT)

설명

INSERT 관련 구문들의 차이점

- **INSERT INTO**
 - 테이블에 하나 또는 다수의 row를 생성한다.
 - 예: INSERT INTO t1 SELECT * FROM t1;
- **INSERT INTO name RETURNING**
 - 테이블에 하나 또는 다수의 row를 생성하고, 생성한 row들을 SELECT 구문과 동일한 방식 (SQLFetch() 등의 API)으로 검색할 수 있다.
 - 예: INSERT INTO t1 SELECT * FROM t1 RETURNING c1;
- **INSERT INTO name RETURNING .. INTO**
 - 한 건 이하의 row를 생성할 수 있으며, 생성한 row가 한 건일 경우 RETURNING INTO 절의 호스트 변수에 값을 얻어온다.
 - 예: INSERT INTO t1 DEFAULT VALUES RETURNING c1 INTO :v1;

사용 예

다음은 INSERT 구문을 이용해 row 하나를 생성하는 예이다.

```
gSQL> INSERT INTO region VALUES ( 0, 'AFRICA' );
1 row created.
```

다음은 INSERT 구문에서 column의 DEFAULT 값 또는 identity 값을 사용하는 예이다.

```
gSQL> CREATE TABLE region
(
    r_regionkey    BIGINT    GENERATED BY DEFAULT AS IDENTITY
, r_name         CHAR(25)  DEFAULT 'N/A'
);
Table created.
gSQL> COMMIT;
Commit complete.
```

- 모든 column을 DEFAULT로 입력한다.

```
gSQL> INSERT INTO region DEFAULT VALUES;
1 row created.
```

- 모든 column을 DEFAULT로 입력한다.

```
gSQL> INSERT INTO region VALUES (DEFAULT, DEFAULT);
1 row created.
```

- Column이 생략된 경우 r_name column의 DEFAULT 값을 사용한다.

```
gSQL> INSERT INTO region(r_regionkey) VALUES (-100);
1 row created.
```

- Column이 생략된 경우 r_regionkey column의 identity 값을 사용한다.

```
gSQL> INSERT INTO region(r_name) VALUES ('ASIA');
1 row created.
```

```
gSQL> SELECT * FROM region;
```

```
R_REGIONKEY R_NAME
```

```
-----
```

```
1 N/A
```

```
2 N/A
```

```
-100 N/A
```

```
3 ASIA
```

```
4 rows selected.
```

다음은 VALUES 구문에 다수의 row를 기술하여 생성하는 예이다.

```
gSQL> INSERT INTO region
VALUES ( 1, 'AFRICA' ),
      ( 2, 'ASIA'   ),
      ( 3, 'EUROPE' );
3 rows created.
```

다음은 subquery를 사용하여 다수의 row를 생성하는 예이다.

```
gSQL> INSERT INTO region SELECT r_regionkey, r_name FROM tmp_region WHERE r_regionkey < 3;
3 rows created.
```

호환성

표 16-57 SQL 표준 호환성

Feature ID	설명	지원 여부
F781	Self-referencing operations	X
F222	INSERT statement: DEFAULT VALUES clause	O
S204	Enhanced structured types	X
S043	Enhanced reference types	X
T111	Updatable joins, unions, and columns	X

참조

관련 내용은 다음을 참조한다.

- `SELECT`
- `INSERT INTO name RETURNING`
- `INSERT INTO name RETURNING .. INTO`

16.115 INSERT INTO name RETURNING

기능

테이블에 새로운 row를 생성하고, 생성한 row들을 검색한다.

구문

```

<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <returning clause>
    ;
<insert source> ::=
    <values clause>
    | <from subquery>
    | <from default>
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
<returning clause> ::=
    [ RETURN | RETURNING ] { * | { <value expression> [ [AS] alias_name ] } [, ...]

```

사용 범위 및 접근 권한

<insert returning query statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- INSERT 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Insert 대상이 되는 모든 column에 대해 INSERT(columns) ON TABLE
 - 테이블에 대해 (INSERT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (INSERT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - INSERT ANY TABLE ON DATABASE

- <from subquery>를 사용할 경우, 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE
- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 생성할 대상 테이블의 이름이다.

[(column_name [, ...])]

테이블의 column 이름이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

<values clause>

대응하는 column에 할당할 값의 리스트이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

<from subquery>

Row들을 생성할 질의이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

DEFAULT VALUES

모든 column들을 기본값으로 채운다.

자세한 내용은 INSERT INTO 구문을 참조한다.

<returning clause>

INSERT 된 row들을 반환한다.

- 생성된 row들을 결과 집합으로 하고, 이들 중 검색할 column을 기술한다.
 - RETURNING 절은 INSERT 구문으로 삽입된 row들을 결과 집합으로 하는 결과를 반환한다.
 - <value expression>
 - SELECT 구문의 <select list>와 동일하지만 aggregation 등을 사용할 수 없다.
 - [[AS] alias_name]
 - AS 절을 사용하여 value expression의 이름을 지정할 수 있다.

RETURN과 RETURNING은 동일한 의미의 키워드이다.

설명

자세한 내용은 INSERT 관련 구문들의 차이점을 참조한다.

사용 예

다음은 INSERT 구문으로 생성된 column 값을 검색하는 예이다.

```
gSQL> CREATE TABLE region
(
    r_regionkey    BIGINT    GENERATED BY DEFAULT AS IDENTITY
  , r_name        CHAR(25)  DEFAULT 'N/A'
);
Table created.
gSQL> COMMIT;
Commit complete.
```

- 생성된 DEFAULT 값을 RETURNING 하는 경우

```
gSQL> INSERT INTO region VALUES ( DEFAULT, DEFAULT ) RETURNING r_regionkey, r_name;
R_REGIONKEY R_NAME
-----
          1 N/A
1 row created.
```

- 생략된 column의 값을 RETURNING 하는 경우

```
gSQL> INSERT INTO region(r_name) VALUES ('ASIA') RETURNING r_regionkey;
R_REGIONKEY
-----
                2
1 row created.
```

다음은 subquery로부터 생성된 row들을 검색하는 예이다.

```
gSQL> INSERT INTO region
      SELECT r_regionkey, r_name FROM tmp_region WHERE r_regionkey < 3
      RETURNING r_regionkey, r_name;
R_REGIONKEY R_NAME
-----
0 AFRICA
1 AMERICA
2 ASIA
3 rows created.
```

호환성

SQL 표준에서는 <insert returning query statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- INSERT INTO
- INSERT INTO name RETURNING .. INTO

16.116 INSERT INTO name RETURNING .. INTO

기능

테이블에 row 하나를 생성하고, 생성한 row의 값을 호스트 변수에 얻어온다.

구문

```

<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <returning into clause>
    ;
<insert source> ::=
    <values clause>
    | <from subquery>
    | <from default>
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
<returning into clause> ::=
    [ RETURN | RETURNING ] { * | { <value expression> [ [AS] alias_name ] } [, ...] INTO
    variable_name [, ...]

```

사용 범위 및 접근 권한

<insert returning into statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- INSERT 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Insert 대상이 되는 모든 column에 대해 INSERT(columns) ON TABLE
 - 테이블에 대해 (INSERT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (INSERT TABLE 또는 CONTROL SCHEMA) ON SCHEMA

- INSERT ANY TABLE ON DATABASE
- <from subquery>를 사용할 경우, 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나가 있어야 한다.
 - 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE
- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 생성할 대상 테이블의 이름이다.

[(column_name [, ...])]

테이블의 column 이름이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

<values clause>

대응하는 column에 할당할 값의 리스트이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

<from subquery>

Row들을 생성할 질의이다.

자세한 내용은 INSERT INTO 구문을 참조한다.

DEFAULT VALUES

모든 column들을 기본값으로 채운다.

자세한 내용은 INSERT INTO 구문을 참조한다.

〈returning clause〉

INSERT 된 row를 반환한다.

INSERT INTO name RETURNING 구문의 〈returning clause〉 절을 참조한다.

INTO variable_name [, ...]

INTO 절에 기술된 변수의 개수는 RETURNING 절에 기술된 expression의 개수와 동일해야 한다.

생성할 row가 한 건 이하여야 한다. Row가 두 건 이상 생성될 경우, 에러가 발생한다.

설명

자세한 내용은 INSERT 관련 구문들의 차이점을 참조한다.

사용 예

다음은 생성된 row의 값을 호스트 변수에 얻어오는 예이다.

```
gSQL> CREATE TABLE region
(
    r_regionkey    BIGINT    GENERATED BY DEFAULT AS IDENTITY
  , r_name        CHAR(25)  DEFAULT 'N/A'
);
Table created.
gSQL> COMMIT;
Commit complete.
```

- 호스트 변수들을 선언한다.

```
\VAR v_key    BIGINT
\VAR v_name    VARCHAR(128)
```

- 생성된 DEFAULT 값을 호스트 변수에 얻어오는 경우

```
gSQL> INSERT INTO region
VALUES ( DEFAULT, DEFAULT )
RETURNING r_regionkey, r_name
INTO :v_key, :v_name;
V_KEY V_NAME
```

```
-----
1 N/A
1 row created.
```

- 생략된 column의 값을 호스트 변수에 얻어오는 경우

```
gSQL> INSERT INTO region(r_name)
      VALUES ('ASIA')
      RETURNING r_regionkey
      INTO :v_key;
V_KEY
-----
2
1 row created.
```

호환성

SQL 표준은 <insert returning into statement> 구문을 정의하지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- INSERT INTO
- INSERT INTO name RETURNING

16.117 LOCK TABLE

기능

하나 이상의 테이블에 lock을 설정한다.

구문

```

<lock table statement> ::=
    LOCK TABLE lock target [, ...]
    IN <lock mode> MODE [<wait clause>]
    ;
<lock mode> ::=
    SHARE
    | EXCLUSIVE
    | ROW SHARE
    | ROW EXCLUSIVE
    | SHARE ROW EXCLUSIVE
<wait clause> ::=
    NOWAIT
    | WAIT time
  
```

사용 범위 및 접근 권한

<lock table statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 테이블에 대해 (LOCK 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (LOCK TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- LOCK ANY TABLE ON DATABASE

구문 규칙 및 파라미터

<lock target>

LOCK 대상 테이블을 명시한다.

<lock mode>

LOCK mode를 명시한다.

- SHARE
 - Locked table에 대한 동시성 질의를 허용하지만 테이블 update는 금지한다.
- EXCLUSIVE
 - Locked table에 대한 배타적인 질의 처리를 허용한다.
- ROW SHARE
 - Locked table에 동시성 접근을 허용하지만 exclusive access를 위한 전체 table locking은 금지한다.
- ROW EXCLUSIVE
 - Locked table에 동시성 접근을 허용하지만 exclusive access를 위한 전체 table locking은 금지한다.
 - ROW EXCLUSIVE mode가 설정되어 있는 경우, SHARE mode의 locking은 거부한다.
 - ROW EXCLUSIVE mode는 update, insert, delete 할 때 자동으로 부여된다.
- SHARE ROW EXCLUSIVE
 - Table 전체를 탐색하거나 다른 사용자에게 table의 row들을 탐색하게 할 때 사용한다.
 - SHARE mode의 lock이 부여된 table이나 update되고 있는 row들에 다른 사용자가 접근하지 못하도록 한다.

<wait clause>

Lock을 획득하기 위한 대기 시간을 명시한다.

- NOWAIT
 - 대상에 대한 lock 제어를 즉시 획득한다.
 - 다른 사용자에 의해 이미 lock이 설정된 경우 즉시 제어권을 넘겨받는다.
 - 이 경우 database가 message를 발생시킨다.
- WAIT time
 - Lock을 획득하기 위한 대기 시간을 설정한다.
 - 초 단위이며 0 ~ 10000000000의 값을 사용할 수 있다.
- 명시하지 않을 경우 lock을 획득할 때까지 무기한 WAIT 하도록 한다.

설명

Transaction을 COMMIT 하거나 ROLLBACK 할 경우 획득한 모든 lock은 자동으로 해제된다. ROLLBACK TO SAVEPOINT 구문을 사용할 경우 해당 savepoint 이후에 획득한 모든 lock이 해제된다.

사용 예

다음은 다른 transaction이 TABLE t1에 대해 어떠한 변경 연산도 수행할 수 없도록 하는 예이다.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE;
Table locked.
```

다음은 다수의 table에 LOCK 구문을 수행하는 예이다.

```
gSQL> LOCK TABLE t1, t2 IN EXCLUSIVE MODE;
Table locked.
```

다음은 TABLE t1에 SHARE ROW EXCLUSIVE lock을 획득하는 예이다.

```
gSQL> LOCK TABLE t1 IN SHARE ROW EXCLUSIVE MODE;
Table locked.
```

다음은 해당 TABLE에 즉시 lock을 획득할 수 있을 경우에만 수행할 수 있는 구문이다. Lock을 획득할 수 없을 경우에는 에러가 발생한다.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE NOWAIT;
Table locked.
```

다음은 lock을 획득하기 위해 10 초 동안 대기하도록 하는 예이다.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE WAIT 10;
Table locked.
```

호환성

SQL 표준은 lock table에 대한 개념을 다루지 않고 있다.

참조

관련 내용은 다음을 참조한다.

- COMMIT
- ROLLBACK

16.118 NOAUDIT POLICY

기능

Audit policy를 비활성화한다.

구문

```
<noaudit policy statement> ::=
    NOAUDIT POLICY policy_name
    [ <specified_user_option> ]
    ;
<specified_user_option> ::=
    BY user_name [, ...]
```

사용 범위 및 접근 권한

<noaudit policy statement> 구문을 수행하려면 사용자에게 AUDIT SYSTEM ON DATABASE 권한이 있어야 한다.

구문 규칙 및 파라미터

policy_name

비활성화할 audit policy 객체의 이름이다.

비활성화 된 audit policy는 기존 session에 영향을 미치지 않으며 새로 생성되는 session에만 영향을 준다.

<specified_user_option>

감사 대상에서 제외할 사용자를 명시한다.

AUDIT POLICY 구문과 달리 NOAUDIT POLICY 구문에는 EXCEPT 옵션이 없다.

AUDIT POLICY name BY 절을 사용한 경우 NOAUDIT POLICY name BY 구문으로 비활성화해야 하고

AUDIT POLICY name EXCEPT 절을 사용한 경우 BY 절 없이 NOAUDIT POLICY name 구문으로 비활성화해야 한다.

AUDIT POLICY 구문의 사용 방법에 따라 다음과 같이 NOAUDIT POLICY 구문을 사용하여 해당 옵션을 비활성화해야 한다.

표 16-58 Audit policy 활성화/ 비활성화

유형	AUDIT POLICY 구문	NOAUDIT POLICY 구문
전체 사용자	AUDIT POLICY p1	NOAUDIT POLICY p1
BY를 사용	AUDIT POLICY p1 BY u1	NOAUDIT POLICY p1 BY u1
EXCEPT를 사용	AUDIT POLICY p1 EXCEPT u1	NOAUDIT POLICY p1

활성화된 모든 user들을 비활성화한 경우, audit policy 객체가 완전히 비활성화된다.

설명

Audit policy 객체의 활성화 정보는 다음과 같이 조회한다.

```
SELECT policy_name
       , enabled_opt
       , user_name
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

NOAUDIT POLICY 구문은 AUDIT POLICY 지정 방식에 따라 생성된 개별 활성화 정보를 삭제한다. 위의 질의를 통해 활성화한 정보가 없을 경우, audit policy는 완전히 비활성화된다.

다음과 같이 모든 user를 활성화한 경우, NOAUDIT POLICY BY 절은 영향을 미치지 않는다.

```
AUDIT POLICY p1;
```

- 어떠한 영향도 미치지 않는다.

```
NOAUDIT POLICY p1 BY u1;
```

- 다음과 같이 비활성화해야 한다.

```
NOAUDIT POLICY p1;
```

하나 이상의 user들을 개별적으로 활성화한 경우 AUDIT POLICY 설정 방법에 따라 NOAUDIT POLICY 구문을 사용해야 한다.

BY를 이용해 활성화한 경우

다음과 같이 audit policy를 활성화한 경우,

```
AUDIT POLICY p1 WHENEVER NOT SUCCESSFUL;
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;
```

활성화 정보를 조회하면 다음과 같다.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
P1	BY	ALL USERS	NO	YES
P1	BY	U1	YES	YES
P1	BY	U2	YES	YES

다음은 NOAUDIT 구문을 수행하고 활성화 정보를 조회하는 예이다.

```
NOAUDIT POLICY p1;
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
P1	BY	U1	YES	YES
P1	BY	U2	YES	YES

ALL USERS의 failure에 대한 감사가 비활성화되었으며, u1, u2 사용자에 대한 감사는 여전히 활성화되어 있다.

다음과 같이 BY 옵션을 통해 NOAUDIT POLICY 구문을 추가적으로 사용하면 audit policy p1은 완전히 비활성화된다.

```
NOAUDIT POLICY p1 BY u1, u2;
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
no rows selected.
```

EXCEPT를 이용해 활성화한 경우

다음과 같이 audit policy를 활성화한 경우,

```
AUDIT POLICY p1 EXCEPT u1, sys;
```

활성화 정보를 조회하면 다음과 같다.

```
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

POLICY_NAME	ENABLED_OPT	USER_NAME	WHEN_SUCCESS	WHEN_FAILURE
-----	-----	-----	-----	-----
P1	EXCEPT	U1	YES	YES
P1	EXCEPT	SYS	YES	YES

AUDIT POLICY 구문과 달리 NOAUDIT POLICY 구문에는 EXCEPT option이 없으므로 다음과 같이 옵션 없이 구문을 수행한다.

```
NOAUDIT POLICY p1;
SELECT policy_name
       , enabled_opt
       , user_name
       , when_success
       , when_failure
FROM audit_policy_enabled
WHERE policy_name = 'P1';
```

```
no rows selected.
```

즉, EXCEPT 옵션을 이용해 audit policy를 활성화한 경우, NOAUDIT POLICY 구문으로 개별 사용자를 다시 비활성화할 수 없다.

사용 예

다음은 전체 사용자를 비활성화한 예이다.

```
NOAUDIT POLICY table_pol;
```

다음은 BY를 사용하여 활성화된 특정 사용자를 비활성화하는 예이다.

```
NOAUDIT POLICY table_pol BY u1;
```

호환성

SQL 표준에는 audit policy가 없다.

참조

관련 내용은 다음을 참조한다.

- Audit policy 객체 관리
 - CREATE AUDIT POLICY
 - DROP AUDIT POLICY
 - ALTER AUDIT POLICY
- Audit policy 활성화/ 비활성화
 - AUDIT POLICY
 - NOAUDIT POLICY
- Audit trail 조회: **AUDIT_TRAIL**
- Audit trail 제거: **ALTER DATABASE CLEAR AUDIT TRAIL**

16.119 OPEN cursor_name

기능

커서를 연다.

구문

```

<open statement> ::=
    OPEN cursor_name [ <parameter using clause> ]
    ;
<parameter using clause> ::=
    <using parameter arguments>
<using parameter arguments> ::=
    USING variable_name [, ...]
  
```

사용 범위 및 접근 권한

cursor_name이 **PREPARE statement_name** 구문과 **DECLARE cursor_name** 구문을 사용해 선언한 동적 커서인 경우 embedded SQL에서 사용 가능하다.

cursor_name을 선언한 **DECLARE cursor_name** 구문에 포함된 <cursor query>의 권한과 동일하다.

구문 규칙 및 파라미터

cursor_name

세션 내에서 **DECLARE cursor_name** 구문으로 선언된 커서여야 한다.

<parameter using clause>

Embedded SQL에서 사용할 수 있다.

<parameter using clause> 구문이 사용될 경우, cursor_name이 **PREPARE statement_name** 구문과 **DECLARE cursor_name** 구문을 이용해 선언한 동적 커서여야 한다.

<using parameter arguments>

<using parameter arguments> 구문이 사용될 경우, variable_name의 개수는 **PREPARE statement_name** 구문이 참조하는 query 문장에 포함된 parameter의 개수와 동일해야 한다.

variable_name은 나열된 순서에 따라 dynamic parameter에 순서대로 대응된다.

```
{
  ...
  EXEC SQL PREPARE stmt1 FROM 'SELECT c1, c2 FROM t1 WHERE c1 IN ( ?, ?, ? )';
  EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
  EXEC SQL OPEN cur1 USING :sValue1, :sValue2, :sValue3;
  ...
  EXEC SQL WHENEVER NOT FOUND DO break;
  for(;;)
  {
    EXEC SQL FETCH cur1 INTO :sC1, :sC2;
  }
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  ...
  EXEC SQL CLOSE cur1;
  ...
}
```

설명

Cursor는 session 내에서 구별되는 객체이며, 현재 session 내에서 사용되고 있는 cursor는 다른 session에서 사용되고 있는 cursor와 무관하다.

OPEN cursor_name 구문을 사용하려면 **DECLARE cursor_name** 구문으로 선언된 커서여야 하며, 커서는 닫혀 있는 상태여야 한다.

사용 예

다음은 interactive SQL (gsq)에서 cursor를 선언하고 OPEN cursor 구문을 사용하는 예이다.

```
gsqSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gsqSQL> OPEN cur1;
```

```
Cursor is open.
gSQL> \var v_id    INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
2 data_2
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
3 data_3
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
4 data_4
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
----
5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

호환성

표 16-59 SQL 표준 호환성

Feature ID	설명	지원 여부
B031	Basic Dynamic SQL	O

참조

관련 내용은 다음을 참조한다.

- DECLARE cursor_name
- FETCH cursor_name
- CLOSE cursor_name
- PREPARE statement_name

16.120 PREPARE statement_name

기능

반복 수행을 위한 dynamic SQL 문장을 준비한다.

구문

```

<prepare statement> ::=
    PREPARE statement_name FROM <SQL statement variable>
    ;
<SQL statement variable> ::=
    variable_name
    | 'sql statement'
    | "sql statement"
    | sql statement
  
```

사용 범위 및 접근 권한

Embedded SQL에서 사용할 수 있다.

Dynamic SQL 구문의 종류에 부합하는 수행 권한이 있어야 한다.

구문 규칙 및 파라미터

statement_name

준비할 statement의 이름이다.

statement 이름의 길이는 128 바이트보다 작아야 한다.

이후에 수행될 EXECUTE statement_name 구문 또는 DECLARE cursor_name 구문은 statement_name을 참조한다.

동일한 statement_name이 존재할 경우, 이전에 준비된 dynamic SQL은 삭제된다.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1';
    ...
    EXEC SQL PREPARE stmt1 FROM 'UPDATE t1 SET c1 = c1 + 10';
    ...
}
```

〈SQL statement variable〉

〈SQL statement variable〉은 다음과 같이 네 가지 유형으로 사용된다.

- variable_name: SQL이 저장된 변수
- 'sql statement': Single quote (')로 묶인 SQL 문장
- "sql statement": Double quote (")로 묶인 SQL 문장
- sql statement: Quote 없는 SQL 문장

Single-quoted string 내에 문자열 data를 표현하려면 다음과 같이 single quote (')를 두 번 기술한다.

```
{
    ...
    PREPARE stmt_name FROM 'INSERT INTO t1 VALUES ( ''literal data'' )';
    ...
}
```

〈SQL statement variable〉이 참조하는 dynamic SQL 문장은 host 변수 (:var)나 parameter marker (?)를 사용할 수 있다.

단, quote 없는 SQL 문장을 사용할 경우 parameter marker (?)를 사용할 수 없다.

참조되는 dynamic SQL 문장의 특성에 따라 변수는 input 또는 output dynamic parameter가 된다.

Dynamic SQL 문장 내에 기술된 dynamic parameter는 변수의 이름이 아무 의미가 없으며 종류에 관계없이 구문에 기술된 순서에 따라 식별된다.

- 예제 1

```
{
    ...
    int sValue1;
    int sValue2;
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 BETWEEN ? AND ?';
    EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2;
```

```
...
}
```

- 모든 parameter marker가 input dynamic parameter이다.
- 식별 순서
 - 1번 - BETWEEN ?
 - Input dynamic parameter
 - :sValue1 값을 사용한다.
 - 2번 - AND ?
 - Input dynamic parameter
 - :sValue2 값을 사용한다.
- 예제 2

```
{
    ...
    int sValue1;
    int sValue2;
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT SUM(c2) INTO :v1 FROM t1 WHERE c1 > :v2';
    EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2;
    ...
}
```

- Input dynamic parameter와 output dynamic parameter가 존재한다.
- 식별 순서
 - 1번 - :v1
 - Output dynamic parameter
 - :sValue1에 값이 저장된다.
 - 2번 - :v2
 - Input dynamic parameter
 - :sValue2 값을 사용한다.

variable_name

variable_name에 대응하는 type은 character string이어야 한다.

variable_name에 정의된 dynamic SQL 문장은 유효한 문장이어야 한다.

sql statement

sql statement에 정의된 dynamic SQL 문장은 유효한 문장이어야 한다.

설명

PREPARE statement_name FROM sql_string 구문은 EXECUTE나 cursor를 사용하기 위해 SQL 문을 분석한다. statement_name은 embedded SQL 소스 코드에서 precompiler에게 statement를 알려주는 식별자로서 host variable이 아니기 때문에 별도의 type이나 선언이 필요하지 않다.

자세한 내용은 **Embedded Dynamic SQL**을 참조한다.

사용 예

다음은 embedded SQL 소스 코드 내에서 PREPARE statement_name를 사용하는 예이다.

```
{
    ...
    sprintf( sUpdateSql, "UPDATE EMP SET sal = sal * :v1 WHERE JOB = 'SALES'");
    EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sRatio = 1.1;
    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}
```

PREPARE statement_name이 사용된 전체 소스 코드는 **Dynamic Embedded SQL Example Program**에서 확인할 수 있다.

호환성

표 16-60 SQL 표준 호환성

Feature ID	설명	지원 여부
B031	Basic Dynamic SQL	O
B034	Dynamic specification of cursor attributes	X

참조

관련 내용은 다음을 참조한다.

- EXECUTE statement_name
- DECLARE cursor_name
- EXECUTE IMMEDIATE 'sql_string'
- Embedded Dynamic SQL

16.121 RELEASE SAVEPOINT savepoint_specifier

기능

저장점을 제거한다.

구문

```
<release savepoint statement> ::=  
    RELEASE SAVEPOINT savepoint_name  
    ;
```

구문 규칙 및 파라미터

savepoint_name

저장점의 이름으로써 반드시 존재해야 한다.
이름의 길이는 128 바이트보다 작아야 한다.

설명

다수의 savepoint가 정의되어 있을 경우, RELEASE SAVEPOINT savepoint_name 구문을 수행할 때 savepoint_name 이후에 정의된 savepoint도 함께 제거된다.

사용 예

다음은 savepoint를 제거하는 예이다.

```
gSQL> RELEASE SAVEPOINT sp2;  
Savepoint dropped.
```

호환성

표 16-61 SQL 표준 호환성

Feature ID	설명	지원 여부
T271	Savepoints	O

참조

관련 내용은 다음을 참조한다.

- COMMIT
- ROLLBACK
- SAVEPOINT savepoint_specifier

16.122 REVOKE privileges FROM

기능

사용자에게 부여된 권한을 취소한다.

구문

```

<revoke privilege statement> ::=
    REVOKE [ <revoke option extention> ] <privilege>
        FROM <grantee> [, ...]
        [ <revoke behavior> ]
    ;
<revoke option extention> ::=
    GRANT OPTION FOR
<revoke behavior> ::=
    RESTRICT
    | CASCADE
    | CASCADE CONSTRAINTS

```

구문 규칙 및 파라미터

<privilege>

Revoker (권한을 취소당할 사용자)로부터 취소할 권한이다.

Revoker (구문을 수행하는 사용자)는 다음 조건 중 하나를 만족해야 한다.

- Revoker가 revokee에게 부여한 <privilege>의 경우
 - Revoker가 revokee에게 부여한 <privilege>만 취소한다.
- Revoker가 ACCESS CONTROL ON DATABASE 권한을 소유한 경우
 - 다른 grantor들이 revokee에게 부여한 <privilege>들을 취소한다.

ALL [PRIVILEGES]를 사용하는 경우, 만족하는 <privilege>가 없더라도 성공한다.

<privilege> 종류에 대한 내용은 **GRANT privileges TO** 구문의 <privilege> 절을 참조한다.

〈grantee〉

권한을 취소당할 사용자이다.

- user_identifier
 - 해당 사용자의 권한을 취소한다.
- PUBLIC
 - 모든 사용자를 의미하는 authorization 객체이다.

GRANT OPTION FOR

권한에 포함된 WITH GRANT OPTION을 삭제한다.

Dependent privilege의 WITH GRANT OPTION도 함께 삭제한다.

권한은 그대로 유지된다.

〈revoke behavior〉

- Dependent privilege: WITH GRANT OPTION으로 〈privilege〉를 부여받은 revokee가 다른 사용자에게 부여한 것과 동일한 〈privilege〉이다.
- RESTRICT
 - Dependent privilege가 존재할 경우 revoke 할 수 없다.
- CASCADE
 - Dependent privilege도 함께 revoke 한다.
- CASCADE CONSTRAINTS
 - Dependent privilege도 함께 revoke 한다.
- 생략할 경우, 기본값은 CASCADE 이다.

설명

REVOKE privilege와 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

다음과 같은 DROP 구문을 수행할 경우, 별도로 REVOKE 구문을 수행하지 않더라도 해당 객체와 관련된 모든 권한 정보가 삭제된다.

- SQL schema object 관련 DROP 구문
 - DROP TABLE
 - DROP VIEW
 - DROP SEQUENCE
 - ALTER TABLE name SET UNUSED COLUMN

- DROP FUNCTION
- DROP PROCEDURE
- Non-schema object 관련 DROP 구문
 - DROP SCHEMA
 - DROP TABLESPACE
 - DROP USER

사용 예

다음은 table t1에 대한 다수의 권한을 REVOKE하는 예이다.

```
gSQL> REVOKE INSERT, UPDATE, DELETE, LOCK, ALTER, INDEX ON t1 FROM u1;
Revoke succeeded.
```

다음은 모든 사용자를 의미하는 PUBLIC 계정에 부여된 SELECT ON TABLE t1 권한을 REVOKE 하는 예이다. 단, PUBLIC 계정의 권한만 제거될 뿐 특정 사용자에게 명시적으로 부여된 SELECT ON TABLE t1 권한이 제거되는 것은 아니다.

```
gSQL> REVOKE SELECT ON t1 FROM PUBLIC;
Revoke succeeded.
```

다음은 user u1에게 부여된 SELECT ON TABLE t1 권한은 그대로 두고 다른 사용자에게 해당 권한을 부여할 수 있는 GRANT OPTION만 REVOKE하는 예이다.

```
gSQL> REVOKE GRANT OPTION FOR SELECT ON t1 FROM u1;
Revoke succeeded.
```

다음은 RESTRICT 옵션을 이용해 user u1에게 부여한 권한을 REVOKE 하면서 u1이 다른 사용자에게 해당 권한을 부여할 경우 예러가 발생하는 예이다. 이런 dependent privilege들도 함께 제거하려 할 경우 CASCADE 옵션을 사용한다.

```
gSQL> REVOKE SELECT ON t1 FROM u1 RESTRICT;
ERR-2B000(16235): dependent privilege descriptors still exist
gSQL> REVOKE SELECT ON t1 FROM u1 CASCADE;
Revoke succeeded.
```

호환성

SQL 표준에서는 다음 privilege들을 정의하지 않고 있다.

- <database privilege>
- <tablespace privilege>
- <schema privilege>

SQL 표준의 <revoke behavior>와는 다음과 같은 차이가 있다.

- SQL 표준의 기본값은 RESTRICT 이다.
- SQL 표준에는 CASCADE CONSTRAINTS 가 없다.

표 16-62 SQL 표준 호환성

Feature ID	설명	지원 여부
T311	Basic roles	X
F034	Extended REVOKE statement	X
S081	Subtables	X

참조

관련 내용은 다음을 참조한다.

- GRANT privileges TO
- <database privilege>
- <tablespace privilege>
- <schema privilege>
- <table privilege>
- Column privilege
- <sequence privilege>

16.123 ROLLBACK

기능

트랜잭션을 취소하거나, 저장점 이후의 작업을 취소한다.

구문

```

<rollback statement> ::=
    ROLLBACK [ WORK ] [ <rollback force clause> | <savepoint clause> ]
    ;
<rollback force clause> ::=
    FORCE 'xid_string' [ COMMENT 'comment_string' ]
<savepoint clause> ::=
    TO SAVEPOINT savepoint_name

```

구문 규칙 및 파라미터

WORK

동작에 영향을 미치지 않는 예약어이다.

<rollback force clause>

분산 트랜잭션을 수동으로 rollback 할 때 사용한다.

- FORCE 'xid_string'
 - 'xid_string'에 해당하는 분산 트랜잭션을 rollback 한다.
 - 'xid_string'은 'format_id.transaction_id.branch_id'로 구성된다.
- COMMENT 'comment_string'
 - 분산 트랜잭션을 rollback 할 때 트랜잭션에 주석을 명시한다.

〈savepoint clause〉

현재 트랜잭션의 ROLLBACK 범위를 명시한다.

- 명시하지 않은 경우
 - 현재 트랜잭션의 모든 작업을 취소한다.
 - 트랜잭션을 종료한다.
 - 모든 savepoint들을 제거한다.
 - 모든 lock들을 해제한다.
- TO SAVEPOINT savepoint_name
 - 현재 트랜잭션에서 savepoint_name 이후의 작업을 취소한다.
 - 트랜잭션을 종료하지는 않는다.
 - savepoint_name 이후의 savepoint들을 제거한다.
 - savepoint_name 이후에 획득한 lock들을 해제한다.

설명

ROLLBACK 구문은 트랜잭션 내에서 수행된 다음 구문들을 rollback 한다.

- Data Manipulation Language (DML) 구문
 - 데이터를 변경하는 INSERT, UPDATE, DELETE 등의 구문
- Data Definition Language (DDL) 구문
 - 객체의 구조 및 정의를 변경하는 CREATE, DROP, ALTER, TRUNCATE, GRANT, REVOKE 등의 구문

예외적으로, DDL 중에 OS 자원을 다루거나 DATA TYPE을 변경하는 다음 구문들은 rollback 되지 않고 구문을 수행할 때 자동으로 COMMIT 된다.

- CREATE TABLESPACE
- DROP TABLESPACE
- ALTER TABLESPACE
- ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE: 〈alter column data type clause〉

사용 예

다음은 INSERT 구문을 ROLLBACK 하는 예이다.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );
1 row created.
```



```
gSQL> SELECT * FROM t1;
ID DATA
-- -----
1 anonymous
1 row selected.
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
no rows selected.
```

다음은 DROP TABLE 구문을 수행한 후에 이를 ROLLBACK 하는 예이다.

```
gSQL> DROP TABLE t1;
Table dropped.
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
          *
ERROR at line 1:
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- -----
1 anonymous
1 row selected.
```

호환성

표 16-63 SQL 표준 호환성

Feature ID	설명	지원 여부
T271	Savepoints	O
T261	Chained transactions	X

참조

관련 내용은 다음을 참조한다.

- COMMIT
- SAVEPOINT `savepoint_specifier`

16.124 SAVEPOINT savepoint_specifier

기능

저장점을 정의한다.

구문

```
<savepoint statement> ::=
    SAVEPOINT savepoint_name
    ;
```

구문 규칙 및 파라미터

savepoint_name

저장점 이름이다.

저장점 이름이 기존의 저장점 이름과 중복될 경우 기존의 저장점이 삭제된다.

이름의 길이는 128 바이트보다 작아야 한다.

설명

정의된 savepoint는 ROLLBACK TO SAVEPOINT 구문 (ROLLBACK 구문 참조)에서 사용되며, 해당 savepoint까지 수행된 DML, DDL 구문이 철회되고 해당 구문이 획득한 lock도 해제된다.

정의된 savepoint는 transaction을 COMMIT 하거나 ROLLBACK 할 때 자동으로 제거되는데 **RELEASE SAVEPOINT savepoint_specifier** 구문을 사용하여 명시적으로 제거할 수도 있다.

사용 예

다음은 savepoint를 정의하고 ROLLBACK TO SAVEPOINT 구문을 사용하는 예이다.

```
gSQL> SAVEPOINT sp1;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );
1 row created.
gSQL> SAVEPOINT sp2;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 2, 'someone' );
1 row created.
gSQL> SAVEPOINT sp3;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 3, 'anyone' );
1 row created.
gSQL> SELECT * FROM t1;
ID DATA
-- -----
1 anonymous
2 someone
3 anyone
3 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp3;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- -----
1 anonymous
2 someone
2 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp2;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- -----
1 anonymous
1 row selected.
gSQL> ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
gSQL> SELECT * FROM t1;
no rows selected.
```

호환성

표 16-64 SQL 표준 호환성

Feature ID	설명	지원 여부
T271	Savepoints	O

참조

관련 내용은 다음을 참조한다.

- COMMIT
- ROLLBACK
- RELEASE SAVEPOINT savepoint_specifier

16.125 SELECT

query expression

기능

하나 이상의 table 또는 view에서 원하는 row를 검색한다.

구문

```

<query expression> ::=
    <query expression body> [ <order by clause> ] [ <offset limit clause> ]
<query expression body> ::=
    <query term>
    | <set operator>
<query term> ::=
    <query specification>
    | <left paren> <query expression body> [ <order by clause> ] [ <offset limit clause> ] <
    right paren>
  
```

사용 범위 및 접근 권한

<query expression> 구문을 수행하려면 사용자가 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나를 가져야 한다.

- 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
- 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

<set operator>

부질의 (subquery) 간의 집합 연산을 수행한다.

자세한 내용은 **set operator** 절을 참조한다.

〈query specification〉

하나의 부질의 (subquery)를 기술한다.

자세한 내용은 **query specification** 절을 참조한다.

〈order by clause〉

검색 결과에 대한 정렬 정보를 기술한다.

자세한 내용은 **order by clause**를 참조한다.

〈offset limit clause〉

검색 결과 집합에서 skip 할 row의 개수와 fetch 할 row의 개수를 기술한다.

자세한 내용은 **offset limit clause**를 참조한다.

설명

SELECT 구문을 기술하는 것으로써 〈order by clause〉, 〈offset limit clause〉는 생략할 수 있고 〈set operator〉를 사용하여 둘 이상의 부질의 (subquery)를 가질 수 있다.

사용 예

다음은 SELECT 구문의 예이다.

```
gSQL> SELECT s_name, s_nation FROM supplier;
```

S_NAME	S_NATION
Supplier#1	FRANCE
Supplier#2	KOREA
Supplier#3	GERMANY
Supplier#4	UNITED STATES
Supplier#5	CANADA

5 rows selected.

다음은 〈order by clause〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT s_name, s_nation FROM supplier ORDER BY s_name DESC;
```

S_NAME	S_NATION
Supplier#5	CANADA
Supplier#4	UNITED STATES
Supplier#3	GERMANY
Supplier#2	KOREA

```
Supplier#1          FRANCE
5 rows selected.
```

다음은 <offset limit clause>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT s_name, s_nation FROM supplier OFFSET 1;
S_NAME          S_NATION
-----
Supplier#2      KOREA
Supplier#3      GERMANY
Supplier#4      UNITED STATES
Supplier#5      CANADA
4 rows selected.
gSQL> SELECT s_name, s_nation FROM supplier LIMIT 1;
S_NAME          S_NATION
-----
Supplier#1      FRANCE
1 row selected.
```

다음은 <order by clause>와 <offset limit clause>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT s_name, s_nation FROM supplier ORDER BY s_name DESC OFFSET 3 LIMIT 1;
S_NAME          S_NATION
-----
Supplier#2      KOREA
1 row selected.
```

호환성

표 16-65 SQL 표준 호환성

Feature ID	설명	지원 여부
T121	WITH (excluding RECURSIVE) in query expression	X
T122	WITH (excluding RECURSIVE) in subquery	X
T131	Recursive query	X
T132	Recursive query in subquery	X
F661	Simple tables	O
F302	INTERSECT table operator	O
F301	CORRESPONDING in query expressions	X
T551	Optional key words for default syntax	O
F304	EXCEPT ALL table operator	O
F850	Top-level <order by clause>in <query expression>	X

Feature ID	설명	지원 여부
F851	⟨order by clause⟩in subqueries	O
F855	Nested ⟨order by clause⟩in ⟨query expression⟩	O
F856	Nested ⟨fetch first clause⟩in ⟨query expression⟩	X
F857	Top-level ⟨fetch first clause⟩in ⟨query expression⟩	X
F858	⟨fetch first clause⟩in subqueries	X
F860	dynamic ⟨fetch first row count⟩in ⟨fetch first clause⟩	X
F861	Top-level ⟨result offset clause⟩in ⟨query expression⟩	O
F862	⟨result offset clause⟩in subqueries	O
F863	Nested ⟨result offset clause⟩in ⟨query expression⟩	O
F865	dynamic ⟨offset row count⟩in ⟨result offset clause⟩	X
F866	FETCH FIRST clause: PERCENT option	X
F867	FETCH FIRST clause: WITH TIES option	X

query specification

기능

〈table expression〉 결과로부터 파생된 table을 기술한다.

구문

```

〈query specification〉 ::=
    SELECT [ 〈hint clause〉 ] [ 〈set quantifier〉 ] 〈select list〉 〈table expression〉
〈set quantifier〉 ::=
    ALL
    | DISTINCT
〈table expression〉 ::=
    〈from clause〉 [ 〈where clause〉 ] [ 〈group by clause〉 ] [ 〈having clause〉 ]

```

사용 범위 및 접근 권한

〈query specification〉 구문을 수행하려면 사용자가 다음 조건 중 하나를 만족해야 한다.

- 테이블의 소유자
- 테이블에 대한 SELECT 권한
- 테이블이 속한 스키마에 대해 SELECT TABLE, CONTROL TABLE, CONTROL 권한 중 하나를 소유
- Database에 대한 SELECT TABLE 권한을 소유

구문 규칙 및 파라미터

〈hint clause〉

질의 수행에 필요한 힌트를 기술한다.

자세한 내용은 **hint clause**를 참조한다.

〈set quantifier〉

질의 결과의 중복 제거 여부를 기술한다.

생략할 경우, ALL과 동일하게 동작한다.

〈select list〉

검색 결과 중 결과로 내보낼 expression들을 기술한다.
자세한 내용은 **select list**를 참조한다.

〈from clause〉

검색할 table들을 기술한다.
자세한 내용은 **from clause**를 참조한다.

〈where clause〉

검색 조건을 기술한다.
자세한 내용은 **where clause**를 참조한다.

〈group by clause〉

검색 결과에 대한 grouping을 기술한다.
자세한 내용은 **group by clause**를 참조한다.

〈having clause〉

Grouping 된 결과에 대한 조건을 기술한다.
자세한 내용은 **having clause**를 참조한다.

설명

〈hint clause〉

〈hint clause〉는 optimizer가 선택한 실행 계획보다 더 좋은 실행 계획이 있는 경우 사용자가 optimizer에게 더 좋은 실행 계획을 수행하도록 지시하는 구문이다.

GOLDILOCKS의 optimizer는 사용자가 기술한 〈hint clause〉를 우선 적용한다.
만약 적용할 수 없을 경우에는 cost 계산을 통해 최적의 실행 계획을 선택한다.

GOLDILOCKS는 〈hint clause〉에 구문상 에러가 발생하더라도 이를 무시하고 진행하도록 기본 설정되어 있다. 〈hint clause〉에 구문상 에러가 있는지 확인하려면 "HINT_ERROR" property를 on으로 설정하고 질의를 수행하도록 한다.

〈set quantifier〉

〈set quantifier〉는 〈select list〉에 기술된 expression들로서 구성된 각 row의 중복 제거 여부를 설정하며 각 set quantifier의 의미는 다음과 같다.

- ALL: 결과 집합에서 중복을 제거하지 않는다.
- DISTINCT: 결과 집합에서 중복을 제거한다.

〈set quantifier〉는 생략할 수 있는데 생략할 경우, default로 ALL이 설정된 것과 동일하게 동작한다.

〈select list〉

〈select list〉는 결과 집합의 각 row에서 반환할 expression들의 목록을 기술한다. 이 목록은 콤마 (,) 리스트로 구분하여 기술할 수 있으며, 〈from clause〉에 기술한 모든 column들을 기술하고 싶은 경우에는 별표 (*)를 사용한다.

〈from clause〉

〈from clause〉는 검색할 table 또는 view들을 기술한다.

〈where clause〉

〈where clause〉는 〈table expression〉으로부터 얻은 결과 집합 중에 원하는 결과만 가져오도록 검색 조건을 기술한다.

〈group by clause〉

〈group by clause〉는 〈table expression〉으로부터 얻은 결과 집합을 grouping 할 방법을 기술한다.

〈group by clause〉가 기술된 경우, 〈select list〉에는 group을 결정하는 group key나 group에 속하는 데이터에 대한 집계 함수만 사용할 수 있다.

〈having clause〉

〈having clause〉는 grouping된 결과 집합 중에 원하는 결과만 가져오도록 group에 대한 검색 조건을 기술한다.

일반적으로 〈group by clause〉와 함께 사용되며, 〈group by clause〉 없이 사용된 경우 〈table expression〉에서 반환된 결과 집합 모두가 하나의 group인 〈group by clause〉가 기술된 것과 동일하게 동작한다.

사용 예

다음은 〈hint clause〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT /*+ INDEX_DESC(supplier, supplier_pk_index) */ s_name, s_nation FROM supplier;
```

S_NAME	S_NATION
Supplier#5	CANADA
Supplier#4	UNITED STATES
Supplier#3	GERMANY
Supplier#2	KOREA
Supplier#1	FRANCE

5 rows selected.

다음은 〈set quantifier〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT ALL p_type FROM part;
P_TYPE
-----
COPPER
NICKEL
STEEL
NICKEL
STEEL
5 rows selected.
gSQL> SELECT DISTINCT p_type FROM part;
P_TYPE
-----
COPPER
STEEL
NICKEL
3 rows selected.
```

다음은 <where clause>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT p_name, p_brand, p_type, p_size FROM part where p_size < 10;
P_NAME P_BRAND   P_TYPE P_SIZE
-----
Part#1 Brand#1  COPPER    7
Part#2 Brand#1  NICKEL    1
2 rows selected.
```

다음은 <group by clause>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT ps_partkey, SUM(ps_availqty) FROM partsupp GROUP BY ps_partkey;
PS_PARTKEY SUM(PS_AVAILQTY)
-----
1          11401
2           8025
3          13864
4          11564
5           8744
5 rows selected.
```

다음은 <having clause>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT ps_partkey, SUM(ps_availqty) FROM partsupp GROUP BY ps_partkey having SUM(ps_
availqty) > 10000;
PS_PARTKEY SUM(PS_AVAILQTY)
-----
1          11401
3          13864
4          11564

3 rows selected.
```

호환성

표 16-66 SQL 표준 호환성

Feature ID	설명	지원 여부
F801	Full set function	X
T051	Row types	X
T301	Functional dependencies	X
T325	Qualified SQL parameter references	X
T053	Explicit aliases for all-fields reference	O
T285	Enhanced derived column names	O

참조

관련 내용은 **query expression**을 참조한다.

select list

기능

질의 결과로부터 검색할 column을 기술한다.

구문

```

<select list> ::=
    <asterisk>
  | <select sublist> [ { <comma> <select sublist> } ... ]
<select sublist> ::=
    <derived column>
  | <qualified asterisk>
<qualified asterisk> ::=
    <asterisked identifier chain> <period> <asterisk>
<asterisked identifier chain> ::=
    <asterisked identifier> [ { <period> <asterisked identifier> } ... ]
<derived column> ::=
    <value expression> [ <as clause> ]
<as clause> ::=
    [ AS ] <column name>

```

사용 범위 및 접근 권한

<select list> 구문에 column이나 subquery가 존재할 때 <select list> 구문을 수행하려면 사용자가 column에 대한 접근 권한과 subquery에 존재하는 table 및 column에 대한 접근 권한을 만족해야 한다.

구문 규칙 및 파라미터

<select list>

<asterisk>나 <select sublist>를 갖는다.

<asterisk>는 <select list>에 단독으로만 쓰일 수 있다. 즉, <select sublist>를 기술할 수 없다.

둘 이상의 <select sublist>를 기술할 경우, 각 <select sublist>를 콤마 (,)로 구분해야 한다.

<select sublist>

<derived column>이나 <qualified asterisk>를 갖는다.

테이블 하나당 한 개의 <qualified asterisk>만 가질 수 있다.

<derived column>은 AS를 사용하여 출력 이름을 변경할 수 있으며, AS는 생략할 수 있다.

설명

<select list>

<select list>는 결과 집합에 포함될 column들을 기술한다. <select list>에 <asterisk>를 기술할 경우, <from clause>에 있는 모든 column들을 select list로 설정하며, <select sublist>를 추가로 기술할 수 없다.

<select sublist>

<select sublist>는 <derived column> 또는 <qualified asterisk>를 갖는다. <select sublist>를 둘 이상 기술할 경우에는 반드시 콤마 (,)로 구분해야 한다.

<qualified asterisk>는 특정 table이나 view에 속하는 모든 column들을 select list로 설정하고 <derived column>은 column을 직접 기술하거나 <value expression>을 기술할 수 있다.

<derived column>은 <as clause>를 사용하여 column name을 변경할 수 있으며, 이 때 AS는 생략할 수 있다.

select list에 설정되는 이름

- <derived column>에 <column name>이 명시된 경우, 해당 이름이 select list 이름으로 설정된다.
- <derived column>에 <column name>이 명시되지 않은 경우, 다음과 같이 select list 이름이 설정된다.
 - <derived column>이 single column reference인 경우 single column의 column name이 select list 이름으로 설정된다.
 - <derived column>이 SQL parameter reference인 경우 SQL parameter의 <SQL parameter name>이 select list 이름으로 설정된다.
 - 그 밖의 경우에는 <derived column>의 column name이 설정되지 않는다.
- <from clause>에 둘 이상의 table이 있고 이 중에 이름이 같은 column이 포함되어 있을 때 이 column을 참조하려면 table name이나 table alias를 반드시 명시해야 한다.

사용 예

다음은 <asterisk>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT * FROM supplier;
```

S_SUPPKEY	S_NAME	S_NATION	S_PHONE
1	Supplier#1	FRANCE	27-918-335-1736
2	Supplier#2	KOREA	15-679-861-2259
3	Supplier#3	GERMANY	11-383-516-1199
4	Supplier#4	UNITED STATES	25-843-787-7479
5	Supplier#5	CANADA	21-151-690-3663

5 rows selected.

다음은 <select sublist>를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT revenue.* FROM revenue;
```

```
SUPPLIER_NO TOTAL_REVENUE
```

```
-----
1      11978.64
2      20321.5
3      41844.68
```

3 rows selected.

```
gSQL> SELECT supplier_no suppno, total_revenue AS TOTAL FROM revenue;
```

```
SUPPNO    TOTAL
```

```
-----
1 11978.64
2 20321.5
3 41844.68
```

3 rows selected.

```
gSQL> SELECT 1, revenue.*, CAST( total_revenue AS NATIVE_INTEGER ) TOTAL FROM revenue;
```

```
1 SUPPLIER_NO TOTAL_REVENUE TOTAL
```

```
- - - - -
1      1      11978.64 11979
1      2      20321.5 20322
1      3      41844.68 41845
```

3 rows selected.

호환성

<select list>와 달리 SQL 표준에서는 <all fields reference>를 지원하지 않는다.

참조

관련 내용은 **query specification**을 참조한다.

from clause

기능

하나 이상의 table들로부터 파생된 table을 기술한다.

구문

```

<from clause> ::=
    FROM <table reference list>
<table reference list> ::=
    <table reference> [ { , <table reference> } ... ]
<table reference> ::=
    <table factor>
    | <joined table>
<table factor> ::=
    <table primary>
<table primary> ::=
    <table name> [ <cluster domain> ] [ [ AS ] <correlation name> ]
    | <derived table> [ <cluster domain> ] [ [ AS ] <correlation name> [ <left paren> <derived
column list> <right paren> ] ]
    | <parenthesized joined table>
<derived table> ::=
    <table subquery>
<parenthesized joined table> ::=
    <left paren> <parenthesized joined table> <right paren>
    | <left paren> <joined table> <right paren>
<derived column list> ::=
    <column name list>
<cluster domain> ::=
    @ <cluster domain name>
<cluster domain name> ::=
    GLOBAL
    | LOCAL
    | LOCAL_OFFLINE
    | <identifier>

```

사용 범위 및 접근 권한

〈table reference list〉에 기술한 table 또는 view에 대한 접근 권한이 있어야 한다.

구문 규칙 및 파라미터

〈table reference list〉

- 〈table reference list〉에는 한 개 이상의 테이블들을 콤마 (,)를 사용하여 기술할 수 있다.
- 두 개 이상의 테이블이 기술된 경우
 - 테이블은 왼쪽에서 오른쪽 방향으로 평가 (evaluation)된다.
 - 〈select list〉에 *를 기술한 경우 왼쪽 테이블의 column부터 오른쪽 테이블의 column까지 순서대로 〈select list〉에 매핑된다.

〈table primary〉

〈correlation name〉을 사용하여 별칭 (alias name)을 기술할 수 있다.

〈derived table〉에는 〈table subquery〉가 있고 〈correlation name〉을 이용하여 별칭 (alias name)을 기술할 수 있다.

〈derived table〉에 〈derived column list〉를 기술할 수 있으며, 〈derived column list〉의 〈column name〉 개수는 〈table subquery〉에 기술한 〈select list〉의 target 개수와 동일해야 한다.

〈derived table〉에 〈derived column list〉를 기술한 경우 〈table subquery〉에 기술한 〈select list〉의 target과 순서대로 1 : 1 매핑된다.

〈derived table〉에 〈derived column list〉를 기술한 경우 해당 〈derived table〉내 〈table subquery〉의 〈select list〉를 참조하려면 반드시 〈derived column list〉에 기술한 〈column name〉을 사용해야 한다.

〈correlation name〉

〈table reference list〉에는 동일한 〈correlation name〉이 두 개 이상 존재할 수 없다.

〈table name〉이나 〈derived table〉에 〈correlation name〉이 기술된 경우 해당 〈table name〉이나 〈derived table〉을 참조하려면 반드시 〈correlation name〉을 사용해야 한다.

〈correlation name〉을 기술할 때 그 앞의 AS는 생략할 수 있다.

〈derived column list〉

〈derived column list〉에는 동일한 〈column name〉이 두 개 이상 존재할 수 없다.

〈cluster domain〉

- 〈cluster domain〉은 table이나 view, table subquery가 될 수 있지만 괄호로 묶인 joined table은 〈cluster domain〉이 될 수 없다.
- 구조나 데이터가 변경될 table이나 view에는 〈cluster domain〉을 기술할 수 없다.

- 단, **Cluster**의 **SELECT** 처리가 생성한 데이터 변경 query에는 대상 table에 "@LOCAL"을 기술한다.

<cluster domain name>

Cluster group name이나 cluster member name만이 <cluster domain name>의 <identifier>가 될 수 있다.

설명

<table reference list>

<table reference list>에는 콤마 (,)를 사용하여 두 개 이상의 테이블들을 기술할 수 있다. 두 개 이상의 테이블을 기술하면 해당 테이블들을 왼쪽부터 오른쪽으로 각각 cross join하듯 작동하며 이 때 <where clause>에 두 테이블의 join 조건이 존재하는 경우 해당하는 두 테이블은 <where clause>를 join 조건으로 갖는 inner join처럼 동작한다. 만약 <where clause>에 outer join operator (+)를 사용한 경우, outer join과 동일하게 작동한다.

Outer join operator (+)에 대한 자세한 내용은 **joined table** 절의 outer join operator specification을 참조한다.

<table reference>

단일 table이나 view, table subquery, joined table 등이 <table reference>가 될 수 있다. Joined table을 제외한 나머지는 correlation name을 가질 수 있다.

Joined table에 대한 자세한 내용은 **joined table** 절을 참조한다.

<table primary>

Table이나 view, table subquery, 괄호로 묶인 joined table이 <table primary>가 될 수 있다. Table이나 view, table subquery는 correlation name을 가질 수 있는데, 이 때 AS는 생략할 수 있다. Correlation name이 기술될 경우, <select list>나 <where clause>와 같이 해당 table이나 view, table subquery를 참조하는 모든 경우에 correlation name을 사용해야 한다.

Table subquery는 괄호를 이용하여 <derived column list>에 기술할 수 있으며, correlation name과 마찬가지로 해당 table subquery의 column을 참조하는 모든 경우에 <derived column list>에 기술한 이름을 사용하여야 한다. Table subquery에 <derived column list>를 사용하려면 correlation name을 반드시 기술하여야 한다.

괄호로 묶은 joined table은 join 연산에 참여하는 table들의 논리적 join 순서를 기술한다. 이 때 괄호로 묶은 모든 table들에 대한 join이 모두 cross join과 inner join일 경우, optimizer가 join 순서를 변경할 수 있다.

<cluster domain>

<cluster domain>이 생략된 경우 <cluster domain name>으로 GLOBAL을 사용한 것과 동일한 의미를 갖는다.

<cluster domain name>

<cluster domain name>에 정의된 예약어는 다음과 같은 의미를 가진다.

- GLOBAL
 - 모든 cluster group을 cluster domain으로 선정한다.
- LOCAL
 - 사용자 질의를 수행하는 server만 cluster domain으로 선정한다.
- LOCAL_OFFLINE
 - Domain을 적용할 table이 offline 상태인 경우 사용자 질의를 수행하는 server만 cluster domain으로 선정한다.
 - Online table에 LOCAL_OFFLINE domain을 기술하면 에러가 발생한다.

<cluster domain name>에 <identifier>를 기술한 경우 해당 이름의 cluster group 또는 cluster member를 **Cluster Domain**으로 선정한다.

사용 예

다음은 <table name>을 이용하여 단일 table을 검색하는 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, c_nation FROM customer;
C_NAME      C_NATION
-----
Customer#1  KOREA
Customer#2  CANADA
Customer#3  KOREA
Customer#4  GERMANY
Customer#5  UNITED STATES
5 rows selected.
```

다음은 <derived table>을 이용하는 SELECT 구문의 예이다.

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer);
C_NAME      C_NATION
-----
Customer#1  KOREA
Customer#2  CANADA
Customer#3  KOREA
Customer#4  GERMANY
Customer#5  UNITED STATES
5 rows selected.
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer) AS CUST ("CUSTOMER_NAME", "
CUSTOMER_NATION");
CUSTOMER_NAME CUSTOMER_NATION
-----
Customer#1    KOREA
```

```
Customer#2    CANADA
Customer#3    KOREA
Customer#4    GERMANY
Customer#5    UNITED STATES
5 rows selected.
```

다음은 괄호를 이용한 joined table에 대한 SELECT 구문의 예이다.

```
gSQL> SELECT customer.c_name, o_totalprice FROM (customer INNER JOIN orders ON customer.c_
custkey = orders.o_custkey);
C_NAME      O_TOTALPRICE
-----
Customer#1   173665.47
Customer#2   46929.18
Customer#4   193846.25
Customer#3   32151.78
Customer#5   144659.2
5 rows selected.
```

다음은 콤마 (,)로 구분한 두 개의 table들을 사용하는 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, o_totalprice FROM customer, orders;
C_NAME      O_TOTALPRICE
-----
Customer#1   173665.47
Customer#1   46929.18
Customer#1   193846.25
Customer#1   32151.78
Customer#1   144659.2
Customer#2   173665.47
Customer#2   46929.18
Customer#2   193846.25
Customer#2   32151.78
Customer#2   144659.2
Customer#3   173665.47
Customer#3   46929.18
Customer#3   193846.25
Customer#3   32151.78
Customer#3   144659.2
Customer#4   173665.47
Customer#4   46929.18
Customer#4   193846.25
```

```

Customer#4      32151.78
Customer#4      144659.2
C_NAME          O_TOTALPRICE
-----
Customer#5      173665.47
Customer#5      46929.18
Customer#5      193846.25
Customer#5      32151.78
Customer#5      144659.2
25 rows selected.

```

다음은 <cluster domain>을 이용하는 SELECT 구문의 예이다.

- GLOBAL 예약어 사용

```

gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@GLOBAL);
C_NAME          C_NATION
-----
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.

```

- LOCAL 예약어 사용

```

gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer)@LOCAL;
C_NAME          C_NATION
-----
Customer#1 KOREA
Customer#2 CANADA
2 rows selected.

```

- G1이라는 이름의 cluster group name을 사용

```

gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@G1);
C_NAME          C_NATION
-----
Customer#1 KOREA
Customer#2 CANADA
2 rows selected.

```

- G2N1이라는 이름의 cluster member name을 사용

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@G2N1);
C_NAME      C_NATION
-----
Customer#3  KOREA
Customer#4  GERMANY
2 rows selected.
```

호환성

SQL 표준은 GOLDDILOCKS와 비교하여 다음과 같은 차이가 있다.

- <table factor>에서 <sample clause>를 지원한다.
- <table primary>에서 <lateral derived table>, <collection derived table>, <table function derived table>, <only spec>, <data change delta table>를 지원한다.
 - <lateral derived table>은 <from clause>로 하여금 <lateral derived table>의 subquery 내에서 해당 구문의 왼쪽 테이블들을 참조할 수 있게 한다. (예: select * from dual a, (select * from dual b where a.dummy = b.dummy);)
 - <only spec>은 <from clause>의 hierarchy에 의한 view가 subview의 결과를 포함하지 않도록 한다.
- <table primary>에서 system-versioned table을 지원한다.
- <table primary>에서 <derived column list>를 지원한다.
- <table primary>에서 <transition table name>과 <query name>을 지원한다.
- <derived table>에 대해 <correlation name>이 반드시 존재해야 한다.

참조

관련 내용은 **subquery**를 참조한다.

joined table

기능

Cartesian product, inner join, outer join 등에서 파생되는 table을 기술한다.

구문

```

<joined table> ::=
    <cross join>
  | <qualified join>
  | <natural join>
<cross join> ::=
    <table reference> CROSS JOIN <table factor>
<qualified join> ::=
    <table reference> [ <join type> ] JOIN <table reference> <join specification>
<natural join> ::=
    <table reference> NATURAL [ <join type> ] JOIN <table factor>
<join specification> ::=
    <join condition>
  | <named columns join>
<join condition> ::=
    ON <search condition>
<named columns join> ::=
    USING ( <join column list> )
<join type> ::=
    INNER
  | { LEFT | RIGHT | FULL } [ OUTER ]
<join column list> ::=
    <column name list>

```

사용 범위 및 접근 권한

Joined table에 기술된 모든 table 및 view에 대한 접근 권한이 있어야 한다.

구문 규칙 및 파라미터

〈cross join〉

〈join specification〉은 〈cross join〉 위치에 오지 않으며, 〈cross join〉의 오른쪽에는 단일 테이블이나 〈table subquery〉, 괄호로 묶은 〈joined table〉이 올 수 있다.

〈qualified join〉

- 〈join specification〉을 반드시 기술해야 한다.
- 〈join type〉은 생략할 수 있는데 생략할 경우 INNER로 처리한다.
- 〈join type〉에서 OUTER는 생략할 수 있다.
- 〈join type〉이 OUTER JOIN인 경우 〈join specification〉에는 〈join condition〉만 올 수 있다. (〈named columns join〉이 오는 경우 예러가 발생한다.)

〈natural join〉

- 〈join specification〉이 오지 않으며, 오른쪽에는 단일 테이블이나 〈table subquery〉, 괄호로 묶은 〈joined table〉이 올 수 있다.
- 〈join type〉은 생략할 수 있는데 생략할 경우 INNER로 처리한다.
- 〈join type〉에 OUTER를 지원하지 않는다.
- NATURAL JOIN의 왼쪽 row와 오른쪽 row에 동일한 〈column name〉이 하나도 없는 경우 〈cross join〉으로 처리한다.
- NATURAL JOIN의 왼쪽 row와 오른쪽 row에 동일한 〈column name〉이 있을 경우, USING 구문을 기술한 것과 동일하게 동작한다.
- 대응되는 두 column에 대한 비교 연산이 없으면 예러가 발생한다.

〈join specification〉

- 〈join condition〉이나 〈named columns join〉 중에 하나만 기술할 수 있다.
- 〈named columns join〉을 기술한 경우
 - 〈join column list〉에는 반드시 하나 이상의 column name을 기술해야 한다.
 - Column name은 〈table name〉.〈column name〉과 같이 기술할 수 없다.
 - 〈join column list〉에 나열된 column들은 JOIN의 왼쪽 row와 오른쪽 row에 반드시 존재해야 하며, 비교 가능해야 한다.
 - 〈select list〉에 *를 사용한 경우 〈join column list〉에 나열된 column들의 display column name이 해당 column name으로 출력되며, JOIN의 왼쪽 row와 오른쪽 row에서 해당 column은 제외된다.
 - 〈select list〉에 *를 사용한 경우 target은 〈join column list〉에 기술된 column들, 왼쪽 row들 중에서 〈join column list〉에 해당되지 않는 column들, 오른쪽 row들 중에서 〈join column list〉에 해당되지 않는 column들의 순서로 출력된다.
 - 해당 JOIN 구문의 〈join column list〉에 기술된 column들 중에 왼쪽 row나 오른쪽 row에 해당하는 column은 참조할 수 없다. (즉, 〈join column list〉에 기술된 〈column name〉은 〈table name〉.〈column name〉과 같이 참조할 수 없고, 〈column name〉으로만 참조할 수 있다.)
 - 〈join column list〉에 나열된 column들을 조인 조건으로 처리할 때 각 〈column name〉에 대하여 〈left table name〉.〈column name〉 = 〈right table name〉.〈column name〉 조건이 생성되고 각 〈column name〉에 대한 조건들을 AND로 처리하는 조건이 생성된다.

- <select list>에 특정 테이블의 모든 row를 반환하는 '<table name>.*' 구문을 사용할 수 없다.
- 대응하는 두 column에 대한 비교 연산이 없으면 에러가 발생한다.

설명

<cross join>

<cross join>은 왼쪽의 각 row를 오른쪽의 모든 row들과 결합한 결과를 반환한다.

<cross join>에는 join 조건을 명시적으로 기술할 수 없지만, <where clause>를 통해 두 table에 대한 join 조건을 기술할 수 있으며, 이 경우 inner join과 동일하게 동작한다.

<qualified join>

<qualified join>은 왼쪽의 각 row들을 오른쪽의 모든 row들과 결합한 후 join 조건을 만족하는 row들만 결과로 반환한다. <where clause>가 존재하면 join 구문을 수행할 때 join 조건을 적용한 결과 row들에 <where clause>의 조건들을 적용한다.

Inner join은 <where clause>에 존재하는 조건들을 join 조건처럼 처리해도 결과가 동일하지만, outer join은 <where clause>에 존재하는 조건들을 join 조건처럼 처리하면 결과가 달라지게 된다.

Left outer join은 왼쪽 row에 대하여 join 조건을 만족하는 오른쪽 row가 있을 경우, 해당 row들을 결합한 row를 결과로 반환하며, join 조건을 만족하는 오른쪽 row가 존재하지 않을 경우 왼쪽 row의 값은 그대로 유지하고 오른쪽 row의 값은 모두 NULL로 채운 row를 결과로 반환한다.

Right outer join은 left outer join과 정확히 반대로 동작한다.

Full outer join은 left outer join의 결과와 함께 join 조건을 만족하지 않는 모든 오른쪽 row에 왼쪽 row의 값을 NULL로 채운 row들을 결과로 반환한다.

<natural join>

<natural join>은 join에 참여하는 두 table에서 동일한 이름을 갖는 모든 column들을 각각 equal 조건으로 join 한다. 즉, join에 참여하는 두 table에서 동일한 이름을 갖는 모든 column들을 inner join에서 USING 구문에 기술한 것과 동일하다.

<join specification>

<join condition>은 join 구문의 왼쪽 row와 오른쪽 row를 조인할 조건을 기술하며, <named columns join>은 왼쪽 row와 오른쪽 row에 대해 동일한 <column name>이 존재하는 경우 이를 나열하여 조인 조건을 기술한다.

사용 예

다음은 <cross join>을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, o_totalprice FROM customer CROSS JOIN orders;
```

C_NAME	O_TOTALPRICE
Customer#1	173665.47
Customer#1	46929.18
Customer#1	193846.25
Customer#1	32151.78
Customer#1	144659.2
Customer#2	173665.47
Customer#2	46929.18
Customer#2	193846.25
Customer#2	32151.78
Customer#2	144659.2
Customer#3	173665.47
Customer#3	46929.18
Customer#3	193846.25
Customer#3	32151.78
Customer#3	144659.2
Customer#4	173665.47
Customer#4	46929.18
Customer#4	193846.25
Customer#4	32151.78
Customer#4	144659.2
Customer#5	173665.47
Customer#5	46929.18
Customer#5	193846.25
Customer#5	32151.78
Customer#5	144659.2

25 rows selected.

다음은 inner join을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, o_totalprice FROM customer INNER JOIN orders ON c_custkey = o_custkey;
```

C_NAME	O_TOTALPRICE
Customer#1	173665.47
Customer#2	46929.18
Customer#4	193846.25
Customer#3	32151.78

```
Customer#5      144659.2
5 rows selected.
```

다음은 outer join을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, o_totalprice FROM customer LEFT OUTER JOIN orders ON c_custkey = o_
custkey AND o_orderdate < '1996-01-01';
```

```
C_NAME      O_TOTALPRICE
-----
Customer#1      null
Customer#2      null
Customer#3      32151.78
Customer#4      193846.25
Customer#5      144659.2
```

5 rows selected.

```
gSQL> SELECT c_name, o_totalprice FROM customer RIGHT OUTER JOIN orders ON c_custkey = o_
custkey AND c_nation = 'KOREA';
```

```
C_NAME      O_TOTALPRICE
-----
Customer#1      173665.47
null            46929.18
null            193846.25
Customer#3      32151.78
null            144659.2
```

5 rows selected.

```
gSQL> SELECT c_name, o_totalprice FROM customer FULL OUTER JOIN orders ON c_custkey = o_
custkey AND c_nation = 'KOREA' AND o_orderdate < '1996-01-01';
```

```
C_NAME      O_TOTALPRICE
-----
Customer#1      null
Customer#2      null
Customer#3      32151.78
Customer#4      null
Customer#5      null
null            173665.47
null            46929.18
null            193846.25
null            144659.2
```

9 rows selected.

다음은 natural join을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, o_totalprice FROM (SELECT c_custkey custkey, c_name FROM customer)
NATURAL JOIN (SELECT o_custkey custkey, o_totalprice FROM orders);
```

```
C_NAME      O_TOTALPRICE
```

```
-----
```

```
Customer#1    173665.47
```

```
Customer#2    46929.18
```

```
Customer#4    193846.25
```

```
Customer#3    32151.78
```

```
Customer#5    144659.2
```

```
5 rows selected.
```

호환성

SQL 표준은 GOLDBLOCKS와 비교하여 다음과 같은 차이가 있다.

- <qualified join>과 <natural join>에 대하여 <partitioned join table>을 지원한다.
- Outer join에 대하여 USING 구문을 지원한다.
- <natural join>의 <join type>으로 outer join에 대한 type을 지원한다.
- <join condition>에 <set function specification>을 지원한다.

표 16-67 SQL 표준 호환성

Feature ID	설명	지원 여부
F401	Extended joined table	O
F402	Named column joins for LOBs, arrays, and multisets	X
F403	Partitioned join tables	X

참조

관련 내용은 **from clause**를 참조한다.

where clause

기능

〈from clause〉 결과에 〈search condition〉을 적용한다.

구문

```
〈where clause〉 ::=
    WHERE 〈search condition〉
```

구문 규칙 및 파라미터

〈where clause〉

WHERE 키워드 뒤에는 boolean type을 반환하는 〈search condition〉이 와야 한다.

설명

〈where clause〉에 대한 자세한 내용은 **Conditions**를 참조한다.

사용 예

다음은 〈where clause〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT s_name, s_nation FROM supplier WHERE s_nation = 'KOREA';
S_NAME                S_NATION
-----
Supplier#2            KOREA
1 row selected.
gSQL> SELECT s_name, ps_availqty, ps_supplycost FROM supplier, partsupp WHERE s_nation = '
KOREA' AND s_suppkey = ps_suppkey;
S_NAME                PS_AVAILQTY PS_SUPPLYCOST
-----
Supplier#2            8076        993.49
Supplier#2            4069        357.84
2 rows selected.
```

호환성

표 16-68 SQL 표준 호환성

Feature ID	설명	지원 여부
F441	Extended set function support	O

참조

관련 내용은 `query specification`을 참조한다.

group by clause

기능

이전 구문들이 처리한 결과에 <group by clause>를 적용한 grouped table을 기술한다.

구문

```

<group by clause> ::=
    GROUP BY <grouping element list>
<grouping element list> ::=
    <grouping element> [ { , <grouping element> } ... ]
<grouping element> ::=
    <ordinary grouping set>
    | <empty grouping set>
<ordinary grouping set> ::=
    <grouping column reference>
<grouping column reference> ::=
    <column reference>
    | <value_expression>
<empty grouping set> ::=
    <left paren> <right paren>

```

사용 범위 및 접근 권한

<group by clause>를 수행하기 위해 별도의 접근 권한이 필요한 것은 아니다.

구문 규칙 및 파라미터

<ordinary grouping set>

하나 이상의 <grouping column reference>로 구성한다.

LONG type은 지원하지 않는다.

<empty grouping set>

괄호만 사용하여 기술할 수 있으며, 괄호 안에는 어떠한 expression도 쓸 수 없다.

설명

〈grouping element list〉

〈group by clause〉에 기술된 〈grouping element〉를 순서대로 결합하여 하나의 GROUPING SET로 만드는 grouping을 수행한다. 이 때 GROUPING SET에 존재하는 모든 〈grouping element〉들의 값이 순서대로 일치하는 group은 동일한 group으로 처리한다.

〈group by clause〉가 기술된 경우, 〈select list〉에는 〈group by clause〉에 기술된 column이나 〈group by clause〉에 기술되어 있지 않는 column 중에 집계 함수에 사용되는 column만 올 수 있다.

〈grouping column reference〉

〈grouping column reference〉에는 〈column reference〉나 〈value expression〉이 올 수 있다.

〈column reference〉는 〈query specification〉의 〈from clause〉에 속하는 column들만 참조할 수 있으며, 동일한 column 이름이 존재하는 경우 table 이름 등을 사용하여 column 이름을 명확하게 기술하여야 한다.

〈value expression〉은 〈column reference〉를 포함한 expression이나 〈column reference〉를 포함하지 않는 expression으로 구성될 수 있다. 전자의 경우 〈column reference〉를 사용하여 여러 group으로 구분할 수 있지만, 후자의 경우 〈value expression〉의 값이 모두 동일한 상수값이기 때문에 모든 레코드가 단일 group으로 구성된다.

〈value expression〉에 null 값을 기술하는 경우, null 값들은 동일한 값으로 취급되어 모든 레코드가 단일 group으로 구성된다.

〈empty grouping set〉

〈empty grouping set〉는 grouping 대상 column이 존재하지 않는다는 의미이다. 따라서, 모든 레코드가 단일 group으로 구성된다.

사용 예

다음은 GROUP BY를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_nation, COUNT(c_name) FROM customer GROUP BY c_nation;
```

```
C_NATION      COUNT(C_NAME)
```

```
-----
```

```
UNITED STATES      1
```

```
CANADA             1
```

```
KOREA              2
```

```
GERMANY            1
```

```
4 rows selected.
```

```
gSQL> SELECT COUNT(c_name) FROM customer GROUP BY NULL;
```

```
COUNT(C_NAME)
```

```
-----
```

```
5
1 row selected.
gSQL> SELECT COUNT(c_name) FROM customer GROUP BY ();
COUNT(C_NAME)
-----
5
1 row selected.
```

호환성

표 16-69 SQL 표준 호환성

Feature ID	설명	지원 여부
T431	Extended grouping capabilities	X
T432	Nested and concatenated GROUPING SETS	X
T434	GROUP BY DISTINCT	X

참조

관련 내용은 다음을 참조한다.

- [having clause](#)
- [query specification](#)

having clause

기능

〈search condition〉을 만족하지 않는 group을 제거한 grouped table을 기술한다.

구문

```
〈having clause〉 ::=
    HAVING 〈search condition〉
```

사용 범위 및 접근 권한

〈having clause〉를 수행하기 위해 별도의 접근 권한이 필요한 것은 아니다.

구문 규칙 및 파라미터

〈having clause〉

〈group by clause〉에 기술된 column만 〈search condition〉에 집계 함수 없이 사용할 수 있다.

〈group by clause〉에 기술되지 않은 column은 집계 함수를 사용하여 기술할 수 있다.

설명

〈having clause〉

〈having clause〉는 grouping된 데이터들에 대한 검색 조건을 기술한다. 일반적으로 〈group by clause〉와 같이 사용되며, 〈group by clause〉 없이 〈having clause〉를 사용할 경우, "GROUP BY ()"가 있는 것으로 간주한다.

〈having clause〉에는 〈group by clause〉에 기술된 column만 단독으로 기술할 수 있다. 이 외의 column을 단독으로 기술하려면 집계 함수를 사용해야 한다.

사용 예

다음은 〈having clause〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_nation, COUNT(c_name) FROM customer GROUP BY c_nation HAVING COUNT(c_name) > 1;
C_NATION COUNT(C_NAME)
-----
KOREA                2
```

```
1 row selected.
gSQL> SELECT COUNT(c_name) FROM customer HAVING COUNT(c_name) > 1;
COUNT(C_NAME)
-----
5
1 row selected.
```

호환성

표 16-70 SQL 표준 호환성

Feature ID	설명	지원 여부
T301	Functional dependencies	O

참조

관련 내용은 다음을 참조한다.

- group by clause
- Conditions

order by clause

기능

검색 결과의 정렬 순서를 기술한다.

구문

```

<order by clause> ::=
    ORDER BY <sort specification list>
<sort specification list> ::=
    <sort specification> [ { <comma> <sort specification> }... ]
<sort specification> ::=
    <sort key> [ <ordering specification> ] [ <null ordering> ]
<sort key> ::=
    <value expression>
<ordering specification> ::=
    ASC
    | DESC
<null ordering> ::=
    NULLS FIRST
    | NULLS LAST

```

사용 범위 및 접근 권한

정렬을 위해 기술한 sort key에 column이 존재하는 경우 사용자에게 column에 대한 접근 권한이 있어야 한다.

구문 규칙 및 파라미터

<order by clause>

- <query specification>의 <select list>에 사용되지 않은 column을 <sort key>로 사용하는 경우, <set quantifier> DISTINCT나 하나 이상의 <set function specification>을 기술할 수 없다.
- 단, 다음과 같은 경우에는 DISTINCT 구문이 기술되었어도 생략할 수 있기 때문에 위의 제약 사항이 적용되지 않는다.
 - <group by clause>나 <having clause>가 기술되지 않았으며, 하나 이상의 <set function specification>을 기술한 경우
 - 하나 이상의 중첩된 <set function specification>을 기술한 경우
- <set operator> 구문에서 <order by clause>를 명시한 경우, 가장 먼저 기술된 <query specification>을 기준으로 <sort key>를 분석한다.

〈sort specification list〉

〈ordering specification〉을 명시하지 않은 경우, 기본값은 ASC이다.

〈null ordering〉을 명시하지 않은 경우, 기본값은 NULLS LAST이다.

〈sort key〉

- 〈sort key〉의 〈value expression〉이 scale 0의 양의 정수값이면 그 값을 sort key index로 사용한다.
 - 해당 값에 대응되는 〈query specification〉의 i 번째 〈select sublist〉를 sort key로 사용한다.
 - 해당 값에 대응되는 〈query specification〉의 i 번째 〈select sublist〉가 존재하지 않는 경우 error를 반환한다.
- Row subquery나 relation subquery는 〈value expression〉로 지원되지 않는다.
- 그 외의 〈value expression〉은 sort key로 사용된다.

설명

〈order by clause〉

〈order by clause〉는 검색 결과를 정렬하기 위한 방법을 기술한다. 〈order by clause〉에는 〈sort key〉들을 콤마 (,) 리스트로 나열할 수 있으며, 나열한 순서대로 각 레코드들의 〈sort key〉를 비교하여 순서대로 정렬한다.

〈sort key〉에는 오름차순 정렬이나 내림차순 정렬을 지정하는 〈ordering specification〉을 기술할 수 있는데 생략할 경우에는 오름차순으로 정렬된다. 또한 〈sort key〉에는 NULL 값과 NULL이 아닌 값의 순서를 〈null ordering〉을 사용하여 지정할 수 있는데 생략할 경우에는 NULLS LAST로 정렬된다.

〈sort key〉에 상수값을 기술할 경우 〈select list〉에서 해당 값의 순번에 위치한 expression을 〈sort key〉로 간주한다. 그리고 이 때 기술하는 상수값은 0보다 크면서 〈select list〉에 기술한 expression의 전체 개수와 같거나 작아야 하고 scale은 0이어야 한다.

〈sort key〉에는 LONG type을 기술할 수 없다.

Null value와의 비교

- Null value끼리 비교할 경우에는 동일한 값으로 간주한다.
- Null value와 null value가 아닌 값을 비교할 경우에는 다음 규칙을 따른다.
 - NULLS FIRST이고 ASC인 경우: null value < not null value
 - NULLS LAST이고 ASC인 경우: null value > not null value
 - NULLS FIRST이고 DESC인 경우: null value > not null value
 - NULLS LAST이고 DESC인 경우: null value < not null value
- Null value 비교 결과가 UNKNOWN인 경우, 탐색 순서에 따라 정렬한다.

동일한 sort key 값을 가지는 row들의 정렬

Sort key로 구분할 수 없는 row들을 peer라고 하며, peer들은 탐색 순서에 따라 정렬된다.

〈sort key〉로 사용되는 〈aggregation function〉

〈query specification〉에서 〈aggregation function〉이 사용되거나 〈group by clause〉가 기술된 경우, 〈aggregation function〉을 〈sort key〉로 사용할 수 있다. 단, 〈group by clause〉가 기술된 경우에만 중첩된 〈aggregation function〉을 〈sort key〉로 사용할 수 있다.

사용 예

다음은 ORDER BY를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, c_nation FROM customer ORDER BY c_nation;
C_NAME      C_NATION
-----
Customer#2  CANADA
Customer#4  GERMANY
Customer#1  KOREA
Customer#3  KOREA
Customer#5  UNITED STATES
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer ORDER BY c_nation DESC;
C_NAME      C_NATION
-----
Customer#5  UNITED STATES
Customer#1  KOREA
Customer#3  KOREA
Customer#4  GERMANY
Customer#2  CANADA
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer ORDER BY 2 DESC;
C_NAME      C_NATION
-----
Customer#5  UNITED STATES
Customer#1  KOREA
Customer#3  KOREA
Customer#4  GERMANY
Customer#2  CANADA
5 rows selected.
```


호환성

〈order by clause〉는 SQL 표준과 비교하여 다음과 같은 차이가 있다.

- SQL 표준은 〈sort key〉의 〈value expression〉로 〈column reference〉만 지원한다.
- SQL 표준은 〈sort key〉의 〈value expression〉을 sort key index로 사용하지 않는다.

표 16-71 SQL 표준 호환성

Feature ID	설명	지원 여부
F850	Top-level 〈order by clause〉 in 〈query expression〉	X
F851	〈order by clause〉 in subqueries	O
F852	Top-level 〈order by clause〉 in views	O
F855	Nested 〈order by clause〉 in 〈query expression〉	O

참조

관련 내용은 **query expression**을 참조한다.

offset limit clause

기능

검색 결과 중 skip 할 row의 개수와 fetch 할 row의 개수를 기술한다.

구문

```

<offset limit clause> ::=
    <result offset clause>
    | <fetch limit clause>
    | <result offset clause> <fetch limit clause>
<result offset clause> ::=
    OFFSET <offset row count> [ { ROW | ROWS } ]
<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ <fetch row count> ] [ ROW ONLY | ROWS ONLY ]
<limit clause> ::=
    LIMIT { <fetch row count> | <offset row count> , <fetch row count> | ALL }

```

사용 범위 및 접근 권한

<offset limit clause>는 접근 권한을 필요로 하지 않는다.

구문 규칙 및 파라미터

<result offset clause>

- <offset row count> 값은 0과 같거나 큰 양의 정수이어야 한다.
- ROW와 ROWS는 동일한 의미의 키워드로 생략 가능하다.
- 구문을 생략할 경우 OFFSET 0 ROWS 라는 의미이다.

<fetch limit clause>

- 검색 결과 중 skip 할 row의 개수를 명시한다.
- 구문을 생략할 경우 LIMIT ALL 이라는 의미이다.

〈fetch first clause〉

- Fetch 할 row의 개수를 명시한다.
- 〈limit clause〉와 함께 사용할 수 없다.
- FIRST와 NEXT는 동일한 의미의 키워드로 생략 가능하다.
- 〈fetch row count〉 값은 0 보다 큰 양의 정수이어야 한다.
- ROW ONLY와 ROWS ONLY는 동일한 의미의 키워드로 생략 가능하다.
- 〈fetch row count〉는 생략 가능하며 생략할 경우 그 값은 1이다.

〈limit clause〉

- Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너 뛴 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
- 〈fetch first clause〉와 함께 사용할 수 없다.
- LIMIT 〈fetch row count〉로 사용한 경우
 - 〈fetch row count〉는 0보다 큰 양의 정수이어야 한다.
 - 이 구문은 FETCH FIRST 〈fetch row count〉 ROWS ONLY와 동일한 의미이다.
- LIMIT 〈offset row count〉, 〈fetch row count〉로 사용한 경우
 - 〈result offset clause〉와 동시에 사용할 수 없다.
 - 〈offset row count〉는 0과 같거나 큰 양의 정수이어야 한다.
 - 〈fetch row count〉는 0보다 큰 양의 정수이어야 한다.
 - 이 구문은 OFFSET 〈offset row count〉 ROWS FETCH FIRST 〈fetch row count〉 ROWS ONLY와 동일한 의미이다.
- LIMIT ALL로 사용한 경우 fetch 할 row의 개수에 제한이 없다.

설명

〈result offset clause〉

검색한 결과 중 〈offset row count〉 번째의 row부터 사용자에게 보낸다. 만약 〈offset row count〉가 검색한 결과 row의 개수와 같거나 크면 사용자에게 보내지는 결과의 개수는 0이다.

〈fetch first clause〉

검색한 결과 중에 〈fetch row count〉 개수만큼만 사용자에게 반환한다.

〈limit clause〉

LIMIT 〈fetch_row_count〉로 사용된 경우 검색한 결과 중 〈fetch row count〉 개수만큼만 사용자에게 반환한다.

LIMIT 〈offset row count〉, 〈fetch row count〉로 사용한 경우, 검색한 결과 중 〈offset row count〉 번째의 row부터 〈fetch row count〉 개수만큼만 사용자에게 반환한다.

LIMIT ALL로 사용한 경우 개수 제한없이 검색한 결과를 사용자에게 반환한다.

사용 예

다음은 <result offset clause>을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, c_nation FROM customer OFFSET 1;
C_NAME      C_NATION
-----
Customer#2  CANADA
Customer#3  KOREA
Customer#4  GERMANY
Customer#5  UNITED STATES
4 rows selected.
```

다음은 <fetch first clause>을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, c_nation FROM customer FETCH FIRST ROW ONLY;
C_NAME      C_NATION
-----
Customer#1  KOREA
1 row selected.
gSQL> SELECT c_name, c_nation FROM customer FETCH FIRST 2 ROW ONLY;
C_NAME      C_NATION
-----
Customer#1  KOREA
Customer#2  CANADA
2 rows selected.
```

다음은 <limit clause>을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_name, c_nation FROM customer LIMIT 1;
C_NAME      C_NATION
-----
Customer#1  KOREA
1 row selected.
gSQL> SELECT c_name, c_nation FROM customer LIMIT 1, 2;
C_NAME      C_NATION
-----
Customer#2  CANADA
Customer#3  KOREA
2 rows selected.
gSQL> SELECT c_name, c_nation FROM customer LIMIT ALL;
C_NAME      C_NATION
```

```

-----
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.

```

다음은 <result offset clause>와 <fetch limit clause>를 사용한 SELECT 구문의 예이다.

```

gSQL> SELECT c_name, c_nation FROM customer OFFSET 1 FETCH 2;
C_NAME      C_NATION
-----
Customer#2 CANADA
Customer#3 KOREA
2 rows selected.
gSQL> SELECT c_name, c_nation FROM customer OFFSET 1 LIMIT 2;
C_NAME      C_NATION
-----
Customer#2 CANADA
Customer#3 KOREA
2 rows selected.

```

호환성

SQL 표준은 GOLDBLOCKS와 비교하여 다음과 같은 차이가 있다.

- ROW나 ROWS를 생략할 수 없다.
- FIRST나 NEXT를 생략할 수 없다.
- <fetch first clause>에 <fetch first percentage>를 지원한다.
 - <fetch first percentage>에서 <simple value specification>은 numeric이면 된다. (즉, 소수점 이하도 허용된다.)
 - <fetch first percentage>를 기술한 경우 해당 값을 <fetch row count>로 변환하여 사용하며, 이 때 변환 공식은 다음과 같다.
 - $FRC = \text{CEILING} (FFP * LOCT / 100.0E0)$
 - FFP: <simple value specification>의 값
 - LOCT: 검색 결과의 row 개수
 - FRC: <fetch row count>로 변환된 값
- <fetch first clause>에 WITH TIES를 지원한다.
 - WITH TIES를 기술한 <fetch first clause>를 사용한 경우 반드시 <order by clause>가 존재해야 한다.
 - WITH TIES를 기술한 경우 <order by clause>의 sort key가 모두 동일한 row들의 단위인 peers를 기준으로 <fetch row count> 개수 만큼의 peers들을 반환한다.

노트**OFFSET and LIMIT 구문**

- SQL 표준은 OFFSET .. FETCH {FIRST|NEXT} ... 구문을 정의하고 있다.
- IBM DB2와 Postgres는 SQL 표준과 동일한 구문을 제공한다.
- Postgres와 MySQL은 OFFSET .. LIMIT 구문을 제공한다.
- Oracle은 ROWNUM column을 통해 유사한 기능을 수행할 수 있다.

표 16-72 SQL 표준 호환성

Feature ID	설명	지원 여부
F861	Top-level <result offset clause> in <query expression>	O
F862	<result offset clause> in subqueries	O
F863	Nested <result offset clause> in <query expression>	O
F864	Top-level <result offset clause> in views	O
F865	dynamic <offset row count> in <result offset clause>	X

set operator

기능

부질의 (subquery) 결과들에 대한 집합 (set) 연산을 수행한다.

구문

```

<set operator> ::=
    <set operator term>
  | <query expression body> UNION [ ALL | DISTINCT ] <set operator term>
  | <query expression body> EXCEPT [ ALL | DISTINCT ] <set operator term>
  | <query expression body> MINUS [ ALL | DISTINCT ] <set operator term>
<set operator term> ::=
    <query term>
  | <set operator term> INTERSECT [ ALL | DISTINCT ] <set operator term>

```

사용 범위 및 접근 권한

<set operator> 구문을 사용하려면 각 <set operator term>에 나타나는 <query expression>에 대한 모든 접근 권한이 있어야 한다.

구문 규칙 및 파라미터

<set operator>

- 부질의 (subquery) 간의 집합 연산을 기술한다.
- 각 부질의 (subquery)의 <select list> target 개수가 모두 동일해야 하며, 매칭되는 target들은 모두 동일한 data type group에 속해야 한다.
- 첫 번째 부질의 (subquery)의 <select list> target 이름이 <set operator> 결과 target의 대표 이름이 된다.
- 괄호 등을 사용하여 명확하게 수행 순서를 기술하지 않을 경우, 왼쪽에 기술한 부질의 (subquery)부터 오른쪽에 기술한 부질의 (subquery) 순서로 평가하여 처리한다.
- <set operator>의 각 operator는 다음을 의미한다.
 - UNION
 - UNION ALL: 부질의 (subquery) 결과들에서 중복을 제거하지 않고 합집합한다.
 - UNION DISTINCT: 부질의 (subquery) 결과들에서 중복을 제거하여 합집합한다.
 - ALL/ DISTINCT 중 하나도 기술하지 않을 경우, DISTINCT를 기술한 것과 동일하게 동작한다.
 - EXCEPT
 - EXCEPT ALL: 부질의 (subquery) 결과들에서 중복을 제거하지 않고 차집합한다.

- EXCEPT DISTINCT: 부질의(subquery) 결과들에서 중복을 제거하여 차집합한다.
- ALL/ DISTINCT 중 하나도 기술하지 않을 경우, DISTINCT를 기술한 것과 동일하게 동작한다.
- MINUS
 - EXCEPT의 alias로써 EXCEPT와 동일하게 동작한다.
- INTERSECT
 - INTERSECT ALL: 부질의 (subquery) 결과들에서 중복을 제거하지 않고 교집합한다.
 - INTERSECT DISTINCT: 부질의 (subquery) 결과들에서 중복을 제거하여 교집합한다.
 - ALL/ DISTINCT 중 하나도 기술하지 않을 경우, DISTINCT를 기술한 것과 동일하게 동작한다.

〈query term〉

하나의 부질의 (subquery)를 기술한다.

자세한 내용은 **query expression** 절을 참조한다.

설명

〈set operator〉의 ALL과 DISTINCT의 차이

예를 들어 R1과 R2 table의 데이터가 다음과 같을 경우, 각 〈set operator〉의 결과는 다음과 같다.

- TABLE 데이터
 - R1 TABLE = {1, 1, 1, 2, 2, 2, 3, 4, 4, 5}
 - R2 TABLE = {1, 1, 3, 3, 4}
- SELECT * FROM R1 UNION ALL SELECT * FROM R2;
 - result = {1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5}
- SELECT * FROM R1 UNION DISTINCT SELECT * FROM R2;
 - result = {1, 2, 3, 4, 5}
- SELECT * FROM R1 MINUS ALL SELECT * FROM R2;
 - result = {1, 2, 2, 2, 4, 5}
- SELECT * FROM R1 MINUS DISTINCT SELECT * FROM R2;
 - result = {2, 5}
- SELECT * FROM R1 INTERSECT ALL SELECT * FROM R2;
 - result = {1, 1, 3, 4}
- SELECT * FROM R1 INTERSECT DISTINCT SELECT * FROM R2;
 - result = {1, 3, 4}

그림 2 SET 연산 결과

R1 table	1 1 1 2 2 2 3 4 4 5
R2 table	1 1 3 3 4
R1 UNION ALL R2	1 1 1 1 1 2 2 2 3 3 3 4 4 4 5
R1 UNION R2	1 2 3 4 5
R1 MINUS ALL R2	1 2 2 2 4 5
R1 MINUS R2	2 5
R1 INTERSECT ALL R2	1 1 3 4
R1 INTERSECT R2	1 3 4

연산자 우선 순위

<set operator>의 연산자 우선순위는 다음과 같다.

- 괄호 () 우선
- INTERSECT 우선
- UNION, EXCEPT는 left-right로 기술한 순서 우선

<set operator>의 결과 타입

<set operator> 모든 부질의의 i 번째 column은 동일한 계열의 데이터 타입이어야 하며, 결과 타입 조합 규칙에 따라 결과 타입이 결정된다.

단, LONG VARCHAR와 LONG VARBINARY 타입은 UNION ALL만 사용할 수 있다.

ORDER BY 구문

<set operator>를 ORDER BY와 함께 사용할 때 부질의 간에 column 이름이 다를 경우, 다음과 같이 사용할 수 있다.

- ORDER BY indicator
 - 결과 column의 순서를 기술한다.

```
SELECT c1 FROM t1
UNION ALL
SELECT c2 FROM t2
ORDER BY 1;
```
- ORDER BY left_column_name

- 첫 번째 subquery의 column 이름을 기술한다.

```
SELECT c1 FROM t1
UNION ALL
SELECT c2 FROM t2
ORDER BY c1;
```

사용 예

다음은 UNION 연산을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT s_nation nation FROM supplier UNION ALL SELECT c_nation FROM customer;
NATION
-----
FRANCE
KOREA
GERMANY
UNITED STATES
CANADA
KOREA
CANADA
KOREA
GERMANY
UNITED STATES
10 rows selected.
gSQL> SELECT s_nation nation FROM supplier UNION DISTINCT SELECT c_nation FROM customer;
NATION
-----
UNITED STATES
CANADA
KOREA
GERMANY
FRANCE
5 rows selected.
```

다음은 EXCEPT 연산을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_nation nation FROM customer EXCEPT ALL SELECT s_nation FROM supplier;
NATION
-----
KOREA
1 row selected.
```

```
gSQL> SELECT c_nation nation FROM customer EXCEPT DISTINCT SELECT s_nation FROM supplier;
no rows selected.
```

다음은 INTERSECT 연산을 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT c_nation nation FROM customer INTERSECT ALL SELECT s_nation FROM supplier;
NATION
-----
UNITED STATES
CANADA
KOREA
GERMANY
4 rows selected.
gSQL> SELECT c_nation nation FROM customer INTERSECT DISTINCT SELECT s_nation FROM supplier;
NATION
-----
UNITED STATES
CANADA
KOREA
GERMANY
4 rows selected.
```

호환성

<set operator>는 SQL 표준과 비교하여 다음과 같은 차이가 있다.

- SQL 표준에서는 MINUS를 지원하지 않는다.
- SQL 표준에서는 <set operator>와 ORDER BY indicator를 함께 사용할 수 없다.
- SQL 표준에서는 <set operator>와 ORDER BY column_name이 함께 사용될 경우, 모든 부질의의 column 이름과 동일해야 한다.

표 16-73 SQL 표준 호환성

Feature ID	설명	지원 여부
F302	INTERSECT table operator	O
F301	CORRESPONDING	X
T551	Optional key words for default syntax	O
F304	EXCEPT ALL table operator	O

참조

관련 내용은 **query expression**을 참조한다.

subquery

기능

〈query expression〉에서 파생되는 scalar value, row, table 등을 기술한다.

구문

```

<scalar subquery> ::=
    <subquery>
<row subquery> ::=
    <subquery>
<table subquery> ::=
    <subquery>
<subquery> ::=
    ( <query expression> )

```

사용 범위 및 접근 권한

〈subquery〉에 존재하는 〈query expression〉에 대한 접근 권한이 있어야 한다.

구문 규칙 및 파라미터

〈scalar subquery〉

- 〈query expression〉에 존재하는 target의 개수는 한 개이어야 한다.
- 〈query expression〉에서 반환된 row의 개수에 따른 결과값은 다음과 같다.
 - 0 개의 row가 반환된 경우, 결과값은 NULL 이다.
 - 한 개의 row가 반환된 경우, 결과값은 해당 row에 포함된 결과값이다.
 - 두 개 이상의 row가 반환된 경우, exception error가 발생한다.

〈row subquery〉

- 〈query expression〉에 존재하는 target의 개수는 두 개 이상이어야 한다.
- 〈query expression〉에서 반환된 row의 개수에 따른 결과값은 다음과 같다.
 - 0 개의 row가 반환된 경우, 결과값은 모든 column이 NULL인 row이다.
 - 한 개의 row가 반환된 경우, 결과값은 해당 row이다.
 - 두 개 이상의 row가 반환된 경우, exception error가 발생한다.

〈table subquery〉

- 〈query expression〉에 존재하는 target의 개수는 한 개 이상이어야 한다.
- 〈query expression〉에서 반환된 row의 개수에 따른 결과값은 다음과 같다.
 - 0 개의 row가 반환된 경우, 결과값은 no rows이다.
 - 한 개 이상의 row가 반환된 경우, 결과값은 해당 row이다.

설명

〈scalar subquery〉

〈scalar subquery〉는 결과값으로 한 개의 column을 갖는 한 개의 row를 반환하는 subquery이다. 〈scalar subquery〉의 target은 하나만 존재해야 하며, 결과의 data type은 target의 data type을 따른다.

〈scalar subquery〉는 〈select list〉의 target에 단독으로 쓰일 수 있으며, 단일 column만 갖는 연산자에 쓰일 수 있다.

〈row subquery〉

〈row subquery〉는 결과값으로 두 개 이상의 column을 갖는 한 개의 row를 반환하는 subquery이다. 〈row subquery〉의 target은 두 개 이상 존재해야 하며, 결과의 data type은 target들 각각의 data type을 따른다.

〈row subquery〉는 〈select list〉의 target에 단독으로 쓰일 수 없으며, 둘 이상의 column을 갖는 row 연산자에만 쓰일 수 있다.

〈table subquery〉

〈table subquery〉는 결과값으로 한 개 이상의 column을 갖는 한 개 이상의 row를 반환하는 subquery이다. 〈table subquery〉의 target은 한 개 이상 존재해야 하며, 결과의 data type은 target들 각각의 data type을 따른다.

〈table subquery〉는 〈select list〉의 target에 단독으로 쓰일 수 없으며, IN, NOT IN, EXISTS, NOT EXISTS, quantify operator 등의 연산자에 쓰일 수 있다.

사용 예

다음은 〈scalar subquery〉를 사용한 SELECT 구문의 예이다.

```
gSQL> SELECT (SELECT c_name FROM dual) FROM customer;
(SELECT C_NAME FROM DUAL)
-----
Customer#1
Customer#2
Customer#3
Customer#4
```

```

Customer#5
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer WHERE c_nation = (SELECT 'CANADA' FROM dual);
C_NAME      C_NATION
-----
Customer#2 CANADA
1 row selected.

```

다음은 <row subquery>를 사용한 SELECT 구문의 예이다.

```

gSQL> SELECT p_name, p_brand, p_type FROM part WHERE (p_brand, p_type) = (SELECT 'Brand#1', '
NICKEL' FROM dual);
P_NAME P_BRAND    P_TYPE
-----
Part#2 Brand#1    NICKEL
1 row selected.

```

다음은 <table subquery>를 사용한 SELECT 구문의 예이다.

```

gSQL> SELECT s_name, s_nation FROM supplier WHERE s_nation IN (SELECT c_nation FROM customer);
S_NAME      S_NATION
-----
Supplier#2   KOREA
Supplier#3   GERMANY
Supplier#4   UNITED STATES
Supplier#5   CANADA
4 rows selected.
gSQL> SELECT * FROM (SELECT s_name, s_nation FROM supplier);
S_NAME      S_NATION
-----
Supplier#1   FRANCE
Supplier#2   KOREA
Supplier#3   GERMANY
Supplier#4   UNITED STATES
Supplier#5   CANADA
5 rows selected.

```

호환성

표 16-74 SQL 표준 호환성

Feature ID	설명	지원 여부
F471	Scalar subquery values	O
F641	Row and table constructors	X
T501	Enhanced EXISTS predicate	O
E061-11	Subqueries in IN predicate	O
E061-12	Subqueries in quantified comparison predicate	O
E061-12	Correlated subqueries	O

참조

관련 내용은 다음을 참조한다.

- from clause
- where clause

hint clause

기능

Query를 수행할 때 사용할 hint를 기술한다.

구문

```

<hint clause> ::=
    /*+ <hint element> [ comment ] [ [ , ] <hint element> [ comment ] ] */
<hint element> ::=
    <access path hints>
  | <join order hints>
  | <join operation hints>
  | <query transformation hints>
  | <other hints>
<access_path_hints> ::=
    FULL( table_name )
  | INDEX( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
  | NO_INDEX( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
  | INDEX_ASC( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
  | INDEX_DESC( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
  | INDEX_COMBINE( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
  | ROWID( table_name )
<join order hints> ::=
    ORDERED
  | ORDERING( table_name [ LEFT | RIGHT ] [ , table_name [ LEFT | RIGHT ] ] )
  | LEADING( table_name [ [ , ] table_name ] )
<join operation hints> ::=
    USE_HASH( table_name [ [ , ] table_name ] )
  | NO_USE_HASH( table_name [ [ , ] table_name ] )
  | USE_MERGE( table_name [ [ , ] table_name ] )
  | NO_USE_MERGE( table_name [ [ , ] table_name ] )
  | USE_NL( table_name [ [ , ] table_name ] )
  | NO_USE_NL( table_name [ [ , ] table_name ] )
  | USE_INL( table_name [ [ , ] table_name ] )
  | NO_USE_INL( table_name [ [ , ] table_name ] )
<query transformation hints> ::=
    UNNEST
  | NO_UNNEST

```

```

| NL_SJ
| NL_ISJ
| MERGE_SJ
| HASH_SJ
| HASH_ISJ
| HASH_AJ
| TRANSITIVE_CLOSURE
| NO_TRANSITIVE_CLOSURE
| MERGE( view_name )
| NO_MERGE( view_name )
| NO_QUERY_TRANSFORMATION
<other hints> ::=
    PUSH_SUBQ
| NO_PUSH_SUBQ

```

사용 범위 및 접근 권한

<hint clause> 구문을 수행하려면 사용자에게 query를 수행할 수 있는 권한이 필요하다.

구문 규칙 및 파라미터

<hint clause> 사용의 기본 구문 규칙은 다음과 같다.

- <hint clause>에는 공백이나 ','를 이용하여 다수의 <hint element>를 기술할 수 있다.
- 동일한 object에 대한 <hint element>가 둘 이상이고 동시에 적용할 수 없는 경우, 먼저 기술된 <hint element> 하나만 적용된다.
- <hint element>에 구문상 에러가 발생하면 기본적으로 해당 <hint element>를 무시하며, "hint_error" property를 on으로 하면 <hint clause>에 대한 validation error로 처리된다.
- <hint clause>에 기술된 table name은 <from clause>에 기술한 table name (alias name가 기술된 table의 경우 alias name) 중 하나와 일치하여야 한다.
- Table name은 schema name과 함께 기술할 수 없다.
- <hint element>가 올바르게 기술되었더라도 그것을 적용할 수 없는 경우에는 optimizer가 해당 <hint element>를 무시한다.

<access_path_hints>

FULL

Optimizer로 하여금 기술한 table에 대하여 table full scan 하도록 지시한다. 이 hint가 기술된 경우, optimizer는 해당 table에 대하여 index scan을 이용한 최적화나 rowid scan을 이용한 최적화, index combine을 이용하는 최적화 등을 고려하지 않는다.

FULL hint를 기술할 때 table name을 반드시 기술해야 하는데 table name은 하나만 기술할 수 있다. 또한 기술된 t

able name은 <from clause>에 반드시 존재해야 한다.

다음은 T1 table에 대하여 table full scan 하도록 하는 hint를 적용하는 예이다.

- 유형 1: <from clause>에 table name을 기술한 경우

```
SELECT /*+ FULL(T1) */ I1
FROM T1;
```

- 유형 2: <from clause>에 alias name을 기술한 경우

```
SELECT /*+ FULL(T1_ALIAS) */ I1
FROM T1 T1_ALIAS;
```

INDEX

Optimizer로 하여금 기술한 table에 대하여 index scan 하도록 지시한다. 이 hint가 기술된 경우, optimizer는 해당 table에 대하여 table scan을 이용한 최적화나 다른 index에 의한 index scan을 이용한 최적화, rowid scan을 이용한 최적화, index combine을 이용한 최적화 등을 고려하지 않는다.

INDEX hint를 기술할 때는 table name이 <from clause>에 존재해야 하며, index name도 해당 table에 존재하는 index의 이름이어야 한다.

Index name은 하나 이상 기술하거나 생략할 수도 있다. Index name을 생략한 경우에는 해당 table에 속한 모든 index들을 대상으로 한다.

만약 index name을 둘 이상 나열하거나 index가 둘 이상인 table에 대하여 index name을 생략한 경우에는 optimizer가 해당 index들의 index scan cost를 계산하여 최적의 index scan 방법을 선택한다.

다음은 T1 table을 index scan 하도록 hint를 적용한 예이다.

- 유형 1: index name을 하나만 기술한 경우

```
SELECT /*+ INDEX(T1, T1_PK_INDEX) */ I1
FROM T1;
```

- 유형 2: index name을 둘 이상 기술한 경우

```
SELECT /*+ INDEX(T1, T1_PK_INDEX T1_UNIQUE_INDEX) */ I1
FROM T1;
```

- 유형 3: index name을 생략한 경우

```
SELECT /*+ INDEX(T1) */ I1
FROM T1;
```

NO_INDEX

Optimizer로 하여금 기술한 table에서 index name에 해당하는 index들을 index scan 하지 않도록 지시한다. 이 hint가 기술되면 optimizer는 해당 table에 대하여 기술된 index들에 대해 index scan을 이용하여 최적화하는 것을 고려하지 않는다.

NO_INDEX hint를 기술할 때 table name은 <from clause>에 존재해야 하며, index name도 해당 table에 존재하는 index의 이름이어야 한다.

Index name은 하나 이상 기술하거나 생략할 수도 있으며, index name을 생략한 경우에는 optimizer가 해당 table에 대한 index scan을 고려하지 않는다.

NO_INDEX hint에 기술되지 않은 index들이 존재할 경우, optimizer가 해당 index들의 index scan cost를 계산하고, table scan cost와 rowid scan cost들까지 고려하여 최적의 scan 방법을 선택한다.

다음은 T1 table에 대하여 NO_INDEX hint를 적용한 예이다.

- 유형 1: index name을 하나만 기술한 경우

```
SELECT /*+ NO_INDEX(T1, T1_PK_INDEX) */ I1
FROM T1;
```

- 유형 2: index name을 둘 이상 기술한 경우

```
SELECT /*+ NO_INDEX(T1, T1_PK_INDEX T1_UNIQUE_INDEX) */ I1
FROM T1;
```

- 유형 3: index name을 생략한 경우

```
SELECT /*+ NO_INDEX(T1) */ I1
FROM T1;
```

INDEX_ASC

Optimizer로 하여금 기술한 table에서 ascending index scan 하도록 지시한다. 이 hint가 기술되면 optimizer는 해당 table에 대하여 table scan을 이용한 최적화나 다른 index에 의한 index scan을 이용한 최적화, rowid scan을 이용한 최적화, index combine을 이용한 최적화 등을 고려하지 않는다.

만약 선택된 index scan의 index가 ascending order로 구성되어 있다면 ascending order로, index가 descending order로 구성되어 있다면 descending order로 index를 scan한다.

INDEX_ASC hint에 대한 구문 규칙은 INDEX hint와 동일하다.

INDEX_DESC

Optimizer로 하여금 기술한 table에서 descending index scan 하도록 지시한다. 이 hint가 기술되면 optimizer는 해당 table에 대하여 table scan을 이용한 최적화나 다른 index에 의한 index scan을 이용한 최적화, rowid scan을

이용한 최적화, index combine을 이용한 최적화 등을 고려하지 않는다.

만약 선택된 index scan의 index가 ascending order로 구성되어 있다면 descending order로, index가 descending order로 구성되어 있다면 ascending order로 index를 scan한다.

INDEX_DESC hint에 대한 구문 규칙은 INDEX hint와 동일하다.

INDEX_COMBINE

Optimizer로 하여금 기술한 table에 대해 or 구문을 분리하여 각각 index scan을 수행한 후 결과들을 합치도록 지시한다. 이 hint가 기술되면 index combine을 이용한 최적화를 우선적으로 고려하며, 만약 index combine이 가능하지 않을 경우, table scan이나 index scan, rowid scan 등에 대한 cost를 계산하여 최적의 scan 방법을 선택한다.

INDEX_COMBINE hint를 기술할 때 table name이 <from clause>에 존재해야 하며, index name도 해당 table에 존재하는 index의 이름이어야 한다.

Index name은 하나 이상 기술할 수 있고 생략도 가능하다. Index name을 생략할 경우에는 해당 table에 속한 모든 index들을 대상으로 한다.

INDEX_COMBINE hint를 수행하려면 해당 table을 scan하기 위한 조건에 or 구문이 반드시 존재해야 한다. 만약 or 구문이 존재하지 않으면 optimizer가 해당 hint를 무시하며, 이 경우 table scan, index scan, rowid scan에 대한 cost를 계산하여 최적의 scan 방법을 선택한다.

Index name을 둘 이상 나열하거나 index가 둘 이상인 table에 대하여 index name을 생략할 경우 optimizer가 각 or 구문에 대하여 해당 index들의 index scan cost를 계산하여 최적의 index scan을 선택한다. 따라서 or 구문으로 분리된 조건들에 의해 각각 다른 index를 사용하는 index scan을 선택할 수 있다.

다음은 T1 table을 index combine 하도록 hint를 적용한 예이다.

- 유형 1: index name을 하나만 기술한 경우

```
SELECT /*+ INDEX_COMBINE(T1, T1_PK_INDEX) */ I1
FROM T1
WHERE I1 = 1
       OR I1 = 2;
```

- 유형 2: index name을 둘 이상 기술한 경우

```
SELECT /*+ INDEX_COMBINE(T1, T1_PK_INDEX T1_UNIQUE_INDEX) */ I1
FROM T1
WHERE I1 = 1
       OR I2 = 2;
```

- 유형 3: index name을 생략한 경우

```
SELECT /*+ INDEX_COMBINE(T1) */ I1
  FROM T1
 WHERE I1 = 1
        OR I2 = 2;
```

- 유형 4: INDEX_COMBINE hint를 적용할 수 없는 경우 (or 구문이 없는 경우)

```
SELECT /*+ INDEX_COMBINE(T1, T1_PK_INDEX) */ I1
  FROM T1
 WHERE I1 = 1;
```

ROWID

Optimizer로 하여금 기술한 table에 대하여 rowid scan 하도록 지시한다. 이 hint가 기술되면 rowid scan을 이용한 최적화를 우선적으로 고려하며, 만약 rowid scan이 가능하지 않을 경우, table scan이나 index scan, index combine 등에 대한 cost를 계산하여 최적의 scan 방법을 선택한다.

ROWID hint를 기술할 때 table name을 반드시 기술해야 하는데 table name은 하나만 기술할 수 있다. 또한 기술된 table name은 반드시 <from clause>에 존재해야 한다.

ROWID hint를 수행하려면 해당 table을 scan하기 위한 조건에 ROWID를 이용한 equal 조건이 반드시 존재해야 한다. 만약 존재하지 않으면 optimizer가 해당 hint를 무시하며, 이 경우 table scan과 index scan, index combine에 대한 cost를 계산하여 최적의 scan 방법을 선택한다.

다음은 T1 table을 rowid scan 하도록 hint를 적용한 예이다.

- 유형 1: <from clause>에 table name을 기술한 경우

```
SELECT /*+ ROWID(T1) */ I1
  FROM T1
 WHERE ROWID = 'AAAAAAAAADXAACAAAGA1AAA';
```

- 유형 2: <from clause>에 alias name을 기술한 경우

```
SELECT /*+ ROWID(T1_ALIAS) */ I1
  FROM T1 T1_ALIAS
 WHERE ROWID = 'AAAAAAAAADXAACAAAGA1AAA';
```

- 유형 3: ROWID hint를 적용할 수 없는 경우 (rowid 조건이 없는 경우)

```
SELECT /*+ ROWID(T1) */ I1
  FROM T1
 WHERE I1 = 1;
```

<join_order_hints>

ORDERED

Optimizer로 하여금 <from clause>에 기술한 순서대로 table들을 join 하도록 지시한다. 이 hint는 <from clause>에 ','로 구분된 table들을 기술하거나 inner join으로 table들을 기술하는 경우에만 적용할 수 있다.

다음은 T1, T2 table에 대한 join에 ORDERED hint를 적용한 예이다.

- 유형 1: <from clause>에 ','로 구분된 table들을 기술한 경우

```
SELECT /*+ ORDERED */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 2: <from clause>에 inner join으로 table들을 기술한 경우

```
SELECT /*+ ORDERED */ *
  FROM T1 INNER JOIN T2 ON T1.I1 = T2.I1;
```

- 유형 3: ORDERED hint를 적용할 수 없는 경우 (outer join인 경우)

```
SELECT /*+ ORDERED */ *
  FROM T1 LEFT OUTER JOIN T2 ON T1.I1 = T2.I1;
```

ORDERING

Optimizer로 하여금 이 hint에 기술된 table들을 순서대로 join 하도록 지시한다. 이 hint는 <from clause>에 ','로 구분된 table들을 기술하거나 inner join으로 table들을 기술하는 경우에만 적용할 수 있다.

ORDERING hint는 각 table에 위치 지정 옵션을 기술할 수 있는데 첫 번째와 두 번째 table에 대해서는 기술할 수 없고 세 번째 table부터 기술할 수 있다. 첫 번째와 두 번째 table의 위치는 ORDERING hint에 기술된 순서에 따라 결정된다. 위치 지정 옵션에는 LEFT와 RIGHT가 있는데 LEFT는 해당 table을 join의 left node (outer node)에 위치하도록 하며, RIGHT는 해당 table을 join의 right node (inner node)로 위치하도록 한다.

만약 table에 위치 지정 옵션을 기술한 경우, 해당 table은 기술된 위치에 고정된 채로 join을 수행한다. Table에 위치 지정 옵션을 기술하지 않은 경우, optimizer가 해당 table을 left node (outer node)에 배치하는 경우와 right node (inner node)에 배치하는 경우에 대해 각각의 cost를 계산하여 최적의 join 순서를 선택한다.

다음은 T1, T2, T3 table들의 join에 대하여 ORDERING hint를 적용한 예이다.

- 유형 1: <from clause>에 ','로 구분된 table들을 기술한 경우

```
SELECT /*+ ORDERING(T2, T3, T1) */ *
  FROM T1, T2, T3
 WHERE T1.I1 = T2.I1
```

```
AND T2.I2 = T3.I2;
```

- 유형 2: <from clause>에 inner join으로 table들을 기술한 경우

```
SELECT /*+ ORDERING(T2, T3, T1) */ *
FROM (T1 INNER JOIN T2 ON T1.I1 = T2.I1) INNER JOIN T3 ON T2.I2 = T3.I2;
```

- 유형 3: ORDERING hint에 위치 지정 옵션을 기술한 경우

```
SELECT /*+ ORDERING(T2, T3, T1 LEFT) */ *
FROM T1, T2, T3
WHERE T1.I1 = T2.I1
AND T2.I2 = T3.I2;
```

- 유형 4: ORDERING hint를 적용할 수 없는 경우 (outer join인 경우)

```
SELECT /*+ ORDERING(T1, T2) */ *
FROM T1 LEFT OUTER JOIN T2 ON T1.I1 = T2.I1;
```

- 유형 5: ORDERING hint에서 위치 지정 옵션을 잘못 사용한 경우 (첫 번째 table에 사용한 경우)

```
SELECT /*+ ORDERING(T2 RIGHT, T3, T1) */ *
FROM T1, T2, T3
WHERE T1.I1 = T2.I1
AND T2.I2 = T3.I2;
```

LEADING

Optimizer로 하여금 이 hint에 기술된 table들을 순서대로 join 하도록 지시한다. 이 hint는 <from clause>에 ','로 구분된 table들을 기술하거나 inner join으로 table들을 기술하는 경우에만 적용할 수 있다.

LEADING hint는 ORDERING hint와 달리 위치를 지정할 수 없기 때문에 join에 참여하는 table 순서만 지정할 수 있다. 따라서 첫 번째와 두 번째 table은 순서에 따라 각각 left node (outer node)와 right node (inner node)에 배치되며, 세 번째 table부터는 optimizer가 해당 table을 left node (outer node)에 배치하는 경우와 right node (inner node)에 배치하는 경우의 cost를 계산하여 둘 중 더 좋은 위치에 배치한다.

다음은 T1, T2, T3 table들의 join에 대하여 LEADING hint를 적용한 예이다.

- 유형 1: <from clause>에 ','로 구분된 table들을 기술한 경우

```
SELECT /*+ LEADING(T2, T3, T1) */ *
FROM T1, T2, T3
WHERE T1.I1 = T2.I1
AND T2.I2 = T3.I2;
```


- 유형 2: <from clause>에 inner join으로 table들을 기술한 경우

```
SELECT /*+ LEADING(T2, T3, T1) */ *
FROM (T1 INNER JOIN T2 ON T1.I1 = T2.I1) INNER JOIN T3 ON T2.I2 = T3.I2;
```

- 유형 3: LEADING hint를 적용할 수 없는 경우 (outer join인 경우)

```
SELECT /*+ LEADING(T1, T2) */ *
FROM T1 LEFT OUTER JOIN T2 ON T1.I1 = T2.I1;
```

<join operation_hints>

USE_HASH

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 hash join 기법으로 join 하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

USE_HASH hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_MERGE, USE_NL, USE_INL 등의 hint에 기술되면 안된다. 기술하는 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 대하여 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

USE_HASH hint에 기술된 table이 join에 참여하는 경우 join 조건에 hash join이 가능한 조건 (equi-join이어야 하고 비교 가능한 column이어야 함)이 존재해야 한다. 만약 hash join이 가능한 조건이 존재하지 않을 경우, optimizer가 해당 hint를 무시하고 cost 계산을 통해 최적의 join operation을 선택한다.

다음은 T1, T2 table들의 join에 대하여 USE_HASH hint를 적용한 예이다.

- 유형 1: 한 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ USE_HASH(T1) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 대하여 각각 다른 join operation hint를 기술한 경우
(ORDERED hint에 의해 순서가 지정되어 left node (outer node)에 위치한 T1 table에 대한 join operation hint (USE_HASH)가 적용됨)

```
SELECT /*+ ORDERED USE_HASH(T1) USE_MERGE(T2) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

- 유형 3: USE_HASH hint를 적용할 수 없는 경우 (hash join 조건이 없는 경우)

```
SELECT /*+ USE_HASH(T1, T2) */ *
      FROM T1, T2
      WHERE T1.I1 < T2.I1;
```

NO_USE_HASH

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 hash join 기법을 제외한 나머지 기법 중 하나를 선택하여 join 하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

NO_USE_HASH hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_HASH hint를 제외한 다른 join operation hint에 기술할 수 있다. 이 hint에 기술된 table이 USE_HASH hint에 기술되고 "hint_error" property가 on인 경우 validation error로 처리되며, off인 경우 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

NO_USE_HASH hint에 기술된 table이 join에 참여하는 경우, optimizer가 hash join을 제외한 나머지 join operation 기법들에 대한 cost를 계산하여 그 중 최적의 join operation을 선택한다.

다음은 T1, T2 table들의 join에 대하여 NO_USE_HASH hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ NO_USE_HASH(T1) */ *
      FROM T1, T2
      WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우
(T2 table에 기술한 hint가 적용되어 merge join 처리됨)

```
SELECT /*+ NO_USE_HASH(T1) USE_MERGE(T2) */ *
      FROM T1, T2
      WHERE T1.I1 = T2.I1;
```

USE_MERGE

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 merge join 기법으로 join 하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

USE_MERGE hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_HASH, USE_NL, USE_INL 등의 hint에 기술되면 안된다. 기술하는 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

USE_MERGE hint에 기술된 table이 join에 참여하는 경우, join 조건에 merge join이 가능한 조건 (equi-join이어야 하고 비교 가능한 column이어야 함)이 존재해야 하며, 만약 merge join이 가능한 조건이 존재하지 않을 경우, optimizer가 해당 hint를 무시하고 cost를 계산하여 최적의 join operation을 선택한다.

다음은 T1, T2 table들의 join에 대하여 USE_MERGE hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ USE_MERGE(T1) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우
(ORDERED hint가 순서를 결정하여 left node (outer node)에 위치한 T1 table에 join operation hint (USE_MERGE)가 적용됨)

```
SELECT /*+ ORDERED USE_MERGE(T1) USE_HASH(T2) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 3: USE_MERGE hint를 적용할 수 없는 경우 (hash join 조건이 없음)

```
SELECT /*+ USE_MERGE(T1, T2) */ *
  FROM T1, T2
 WHERE T1.I1 < T2.I1;
```

NO_USE_MERGE

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 merge join 기법을 제외한 나머지 기법 중 하나를 선택하여 join 하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

NO_USE_MERGE hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_MERGE hint를 제외한 다른 join operation hint에 기술할 수 있다. 이 hint에 기술된 table이 USE_MERGE hint에 기술된 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

NO_USE_MERGE hint에 기술된 table이 join에 참여하는 경우, optimizer가 merge join을 제외한 나머지 join operation 기법들의 cost를 계산하여 최적의 join operation을 선택한다.

다음은 T1, T2 table들의 join에 대하여 NO_USE_MERGE hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ NO_USE_MERGE(T1) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우

(T2 table에 기술한 hint가 적용되어 hash join 됨)

```
SELECT /*+ NO_USE_MERGE(T1) USE_HASH(T2) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

USE_NL

Join을 수행할 때 이 hint에 기술한 table이 포함되면 optimizer로 하여금 nested loops join 기법으로 join하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

USE_NL hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table이 USE_HASH, USE_MERGE, USE_INL 등의 hint에 기술되면 안되며, 기술된 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

Nested loops join은 hash join이나 merge join과 다르게 어떠한 제약도 없이 join할 수 있는 기법이다.

다음은 T1, T2 table들의 join에 대하여 USE_NL hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ USE_NL(T1) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우
(ORDERED hint가 순서를 결정하여 left node (outer node)에 위치한 T1 table에 join operation hint (USE_NL)가 적용됨)

```
SELECT /*+ ORDERED USE_NL(T1) USE_HASH(T2) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

NO_USE_NL

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 nested loops join 기법을 제외한 나머지 기법 중 하나를 선택하여 join하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

NO_USE_NL hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_NL hint를 제외한 다른 join operation hint에도 기술할 수 있다. 이 hint에 기술된 table이 USE_NL hint에 기술된 경우, "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

NO_USE_NL hint에 기술된 table이 join에 참여하는 경우 optimizer가 nested loops join을 제외한 나머지 join o

operation 기법들의 cost를 계산하여 최적의 join operation을 선택한다. 단, 제약때문에 다른 join operation 기법들을 사용할 수 없는 경우, optimizer가 이 hint를 무시하고 nested loops join을 이용한 기법을 선택한다.

다음은 T1, T2 table들의 join에 대하여 NO_USE_NL hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ NO_USE_NL(T1) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우
(T2 table에 기술한 hint가 적용되어 hash join 됨)

```
SELECT /*+ NO_USE_NL(T1) USE_HASH(T2) */ *
FROM T1, T2
WHERE T1.I1 = T2.I1;
```

- 유형 3: NO_USE_NL hint를 기술하였지만 제약 조건 때문에 hash join과 같은 다른 join operation을 적용할 수 없는 경우 (nested loops join 됨)

```
SELECT /*+ NO_USE_NL(T1) */ *
FROM T1, T2
WHERE T1.I1 < T2.I1;
```

USE_INL

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 sort instant를 이용한 nested loops join 기법으로 join 하도록 한다. 이 hint는 <from clause>에 ','로 구분된 table들을 기술하거나 inner join으로 table들을 기술하는 경우에만 적용할 수 있다.

USE_INL hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_HASH, USE_MERGE, USE_NL 등의 hint에 기술되면 안되며, 기술하는 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

Sort instant를 이용한 nested loops join은 sort key가 right node (inner node)의 table에 대한 조인 조건을 만족하는 expression들인 sort instant를 생성하여 join하는 방법이다. 이 때 sort key는 key compare가 가능한 type이어야 한다. 조인 조건에 해당하는 expression들이 모두 key compare가 불가능한 type인 경우, 이 기법은 적용할 수 없다. 이 경우, 다른 join operation 기법들의 cost를 계산하여 최적의 기법을 적용한다.

다음은 T1, T2 table들의 join에 대하여 USE_INL hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ USE_INL(T1) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 각각 다른 join operation hint를 기술한 경우
(ORDERED hint가 순서를 결정하여 left node (outer node)에 위치한 T1 table에 join operation hint (USE_INL)가 적용됨)

```
SELECT /*+ ORDERED USE_INL(T1) USE_HASH(T2) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

NO_USE_INL

Join을 수행할 때 이 hint에 기술한 table이 포함될 경우, optimizer로 하여금 sort instant를 이용한 nested loops join 기법을 제외한 나머지 기법 중 하나를 선택하여 join 하도록 한다. 이 hint는 모든 join type에 적용할 수 있다.

NO_USE_INL hint에는 하나 이상의 table을 기술해야 하며, 동일한 table을 둘 이상 기술할 수 없다. 이 hint에 기술된 table은 USE_INL hint를 제외한 다른 join operation hint에 기술할 수 있다. 이 hint에 기술된 table이 USE_INL hint에 기술된 경우 "hint_error" property가 on이면 validation error로 처리되며, off이면 먼저 기술된 hint가 적용된다. 또한 join에 참여하는 두 table에 각각 다른 join operation hint가 기술되면 left node (outer node)에 기술된 hint가 우선 적용된다.

NO_USE_INL hint에 기술된 table이 join에 참여하는 경우, optimizer가 sort instant를 이용한 nested loops join을 제외한 나머지 join operation 기법들의 cost를 계산하여 최적의 join operation을 선택한다.

다음은 T1, T2 table들의 join에 대하여 NO_USE_INL hint를 적용한 예이다.

- 유형 1: 하나의 table에 대해서만 join operation hint를 기술한 경우

```
SELECT /*+ NO_USE_INL(T1) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

- 유형 2: 두 table에 대하여 각각 다른 join operation hint를 기술한 경우
(T2 table에 기술한 hint가 적용되어 hash join 됨)

```
SELECT /*+ NO_USE_INL(T1) USE_HASH(T2) */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1;
```

<query_transformation_hints>

UNNEST

Optimizer로 하여금 subquery에 대하여 동일한 결과를 보장하는 join 구문으로 변경하도록 한다. 이는 subquery를 별도로 수행하지 않고 상위 레벨 query와의 join을 이용하여 처리함으로써 subquery가 반복적으로 수행되는 것을 방지한다.

UNNEST hint는 subquery의 <hint clause>에만 기술할 수 있으며 해당 subquery에만 적용될 뿐 하위 subquery들에는 적용되지 않는다. 만약 다수의 subquery가 존재할 때 이를 모두 unnest하고 싶은 경우에는 해당 subquery 모두에 UNNEST hint를 기술하여야 하며, subquery 내부에 존재하는 subquery에 대하여 unnest하고 싶은 경우에는 해당 subquery에 UNNEST hint를 기술하여야 한다.

UNNEST hint가 NO_QUERY_TRANSFORMATION hint와 함께 기술된 경우, NO_QUERY_TRANSFORMATION hint 때문에 UNNEST hint가 무시된다.

UNNEST hint는 NO_UNNEST hint와 동시에 사용할 수 없다. 동시에 사용하는 경우 "hint_error" property를 on으로 설정하면 validation error로 처리되며, off로 설정하면 먼저 기술한 hint가 적용된다.

다음은 UNNEST hint를 이용하여 subquery의 unnesting을 적용한 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ UNNEST */ I1
              FROM T2 );
```

- 유형 2: UNNEST hint와 NO_QUERY_TRANSFORMATION hint를 기술한 경우
(NO_QUERY_TRANSFORMATION hint 때문에 subquery를 unnesting하지 않음)

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ UNNEST NO_QUERY_TRANSFORMATION */ I1
              FROM T2 );
```

NO_UNNEST

Optimizer로 하여금 subquery를 unnesting을 하지 않고 filter로 처리하도록 한다. 이는 필요할 때마다 매번 subquery를 수행한다.

NO_UNNEST hint는 subquery의 <hint clause>에만 기술할 수 있으며, 해당 subquery에만 적용될 뿐 하위 subquery들에는 적용되지 않는다. 만약 다수의 subquery가 존재할 때 이를 모두 unnest하고 싶지 않은 경우에는 해당 subquery 모두에 NO_UNNEST hint를 기술하여야 하며, subquery 내부에 존재하는 subquery에 대하여 unnest하고 싶지 않은 경우에는 해당 subquery에 NO_UNNEST hint를 기술하여야 한다.

NO_UNNEST hint가 NO_QUERY_TRANSFORMATION hint와 함께 기술된 경우 NO_QUERY_TRANSFORMATION hint 때문에 NO_UNNEST hint가 무시된다.

NO_UNNEST hint는 UNNEST hint와 동시에 사용할 수 없다. 동시에 사용하는 경우 "hint_error" property를 on으로 설정하면 validation error로 처리되며, off로 설정하면 먼저 기술한 hint가 적용된다.

다음은 NO_UNNEST hint를 이용하여 subquery의 no unnesting을 적용한 예이다.

- IN subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE I1 IN ( SELECT /*+ NO_UNNEST */ I1
               FROM T2 );
```

NL_SJ

Optimizer로 하여금 subquery를 nested loops semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 semi join 형태로 풀어낸다.

NL_SJ는 EXISTS, IN, ANY quantify 연산자에 사용할 수 있으며, NOT EXISTS, NOT IN, ALL quantify 연산자에는 사용할 수 없다.

NL_SJ hint는 해당 subquery를 semi join 형태로 풀어내기 때문에 anti-semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우 optimizer가 이 hint를 무시한다.

다음은 NL_SJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE I1 IN ( SELECT /*+ NL_SJ */ I1
               FROM T2 );
```

- 유형 2: EXISTS subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE EXISTS ( SELECT /*+ NL_SJ */ I1
                FROM T2
                WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE I1 < ANY ( SELECT /*+ NL_SJ */ I1
```



```
FROM T2 );
```

- 유형 4: NOT IN subquery에 사용한 경우
(anti-semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
FROM T1
WHERE I1 NOT IN ( SELECT /*+ NL_SJ */ I1
                  FROM T2 );
```

NL_ISJ

Optimizer로 하여금 subquery를 nested loops inverted semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 inverted semi join 형태로 풀어낸다.

Inverted semi join은 right node (inner node)에 semi join을 위한 key를 unique sort key로 갖는 sort instant를 생성하여 역으로 left node (outer node)에서 해당 key와 일치하는 레코드를 찾는 semi join 방법이다.

NL_ISJ hint는 right node (inner node)가 semi join을 위한 key와 많이 중복되고, left node (outer node)에서 해당 key로 index scan을 할 수 있는 경우에 유용하다. Optimizer는 semi join을 위한 key를 left node (outer node)에서 index scan 할 수 없는 경우에 hint를 무시한다. 또한, semi join을 위한 key를 key compare 할 수 없는 경우에도 이 hint를 무시한다.

NL_ISJ는 EXISTS, IN, = ANY quantify 연산자에 사용할 수 있고 NOT EXISTS, NOT IN, ALL quantify 연산자 또는 = ANY를 제외한 ANY quantify 연산자에는 사용할 수 없다.

NL_ISJ hint는 해당 subquery를 semi join 형태로 풀어내기 때문에 anti-semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우, optimizer가 이 hint를 무시한다.

다음은 NL_ISJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ NL_ISJ */ I1
               FROM T2 );
```

- 유형 2: EXISTS subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE EXISTS ( SELECT /*+ NL_ISJ */ I1
                FROM T2
                WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 = ANY ( SELECT /*+ NL_ISJ */ I1
                FROM T2 );
```

- 유형 4: NOT IN subquery에 사용한 경우
(anti-semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
FROM T1
WHERE I1 NOT IN ( SELECT /*+ NL_ISJ */ I1
                 FROM T2 );
```

MERGE_SJ

Optimizer로 하여금 subquery를 merge semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 semi join 형태로 풀어낸다.

MERGE_SJ는 EXISTS, IN, = ANY quantify 연산자에 사용할 수 있고 NOT EXISTS, NOT IN, ALL quantify 연산자, = ANY를 제외한 ANY quantify 연산자에는 사용할 수 없다.

MERGE_SJ hint는 해당 subquery를 semi join 형태로 풀어내기 때문에 anti-semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우, optimizer가 이 hint를 무시한다. 또한, merge semi join에 사용할 key를 key compare 할 수 없는 경우에도 optimizer가 이 hint를 무시한다.

다음은 MERGE_SJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ MERGE_SJ */ I1
             FROM T2 );
```

- 유형 2: EXISTS subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE EXISTS ( SELECT /*+ MERGE_SJ */ I1
              FROM T2
              WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE I1 = ANY ( SELECT /*+ MERGE_SJ */ I1
                  FROM T2 );
```

- 유형 4: NOT IN subquery에 사용한 경우
(anti-semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
  FROM T1
 WHERE I1 NOT IN ( SELECT /*+ MERGE_SJ */ I1
                  FROM T2 );
```

HASH_SJ

Optimizer로 하여금 subquery를 hash semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 semi join 형태로 풀어낸다.

HASH_SJ는 EXISTS, IN, = ANY quantify 연산자에 사용할 수 있고 NOT EXISTS, NOT IN, ALL quantify 연산자, = ANY를 제외한 ANY quantify 연산자에는 사용할 수 없다.

HASH_SJ hint는 해당 subquery를 semi join 형태로 풀어내기 때문에 anti-semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우 optimizer가 이 hint를 무시한다. 또한, hash semi join에 사용할 key를 key compare 할 수 없는 경우에도 optimizer가 이 hint를 무시한다.

다음은 HASH_SJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE I1 IN ( SELECT /*+ HASH_SJ */ I1
               FROM T2 );
```

- 유형 2: EXISTS subquery에 사용한 경우

```
SELECT *
  FROM T1
 WHERE EXISTS ( SELECT /*+ HASH_SJ */ I1
                FROM T2
                WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 = ANY ( SELECT /*+ HASH_SJ */ I1
                  FROM T2 );
```

- 유형 4: NOT IN subquery에 사용한 경우
(anti-semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
FROM T1
WHERE I1 NOT IN ( SELECT /*+ HASH_SJ */ I1
                  FROM T2 );
```

HASH_ISJ

Optimizer로 하여금 subquery를 hash inverted semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 inverted semi join 형태로 풀어낸다.

HASH_ISJ hint는 left node (outer node)의 row 개수가 적고, right node (inner node)의 row 개수가 많은 경우에 유용하다. Optimizer는 semi join을 위한 key를 key compare 할 수 없는 경우에 이 hint를 무시한다.

HASH_ISJ는 EXISTS, IN, = ANY quantify 연산자에 사용할 수 있고 NOT EXISTS, NOT IN, ALL quantify 연산자 및 = ANY를 제외한 ANY quantify 연산자에는 사용할 수 없다.

HASH_ISJ hint는 해당 subquery를 semi join 형태로 풀어내기 때문에 anti-semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우 optimizer가 이 hint를 무시한다. 또한, hash semi join에 사용할 key를 key compare 할 수 없는 경우에도 optimizer가 이 hint를 무시한다.

다음은 HASH_ISJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: IN subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ HASH_ISJ */ I1
               FROM T2 );
```

- 유형 2: EXISTS subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE EXISTS ( SELECT /*+ HASH_ISJ */ I1
                FROM T2
```

```
WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 = ANY ( SELECT /*+ HASH_ISJ */ I1
                 FROM T2 );
```

- 유형 4: NOT IN subquery에 사용한 경우
(anti-semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
FROM T1
WHERE I1 NOT IN ( SELECT /*+ HASH_ISJ */ I1
                 FROM T2 );
```

HASH_AJ

Optimizer로 하여금 subquery를 hash anti-semi join으로 처리하도록 한다. 이는 subquery를 동일한 결과를 갖는 anti-semi join 형태로 풀어낸다.

HASH_AJ는 NOT EXISTS, NOT IN, != ALL quantify 연산자에 사용할 수 있고 EXISTS, IN, ANY quantify 연산자, != ALL quantify 연산자를 제외한 ALL quantify 연산자에는 사용할 수 없다.

HASH_AJ hint는 해당 subquery를 anti-semi join 형태로 풀어내기 때문에 semi join 형태로만 풀릴 수 있는 subquery에 기술한 경우 optimizer가 이 hint를 무시한다. 또한, hash semi join에 사용할 key를 key compare 할 수 없는 경우에도 optimizer가 이 hint를 무시한다.

다음은 HASH_AJ hint를 이용하여 subquery를 join 형태로 풀어낸 예이다.

- 유형 1: NOT IN subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 NOT IN ( SELECT /*+ HASH_AJ */ I1
                 FROM T2 );
```

- 유형 2: NOT EXISTS subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE NOT EXISTS ( SELECT /*+ HASH_AJ */ I1
                  FROM T2
```

```
WHERE T1.I1 = T2.I1 );
```

- 유형 3: Quantify 연산자의 subquery에 사용한 경우

```
SELECT *
FROM T1
WHERE I1 != ALL ( SELECT /*+ HASH_AJ */ I1
                  FROM T2 );
```

- 유형 4: IN subquery에 사용한 경우
(semi join으로 풀 수 있는 연산자이므로 hint가 무시됨)

```
SELECT *
FROM T1
WHERE I1 IN ( SELECT /*+ HASH_AJ */ I1
              FROM T2 );
```

TRANSITIVE_CLOSURE

A=B 이고 B = C 이면, A=C 이다. 이와 같은 관계를 join predicate에 적용한다.

- 'T1.i1 = T2.i1'과 'T2.i1 = T3.i1'이 'T1.i1 = T3.i1' predicate을 생성한다.
하지만 join ordering에 사용되지 않을 경우, 자동으로 삭제 된다.

```
SELECT *
FROM T1, T2, T3
WHERE T1.i1 = T2.i1 AND T2.i1 = T3.i1;
```

위 예제에서 join ordering이 ((t1 ⋈ t2) ⋈ t3)일 경우, 'T2.i1 = T3.i1'과 'T1.i1 = T3.i1' 중 선택도가 높은 predicate 하나만 선택된다. Join ordering이 ((t1 ⋈ t3) ⋈ t2)일 경우, 'T1.i1 = T2.i1'과 'T2.i1 = T3.i1' 중 선택도가 높은 predicate 하나만 선택된다.

NO_TRANSITIVE_CLOSURE

Join transitive closure를 join predicate에 적용하지 않는다.

- 'T1.i1 = T2.i1'과 'T2.i1 = T3.i1'이 'T1.i1 = T3.i1' predicate을 생성하지 않는다.

```
SELECT *
FROM T1, T2, T3
WHERE T1.i1 = T2.i1 AND T2.i1 = T3.i1;
```

Join transitive closure가 적용되지 않으므로 'T1.i1 = T3.i1'인 predicate이 없다. 따라서 ((t1 ⋈ t3) ⋈ t2) 과 같은 join ordering이 발생하지 않는다.

MERGE

기술한 view가 simple view merging이 가능하면 이를 적용하도록 optimizer에 지시한다. 이 hint는 기술한 view를 포함하는 상위 query block에 명시해야 한다.

MERGE hint는 <join order hints>, <join operation hints>, <other hints>, TRANSITIVE_CLOSURE/NO_TRANSITIVE hint, NO MERGE hint와 함께 쓰이면 적용 방법이 모호해진다.

따라서 이들 hint들과 MERGE hint가 함께 쓰였을 경우, "hint_error" property가 on 이면 validation error로 처리되며, off 이면 먼저 기술한 hint가 적용된다.

다음은 MERGE hint를 이용하여 simple view merging을 적용한 예이다.

- 유형 1: 하나의 view에 MERGE hint를 기술한 경우, v1이 simple view merging 된다.

```
SELECT /*+ MERGE(v1) */ *
FROM ( SELECT * FROM t1 ) v1;
```

- 유형 2: 여러 개의 view에 MERGE hint를 기술한 경우, v1, v2만 simple view merging 되고, v3는 simple view merging 되지 않는다.

```
SELECT /*+ MERGE(v1) MERGE(v2) NO_MERGE(v3) */ *
FROM ( SELECT i1 FROM t1 ) v1,
      ( SELECT i1 FROM t2 ) v2,
      ( SELECT i1 FROM t3 ) v3
WHERE v1.i1 = v2.i1 AND v2.i1 = v3.i1;
```

- 유형 3: <join order hints>와 함께 기술할 경우, simple view merging하면 t3와 t4의 적용 순서와 방향이 모호해진다. 따라서 먼저 명시된 hint 만 적용되고 simple view merging은 수행되지 않는다.

```
SELECT /*+ ORDERING( t1, t2, v1 RIGHT ) MERGE(v1) */ *
FROM t1,
      t2,
      ( SELECT * FROM t3, t4 WHERE t3.i1 = t4.i1 ) v1
WHERE t1.i1 = t2.i1 AND t2.i1 = v1.i1;
```

→

- 유형 4: <join operation hints>와 함께 기술한 경우, simple view merging 하면 hashing 대상 view가 사라진다. 따라서 먼저 명시된 hint만이 적용되고 simple view merging은 수행되지 않는다.

```
SELECT /*+ USE_HASH(v1) MERGE(v1) */ *
FROM t1,
      t2,
      ( SELECT * FROM t3, t4 WHERE t3.i1 = t4.i1 ) v1
WHERE t1.i1 = t2.i1 AND t2.i1 = v1.i1;
```

- 유형 5: <other hints>와 함께 기술할 경우, simple view merging 할 때 view와 상위 query 사이의 other hints 처리가 서로 충돌한다. 따라서 먼저 명시된 hint만 적용되고 simple view merging은 수행되지 않는다.

```
SELECT /*+ PUSH_PRED MERGE(v1) */ *
FROM ( SELECT /*+ NO_PUSH_PRED */ * FROM t1 WHERE t1.i1 = 1 ) v1
WHERE v1.i1 = 1;
```

- 유형 6: TRANSITIVE CLOSURE hint와 함께 기술할 경우, simple view merging 할 때 view와 상위 query 사이의 other hints 처리가 서로 충돌한다. 따라서 먼저 명시된 hint만이 적용되고 simple view merging은 수행되지 않는다.

```
SELECT /*+ TRANSITIVE_CLOSURE MERGE(v1) */ *
FROM ( SELECT /*+ NO_TRANSITIVE_CLOSURE */ *
      FROM t1, t2, t3
      WHERE t1.i1 = t2.i1 AND t2.i1 = t3.i1 ) v1;
```

NO_MERGE

기술한 view를 simple view merging 할 수 있더라도 optimizer로 하여금 이를 적용하지 않도록 한다. 이 hint는 기술한 view를 포함하는 상위 query block에 명시해야 한다.

NO_MERGE hint가 <join order hints>, <join operation hints>, <other hints>, TRANSITIVE_CLOSURE/ NO_TRANSITIVE hint, MERGE hint와 함께 쓰일 경우 적용 방법이 모호해진다.

따라서 이들 hint와 NO_MERGE hint가 함께 쓰였을 경우, "hint_error" property가 on 이면 validation error로 처리되고 off 이면 먼저 기술한 hint가 적용된다.

다음은 NO_MERGE hint를 이용하여 simple view merging을 적용하지 않도록 하는 예이다.

- 유형 1: 하나의 view에 NO_MERGE hint를 기술한 경우, v1이 simple view merging 되지 않는다.

```
SELECT /*+ NO_MERGE(v1) */ *
FROM ( SELECT * FROM t1 ) v1;
```

- 유형 2: 여러 개의 view에 NO_MERGE hint를 기술한 경우, v1, v2만 simple view merging 되고, v3는 simple view merging 되지 않는다.

```
SELECT /*+ MERGE(v1) MERGE(v2) NO_MERGE(v3) */ *
FROM ( SELECT i1 FROM t1 ) v1,
      ( SELECT i1 FROM t2 ) v2,
      ( SELECT i1 FROM t3 ) v3
WHERE v1.i1 = v2.i1 AND v2.i1 = v3.i1;
```


NO_QUERY_TRANSFORMATION

Optimizer가 query를 변경하지 않도록 한다. 이 hint를 기술하면 heuristic optimizer나 cost based optimizer 등에서 최적의 성능을 위해 optimizer가 query를 변형하는 과정들이 전혀 수행되지 않는다.

NO_QUERY_TRANSFORMATION hint는 최상위 레벨의 query나 subquery에 기술할 수 있으며, 기술한 query 뿐만 아니라 그 하위의 subquery 모두에 대하여 적용된다

다음은 NO_QUERY_TRANSFORMATION을 이용하여 query를 변형하지 못하도록 한 예이다.

- 유형 1: Subquery에 NO_QUERY_TRANSFORMATION hint를 사용한 경우
(Subquery의 query만 변경하지 않음)

```
SELECT *
  FROM T1
 WHERE I1 IN ( SELECT /*+ NO_QUERY_TRANSFORMATION */ I1
               FROM T2 );
```

- 유형 2: 최상위 query에 NO_QUERY_TRANSFORMATION hint를 사용한 경우
(최상위 query와 subquery의 모든 query를 변경하지 않음)

```
SELECT /*+ NO_QUERY_TRANSFORMATION */ *
  FROM T1
 WHERE I1 IN ( SELECT I1
               FROM T2 );
```

- 유형 3: 최상위 query에 NO_QUERY_TRANSFORMATION hint를 사용하고 subquery에 UNNEST hint를 사용한 경우
(Subquery의 UNNEST hint를 무시함)

```
SELECT /*+ NO_QUERY_TRANSFORMATION */ *
  FROM T1
 WHERE I1 IN ( SELECT /*+ UNNEST */ I1
               FROM T2 );
```

<other_hints>

PUSH_PRED

Optimizer로 하여금 single table에 적용 가능한 filter들을 push하도록 한다. 이 hint는 해당 select 구문의 적용 가능한 filter를 single table에 push하여 선행 처리하도록 한다. 해당 힌트는 select로 시작되는 SubQuery들에도 각각 기술할 수 있으며, 각 질의 block 내에서만 적용되고, 하위 SubQuery로 확산되지 않는다.

각 SubQuery들에는 독립적으로 이 힌트를 기술할 수 있으며, 기술하지 않으면 default 값이 적용된다. Default 값은 PUSH_PRED이다.

다음은 PUSH_PRED hint를 이용하여 subquery를 push하는 예이다.

- T1.i1 = 1을 T1 table에 push 하는 경우

```
SELECT /*+ PUSH_PRED */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1
       AND T1.I1 = 1;
```

NO_PUSH_PRED

Optimizer로 하여금 single table에 적용 가능한 filter들을 push하지 못하도록 한다. 이 hint는 해당 select 구문의 적용 가능한 filter를 single table에 push하여 선행 처리되는 것을 막는다. 해당 힌트는 select로 시작되는 SubQuery들에도 각각 기술할 수 있으며, 각 질의 block 내에서만 적용되고, 하위 SubQuery로 확산되지 않는다.

각 SubQuery들에는 독립적으로 이 힌트를 기술할 수 있으며, 기술하지 않으면 default 값이 적용된다. Default 값은 PUSH_PRED이다.

이 hint를 기술했을 때 from절에 join이 존재할 경우, 해당 필터를 처리할 수 있는 가장 하위 join에서 처리된다. 즉, 해당 filter를 처리할 수 있는 single table에서 선행 처리 되는 것을 방지한다. 또한, 이 hint를 기술했을 때 from절에 view가 존재할 경우, 해당 view의 하위로 filter가 push되지 않는다.

다음은 NO_PUSH_PRED hint를 이용하여 filter를 push하는 예이다.

- 유형 1: T1과 T2를 join한 후에 T1.i1 = 1을 수행한다.

```
SELECT /*+ NO_PUSH_PRED */ *
  FROM T1, T2
 WHERE T1.I1 = T2.I1
       AND T1.I1 = 1;
```

- 유형 2: View A를 처리한 후에 A.i1 = 1을 수행한다.

```
SELECT /*+ NO_PUSH_PRED */ *
  FROM (SELECT I1 FROM T1) AS A
 WHERE A.I1 = 1;
```

PUSH_SUBQ

Optimizer로 하여금 subquery의 적용 가능한 최하위 노드까지 push하도록 한다. 이 hint는 subquery 상위 레벨의 query가 join으로 구성되어 있을 때 해당 join node 구조의 최상위 node부터 최하위 노드까지 탐색하여 해당 subquery를 처리할 수 있는 node를 찾아서 push 가능한 가장 하위 node에서 subquery를 처리하도록 한다.

PUSH_SUBQ hint는 subquery에만 기술할 수 있으며, 다수의 subquery가 존재할 경우, PUSH_SUBQ hint를 subquery마다 각각 기술해야 한다.

PUSH_SUBQ hint를 기술하지 않으면 optimizer가 cost를 계산하여 가장 cost가 좋은 node에 subquery를 push한다.

다음은 PUSH_SUBQ hint를 이용하여 subquery를 push하는 예이다.

- 유형 1: Subquery를 T1 table에 push 하는 경우

```
SELECT *
  FROM T1, T2
 WHERE T1.I1 = T2.I1
        AND T1.I1 IN ( SELECT /*+ PUSH_SUBQ */ I1
                      FROM T3 );
```

- 유형 2: Subquery를 최상위 node에서만 처리할 수 있는 경우
(Subquery를 하위 노드로 push하지 못함)

```
SELECT *
  FROM T1, T2
 WHERE T1.I1 = T2.I1
        AND T1.I1 IN ( SELECT /*+ PUSH_SUBQ */ I1
                      FROM T3
                      WHERE T3.I2 = T1.I2
                        AND T3.I3 = T2.I3 );
```

NO_PUSH_SUBQ

Optimizer로 하여금 subquery를 하위 노드로 push하지 못하도록 한다. 이 hint는 subquery의 상위 레벨 query가 join으로 구성되어 있을 때 해당 join node 구조의 최상위 node에서 subquery를 처리하도록 한다.

NO_PUSH_SUBQ hint는 subquery에만 기술할 수 있으며, 다수의 subquery가 존재할 경우, NO_PUSH_SUBQ hint를 subquery마다 각각 기술해야 한다.

NO_PUSH_SUBQ hint를 기술하지 않으면 optimizer가 cost를 계산하여 가장 cost가 좋은 node에 subquery를 push한다.

다음은 NO_PUSH_SUBQ hint를 이용하여 subquery를 push하는 예이다.

- T1 table에서 처리 가능한 subquery에 NO_PUSH_SUBQ hint를 사용한 경우
(T1 table과 T2 table을 join한 후에 subquery를 수행함)

```
SELECT *
  FROM T1, T2
 WHERE T1.I1 = T2.I1
        AND T1.I1 IN ( SELECT /*+ NO_PUSH_SUBQ */ I1
                      FROM T3 );
```

USE_GROUP_HASH

Group by 절이 있는 경우에 grouping 하기 위해 optimizer로 하여금 hash instant를 사용하도록 한다.

USE_GROUP_HASH hint는 group by 절이 있는 구문에만 기술할 수 있으며, 다수의 subquery가 존재할 경우 sub query마다 각각 기술해야 한다.

USE_GROUP_HASH hint를 사용하지 않고 grouping을 위한 group key들이 하위 노드로부터 정렬된 상태일 때는 group node를 사용하는 방법을 고려해야 한다.

다음은 USE_GROUP_HASH hint를 이용하는 예이다.

- T1 table에 group key들로 구성된 index가 존재할 때 USE_GROUP_HASH를 사용한 경우

```
SELECT /*+ USE_GROUP_HASH */ *
      FROM T1
      GROUP BY I1;
```

USE_DISTINCT_HASH

Distinct 절이 있는 경우에 distinct를 수행하기 위해 optimizer로 하여금 hash instant를 사용하도록 한다.

USE_DISTINCT_HASH hint는 group by 절이 있는 구문에만 기술할 수 있으며, 다수의 subquery가 존재할 경우 sub query마다 각각 기술해야 한다.

USE_DISTINCT_HASH hint를 사용하지 않고 distinct를 위한 distinct key들이 하위 노드로부터 정렬된 상태일 때는 group node를 사용하는 방법을 고려해야 한다.

다음은 USE_DISTINCT_HASH hint를 이용하는 예이다.

- T1 table에 distinct key들로 구성된 index가 존재할 때 USE_DISTINCT_HASH를 사용한 경우

```
SELECT /*+ USE_DISTINCT_HASH */ I1, I2
      FROM T1;
```

설명

Hint는 사용자가 GOLDILOCKS optimizer에게 SQL 구문을 수행하는 방법을 직접 지시하기 위하여 사용하는 comment이다. GOLDILOCKS optimizer가 실행 계획을 정확히 판단할 수 없는 경우에 사용자가 hint를 사용하여 직접 실행 계획을 선택한다.

GOLDILOCKS optimizer가 실행 계획을 결정할 때는 사용자가 기술한 hint를 최우선으로 선택하게 되며, 만약 사용자가 기술한 hint를 사용할 수 없을 경우 GOLDILOCKS optimizer가 판단해서 결정한다.

사용자가 hint를 사용하면 GOLDILOCKS optimizer가 가능한 사용자의 hint대로 동작하기 때문에 사용자가 GOLDILOCKS optimizer의 SQL구문 실행 계획이 잘못되었다고 판단하는 경우에만 사용할 것을 권장한다.

특히, 동일한 SQL 구문을 반복해서 사용할 때 hint를 사용하면 GOLDILOCKS optimizer는 사용자가 기술한 hint대

로 수행한다. 이 경우, 해당 SQL 구문에 포함된 TABLE이나 VIEW의 데이터가 변경되어 다른 실행 계획의 성능 저하를 가져올 수 있으므로 주의한다.

GOLDILOCKS에서 hint는 SELECT, INSERT SELECT, DELETE, UPDATE 등의 구문에 사용할 수 있다. Hint는 각 구문의 키워드 다음 위치에 /*+ 키워드와 */ 키워드를 양 옆에 사용하여 그 사이에 기술한다.

사용자가 잘못된 hint를 기술하거나 동일한 실행 계획에 영향을 미치는 hint를 둘 이상 기술하여 hint가 서로 충돌하는 경우에는 해당 hint를 무시하거나 먼저 기술된 hint를 우선 적용한다. Hint가 잘못 기술되었는지 알고 싶다면 ALTER 구문을 이용하여 HINT_ERROR property를 on으로 설정하고 이용하면 된다.

사용 예

다음은 INSERT SELECT, DELETE, UPDATE, SELECT 구문에서 <hint clause>를 사용한 예이다.

- INSERT SELECT 구문에서 사용한 경우

```
gSQL> INSERT INTO T1 SELECT /*+ INDEX(T1, T1_IDX) */ * FROM T1;
1 row created.
```

- SELECT 구문에서 사용한 경우

```
gSQL> SELECT /*+ INDEX(T1, T1_IDX) */ * FROM T1;
I1
--
1
1
2 rows selected.
```

- UPDATE 구문에서 사용한 경우

```
gSQL> UPDATE /*+ INDEX(T1, T1_IDX) */ T1 SET I1 = 2;
2 rows updated.
```

- DELETE 구문에서 사용한 경우

```
gSQL> DELETE /*+ INDEX(T1, T1_IDX) */ T1 WHERE I1 = 2;
2 rows deleted.
```

다음은 ALTER 구문을 이용하여 HINT_ERROR property를 on으로 변경한 후에 hint를 잘못 사용한 예이다.

```
gSQL> ALTER SESSION SET HINT_ERROR = ON;
Session altered.
gSQL> SELECT /*+ INDEX(T2, T1_IDX) */ * FROM T1;
```

```
ERR-42000(16058): not applicable hint :  
SELECT /*+ INDEX(T2, T1_IDX) */ * FROM T1  
      *  
ERROR at line 1:
```

호환성

SQL 표준에서는 hint를 정의하지 않고 있으며, Oracle 등의 타 벤더에서는 각각 독립적인 형태로 hint를 지원하고 있다.

GOLDILOCKS에서 제공하는 hint는 대부분 Oracle에서 제공하는 hint와 호환되며, GOLDILOCKS에서 제공하는 hint와 동일한 Oracle hint 구문은 Oracle에서도 동일하게 동작한다. 단, INDEX_COMBINE hint는 Oracle과 다른 의미이며 ORDERING hint는 Oracle에서 제공하지 않는다.

참조

관련 내용은 [query specification](#)을 참조한다.

16.126 SELECT .. FOR UPDATE

기능

SELECT 구문의 결과 집합을 갱신할지 여부를 설정한다.

구문

```

<select for update statement> ::=
    <query expression> <updatability clause>
    ;
<updatability clause> ::=
    FOR READ ONLY
    | FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
    | WAIT
    | WAIT second
    | NOWAIT
  
```

사용 범위 및 접근 권한

<select for update statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- <query expression> 구문을 수행하려면 구문에 사용된 모든 테이블에 대한 다음 권한 중 하나가 사용자에게 있어야 한다.
 - 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE
- FOR UPDATE 구문을 사용할 경우, lock 대상이 되는 테이블에 대한 다음 권한 중 하나가 있어야 한다.
 - 해당 테이블에 대해 (LOCK 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (LOCK TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - LOCK ANY TABLE ON DATABASE

구문 규칙 및 파라미터

<query expression>

SELECT 구문에 INTO 절이 없어야 한다.

FOR UPDATE를 사용하려면 query가 base table의 row 변화를 식별하거나 row에 lock을 획득할 수 있는 updatable query여야 한다.

Updatable query는 다음 조건을 모두 만족해야 한다.

- 최상위 query에 DISTINCT가 존재하지 않아야 한다.
 - (X) SELECT DISTINCT * FROM t1;
- 최상위 query에 GROUP BY, HAVING, aggregation function이 존재하지 않아야 한다.
 - (X) SELECT MAX(c1) FROM t1;
- Set 연산자가 존재하지 않아야 한다.
 - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- FROM 절에 나열된 table 들에 하나 이상의 updatable column이 존재해야 한다.
 - Join에 포함되는 테이블 중 cross join에 해당되지 않는 테이블의 column은 updatable column이 아니다.
 - FULL OUTER JOIN은 cross join이 아니다.
 - NATURAL JOIN 은 cross join이 아니다.
 - INNER JOIN에 USING 구문이 사용된 경우 cross join이 아니다.
 - 다음과 같은 table들의 column은 updatable column이 아니다.
 - Dictionary table, fixed table, performance view
 - View의 column은 updatable table이 아니다.

SELECT 구문에 대한 자세한 내용은 **query expression**을 참조한다.

<updatability clause>

결과 집합에 대한 row를 변경할지 여부를 지정한다.

- FOR READ ONLY
 - 읽기 전용 질의임을 선언한다.
- FOR UPDATE
 - 쓰기 가능한 질의임을 선언한다.
 - 질의를 수행할 때 해당 트랜잭션이 종료될 때까지 다른 트랜잭션에 의해 변경되지 않도록 해당 row 들에 대한 x lock을 획득한다.
 - <query expression>이 updatable query 여야 한다.

FOR UPDATE OF ...

질의를 수행할 때 lock 획득과 관련된 column들을 나열한다.

- FOR UPDATE OF 구문에 나열하는 column
 - <query expression>의 FROM 절에 나열된 table의 갱신 가능한 column이어야 한다.
 - 나열된 column의 table에 대해 lock을 획득한다.
- FOR UPDATE만 사용하는 경우
 - <query expression>의 FROM 절에 나열된 table들의 모든 갱신 가능한 column을 나열한 것과 동일한 의미이다.
 - 모든 column의 table에 대해 lock을 획득한다.

<lock wait mode>

FOR UPDATE 구문과 함께 사용하며, lock 획득 방법을 지정한다.

- WAIT
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock을 획득하며
 - Lock을 획득할 때까지 대기한다.
- WAIT second
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock을 획득하며
 - 지정한 시간동안 lock을 획득하지 못하면 에러가 발생한다.
 - 초 단위이며 0 ~ 10000000000 까지의 값을 사용할 수 있다.
- NOWAIT
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock을 획득하며
 - 즉시 lock을 획득하지 못하면 에러가 발생한다.
- 명시하지 않을 경우, 기본값은 WAIT이다.

설명

SELECT 구문은 transaction의 종료 여부와 관계없이 row에 대한 fetch를 지속할 수 있는 반면에, SELECT .. FOR UPDATE 구문은 row들에 대한 lock을 획득하기 때문에 transaction이 종료되면 fetch 할 수 없다.

노트

Cursor holdability

- WITH HOLD
 - Transaction 종료 여부와 관계없이 fetch를 지속할 수 있다.
 - Fetch across commit 이라고도 한다.

- WITHOUT HOLD
 - Transaction이 종료되면 fetch 할 수 없다.

사용 예

다음은 FOR UPDATE 구문을 사용하여 row에 대한 lock을 획득하는 예이다.

```
gSQL> SELECT id, data FROM t1 WHERE id = 3 FOR UPDATE;
ID DATA
-- -----
3 data_3
1 row selected.
```

다음과 같이 join과 ORDER BY 구문을 사용하더라도 updatable query이면 FOR UPDATE 구문을 사용할 수 있다.

```
gSQL> SELECT t1.id, t1.name, t2.addr
        FROM t1, t2
        WHERE t1.id = t2.id
        ORDER BY 1
        FOR UPDATE;
ID NAME      ADDR
-- -
1 someone somewhere
2 anyone anywhere
3 unknown N/A
4 leekmo leekmo's home
5 mkkim seoul
5 rows selected.
```

다음과 같이 updatable query가 아닌 경우에는 FOR UPDATE 구문을 사용할 수 없다.

```
gSQL> SELECT id, COUNT(*)
        FROM t1
        GROUP BY id
        FOR UPDATE;
ERR-42000(16112): query expression is not updatable
```

호환성

SQL 표준에서는 <select for update statement>를 정의하지 않고 있는데, 이는 **DECLARE cursor_name** 구문을 사용하여 정의할 수 있다.

16.127 SELECT .. INTO

기능

질의를 통해 row 하나를 검색하고, 검색한 row의 값을 호스트 변수로 얻어온다.

구문

```
<select statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
        <table expression>
    ;
<select target list> ::=
    variable_name [, ...]
```

사용 범위 및 접근 권한

<select statement: single row> 구문을 수행하려면 사용자가 구문에 사용된 모든 테이블에 대해 다음 권한 중 하나를 가져야 한다.

- 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
- 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

<hint clause>

질의를 수행하기 위한 힌트를 기술한다.

자세한 내용은 SELECT 구문의 hint clause를 참조한다.

〈set quantifier〉

질의 결과에서 중복을 제거할지 여부를 기술한다.
자세한 내용은 **query specification** 절을 참조한다.

〈select list〉

질의 결과로부터 검색할 column을 기술한다.
자세한 내용은 **select list** 절을 참조한다.

INTO 〈select target list〉

INTO 절에 기술된 변수의 개수는 〈select list〉에 기술된 expression의 개수와 동일해야 한다.

〈table expression〉

검색 조건 등 질의 내용을 기술한다.
자세한 내용은 **query specification** 절을 참조한다.

설명

검색할 row가 한 건 이하여야 한다.
두 건 이상의 row가 검색될 경우, 에러가 발생한다.

SELECT 구문들의 차이점

- 〈select statement〉
 - 조건에 부합하는 다수의 row를 검색하고, 검색한 row를 SQLFetch() 등의 API로 검색할 수 있다.
 - 예: SELECT c1 FROM t1 WHERE c1 > 0;
- 〈select statement: single row〉
 - 조건에 부합하는 한 건 이하의 row를 검색할 수 있으며, 검색한 row가 한 건일 경우 INTO 절의 호스트 변수에 값을 얻어온다.
 - 예: SELECT c2 INTO :v1 FROM t1 WHERE c1 = 0;

사용 예

다음은 interactive SQL (gsq)을 사용하여 host 변수에 값을 얻어오는 예이다.

```
gsq> \var v_id    INTEGER
gsq> \var v_data VARCHAR(128)
gsq> SELECT id, data INTO :v_id, :v_data FROM t1 WHERE id = 3;
V_ID V_DATA
---- -
      3 data_3
1 row selected.
```

16.128 SELECT .. INTO .. FOR UPDATE

기능

질의를 통해 row 하나를 검색하여 갱신을 수행할지 여부를 설정한 후, 검색한 row의 값을 호스트 변수에 얻어온다.

구문

```
<select for update statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
        <table expression> <updatability clause>
    ;
<select target list> ::=
    variable_name [ , ... ]
<updatability clause> ::=
    FOR READ ONLY
    | FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
    | WAIT
    | WAIT second
    | NOWAIT
```

사용 범위 및 접근 권한

<select statement: single row> 구문을 수행하려면 사용자에게 구문에 사용된 모든 테이블에 대한 다음 권한 중 하나가 있어야 한다.

- 테이블의 column 중 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
- 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
- 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- SELECT ANY TABLE ON DATABASE

FOR UPDATE 구문을 사용할 경우, lock 대상이 되는 테이블에 대해 다음 권한 중 하나가 있어야 한다.

- 해당 테이블에 대해 (LOCK 또는 CONTROL TABLE) ON TABLE

- 테이블이 속한 스키마에 대해 (LOCK TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- LOCK ANY TABLE ON DATABASE

구문 규칙 및 파라미터

〈select for update statement: single row〉

FOR UPDATE를 사용하려면 query가 base table의 row 변화를 식별하거나 row에 lock을 획득할 수 있는 updatable query여야 한다.

Updatable query는 다음 조건을 모두 만족해야 한다.

- 최상위 query에 DISTINCT가 존재하지 않아야 한다.
 - (X) SELECT DISTINCT * FROM t1;
- 최상위 query에 GROUP BY, HAVING, aggregation function이 존재하지 않아야 한다.
 - (X) SELECT MAX(c1) FROM t1;
- Set 연산자가 존재하지 않아야 한다.
 - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- FROM 절에 나열된 table들에 하나 이상의 updatable column이 존재해야 한다.
 - Join에 포함되는 테이블 중 cross join에 해당하지 않는 테이블의 column은 updatable column이 아니다.
 - FULL OUTER JOIN은 cross join이 아니다.
 - NATURAL JOIN은 cross join이 아니다.
 - INNER JOIN에 USING 구문이 사용된 경우 cross join이 아니다.
 - 다음과 같은 table들의 column은 updatable column이 아니다.
 - Dictionary table, fixed table, performance view
 - View의 column은 updatable table이 아니다.

〈updatability clause〉

결과 집합에 대해 row를 변경할지 여부를 지정한다.

- FOR READ ONLY
 - 읽기 전용 질의임을 선언한다.
- FOR UPDATE
 - 쓰기 가능한 질의임을 선언한다.
 - 질의를 수행할 때 해당 트랜잭션이 종료될 때까지, 다른 트랜잭션에 의해 변경되지 않도록 해당 row들에 대한 x lock을 획득한다.
 - 〈query expression〉이 updatable query이어야 한다.

FOR UPDATE OF ...

질의를 수행할 때 lock 획득과 관련된 column을 나열한다.

- FOR UPDATE OF 구문에 나열하는 column
 - <query expression>의 FROM 절에 나열된 table의 갱신 가능한 column이어야 한다.
 - 나열된 column의 table에 대해 lock을 획득한다.
- FOR UPDATE 만 사용하는 경우
 - <query expression>의 FROM 절에 나열된 table들의 모든 갱신 가능한 column을 나열한 것과 동일한 의미이다.
 - 모든 column의 table에 대해 lock을 획득한다.

<lock wait mode>

FOR UPDATE 구문과 함께 사용하며, lock 획득 방법을 지정한다.

- WAIT
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock을 획득하며
 - Lock을 획득할 수 있을 때까지 대기한다.
- WAIT second
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock 을 획득하며
 - 지정한 시간동안 lock을 획득할 수 없을 경우, 에러가 발생한다.
 - 초 단위이며 0 ~ 10000000000 까지의 값을 사용할 수 있다.
- NOWAIT
 - 질의 결과를 얻기 전에 질의 결과의 모든 row들에 대한 lock을 획득하며
 - 즉시 lock을 획득할 수 없을 경우, 에러가 발생한다.
- 명시하지 않을 경우, 기본값은 WAIT 이다.

<hint clause>

질의를 수행하기 위한 힌트를 기술한다.

자세한 내용은 SELECT 구문의 hint clause 절을 참조한다.

<set quantifier>

질의 결과에서 중복을 제거할지 여부를 기술한다.

자세한 내용은 query specification 절을 참조한다.

〈select list〉

질의 결과로부터 검색할 column을 기술한다.

자세한 내용은 **select list** 절을 참조한다.

INTO 〈select target list〉

INTO 절에 기술된 변수의 개수는 〈select list〉에 기술된 expression의 개수와 동일해야 한다.

〈table expression〉

검색 조건 등의 질의 내용을 기술한다.

자세한 내용은 **query specification** 절을 참조한다.

설명

검색할 row가 한 건 이하여야 한다.

두 건 이상의 row가 검색될 경우, 에러가 발생한다.

SELECT 구문은 transaction의 종료 여부와 관계없이 row에 대한 fetch를 지속할 수 있는 반면에, SELECT .. FOR UPDATE 구문은 row들에 대한 lock을 획득하기 때문에 transaction이 종료되면 fetch 할 수 없다.

노트

Cursor holdability

- WITH HOLD
 - Transaction 종료 여부와 관계없이 fetch를 지속할 수 있다.
 - Fetch across commit 라고도 한다.
- WITHOUT HOLD
 - Transaction이 종료되면 fetch 할 수 없다.

SELECT 구문들의 차이점

- <select for update statement>
 - 조건에 부합하는 다수의 row를 검색하여 갱신 수행 여부를 설정하고, 검색한 row를 SQLFetch() 등의 API로 검색할 수 있다.
 - 예: SELECT c1 FROM t1 WHERE c1 > 0 FOR UPDATE;
- <select for update statement: single row>
 - 조건에 부합하는 한 건 이하의 row를 검색하여 갱신 수행 여부를 설정하고, 검색한 row가 한 건일 경우 INTO 절의 호스트 변수에 값을 얻어온다.
 - 예: SELECT c2 INTO :v1 FROM t1 WHERE c1 = 0 FOR UPDATE;

사용 예

다음은 FOR UPDATE 구문을 사용하여 row에 lock을 획득하고, interactive SQL (gsq)을 사용하여 host 변수에 값을 얻어오는 예이다.

```
gsQL> \var v_id    INTEGER
gsQL> \var v_data VARCHAR(128)
gsQL> SELECT id, data INTO :v_id, :v_data FROM t1 WHERE id = 3 FOR UPDATE;
V_ID V_DATA
----
      3 data_3
1 row selected.
```

다음과 같이 join과 ORDER BY 구문을 사용하더라도 updatable query이면 FOR UPDATE 구문을 사용할 수 있다.

```
gsQL> \var v_id    INTEGER
gsQL> \var v_name VARCHAR(128)
gsQL> \var v_addr VARCHAR(128)
gsQL> SELECT t1.id, t1.name, t2.addr
        INTO :v_id, :v_name, :v_addr
        FROM t1, t2
        WHERE t1.id = t2.id
        ORDER BY 1
        LIMIT 1
        FOR UPDATE;
ID NAME    ADDR
--
1 someone somewhere
1 row selected.
```

다음과 같이 updatable query가 아닌 경우에는 FOR UPDATE 구문을 사용할 수 없다.

```
gSQL> \var v_id    INTEGER
gSQL> \var v_count INTEGER
gSQL> SELECT id, COUNT(*)
        INTO :v_id, :v_count
        FROM t1
        GROUP BY id
        FOR UPDATE;
ERR-42000(16112): query expression is not updatable
```

참조

관련 내용은 다음을 참조한다.

- SELECT .. FOR UPDATE
- SELECT .. INTO

16.129 SET CONSTRAINTS

기능

트랜잭션 내에서 지연 가능한 제약 조건들의 검사 시점을 IMMEDIATE 또는 DEFERRED로 설정한다.

구문

```
<set constraints mode statement> ::=
    SET { CONSTRAINT | CONSTRAINTS } <constraint name list> { DEFERRED | IMMEDIATE }
    ;
<constraint name list> ::=
    ALL
    | <constraint name> [, ...]
```

사용 범위 및 접근 권한

SET CONSTRAINTS를 수행하기 위해 별도의 접근 권한이 필요한 것은 아니다.

주의

Cluster system에서 지원하지 않는다.

구문 규칙 및 파라미터

CONSTRAINT | CONSTRAINTS

CONSTRAINT와 CONSTRAINTS는 동일한 의미의 키워드인데 SQL 표준은 CONSTRAINTS 이다.

<constraint name list>

제약 조건 이름 목록을 기술하거나, ALL 키워드를 사용하여 지연 가능한 제약 조건을 모두 명시할 수 있다.

<constraint name>을 기술할 경우, 제약 조건은 지연 가능해야 한다.

ALL은 지연 가능한 모든 제약 조건을 의미한다.

DEFERRED | IMMEDIATE

명시된 지연 가능한 제약 조건들의 검사 시점을 설정한다.

- IMMEDIATE
 - DML을 수행할 때 해당 제약 조건들을 검사한다.
 - 트랜잭션이 해당 제약 조건들을 위반할 경우, 에러가 발생한다
- DEFERRED
 - COMMIT을 수행할 때 해당 제약 조건들을 검사한다.

트랜잭션이 진행 중이면, 검사 시점은 현재 트랜잭션에 설정되며, 트랜잭션이 진행 중이 아닌 경우에는 다음 트랜잭션에 설정된다.

트랜잭션이 종료되면 다음 트랜잭션에 영향을 미치지 않는다.

설명

지연 가능한 제약 조건

지연 가능한 (DEFERRABLE) 제약 조건은 검사 시점을 변경할 수 있다.

다음은 지연 가능한 제약 조건을 가진 테이블을 생성하고, 데이터를 추가하는 예이다.

```
gSQL> CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(128) CONSTRAINT t1_uk UNIQUE
                                DEFERRABLE INITIALLY IMMEDIATE
);
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
```

```
gSQL> COMMIT;
Commit complete.
```

위의 예에서 name column에 지연 가능한 UNIQUE 제약 조건을 생성하였으며, 초기 검사 시점이 INITIALLY IMMEDIATE로 설정되어 DML을 수행할 때마다 제약 조건을 검사한다.

이 때, 다음과 같이 두 row의 name 값을 서로 교체하려고 하면 검사 시점이 IMMEDIATE라서 모두 제약 조건을 위반하게 된다.

```
gSQL> UPDATE t1 SET name = 'mkkim' WHERE id = 1;
ERR-23000(16057): unique constraint (PUBLIC.T1_UK) violated
gSQL> UPDATE t1 SET name = 'leekmo' WHERE id = 2;
ERR-23000(16057): unique constraint (PUBLIC.T1_UK) violated
```

다음과 같이 검사 시점을 DEFERRED로 변경하면 COMMIT 시점에 제약 조건을 검사하므로 위의 예와 동일한 변경 작업이 모두 성공한다.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> UPDATE t1 SET name = 'mkkim' WHERE id = 1;
1 row updated.
gSQL> UPDATE t1 SET name = 'leekmo' WHERE id = 2;
1 row updated.
gSQL> COMMIT;
Commit complete.
```

검사 시점을 DEFERRED로 설정하면 COMMIT 시점에 제약 조건을 검사하므로 제약 조건을 위반한 상태에서 트랜잭션을 COMMIT 할 경우 다음과 같이 트랜잭션은 실패하고 ROLLBACK 된다.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> INSERT INTO t1 VALUES ( 3, 'leekmo' );
1 row created.
gSQL> COMMIT;
ERR-40002(16291): transaction rollback: integrity constraint violation : PUBLIC.T1_UK(1)
```

지연 제약 조건을 위반한 트랜잭션

트랜잭션이 DEFERRED로 설정된 제약 조건을 위반한 상태에서 다음 구문들을 수행할 경우 다음과 같은 에러가 발생한다.

- COMMIT

- 에러가 발생하고, 트랜잭션이 ROLLBACK 된다.
- SET CONSTRAINTS ALL IMMEDIATE
 - 구문 에러가 발생한다.
- DDL
 - 구문 에러가 발생한다.

COMMIT 할 경우 원치 않는 ROLLBACK이 발생할 수 있으므로, SET CONSTRAINTS ALL IMMEDIATE 구문을 수행하여 트랜잭션이 제약 조건을 위반한 상태인지 확인해야 한다.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> INSERT INTO t1 VALUES ( 3, 'leekmo' );
1 row created.
gSQL> SET CONSTRAINTS ALL IMMEDIATE;
ERR-23000(16038): integrity constraint violation : PUBLIC.T1_UK(1)
gSQL> SELECT * FROM t1 ORDER BY id;
ID NAME
-- -----
1 mkkim
2 leekmo
3 leekmo
3 rows selected.
gSQL> UPDATE t1 SET name = 'xcom73' WHERE id = 3;
1 row updated.
gSQL> SET CONSTRAINTS ALL IMMEDIATE;
Constraints set.
gSQL> COMMIT;
Commit complete.
```

트랜잭션 제어 언어

SET CONSTRAINTS 구문은 **16.124** SAVEPOINT *savepoint_specifier* 구문과 같이 트랜잭션 진행 중에 사용하는 트랜잭션 제어 언어이다.

SET CONSTRAINTS 구문에는 COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT 구문 등의 트랜잭션 제어가 적용된다.

다음은 여러 개의 지연 가능한 제약 조건을 가진 테이블의 예이다.

```
CREATE TABLE t1
(
  id1 INTEGER CONSTRAINT t1_uk1 UNIQUE DEFERRABLE INITIALLY IMMEDIATE,
```



```

    id2 INTEGER CONSTRAINT t1_uk2 UNIQUE DEFERRABLE INITIALLY IMMEDIATE,
    id3 INTEGER CONSTRAINT t1_uk3 UNIQUE DEFERRABLE INITIALLY IMMEDIATE
);

```

다음과 같이 트랜잭션이 진행되는 중에 <set constraints mode statement> 구문을 수행할 경우, 각 시점에 따라 지연 가능한 제약 조건들의 검사 시점이 변경된다.

- result: success

```

INSERT INTO t1 VALUES ( 1, 1, 1 );
1 row created.
COMMIT;
Commit complete.

```

- result: success

```

SAVEPOINT sp1;
Savepoint created.

```

- result: success
- t1_uk1 constraint is DEFERRED

```

SET CONSTRAINTS t1_uk1 DEFERRED;
Constraints set.

```

- result: success

```

SAVEPOINT sp2;
Savepoint created.

```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```

SET CONSTRAINTS t1_uk2 DEFERRED;
Constraints set.

```

- result: success

```

SAVEPOINT sp3;
Savepoint created.

```

- result: success
- ALL constraints are DEFERRED

```
SET CONSTRAINTS ALL DEFERRED;
Constraints set.
```

- result: success

```
SAVEPOINT sp4;
Savepoint created.
```

- result: success
- ALL constraints are IMMEDIATE

```
SET CONSTRAINTS ALL IMMEDIATE;
Constraints set.
```

다음과 같이 ROLLBACK TO SAVEPOINT 구문을 사용하여 트랜잭션을 부분 철회할 경우 SET CONSTRAINTS 구문도 함께 부분 철회되어 검사 시점이 변경된다.

- result: error

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK1) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```
ROLLBACK TO SAVEPOINT sp4;
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 3, 1, 3 );  
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 4, 4, 1 );  
1 row created.
```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```
ROLLBACK TO SAVEPOINT sp3;  
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );  
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 3, 1, 3 );  
1 row created.
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );  
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- t1_uk1 constraint is DEFERRED

```
ROLLBACK TO SAVEPOINT sp2;  
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );  
1 row created.
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- all constraint are IMMEDIATE

```
ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
```

- result: error

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK1) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: 1 row
- 1 1 1

```
SELECT * FROM t1;
ID1 ID2 ID3
--- --- ---
1 1 1
1 row selected.
```

트랜잭션을 COMMIT 하거나 ROLLBACK 할 경우, SET CONSTRAINTS 구문의 영향은 종료되며, 모든 지연 가능한 제약 조건들은 제약 조건의 특성으로 설정한 INITIALLY IMMEDIATE 또는 INITIALLY DEFERRED 값을 따른다.

사용 예

다음은 제약 조건 이름을 기술하여 검사 시점을 변경하는 예이다.

```
gSQL> SET CONSTRAINTS t1_uk1 DEFERRED;
Constraints set.
```

다음은 모든 지연 가능한 제약 조건의 검사 시점을 변경하는 예이다.

```
gSQL> SET CONSTRAINTS ALL DEFERRED;
Constraints set.
```

호환성

SQL 표준에서는 CONSTRAINT 키워드 절을 정의하지 않고 있다.

표 16-75 SQL 표준 호환성

Feature ID	설명	지원 여부
F721	Deferrable constraints	O

참조

관련 내용은 다음을 참조한다.

- 제약 조건 생성
 - CREATE TABLE
 - ALTER TABLE name ADD CONSTRAINT
 - ALTER TABLE name ADD COLUMN
 - ALTER TABLE name ALTER COLUMN
- 제약 조건 변경: ALTER TABLE name ALTER CONSTRAINT
- 제약 조건 검사시점 제어: SET CONSTRAINTS

16.130 SET SESSION AUTHORIZATION user_identifier

기능

Session user와 current user를 변경한다.

구문

```
<set session user identifier statement> ::=
    SET SESSION AUTHORIZATION user_identifier
    ;
```

사용 범위 및 접근 권한

<set session user identifier statement> 구문을 수행하려면 logon 사용자에게 ACCESS CONTROL ON DATABASE 권한이 있어야 한다.

사용자 정보는 다음과 같은 세 가지 형태로 관리된다.

- Logon user
 - Login 한 user로써 connection을 닫을 때까지 유지된다.
- Session user
 - 최초 logon user와 동일하지만 SET SESSION AUTHORIZATION 구문을 이용해 변경할 수 있다.
- Current user
 - 일반적으로 session user와 동일하지만 PSM이나 view 등을 사용할 때 접근 제어등을 위해 시스템 내부적으로 잠시 변경된다.
 - Session user와 current user는 unix system의 real user와 effective user의 차이와 유사한 개념이다.

구문 규칙 및 파라미터

user_identifier

변경할 사용자의 이름이다.

설명

SET SESSION AUTHORIZATION 구문을 수행한 이후의 모든 구문은 session user를 기준으로 수행되므로, session user에 대한 권한을 검사하고 객체를 생성할 때의 소유자 역시 session user가 된다.

사용 예

다음은 ACCESS CONTROL ON DATABASE 권한을 가진 test 사용자가 session user를 u1 사용자로 변경한 예이다.

```
gSQL> SET SESSION AUTHORIZATION u1;
Session set.
gSQL> SELECT LOGON_USER(), SESSION_USER(), CURRENT_USER FROM dual;
LOGON_USER() SESSION_USER() CURRENT_USER
-----
TEST          U1          U1
1 row selected.
```

호환성

표 16-76 SQL 표준 호환성

Feature ID	설명	지원 여부
F321	User authorization	O

16.131 SET SESSION CHARACTERISTICS AS transaction_mode

기능

세션의 트랜잭션 속성을 설정한다.

구문

```

<set session characteristics statement> ::=
    SET SESSION CHARACTERISTICS AS TRANSACTION <transaction_mode>
    ;
<transaction_mode> ::=
    { <transaction_access_mode> | ISOLATION LEVEL <isolation_level> }
<transaction_access_mode> ::=
    READ { ONLY | WRITE }
<isolation_level> ::=
    { READ COMMITTED | SERIALIZABLE }

```

구문 규칙 및 파라미터

<transaction_access_mode>

다음 트랜잭션의 ACCESS MODE 이다.

- READ ONLY
- READ WRITE

<isolation_level>

다음 트랜잭션의 ISOLATION LEVEL 이다.

- READ COMMITTED
- SERIALIZABLE

설명

SET SESSION CHARACTERISTICS은 세션의 트랜잭션 속성을 설정한다. 즉, session 내에서 생성되는 모든 transaction의 속성이 이를 따른다.

참고로 SET TRANSACTION transaction_mode 구문의 경우, 이후에 수행되는 하나의 transaction 속성만 변경한다.

사용 예

다음은 session 내에서 생성될 모든 transaction을 READ ONLY로 설정하는 예이다.

```
gSQL> SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY;
Session set.
```

다음은 session 내에서 생성될 모든 transaction의 isolation level을 READ COMMITTED로 설정하는 예이다.

```
gSQL> SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL READ COMMITTED;
Session set.
```

호환성

표 16-77 SQL 표준 호환성

Feature ID	설명	지원 여부
F761	Session management	O

참조

관련 내용은 SET TRANSACTION transaction_mode를 참조한다.

16.132 SET TIME ZONE

기능

세션의 TIMEZONE을 설정한다.

구문

```

<set local time zone statement> ::=
    SET TIME ZONE <set time zone value>
    ;
<set time zone value> ::=
    { '[+|-]hh:mm' | LOCAL }

```

구문 규칙 및 파라미터

<set time zone value>

설정할 TIMEZONE 값이다.

- hh:mm: 설정할 TIMEZONE의 GMT OFFSET이다.
 - Offset 값의 범위는 '-14:00' ~ '+14:00' 이다.
- LOCAL: Session을 생성한 시점의 TIME ZONE 이다.
 - Session을 생성할 때의 TIME ZONE은 client OS의 TIME ZONE으로 설정된다.

설명

Session의 time zone을 변경하면 함수 CURRENT_TIME, CURRENT_TIMESTAMP 등의 결과값에 영향을 미친다.

사용 예

다음은 session의 time zone을 '+09:00' 으로 변경하는 예이다.

```
gSQL> SET TIME ZONE '+09:00';  
Session set.
```

호환성

표 16-78 SQL 표준 호환성

Feature ID	설명	지원 여부
F411	Time zone specification	O

16.133 SET TRANSACTION transaction_mode

기능

다음 트랜잭션의 속성을 설정한다.

구문

```

<set transaction statement> ::=
    SET TRANSACTION <transaction_mode>
    ;
<transaction_mode> ::=
    { <transaction_access_mode> | ISOLATION LEVEL < isolation_level > }
<transaction_access_mode> ::=
    READ { ONLY | WRITE }
< isolation_level > ::=
    { READ COMMITTED | SERIALIZABLE }

```

구문 규칙 및 파라미터

<transaction_access_mode>

다음 트랜잭션의 ACCESS MODE 이다.

- READ ONLY
- READ WRITE

<isolation_level>

다음 트랜잭션의 ISOLATION LEVEL 이다.

- READ COMMITTED
- SERIALIZABLE

설명

SET TRANSACTION은 다음 트랜잭션의 속성을 설정하며, 다음 트랜잭션이 종료되면 트랜잭션 속성은 기본값으로 복원된다.

사용 예

다음에 수행될 transaction을 읽기 전용으로 설정한 예이다.

```
gSQL> SET TRANSACTION READ ONLY;  
Transaction set.
```

호환성

표 16-79 SQL 표준 호환성

Feature ID	설명	지원 여부
T251	SET TRANSACTION statement: LOCAL option	X

참조

관련 내용은 SET SESSION CHARACTERISTICS AS transaction_mode를 참조한다.

16.134 TRUNCATE TABLE

기능

테이블의 모든 row들을 제거한다.

구문

```
<truncate table statement> ::=  
    TRUNCATE TABLE table_name  
        [ RESTART IDENTITY | CONTINUE IDENTITY ]  
        [ DROP STORAGE | DROP ALL STORAGE ]  
    ;
```

사용 범위 및 접근 권한

<truncate table statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 테이블의 소유자
- 테이블에 대해 CONTROL TABLE ON TABLE
- 테이블이 속한 스키마에 대해 (DROP TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row들을 제거할 대상 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

[RESTART IDENTITY | CONTINUE IDENTITY]

- RESTART IDENTITY
 - 자동 생성값을 갖는 column (identity column)이 해당 테이블에 존재할 경우 자동으로 값을 재시작한다.
- CONTINUE IDENTITY
 - 자동 생성값을 갖는 column (identity column)이 해당 테이블에 존재할 경우 기존값을 변경하지 않는다.
- 명시하지 않을 경우, 기본값은 CONTINUE IDENTITY이다.

[DROP STORAGE | DROP ALL STORAGE]

- DROP STORAGE
 - 테이블에 할당된 extent 중에 MINSIZE 만큼만 제외하고 나머지 extent들을 삭제한다.
- DROP ALL STORAGE
 - 테이블에 할당된 모든 extent 들을 삭제한다.
- 명시하지 않을 경우, 기본값은 DROP STORAGE 이다.

설명

TRUNCATE TABLE과 같은 Data Definition Language (DDL) 구문도 트랜잭션이 COMMIT 되기 전이라면 ROLLBACK 할 수 있다.

사용 예

다음은 TRUNCATE TABLE 구문을 수행하는 예이다.

```
gSQL> TRUNCATE TABLE t1;
Table truncated.
```

다음은 TRUNCATE TABLE을 수행할 때 identity column 값을 재시작하는 예이다.

```
TRUNCATE TABLE t1 RESTART IDENTITY;
Table truncated.
```

호환성

SQL 표준에서는 [DROP STORAGE | DROP ALL STORAGE] 절을 정의하지 않고 있다.

표 16-80 SQL 표준 호환성

Feature ID	설명	지원 여부
F200	TRUNCATE TABLE statement	O
F202	TRUNCATE TABLE: identity column restart option	O

16.135 UPDATE

기능

테이블의 row들을 갱신한다.

구문

```

<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
    ;

<set clause> ::=
    column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )

<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]

<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>

<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]

<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

```

사용 범위 및 접근 권한

<update statement: searched> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- Update 대상인 모든 column에 대해 UPDATE(columns) ON TABLE
- 테이블에 대해 (UPDATE 또는 CONTROL TABLE) ON TABLE

- 테이블이 속한 스키마에 대해 (UPDATE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
- UPDATE ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 갱신할 대상 테이블의 이름이다.

schema_name.table_name과 같이 테이블이 속한 스키마를 정의할 수 있는데 schema_name을 생략할 경우 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

[AS alias_name]

table_name의 alias 이다.

<set clause>

갱신할 column과 할당할 값을 정의하며, <set clause>의 column 개수와 값의 개수는 동일해야 한다.

다음과 같은 방법으로 정의할 수 있다.

- column_name = { <value expression> | DEFAULT }

```
UPDATE table_name
    SET column1 = value1, column2 = value2, column3 = value3
```

- (column_name [, ...]) = ({ <value expression> | DEFAULT } [, ...])

```
UPDATE table_name
    SET ( column1, column2, column3 ) = ( value1, value2, value3 )
```

- (column_name [, ...]) = (<query expression>)

```
UPDATE table_name
    SET column1 = ( SELECT max(value1) FROM other_table_name )
```

<query expression>은 row 하나를 생성하는 질의여야 한다.

Column 값으로 DEFAULT를 사용할 경우, **CREATE TABLE**을 수행할 때 정의한 기본값 (<default clause> 참조)을 사용하며, 정의되지 않은 경우에는 NULL 값이 할당된다.

WHERE <search condition>

WHERE 조건을 만족하는 row를 갱신한다.

WHERE 조건을 명시하지 않을 경우, 모든 row들을 갱신한다.

WHERE 조건에 대한 자세한 내용은 SELECT 구문의 **where clause**를 참조한다.

<result offset clause>

질의 결과 중 건너될 row의 개수를 명시한다.

자세한 내용은 SELECT 구문의 <result offset clause>를 참조한다.

<fetch limit clause>

Fetch 할 row의 개수를 명시하는 구문으로써 다음 두 가지 방법을 사용할 수 있다.

- <fetch first clause>
 - Fetch 할 row의 개수를 명시한다.
 - 자세한 내용은 SELECT 구문의 <fetch first clause>를 참조한다.
- <limit clause>
 - Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너될 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
 - 자세한 내용은 SELECT 구문의 <limit clause>를 참조한다.

설명

UPDATE 관련 구문들의 차이점

- UPDATE
 - 조건에 부합하는 다수의 row를 갱신한다.
 - 예: UPDATE t1 SET c2 = c2 + 1 WHERE c1 > 0;
- UPDATE name WHERE CURRENT OF cursor_name
 - 현재 cursor가 가리키는 row를 갱신한다.
 - 예: UPDATE t1 WHERE CURRENT OF cursor;
- UPDATE name RETURNING
 - 조건에 부합하는 다수의 row를 갱신하며, 갱신한 row들을 SELECT 구문과 동일한 방식 (SQLFetch() 등의 API)으로 검색할 수 있다.
 - 예: UPDATE t1 SET c2 = c2 + 1 WHERE c1 > 0 RETURNING c2;
- UPDATE name RETURNING .. INTO
 - 한 건 이하의 row를 갱신할 수 있으며, 갱신한 row가 한 건일 경우 RETURNING INTO 절의 호스트 변수에 값을 얻어온다.

- 예: UPDATE t1 SET c2 = c2 + 1 WHERE c1 = 0 RETURNING c2 INTO :v1;

사용 예

다음은 조건에 부합하는 다수의 row를 갱신하는 예이다.

```
gSQL> UPDATE lineitem
      SET l_shipdate = CURRENT_DATE
      WHERE l_returnflag = 'R';
5 rows updated.
```

다음은 여러 column의 값을 갱신하는 예이다.

```
gSQL> UPDATE lineitem
      SET l_shipdate = CURRENT_DATE
        , l_returnflag = 'A'
      WHERE l_returnflag = 'R';
5 rows updated.
```

다음은 여러 column을 괄호로 묶어 갱신하는 예이다.

```
gSQL> UPDATE lineitem
      SET ( l_shipdate , l_returnflag )
        = ( CURRENT_DATE, 'A' )
      WHERE l_returnflag = 'R';
5 rows updated.
```

다음은 subquery를 사용하여 column의 값을 갱신하는 예이다.

```
gSQL> UPDATE lineitem
      SET l_discount = ( SELECT MAX(l_discount) + 0.01 FROM lineitem )
      WHERE l_returnflag = 'R';
5 rows updated.
```

다음은 OFFSET과 FETCH 절을 사용하여 조건에 부합하는 row들 중 일부만 갱신하는 예이다.

```
gSQL> UPDATE lineitem
      SET l_discount = l_discount + 0.01
      WHERE l_returnflag = 'R'
      OFFSET 3
      FETCH 2;
```

2 rows updated.

호환성

SQL 표준은 UPDATE 구문에서 다음 절을 정의하지 않고 있다.

- <result offset clause>
- <fetch limit clause>

표 16-81 SQL 표준 호환성

Feature ID	설명	지원 여부
F781	Self-referencing operations	X
T111	Updatable joins, unions, and columns	X

16.136 UPDATE name RETURNING

기능

테이블의 row들을 갱신하고, 갱신 전의 row들이나 갱신 후의 row들을 검색한다.

구문

```

<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning clause>

<set clause> ::=
    column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )

<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]

<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>

<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]

<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

<returning clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name ] } [,
    ...] }

```

사용 범위 및 접근 권한

〈update returning query statement〉 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- UPDATE 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Update 대상인 모든 column에 대해 UPDATE(columns) ON TABLE
 - 테이블에 대해 (UPDATE 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (UPDATE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - UPDATE ANY TABLE ON DATABASE
- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 갱신할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias이다.

〈set clause〉

갱신할 column과 할당할 값을 정의하며, 〈set clause〉의 column 개수와 값의 개수는 동일해야 한다. 자세한 내용은 UPDATE 구문을 참조한다.

WHERE 〈search condition〉

WHERE 조건을 만족하는 row를 갱신한다.

WHERE 조건을 명시하지 않을 경우, 모든 row들을 갱신한다.

WHERE 조건에 대한 자세한 내용은 SELECT 구문의 **where clause**를 참조한다.

〈result offset clause〉

질의 결과 중 건너될 row의 개수를 명시한다.

자세한 내용은 SELECT 구문의 〈result offset clause〉를 참조한다.

〈fetch limit clause〉

Fetch 할 row의 개수를 명시하는 구문으로써 다음 두 가지 방법을 사용할 수 있다.

- 〈fetch first clause〉
 - Fetch 할 row의 개수를 명시한다.
 - 자세한 내용은 SELECT 구문의 〈fetch first clause〉를 참조한다.
- 〈limit clause〉
 - Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너될 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
 - 자세한 내용은 SELECT 구문의 〈limit clause〉를 참조한다.

〈returning clause〉

갱신된 row들을 결과 집합으로 하고, 이들 중에서 검색할 column을 기술한다.

- RETURN과 RETURNING은 동일한 의미의 키워드이다.
- NEW | OLD
 - NEW: 갱신된 row들 중에 갱신 후 row를 기준으로 검색한다.
 - OLD: 갱신된 row들 중에 갱신 전 row를 기준으로 검색한다.
 - 생략할 경우, 기본값은 NEW 이다.
- 〈value expression〉
 - SELECT 구문의 〈select list〉와 동일하지만 aggregation 등을 사용할 수 없다.
- [[AS] alias_name]
 - AS 절을 사용하여 〈value expression〉의 이름을 지정할 수 있다.

설명

자세한 내용은 UPDATE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 RETURNING 절을 사용하여 갱신된 row들의 값을 얻는 예이다.

```
gSQL> UPDATE lineitem
      SET l_discount = l_discount + 0.01
      WHERE l_returnflag = 'R'
      RETURNING l_orderkey, l_linenum, l_discount;
```

L_ORDERKEY	L_LINENUMBER	L_DISCOUNT
8	1	.07
9	2	.11
12	5	.05
15	1	.03
16	2	.08

8	1	.07
9	2	.11
12	5	.05
15	1	.03
16	2	.08

5 rows updated.

다음은 RETURNING OLD 절을 사용하여 갱신된 row들의 갱신 전 값을 얻는 예이다.

```
gSQL> UPDATE lineitem
      SET l_discount = l_discount + 0.01
      WHERE l_returnflag = 'R'
      RETURNING OLD l_orderkey, l_linenum, l_discount;
```

L_ORDERKEY	L_LINENUMBER	L_DISCOUNT
8	1	.06
9	2	.1
12	5	.04
15	1	.02
16	2	.07

8	1	.06
9	2	.1
12	5	.04
15	1	.02
16	2	.07

5 rows updated.

호환성

SQL 표준에는 <update returning query statement> 구문이 존재하지 않는다.

16.137 UPDATE name RETURNING .. INTO

기능

테이블 row 한 개를 갱신하고, 갱신한 row의 값을 호스트 변수에 얻어온다.

구문

```

<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning into clause>
    ;

<set clause> ::=
    column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )

<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]

<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>

<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]

<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

<returning into clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name] } [,
    ...] } INTO variable_name [, ...]

```

사용 범위 및 접근 권한

〈update returning query statement〉 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- UPDATE 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Update 대상인 모든 column에 대해 UPDATE(columns) ON TABLE
 - 테이블에 대해 (UPDATE 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (UPDATE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - UPDATE ANY TABLE ON DATABASE
- RETURNING 절에 사용된 모든 column에 대해 다음 권한 중 하나가 있어야 한다.
 - RETURNING 구문에 사용된 모든 column에 대해 SELECT(columns) ON TABLE
 - 테이블에 대해 (SELECT 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (SELECT TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - SELECT ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 갱신할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias이다.

〈set clause〉

갱신할 column과 할당할 값을 정의하며, 〈set clause〉의 column 개수와 값의 개수는 동일해야 한다. 자세한 내용은 UPDATE 구문을 참조한다.

WHERE 〈search condition〉

WHERE 조건을 만족하는 row를 갱신한다.

WHERE 조건을 명시하지 않을 경우, 모든 row들을 갱신한다.

WHERE 조건에 대한 자세한 내용은 SELECT 구문의 **where clause**를 참조한다.

〈result offset clause〉

질의 결과 중 건너될 row의 개수를 명시한다.

자세한 내용은 SELECT 구문의 〈result offset clause〉를 참조한다.

〈fetch limit clause〉

Fetch 할 row의 개수를 명시하는 구문으로써 다음 두 가지 방법을 사용할 수 있다.

- 〈fetch first clause〉
 - Fetch 할 row의 개수를 명시한다.
 - 자세한 내용은 SELECT 구문의 〈fetch first clause〉를 참조한다.
- 〈limit clause〉
 - Fetch 할 row의 개수를 명시하거나 질의 결과 중 건너될 row의 개수와 fetch 할 row의 개수를 동시에 명시한다.
 - 자세한 내용은 SELECT 구문의 〈limit clause〉를 참조한다.

RETURNING .. AS ..

갱신된 row들을 결과 집합으로 하고, 이들 중에서 검색할 column을 기술한다.

자세한 내용은 UPDATE name RETURNING 구문의 〈returning clause〉를 참조한다.

INTO variable_name [, ...]

INTO 절에 기술된 변수의 개수는 RETURNING 절에 기술된 expression의 개수와 동일해야 한다.

갱신할 row가 한 건 이하여야 한다.

Row가 두 건 이상 갱신될 경우, 에러가 발생한다.

설명

자세한 내용은 UPDATE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 갱신한 row의 column 값을 host 변수에 얻어오는 예이다.

- Host 변수를 선언한다.

```
gSQL> \VAR v_discount NUMBER
gSQL> UPDATE lineitem
      SET l_discount = l_discount + 0.01
      WHERE l_orderkey = 12 AND l_linenumber = 5
      RETURNING l_discount INTO :v_discount;
V_DISCOUNT
-----
          .05
1 row updated.
```

호환성

SQL 표준에는 <update returning into statement> 구문이 존재하지 않는다.

16.138 UPDATE name WHERE CURRENT OF cursor_name

기능

커서가 가리키는 row 하나를 갱신한다.

구문

```
<update statement: positioned> ::=  
    UPDATE table_name [ [ AS ] alias_name ]  
        SET <set clause> [, ...]  
        WHERE CURRENT OF cursor_name  
    ;
```

사용 범위 및 접근 권한

<update statement: positioned> 구문을 수행하려면 사용자가 다음 조건을 만족해야 한다.

- UPDATE 구문을 수행하려면 다음 권한 중 하나가 있어야 한다.
 - Update 대상인 모든 column에 대해 UPDATE(columns) ON TABLE
 - 테이블에 대해 (UPDATE 또는 CONTROL TABLE) ON TABLE
 - 테이블이 속한 스키마에 대해 (UPDATE TABLE 또는 CONTROL SCHEMA) ON SCHEMA
 - UPDATE ANY TABLE ON DATABASE

구문 규칙 및 파라미터

table_name

Row를 갱신할 대상 테이블의 이름이다.

[AS alias_name]

table_name의 alias 이다.

<set clause>

갱신할 column과 할당할 값을 정의하며, <set clause>의 column 개수와 값의 개수는 동일해야 한다. 자세한 내용은 UPDATE 구문을 참조한다.

cursor_name

cursor_name에 해당하는 커서는 다음 조건들을 만족해야 한다.

- OPEN 된 커서이어야 한다. (OPEN cursor_name 참조)
- 커서를 이용해 FETCH 한 row가 있어야 한다. (FETCH cursor_name 참조)
- 커서에 사용된 질의가 table_name을 식별할 수 있어야 한다. (DECLARE cursor_name 참조)
- table_name에 대해 갱신할 수 있는 커서이어야 한다. (DECLARE cursor_name 참조)

설명

자세한 내용은 UPDATE 관련 구문들의 차이점을 참조한다.

사용 예

다음은 interactive SQL (gsq)에서 cursor를 사용하여 <update statement: positioned> 구문을 수행하는 예이다.

- Host 변수를 선언한다.

```
gSQL> \VAR v_discount NUMBER
```

- Cursor를 선언한다.

```
gSQL> DECLARE update_cursor CURSOR FOR
      SELECT l_discount
      FROM lineitem
      WHERE l_orderkey = 8 AND l_linenumber = 1
      FOR UPDATE;
```

Cursor declared.

- Cursor를 open한다.

```
gSQL> OPEN update_cursor;
Cursor is open.
```

- Row를 fetch한다.

```
gSQL> FETCH update_cursor INTO :v_discount;
V_DISCOUNT
-----
          .06
1 row fetched.
```

- Current row를 update한다.

```
gSQL> UPDATE lineitem
      SET l_discount = l_discount + 0.01
      WHERE CURRENT OF update_cursor;
1 row updated.
```

- Cursor를 close한다.

```
gSQL> CLOSE update_cursor;
Cursor closed.
gSQL> COMMIT;
Commit complete.
```

다음은 embedded SQL 프로그램에서 cursor를 사용하여 <update statement: positioned> 구문을 수행하는 예이다.

```
{
...
EXEC SQL BEGIN DECLARE SECTION;
...
double v_discount;
...
EXEC SQL END DECLARE SECTION;
...
EXEC SQL DECLARE update_cursor CURSOR FOR
      SELECT l_discount
      FROM lineitem
      WHERE l_orderkey = 8 AND l_linenumber = 1
```



```
        FOR UPDATE;

...
EXEC SQL OPEN update_cursor;

...
EXEC SQL FETCH NEXT update_cursor INTO :v_discount;

...
EXEC SQL UPDATE lineitem
        SET l_discount = l_discount + 0.01
        WHERE CURRENT OF update_cursor;

...
EXEC SQL CLOSE update_cursor;

...
EXEC SQL COMMIT WORK;

...
}
```

호환성

표 16-82 SQL 표준 호환성

Feature ID	설명	지원 여부
F831	Full cursor update	O
B031	Basic dynamic SQL	O

참조

관련 내용은 CLOSE cursor_name을 참조한다.

파트 IV.

PSM Manual

17. Overview of PSM	2,105
17.1 PSM의 특징점	2,106
SQL과의 긴밀한 연동	2,106
성능 향상	2,106
생산성 향상	2,106
이식성	2,106
관리의 용이성	2,107
17.2 Language Elements	2,108
Data Types	2,108
Variables	2,108
Control Structures	2,108
Subprograms	2,108
17.3 Processing Transaction in PSM	2,109
18. PSM DataTypes	2,111
18.1 Built-in Data Types	2,112
숫자 타입	2,112
CHARACTER STRING 타입	2,112
BINARY STRING 타입	2,113
날짜/시간 타입	2,113
INTERVAL 타입	2,113
BOOLEAN 타입	2,113
ROWID 타입	2,114
Built-in 데이터 타입 변수 선언	2,114
18.2 Attribute Data Types	2,115
%TYPE	2,115
%ROWTYPE	2,115
Constraint 속성 상속	2,116
18.3 User-defined Record Type	2,117
18.4 User-defined Collection Type	2,119
Associative Array	2,119
Assign Values to Collection Variables	2,121
Collection Method	2,124
18.5 SYS_REFCURSOR	2,127
19. PSM Control Statements	2,129
19.1 Assignment	2,130
Assignment Target	2,130
Assigning Expression	2,130
Assignment 호환성	2,132

19.2	PL Block	2,134
	PL Block의 구성	2,134
19.3	NULL Statement	2,137
19.4	Testing Conditions	2,138
	IF	2,138
	CASE	2,139
19.5	Iterative Control	2,141
	Basic Loop	2,141
	FOR Loop	2,142
	WHILE Loop	2,143
19.6	Sequential Control	2,144
	GOTO	2,144
	CONTINUE	2,146
	EXIT	2,148
19.7	Error Handling	2,150
	Errors at Compile Time	2,150
	Errors at Run-time	2,152
	Cursor Attributes	2,153
	EXCEPTION Handling	2,156
	PRAGMA EXCEPTION_INIT	2,161
	DBMS_STANDARD.RAISE_APPLICATION_ERROR	2,162
20.	PSM Cursor Statements	2,165
20.1	Declaration	2,166
20.2	OPEN	2,170
20.3	FETCH	2,171
20.4	CLOSE	2,173
20.5	EXPLICIT CURSOR ATTRIBUTES	2,174
20.6	IMPLICIT_CURSOR_ATTRIBUTES	2,176
20.7	CURSOR VARIABLES	2,178
	OPEN and Close Cursor Variables	2,180
	Fetching Data With Cursor Variables	2,181
	Assign Values to Cursor Variables	2,181
21.	Using PSM Subprograms	2,185
21.1	Anonymous PL Block	2,186
21.2	Nested Procedure	2,187
21.3	Nested Function	2,189
21.4	Schema-level Procedure	2,190
	생성	2,190
	사용	2,191

제거	2,192
Recompile	2,193
21.5 Schema-level Function	2,194
생성	2,194
사용	2,194
제거	2,195
21.6 Built-in Procedures	2,197
22. Using SQLs in PSM	2,199
22.1 Static SQLs	2,200
SELECT	2,200
INSERT	2,203
UPDATE	2,207
DELETE	2,208
RETURNING INTO	2,209
COMMIT, ROLLBACK, SAVEPOINT	2,214
22.2 Dynamic SQL	2,217
EXECUTE IMMEDIATE	2,217
OPEN FOR, FETCH and CLOSE	2,219
23. PSM Language Element References	2,221
23.1 Assignment Statement	2,222
기능	2,222
구문	2,222
사용 범위 및 접근 권한	2,222
구문 규칙 및 파라미터	2,222
설명	2,223
사용 예	2,223
호환성	2,224
참조	2,225
23.2 Basic LOOP Statement	2,226
기능	2,226
구문	2,226
사용 범위 및 접근 권한	2,226
구문 규칙 및 파라미터	2,226
설명	2,226
사용 예	2,227
호환성	2,227
참조	2,227
23.3 Block (BEGIN .. END)	2,228
기능	2,228

구문	2,228
사용 범위 및 접근 권한	2,228
구문 규칙 및 파라미터	2,229
설명	2,229
사용 예	2,229
호환성	2,230
참조	2,230
23.4 CASE Statement	2,231
기능	2,231
구문	2,231
사용 범위 및 접근 권한	2,231
구문 규칙 및 파라미터	2,232
설명	2,232
사용 예	2,232
호환성	2,233
23.5 CLOSE Statement	2,234
기능	2,234
구문	2,234
사용 범위 및 접근 권한	2,234
구문 규칙 및 파라미터	2,234
설명	2,234
사용 예	2,235
호환성	2,235
참조	2,235
23.6 Collection Method Invocation	2,236
기능	2,236
구문	2,236
사용 범위 및 접근 권한	2,236
구문 규칙 및 파라미터	2,236
설명	2,237
사용 예	2,237
참조	2,239
23.7 COLLECTION Variable Declaration	2,240
기능	2,240
구문	2,240
사용 범위 및 접근 권한	2,240
구문 규칙 및 파라미터	2,241
설명	2,241
사용 예	2,241

호환성	2,243
참조	2,243
23.8 CONTINUE Statement	2,244
기능	2,244
구문	2,244
사용 범위 및 접근 권한	2,244
구문 규칙 및 파라미터	2,244
설명	2,245
사용 예	2,245
호환성	2,246
참조	2,246
23.9 Cursor FOR LOOP Statement	2,247
기능	2,247
구문	2,247
사용 범위 및 접근 권한	2,247
구문 규칙 및 파라미터	2,248
설명	2,248
사용 예	2,248
참조	2,250
23.10 Cursor Variable Declaration	2,251
기능	2,251
구문	2,251
사용 범위 및 접근 권한	2,251
구문 규칙 및 파라미터	2,251
설명	2,251
사용 예	2,252
참조	2,252
23.11 DELETE Statement Extension	2,253
기능	2,253
구문	2,253
사용 범위 및 접근 권한	2,254
구문 규칙 및 파라미터	2,254
설명	2,254
사용 예	2,254
참조	2,255
23.12 EXCEPTION_INIT Pragma	2,256
기능	2,256
구문	2,256
사용 범위 및 접근 권한	2,256

구문 규칙 및 파라미터	2,256
설명	2,256
사용 예	2,257
호환성	2,257
참조	2,257
23.13 Exception Declaration	2,258
기능	2,258
구문	2,258
사용 범위 및 접근 권한	2,258
구문 규칙 및 파라미터	2,258
설명	2,258
사용 예	2,258
호환성	2,259
참조	2,259
23.14 Exception Handler	2,260
기능	2,260
구문	2,260
사용 범위 및 접근 권한	2,260
구문 규칙 및 파라미터	2,260
설명	2,261
사용 예	2,261
호환성	2,262
참조	2,262
23.15 EXECUTE IMMEDIATE Statement	2,263
기능	2,263
구문	2,263
사용 범위 및 접근 권한	2,263
구문 규칙 및 파라미터	2,264
설명	2,265
사용 예	2,265
23.16 EXIT Statement	2,267
기능	2,267
구문	2,267
사용 범위 및 접근 권한	2,267
구문 규칙 및 파라미터	2,267
설명	2,267
사용 예	2,268
호환성	2,268
23.17 Explicit Cursor Attribute	2,269

기능	2,269
구문	2,269
사용 범위 및 접근 권한	2,269
구문 규칙 및 파라미터	2,269
설명	2,269
사용 예	2,270
호환성	2,271
23.18 Explicit Cursor Declaration and Definition	2,272
기능	2,272
구문	2,272
사용 범위 및 접근 권한	2,272
구문 규칙 및 파라미터	2,272
설명	2,273
사용 예	2,274
호환성	2,275
참조	2,275
23.19 FETCH Statement	2,276
기능	2,276
구문	2,276
사용 범위 및 접근 권한	2,276
구문 규칙 및 파라미터	2,276
설명	2,276
사용 예	2,277
호환성	2,278
참조	2,278
23.20 FOR LOOP Statement	2,279
기능	2,279
구문	2,279
사용 범위 및 접근 권한	2,279
구문 규칙 및 파라미터	2,279
설명	2,280
사용 예	2,280
호환성	2,280
참조	2,281
23.21 Function Declaration and Definition	2,282
기능	2,282
구문	2,282
사용 범위 및 접근 권한	2,282
구문 규칙 및 파라미터	2,282

설명	2,283
사용 예	2,283
호환성	2,284
참조	2,284
23.22 GOTO Statement	2,285
기능	2,285
구문	2,285
사용 범위 및 접근 권한	2,285
구문 규칙 및 파라미터	2,285
설명	2,285
사용 예	2,286
호환성	2,286
참조	2,286
23.23 IF Statement	2,287
기능	2,287
구문	2,287
사용 범위 및 접근 권한	2,287
구문 규칙 및 파라미터	2,287
설명	2,288
사용 예	2,288
호환성	2,288
23.24 Implicit Cursor Attribute	2,289
기능	2,289
구문	2,289
사용 범위 및 접근 권한	2,289
설명	2,289
사용 예	2,289
호환성	2,290
23.25 INSERT Statement Extension	2,291
기능	2,291
구문	2,291
사용 범위 및 접근 권한	2,291
구문 규칙 및 파라미터	2,292
설명	2,292
사용 예	2,292
호환성	2,293
23.26 NULL Statement	2,294
기능	2,294
구문	2,294

사용 범위 및 접근 권한	2,294
설명	2,294
사용 예	2,294
호환성	2,295
23.27 OPEN Statement	2,296
기능	2,296
구문	2,296
사용 범위 및 접근 권한	2,296
구문 규칙 및 파라미터	2,296
설명	2,296
사용 예	2,297
호환성	2,297
참조	2,298
23.28 OPEN FOR Statement	2,299
기능	2,299
구문	2,299
사용 범위 및 접근 권한	2,299
구문 규칙 및 파라미터	2,299
설명	2,299
사용 예	2,300
호환성	2,301
참조	2,301
23.29 Procedure Call	2,302
기능	2,302
구문	2,302
사용 범위 및 접근 권한	2,302
구문 규칙 및 파라미터	2,302
설명	2,303
사용 예	2,303
호환성	2,303
23.30 Procedure Declaration and Definition	2,304
기능	2,304
구문	2,304
사용 범위 및 접근 권한	2,304
구문 규칙 및 파라미터	2,304
설명	2,305
사용 예	2,305
호환성	2,305
참조	2,305

23.31	RAISE Statement	2,306
	기능	2,306
	구문	2,306
	사용 범위 및 접근 권한	2,306
	구문 규칙 및 파라미터	2,306
	설명	2,306
	사용 예	2,307
	호환성	2,307
	참조	2,308
23.32	Record Variable Declaration	2,309
	기능	2,309
	구문	2,309
	사용 범위 및 접근 권한	2,309
	구문 규칙 및 파라미터	2,309
	설명	2,310
	사용 예	2,310
	호환성	2,310
23.33	RETURN Statement	2,311
	기능	2,311
	구문	2,311
	사용 범위 및 접근 권한	2,311
	구문 규칙 및 파라미터	2,311
	설명	2,311
	사용 예	2,312
	호환성	2,312
23.34	RETURNING INTO clause	2,313
	기능	2,313
	구문	2,313
	사용 범위 및 접근 권한	2,313
	구문 규칙 및 파라미터	2,313
	설명	2,313
	사용 예	2,314
	호환성	2,314
23.35	%ROWTYPE Attribute	2,315
	기능	2,315
	구문	2,315
	사용 범위 및 접근 권한	2,315
	구문 규칙 및 파라미터	2,315
	설명	2,315

사용 예	2,316
호환성	2,316
23.36 Scalar Variable Declaration	2,317
기능	2,317
구문	2,317
사용 범위 및 접근 권한	2,317
구문 규칙 및 파라미터	2,317
설명	2,318
사용 예	2,318
호환성	2,319
23.37 SELECT INTO Statement	2,320
기능	2,320
구문	2,320
사용 범위 및 접근 권한	2,320
구문 규칙 및 파라미터	2,320
설명	2,320
사용 예	2,321
호환성	2,321
23.38 SQLCODE Function	2,322
기능	2,322
구문	2,322
사용 범위 및 접근 권한	2,322
구문 규칙 및 파라미터	2,322
설명	2,322
사용 예	2,322
호환성	2,323
23.39 SQLERRM Function	2,324
기능	2,324
구문	2,324
사용 범위 및 접근 권한	2,324
구문 규칙 및 파라미터	2,324
설명	2,324
사용 예	2,324
호환성	2,325
23.40 %TYPE Attribute	2,326
기능	2,326
구문	2,326
사용 범위 및 접근 권한	2,326
구문 규칙 및 파라미터	2,326

설명	2,326
사용 예	2,327
호환성	2,327
23.41 UPDATE Statement Extension	2,328
기능	2,328
구문	2,328
사용 범위 및 접근 권한	2,329
구문 규칙 및 파라미터	2,329
설명	2,329
사용 예	2,329
호환성	2,330
23.42 WHILE LOOP Statement	2,331
기능	2,331
구문	2,331
사용 범위 및 접근 권한	2,331
구문 규칙 및 파라미터	2,331
설명	2,331
사용 예	2,332
호환성	2,332
참조	2,332
24. PSM SQL References	2,333
24.1 ALTER FUNCTION	2,334
기능	2,334
구문	2,334
사용 범위 및 접근 권한	2,334
구문 규칙 및 파라미터	2,334
설명	2,334
사용 예	2,335
호환성	2,335
참조	2,335
24.2 ALTER PROCEDURE	2,336
기능	2,336
구문	2,336
사용 범위 및 접근 권한	2,336
구문 규칙 및 파라미터	2,336
설명	2,336
사용 예	2,337
호환성	2,337
참조	2,338

24.3 CALL Statement	2,339
기능	2,339
구문	2,339
사용 범위 및 접근 권한	2,339
구문 규칙 및 파라미터	2,339
설명	2,340
사용 예	2,340
호환성	2,341
24.4 CREATE FUNCTION	2,342
기능	2,342
구문	2,342
사용 범위 및 접근 권한	2,342
구문 규칙 및 파라미터	2,343
설명	2,344
사용 예	2,344
호환성	2,344
참조	2,346
24.5 CREATE PROCEDURE	2,347
기능	2,347
구문	2,347
사용 범위 및 접근 권한	2,347
구문 규칙 및 파라미터	2,348
설명	2,349
사용 예	2,349
호환성	2,349
참조	2,351
24.6 DROP FUNCTION	2,352
기능	2,352
구문	2,352
사용 범위 및 접근 권한	2,352
구문 규칙 및 파라미터	2,352
설명	2,353
사용 예	2,353
호환성	2,353
참조	2,353
24.7 DROP PROCEDURE	2,355
기능	2,355
구문	2,355
사용 범위 및 접근 권한	2,355

구문 규칙 및 파라미터 2,355

설명 2,356

사용 예 2,356

호환성 2,356

참조 2,357

17.

Overview of PSM

17.1 PSM의 특징점

SQL과의 긴밀한 연동

GOLDILOCKS PSM은 GOLDILOCKS SQL과 긴밀하게 연동하여 사용할 수 있다.

- GOLDILOCKS SQL에서 지원하는 모든 데이터 타입을 지원한다.
- Attribute type (%TYPE, %ROWTYPE)을 지원하므로 테이블과 column 타입을 유연하게 사용할 수 있다.
- GOLDILOCKS SQL에서 지원하는 모든 연산자와 built-in 함수를 지원한다.
- GOLDILOCKS SQL에서 지원하는 모든 DML, DCL (COMMIT/ ROLLBACK 등) 들을 지원한다.
- Cursor를 선언하고 OPEN, FETCH, CLOSE 구문을 사용하여 SQL SELECT 구문을 지원한다.
- Dynamic SQL statement 기능을 통해 SQL DDL 구문을 지원한다.

성능 향상

GOLDILOCKS PSM은 서버 내부에서만 실행되므로 사용자 응용 프로그램과 DBMS 서버 사이의 통신 횟수를 줄여 전체적인 성능을 향상시킨다.

생산성 향상

GOLDILOCKS PSM의 언어는 script 언어와 유사하므로 적은 노력으로 원하는 기능을 위한 코드를 작성할 수 있다. 사용자의 업무 로직을 모듈화하여 procedure/ 함수로 제작하면 client 응용 프로그램들을 작성하는 시간을 줄일 수 있다.

이식성

GOLDILOCKS PSM으로 작성된 procedure/ 함수는 ODBC, JDBC, embedded SQL 및 다양한 개발 tool 들에서 동일하게 사용할 수 있다. 또한 서버나 client의 platform 종류에 관계없이 동일하게 작동하므로 종류가 다른 platform 에도 쉽게 이식할 수 있다.

관리의 용이성

GOLDILOCKS PSM은 개별 client에 각각 존재하는 유사 로직을 단 하나의 모듈로 서버에 구현하므로 관리가 용이하며 해당 모듈을 사용하는 도중에 필요에 따라 변경할 수도 있다.

17.2 Language Elements

Data Types

GOLDILOCKS PSM은 GOLDILOCKS SQL에서 제공하는 모든 built-in data type들을 제공하며, 사용자가 정의하는 record와 collection 타입까지 지원한다.

자세한 내용은 **PSM DataTypes**를 참조한다.

Variables

GOLDILOCKS PSM에서 사용자가 필요에 의해 선언한 변수는 모든 expression에서 사용할 수 있다.

자세한 내용은 다음을 참조한다.

- 선언부 (Declarative Part)
- Assignment

Control Structures

GOLDILOCKS PSM은 일반적인 script 언어들이 제공하는 판단 분기문, GOTO와 같은 무조건 분기문, 반복 수행을 위한 loop 구문들을 대부분 지원한다.

자세한 내용은 **PSM Control Statements**를 참조한다.

Subprograms

GOLDILOCKS PSM subprogram은 이름을 가지고 있고 반복적으로 수행될 수 있는 PSM block이다. 인자를 갖는 subprogram의 경우, 호출할 때마다 서로 다른 인자를 주어 실행시킬 수 있다.

Subprogram은 procedure나 함수 두 가지 형태 중 하나이며 함수 형태는 반환값을 가진다. 또한 특정 block 내에서 선언되고 그 안에서만 사용되는 nested subprogram도 지원한다.

자세한 내용은 **Using PSM Subprograms**를 참조한다.

17.3 Processing Transaction in PSM

데이터베이스 트랜잭션은 한 개 이상의 SQL 문장으로 구성된 분해할 수 없는 작업 단위이다. GOLDILOCKS 데이터베이스에서 트랜잭션을 사용하는 SQL 구문은 다음과 같다.

- SELECT 구문을 제외한 DML들
- 모든 DDL 들

다음과 같은 경우에 트랜잭션이 시작된다.

- 접속 직후에 처음으로 트랜잭션을 사용하는 SQL을 수행할 때
- COMMIT이나 ROLLBACK 이후에 처음으로 트랜잭션을 사용하는 SQL을 수행할 때

사용자가 COMMIT이나 ROLLBACK을 수행하거나 접속을 종료할 경우, 트랜잭션이 종료된다.

GOLDILOCKS PSM으로 작성된 subprogram 모듈은 자체적으로는 트랜잭션을 사용하지 않는 구문이기 때문에 호출될 때 새로운 트랜잭션을 발생시키지 않는다.

다만, SQL statement의 atomicity를 보장하기 위해 해당 subprogram이 호출된 상황에 따라 다음과 같이 subprogram 내에서 사용될 수 있는 SQL의 종류가 제한된다.

- 사용자가 직접 CALL 구문을 사용하여 호출하거나 anonymous block을 수행한 경우
 - 상위 statement가 없는 상황이므로, subprogram 내에서 모든 종류의 SQL을 사용할 수 있고 COMMIT이나 ROLLBACK도 가능하다.
- SELECT를 제외한 DML 구문 (INSERT/ UPDATE/ DELETE 등) 내에서 사용되었을 경우
 - 상위 statement가 transaction을 가지기 때문에 subprogram 내에서는 모든 SQL구문을 사용할 수 있지만 상위 statement의 atomicity를 보장하기 위해 COMMIT이나 ROLLBACK은 할 수 없다.
- SELECT 구문 내에서 호출된 경우
 - 상위 statement가 트랜잭션을 사용하지 않기 때문에 호출된 subprogram 내에서도 트랜잭션을 사용하는 모든 SQL을 사용할 수 없고 COMMIT이나 ROLLBACK도 불가능하며 SELECT 구문만 사용할 수 있다.

18.

PSM DataTypes

18.1 Built-in Data Types

GOLDILOCKS PSM은 GOLDILOCKS SQL에서 제공하는 모든 기본 데이터 타입들을 동일하게 지원한다.

기본 데이터 타입들의 종류는 다음과 같다.

자세한 내용은 **Data Type**을 참조한다.

숫자 타입

- NUMBER
- NUMERIC
- FLOAT
- NATIVE_INTEGER
- NATIVE_DOUBLE

CHARACTER STRING 타입

- CHARACTER (CHAR)
- CHARACTER VARYING (VARCHAR, VARCHAR2)
- CHARACTER LONG VARYING (LONG VARCHAR)

노트

예외적으로 GOLDILOCKS SQL에서는 지원하지 않는 타입 중에 precision이 명시되지 않은 VARCHAR (VARCHAR2, CHAR VARYING, CHARACTER VARYING) 타입도 GOLDILOCKS PSM에서는 다른 데이터베이스와의 호환을 위해 제공한다.

이 타입은 일반 변수를 선언할 때는 사용할 수 없고 subprogram의 인자나 반환 타입, 그리고 커서의 인자 타입을 명시할 때만 사용할 수 있다. 이 타입이 명시되면 해당 인자나 반환 타입은 VARCHAR 타입 중 최대 크기를 가질 수 있는 타입으로 결정된다. (precision = 4000)

BINARY STRING 타입

- BINARY
- BINARY VARYING (VARBINARY)
- BINARY LONG VARYING (LONG VARBINARY)

날짜/시간 타입

- DATE
- TIME [WITH/WITHOUT TIME ZONE]
- TIMESTAMP [WITH/WITHOUT TIME ZONE]

INTERVAL 타입

- INTERVAL YEAR
- INTERVAL MONTH
- INTERVAL YEAR TO MONTH
- INTERVAL DAY
- INTERVAL HOUR
- INTERVAL MINUTE
- INTERVAL SECOND
- INTERVAL DAY TO HOUR
- INTERVAL DAY TO MINUTE
- INTERVAL DAY TO SECOND
- INTERVAL HOUR TO MINUTE
- INTERVAL HOUR TO SECOND
- INTERVAL MINUTE TO SECOND

BOOLEAN 타입

BOOLEAN

ROWID 타입

ROWID

Built-in 데이터 타입 변수 선언

변수는 anonymous block이나 procedure, 함수 내부의 각 선언부 (declaration section)에서 선언할 수 있다.

```
DECLARE
  V_MSG VARCHAR(20) := 'HELLO, WORLD!';
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'My First Message Is : ' || V_MSG );
END;
/
```

자세한 내용은 **Built-in Data Type References**를 참조한다.

18.2 Attribute Data Types

다른 PSM 변수나 커서, 테이블, 또는 테이블의 특정 column 등의 타입을 명시할 때 사용되는 데이터 타입이다.

Attribute 타입으로 선언된 변수나 함수의 대상 객체 (테이블 등)가 변경되면 PSM (procedure, 함수)은 자동으로 변경된 타입에 맞춰 다시 컴파일 되어 적용된다.

%TYPE

다른 변수나 특정 테이블의 column 등의 타입을 명시할 때 사용된다. 참조할 수 있는 대상은 다음과 같다.

- Scalar 타입 (built-in 데이터 타입 포함) 변수
- 사용자 정의 레코드 타입 변수
- 레코드 타입 변수의 특정 필드
- Collection 타입 변수
- Collection 타입 변수의 특정 필드
- 테이블의 특정 column

%TYPE은 다음과 같이 사용된다.

```
CREATE TABLE EMP ( ID INTEGER, NAME VARCHAR(32) );
INSERT INTO EMP VALUES ( 1001, 'Tom Jackson' );
COMMIT;
DECLARE
    V_NAME EMP.NAME%TYPE;
BEGIN
    SELECT NAME INTO V_NAME FROM EMP;
    DBMS_OUTPUT.PUT_LINE( 'EMP.NAME = ' || V_NAME );
END;
/
```

%ROWTYPE

특정 테이블의 구조 또는 특정 커서의 반환 타입과 동일한 레코드 타입을 명시할 때 사용된다. 참조할 수 있는 대상은 다음과 같다.

- 테이블
- Cursor

- Cursor variable

레코드 타입 변수나 collection 타입 변수는 %ROWTYPE의 대상이 될 수 없다.

%ROWTYPE을 사용하는 예는 다음과 같다.

```
DECLARE
  V_EMP EMP%ROWTYPE;
BEGIN
  SELECT * INTO V_EMP FROM EMP WHERE ID = 1001;
  DBMS_OUTPUT.PUT_LINE( 'Name of ID 1001 Is : ' || V_EMP.NAME );
END;
/
```

Constraint 속성 상속

Attribute 타입으로 선언된 변수들의 constraint 속성은 다음과 같이 참조 대상의 constraint 속성을 상속받는다.

표 18-1 Attribute type의 constraint 상속 유무

Attribute type	참조 대상	NOT NULL	Default 값
%TYPE	Scalar 변수	O	X
	레코드 타입 변수	O	O
	레코드 타입 변수의 특정 필드	O	X
	Collection 타입의 변수 - scalar element	O	X
	Collection 타입의 변수 - 레코드 element	O	O
	Collection 타입 변수의 특정 필드	O	X
	테이블의 특정 column	X	X
%ROWTYPE	테이블	X	X
	Cursor	X	X
	Cursor variable	X	X

18.3 User-defined Record Type

레코드 타입 변수는 서로 다른 타입의 필드 여러 개로 구성된 복합 구조 타입이다. 레코드 타입 변수는 %ROWTYPE 을 사용하여 다른 테이블이나 커서의 타입을 그대로 복사해 오거나 사용자가 특정 용도에 맞는 데이터 구조를 선언하여 생성할 수 있다.

사용자 정의 레코드 타입은 PSM의 선언부에서 TYPE 키워드를 사용하여 다음과 같이 정의할 수 있고 각 필드는 NOT NULL constraint와 기본값을 선택적으로 명시할 수 있다.

```
DECLARE
  TYPE MY_EMP_TYPE IS RECORD
  (
    ID INTEGER := 99999,
    NAME VARCHAR(32) NOT NULL DEFAULT 'anonymous'
  );
  V_EMP MY_EMP_TYPE;
BEGIN
  SELECT ID, NAME INTO V_EMP.ID, V_EMP.NAME FROM EMP;
  DBMS_OUTPUT.PUT_LINE('ID = ' || V_EMP.ID);
  DBMS_OUTPUT.PUT_LINE('NAME = ' || V_EMP.NAME);
END;
/
```

다음은 nested procedure나 nested 함수에서 사용하는 예이다.

```
DECLARE
  TYPE MY_EMP_TYPE IS RECORD
  (
    ID INTEGER := 99999,
    NAME VARCHAR(32) NOT NULL DEFAULT 'anonymous'
  );
  V_EMP MY_EMP_TYPE;
  PROCEDURE SET_EMP( A_EMP IN OUT MY_EMP_TYPE )
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('ID = ' || A_EMP.ID);
    DBMS_OUTPUT.PUT_LINE('NAME = ' || A_EMP.NAME);
    SELECT ID, NAME INTO A_EMP.ID, A_EMP.NAME FROM EMP;
  END;
BEGIN
```

```
SET_EMP( V_EMP );  
DBMS_OUTPUT.PUT_LINE('ID = ' || V_EMP.ID);  
DBMS_OUTPUT.PUT_LINE('NAME = ' || V_EMP.NAME);  
END;  
/
```

사용자 정의 레코드 타입은 일반 지역 변수, nested procedure, 또는 nested 함수의 인자나 반환 타입으로 사용할 수 있지만, schema-level procedure나 schema-level 함수의 인자나 반환 타입으로는 사용할 수 없다.

18.4 User-defined Collection Type

User-defined collection type은 한 개 이상의 데이터를 저장하는 일종의 array 구조이다. GOLDBLOCKS PSM은 collection type 중에 key/ value pair로 저장할 수 있는 associative array 타입을 지원한다.

Associative type을 선언하기 위한 기본 구문은 다음과 같다.

```
TYPE <type_name> IS TABLE OF <element_data_type> INDEX BY <index_key_data_type>
```

- <type_name>은 사용자가 해당 type에 대해 지정하는 이름이다.
- <element_data_type>은 array를 구성하는 value의 data type을 지정한다.
- <index_key_data_type>은 element를 탐색할 index key의 data type을 지정한다.

예를 들어, (번호)에 해당하는 정보가 (이름, 나이) 형식일 경우, 데이터베이스에 다음과 같은 테이블을 생성하고 저장할 수 있다.

```
CREATE TABLE INFO
(
    NO INTEGER,
    NAME VARCHAR(20),
    AGE INTEGER
)
CREATE UNIQUE INDEX IDX_NO ON INFO (NO)
```

- 위와 같은 정보를 associative array로 저장할 경우 다음과 같이 구성된다.
 - Index_key는 번호 (NO)에 해당된다.
 - Element_data는 이름 (NAME), 나이 (AGE)로 구성된다.

실제 PSM 내에서는 이를 저장할 associative array 변수를 다음과 같이 정의할 수 있다.

```
TYPE rec IS RECORD (NAME VARCHAR(20), AGE INTEGER);
TYPE info IS TABLE OF rec INDEX BY INTEGER;
```

자세한 내용은 **COLLECTION Variable Declaration**을 참조한다.

Associative Array

Associative array type 변수는 index 절에 기술된 data type의 key를 가지며 TABLE OF 절에 기술된 element data type의 value를 key/ value 형태로 한 개 이상 저장할 수 있는 PSM 변수이다.

- GOLDDILOCKS PSM의 associative array type의 특징은 다음과 같다.
 - INDEX BY 절에 기술된 data type 형태의 탐색 key를 가지며 자동으로 정렬되어 저장된다.
 - TABLE OF 절에 기술된 element로 구성된다.
 - Collection method라고 불리는 탐색 함수들을 제공한다.
 - 이미 존재하는 index key에 element를 저장할 경우에는 replace 형태로 저장된다.
 - 최대 저장 공간은 사용 가능한 MEMORY_TEMP_TBS_SIZE 크기 이내로 제한된다.

다음은 SQL data type으로 element type을 선언하여 데이터를 삽입하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
  TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(10);
  V1 rec;
BEGIN
  V1('aa') := 'Dog';
  V1('bb') := 'Cat';
  INSERT INTO T1 VALUES ( V1('aa'), V1('bb') );
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1  C2
--- ---
Dog Cat
1 row selected.
```

다음은 record type을 element로 갖는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
  TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
  V1 rec;
```

```

BEGIN
  V1('person1').C1 := 'seoul';
  V1('person1').C2 := '12';
  V1('person2').C1 := 'busan';
  V1('person2').C2 := '24';
  INSERT INTO T1 VALUES V1('person1'), V1('person2');
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1    C2
----- --
seoul 12
busan 24
2 rows selected.

```

- GOLDDILOCKS PSM의 associative array의 index key에 지정할 수 있는 데이터 타입은 다음과 같다.
 - INTEGER
 - LONG
 - CHAR
 - VARCHAR
- GOLDDILOCKS PSM의 associative array의 element type으로 사용할 수 있는 데이터 타입은 다음과 같다.
 - SQL data type
 - %TYPE
 - %ROWTYPE
 - User-defined record type

자세한 내용은 **Built-in Data Types**, **%TYPE**, **%ROWTYPE**, **User-defined Record Type**을 참조한다.

Assign Values to Collection Variables

Collection 변수 간의 assign에는 다음 규칙이 적용된다.

- Element assign에는 <Assign Statement>와 동일한 규칙이 적용된다.
- Collection 변수 전체를 assign하려면 collection 변수의 타입이 동일해야만 한다.
- Collection 변수의 element를 assign 할 경우에는 element data type에 따라 다음과 같이 작동한다.
 - User-defined type element의 assign은 동일한 타입 사이에서만 허용된다.
 - 그 외의 경우에는 element를 구성하는 필드의 data type 간의 run-time 시점에 호환이 가능할 때만 assign 할 수 있다.

다음은 다른 타입을 assign하여 오류가 발생하는 경우의 예이다.

```
DECLARE
  TYPE udr1 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE udr2 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE rec1 IS TABLE OF udr1 INDEX BY VARCHAR(10);
  TYPE rec2 IS TABLE OF udr2 INDEX BY VARCHAR(10);
  V1 rec1;
  V2 rec2;
BEGIN
  V2('person1').F1 := 24;
  V2('person1').F2 := 'seoul korea';
  V1 := V2;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (14:9): ERR-HY000(17007): invalid expression
```

위의 예에서 user-defined type으로 구성된 element의 각 필드 구성은 동일하지만 변수의 type이 다르기때문에 assign할 수 없다.

다음은 동일한 type을 사용하여 정상적으로 assign하는 예이다.

```
DECLARE
  TYPE udr1 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE rec1 IS TABLE OF udr1 INDEX BY VARCHAR(10);
  V1 rec1;
  V2 rec1;
BEGIN
  V2('person1').F1 := 24;
  V2('person1').F2 := 'seoul korea';
  V1 := V2;
END;
/
Anonymous PL block executed.
```

Associative type을 element 단위로 assign하려면 element의 data type이 호환 가능해야 한다.

다음 예를 참조한다.

```
gSQL> CREATE TABLE T1
(
  c1 VARCHAR(20),
```

```

    c2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
TYPE record_org1 IS RECORD (f1 VARCHAR(20), f2 VARCHAR(20));
TYPE rec1 IS TABLE OF t1%rowtype INDEX BY VARCHAR(10);
TYPE rec2 IS TABLE OF record_org1 INDEX BY VARCHAR(10);
v1 record_org1;
v2 rec1;
v3 rec2;
v4 t1%rowtype;
BEGIN

    -- From record_org1 type to %rowtype
    V2('first') := v1;
    -- From record_org1 type to record_org1 type
    V3('first') := v1;

    -- From t1%rowtype to record_org1 type
    V3('second') := V2('first');
END;
/
Anonymous PL block executed.

```

Associative type 변수는 휘발성 메모리 공간에 저장되는데 공간이 부족할 경우 사용자가 접근 가능한 TEMP TABLE SPACE를 확장해야 한다. 다음은 공간 부족으로 인해 오류가 발생하는 예이다.

```

DECLARE
TYPE rec IS TABLE OF t1%rowtype INDEX BY varchar(20);
v1 rec;
BEGIN
    BEGIN
        FOR i IN 1 .. 100000
        LOOP
            v1(i).c1 := i;
            v1(i).c2 := i;
            v1(i).c3 := i;
        END LOOP;
    EXCEPTION WHEN OTHERS THEN
        dbms_output.put_line('error: count=' || v1.count());
        dbms_output.put_line('sqlcode=' || SQLCODE);
    END;
END;

```

```

        dbms_output.put_line('sqlmsg =' || SQLERRM);
    END;
    dbms_output.put_line('v1.count=' || v1.count());
END;
/
error: count=95004
sqlcode=-14015
sqlmsg =[SUNJESOFT][PSM][GOLDILOCKS]there is no extendible datafile in tablespace 'MEM_TEMP_
TBS'
v1.count=95004
Anonymous PL block executed.

```

Collection Method

Collection method는 collection type 변수를 쉽게 operation 할 수 있도록 제공되는 function이나 procedure를 의미한다. Associative array에는 다음과 같은 method가 제공된다.

표 18-2 Collection method

Method	유형	입력 인자	반환 타입	설명
FIRST	Function	X	Index key data type	첫 번째 index key를 반환한다.
LAST	Function	X	Index key data type	마지막 index key를 반환한다.
COUNT	Function	X	INTEGER	Element의 개수를 반환한다.
EXISTS	Function	O	BOOLEAN	Index key의 존재 여부를 반환한다.
PRIOR	Function	O	Index key data type	입력된 index key 이전의 index key를 반환한다.
NEXT	Function	O	Index key data type	입력된 index key 이후 index key를 반환한다.
DELETE	Procedure	O	N/A	입력된 index key에 해당하는 element를 삭제한다.

Collection method는 다음과 같이 사용할 수 있다.

```

gSQL> CREATE TABLE T1
(
    C1 VARCHAR(20),
    C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
    TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
    V1 rec;
BEGIN

```

```

V1('person1').C1 := 'seoul';
V1('person1').C2 := '12';
V1('person2').C1 := 'busan';
V1('person2').C2 := '24';
V1('person3').C1 := 'Daegu';
V1('person3').C2 := '36';

-- First method
DBMS_OUTPUT.PUT_LINE('First Index Key = ' || V1.first() );

-- Last method
DBMS_OUTPUT.PUT_LINE('Last Index Key = ' || V1.last() );

-- Count method
DBMS_OUTPUT.PUT_LINE('Count of element = ' || V1.count() );

-- Prior Method
DBMS_OUTPUT.PUT_LINE('Prior (person1) = ' || V1.prior('person1') ); -- return NULL
DBMS_OUTPUT.PUT_LINE('Prior (person3) = ' || V1.prior('person3') );

-- Next Method
DBMS_OUTPUT.PUT_LINE('Next (person1) = ' || V1.next('person1') );
DBMS_OUTPUT.PUT_LINE('Next (person3) = ' || V1.next('person3') ); -- return NULL

-- Exists Method
DBMS_OUTPUT.PUT_LINE('Exists (person2) = ' || V1.exists('person2') );

-- Delete Method
V1.delete('person2');

-- Exists Method
DBMS_OUTPUT.PUT_LINE('After delete, Exists (person2) = ' || V1.exists('person2') );

END;

/

First Index Key = person1
Last Index Key = person3
Count of element = 3
Prior (person1) =
Prior (person3) = person2
Next (person1) = person2
Next (person3) =
Exists (person2) = TRUE
After delete, Exists (person2) = FALSE
Anonymous PL block executed.

```

한 개의 element를 제거하는 delete procedure가 입력된 인자에 해당하는 index key를 찾지 못할 경우 다음과 같은 오류가 발생한다.

```
gSQL> DECLARE
  TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
  V1 rec;
BEGIN
  V1('person1').C1 := 'seoul';
  V1('person1').C2 := '12';
  -- Call delete procedure
  V1.delete('person2');
END;
/
ERR-HY000(17045): no data found :
  V1.delete('person2');
*
ERROR at line 9:
Anonymous PL block executed.
```


18.5 SYS_REFCURSOR

SYS_REFCURSOR는 cursor variable에 대한 predefined type이며 cursor variable을 선언하는 용도로 사용된다.

다음과 같은 형태로 cursor variable을 선언할 때 사용된다.

```
cursor_variable_name SYS_REFCURSOR;
```

Cursor variable은 다음과 같이 OPEN FOR, FETCH, CLOSE 구문과 함께 사용할 수 있다.

```
DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
  cv2 SYS_REFCURSOR;
BEGIN
  OPEN cv1 FOR SELECT * FROM T1;
  cv2 := cv1;
  FETCH cv2 INTO V1, V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);

END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

- SYS_REFCURSOR로 선언된 변수는 다음과 같이 사용된다.
 - 여러 cursor variable 사이에 서로 assign할 수 있다. Assign의 오른쪽에 위치한 cursor variable의 기존에 이미 열려있는 cursor는 더 이상 사용할 수 없다.
 - Explicit cursor를 cursor variable에 assign 할 수 없다.
 - 다른 data type의 변수를 cursor variable에 assign 할 수 없다.
 - Nested function/ procedure 또는 schema-level function/ procedure의 parameter로 사용할 수 있다.

자세한 내용은 **CURSOR VARIABLES**, **Cursor Variable Declaration**, **OPEN FOR Statement**를 참조한다.

19.

PSM Control Statements

19.1 Assignment

Assignment 연산자 (':=')를 사용하여 오른쪽 expression이 계산된 결과 값을 왼쪽 변수에 삽입한다.

Assignment Target

Assignment 연산자의 왼쪽에는 assignment 대상이 위치하는데 여기에 올 수 있는 item들은 다음과 같다.

- 변수 (IN 타입이 아닌 procedure나 함수의 인자 포함)
- '?' 나 ':V1' 과 같은 바인드 parameter (Anonymous PL block에서만 사용 가능)

Assigning Expression

Assignment 연산자의 오른쪽에는 PSM에서 사용 가능한 expression이 올 수 있다. PSM의 expression에서 사용 가능한 표현식들은 다음과 같다.

- 상수
- GOLDILOCKS SQL에서 제공하는 모든 연산자, built-in 함수 또는 pseudo column들
- IN 또는 IN OUT 타입으로 바인드 된 바인드 parameter (Anonymous PL block에서만 사용 가능)
- PSM 변수
- PSM nested 함수
- Schema-level 함수

다음은 각 유형별 assignment 구문의 예이다.

```
CREATE OR REPLACE FUNCTION ADD_TEN( A1 INTEGER )
RETURN INTEGER
IS
BEGIN
    RETURN A1 + 10;
END;
/
COMMIT;
DECLARE
    FUNCTION ADD_ONE( A1 INTEGER )
    RETURN INTEGER
    IS
```

```

BEGIN
    RETURN A1 + 1;
END;
V_NUM INTEGER;
V_STR VARCHAR(10);
BEGIN
    V_NUM := 10;           ❶ Numeric literal
    V_STR := 'ABC';        ❷ String literal
    :V_PARAM := V_STR || 'DEF';  ❸ PSM variable, SQL operator
    V_NUM := ADD_ONE( 100 ) + ADD_TEN( 1000 );
                                ❹ Nested function, Schema-level function
END;
/

```

다음 표현식들은 PSM의 expression에서 사용할 수 없다.

- 테이블, view나 해당 객체의 column
- 인덱스, 시퀀스, 동의어 (synonym) 등의 SQL 객체
- Subquery
- Nested procedure
- Schema-level procedure

다음은 잘못된 expression을 사용하여 에러가 발생하는 예이다.

```

gSQL> CREATE SEQUENCE SEQ1;
Sequence created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    V1 INTEGER;
BEGIN
    V1 := SEQ1.NEXTVAL;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (4:9): ERR-42000(16074): sequence number not allowed here

```

PSM expression에서 사용할 수 없는 종류의 객체 값을 할당하려면 다음과 같이 SELECT INTO 구문을 사용하여야 한다.

```

gSQL> DECLARE
    V1 INTEGER;

```

```

BEGIN
    SELECT SEQ1.NEXTVAL INTO V1 FROM DUAL;
END;
/
Anonymous PL block executed.

```

Assignment 호환성

Assignment 구문의 expression을 사용할 수 있는 구문이라도 target 타입과 expression의 최종 결과 타입에 따라 성공할 수도 있고 실패할 수도 있다. Target의 타입에 따라 사용 가능한 expression 타입은 다음과 같다.

표 19-1 Assignment 호환성

Target 타입	Expression의 최종 결과 타입
Scalar 타입 - Scalar 타입 변수 - 레코드 타입 변수의 특정 필드 - Key 값을 가지는 collection 변수의 scalar element	Target의 타입으로 conversion 가능한 값을 가진 Scalar 타입 인 경우에만 허용된다.
Attribute 레코드 타입 (%ROWTYPE)	필드의 개수가 같고, 각 필드의 값이 target의 해당 순번의 필드로 conversion 가능한 경우에만 허용된다.
사용자 정의 레코드 타입	정확하게 같은 타입만 허용된다.
Collection 타입 Key 값이 명시되지 않은 collection 타입 변수	정확하게 같은 타입만 허용된다.

내부 구조가 같은 사용자 정의 레코드 타입 변수이더라도 타입 이름이 다르면 다음과 같은 오류가 발생한다.

```

gSQL> DECLARE
    TYPE MY_REC1 IS RECORD ( F1 INTEGER := 1, F2 VARCHAR(10) := 'AAA' );
    TYPE MY_REC2 IS RECORD ( F1 INTEGER := 1, F2 VARCHAR(10) := 'AAA' );
    V1 MY_REC1;
    V2 MY_REC2;
BEGIN
    V2 := V1;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (7:9): ERR-HY000(17007): invalid expression

```

그 외에 target이 IN 속성으로 정의된 인자 변수이거나 IN 속성으로 바인딩 된 바인드 parameter인 경우에도 오류가 발생한다.

```
gSQL> DECLARE
FUNCTION ADD_ONE( A1 IN INTEGER )
RETURN INTEGER
IS
BEGIN
    A1 := A1 + 1;
    RETURN A1;
END;
V1 INTEGER;
BEGIN
    V1 := ADD_ONE( 10 );
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (6:5): ERR-HY000(17024): (A1) cannot be used as assignment target
```

Assignment 호환성 규칙은 함수나 procedure를 호출하는 중에 actual parameter 값을 formal parameter 변수로 할당할 때도 동일하게 적용된다.

19.2 PL Block

PL block은 PSM을 구성하는 가장 기본적인 단위이다. PL block은 다른 PL block 안에 다시 nested 될 수 있으며, PL block 안에서 선언된 item들은 모두 해당 block의 범위 내에서만 (하위 block 포함) 참조할 수 있다.

PL Block의 구성

한 개의 PL block은 다음과 같이 세 부분으로 나누어진다.
자세한 내용은 **Block (BEGIN .. END)**를 참조한다.

```
[DECLARE
  ❶ 선언부 (Declarative Part)]
BEGIN
  ❷ Statements
[EXCEPTION
  ❸ Handlers]
END;
```

선언부 (Declarative Part)

선언부는 PL block 내부에서 사용할 지역 (local) item들을 선언하는 구역이다. 선언할 지역 변수 등의 item이 없을 경우에는 선언부에 대한 정의 없이 executable part로 시작된다.

선언부에서 선언될 수 있는 지역 item들은 다음과 같다.

- 변수 (PSM DataTypes 참조)
- 사용자 정의 타입
- 커서
- Nested 함수 또는 procedure (subprogram)
- 사용자 정의 exception

하나의 PL block 내에서 선언된 모든 item들의 이름은 종류에 관계없이 unique 해야 한다. 예를 들어 변수와 같은 이름의 커서는 선언할 수 없다.

```
gSQL> DECLARE
  C1 INTEGER;
  CURSOR C1 IS SELECT * FROM DUAL;
BEGIN
  OPEN C1;
```



```

    FETCH C1 INTO C1;
    CLOSE C1;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (3:10): ERR-HY000(17027): duplicated identifier
(2) at (5:8): ERR-HY000(17031): mismatch identifier type
(3) at (6:9): ERR-HY000(17031): mismatch identifier type
(4) at (7:9): ERR-HY000(17031): mismatch identifier type

```

단, 하위 PL block에서는 상위 PL block에 선언된 item과 이름이 같은 item을 선언할 수 있다. 이 때 해당 이름은 현재 위치에서 상위 block 방향으로 탐색할 때 가장 먼저 발견되는 item을 가리킨다.

```

gSQL> <<PP>>
DECLARE ❶ Parent scope begin
    V1 VARCHAR(10) := 'AAA';
BEGIN
    <<CC>>
    DECLARE ❷ Child scope begin
        V1 INTEGER := 99;
    BEGIN
        DBMS_OUTPUT.PUT_LINE( 'PP.V1 = ' || PP.V1 );
        DBMS_OUTPUT.PUT_LINE( 'CC.V1 = ' || CC.V1 );
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END; ❸ Child scope end
END; ❹ Parent scope end
/
PP.V1 = AAA
CC.V1 = 99
V1 = 99
Anonymous PL block executed.

```

실행부 (Executable Part)

실행부에서는 선언된 item들을 조작하고, 포함하고 있는 여러 statement들을 수행한다. 실행부에 올 수 있는 statement 종류는 다음과 같다.

- Control statements (**PSM Control Statements** 참조)
- Cursor statements (**PSM Cursor Statements** 참조)
- SQL statements (Static/ dynamic) (**Using SQLs In PSM** 참조)

예외 처리부 (Exception Handling Part)

실행 중 발생하는 여러가지 예외 (exception)들을 처리할 handler들을 정의한다.

실행부의 여러 statement들을 수행하는 도중에 exception이 발생하면 해당 PL block 위치부터 상위 PL block으로 가면서 발생한 exception을 처리할 handler가 등록된 곳이 있는지 탐색한다. 만일 존재할 경우, 해당 handler의 내용을 수행한 후 exception 발생 상태를 초기화한다.

```
DECLARE
    V1 INTEGER;
BEGIN
    SELECT C1 INTO V1 FROM T1 WHERE C1 = 100;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
        DBMS_OUTPUT.PUT_LINE('SQL%FOUND     = ' || SQL%FOUND);
        DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
        DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
```

만일 최상위 PL block까지 탐색한 후에도 적당한 handler를 발견하지 못할 경우, 발생한 exception이 caller에게 반환되며 종료된다

```
gSQL> DECLARE
    v1 INTEGER;
BEGIN
    v1 := 1/ 0;
END;
/
ERR-HY000(17007): invalid expression :
    v1 := 1/ 0;
        *
ERROR at line 4:
ERR-22012(12122): divisor is equal to zero
```

사용자 정의 exception 선언, built-in handler의 종류 및 handler의 설치 등에 대한 자세한 내용은 **Error Handling** 을 참조한다.

19.3 NULL Statement

PSM 내에서 no-op와 같이 아무런 역할도 하지 않는 statement를 명시할 때 사용한다.
자세한 내용은 **NULL Statement**를 참조한다.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  IF V1 < 10 THEN
    V1 := V1 + 1;
  ELSE
    NULL; -- do nothing
  END IF;
END;
/
```

19.4 Testing Conditions

조건 분기 구문들은 주어진 조건의 TRUE/ FALSE에 따라 statement들을 수행한다. GOLDILOCKS PSM은 IF와 CASE의 두 가지 조건 분기 구문을 제공한다.

IF

IF 구문은 주어진 expression의 조건들 중 TRUE인 것들에 해당하는 statement들을 수행한다. Expression 조건들은 IF, ELSIF 다음에 기술할 수 있다. 실행할 때 기술된 순서대로 조건을 검사하여 해당 expression의 조건이 TRUE인 경우, THEN 절 이하의 statement들을 수행한다. 모든 IF 조건이나 ELSIF 조건이 거짓이고 ELSE 절이 기술되었을 경우에는 ELSE 절 이하의 statement들이 수행된다.

IF 구문은 항상 END IF 키워드로 종료된다.

자세한 내용은 **IF Statement**를 참조한다.

```
DECLARE
V1 INTEGER := 1;
BEGIN
  -- IF COND
  IF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
  ELSIF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
  END IF;
  -- ELSIF COND
  IF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
  ELSIF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
  END IF;
  -- ELSE COND
  IF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
  ELSIF V1 < 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
```

```

ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
END IF;
END;
/

```

ELSIF 절이나 ELSE 절은 선택사항이므로 기술될 수도 있고 기술되지 않을 수도 있다.

ELSE 절이 기술되지 않고 IF나 ELSIF의 모든 조건이 true가 아닐 경우에는 어떤 statement도 수행하지 않고 exception 발생없이 다음 statement로 진행된다.

CASE

CASE 문은 IF 문과 같은 여러 statement들이 배열된 것들 중에서 한 개 배열을 선택하여 수행한다. CASE 문은 END CASE 키워드로 종료되며, ELSE 절은 선택사항이므로 기술될 수도 있고 기술되지 않을 수도 있다.

적당한 조건을 가진 WHEN 절을 찾을 수 없고 ELSE 절이 기술되어 있을 경우, ELSE 절 이하의 statement 배열을 수행한다. 적당한 조건이 없고 ELSE 절이 기술되어 있지 않은 경우, CASE_NOT_FOUND exception이 발생한다. 자세한 내용은 **CASE Statement**를 참조한다.

CASE의 WHEN 절들은 기술된 순서대로 evaluate되며, 적당한 WHEN 절을 찾아서 해당 WHEN 절의 statement 배열이 수행되면 그 다음에 기술된 모든 WHEN 절은 무시된다.

CASE 문에는 다음과 같은 두 가지 형태가 있다.

- Simple CASE statement
- Searched CASE statement

Simple CASE Statement

Simple CASE statement는 CASE 키워드 다음에 기술된 selector expression을 사용하여 WHEN 절 이후에 기술된 expression들 중에 selector expression과 같은 값을 가지는 WHEN절의 statement 배열을 수행한 후 종료한다.

```

DECLARE
V1 VARCHAR(1) := '2';
BEGIN
    CASE V1 WHEN '0' THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
            WHEN '1' THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
            WHEN '2' THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
            ELSE DBMS_OUTPUT.PUT_LINE ('Result = Other');
    END CASE;
END;

```

/

Searched CASE Statement

Searched CASE statement에는 selector expression을 가지지 않는 대신 각 WHEN 절마다 boolean 타입으로 evaluate되는 조건 expression이 기술된다. 실행할 때는 각 WHEN 절의 expression을 evaluate하여 TRUE가 되는 첫 WHEN절이 보유한 statement를 배열한 후에 종료한다.

```
DECLARE
V1 VARCHAR(1) := '2';
BEGIN
CASE WHEN V1 = 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
      WHEN V1 = 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
      WHEN V1 = 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
      ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
END CASE;
END;
/
```

19.5 Iterative Control

Iterative control 구문들은 일련의 statement들을 여러 번 수행한다. GOLDBLOCKS PSM에서 제공하는 iterative control 구문의 종류는 다음과 같다.

- Basic loop
- FOR loop
- WHILE loop

Basic Loop

Basic loop 구문은 LOOP와 END LOOP 키워드로 반복 수행할 일련의 statement들을 감싸고 있다. 이 구문은 내부의 statement들을 무한 반복하여 수행하며, sequential control 구문인 EXIT나 GOTO 구문을 이용하여 loop 밖으로 탈출할 수 있다.

자세한 내용은 **Basic LOOP Statement**를 참조한다.

다음은 EXIT WHEN 구문을 사용하여 해당 위치에서 가장 가까운 scope를 가지는 basic loop를 탈출하는 예이다. 자세한 내용은 **EXIT Statement**를 참조한다.

```
gSQL> DECLARE
    V1 INTEGER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
        V1 := V1 + 1;
        EXIT WHEN V1 > 10;  -- Escape Condition
    END LOOP;
END;
/
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
V1 = 6
V1 = 7
V1 = 8
V1 = 9
V1 = 10
```

Anonymous PL block executed.

여러 loop가 중첩되어 있는 상황에서 특정 loop 밖으로 탈출하려면 루프 앞에 label을 명시하고 EXIT 구문에서 해당 label을 target label로 명시하면 된다.

```
gSQL> DECLARE
  V1 INTEGER := 1;
BEGIN
  <<OUTER_LOOP>>
  LOOP
    <<INNER_LOOP>>
    LOOP
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      V1 := V1 + 1;
      EXIT OUTER_LOOP WHEN V1 > 5; -- Escape Condition
    END LOOP INNER_LOOP;
    DBMS_OUTPUT.PUT_LINE( 'END OF INNER LOOP' );
    V1 := V1 + 5;
  END LOOP OUTER_LOOP;
  DBMS_OUTPUT.PUT_LINE( 'END OF OUTER LOOP' );
END;
/
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
END OF OUTER LOOP
Anonymous PL block executed.
```

FOR Loop

FOR loop 구문은 주어진 범위의 정수 개수 만큼 일련의 statement들을 반복 수행한다.

자세한 내용은 **FOR LOOP Statement**를 참조한다.

사용자가 FOR 키워드 뒤에 명시한 이름으로 인덱스 변수를 생성하고, IN 키워드 뒤의 범위 연산자 ('..') 왼쪽에 명시된 시작값을 초기값으로 하여 오른쪽에 명시된 종료값과 같아질 때까지 loop 한 번당 1씩 증가시켜 주어진 일련의 statement들을 수행한다.


```

BEGIN
  FOR I IN 0 .. 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/

```

범위 앞에 REVERSE 키워드를 명시하면 인덱스 변수의 값을 1씩 감소시키면서 loop를 수행한다.

```

BEGIN
  FOR I IN REVERSE 10 .. 1 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/

```

WHILE Loop

WHILE loop 구문은 주어진 조건이 TRUE인 동안 주어진 일련의 statement들을 반복해서 수행한다. 자세한 내용은 **WHILE LOOP Statement**를 참조한다.

```

DECLARE
  V1 integer := 0;
BEGIN
  WHILE V1 < 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
  END LOOP;
END;
/

```

WHILE loop는 body의 statement들을 수행하기 전에 주어진 조건을 검사하기 때문에 시작 시점부터 조건이 거짓이면 body의 statement들이 한 번도 수행되지 않을 수도 있다.

19.6 Sequential Control

Sequential control 구문들은 프로그램 수행 위치를 현재 위치에서 다른 위치로 옮긴다.

GOLDILOCKS PSM에서는 다음과 같은 세 가지 sequential control 구문들을 제공한다

- GOTO
- CONTINUE
- EXIT

GOTO

GOTO 구문은 수행 위치를 주어진 label이 있는 statement로 이동시킨다. Label의 PSM 내 실행 가능한 모든 statement 앞에 명시할 수 있다. GOTO가 이동하는 target label은 GOTO 구문의 이전이나 이후 모두에 존재할 수 있지만 현재 위치에서 visible한 statement인 경우에만 GOTO를 수행할 수 있다.

자세한 내용은 **GOTO Statement**를 참조한다.

Visible한 statement의 조건은 다음과 같다.

- 현재 수행 위치의 앞 뒤에 존재하는 sibling statement들
- 현재 위치의 상위 statement와 그 앞 뒤에 존재하는 sibling statement들

다음은 현재 위치의 sibling statement로 이동하는 예이다.

```
gSQL> BEGIN
  <<PARENT_PREV>>
  DBMS_OUTPUT.PUT_LINE('PARENT PREV');
  <<PARENT>>
  IF 1 > 0 THEN
    <<SIBLING_PREV>>
    DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
    <<CURRENT_POSITION>>
    GOTO SIBLING_NEXT;
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    <<SIBLING_NEXT>>
    DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
  ELSE
    <<NON_SIBLING>>
    DBMS_OUTPUT.PUT_LINE('NON SIBLING');
```

```

END IF;
<<PARENT_NEXT>>
DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
PARENT PREV
SIBLING PREV
SIBLING NEXT
PARENT NEXT
Anonymous PL block executed.

```

다음은 현재 위치의 상위 statement로 이동하는 예이다.

```

gSQL> BEGIN
<<PARENT_PREV>>
DBMS_OUTPUT.PUT_LINE('PARENT PREV');
<<PARENT>>
IF 1 > 0 THEN
<<SIBLING_PREV>>
DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
<<CURRENT_POSITION>>
GOTO PARENT_NEXT;
DBMS_OUTPUT.PUT_LINE('SIBLINGS');
DBMS_OUTPUT.PUT_LINE('SIBLINGS');
<<SIBLING_NEXT>>
DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
ELSE
<<NON_SIBLING>>
DBMS_OUTPUT.PUT_LINE('NON SIBLING');
END IF;
<<PARENT_NEXT>>
DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
PARENT PREV
SIBLING PREV
PARENT NEXT
Anonymous PL block executed.

```

다음은 sibling이 아닌 statement로 이동하려다 오류가 발생하는 예이다.

```

gSQL> BEGIN
  <<PARENT_PREV>>
  DBMS_OUTPUT.PUT_LINE('PARENT PREV');
  <<PARENT>>
  IF 1 > 0 THEN
    <<SIBLING_PREV>>
    DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
    <<CURRENT_POSITION>>
    GOTO NON_SIBLING;
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    <<SIBLING_NEXT>>
    DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
  ELSE
    <<NON_SIBLING>>
    DBMS_OUTPUT.PUT_LINE('NON SIBLING');
  END IF;
  <<PARENT_NEXT>>
  DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (12:10): ERR-HY000(17010): can not find label name

```

마찬가지로 현재 statement의 상위 sibling이 아닌 statement로 이동하려 해도 오류가 발생한다.

GOTO는 PSM 프로그램 로직을 유연하게 만들어 줄 수 있는 좋은 도구이다. 하지만 다른 언어에서와 마찬가지로 너무 많이 사용할 경우 가독성이 떨어져서 유지 보수가 힘들어질 수 있다. 따라서 일반적인 로직들은 FOR나 WHILE 등의 loop 구문을 사용하고, GOTO는 반드시 필요한 경우에만 사용하는 것이 좋다.

CONTINUE

CONTINUE 구문은 FOR나 WHILE 등의 loop 구문 내에서 현재 iteration을 종료하고 다음 iteration을 시작한다. 자세한 내용은 **CONTINUE Statement**를 참조한다.

```

DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP

```

```

    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    IF V1 <= 2 THEN
        DBMS_OUTPUT.PUT_LINE( 'CONTINUE' );
        CONTINUE;
    ELSE
        EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE( 'END-OF-WHILE' );
END LOOP AAA;
END;
/

```

Target label이 기술된 경우에는 해당 label을 가지는 루프의 다음 iteration을 시작하고, 기술되지 않았을 경우에는 가장 가까운 (안쪽의) loop의 다음 iteration을 시작한다.

```

DECLARE
    V1 INTEGER := 1;
BEGIN
    <<AAA>>
    FOR I IN 0..5 LOOP
        <<BBB>>
        WHILE V1 <= 10 LOOP
            DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
            DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
            V1 := V1 + 1;
            IF V1 <= 2 THEN
                CONTINUE BBB;
            ELSE
                EXIT AAA;
            END IF;
        END LOOP BBB;
    END LOOP AAA;
END;
/

```

CONTINUE 구문에는 선택적으로 WHEN 절을 기술할 수 있으며, 이 경우 WHEN 절 다음의 expression이 TRUE일 때만 현재 iteration을 끝내고 다음 iteration을 시작한다.

```

DECLARE
    V1 INTEGER :=0 ;
BEGIN

```

```

LOOP
    V1 := V1 + 1;
    CONTINUE WHEN V1 < 5;
    EXIT;
END LOOP;
END;
/

```

EXIT

EXIT 구문은 현재 loop를 탈출하여 그 다음 statement를 수행한다.
자세한 내용은 **EXIT Statement**를 참조한다.

```

DECLARE
    V1 INTEGER := 1;
BEGIN
    WHILE V1 <= 10 LOOP
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
        IF V1 > 5 THEN
            EXIT;
        END IF;
        V1 := V1 + 1;
    END LOOP;
END;
/

```

Target loop가 기술된 경우, 해당 label을 가진 loop를 탈출한다.

```

DECLARE
    V1 INTEGER := 1;
BEGIN
    <<AAA>>
    FOR I IN 0..5 LOOP
        <<BBB>>
        WHILE V1 <= 10 LOOP
            DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
            IF V1 >= 5 THEN
                EXIT AAA;
            END IF;
            V1 := V1 + 1;
        END LOOP;
    END LOOP;
END;
/

```

```

    END LOOP BBB;
  END LOOP AAA;
END;
/

```

CONTINUE 구문과 마찬가지로 EXIT 구문에도 선택적으로 WHEN 절을 기술할 수 있으며, 이 경우 주어진 조건이 TRUE일 때만 EXIT를 수행한다.

```

DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  FOR I IN 0..5 LOOP
    <<BBB>>
    WHILE V1 <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      EXIT AAA WHEN V1 >= 5;
      V1 := V1 + 1;
    END LOOP BBB;
  END LOOP AAA;
END;
/

```

19.7 Error Handling

PSM에서 발생할 수 있는 error의 유형은 발생 시점에 따라 다음과 같이 분류할 수 있다.

- Errors at compile time
- Errors at run-time

Errors at Compile Time

Compile error는 procedure/ function에 기술된 PSM statement를 validation 하는 과정에서 발생하는 오류를 출력한다.

Syntax 오류의 경우, 다음과 같이 첫 번째 발견된 syntax 오류의 위치만 출력하기 때문에 사용자는 수정 후에 다시 컴파일해서 다른 오류가 없는지 확인해야 한다.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
    V1 = 1;
    V2 = 1;
END;
/
ERR-42000(40000): syntax error:
    V1 = 1;
    .....^
Error at line 4
```

Syntax 오류가 없으면 GOLDBLOCKS가 PSM statement를 compile 하는 동안 validation 하는데 이 때 발생하는 오류들은 다음과 같이 출력된다.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
    V1 := 1;
    V2 := 2;
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
```



```
(1) at (4:3): ERR-HY000(17006): unknown variable or column name (V1)
(2) at (5:3): ERR-HY000(17006): unknown variable or column name (V2)
Procedure created.
```

Compile error들은 GOLDILOCKS 자체의 에러 출력 버퍼가 허용하는 size 이내에서 오류 내역을 출력한다. 따라서, 출력되지 않은 오류가 있을 수 있다. 이 경우, 출력되지 않은 오류가 더 있다는 것을 다음과 같은 형태로 출력한다.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
V1 := 1;
V2 := 2;
V3 := 2;
V4 := 2;
V5 := 2;
V6 := 2;
V7 := 2;
V8 := 2;
V10 := 2;
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
(1) at (4:3): ERR-HY000(17006): unknown variable or column name (V1)
(2) at (5:3): ERR-HY000(17006): unknown variable or column name (V2)
(3) at (6:3): ERR-HY000(17006): unknown variable or column name (V3)
(4) at (7:3): ERR-HY000(17006): unknown variable or column name (V4)
(5) at (8:3): ERR-HY000(17006): unknown variable or column name (V5)
(6) at (9:3): ERR-HY000(17006): unknown variable or column name (V6)
(7) at (10:3): ERR-HY000(17006): unknown variable or column name (V7)
2 more errors...
```

에러 메시지는 다음과 같은 형태로 출력된다.

```
(Index) at (Line_Number : Column_Position): ERR-SQLState( Internal_ErrorCode): Detail_Error_
Message
```

- Index
 - 에러가 발생한 순서를 의미한다.
- Line_Number
 - 에러가 발생한 source의 line 위치를 의미한다.
- Column_Position

- 에러가 발생한 source의 column 위치를 의미한다.
- ERR-SQLState
 - 표준 SQLState를 의미한다.
- Internal_ErrorCode
 - 내부 에러코드이다.
- Detail_Error_Message
 - 에러 메시지의 상세 내역이다.

Errors at Run-time

정상적으로 PSM이 적용된 실행 시점에서도 다양한 이유로 PSM statement에서 오류가 발생할 수 있다. 이 경우 GOLDILOCKS는 exception handling 등을 제공하여 사용자가 오류를 제어할 수 있도록 한다.

다음은 실행 시점에 오류가 발생하는 예이다. GOLDILOCKS PMS 실행 시점에 오류가 발생하면 다음과 같이 에러가 발생한 위치와 함께 PSM 오류와 internal error를 함께 출력한다. 즉, 발생한 오류가 한 개 이상일 수 있으므로 사용자가 ODBC와 같은 개발 과정에서 정확한 오류 메시지를 확인하려면 SQLGetDiagRec와 같은 함수를 통해 모든 에러를 데이터베이스로부터 읽어들이어야 한다.

```
DECLARE
  V1 INTEGER;
BEGIN
  V1 := 1 / 0;
END;
/
ERR-HY000(17007): invalid expression :
  V1 := 1 / 0;
      *
ERROR at line 4:
ERR-22012(12122): divisor is equal to zero
```

PSM은 실행 시점에 오류가 발생하면 즉시 중단되고 사용자에게 오류를 알린다. 그러나 사용자가 이러한 오류를 직접 제어하고 프로그램을 계속 실행하고자 할 경우에는 exception handling을 해야 한다.

다음은 오류가 발생하는 위치를 exception handling 하여 중단없이 사용자 프로그램이 수행되도록 정의한 예이다.

```
DECLARE
  V1 INTEGER;
BEGIN
  BEGIN
    V1 := 1 / 0;
  EXCEPTION WHEN OTHERS
```

```

        THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
END;

DBMS_OUTPUT.PUT_LINE('next code');
END;
/
SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero
next code
Anonymous PL block executed.

```

자세한 내용은 EXCEPTION Handling, SQLCODE Function, SQLERRM Function을 참조한다.

Cursor Attributes

Cursor attributes는 PSM을 처리하는 과정에서 데이터베이스 내부나 사용자가 명시한 cursor의 상태를 알 수 있도록 제공되는 속성 (변수)을 의미한다.

- Cursor는 다음과 같이 분류된다.
 - Implicit cursor
 - 데이터베이스 내부에서 필요할 때 OPEN/ FETCH/ CLOSE 되는 cursor
 - Explicit cursor
 - 사용자가 명시적으로 declaration/ definition한 후에 OPEN/ FETCH/ CLOSE 하는 cursor

Implicit Cursor Attributes

PSM에서 implicit cursor attributes는 직전에 수행한 SQL 문의 처리 상태를 알기 위해 사용된다. Attributes의 각 값들은 조건식과 같은 expression에 사용할 수 있기 때문에 사용자가 해당 값을 통해 프로그램의 로직을 분기하는 등의 제어가 가능하다.

각 속성에 대한 자세한 내용은 다음 표를 참조한다.

표 19-2 Implicit cursor attributes

Attribute name	반환 타입	설명
ISOPEN	BOOLEAN	내부적으로 close 되므로 항상 FALSE 이다.
FOUND	BOOLEAN	직전 statement에 의해 데이터가 반환된 경우에는 TRUE이고, 그렇지 않을 경우에는 FALSE 이다.
NOTFOUND	BOOLEAN	FOUND와 반대되는 값을 갖는다.
ROWCOUNT	INTEGER	직전 statement에 의해 영향 받은 row의 개수이다.

사용하는 구문의 형태는 다음과 같다.

```
SQL%Attribute_name
```

```
Attribute_name := ISOPEN | FOUND | NOTFOUND | ROWCOUNT
```

다음 예와 같이 각 속성을 확인할 수 있다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> BEGIN
    UPDATE T1 SET C2 = C2 + 1;

    DBMS_OUTPUT.PUT_LINE('ISOPEN   = ' || SQL%ISOPEN );
    DBMS_OUTPUT.PUT_LINE('FOUND    = ' || SQL%FOUND  );
    DBMS_OUTPUT.PUT_LINE('NOTFOUND = ' || SQL%NOTFOUND );
    DBMS_OUTPUT.PUT_LINE('ROWCOUNT = ' || SQL%ROWCOUNT );
END;
/
ISOPEN   = FALSE
FOUND    = TRUE
NOTFOUND = FALSE
ROWCOUNT = 2
Anonymous PL block executed.
```

Explicit Cursor Attributes

Explicit cursor attributes는 explicit cursor의 상태를 조회하기 위해 사용된다.

각 속성에 대한 자세한 내용은 다음 표를 참조한다.

표 19-3 Cursor attributes

속성	반환 타입	설명
%ISOPEN	BOOLEAN	Cursor가 정상적으로 열려진 경우에만 TRUE이고 그 외에는 FALSE이다.
%FOUND	BOOLEAN	FETCH 이전에는 NULL이고 FETCH가 정상적으로 수행된 경우에는 TRUE이며 데이

속성	반환 타입	설명
		터가 없는 경우에는 FALSE이다. CLOSE 이후에는 NULL이다.
%NOTFOUND	BOOLEAN	%FOUND와 반대되는 값을 갖는다.
%ROWCOUNT	INTEGER	OPEN 이전에는 NULL이고 정상적으로 OPEN한 경우 0이며 FETCH가 성공할 때마다 1씩 증가한다.

PSM 내에서는 다음과 같이 사용한다.

```
Cursor_Name % Attribute_name
Attribute_name := ISOPEN
                | FOUND
                | NOTFOUND
                | ROWCOUNT
```

다음은 explicit cursor attribute를 사용하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  CURSOR C1 IS SELECT * FROM T1;
  v1 c1%ROWTYPE;
BEGIN
```

- ISOPEN을 통해 cursor가 열려 있는 상태인지 확인한다.

```
IF C1%ISOPEN = FALSE
  THEN
    OPEN C1;
  END IF;
LOOP
  FETCH C1 INTO v1;
```

- NOTFOUND을 통해 더 이상 데이터가 없는지 확인한다.

```
EXIT WHEN C1%NOTFOUND;
```

- ROWCOUNT를 통해 fetch 된 row의 개수를 확인한다.

```
DBMS_OUTPUT.PUT_LINE('COUNT = ' || C1%ROWCOUNT );
END LOOP;
CLOSE C1;
```

```
END;
```

```
/
```

```
COUNT = 1
```

```
COUNT = 2
```

```
Anonymous PL block executed.
```

EXCEPTION Handling

PSM 내의 exception은 사용자가 PSM을 실행하는 과정에서 발생하는 다양한 오류를 정의한 것이다. PSM 실행 과정에서 오류가 발생하면 즉시 중단하고 사용자에게 오류를 반환한다. EXCEPTION handling은 사용자가 EXCEPTION 상황을 제어하고 중단없이 프로그램을 진행하도록 하기 위해 제공하는 기능이다.

Exception Handler

다음과 같이 PSM BLOCK에 정의된 exception handler를 통해 여러 상황을 제어할 수 있다.

```
DECLARE
  V1 INTEGER;
BEGIN
  BEGIN
    V1 := 1 / 0;
```

- Exception handler

```
EXCEPTION WHEN OTHERS
  THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);

END;

DBMS_OUTPUT.PUT_LINE('next code');
END;
/
```

```

SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero
next code

```

위의 예에서 실행 시점에 divide by zero에 의해 프로그램이 중단된다. 그러나 exception handler에 의해 예러가 발생한 경우, 해당 BLOCK만 중단하고 다음 위치부터 계속 수행한다.

Exception handler는 다음과 같이 기술한다.

```

BEGIN
EXCEPTION WHEN exception_name, [exception_name, ...] THEN pl_statements;
           [WHEN exception_name, [exception_name, ...] THEN pl_statements;
END

```

- Exception handler의 특징은 다음과 같다.
 - 한 개의 exception handler를 BLOCK에 기술한다.
 - Exception handler 내에는 한 개 이상의 Exception_Name을 지정하고 해당 동작을 정의할 수 있다.
 - Exception handler 내의 Exception_name은 중복되면 안된다.
 - Exception handler가 기술된 PL BLOCK SCOPE 내에서 발생한 오류에 대해서만 동작한다.
 - 모든 예외 상황을 가리키는 OTHERS는 유일해야 하며 가장 마지막에 기술되어야 한다.
 - Exception handler에서 정상적으로 완료되면 이전에 발생한 오류는 clear 된다. 필요한 경우 사용자가 별도의 PSM 변수에 저장해야 한다.

Exception은 GOLDILOCKS에서 다음 표와 같은 특성을 갖는다.

표 19-4 EXCEPTION 유형

유형	Definer	Error code	Name	Raise implicitly	Raise explicitly
Predefined	Database	O	O	O	Optionally
User-defined	User	사용자가 지정	사용자가 지정	X	O

노트

GOLDILOCKS에서 제공하는 predefined exception이나 user-defined exception에 대한 자세한 내용은 **Exception Declaration**을 참조한다.

Exception 전파

Exception이 발생했을 때 현재 BLOCK 내에 적절한 exception handler가 없으면 exception이 처리될 때까지 상위 BLOCK의 exception handler로 전파된다.

다음과 같은 코드의 LINE_1에서 오류가 난 경우 exception handler의 동작 여부에 따라 수행 결과가 달라진다.

```

BEGIN
    ...
    BEGIN
        LINE_1
        EXCEPTION HANDLER_1
    END;
    LINE_2
    EXCEPTION HANDLER_2
END;
/

```

- EXCEPTION_HANDLER_1에서 처리된 경우
 - Handler 동작을 정상적으로 완료하면 LINE_2부터 수행한다.
- EXCEPTION_HANDLER_1에 적절한 handler가 없는 경우
 - LINE_2는 수행할 수 없으며 현재 발생한 EXCEPTION은 EXCEPTION_HANDLER_2로 전달된다.
 - EXCEPTION_HANDLER_2에 적절한 handler가 존재하는 경우 handler 동작을 정상적으로 완료하면 프로그램이 종료된다.
 - EXCEPTION_HANDLER_2에 handler가 없는 경우 프로그램 진행이 중단되며 현재 발생한 오류 상황을 사용자에게 전달한다.

주의

Exception을 처리하던 exception handler에 오류가 발생하면 그 오류 코드가 상위 BLOCK으로 전파된다.

```

DECLARE
    V1 INTEGER;
    E1 EXCEPTION;
    E2 EXCEPTION;
BEGIN
    BEGIN
        RAISE E1;
    EXCEPTION WHEN E1 THEN V1 := 1 / 0; ❶ 새로운 오류가 발생한다.
    END;
    EXCEPTION WHEN E1
        THEN DBMS_OUTPUT.PUT_LINE('User Exception');
        WHEN ZERO_DIVIDE
        THEN DBMS_OUTPUT.PUT_LINE('Zero divide');
END;
/

```


Zero divide

Anonymous PL block executed.

User-defined Exception

User-defined exception은 predefined 외에 사용자가 exception name과 error-code를 설정하여 사용하는 exception이다. User-defined exception은 implicate하게 발생할 수 없으며 사용자가 명시적으로 RAISE statement를 사용하여 발생시켜야 한다.

다음은 RAISE statement를 이용하여 exception을 발생시키는 예이다.

```
DECLARE
  V1  INTEGER;
  high EXCEPTION;
  low  EXCEPTION;
BEGIN
  BEGIN
    V1 := 10;
    IF V1 > 10
    THEN
      RAISE high;
    ELSE
      RAISE low;
    END IF;
  EXCEPTION WHEN high
    THEN DBMS_OUTPUT.PUT_LINE('High');
    WHEN low
    THEN DBMS_OUTPUT.PUT_LINE('Low');
  END;
END;
/
Low
```

- User-defined exception은 다음 두 가지 경우로 나뉘어 처리된다.
 - Error code를 설정한 경우
 - Error code를 설정하지 않은 경우

사용자가 에러 코드를 설정한 경우에는 다음과 같이 동작한다.

```
DECLARE
  V1 INTEGER;
  E1 EXCEPTION;
```

```

PRAGMA EXCEPTION_INIT( E1, -12122);
BEGIN
    BEGIN
        V1 := 1 / 0;
    EXCEPTION WHEN E1
        THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);

    END;
END;
/
SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero

```

ZERO_DIVIDE predefined exception이 존재하지만 사용자가 오류 코드를 E1 exception으로 설정하여 handler를 설정한다. 위와 같이 서로 다른 오류 코드를 가진 벤더 간의 프로그램 호환성을 위해 사용자가 직접 exception의 오류 코드를 재설정한 exception handler를 사용할 수 있다.

Error code가 설정되지 않은 user-defined exception은 exception handler에서 다음과 같은 오류코드와 메시지로 설정된다.

```

DECLARE
    V1 INTEGER;
    E1 EXCEPTION;
BEGIN
    BEGIN
        RAISE E1;
    EXCEPTION WHEN E1
        THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);

    END;
END;
/
SQLCODE = 1
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]User-Defined Exception
Anonymous PL block executed.

```

User-defined exception의 전파 과정은 predefined exception의 전파 과정과 동일하다. 단, user-defined exception에 error code가 설정되지 않은 경우, 상위 BLOCK으로 전파되는 시점에 unhandled user exception 오류로 전달된다.

```

DECLARE
    V1 INTEGER;

```

```

E1 EXCEPTION;
E2 EXCEPTION;
BEGIN
  BEGIN
    RAISE E1;
  EXCEPTION WHEN E2
    THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
    DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;
END;
/
ERR-HY000(17017): Unhandled user exception :
    RAISE E1;
    *
ERROR at line 7:

```

Unhandled user exception은 exception handler 내에 user-defined exception을 기술하거나 predefined exception인 OTHERS를 사용하여 catch할 수 있다.

PRAGMA EXCEPTION_INIT

User-defined exception을 사용할 때 사용자가 특정 DBMS 에러 코드를 설정하기 위해 사용한다. DBMS 오류 코드는 벤더마다 다르기 때문에 PSM code의 exception handler 호환성을 위해 사용할 수도 있다.

그 문법은 다음과 같다.

```

<PRAGMA EXCEPTION_INIT> ::=
    PRAGMA EXCEPTION_INIT ( exception_name, internal_errorcode )

```

- 다음과 같은 제약이 있다.
 - exception_name은 미리 선언되어 있어야 한다.
 - exception_name은 predefined exception을 사용할 수 없다.
 - internal_errorcode는 GOLDILOCKS의 internal error code만 가능하다.
 - SUCCESS를 의미하는 0 값을 설정할 수 없다.

다음은 NOT NULL constraints가 적용된 PSM 변수인 V1에 NULL을 assign할 경우의 exception을 user-defined exception으로 handling 하는 예이다.

```

DECLARE
  V1 VARCHAR(20) NOT NULL := 10;
  USER_EXCEPT EXCEPTION;

```

- PSM NOT NULL constraint 오류를 사용자 정의 exception으로 선언한다.

```
PRAGMA EXCEPTION_INIT( USER_EXCEPT, -17009 );
BEGIN
```

- Exception 상황을 발생시킨다.

```
V1 := NULL;
    EXCEPTION WHEN USER_EXCEPT THEN DBMS_OUTPUT.PUT_LINE('Check Value');
END;
/
Check Value
Anonymous PL block executed.
```

GOLDILOCKS의 internal error-code가 아닌 값을 설정할 경우 다음과 같은 오류가 발생한다.

```
DECLARE
    E1 EXCEPTION;
    PRAGMA EXCEPTION_INIT( E1, 99999);
BEGIN
    NULL;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (3:30): ERR-HY000(17021): Invalid error number for PRAGMA EXCEPTION_INIT
```

DBMS_STANDARD.RAISE_APPLICATION_ERROR

명시적으로 사용자 exception을 정의하지 않고 임의로 사용자 에러 코드와 메시지를 통해 간단하게 exception을 발생시키고 싶을 때 사용한다. DBMS_STANDARD 모듈에 속하며, 이 모듈의 루틴들은 해당 모듈명을 생략해도 호출할 수 있다.

다음과 같은 세 가지 인자가 있다.

표 19-5 RAISE_APPLICATION_ERROR

인자	설명
error_code IN NATIVE_INTEGER	임의로 발생시킬 에러 코드 값이다. -20000 ~ -20999 범위의 값만 사용할 수 있다.
error_message IN VARCHAR(4000)	에러가 발생할 때 SQLERRM에 저장될 메시지이다.
stack_flag IN BOOLEAN := F	기존 에러들 위에 쌓을 것인지 (TRUE), 기존 에러를 모두 대체할 것인지 (FALSE)에 관한 flag

인자	설명
ALSE	이다. 기본값은 FALSE이다.

다음은 간단한 사용 예이다.

```

CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
    RAISE_APPLICATION_ERROR (-20000, 'USER DEFINED ERROR TEST');
END;
/
Procedure created.
BEGIN
    PROC1;
EXCEPTION WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE( 'SQLCODE : ' || SQLCODE || ' SQLERRM : ' || SQLERRM );
END;
/
SQLCODE : -20000 SQLERRM : [SUNJESOFT][PSM][GOLDILOCKS]USER DEFINED ERROR TEST
Anonymous PL block executed.

```


20.

PSM Cursor Statements

20.1 Declaration

PSM statement를 통해 조회할 경우 데이터베이스로부터 두 개 이상의 결과를 반환 받을 수 없다. 따라서, 두 개 이상의 결과를 조회하려면 cursor를 선언하고 정의하여 사용해야 한다.

- Cursor는 다음과 같이 분류된다.
 - Explicit cursor
 - Cursor declaration/ definition을 통해 사용자가 명시적으로 선언한다.
 - Implicit cursor
 - PSM statement를 통해 데이터베이스 내부적으로 처리되는 cursor로써 사용자가 정의하거나 선언할 수 없다.

Explicit cursor를 사용하기 전에 PSM declare section에서 cursor를 선언하고 (declaration) 정의해야(definition) 한다. (이후의 설명에서 cursor는 explicit cursor를 의미한다.)

- Cursor 선언 (declaration)은 다음 항목들을 정의한다.
 - Cursor 이름
 - Cursor가 갖는 parameter list
 - Cursor의 return type
- Cursor 정의 (definition)는 다음과 같은 항목을 정의한다.
 - Cursor의 이름
 - Cursor가 갖는 parameter list
 - Cursor의 return type
 - Cursor가 수행할 SELECT 및 SELECT .. FOR UPDATE 구문
- Cursor의 선언과 정의에서 사용되는 각 항목은 다음과 같은 특성을 갖는다.
 - Cursor 이름
 - Cursor가 선언된 SCOPE 내에서 유일해야 한다.
 - 이름은 128 byte 보다 작아야 한다.
 - Parameter list
 - Cursor가 실행할 SQL에 PSM 변수나 값을 binding 해야 할 경우에 정의한다.
 - Return type
 - Cursor가 가진 SQL 문을 validation한 결과 집합의 형태를 엄격하게 정의하고자 할 때 사용한다.
 - 만일 cursor가 가진 SQL 문의 결과 집합에 대한 정의가 return type의 정의와 다를 경우 오류가 발생한다.

노트

Cursor 정의는 이전에 선언된 cursor와 동일한 형태 (name, parameter list, return type)이어야 한다.

Cursor의 declaration/ definition에 대해서는 다음을 참조한다.

```
DECLARE
```

- Declaration

```
CURSOR C1 RETURN T1%ROWTYPE;
```

- Declaration & definition

```
CURSOR C2 IS SELECT * FROM T1;
```

- Definition for C1

```
CURSOR C1 RETURN T1%ROWTYPE IS SELECT * FROM T1;
BEGIN
    NULL;
END;
/
```

만일, 동일한 cursor_name을 갖는 Cursor RETURN TYPE의 선언과 정의가 다를 경우, 다음과 같은 오류가 발생할 수 있다.

```
gSQL> CREATE TABLE T1
(
    C1 VARCHAR(20),
    C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
    TYPE rec IS RECORD (C1 VARCHAR(20), C2 VARCHAR(20));
```

다음 두 개의 cursor return type의 형태는 같지만 다른 type을 사용하는 것으로 인식되기 때문에 동일한 cursor name 사용에 따른 오류가 발생한다.

```
CURSOR C1 RETURN rec;
    CURSOR C1 RETURN T1%ROWTYPE IS SELECT * FROM T1;
BEGIN
    NULL;
END;
/
ERR-HY000(17032): PSM compilation error :
```

(1) at (4:12): ERR-HY000(17033): duplicated cursor name

Parameter list 항목의 선언과 정의가 다른 경우에도 다음과 같은 오류가 발생한다.

DECLARE

- Parameter의 선언과 정의가 다르다. (Parameter name)

```
CURSOR C1 (A1 INTEGER, A2 INTEGER DEFAULT 10) RETURN T1%ROWTYPE;
      CURSOR C1 (A1 INTEGER, A3 INTEGER DEFAULT 10) RETURN T1%ROWTYPE IS
          SELECT * FROM T1;
BEGIN
    NULL;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (3:12): ERR-HY000(17033): duplicated cursor name
```

Cursor에 정의된 SQL 문을 run-time 시점에 PSM 변수 등을 참조하여 수행해야 할 경우, cursor의 parameter로 정의할 수 있다.

```
DECLARE
    cursor c1 (a1 varchar ) return t1%rowtype;
    cursor c2 (a1 varchar(20) ) return t1%rowtype;
BEGIN
    NULL;
END;
/
```

DEFAULT 절을 정의한 경우 cursor를 실행하는 시점에 parameter 정의를 생략할 수 있다.

다음은 parameter를 모두 입력하여 수행하는 경우와 DEFAULT 절에 parameter 정의를 생략하여 수행하는 경우의 예이다.

```
gSQL> CREATE TABLE T1
(
    C1 VARCHAR(20),
    C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
```

1 row created.

```
gSQL> DECLARE
    V1 VARCHAR(20);
    V2 VARCHAR(20);
    CURSOR C1 (A1 INTEGER, A2 INTEGER DEFAULT 10)
        IS SELECT * FROM T1 WHERE C2 >= A1 AND C2 <= A2;
BEGIN
```

- Parameter를 모두 입력하여 수행한다.

```
OPEN C1 (10, 50);
    FETCH C1 INTO V1, V2;
    DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);
    CLOSE C1;
```

- (A2) default parameter를 생략하고 수행한다.

```
OPEN C1 (10);
    FETCH C1 INTO V1, V2;
    IF C1%NOTFOUND = TRUE
    THEN
        DBMS_OUTPUT.PUT_LINE('NO DATA');
    END IF;
    CLOSE C1;
END;
/
V1 = Seoul , V2 = 24
NO DATA
Anonymous PL block executed.
```

20.2 OPEN

Explicit cursor를 declaration/ definition한 후에는 OPEN statement를 통해 cursor의 select statement를 수행할 수 있다. 만일, SELECT .. FOR UPDATE 문인 경우에는 결과 집합의 row를 lock 한다.

Cursor가 정상적으로 OPEN 된 CURSOR를 다시 OPEN하면 오류가 발생한다.

```
DECLARE
  V1 VARCHAR(20);
  V2 VARCHAR(20);
  CURSOR C1 IS SELECT * FROM T1;
BEGIN

  OPEN C1;
  OPEN C1;
END;
/
ERR-HY000(17037): cursor is already open :
  OPEN C1;
  *
ERROR at line 9:
```

Explicit cursor가 parameter를 가질 경우, 다음과 같이 사용할 수 있다.

```
DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cursor c1 (a1 varchar(20) ) return t1%rowtype;
  cursor c1 (a1 varchar(20) ) return t1%rowtype
    is select * from t1 where c1 = a1;
BEGIN
  OPEN c1( 'Seoul' );
  FETCH c1 INTO V1, V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);

END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

자세한 내용은 **OPEN Statement**를 참조한다.

20.3 FETCH

Explicit cursor를 정상적으로 OPEN 한 경우, FETCH statement를 통해 결과를 PSM 변수로 저장할 수 있다.

- 결과는 다음과 같은 PSM 변수에 저장할 수 있다.
 - SQL data type 변수
 - Single row를 저장할 수 있는 record type 변수
 - Single row를 저장할 수 있는 key가 명시된 associative array type 변수

다음은 SQL data type의 변수를 이용하거나 %ROWTYPE을 통해 fetch 하는 예이다.

```
DECLARE
V1 VARCHAR(20);
V2 VARCHAR(20);
V3 T1%ROWTYPE;
CURSOR C1 IS SELECT * FROM T1;
BEGIN
OPEN C1;
```

- List of SQL type variables

```
FETCH C1 INTO V1, V2;
DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);
```

- Single record variables

```
FETCH C1 INTO V3;
DBMS_OUTPUT.PUT_LINE('V3.C1 = ' || V3.C1 || ' , V3.C2 = ' || V3.C2);
END;
/
V1 = Seoul , V2 = 24
V3.C1 = Pusan , V3.C2 = 44
```

FETCH에서 사용되는 record type 변수는 단독으로 쓰여야 하며 다른 유형의 변수와 함께 사용할 경우 다음과 같이 오류가 발생한다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
```

Table created.

```
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
```

1 row created.

```
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
```

1 row created.

```
gSQL> DECLARE
```

```
    V1 VARCHAR(20);
```

```
    V2 VARCHAR(20);
```

```
    V3 T1%ROWTYPE;
```

```
    CURSOR C1 (A1 INTEGER, A2 INTEGER DEFAULT 10)
```

```
        IS SELECT * FROM T1 WHERE C2 >= A1 AND C2 <= A2;
```

```
BEGIN
```

```
    OPEN C1 (10, 50);
```

V3는 record type이고 scalar type return 결과를 record type에 저장할 수 없으므로 오류가 발생한다.

```
    FETCH C1 INTO V1, V3;
```

```
    CLOSE C1;
```

```
END;
```

```
/
```

```
ERR-HY000(17032): PSM compilation error :
```

```
(1) at (10:23): ERR-HY000(17007): invalid expression
```

자세한 내용은 **FETCH Statement**를 참조한다.

20.4 CLOSE

정상적으로 OPEN 된 cursor를 close 한다. Close 된 cursor로부터는 결과나 cursor-attributes를 얻을 수 없다. CLOSE된 cursor에 접근하여 fetch등을 수행할 경우 다음과 같이 INVALID_CURSOR exception이 발생한다.

```
DECLARE
  V1 VARCHAR(20);
  V2 VARCHAR(20);
  CURSOR C1 IS SELECT * FROM T1;
BEGIN

  OPEN C1;
  CLOSE C1;
  BEGIN
    FETCH C1 INTO V1, V2;
  EXCEPTION WHEN INVALID_CURSOR
    THEN DBMS_OUTPUT.PUT_LINE('invalid cursor exception');
  END;
END;
/
invalid cursor exception
Anonymous PL block executed.
```

자세한 내용은 **CLOSE Statement**를 참조한다.

20.5 EXPLICIT CURSOR ATTRIBUTES

Explicit cursor attributes는 explicit cursor의 현재 상태 정보를 가지고 있다. 이 속성들은 expression과 조건식 모두에 사용할 수 있다.

Explicit cursor attribute는 다음 구문과 같이 사용되며 각 속성에 대한 자세한 정보는 다음 표를 참조한다.

```
Cursor_Name % Attribute_name
Attribute_name := ISOPEN
                | FOUND
                | NOTFOUND
                | ROWCOUNT
```

표 20-1 Cursor attributes

속성	리턴 타입	설명
%ISOPEN	BOOLEAN	Cursor가 정상적으로 열린 경우에만 TRUE이다. 그 외에는 FALSE이다.
%FOUND	BOOLEAN	FETCH 이전에는 NULL, FETCH가 정상적으로 수행된 경우 TRUE이고, 데이터가 없는 경우에는 FALSE이며, CLOSE 이후에는 NULL이다.
%NOTFOUND	BOOLEAN	%FOUND와 반대되는 값을 갖는다.
%ROWCOUNT	INTEGER	OPEN 이전까지는 NULL이고, 정상적으로 OPEN 한 경우 0이며 FETCH가 성공할 때 마다 1씩 증가한다.

다음은 각 cursor attribute에 대하여 단계별로 값이 어떻게 변화하는지 보여주는 예이다.

```
DECLARE
    V1 VARCHAR(20);
    V2 VARCHAR(20);
    CURSOR C1 IS SELECT * FROM T1;
BEGIN

    DBMS_OUTPUT.PUT_LINE('<BEFORE OPEN>');
    DBMS_OUTPUT.PUT_LINE('%ISOPEN   = [' || C1%ISOPEN   || ']);
    DBMS_OUTPUT.PUT_LINE('%FOUND    = [' || C1%FOUND    || ']);
    DBMS_OUTPUT.PUT_LINE('%NOTFOUND = [' || C1%NOTFOUND || ']);
    DBMS_OUTPUT.PUT_LINE('%ROWCOUNT = [' || C1%ROWCOUNT || ']);
    OPEN C1;
    DBMS_OUTPUT.PUT_LINE('<AFTER OPEN>');
    DBMS_OUTPUT.PUT_LINE('%ISOPEN   = [' || C1%ISOPEN   || ']);
    DBMS_OUTPUT.PUT_LINE('%FOUND    = [' || C1%FOUND    || ']);
    DBMS_OUTPUT.PUT_LINE('%NOTFOUND = [' || C1%NOTFOUND || ']);
```



```

DBMS_OUTPUT.PUT_LINE('%ROWCOUNT = [' || C1%ROWCOUNT || ']);
FETCH C1 INTO V1, V2;
DBMS_OUTPUT.PUT_LINE('<AFTER FETCH>');
DBMS_OUTPUT.PUT_LINE('%ISOPEN   = [' || C1%ISOPEN   || ']);
DBMS_OUTPUT.PUT_LINE('%FOUND     = [' || C1%FOUND     || ']);
DBMS_OUTPUT.PUT_LINE('%NOTFOUND  = [' || C1%NOTFOUND  || ']);
DBMS_OUTPUT.PUT_LINE('%ROWCOUNT = [' || C1%ROWCOUNT || ']);
CLOSE C1;
DBMS_OUTPUT.PUT_LINE('<AFTER CLOSE>>');
DBMS_OUTPUT.PUT_LINE('%ISOPEN   = [' || C1%ISOPEN   || ']);
DBMS_OUTPUT.PUT_LINE('%FOUND     = [' || C1%FOUND     || ']);
DBMS_OUTPUT.PUT_LINE('%NOTFOUND  = [' || C1%NOTFOUND  || ']);
DBMS_OUTPUT.PUT_LINE('%ROWCOUNT = [' || C1%ROWCOUNT || ']);
END;
/
<BEFORE OPEN>
%ISOPEN   = [FALSE]
%FOUND     = []
%NOTFOUND  = []
%ROWCOUNT = []
<AFTER OPEN>
%ISOPEN   = [TRUE]
%FOUND     = []
%NOTFOUND  = []
%ROWCOUNT = [0]
<AFTER FETCH>
%ISOPEN   = [TRUE]
%FOUND     = [TRUE]
%NOTFOUND  = [FALSE]
%ROWCOUNT = [1]
<AFTER CLOSE>>
%ISOPEN   = [FALSE]
%FOUND     = []
%NOTFOUND  = []
%ROWCOUNT = []
Anonymous PL block executed.

```

20.6 IMPLICIT_CURSOR_ATTRIBUTES

Implicit cursor는 PSM statement를 통해 데이터베이스 내부적으로 처리된 cursor이며 explicit cursor가 가진 attributes와 이름이 같은 attributes를 가진다.

자세한 내용은 다음 표를 참조한다.

표 20-2 Implicit cursor attributes

Attribute name	반환 타입	설명
ISOPEN	BOOLEAN	내부적으로 close되므로 항상 FALSE이다.
FOUND	BOOLEAN	직전 statement에 의해 데이터가 반환된 경우에는 TRUE이고, 그렇지 않을 경우에는 FALSE이다.
NOTFOUND	BOOLEAN	FOUND와 반대되는 값을 갖는다.
ROWCOUNT	INTEGER	직전 statement에 의해 영향 받은 row의 개수이다.

다음은 implicit cursor attributes의 예이다.

```

gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
V1 INTEGER;
BEGIN
  SELECT COUNT(*) INTO V1 FROM T1;
  DBMS_OUTPUT.PUT_LINE('COUNT RET    = ' || V1);
  DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('SQL%FOUND    = ' || SQL%FOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
COUNT RET    = 2
SQL%ISOPEN    = FALSE

```

```
SQL%FOUND      = TRUE
```

```
SQL%NOTFOUND   = FALSE
```

```
SQL%ROWCOUNT  = 1
```

```
Anonymous PL block executed.
```

20.7 CURSOR VARIABLES

Cursor variable은 explicit cursor와 유사한 개념이다. 하지만 explicit cursor가 한 개의 statement에 종속적인 반면, cursor variable은 한 개 이상의 cursor를 가리키는 pointer와 같은 역할을 한다. 즉, 한 개의 cursor variable을 이용하여 N 개의 SQL syntax를 변경해가며 cursor를 사용할 수 있다.

- Explicit cursor와의 차이점은 다음과 같다.
 - 한 개 이상의 SELECT 질의를 변경하며 cursor로 처리할 수 있다.
 - Cursor variable 간에 assign 할 수 있다.
 - Expression 내에서 사용할 수 있다.
 - Subprogram과 schema-level procedure/ function의 parameter로 사용할 수 있다.
 - Parameter를 가질 수 없다.

Cursor variable을 통해 open 된 cursor가 선언된 SCOPE를 벗어나면 자동으로 close 된다. 하지만 subprogram과 schema_level procedure/ function의 parameter로 사용된 경우에는, 해당 procedure/ function을 종료하더라도 호출한 곳의 SCOPE를 벗어날 때까지는 유효한 상태를 유지한다.

노트

Cursor variable을 parameter로 사용하려면 해당 parameter의 binding-mode는 반드시 OUT이나 IN OUT으로 선언되어야 한다.

Cursor variable도 explicit cursor와 동일한 형태의 cursor attributes를 사용할 수 있다.

```
DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Before Open');
  DBMS_OUTPUT.PUT_LINE('ISOPEN   = ' || cv1%ISOPEN );
  DBMS_OUTPUT.PUT_LINE('FOUND     = ' || cv1%FOUND );
  DBMS_OUTPUT.PUT_LINE('NOTFOUND = ' || cv1%NOTFOUND );
  DBMS_OUTPUT.PUT_LINE('ROWCOUNT = ' || cv1%ROWCOUNT );
  OPEN cv1 FOR SELECT * FROM T1;
  DBMS_OUTPUT.PUT_LINE('After Open');
  DBMS_OUTPUT.PUT_LINE('ISOPEN   = ' || cv1%ISOPEN );
  DBMS_OUTPUT.PUT_LINE('FOUND     = ' || cv1%FOUND );
```

```

DBMS_OUTPUT.PUT_LINE('NOTFOUND = ' || cv1%NOTFOUND );
DBMS_OUTPUT.PUT_LINE('ROWCOUNT = ' || cv1%ROWCOUNT );
FETCH cv1 INTO V1, V2;
DBMS_OUTPUT.PUT_LINE('After fetch');
DBMS_OUTPUT.PUT_LINE('ISOPEN   = ' || cv1%ISOPEN );
DBMS_OUTPUT.PUT_LINE('FOUND     = ' || cv1%FOUND );
DBMS_OUTPUT.PUT_LINE('NOTFOUND = ' || cv1%NOTFOUND );
DBMS_OUTPUT.PUT_LINE('ROWCOUNT = ' || cv1%ROWCOUNT );

```

```
END;
```

```
/
```

```
Before Open
```

```
ISOPEN   = FALSE
```

```
FOUND     =
```

```
NOTFOUND =
```

```
ROWCOUNT =
```

```
After Open
```

```
ISOPEN   = TRUE
```

```
FOUND     =
```

```
NOTFOUND =
```

```
ROWCOUNT = 0
```

```
After fetch
```

```
ISOPEN   = TRUE
```

```
FOUND     = TRUE
```

```
NOTFOUND = FALSE
```

```
ROWCOUNT = 1
```

```
Anonymous PL block executed.
```

Cursor variable은 다음과 같이 subprogram과 schema-level procedure/ function의 parameter로 사용할 수 있다.

```
gSQL> CREATE TABLE T1
```

```
(
```

```
  C1 VARCHAR(20),
```

```
  C2 VARCHAR(20)
```

```
);
```

```
Table created.
```

```
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
```

```
1 row created.
```

```
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
```

```
1 row created.
```

```
gSQL> DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
  PROCEDURE PROC1 ( A_CV1 IN OUT SYS_REFCURSOR )
  IS
  BEGIN
```

- 입력 parameter cursor variable을 사용하여 cursor를 open한다.

```
OPEN A_CV1 FOR SELECT * FROM T1;
END;
BEGIN
  PROC1( cv1 );
```

- Parameter로 사용된 cursor variable을 이용하여 fetch한다.

```
FETCH cv1 INTO V1, V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);

END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

자세한 내용은 **Cursor Variable Declaration**을 참조한다.

OPEN and Close Cursor Variables

PSM 내에서 OPEN FOR statement를 사용하면 cursor variable을 통해 cursor를 OPEN 할 수 있다. 또한, explicit cursor와 동일하게 CLOSE statement를 통해 cursor variable의 cursor를 CLOSE 할 수도 있다.

OPEN FOR statement를 통해 cursor를 실행할 시점에 cursor variable에 이미 열려 있는 cursor가 있을 경우, 해당 cursor는 자동으로 close 되며 새로운 상태의 cursor가 OPEN된다.

다음은 cursor variable을 사용하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
```

```
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
BEGIN
  OPEN cv1 FOR SELECT * FROM T1;
```

- Fetch with cursor-variable

```
FETCH cv1 INTO V1, V2;

DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);
CLOSE cv1;
END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

자세한 내용은 **OPEN FOR Statement**, **CLOSE Statement**를 참조한다.

Fetching Data With Cursor Variables

Explicit cursor와 동일하게 fetch statement를 통해 데이터베이스에서 처리된 결과를 PSM 변수로 저장할 수 있다. 자세한 내용은 **FETCH Statement**를 참조한다.

Assign Values to Cursor Variables

Cursor variable은 RETURN-TYPE이 동일한 경우에 서로 assign할 수 있다. Cursor variable type이 아닌 경우에는 오류가 발생한다.

다음과 같은 assign statement로 수행한다.

```
target_cursor_variable := source_cursor_variable
```

Cursor variable을 assign 하면 target_cursor_variable이 가리키는 cursor pointer가 Source_cursor_variable이 가진 cursor를 가리키게 된다. 따라서 정상적으로 assign된 이후에는 target cursor variable에 이미 열려 있는 cursor는 유효하지 않으며 접근도 할 수 없다. 또한, cursor는 cursor variable이 선언된 SCOPE를 벗어날 때 내부적으로 정리된다.

```
DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
  cv2 SYS_REFCURSOR;
BEGIN
```

- Open cv1

```
OPEN cv1 FOR SELECT * FROM T1;
```

- Assign cv1 to cv2

```
cv2 := cv1;
```

- Fetch from cv2

```
FETCH cv2 INTO V1, V2;
DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);

END;
/
V1 = Seoul , V2 = 24
```

다음은 각각의 cursor variable이 OPEN을 수행한 후에 assign에 의해 동일한 cursor를 사용하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
```


1 row created.

```
gSQL> DECLARE
  cv1 SYS_REFCURSOR;
  cv2 SYS_REFCURSOR;
  v1 t1%ROWTYPE;
BEGIN
```

- 각각의 cursor variable이 cursor를 OPEN한다.

```
OPEN cv1 FOR select * from t1 where c1 = 'Seoul';
OPEN cv2 FOR select * from t1 where c1 = 'Pusan';
```

- Assign cursor-variable
- cv2의 cursor는 더 이상 유효하지 않다.

```
cv2 := cv1;
```

- cv1의 결과를 fetch한다.

```
FETCH cv2 INTO v1;

  DBMS_OUTPUT.PUT_LINE('C1 = ' || v1.C1 || ' , C2 = ' || v1.C2);
END;
/
C1 = Seoul , C2 = 24
Anonymous PL block executed.
```


21.

Using PSM Subprograms

21.1 Anonymous PL Block

Anonymous PL block은 PSM 구문을 database에 저장하지 않고 일회성으로 실행하기 위한 SQL 구문이다. GOLDILOCKS에서 제공하는 정규 SQL 중 하나이기 때문에, GOLDILOCKS에서 제공하는 ODBC, JDBC 혹은 precompiler에서도 다른 SQL과 동일하게 사용할 수 있다. 문법은 일반 basic block 구문과 동일하다.

자세한 내용은 **Block (BEGIN .. END)**를 참조한다.

```
<<Label>> ❶ (optional)
DECLARE    ❷ (optional)
❸ declare items (variables, cursors, types, ...) (optional)
BEGIN      ❹ (required)
❺ PSM statements to execute (required)
EXCEPTION ❻ Exception Handling Part (optional)
END;
```

Anonymous PL block을 사용할 때는 다음과 같은 점에 유의해야 한다.

Interface	사용 시 주의사항
공통	Schema-level procedure나 함수와 달리 bind parameter를 사용할 수 있다. Prepare-execute 방식도 지원한다.
ODBC	일반 SQL과 완전히 동일하게 사용된다.
JDBC	IN-OUT이나 OUT 속성의 bind parameter를 가진 경우에는 CallableStatement 클래스를 사용해야 한다.
precompiler (gpec)	특별한 유의사항이 없다.
Interactive Command Tool (gsql)	Anonymous PL block을 입력한 후에 '/'<Enter> 를 입력하여 구문의 종료를 알려야 한다.

21.2 Nested Procedure

Nested procedure는 특정 PL block 내부에 선언된 procedure 타입의 subprogram이다. Nested procedure는 선언된 PL block과 그 하위에서만 참조 가능한 scope를 가진다.

자세한 내용은 **Procedure Declaration and Definition**을 참조한다.

```
DECLARE
  PROCEDURE PROC1( A1 INTEGER ) ❶ Define nested procedure
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
  END;
BEGIN
  PROC1( 100 ); ❷ Call nested procedure
END;
/
```

Nested subprogram 내부에서 사용 가능한 item들은 다음과 같다.

- Nested subprogram의 인자 (argument) 변수
- Nested subprogram이 정의된 PL block과 그 상위 scope에 정의된 변수와 각종 item들 (type, cursor,...)
- Anonymous PL block의 경우에는 bind parameter (예: '?', ':V1' 등)

Nested procedure는 forward declaration을 지원하기 때문에 declare와 define 구문을 따로 기술할 수 있다. 이를 이용하면 두 개의 nested procedure 사이에 상호 호출할 수 있는 로직을 구현할 수 있다.

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER ); ❶ declare proc1
  PROCEDURE PROC2( A1 INTEGER ) ❷ define proc2
  IS
  BEGIN
    IF A1 > 0 THEN
      DBMS_OUTPUT.PUT_LINE( '(proc2)A1 = ' || A1 );
      PROC1( A1 -1 ); ❸ call proc1
    END IF;
  END;
  PROCEDURE PROC1( A1 INTEGER ) ❹ define proc1
  IS
  BEGIN
    IF A1 > 0 THEN
```

```

        DBMS_OUTPUT.PUT_LINE( '(proc1)A1 = ' || A1 );
        PROC2( A1 -1 ); ❸ call proc2
    END IF;
END;
BEGIN
    PROC1(5); ❹ call proc1
END;
/

(proc1)A1 = 5
(proc2)A1 = 4
(proc1)A1 = 3
(proc2)A1 = 2
(proc1)A1 = 1
Anonymous PL block executed.

```

GOLDILOCKS PSM은 무한대의 상호참조를 방지하기 위해 최대 Child Statement Depth를 50개로 제한하고 있다. 이를 초과하면 다음과 같은 오류가 발생한다. (Nested function, schema-level procedure/ function 에 동일하게 적용된다.)

```

gSQL> DECLARE
    PROCEDURE PROC1( A1 INTEGER ); ❶ declare proc1
    PROCEDURE PROC2( A1 INTEGER ) ❷ define proc2
    IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
        PROC1( A1 -1 );
    END;
    PROCEDURE PROC1( A1 INTEGER ) ❸ define proc1
    IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
        PROC2( A1 -1 );
    END;
BEGIN
    PROC1(100);
END;
/
ERR-42000(16411): maximum number of recursive SQL levels (50) exceeded.

```

21.3 Nested Function

Nested function은 nested procedure와 동일하지만 함수 형태를 가지는 subprogram이다. 자세한 내용은 **Function Declaration and Definition**을 참조한다.

```
gSQL> DECLARE
  V1 INTEGER := 0;
  FUNCTION FUNC1( A1 INTEGER )
    RETURN INTEGER
  IS
  BEGIN
    RETURN A1 * 10;
  END;
BEGIN
  V1 := FUNC1( 10 );
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 100
Anonymous PL block executed.
```

Nested function은 visible scope 내에 있는 모든 PSM expression에서 사용할 수 있지만, SQL 구문 안에서는 사용할 수 없다.

```
gSQL> DECLARE
  V1 INTEGER := 0;
  FUNCTION FUNC1( A1 INTEGER )
    RETURN INTEGER
  IS
  BEGIN
    RETURN A1 * 10;
  END;
BEGIN
  SELECT FUNC1(10) INTO V1 FROM DUAL;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (10:3): ERR-HY000(17079): a nested function not allowed in executing SQL
```

21.4 Schema-level Procedure

Schema-level subprogram은 database에 저장되는 이름을 가진 SQL 객체이다. Schema-level procedure는 procedure 형태의 schema-level subprogram이다.

생성

Schema-level procedure는 다음과 같이 생성된다. 필요할 경우, 인자들의 타입에 precision과 scale 값을 명시해야 한다.

자세한 내용은 **CREATE PROCEDURE**를 참조한다.

```
CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
   FROM T1
  WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
```

생성된 schema-level procedure에 대한 정보는 INFORMATION_SCHEMA.ROUTINES 테이블로 확인할 수 있다. 자세한 내용은 **ROUTINES**를 참조한다

```
gSQL> SELECT SPECIFIC_NAME, ROUTINE_DEFINITION
  FROM INFORMATION_SCHEMA.ROUTINES
 WHERE SPECIFIC_NAME = 'PROC1';
SPECIFIC_NAME ROUTINE_DEFINITION
-----
PROC1          PROCEDURE "PUBLIC"."PROC1" ( A1 INTEGER, A2 INTEGER )
                IS
                V1 INTEGER;
                BEGIN
                SELECT COUNT(*)
                  INTO V1
                 FROM T1
```



```

        WHERE T1.I1 >= A1 AND T1.I1 <= A2;
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END;

```

1 row selected.

인자 (argument)에 대한 정보는 INFORMATION_SCHEMA.PARAMETERS 테이블로 확인할 수 있다. 자세한 내용은 **PARAMETERS**를 참조한다.

```

gSQL> SELECT P.PARAMETER_NAME, P.ORDINAL_POSITION
      FROM INFORMATION_SCHEMA.ROUTINES    R,
           INFORMATION_SCHEMA.PARAMETERS  P
     WHERE R.SPECIFIC_NAME = 'PROC1'
           AND R.SPECIFIC_SCHEMA = P.SPECIFIC_SCHEMA
           AND R.SPECIFIC_NAME = P.SPECIFIC_NAME
     ORDER BY P.ORDINAL_POSITION;
PARAMETER_NAME ORDINAL_POSITION
-----
A1                1
A2                2

```

2 rows selected.

사용

Schema-level procedure는 다른 PSM의 내부 구문이나 CALL 구문에 의해 사용된다. 다른 PSM 구문은 nested procedure와 같은 형식으로 schema-level procedure를 호출한다.

```

gSQL> BEGIN
      PROC1( 2, 4 ); ❶ call schema-level procedure
    END;
/
V1 = 3
Anonymous PL block executed.

```

주어진 PSM을 실행하는 SQL인 CALL 구문은 다음과 같이 실행된다. 자세한 내용은 **CALL Statement**를 참조한다.

```

gSQL> CALL PROC1( 2, 4 );
V1 = 3
Procedure Call complete.

```

ODBC나 JDBC에서 사용하는 procedure call escape sequence를 지원하기 위해 procedure에 대한 다음 구문을 지원한다.

```
{ CALL procedure_name( param1, param2, ... ) }
```

Procedure call escape sequence 구문은 일반 SQL처럼 사용할 수 있다.

```
gSQL> { CALL PROC1(2, 4) };
V1 = 3
Procedure Call complete.
```

GOLDILOCKS의 interactive command tool인 gsql은 tool 자체 명령어인 \EXEC를 통한 procedure 수행을 지원하지 않는다.

Schema-level procedure는 실행 시 권한에 대한 다음 옵션을 명시할 수 있다. Definer가 아닌 사용자가 이 옵션들에 따라 PSM내의 SQL을 수행하면 해당 사용자의 schema-path 정의에 따라 서로 다른 schema의 이름이 같은 테이블을 참조할 수도 있다. (Item을 선언할 때 사용된 객체명 (예: T1%ROWTYPE)은 항상 definer로 해석된다.)

- AUTHID DEFINER (default): 해당 procedure를 작성한 사용자로 변경된 후 수행된다.
- AUTHID CURRENT_USER: 수행하는 사용자를 변경하지 않고 현재 사용자에게 의해 수행된다.

```
CREATE OR REPLACE PROCEDURE "PROC1"( A1 INTEGER, A2 INTEGER )
AUTHID CURRENT_USER
IS
    V1 INTEGER;
BEGIN
    SELECT COUNT(*)
        INTO V1
        FROM T1
        WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
```

제거

Schema-level procedure는 다음과 같이 DROP PROCEDURE 구문으로 제거한다. 자세한 내용은 **DROP PROCEDURE**를 참조한다.

```
DROP PROCEDURE PROC1;
```

GOLDILOCKS의 다른 DROP 구문들처럼 IF EXISTS 구문도 지원한다.

```
DROP PROCEDURE IF EXISTS PROC1;
```

Recompile

Schema-level subprogram 내부에서 참조된 객체가 변경되면 해당 subprogram도 영향을 받아 실행 플랜을 다시 생성해야 할 수도 있다.

선언부나 인자에서 참조된 객체

선언부의 각종 item을 정의하거나 인자 타입을 정의할 때 사용된 객체가 변경되면 procedure의 플랜이 자동으로 re compile 된다.

- %TYPE, %ROWTYPE에 사용된 객체
- Explicit cursor 정의 구문에 사용된 객체

객체가 변경된 후 처음으로 해당 procedure가 수행되면 다음 순서에 따라 자동으로 플랜을 다시 생성한다.

1. Plan cache로부터 해당 procedure의 플랜을 가져온다
2. 해당 플랜의 객체 리스트를 validation하는 중에 변경된 객체를 찾는다.
3. 현재 플랜을 discard 시키고, dictionary에 저장된 procedure 정의 구문으로부터 새로운 플랜을 생성한다.
4. 새로 생성된 플랜을 plan cache에 등록한다.
5. 플랜을 실행한다.

Body의 SQL에서 참조된 객체

Procedure의 플랜에는 body에 사용된 SQL의 플랜이 저장되지 않고 SQL text만 저장된 상태이다. Procedure를 실행할 때 해당 SQL 구문을 실시간으로 compile 하여 플랜을 생성한 후 수행한다. 따라서 body 내에서 사용된 SQL 객체는 procedure의 플랜 자체에 영향을 주지 않는다.

단, 객체가 변경되면 procedure의 기존 interface인 바인드 개수나 타입이 변경되거나 column이 삭제될 수 있으므로 procedure까지 적절하게 변경하지 않으면 실행할 때 오류가 발생할 수 있다.

21.5 Schema-level Function

Schema-level function은 expression 내에서 사용되는 함수 형태의 schema-level subprogram이다.

생성

Schema-level function은 다음과 같이 생성된다. 필요할 경우, 인자들의 타입에 precision과 scale 값을 명시해야 한다.

자세한 내용은 **CREATE FUNCTION**을 참조한다.

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
RETURN INTEGER
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO V1
  FROM T1
  WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  RETURN V1;
END;
/
Function created.
```

생성된 schema-level function은 procedure와 마찬가지로 INFORMATION_SCHEMA.ROUTINES와 INFORMATION_SCHEMA.PARAMETERS 테이블에서 확인할 수 있다.

사용

Schema-level function은 일반 SQL이나 PSM 내의 SQL 객체가 사용될 수 있는 모든 expression에서 사용할 수 있다.

```
gSQL> SELECT FUNC1( 2, 4 ) FROM DUAL;
FUNC1( 2, 4 )
-----
              3
1 row selected.
```

PSM 내에서도 다음과 같이 사용된다.

```
gSQL> DECLARE
  V1 INTEGER;
BEGIN
  V1 := FUNC1( 2, 4 );
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 3
Anonymous PL block executed.
```

다음과 같이 CALL 구문을 이용하여 수행할 수 있다.
자세한 내용은 **CALL Statement**를 참조한다.

```
gSQL> \var V1 INTEGER
gSQL> CALL FUNC1( 2, 4 ) INTO :V1;
Procedure Call complete.
gSQL> \print V1
V1
--
3
```

Schema-level procedure와 마찬가지로 procedure call escape sequence 구문도 지원한다.

```
{ ? = CALL function_name( param1, param2, ... ) }
```

```
gSQL> { :V1 = CALL FUNC1( 2, 4 ) };
Procedure Call complete.
gSQL> \print V1
V1
--
3
```

제거

Schema-level function은 다음과 같이 DROP FUNCTION 구문으로 제거한다.
자세한 내용은 **DROP FUNCTION**을 참조한다.

```
gSQL> DROP FUNCTION FUNC1;
```

```
Function dropped.
```

21.6 Built-in Procedures

GOLDILOCKS PSM은 procedure와 function을 구현할 때 디버깅하거나 exception을 처리하기 위해 다음과 같은 built-in procedure들을 제공한다.

표 21-1 Built-in procedures

Procedure	기능
DBMS_OUTPUT.ENABLE(buffer_size IN NATIVE_INTEGER := 20000)	주어진 버퍼 크기로 메시지 로깅 기능을 활성화 한다. 만일 버퍼 크기가 주어지지 않으면 디폴트로 20000 byte로 설정된다. 기존에 이미 활성화 되어 있으면 모든 메시지를 버리고 새로 버퍼를 생성한다.
DBMS_OUTPUT.DISABLE	메시지 로깅 기능을 비활성화 한다. 기존에 로깅된 모든 메시지는 버려진다.
DBMS_OUTPUT.SET_LOG(file_path IN VARCHAR(4000))	메시지를 로깅할 때 주어진 경로에 있는 파일에도 동시에 출력한다. 만일 상대 경로 (첫 글자가 /가 아닌 경우)로 주어지면 <SYSTEM_LOGGER_DIR> 프로퍼티에 해당하는 디렉토리 아래에서 대상 파일을 찾는다.
DBMS_OUTPUT.PUT_LINE(item IN VARCHAR(4000))	주어진 expression으로 만들어진 메시지를 버퍼에 저장한다.
DBMS_OUTPUT.GET_LINE(line OUT VARCHAR(4000), status OUT NATIVE_INTEGER)	버퍼에 저장된 메시지들 중에 아직 읽지 않은 가장 오래된 메시지를 한 줄 반환한다. 메시지가 존재하면 status는 0을 반환하고 없으면 1을 반환한다.
DBMS_STANDARD.RAISE_APPLICATION_ERROR(error_code IN NATIVE_INTEGER, error_message IN VARCHAR(4000), stack_flag IN BOOLEAN := FALSE)	임의의 사용자 exception을 발생시킨다. error_code는 -20000 ~ -20999 사이의 값이어야 하며, TRUE일 경우에는 마지막 stack_flag 인자가 기존 error 들 위에 주어진 에러를 쌓고 FALSE이면 해당 에러가 모든 에러를 대체 (replace)한다 (생략할 수 있고 이 경우 default는 FALSE이다.).

```

gSQL> DECLARE
V1 VARCHAR(1024);
V2 INTEGER;
BEGIN
  DBMS_OUTPUT.ENABLE(2000);
  DBMS_OUTPUT.PUT_LINE('TEST MSG');
  DBMS_OUTPUT.GET_LINE( V1, V2);
  DBMS_OUTPUT.PUT_LINE('V1 = ' || v1);
  DBMS_OUTPUT.PUT_LINE('V2 = ' || v2);
END;
/
V1 = TEST MSG
V2 = 0
Anonymous PL block executed.

```

메시지 로깅 기능은 세션별로 관리되며, GOLDILOCKS의 interactive command tool인 gsql에서는 serveroutput 옵션으로 메시지 로깅 기능을 켜고 끌 수 있다. PSM을 수행한 후에 메시지 버퍼에 내용이 있으면 모두 자동으로 출력된다.

```
gSQL> \set serveroutput on
gSQL> \var msg VARCHAR(4000)
gSQL> \var status NATIVE_INTEGER
gSQL> CALL DBMS_OUTPUT.PUT_LINE( 'aaa' );
aaa
Procedure Call complete.
gSQL> CALL DBMS_OUTPUT.PUT_LINE( 'bbb' );
bbb
Procedure Call complete.
gSQL> CALL DBMS_OUTPUT.GET_LINE( :msg, :status );
Procedure Call complete.
gSQL> \print msg
MSG
----
null
gSQL> \print status
STATUS
-----
```


22.

Using SQLs in PSM

22.1 Static SQLs

Static SQL은 PSM에서 사용할 수 있는 SQL로써 다음과 같이 분류한다.

- SELECT
- INSERT
- UPDATE
- DELETE
- RETURNING INTO
- LOCK TABLE
- COMMIT, ROLLBACK, SAVEPOINT

Static SQL은 GOLDDILOCKS에서 지원하는 SQL을 PSM 변수를 통해 실행할 수 있도록 확장된 SQL이다. 본 장에서는 PSM에서만 사용할 수 있는 static SQL의 기능과 주의사항에 대해 설명한다.

SELECT

PSM 내의 SELECT 문은 데이터베이스로부터 결과를 반환받아 INTO 절에 기술된 PSM 변수에 값을 저장한다. SELECT INTO를 통한 GOLDDILOCKS PSM의 결과 집합은 single row만 저장할 수 있다.

다음과 같은 형태로 표현할 수 있다.

```
SELECT target_list INTO variable_list FROM table_expression
```

자세한 내용은 **Data Query Language**를 참조한다.

다음은 PSM 내에서 SELECT 문을 이용하여 SQL data type으로 선언된 변수에 값을 저장하는 예이다.

```
DECLARE
  V1 INTEGER;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 =[ ' || V1 || ' ]');
```

- SELECT INTO를 통해 V1 변수에 결과를 저장한다.

```
SELECT 1 INTO V1 FROM DUAL;
DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1);
END;
/
V1 =[ ]
```

V1 = 1

다음은 record type으로 선언된 변수에 값을 저장하는 예이다.

```
DECLARE
  TYPE rec is RECORD (F1 INTEGER, F2 INTEGER );
  var1 rec;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'Var1.f1 =[ ' || Var1.f1 || ' ]');
  DBMS_OUTPUT.PUT_LINE( 'Var1.f2 =[ ' || Var1.f2 || ' ]');
```

- Record type 변수인 var에 결과를 저장한다.

```
SELECT 1, 2 INTO var1 FROM DUAL;
  DBMS_OUTPUT.PUT_LINE( 'Var1.f1 = ' || Var1.f1);
  DBMS_OUTPUT.PUT_LINE( 'Var1.f2 = ' || Var1.f2);
END;
/
Var1.f1 =[
Var1.f2 =[
Var1.f1 = 1
Var1.f2 = 2
```

SELECT 문의 결과가 두 개 이상인 결과 집합이 데이터베이스로부터 반환될 경우 다음과 같은 오류가 발생한다. 사용하는 predefined exception 중에 TOO_MANY_ROWS exception으로 오류를 catch 할 수 있다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
BEGIN
  BEGIN
    SELECT * INTO V1, V2 FROM T1;
```

```

EXCEPTION WHEN TOO_MANY_ROWS
    THEN DBMS_OUTPUT.PUT_LINE( 'SQLCODE=' || SQLCODE);
         DBMS_OUTPUT.PUT_LINE( 'SQLERRM=' || SQLERRM);

END;
END;
/
SQLCODE=-16289
SQLERRM=[SUNJESOFT][PSM][GOLDILOCKS]into clause can have only one row
Anonymous PL block executed.

```

SELECT INTO에 결과가 존재하지 않으면 오류가 발생한다. 사용자는 predefined exception 중에 NO_DATA_FOUND exception으로 오류를 catch 할 수 있다.

```

DECLARE
    V1 T1%ROWTYPE;
BEGIN
    SELECT * INTO V1 FROM T1 WHERE C1 = 'NONE';
END;
/
ERR-HY000(17041): execution fail :
    SELECT * INTO V1 FROM T1 WHERE C1 = 'NONE';
    *
ERROR at line 4:
ERR-HY000(17045): no data found

```

주의

GOLDILOCKS에서 SELECT INTO 문의 결과를 record type을 통해 저장할 경우 다른 타입 변수와 섞어서 사용할 수 없다. 섞어서 사용할 경우, 다음과 같은 오류가 발생한다.

```

DECLARE
    TYPE rec is RECORD (F1 INTEGER, F2 INTEGER );
    var1 rec;
    v1 integer;
BEGIN
    DBMS_OUTPUT.PUT_LINE( 'Var1.f1 =[ ' || Var1.f1 || ' ]');
    DBMS_OUTPUT.PUT_LINE( 'Var1.f2 =[ ' || Var1.f2 || ' ]');
    SELECT 1, 2, 3 INTO Var1, v1 FROM DUAL;
    DBMS_OUTPUT.PUT_LINE( 'Var1.f1 = ' || Var1.f1);

```

```

    DBMS_OUTPUT.PUT_LINE( 'Var1.f2 = ' || Var1.f2);
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (9:29): ERR-HY000(17046): variables of record type can not be mixed with other
variables

```

다만, record type의 일부 필드와 SQL data type 변수는 다음과 같이 섞어서 사용할 수 있다.

```

gSQL> CREATE TABLE T1
(
    C1 VARCHAR(20),
    C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
    v1 t1%ROWTYPE;
    v2 VARCHAR(20);
BEGIN
    SELECT * INTO V1.C1, V2 FROM T1
    WHERE C1 = 'Seoul';

```

- V1.c2에는 값이 저장되지 않는다.

```

DBMS_OUTPUT.PUT_LINE('V1.c1 = ' || V1.C1 || ' , V1.c2 = ' || V1.C2);
    DBMS_OUTPUT.PUT_LINE('V2      = ' || V2);
END;
/
V1.c1 = Seoul , V1.c2 =
V2      = 24
Anonymous PL block executed.

```

INSERT

PSM에서 INSERT 문은 다음과 같은 확장 기능을 제공한다.

- 사용자가 정의한 record type 변수를 이용하여 데이터베이스에 데이터를 저장할 수 있다.
- Record type 변수를 이용하여 RETURNING INTO 절에 데이터를 저장할 수 있다.

PSM에서는 record type 변수를 통해 다음과 같은 형식으로 데이터를 삽입할 수 있다.

```
INSERT INTO table_name VALUES record_type_variable
```

자세한 내용은 **데이터 추가**를 참조한다.

다음은 SQL data type으로 선언된 PSM 변수를 이용하여 데이터베이스에 레코드를 저장하는 예이다.

```
gSQL>
CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
gSQL>
DECLARE
  V1 INTEGER;
  V2 INTEGER;
BEGIN
  V1 := 10;
  V2 := 20;

  INSERT INTO T1 (C1, C2) VALUES (V1, V2);
  COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1 C2
-- --
10 20
1 row selected.
```

다음은 %ROWTYPE으로 선언된 record type 변수를 이용하여 insert하는 예이다.

```
gSQL>
CREATE TABLE T1
(
```

```

    C1 INTEGER,
    C2 INTEGER
);
Table created.
gSQL>
DECLARE
    V1 T1%ROWTYPE;
BEGIN
    V1.C1 := 10;
    V1.C2 := 20;

    INSERT INTO T1 VALUES V1;
    COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1 C2
-- --
10 20
1 row selected.

```

자세한 내용은 **%ROWTYPE**을 참조한다.

다음은 user-defined record type 변수를 이용하여 데이터베이스에 레코드를 저장하는 예이다.

```

gSQL>
CREATE TABLE T1
(
    C1 INTEGER,
    C2 INTEGER
);
Table created.
gSQL>
DECLARE
    TYPE rec IS RECORD (F1 INTEGER, F2 INTEGER);
    v1 rec;
BEGIN
    V1.F1 := 10;
    V1.F2 := 20;

    INSERT INTO T1 VALUES V1;

```

```

    COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1 C2
-- --
10 20
1 row selected.

```

자세한 내용은 **User-defined Record Type**을 참조한다.

주의

PSM insert 확장이 아닌 다른 형태로 record type 변수를 기술할 경우 다음과 같은 오류가 발생한다.

```

gSQL>
DECLARE
    TYPE rec IS RECORD (F1 INTEGER, F2 INTEGER);
    v1 rec;
    v2 INTEGER;
BEGIN
    V1.F1 := 10;
    V1.F2 := 20;
    V2 := 30;

    INSERT INTO T1 (c1, c2) VALUES (V1);
    COMMIT;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (11:3): ERR-42000(16053): not enough values

```

위의 예에는 두 개의 column이 나열되어 있으며 기술된 VALUES 절의 변수는 한 개이다. Record type의 변수이긴 하지만 binding 되어야 하는 값의 개수가 다르기 때문에 위의 예와 같은 오류가 발생한다. Record type과 다른 유형의 변수를 섞어서 사용하려면 다음과 같이 scalar-type으로 기술해야 한다.

```

gSQL>
DECLARE

```



```

TYPE rec IS RECORD (F1 INTEGER, F2 INTEGER);
v1 rec;
v2 INTEGER;
BEGIN
  V1.F1 := 10;
  V1.F2 := 20;
  V2 := 30;

  INSERT INTO T1 (c1, c2) VALUES (V1.F1, V2);
  COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1 C2
-- --
10 30
1 row selected.

```

UPDATE

PSM에서 확장된 기능은 다음과 같다.

- SET ROW 절을 이용하여 데이터베이스의 값을 record type 변수에 있는 값으로 갱신할 수 있다.
- RETURNING INTO 절을 통해 갱신 전/ 후의 값을 PSM 변수로 저장할 수 있다.

PSM 내 UPDATE 확장 기능을 통해 record type 변수를 다음과 같은 형태로 사용할 수 있다.

```
UPDATE table_list SET ROW = Record_type_variable [WHERE condition]
```

자세한 내용은 **데이터 갱신**을 참조한다.

다음은 PSM record type 변수와 UPDATE ... SET ROW 절을 통해 레코드를 갱신하는 예이다.

```

gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1);

```

```

1 row created.
gSQL> DECLARE
    V1 T1%ROWTYPE;
BEGIN
    V1.C1 := 10;
    V1.C2 := 20;

    UPDATE T1 SET ROW = V1 WHERE c1 = 1;
    COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1 C2
-- --
10 20
1 row selected.

```

DELETE

PSM DELETE의 확장 기능은 다음과 같다.

- RETURNING INTO를 통해 삭제된 값을 record type 변수로 저장할 수 있다.

다음은 SQL data type PSM 변수를 통해 DELETE를 수행하는 예이다.

```

gSQL> CREATE TABLE T1
(
    C1 INTEGER,
    C2 INTEGER
);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1);
1 row created.
gSQL> DECLARE
    V1 INTEGER;
BEGIN
    V1 := 1;

    DELETE FROM T1 WHERE C1 = V1;

```

```

    COMMIT;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
no rows selected.

```

RETURNING INTO

DML (INSERT, UPDATE, DELETE) 처리 결과를 RETURNING INTO 절을 이용해 PSM 변수로 저장할 수 있다.

- PSM의 RETURNING INTO에 결과를 저장할 경우 다음과 같은 제약을 받는다.
 - 데이터베이스로 반환되는 결과는 single row로만 저장할 수 있다. (두 건 이상의 결과를 저장할 수 없다.)
 - Record type 변수로 저장할 경우 다른 유형의 변수와 함께 사용할 수 없다.

INSERT RETURNING INTO

INSERT RETURNING INTO 절은 데이터베이스에 저장된 결과값을 RETURNING INTO에 기술된 PSM 변수로 저장한다.

다음은 insert를 수행한 결과를 PSM 변수에 저장하는 예이다.

```

gSQL>
DECLARE
    v1 INTEGER;
    v2 INTEGER;
    v3 INTEGER;
    v4 INTEGER;
BEGIN
    V1 := 10;
    V2 := 20;

    INSERT INTO T1 (c1, c2) VALUES (V1, V2)
    RETURNING * INTO V3, V4;
    DBMS_OUTPUT.PUT_LINE( 'V3 = ' || V3 );
    DBMS_OUTPUT.PUT_LINE( 'V4 = ' || V4 );
END;
/
V3 = 10
V4 = 20

```

다음은 record type 변수를 이용하여 returning into의 결과를 저장하는 예이다.

```
gSQL>
DECLARE
  v1 INTEGER;
  v2 INTEGER;
  v3 T1%ROWTYPE;
BEGIN
  V1 := 10;
  V2 := 20;

  INSERT INTO T1 (c1, c2) VALUES (V1, V2)
  RETURNING * INTO V3;
  DBMS_OUTPUT.PUT_LINE( 'V3.C1 = ' || V3.C1 );
  DBMS_OUTPUT.PUT_LINE( 'V3.C2 = ' || V3.C2 );
END;
/
V3.C1 = 10
V3.C2 = 20
Anonymous PL block executed.
```

UPDATE RETURNING INTO

UPDATE RETURNING INTO는 RETURNING INTO 절에 기술된 PSM 변수를 통해 갱신되기 전/ 후의 결과를 데이터 베이스에 저장할 수 있다.

다음과 같이 SQL type 변수를 통해 RETURNING INTO의 결과를 저장한다.

```
gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1);
1 row created.
gSQL> DECLARE
  V1 INTEGER;
  V2 INTEGER;
  V3 INTEGER;
  V4 INTEGER;
BEGIN
```

```
V1 := 1;
V2 := 2;
```

- UPDATE에 의해 갱신되기 전의 결과를 V3와 V3 변수에 저장한다.

```
UPDATE T1 SET C2 = V2 WHERE C1 = V1
RETURNING OLD * INTO V3, V4;
DBMS_OUTPUT.PUT_LINE('OLD');
DBMS_OUTPUT.PUT_LINE('V3 = ' || V3);
DBMS_OUTPUT.PUT_LINE('V4 = ' || V4);
ROLLBACK;
```

- UPDATE에 의해 갱신된 후의 결과를 V3와 V3 변수에 저장한다.

```
UPDATE T1 SET C2 = V2 WHERE C1 = V1
RETURNING NEW * INTO V3, V4;
DBMS_OUTPUT.PUT_LINE('NEW');
DBMS_OUTPUT.PUT_LINE('V3 = ' || V3);
DBMS_OUTPUT.PUT_LINE('V4 = ' || V4);
END;
/
OLD
V3 = 1
V4 = 1
NEW
V3 = 1
V4 = 2
Anonymous PL block executed.
```

다음은 record type 변수를 이용하여 RETURNING INTO의 결과를 저장하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1);
1 row created.
gSQL> DECLARE
  V1 INTEGER;
  V2 INTEGER;
```

```
V3 T1%ROWTYPE;
BEGIN
  V1 := 1;
  V2 := 2;
```

- Record type 변수인 V3에 결과를 저장한다.

```
UPDATE T1 SET C2 = V2 WHERE C1 = V1
RETURNING OLD * INTO V3;
DBMS_OUTPUT.PUT_LINE('OLD');
DBMS_OUTPUT.PUT_LINE('V3.C1 = ' || V3.C1);
DBMS_OUTPUT.PUT_LINE('V3.C2 = ' || V3.C2);
ROLLBACK;
```

- Record type 변수인 V3에 결과를 저장한다.

```
UPDATE T1 SET C2 = V2 WHERE C1 = V1
RETURNING NEW * INTO V3;
DBMS_OUTPUT.PUT_LINE('NEW');
DBMS_OUTPUT.PUT_LINE('V3.C1 = ' || V3.C1);
DBMS_OUTPUT.PUT_LINE('V3.C2 = ' || V3.C2);
END;
/
OLD
V3.C1 = 1
V3.C2 = 1
NEW
V3.C1 = 1
V3.C2 = 2
Anonymous PL block executed.
```

DELETE RETURNING INTO

DELETE RETURNING INTO는 데이터베이스에서 삭제한 row의 데이터를 PSM 변수에 저장할 수 있다.

다음은 SQL type 변수들을 사용하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
```

Table created.

```
gSQL> INSERT INTO T1 VALUES (1, 1);
```

1 row created.

```
gSQL> DECLARE
```

```
    V1 INTEGER;
```

```
    V2 INTEGER;
```

```
    V3 INTEGER;
```

```
BEGIN
```

```
    V3 := 1;
```

- V1, V2 변수에 DELETE 이전의 데이터를 저장한다.

```
DELETE FROM T1 WHERE C1 = V3
```

```
RETURNING * INTO V1, V2;
```

```
DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 );
```

```
DBMS_OUTPUT.PUT_LINE('V2 = ' || V2 );
```

```
END;
```

```
/
```

```
V1 = 1
```

```
V2 = 1
```

Anonymous PL block executed.

다음은 record type 변수를 통해 DELETE 이전의 data를 저장하는 예이다.

```
gSQL> CREATE TABLE T1
```

```
(
```

```
    C1 INTEGER,
```

```
    C2 INTEGER
```

```
);
```

Table created.

```
gSQL> INSERT INTO T1 VALUES (1, 1);
```

1 row created.

```
gSQL> DECLARE
```

```
    V1 T1%ROWTYPE;
```

```
    V3 INTEGER;
```

```
BEGIN
```

```
    V3 := 1;
```

- V3 변수에 DELETE 전의 데이터를 저장한다.

```

DELETE FROM T1 WHERE C1 = V3
RETURNING * INTO V1;
DBMS_OUTPUT.PUT_LINE('V1.C1 = ' || V1.C1 );
DBMS_OUTPUT.PUT_LINE('V1.C2 = ' || V1.C2 );
END;
/
V1.C1 = 1
V1.C2 = 1
Anonymous PL block executed.

```

COMMIT, ROLLBACK, SAVEPOINT

COMMIT, ROLLBACK, SAVEPOINT 구문은 PSM 내에서 발생한 트랜잭션 결과를 데이터베이스에 영구 저장하거나 갱신 이전의 상태로 돌려 놓는다.

COMMIT

COMMIT은 트랜잭션 결과를 데이터베이스에 영구적으로 저장한다. 다음과 같이 사용된다.

```

CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
INSERT INTO T1 VALUES (1, 1);
1 row created.
DECLARE
  V3 INTEGER;
BEGIN
  V3 := 1;

  DELETE FROM T1 WHERE C1 = V3;
  COMMIT;
END;
/
Anonymous PL block executed.
SELECT * FROM T1;
no rows selected.

```


ROLLBACK

ROLLBACK은 트랜잭션을 이전 상태로 복구한다. 다음과 같이 사용된다.

```
gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1);
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  V3 INTEGER;
BEGIN
  V3 := 1;

  DELETE FROM T1 WHERE C1 = V3;
  ROLLBACK;
END;
/
Anonymous PL block executed.
```

- DELETE ROLLBACK 한 데이터가 PSM에서 정상적으로 조회되는지 확인한다.

```
gSQL> SELECT * FROM T1;
C1 C2
-- --
1  1
1 row selected.
```

SAVEPOINT

SAVEPOINT는 ROLLBACK TO SAVEPOINT 문을 이용하여 사용자가 지정한 위치로 트랜잭션을 rollback 한다.

```
gSQL> CREATE TABLE T1
(
  C1 INTEGER,
  C2 INTEGER
);
```

Table created.

```
gSQL> INSERT INTO T1 VALUES (1, 1);
```

1 row created.

```
gSQL> DECLARE
```

```
  V1 INTEGER;
```

```
  BEGIN
```

```
    UPDATE T1 SET C2 = 10 WHERE C1 = 1;
```

```
    SELECT C2 INTO V1 FROM T1 WHERE C1 = 1;
```

```
    DBMS_OUTPUT.PUT_LINE('Before delete: V1 = ' || V1 );
```

- 현재 위치를 SAVEPOINT로 지정한다.

```
SAVEPOINT SV1;
```

```
DELETE FROM T1 WHERE C1 = 1;
```

- SV1 지점까지 ROLLBACK 한다.

```
ROLLBACK TO SAVEPOINT SV1;
```

- SAVEPOINT까지 정상적으로 ROLLBACK 되었는지 확인한다.

```
SELECT C2 INTO V1 FROM T1 WHERE C1 = 1;
```

```
DBMS_OUTPUT.PUT_LINE('After Rollback: V1 = ' || V1 );
```

```
END;
```

```
/
```

```
Before delete: V1 = 10
```

```
After Rollback: V1 = 10
```

Anonymous PL block executed.

22.2 Dynamic SQL

Dynamic SQL은 static SQL과 달리 syntax가 실행 시점에 결정된다.

PSM에서는 EXECUTE IMMEDIATE나 OPEN FOR 구문을 통해 run-time에 사용자가 작성한 dynamic SQL을 수행할 수 있다.

- 다음과 같은 경우에 dynamic SQL을 사용한다.
 - Compile 할 때 SQL 문을 결정할 수 없는 경우 (예: 조건에 따라 SQL 문이 달라져야 하는 경우)
 - Static SQL에서 지원되지 않는 SQL을 수행하는 경우 (예: DDL)
- Dynamic SQL은 실행 시점까지 syntax를 알 수 없기 때문에 syntax 오류, 대상 object의 존재 여부, 사용자 권한 등의 validation에 따라 run-time 오류가 발생할 수 있다.
- Dynamic SQL은 다음과 같은 PSM statement에서 사용할 수 있다.
 - **EXECUTE IMMEDIATE Statement**
 - **OPEN FOR Statement**

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE에서는 다양한 dynamic SQL을 수행할 수 있다. 다만, PSM에서 제공되는 SQL extension 형태의 구문은 사용할 수 없다.

다음과 같은 형태의 구문이 제공된다.

```
EXECUTE IMMEDIATE 'dynamic sql' [ USING [IN | OUT | INOUT] variable_list] [INTO variable_list]
[RETURNING INTO variable_list]
```

EXECUTE IMMEDIATE 구문에 제공되는 USING, INTO, RETURNING INTO 구문을 통해 PSM 변수에 저장된 값을 데이터베이스에 적용하거나 데이터베이스로부터 PSM 변수에 값을 저장할 수 있다.

각 구문의 사용 방법에는 다음과 같은 차이가 있다.

- USING
 - PSM 변수의 값을 SQL에 적용할 경우 IN-mode를 사용한다.
 - SQL의 처리 결과를 PSM 변수에 저장하고자 할 경우 OUT-mode를 사용한다.
 - 따라서 OUT-mode로 사용된 경우 반드시 PSM 변수로 기술되어야 한다. (Expression 사용 불가)
 - USING 절에 기술된 변수의 binding type이 명시되지 않을 경우 IN-mode가 적용 된다.
 - SELECT의 target을 USING OUT으로 반환받을 경우 SELECT INTO 절을 사용한 것과 동일하게 동작한다.
 - 데이터베이스로부터는 single row만 반환받을 수 있으며 두 개 이상의 결과가 발생하면 오류가 발생한다.
 - USING IN 절의 PSM 변수는 scalar type 형태로만 사용할 수 있다.
 - USING OUT 절의 PSM 변수는 record type도 가능하지만 다른 유형과 섞어서 사용할 수는 없다.

- INTO
 - 데이터베이스 내부적으로 implicit cursor를 사용하여 처리되는 SELECT의 결과를 저장할 경우에 사용한다.
 - SELECT 절을 dynamic SQL 방식으로 실행했을 경우에만 사용할 수 있다. (SELECT INTO 절 사용 불가)
 - PSM 변수는 record-type도 가능하지만 다른 유형과 섞어서 사용할 수 없다.
- RETURNING INTO
 - INSERT/ UPDATE/ DELETE에 의해 처리된 데이터의 전/ 후를 저장할 경우에 사용한다.
 - Single row만 저장할 수 있다.
 - PSM 변수는 record type도 가능하지만 다른 유형과 섞어서 사용할 수 없다.

다음은 dynamic SQL을 사용하여 SELECT INTO 구문의 결과를 출력하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  V1 VARCHAR(20);
  V2 VARCHAR(20);
BEGIN
  EXECUTE IMMEDIATE 'SELECT * INTO ?, ? FROM T1 WHERE C1 = ''Seoul'''
  USING OUT V1, OUT V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1);
  DBMS_OUTPUT.PUT_LINE('V2 = ' || V2);
END;
/
V1 = Seoul
V2 = 24
Anonymous PL block executed.
```

다음은 갱신 연산을 수행하고 갱신되기 이전의 결과를 RETURNING INTO에 기술된 변수로 저장하는 예이다.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
```

```

Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
    V1 VARCHAR(20);
    V2 VARCHAR(20);
    V3 VARCHAR(20);
    V4 VARCHAR(20);
BEGIN
    V1 := 'Daegu';
    V2 := '50';
    EXECUTE IMMEDIATE
        'UPDATE T1 SET C1 = ? , C2 = ? WHERE C1 = ''Seoul'' RETURNING OLD * INTO ?, ?'
    USING V1, V2 RETURNING INTO V3, V4;
    DBMS_OUTPUT.PUT_LINE('V3 = ' || V3);
    DBMS_OUTPUT.PUT_LINE('V4 = ' || V4);
END;
/
V3 = Seoul
V4 = 24
Anonymous PL block executed.

```

자세한 내용은 EXECUTE IMMEDIATE Statement를 참조한다.

OPEN FOR, FETCH and CLOSE

EXECUTE IMMEDIATE를 사용하여 조회를 처리할 경우 한 건 이상을 데이터베이스로부터 반환받을 수 없다. 처리할 SQL이 dynamic SQL이고 cursor와 같이 두 건 이상의 결과 집합을 fetch 해야 할 경우라면 OPEN FOR를 사용할 수 있다.

다음과 같은 형태로 OPEN FOR에 dynamic SQL을 사용할 수 있다.

```
OPEN Cursor_variable FOR dynamic_sql [USING variable_list]
```

다음은 dynmaic SQL을 통해 OPEN FOR를 수행하는 예이다.

```

gSQL> CREATE TABLE T1
(
    C1 VARCHAR(20),

```

```

    C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
    v1 VARCHAR(20);
    v2 VARCHAR(20);
    v3 VARCHAR(20);
    cv1 SYS_REFCURSOR;
    sqlstr VARCHAR(1024);
BEGIN

    sqlstr := 'SELECT * FROM T1 WHERE C1 >= ?';
    v3 := 'AAAA';
    OPEN cv1 FOR sqlstr USING v3;
    FETCH cv1 INTO v1, v2;
    DBMS_OUTPUT.PUT_LINE('V1 = ' || v1 || ' , V2 = ' || v2);
    CLOSE cv1;
END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.

```

- OPEN FOR Statement의 USING 절에는 다음과 같은 제한이 있다.
 - Binding mode
 - IN mode만 가능하다.
 - 사용 가능한 항목
 - PSM 변수
 - Scalar type으로 결과가 반환되는 value expression이다.

자세한 내용은 **OPEN FOR Statement**, **FETCH Statement**, **CLOSE Statement**를 참조한다.

23.

PSM Language Element References

23.1 Assignment Statement

기능

PSM block의 내부에서 변수나 out-bind parameter에 값을 저장한다.

구문

```

<assignment statement> ::=
    <assignment target> := <value expression>
    ;
<assignment target> ::=
    collection_variable ( index )
    | cursor_variable
    | :host_cursor_variable
    | out_parameter
    | :host_variable [ :indicator_variable ]
    | record_variable . field_name
    | scalar_variable

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- collection_variable
 - DECLARE 영역에서 선언된 COLLECTION 타입의 변수 이름이다.
- index
 - collection_variable의 element들 중에서 한 개를 고를 key 값이다.
- cursor_variable
 - DECLARE 영역에서 선언된 cursor 이름이다.
- host_variable

- PSM을 호출하는 곳에서 전달된 out 또는 in-out 타입의 client-side 변수 이름이다.
- indicator_variable
 - host_variable의 NULL 값을 체크하고 실제 값의 길이를 알아내기 위한 indicator 변수의 이름이다.
- record_variable
 - DECLARE 영역에서 이미 선언된 RECORD 타입의 변수 이름이다.
- field_name
 - RECORD 타입의 변수에 포함된 필드 중 한 개의 이름이다.
- scalar_variable
 - DECLARE 영역에서 이미 선언된 일반 scalar 타입의 변수 이름이다.

설명

Assignment 문의 target은 외부의 bind parameter와 내부의 PSM 변수로 나뉜다.

- 외부 parameter
 - Host variable
 - Host cursor variable
- 내부 변수
 - Out 타입 함수 인자
 - 내부에서 선언된 변수
 - Scalar/ record/ multi-set 타입
 - Record 타입 변수의 특정 필드
 - Multi-set 타입 변수의 특정 element

Procedure/ function parameter를 제외한 모든 변수는 scope를 지정하는 이름을 가질 수 있다.

사용 예

다음은 <assignment statement>를 사용하는 예이다.

- PSM 내부 변수에 대한 assignment

```
gSQL> DECLARE
      V1 INTEGER := 0;
BEGIN
      FOR I IN 1..10 LOOP
          V1 := V1 + I;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
```

```
END;
/
V1 = 55
Anonymous PL block executed.
```

- Host variable에 대한 assignment

```
gSQL> \var P1 INTEGER
gSQL>
BEGIN
  :P1 := 100;
END;
/
gSQL> \print P1
P1
---
100
Anonymous PL block executed.
```

호환성

<assignment statement> 구문은 SQL 표준과 비교하여 다음과 같은 차이가 있다.

- SQL 표준의 assignment statement는 singleton variable assignment와 multiple variable assignment를 정의하지만, GOLDILOCKS는 singleton variable assignment 형식만 지원한다.
- SQL 표준의 assignment statement는 SET 키워드로 시작하지만, GOLDILOCKS는 SET 키워드를 사용하지 않는다.
- SQL 표준의 assignment statement는 target과 value 사이에 equal operator (=)를 사용하지만, GOLDILOCKS는 := 를 사용한다.

표 23-1 SQL 표준 호환성

Feature ID	설명	지원 여부
P002	Computational completeness	X
P006	Multiple assignment	X

참조

자세한 내용은 다음을 참조한다.

- Scalar Variable Declaration
- Record Variable Declaration
- COLLECTION Variable Declaration
- Cursor Variable Declaration

23.2 Basic LOOP Statement

기능

GOTO나 EXIT 등이 수행되어 LOOP를 종료하기 전까지 LOOP 내부의 statement들을 반복 수행한다.

구문

```
<basic loop statement> ::=
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- loop_name
 - <basic loop statement>의 label 이름이다.
 - Comment의 역할만 하기 때문에 실제 <basic loop statement>의 label 이름과 달라도 무방하다.

설명

Basic loop 구문은 LOOP 내부의 statement들을 반복 수행한다.

Basic loop 구문은 loop 계열 statement이므로 GOTO, EXIT, CONTINUE가 label로 지칭하는 target statement가 될 수 있다.

사용 예

```

DECLARE
  V1 INTEGER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    EXIT WHEN V1 > 2;
  END LOOP;
END;
/
V1 = 1
V1 = 2
Anonymous PL block executed.

```

호환성

〈basic loop statement〉 구문은 SQL 표준의 〈loop statement〉와 동일하다.

표 23-2 SQL 표준 호환성

Feature ID	설명	지원여부
P002	Computational completeness	O

참조

자세한 내용은 다음을 참조한다.

- CONTINUE Statement
- EXIT Statement
- GOTO Statement

23.3 Block (BEGIN .. END)

기능

새로운 scope를 생성하며, 변수, 커서, 타입과 exception 등을 정의한다.

구문

```

<PSM block> ::=
    [ DECLARE <declare item>... ] BEGIN <SQL procedure statement list> END
    ;
<declare item> ::=
    <variable declaration>
    | <explicit cursor declaration>
    | <cursor variable declaration>
    | <type declaration>
    | <exception decalration>
<executable statement list> ::=
    [ <label list> ] { <SQL procedure statement> ; }...
<label list> ::=
    { << identifier >> }...
<SQL procedure statement>
    <PSM Static SQL>
    | <PSM Dynamic SQL>
    | <PSM Control Statement>

```

사용 범위 및 접근 권한

PROCEDURE, FUNCTION 또는 anonymous block 내에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Variable declaration
 - Block 내에서 사용할 scalar/ record/ array 타입의 변수들을 선언한다
- Cursor variable declaration
 - Block 내에서 사용할 cursor 변수들을 선언한다.
- Cursor declaration
 - Block 내에서 사용할 cursor를 선언한다.
- Type declaration
 - Block 내에서 변수를 선언할 때 사용할 사용자 정의 타입들을 선언한다.
- Exception declaration
 - Exception 처리 구문에서 처리될 exception들을 정의한다.
- Static SQL
 - PSM에서 수행할 수 있는 Data Manipulation Language (DML)과 Data Control Language (DCL)를 가리킨다.
- Dynamic SQL
 - GOLDILOCKS PSM에서 지원하는 동적 질의 처리 관련 구문들을 가리킨다.
- PSM control statement
 - GOLDILOCKS PSM에서 지원하는 각종 flow control 구문들을 가리킨다.

설명

<psm block>은 PSM의 기본 구성 요소이다.

Block은 선언부 (declaration part)와 예외 처리부 (exception handling part)를 가질 수 있다.

Block은 중첩될 수 있고 중첩된 block은 새로운 하위 변수 scope를 가진다. 상위 block은 하위 block의 변수를 참조할 수 없다.

사용 예

```
gSQL> <<MAIN>>
DECLARE
  V1 INTEGER := 1;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
<<SUB1>>
DECLARE
  V1 VARCHAR(10) := 'ABC';
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  DBMS_OUTPUT.PUT_LINE( 'SUB1.V1 = ' || SUB1.V1 );
  DBMS_OUTPUT.PUT_LINE( 'MAIN.V1 = ' || MAIN.V1 );
END;
END;
/
V1 = 1
V1 = ABC
SUB1.V1 = ABC
MAIN.V1 = 1
Anonymous PL block executed.
```

호환성

SQL 표준의 <compound statement>는 새로운 savepoint를 지정하는 ATOMIC/ NOT ATOMIC 구문을 정의하였지만, GOLDILOCKS는 이를 지원하지 않는다.

표 23-3 SQL 표준 호환성

Feature ID	설명	비고
P002	Computational completeness	ATOMIC 구문을 지원하지 않는다.

참조

자세한 내용은 **Overview of PSM**을 참조한다.

23.4 CASE Statement

기능

주어진 여러 조건들 중에 TRUE를 반환하는 조건에 해당하는 statement list를 수행한다.

구문

```

<case statement> ::=
    <simple case statement>
    | <searched case statement>
    ;
<simple case statement> ::=
    CASE <case operand> <simple case statement when clause>...
    [ <case statement else clause> ]
    END CASE
<searched case statement> ::=
    CASE <searched case statement when clause>...
    [ <case statement else clause> ]
    END CASE
<simple case statement when clause> ::=
    WHEN <when operand>
        THEN <executable statement list>
<searched case statement when clause> ::=
    WHEN <search condition>
        THEN <executable statement list>

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Case operand
 - Scalar 값으로 평가될 수 있는 모든 표현식이다.
 - 단, subquery 구문은 포함할 수 없다.
- When operand
 - <case operand>와 같은지 비교할 표현식이다.
 - Subquery 구문은 포함할 수 없다
- Search condition
 - <searched case statement>에서 평가하여 TRUE인 경우에 실행할 조건식이다.
 - Subquery 구문은 포함할 수 없다.
- Executable statement list
 - PSM 내에서 수행할 수 있는 모든 statement 리스트이다.

설명

IF 구문과 유사하게 조건들을 평가하여 TRUE를 반환하는 WHEN 절의 statement들을 수행한다. 순서대로 먼저 나오는 조건식부터 평가하여 해당 조건식이 TRUE인 경우, 그 이후의 조건식들은 평가하지 않는다. 만일 조건식에 해당하는 경우가 존재하지 않고 ELSE 절이 기술되지 않은 경우에는 에러가 발생한다.

사용 예

Simple CASE 사용

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  SELECT 2 INTO V1 FROM DUAL;
  CASE V1 WHEN 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
          WHEN 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
          WHEN 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
          ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
  END CASE;
END;
/
Result = 2
Anonymous PL block executed.
```

Searched CASE 사용

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  SELECT 2 INTO V1 FROM DUAL;
  CASE WHEN V1 = 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
        WHEN V1 = 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
        WHEN V1 = 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
        ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
  END CASE;
END;
/
Result = 2
Anonymous PL block executed.
```

호환성

SQL 표준의 CASE 구문은 row 타입 (list 타입) value 간의 비교를 정의하였지만, GOLDILOCKS는 지원하지 않는다. SQL 표준의 CASE 구문은 <when operand>에 ','로 구분되는 여러 조건들을 리스트로 정의할 수 있지만, GODILOKS는 지원하지 않는다.

표 23-4 SQL 표준 호환성

Feature ID	설명	비고
P002	Computational completeness	P004, P008을 지원하지 않는다.
P004	Extended CASE statement	-
P008	Comma-separated predicates in simple CASE statement	-

23.5 CLOSE Statement

기능

Open 상태인 cursor를 닫는다.

구문

```
<close statement> ::=  
    CLOSE cursor_name  
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- cursor_name
 - Close 할 cursor의 이름이다.

설명

Open 상태인 cursor를 닫는다.

Close 된 상태인 cursor는 open 구문을 사용하여 다시 open 할 수 있다.

사용 예

```
gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    CURSOR C1 IS SELECT I1 FROM T1;
    V1 T1%ROWTYPE;
BEGIN
    OPEN C1;
    FETCH C1 INTO V1;
    CLOSE C1;
END;
/
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- [FETCH Statement](#)
- [OPEN Statement](#)

23.6 Collection Method Invocation

기능

Collection type의 변수를 탐색할 수 있는 method를 제공한다.

구문

```
<collection method> ::=
    variable_name . <method>
;
<method> ::=
    first ()
  | last ()
  | prior ( expression )
  | next ( expression )
  | count ()
  | exists ( expression )
  | delete ( expression )
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Collection type으로 선언된 변수에만 사용할 수 있다.
- FIRST, LAST, COUNT는 parameter를 가질 수 없다.
- PRIOR, NEXT, EXISTS, DELETE와 같이 대상을 지정해야 할 경우에는 parameter를 명시해야 한다.
- DELETE의 경우에는 PSM statement와 같이 동작하며 다른 변수로 결과를 반환할 수 없다. (Expression에 사용할 수 없다.)

설명

다음 표를 참조한다.

표 23-5 함수

함수명	기능	반환값	인자 필요여부
FIRST	가장 작은 key를 반환한다.	INDEX OF에 지정된 key type	X
LAST	가장 큰 key를 반환한다.	INDEX OF에 지정된 key type	X
PRIOR	입력된 key보다 작은 key를 반환한다.	INDEX OF에 지정된 key type	O
NEXT	입력된 key보다 큰 key를 반환한다.	INDEX OF에 지정된 key type	O
COUNT	저장된 개수를 반환한다.	INTEGER	X
DELETE	Key에 해당하는 값을 삭제한다.	N/A	O
EXISTS	Key의 존재 유무를 반환한다.	BOOLEAN	O

사용 예

```

DECLARE
TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(20);
v1 rec;
v2 VARCHAR(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE( '-----');
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '   || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'prior = '  || v1.prior('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'next = '   || v1.next('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'count = '  || v1.count());
  FOR I IN 1 .. 9
  LOOP
    v1('a' || to_char(i)) := 'a' || to_char(i);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( '-----');
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '   || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'count = '  || v1.count());
  DBMS_OUTPUT.PUT_LINE( '-----');
  DBMS_OUTPUT.PUT_LINE( 'Print all from first to last');
  v2 := v1.first;

```

```

WHILE v2 IS NOT NULL
LOOP
    DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
    v2 := v1.next(v2);
END LOOP;
DBMS_OUTPUT.PUT_LINE( '-----');
DBMS_OUTPUT.PUT_LINE( 'Print all from last to first');
v2 := v1.last;
WHILE v2 IS NOT NULL
LOOP
    DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
    v2 := v1.prior(v2);
END LOOP;
v1.delete(v1.first());
DBMS_OUTPUT.PUT_LINE('count = ' || v1.count() );
DBMS_OUTPUT.PUT_LINE('first = ' || v1.first() );
END;
/

-----

first =
last =
prior =
next =
count = 0

-----

first = a1
last = a9
count = 9

-----

Print all from first to last
Key= a1,Value=a1
Key= a2,Value=a2
Key= a3,Value=a3
Key= a4,Value=a4
Key= a5,Value=a5
Key= a6,Value=a6
Key= a7,Value=a7
Key= a8,Value=a8
Key= a9,Value=a9

-----

Print all from last to first

```



```
Key= a9,Value=a9  
Key= a8,Value=a8  
Key= a7,Value=a7  
Key= a6,Value=a6  
Key= a5,Value=a5  
Key= a4,Value=a4  
Key= a3,Value=a3  
Key= a2,Value=a2  
Key= a1,Value=a1  
count = 8  
first = a2  
Anonymous PL block executed.
```

참조

자세한 내용은 **COLLECTION Variable Declaration**을 참조한다.

23.7 COLLECTION Variable Declaration

기능

Collection 변수를 선언한다.

구문

```

<declare record variable> ::=
    variable_name <collectionType>
    ;

<Collection Type Definition> ::=
    TYPE <Type-Name> IS TABLE OF <Element-Type> INDEX BY <Index-Type>
    ;

<Element-Type> ::=
    Built-in SQL Data Type
    | User-Defined Type
    | %TYPE
    | %ROWTYPE

<Index-Type> ::=
    INTEGER
    | LONG
    | CHAR(n)
    | VARCHAR(n)

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Type-name
 - 사용자가 사용할 collection type의 이름을 지정한다.
- Element-type
 - Collection 변수에 저장될 element의 type을 지정한다.
- Index-type
 - Collection 변수에 저장된 key의 data type을 지정한다.

설명

Collection type을 선언한다.

사용 예

```
gSQL> DECLARE
TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(20);
v1 rec;
v2 VARCHAR(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '   || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'prior = ' || v1.prior('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'next = '   || v1.next('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'count = '  || v1.count());
  FOR I IN 1 .. 10
  LOOP
    v1('a' || i) := 'a' || i;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '   || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'count = '  || v1.count());
  DBMS_OUTPUT.PUT_LINE( 'Print all from first to last');
  v2 := v1.first;
  WHILE v2 IS NOT NULL
  LOOP
    DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
```

```

        v2 := v1.next(v2);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE( 'Print all from last to first');
    v2 := v1.last;
    WHILE v2 IS NOT NULL
    LOOP
        DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
        v2 := v1.prior(v2);
    END LOOP;
END;
/
first =
last =
prior =
next =
count = 0
first = a1
last = a9
count = 10
Print all from first to last
Key= a1,Value=a1
Key= a10,Value=a10
Key= a2,Value=a2
Key= a3,Value=a3
Key= a4,Value=a4
Key= a5,Value=a5
Key= a6,Value=a6
Key= a7,Value=a7
Key= a8,Value=a8
Key= a9,Value=a9
Print all from last to first
Key= a9,Value=a9
Key= a8,Value=a8
Key= a7,Value=a7
Key= a6,Value=a6
Key= a5,Value=a5
Key= a4,Value=a4
Key= a3,Value=a3
Key= a2,Value=a2
Key= a10,Value=a10
Key= a1,Value=a1

```

Anonymous PL block executed.

호환성

SQL 표준에서는 정의하지 않고 있다.

참조

자세한 내용은 **Collection Method Invocation**을 참조한다.

23.8 CONTINUE Statement

기능

현재 진행 중인 statement list의 수행을 중지하고, 상위 loop statement의 다음 iteration을 수행한다.

구문

```
<continue statement> ::=  
    CONTINUE [ label_name ] [ WHEN condition ]  
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

Target label을 가진 statement는 다음 loop 계열 statement 중 하나이어야 한다.

- basic loop statement
- for loop statement
- while statement
- forall statement

구문 규칙 및 파라미터

- Label name
 - identifier chain의 형식을 가질 수 있다.
- Condition
 - 명시된 경우, 해당 조건이 TRUE일 때만 loop statement로 복귀한다.

설명

현재 진행 중인 statement list의 수행을 중지하고 상위 loop statement로 복귀한다.

Label이 명시되면 해당 label 이름을 가진 상위 loop statement로 복귀한다.

Label이 명시되어 있지 않으면 가장 가까운 상위 loop statement로 복귀한다.

같은 label 이름을 가진 여러 개의 상위 statement들이 존재할 경우, 가장 가까운 statement가 선택된다.

현재 위치에서 visible한 (중첩된 scope 내에 존재하는) loop statement로만 복귀할 수 있다.

조건이 명시되면 해당 조건이 TRUE인 경우에만 복귀한다.

조건이 명시되지 않으면 무조건 복귀한다.

사용 예

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    IF V1 <= 2 THEN
      DBMS_OUTPUT.PUT_LINE( 'CONTINUE' );
      CONTINUE;
    ELSE
      EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE( 'END-OF-WHILE' );
  END LOOP AAA;
END;
/
V1 = 1
CONTINUE
V1 = 2
Anonymous PL block executed.
```

호환성

〈continue statement〉 구문은 SQL 표준의 〈iterate statement〉와 기능이 유사하다. 단, 〈iterate statement〉 구문은 WHEN condition 기능을 제공하지 않는다.

참조

자세한 내용은 다음을 참조한다.

- EXIT Statement
- GOTO Statement

23.9 Cursor FOR LOOP Statement

기능

PSM에서 사용자가 선언한 cursor나 query에 의해 생성된 result의 row 개수만큼 loop를 수행한다.

구문

```

<Cursor For Loop statement> ::=
    FOR <Variable_Name> IN <Cursor>
    LOOP
        { <SQL procedure statement> ; }...
    END LOOP [ Label_Name ]
    ;
<Cursor> ::=
    < ( Implicit_Cursor_Query ) >
    | < Explicit_Cursor_Name > [ ( [ <actual param> ] ) ]

<Implicit_Cursor_Query> ::=
    SELECT statement
    | SELECT_FOR_UPDATE statement
    | INSERT_RETURNING_QUERY statement
    | UPDATE_RETURNING_QUERY statement
    | DELETE_RETURNING_QUERY statement
<actual param> ::=
    ( expression [ , expression ] .. )
  
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PSM 내의 body에서만 사용할 수 있다.

구문 규칙 및 파라미터

For~Loop 내에 선언된 변수는 해당 loop scope 내에서만 유효하다. (해당 Cursor For Loop Block Scope 밖에서는 해당 변수를 참조할 수 없다.)

Cursor Name을 사용하는 경우

Cursor name을 사용하여 LOOP를 수행할 경우 cursor가 미리 선언되어 있어야 한다.
Actual param에 대한 자세한 내용은 **OPEN Statement**를 참조한다.

Cursor Query를 사용하는 경우

Select나 returning query와 같이 GOLDILOCKS 내부적으로 implicit cursor로 처리되는 질의만 수행할 수 있다.

설명

Cursor가 생성한 결과 개수만큼 loop를 돌면서 loop 내의 PSM statements를 수행한다.
LOOP 중에 cursor가 invalid한 상태가 (예: closed) 되면 더 이상 loop를 수행하지 않고 오류로 처리한다.
Explicit cursor name을 명시할 경우, 해당 cursor가 already opened이면 오류로 처리한다.
FOR LOOP 절에 명시된 cursor의 결과를 반환받는 변수는 자동으로 생성된다. (Cursor의 실행 결과에 의해 반환될 result set의 row type으로 생성된다.)
다만, 사용자의 cursor query 결과 중 특정 테이블의 column이 아닌 select target expression에 대해 alias 등을 지정하지 않을 경우, 오류가 발생할 수 있다.

사용 예

Explicit Cursor 사용

```
DECLARE
CURSOR c1 IS SELECT * FROM T1;
BEGIN
  FOR rec IN C1
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'RowCount=' || c1%rowcount || ',C1=' || rec.c1 || ', C2=' || rec.c
2);
  END LOOP;
```

```

END;
/
RowCount=1,C1=1, C2=1
RowCount=2,C1=2, C2=2
RowCount=3,C1=3, C2=3
RowCount=4,C1=4, C2=4
RowCount=5,C1=5, C2=5
RowCount=6,C1=6, C2=6
RowCount=7,C1=7, C2=7
RowCount=8,C1=8, C2=8
RowCount=9,C1=9, C2=9
RowCount=10,C1=10, C2=10
Anonymous PL block executed.

```

Cursor Query 사용

```

BEGIN
  FOR rec IN (select * from t1)
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'RowCount=' || sql%rowcount || ',C1=' || rec.c1 || ', C2=' || rec.
c2);
  END LOOP;
END;
/
C1=1, C2=1
C1=2, C2=2
C1=3, C2=3
C1=4, C2=4
C1=5, C2=5
C1=6, C2=6
C1=7, C2=7
C1=8, C2=8
C1=9, C2=9
C1=10, C2=10
Anonymous PL block executed.

```

참조

자세한 내용은 다음을 참조한다.

- **Explicit Cursor Declaration and Definition**
- **GOTO Statement**
- **EXIT Statement**

23.10 Cursor Variable Declaration

기능

PSM의 DECLARE section에서 cursor variable을 선언한다.

구문

```

<cursor variable declaration> ::=
    variable_name <type>
    ;
<cursor type definition> ::=
    TYPE <type_name> IS REF CURSOR [ RETURN <return type> ]
<return type> ::=
    <table_name | view_name | cursor_name | cursor_variable > % ROWTYPE
    | <record_variable_name> % TYPE
    | <record_type_name>

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PSM의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

Cursor 변수의 초기값 지정이나 assign은 cursor 변수 사이에서만 가능하다.

설명

Cursor variable은 특정 cursor에 종속되지 않는 cursor를 가리키는 일종의 pointer 역할을 한다.

사용 예

```
DECLARE
TYPE rec IS RECORD (V1 VARCHAR(20), V2 VARCHAR(20));
TYPE cv IS REF CURSOR RETURN rec;
BEGIN
    NULL;
END;
/
Anonymous PL block executed.
```

참조

자세한 내용은 다음을 참조한다.

- **OPEN Statement**
- **FETCH Statement**
- **CLOSE Statement**

23.11 DELETE Statement Extension

기능

PSM의 record type 변수를 이용하여 RETURNING INTO 절에 결과를 저장할 수 있다.

구문

```

<PSM delete statement extension: searched> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        [ <returning into clause> ]
    ;

<delete statement: positioned> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        WHERE CURRENT OF cursor_name
    ;

<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]

<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>

<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]

<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

<returning into clause> ::=
    { RETURN | RETURNING } { * | { <value expression> [ [AS] alias_name ] } [, ...] } INTO
    variable_name [, ...]
  
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PSM의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

Returning Into를 통해 반환받을 변수의 타입이 record-type인 경우 다른 변수 타입과 섞어서 사용할 수 없다.

설명

PSM의 record type 변수를 이용하여 RETURNING INTO 절에 결과를 저장할 수 있다.

사용 예

```
gSQL> CREATE TABLE T1( C1 VARCHAR(20), C2 VARCHAR(20));
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC'), ('CCC', 'DDD');
3 rows created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    rec t1%ROWTYPE;
BEGIN
    DELETE FROM T1 WHERE C1 = 'AAA' RETURNING * INTO rec;
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT=' || SQL%ROWCOUNT);
    DBMS_OUTPUT.PUT_LINE('rec.c1=' || rec.c1 || ', rec.c2=' || rec.c2);
END;
/
SQL%ROWCOUNT=1
rec.c1=AAA, rec.c2=BBB
Anonymous PL block executed.
```


참조

자세한 내용은 **데이터 삭제**를 참조한다.

23.12 EXCEPTION_INIT Pragma

기능

PL block 내의 internally defined exception을 정의한다.

구문

```
< PRAGMA EXCEPTION_INIT > ::=
    PRAGMA EXCEPTION_INIT ( <Exception-Name>, <Internal-ErrorCode> )
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PSM의 declaration 영역에서만 사용할 수 있다

구문 규칙 및 파라미터

Predefined exception은 argument로 사용되는 exception name에 사용할 수 없다. (predefined exception name은 선언할 수 없다.)

동일한 PL BLOCK DECLARE 절에 argument로 사용되는 exception name이 반드시 미리 선언되어야 한다. (다른 BLOCK의 exception name 선언을 참조할 수 없다.)

<Internal-ErrorCode>는 DB SYSTEM 내에 존재하는 내부 error code이어야 한다. (SUCCESS 코드는 설정할 수 없다.)

설명

사용자가 DB SYSTEM의 error code에 대응하는 exception name을 명시적으로 선언한다.

사용 예

```
gSQL> DECLARE
user_exception_1 EXCEPTION;
PRAGMA EXCEPTION_INIT( user_exception_1, -17001);
BEGIN
  RAISE USER_EXCEPTION_1;
  EXCEPTION WHEN user_exception_1 THEN DBMS_OUTPUT.PUT_LINE('User Exception_1');
END;
/
User Exception_1
Anonymous PL block executed.
```

호환성

Error code는 각 벤더마다 다르기 때문에 서로 호환되지 않는다.

참조

자세한 내용은 다음을 참조한다.

- Exception Declaration
- Exception Handler

23.13 Exception Declaration

기능

PL block 내의 exception name을 선언한다.

구문

```
< Exception-Declaration Statement > ::=
    <Exception-Name> EXCEPTION
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PSM의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

Predefined exception name은 선언할 수 없다.
동일한 SCOPE의 DECLARE 절에 중복으로 선언할 수 없다.

설명

사용자가 명시적으로 exception을 선언한다.

사용 예

```
DECLARE
    user_exception_1 EXCEPTION;
```

```

user_exception_2 EXCEPTION;
user_exception_3 EXCEPTION;
BEGIN
    RAISE USER_EXCEPTION_2;
    EXCEPTION WHEN user_exception_1 THEN DBMS_OUTPUT.PUT_LINE('User Exception_1');
            WHEN user_exception_2 THEN DBMS_OUTPUT.PUT_LINE('User Exception_2');
            WHEN user_exception_3 THEN DBMS_OUTPUT.PUT_LINE('User Exception_3');
END;
/
User Exception_2
Anonymous PL block executed.

```

호환성

표준 SQL의 exception 선언은 다음과 같지만 GOLDDILOCKS는 위와 같은 구문을 지원한다.

```

<condition declaration> ::=
DECLARE <condition name> CONDITION [ FOR <sqlstate value> ]

```

참조

자세한 내용은 다음을 참조한다.

- Exception Declaration
- EXCEPTION_INIT Pragma

23.14 Exception Handler

기능

PL/ SQL을 수행하는 중에 발생한 DB SYSTEM 상의 오류로 인한 암묵적 exception이나 사용자가 명시적으로 발생 시킨 exception에 대해 정의된 동작을 수행한다.

구문

```

< Exception Handler Statement > ::=
    EXCEPTION < Exception_When_List >
    ;
< Exception_When_List > ::=
    WHEN < Exception_Name_List > THEN <executable statement list> [ WHEN OTHERS THEN <
executable statement list> ]
< Exception_Name_List > ::=
    <Exception_Name> [ { OR <Exception_Name> }... ]

```

사용 범위 및 접근 권한

PL block 내에서 사용할 수 있다.

구문 규칙 및 파라미터

Predefined exception인 OTHERS는 OR을 사용하여 다른 exception name과 함께 기술할 수 없다.

Predefined exception인 OTHERS는 exception handler에 중복으로 기술할 수 없으며 가장 마지막에 기술하여야 한다.

설명

Exception 유형

Type	Definer	Has error	Has name	Raise implicitly	Raise explicitly
Predefined	System	Yes	Yes	Yes	Optionally
User-defined	User	If user assign	If user assign	No	Yes

Predefined exception은 GOLDBLOCKS에서 미리 지정한 exception name과 error code를 갖는다. 그 외의 exception은 GOLDBLOCKS 내부의 오류 코드 이름을 사용자가 predefined exception name과 다르게 설정하는 경우 (internally defined)와 별도의 error code를 지정하지 않고 exception name만 선언하는 경우 (user-defined)로 나뉘어진다.

Predefined Exception

표 23-6 Predefined exception 유형

이름	설명
CASE_NOT_FOUND	CASE WHEN의 모든 조건에 맞지 않거나 ELSE 절이 정의되지 않았다.
DUP_VAL_ON_INDEX	INDEX duplicated 오류가 발생하였다.
INVALID_CURSOR	Cursor 상태가 올바르지 않다.
INVALID_NUMBER	숫자로 변환할 수 없다.
NO_DATA_FOUND	SELECT 문이 0 건의 데이터를 반환한다.
ROWTYPE_MISMATCH	두 개의 RowType 변수의 필드 타입이 서로 다르다.
TOO_MANY_ROWS	두 건 이상의 row를 반환한다.
VALUE_ERROR	Type mismatch, invalid casting 같은 error이다.
ZERO_DIVIDE	0으로 나누기를 시도한다.
OTHERS	Predefined에 정의되지 않은 오류를 포함한다.

사용 예

```
gSQL> DECLARE
V1 INTEGER := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Step1');
  V1 := 1 / 0;
  DBMS_OUTPUT.PUT_LINE('Step2');
  EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE( 'Exception V1=' || V1);
```

```
END;  
/  
Step1  
Exception V1=0  
Anonymous PL block executed.
```

호환성

SQL 표준 문법을 지원하지 않는다.

참조

자세한 내용은 다음을 참조한다.

- EXCEPTION_INIT Pragma
- Exception Declaration

23.15 EXECUTE IMMEDIATE Statement

기능

PSM 내에서 dynamic SQL을 실행한다.

구문

```

<EXECUTE IMMEDIATE statement> ::=
    EXECUTE IMMEDIATE <dynamic-sql> [ <binding-parameters> ]
    ;
<dynamic-sql> ::=
    single_quote_string
    | psm_variable
<binding-parameters> ::=
    <into-clause>
    | <using-clause>
    | <returning-into-clause>
    | <into-clause> <using-clause>
    | <using-clause> <returning-into-clause>
<into-clause> ::=
    INTO psm_variable [ {, psm_variable} ... ]
<using-clause> ::=
    USING [ <Bind-Type> ] <expression> [ {, [ <Bind-Type> ] <expression>} ... ]
<Bind-Type> ::=
    IN
    | OUT
    | IN OUT
<returning-into-clause> ::=
    RETURNING INTO psm_variable [ {, psm_variable} ... ]

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
 PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

Dynamic SQL

- `single_quote_string`: Single quote (')로 묶인 SQL 문장이다. (`double_quote_string`은 PSM 내에서 변수로 인식된다.)
- `PSM-variable`: PSM에서 사용되는 변수 내에 저장된 SQL 문장이다.

다음은 수행하려는 SQL 문장 내에 quote를 사용하여 데이터를 표현하는 예이다.

```
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( 'Tom' ) '; -- Tom
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( 'Tom''House' ) '; -- Tom'House
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''''Tom'' ) '; -- 'Tom
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( CHR(39) || 'TOM' ) '; -- 'Tom
```

Dynamic SQL 내의 사용자가 변수를 입력할 부분에는 ? 또는 :V1과 같은 표기 (marker)를 사용한다.

Dynamic SQL 내에 기술되는 SQL 문장은 유효한 문장이어야 한다.

INTO Clause

Dynamic SQL 수행 결과가 존재하고 marker를 통해 binding된 경우가 아닌 SQL 문의 처리 결과를 반환 받는 경우이다. (내부적으로 implicit cursor fetch 형태이다.)

구문은 다음과 같다.

```
EXECUTE IMMEDIATE 'SELECT ... FROM .. WHERE ...';
EXECUTE IMMEDIATE 'INSERT ... RETURNING ...';
EXECUTE IMMEDIATE 'UPDATE ... RETURNING ...';
EXECUTE IMMEDIATE 'DELETE ... RETURNING ...';
```

USING Clause

Dynamic SQL에 입력 변수를 사용할 경우 그 개수만큼 USING (IN 생략 가능)절에 변수나 표현식을 나열한다.

Dynamic SQL 수행 결과가 존재할 경우 결과 column의 개수만큼 USING OUT 절에 변수를 나열한다. (결과를 INTO 절로 반환하는 경우)

USING OUT을 이용하여 다음 구문에 결과를 반환할 수 있다.

```
EXECUTE IMMEDIATE 'SELECT x, y, z INTO :v1, :v2, :v3 ...';
EXECUTE IMMEDIATE 'INSERT INTO ... RETURNING C1, C2 INTO :V1, :V2';
EXECUTE IMMEDIATE 'UPDATE T1 SET .. RETURNING C1, C2 INTO :V1, :V2';
EXECUTE IMMEDIATE 'DELETE FROM ... RETURNING C1, C2 INTO :V1, :V2';
```

RETURNING Clause

INSERT/ UPDATE/ DELETE RETURNING INTO 구문이 dynamic SQL로 사용된 경우, GOLDILOCKS는 USING 절에 기술된 변수를 OUT mode로 binding하여 결과를 반환받을 수 있다.

다른 DBMS와의 호환을 위해 RETURNING INTO 절로도 동일한 결과를 반환받을 수 있다.

기타 규칙

- DDL/ DCL은 어떤 BIND 절 (INTO, USING, RETURNING clause)도 사용할 수 없다.
- INTO 절과 RETURNING INTO 절에는 OUT으로 쓰이기 때문에 별도의 bind type을 지정할 수 없으며 동시에 같이 사용할 수도 없다.
- IN BIND type은 scalar type 변수만 사용할 수 있다.
- OUT BIND type은 record type 변수를 사용할 수 있다. 하지만 scalar나 record 타입을 섞어서 동시에 나열할 수는 없다.
- INTO 절과 USING OUT 또는 RETURNING INTO를 통해 결과를 나누어 받을 수 없다.

설명

- Dynamic SQL이 결과를 INTO 절에 기술된 변수로 반환하는 경우
 - 두 건 이상의 결과를 반환하는 질의는 TOO_MANY_ROWS exception을 발생시킨다.
 - 결과가 0 건인 경우 NO_DATA_FOUND exception을 발생시킨다.
- Dynamic SQL이 결과를 USING이나 RETURNING 절에 기술된 변수로 반환하는 경우
 - ARRAY가 아닌 변수가 두 건 이상 반환되면 TOO_MANY_ROWS exception이 발생한다.
 - 결과가 0 건이면 오류가 발생하지 않는다.
- DDL/ DCL이 implicit cursor SQL%Attribute 변수들을 수행한 결과는 다음과 같다.
 - SQL%ROWCOUNT = 0
 - SQL%ISOPEN = FALSE
 - SQL%FOUND = FALSE
 - SQL%NOTFOUND = TRUE
- 그 외의 dynamic SQL은 해당 SQL 문의 처리 결과에 맞는 SQL%Attribute 값을 저장한다.

사용 예

```
gSQL> DECLARE
V1 INTEGER;
V2 VARCHAR(20);
BEGIN
  V1 := 1;
```

```

V2 := 'abcdef';
DBMS_OUTPUT.PUT_LINE('#INSERT');
EXECUTE IMMEDIATE 'insert into t1 values (:a1, :a2)' USING v1, v2;
EXECUTE IMMEDIATE 'select c1, c2 from t1 where c1 = 1' INTO v1, v2;
DBMS_OUTPUT.PUT_LINE('C1='|| v1 || ', C2=' || v2);
V1 := 1;
V2 := 'xyz';
DBMS_OUTPUT.PUT_LINE('#UPDATE');
EXECUTE IMMEDIATE 'update t1 set c2 = :a1 where c1 = :a2' USING V2, V1;
V1 := 1;
V2 := '';
DBMS_OUTPUT.PUT_LINE('#SELECT');
EXECUTE IMMEDIATE 'select c1, c2 from t1 where c1 = :a1' INTO v1, v2 USING v1;
DBMS_OUTPUT.PUT_LINE('C1='|| v1 || ', C2=' || v2);
V1 := 1;
V2 := '';
DBMS_OUTPUT.PUT_LINE('#DELETE');
EXECUTE IMMEDIATE 'delete from t1 where c1 = :a1' USING v1;
END;
/
#INSERT
C1=1, C2=abcdef
#UPDATE
#SELECT
C1=1, C2=xyz
#DELETE
Anonymous PL block executed.

```

23.16 EXIT Statement

기능

상위 loop statement들 중에서 주어진 label을 가진 loop statement를 탈출하여 그 다음 statement를 수행한다.

구문

```
<exit statement> ::=
    EXIT [ label_name ] [ WHEN condition ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Label name
 - 탈출할 loop statement의 label 이름이다.
 - Identifier chain 형식을 가질 수 있다.
- Condition
 - 조건을 명시할 경우, 해당 조건이 TRUE일 경우에만 loop statement를 탈출한다.

설명

- 현재 진행중인 statement list 수행을 중단하고 상위 loop statement를 탈출한다.
- Target label을 가진 statement는 다음 loop 계열 statement 중 하나이어야 한다.
 - basic loop statement
 - for loop statement
 - while statement

- forall statement
- Label이 명시된 경우, 해당 label 명을 가진 상위 loop statement를 탈출한다.
- Label이 명시되지 않은 경우, 가장 가까운 상위 loop statement를 탈출한다.
- 같은 label 명을 가지는 여러 개의 상위 statement들이 존재할 경우, 가장 가까운 statement를 선택한다.
- 현재 위치에서 visible 한 (중첩된 scope내에 존재하는) loop statement만 탈출할 수 있다.
- 조건 (condition)이 명시된 경우, 해당 조건이 TRUE인 경우에만 탈출한다.
- 조건이 명시되지 않은 경우, 무조건 탈출한다.

사용 예

```
gSQL> DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    EXIT;
    V1 := V1 + 1;
  END LOOP AAA;
END;
/
V1 = 1
Anonymous PL block executed.
```

호환성

SQL 표준에는 존재하지 않는다.

23.17 Explicit Cursor Attribute

기능

PSM에서 정의된 cursor의 상태값을 반환한다.

구문

```
<Explicit cursor attribute> ::=  
    cursor_name '%' { ISOPEN | FOUND | NOTFOUND | ROWCOUNT }
```

사용 범위 및 접근 권한

PSM 내의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Cursor_name
 - 상태값을 알고 싶은 커서의 이름이다.

설명

- 주어진 커서의 상태값을 반환한다.
 - ISOPEN: 현재 커서가 OPEN 상태인지 여부이다.
 - FOUND: 최근 fetch에 의해 데이터가 반환되었는지 여부이다.
 - NOTFOUND: FOUND의 반대이다.
 - ROWCOUNT: 커서가 최근에 OPEN 된 후 fetch 한 record의 개수이다.
- 커서 상태에 따라 다음과 같은 값들을 반환한다.

표 23-7 수행 시점에 따른 결과

Attribute 이름	OPEN 전	OPEN 후	FETCH 후	CLOSE 후
ISOPEN	FALSE	TRUE	TRUE	FALSE

Attribute 이름	OPEN 전	OPEN 후	FETCH 후	CLOSE 후
FOUND	NULL	NULL	TRUE/ FALSE	NULL
NOTFOUND	NULL	NULL	TRUE/ FALSE	NULL
ROWCOUNT	NULL	NULL	N (개수)	NULL

사용 예

```

gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    CURSOR C1 IS SELECT * FROM T1;
    V1 INTEGER := 0;
    V2 INTEGER := 0;
    TOTAL INTEGER := 0;
BEGIN
    FOR I IN 1..100 LOOP
        INSERT INTO T1 VALUES( I );
    END LOOP;
    COMMIT;
    IF NOT C1%ISOPEN THEN
        OPEN C1;
    END IF;
    LOOP
        FETCH C1 INTO V1;
        EXIT WHEN C1%NOTFOUND;
        TOTAL := TOTAL + V1;
        V2 := V1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE( 'COUNT = ' || C1%ROWCOUNT || ' TOTAL = ' || TOTAL );
    CLOSE C1;
END;
/
COUNT = 100 TOTAL = 5050
Anonymous PL block executed.

```


호환성

SQL 표준에 정의되어 있지 않다.

23.18 Explicit Cursor Declaration and Definition

기능

PSM의 DECLARE section에서 커서를 선언한다.

구문

```

<cursor declaration> ::=
    CURSOR cursor_name [ <cursor param spec> ] RETURN rowtype
    ;
<cursor param spec> ::=
    ( <cursor param decl> [ , <cursor param decl> ] .. )
<cursor param decl> ::=
    param_name [ IN ] datatype [ { '=' | DEFAULT } expression ]
<cursor definition> ::=
    CURSOR cursor_name [ <cursor param spec> ] [ RETURN rowtype ]
    IS select_statement
    ;

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다

구문 규칙 및 파라미터

Cursor Name

선언할 커서의 이름이다.

커서 이름의 길이는 128 바이트보다 작아야 한다.

해당 scope 내에서 고유한 이름이어야 한다.

RowType

커서의 레코드 타입을 정의한다.

커서를 정의할 때 명시된 select target들의 개수가 같아야 하며, 데이터 타입이 호환되어야 한다.

Rowtype을 지정하지 않을 경우, cursor를 정의할 때 기술된 select_statement의 SELECT target에 적합한 rowtype이 자동으로 지정된다.

Param Name

특정 커서 내에서 parameter를 구별하는 이름이다.

해당 커서 내에서 고유한 이름이어야 한다.

만일 참조 가능한 scope 내의 다른 변수와 이름이 같을 경우, 해당 커서의 parameter를 우선적으로 참조한다.

DataType

해당 parameter의 데이터 타입을 지정한다.

GOLDILOCKS에서 제공하는 모든 built-in 타입과 PSM 내에 정의된 타입을 사용할 수 있다.

단, built-in 타입에는 범위를 제한하는 구문 (precision/ scale)을 지정할 수 없고 내부적으로 해당 데이터 타입의 최대 범위로 지정된다.

Select Statement

커서가 수행할 SELECT 또는 SELECT ... FOR UPDATE 구문을 지정한다.

SELECT ... INTO 구문은 사용할 수 없다.

설명

- <explicit cursor declaration> 구문은 커서를 선언하거나 정의한다.
 - cursor declaration: 커서의 이름과 포맷만 선언한다.
 - cursor definition: 커서의 이름, 포맷 및 수행할 SELECT 구문까지 상세하게 정의한다.
- explicit cursor는 declaration 후에 definition 하여 사용하거나, declaration 없이 바로 definition 하여 사용할 수 있다.
- declaration 후 definition 하여 사용할 경우에는 커서의 이름, parameter spec, 레코드 타입 정의가 정확하게 일치해야 한다.
- explicit cursor의 declaration과 definition은 같은 block 내에 존재해야 한다.
- explicit cursor는 해당 cursor가 define 된 block에 진입할 때 생성되며 해당 block에서 나갈 때 자동으로 CLOSE 되고 삭제된다.
- 특정 시점에 생성할 수 있는 explicit cursor의 최대 개수는 'MAXIMUM_NAMED_CURSOR_COUNT' 프로퍼티에 의해 제한된다.
- explicit cursor가 수행하는 SELECT 구문에는 다음 변수들을 사용할 수 있다.

- 해당 cursor의 parameter
- declare 시점의 scope에서 참조할 수 있는 모든 PSM 변수 (open 시점의 scope가 아님)
- 외부 bind parameter (Anonymous PL block의 경우)
- SELECT 구문을 수행하는 explicit cursor는 다음 속성을 갖는다.
 - IN_SENSITIVE (다른 트랜잭션에 의한 변경에 영향받지 않는다.)
 - NON_SCROLLABLE (이전 record를 다시 fetch 할 수 없다.)
 - READ_ONLY (read 연산만 가능하다)
 - WITH-HOLD (COMMIT/ ROLLBACK이 수행되어도 cursor가 자동으로 닫히지 않는다.)
- SELECT ... FOR UPDATE 구문을 수행하는 explicit cursor는 다음과 같은 속성을 갖는다.
 - IN_SENSITIVE (다른 트랜잭션에 의한 변경에 영향받지 않는다.)
 - NON_SCROLLABLE (이전 record를 다시 fetch 할 수 없다.)
 - UPDATABLE (커서의 위치를 통해 UPDATE/ DELETE 할 수 있다.)
 - WITHOUT-HOLD (COMMIT/ ROLLBACK이 수행되면 cursor가 자동으로 닫힌다.)

사용 예

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I2 VARCHAR(10) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO T1 VALUES( 1, 'AAA' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 2, 'BBB' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 3, 'CCC' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    CURSOR C1( A1 INTEGER, A2 VARCHAR ) RETURN T1%ROWTYPE IS SELECT * FROM T1 WHERE I1 = A1 AND
I2 = A2;
    V1 T1%ROWTYPE;
BEGIN
    OPEN C1( 2, 'BBB' );
    FETCH C1 INTO V1;
    DBMS_OUTPUT.PUT_LINE( 'V1.I1 = ' || V1.I1 || ' V1.I2 = ' || V1.I2 );
    CLOSE C1;
END;
/
```

```
V1.I1 = 2 V1.I2 = BBB
```

```
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- OPEN FOR Statement
- FETCH Statement
- CLOSE Statement

23.19 FETCH Statement

기능

OPEN 된 cursor의 레코드를 한 건 가져온다.

구문

```
<fetch statement> ::=
    FETCH cursor_name <into clause>
    ;
<into clause> ::=
    INTO { variable [ , variable ] .. | record }
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Cursor name
 - Fetch 할 cursor name이다.
- Variable
 - Fetch한 결과 중에 한 개의 column 값을 저장할 scalar 타입의 변수 또는 bind parameter이다.
- Record
 - Fetch한 결과인 한 개의 레코드 전체를 저장할 record 타입의 변수이다.

설명

Open 된 커서로부터 레코드 한 개를 fetch하여 INTO 절에 명시된 변수로 값을 복사한다.
만일 커서가 declaration만 되어 있고 definition 되어 있지 않으면 에러가 발생한다.
해당 커서는 open 된 상태여야 한다.

INTO 절에 주어진 변수 타입은 fetch된 레코드 결과의 데이터 타입과 서로 호환 가능해야 한다.
INTO 절에 주어진 변수들의 개수는 커서의 SELECT target 개수와 같아야 한다.
단, INTO 절에 주어진 변수가 record type일 경우에는 한 개만 명시해야 한다.
그리고 해당 record 변수의 field 개수는 SELECT target의 개수와 같아야 한다.

Fetch 할 레코드가 없는 상태에서 fetch가 호출되었을 경우, INTO 절의 target 변수 값은 변하지 않는다.

사용 예

```
gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    CURSOR C1 IS SELECT * FROM T1;
    V1 INTEGER := 0;
    V2 INTEGER := 0;
    CNT INTEGER := 0;
    TOTAL INTEGER := 0;
BEGIN
    FOR I IN 1..100 LOOP
        INSERT INTO T1 VALUES( I );
    END LOOP;
    COMMIT;
    OPEN C1;
    LOOP
        FETCH C1 INTO V1;
        CNT := CNT + 1;
        TOTAL := TOTAL + V1;
        V2 := V1;
        EXIT WHEN V1 = 100;
    END LOOP;
    CLOSE C1;
    DBMS_OUTPUT.PUT_LINE( 'CNT = ' || CNT || ' TOTAL = ' || TOTAL );
END;
/
```

```
CNT = 100 TOTAL = 5050
```

```
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- **OPEN Statement**
- **CLOSE Statement**

23.20 FOR LOOP Statement

기능

Index 변수가 주어진 값 범위를 가지는 동안 index 변수를 1씩 증가시키거나 감소시키면서 (REVERSE) 내부의 statement들을 수행한다.

구문

```
<for loop statement> ::=
    FOR index_variable_name IN [ REVERSE ] start_value .. last_value
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Index variable name
 - FOR 구문에서 index로 사용할 변수의 이름이며 내부적으로 NATIVE_BIGINT 타입의 변수가 사용된다.
- Start value
 - Index 변수의 초기 값으로써 정수형이어야 한다. 만일 부동 소수점을 가진 숫자를 사용하면 타입을 변환하는 중에 소수점 이하는 버려진다.
- Last value
 - Index 변수의 최종 값으로써 정수형이어야 한다. 만일 부동 소수점을 가진 숫자를 사용하면 타입을 변환하는 중에 소수점 이하는 버려진다.

설명

for loop 구문은 index 변수의 값을 start_value로부터 last_value까지 증가시키거나 감소시키면서 내부의 statement list를 수행한다.

REVERSE를 명시한 경우에는 start_value로부터 1씩 감소하게 되며 index 변수의 값이 last_value보다 작아지게 되면 for loop statement의 실행을 종료한다.

REVERSE를 명시하지 않으면 index 변수는 start_value 값부터 1씩 증가하고, index 변수의 값이 last_value보다 커지면 for loop statement의 실행을 종료한다.

REVERSE를 명시하지 않은 상태에서 start_value가 last_value보다 크거나 REVERSE를 명시한 상태에서 start_value가 last_value보다 작으면 내부의 statement list는 수행되지 않는다.

사용 예

```
gSQL> BEGIN
  FOR I IN 0 .. 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/
2 3 4 5 6 I = 0
I = 1
I = 2
I = 3
I = 4
I = 5
I = 6
I = 7
I = 8
I = 9
I = 10
```

Anonymous PL block executed.

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- `CONTINUE Statement`
- `EXIT Statement`
- `GOTO Statement`

23.21 Function Declaration and Definition

기능

Nested function을 정의한다.

구문

```
<nested function statement> ::=
    FUNCTION func_name [ ( { param_name [IN|OUT|INOUT] datatype [ { := | DEFAULT } init_expr ]
    } [, ...] ) ]
        RETURN datatype
        { IS | AS } <item_declaration> BEGIN <pl_stmt_list> END
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

func_name

생성할 function의 이름으로써 스키마 내에서 고유한 이름이어야 한다.

Function 이름의 길이는 128 바이트보다 작아야 한다.

Param Name

Function이 사용할 인자의 이름을 정의한다.

각 인자의 이름은 function 내에서 고유한 이름이어야 한다.

Bind Type

각 인자의 bind type을 설정한다.
표기하지 않을 경우 기본 타입은 IN 이다.

Item Declaration

Function 내부에서 사용될 로컬 변수등의 item을 선언한다.
PL block에서 선언될 수 있는 모든 item들을 선언할 수 있다.

PL Stmt List

Function의 body 부분으로써 수행할 PL statement들을 나열한다.

설명

Nested function은 해당 procedure 내에서만 호출할 수 있는 subprogram이다.
그 외의 사용 방법은 schema-level function과 동일하다.

사용 예

```
gSQL> DECLARE
  V1 INTEGER := 0;
  FUNCTION FUNC1( A1 INTEGER )
    RETURN INTEGER
  IS
  BEGIN
    RETURN A1 * 10;
  END;
BEGIN
  V1 := FUNC1( 10 );
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 100
```

Anonymous PL block executed.

호환성

Schema-level function과 동일하다.

참조

자세한 내용은 **CREATE FUNCTION**을 참조한다.

23.22 GOTO Statement

기능

현재 위치에서 접근할 수 있는 statement들 중에 주어진 label을 가진 가장 가까운 statement로 jump를 시도한다.

구문

```
<goto statement> ::=
    GOTO label_name
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Label_name
 - Jump를 시도할 statement의 label 이름이다.
 - Identifier chain 형식을 가질 수 있다.

설명

해당 label 이름을 가진 statement로 jump하여 수행을 시작한다.

여러 개의 후보 statement들이 존재할 경우, 가장 가까운 statement로 jump한다.

현재 위치에서 visible 한 (중첩된 scope 내에 존재하는) statement로만 jump할 수 있다.

Forward jump와 backward jump 모두 가능하다.

사용 예

```
gSQL> DECLARE
      V1 INTEGER := 0;
BEGIN
  <<LABEL1>>
  IF V1 > 0 THEN
    GOTO LABEL2;
  END IF;
  V1 := V1 + 1;
  DBMS_OUTPUT.PUT_LINE('a');
  GOTO LABEL1;
  DBMS_OUTPUT.PUT_LINE('b');
  <<LABEL2>>
  DBMS_OUTPUT.PUT_LINE('c');
END;
/
a
c
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- **EXIT Statement**
- **CONTINUE Statement**

23.23 IF Statement

기능

주어진 여러 조건들 중에 TRUE를 반환하는 조건에 해당하는 statement list를 수행한다.

구문

```

<if statement> ::=
    IF <search condition> <if statement then clause>
    [ <if statement elsif clause> ]
    [ <if statement else clause> ]
    END IF
    ;
<if statement then clause> ::=
    THEN <executable statement list>
<if statement elsif clause> ::=
    ELSIF <search condition> THEN <executable statement list>
<if statement else clause> ::=
    ELSE <executable statement list>

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Search condition
 - boolean 타입으로 최종 평가될 수 있는 표현식이다.
- Executable statement list
 - GOLDBLOCKS PSM에서 지원하는 모든 statement 들의 list 이다.

설명

CASE 구문과 유사하게 조건들을 평가하여 TRUE를 반환하는 IF, ELSIF 절의 statement list를 수행한다. 모든 조건들을 만족시키지 못하고 <if statement else clause>가 존재할 경우에는 해당 구문을 수행한다. ELSIF 구문들은 순서대로 먼저 나오는 조건식부터 평가하여 해당 조건식이 TRUE인 경우 그 이후의 조건식들은 평가하지 않는다

사용 예

```
gSQL> DECLARE
      V1 INTEGER := 10;
BEGIN
  IF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'POSITIVE' );
  ELSIF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ZERO' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'NEGATIVE' );
  END IF;
END;
/
POSITIVE
Anonymous PL block executed.
```

호환성

<if statement> 구문은 SQL 표준과 구문이 같고 동일하게 동작한다.

표 23-8 SQL 표준 호환성

Feature ID	설명	지원여부
P002	Computational completeness	O

23.24 Implicit Cursor Attribute

기능

PSM에서 정의된 implicit cursor의 상태값을 반환한다.

구문

```
<Implicit cursor attribute> ::=
    SQL '%' { ISOPEN | FOUND | NOTFOUND | ROWCOUNT }
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

설명

- 직전에 수행된 SQL 문의 결과를 저장한다.
 - ISOPEN: 현재 커서가 OPEN 상태인지 여부로써 항상 FALSE로 설정된다.
 - FOUND: 직전 SQL 결과에 의해 데이터가 반환 되었는지 여부이다.
 - NOTFOUND: FOUND의 반대이다.
 - ROWCOUNT: 직전 SQL 결과에 의해 영향받은 record의 개수이다.

사용 예

```
DECLARE
V1 INTEGER;
BEGIN
    SELECT COUNT(*) INTO V1 FROM T1;
    DBMS_OUTPUT.PUT_LINE('COUNT RET    = ' || V1);
    DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
```

```
DBMS_OUTPUT.PUT_LINE('SQL%FOUND      = ' || SQL%FOUND);
DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
COUNT RET      = 0
SQL%ISOPEN      = FALSE
SQL%FOUND        = TRUE
SQL%NOTFOUND     = FALSE
SQL%ROWCOUNT   = 1
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.25 INSERT Statement Extension

기능

PSM에서 지원하는 record-type 변수를 VALUES 절에 기술하여 데이터를 입력할 수 있도록 insert statement를 확장 기능이다.

구문

```

<PSM Insert Statement Extension Statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <Insert_source>
        [ <Returning_into_clause> ]
    ;
<Insert_source> ::=
    <value-list>
    | <from_subquery>
    | <from_default>
<from subquery> ::=
    <query_expression>
<from default> ::=
    DEFAULT VALUES
<Value-List> ::=
    VALUES <Value_item> [ , ... ]
<value-Item> ::=
    PSM-Record-Type-Variable
    | ( { <value expression> | DEFAULT } [, ...] )
<Returning_into_clause> ::=
    [ RETURN | RETURNING ] { * | { <value_expression> [ [AS] alias_name ] } [, ...] INTO
variable_name [, ...]

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

EXECUTE IMMEDIATE의 원본 SQL 문에는 PSM insert extension 구문을 사용할 수 없다.

구문 규칙 및 파라미터

Insert statement의 기본 구문과 동일하게 동작한다. 다만, value item에 기존의 value expression을 괄호에 묶어 연속으로 나열하는 방법 외에 PSM record type 변수를 기술하는 기능이 추가되었다.

Value_Item에 괄호 없이 변수를 사용할 경우 반드시 PSM record type의 변수를 기술해야 한다.

Insert extension 구문 형태로 사용할 경우 record type이 아닌 변수를 섞어서 사용할 수 없다.

설명

일반적인 insert statement 외에 PSM의 record type 변수를 이용하여 record를 저장하거나 returning into를 통해 결과를 받아 온다.

Insert에 대한 자세한 내용은 다음 예를 참조한다.

사용 예

```
gSQL> DECLARE
    rec t1%ROWTYPE;
BEGIN
    rec.i1 := 'AAA';
    rec.i2 := 'BBB';
    rec.i3 := 'CCC';
    INSERT INTO t1 (i1, i2, i3) VALUES rec ;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM t1;
I1  I2  I3
---  ---  ---
AAA BBB CCC
```

호환성

SQL 표준에 정의되어 있지 않다.

23.26 NULL Statement

기능

아무 기능도 없는 statement 이다.

구문

```
<null statement> ::=  
    NULL  
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

설명

아무 기능도 하지 않는 statement이며, 주로 특정 위치의 label을 설정하기 위해 사용된다.

사용 예

```
gSQL> BEGIN  
    FOR i in 1..10 LOOP  
        DBMS_OUTPUT.PUT_LINE( i );  
        IF i > 5 THEN  
            GOTO label1;  
        END IF;  
    END LOOP;  
<<label1>>  
    NULL;
```



```
END;
```

```
/
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.27 OPEN Statement

기능

PSM에서 정의된 cursor의 SELECT 구문을 실행한다.

구문

```
<open statement> ::=
    OPEN cursor_name [ <actual param spec> ]
    ;
<actual param spec> ::=
    ( expression [ , expression ] .. )
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Cursor_name
 - Open할 커서의 이름이다.

설명

정의된 커서의 SELECT 또는 SELECT ... FOR UPDATE 구문을 실행한다.
만일 커서가 declaration만 되어 있고 definition 되어 있지 않을 경우, 에러가 발생한다.
Actual parameter의 값들은 해당 parameter의 데이터 타입과 서로 호환 가능해야 한다.

Actual parameter의 개수는 커서의 parameter 개수와 같아야 한다.
만일 커서의 parameter의 개수보다 적을 경우, 나머지 모든 parameter들에 default 값이 명시되어야 한다.

사용 예

```

gSQL> CREATE TABLE T1 ( I1 INTEGER, I2 VARCHAR(10) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO T1 VALUES( 1, 'AAA' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 2, 'BBB' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 3, 'CCC' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    CURSOR C1( A1 INTEGER := 1, A2 VARCHAR DEFAULT 'AAA') IS SELECT * FROM T1 WHERE I1 = A1 AND
I2 = A2;
    V1 INTEGER;
    V2 VARCHAR(10);
BEGIN
    OPEN C1( 1 );
    FETCH C1 INTO V1, V2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 || ' V2 = ' || V2 );
    CLOSE C1;
END;
/
V1 = 1 V2 = AAA
Anonymous PL block executed.

```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- **CLOSE Statement**
- **FETCH Statement**

23.28 OPEN FOR Statement

기능

PSM에서 정의된 cursor 변수를 통해 SELECT 구문을 실행하여 한 개의 cursor를 open한다.

구문

```
<open statement> ::=
    OPEN cursor_variable_name FOR <select_query>
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Cursor_Variable_name
 - cursor_variable의 이름이다.
- Select_query
 - Select query에는 static SQL과 dynamic SQL을 모두 사용할 수 있다.

설명

정의된 cursor 변수의 SELECT 또는 SELECT ... FOR UPDATE 구문을 실행한다.
만일 cursor 변수가 이전에 열어둔 커서가 존재하면 해당 커서가 자동으로 close 된다.

사용 예

```

gSQL> CREATE TABLE T1 (c1 INTEGER, c2 INTEGER, c3 INTEGER);
Table created.
gSQL> INSERT INTO T1 VALUES (1, 1, 1);
1 row created.
gSQL> INSERT INTO T1 VALUES (2, 2, 2);
1 row created.
gSQL> INSERT INTO T1 VALUES (3, 3, 3);
1 row created.
gSQL> INSERT INTO T1 VALUES (4, 4, 4);
gSQL> DECLARE
cv SYS_REFCURSOR;
rec t1%ROWTYPE;
BEGIN
  OPEN cv FOR SELECT * FROM T1;
  DBMS_OUTPUT.PUT_LINE('After Open> CV%ISOPEN=' || CV%ISOPEN);
  LOOP
    FETCH cv INTO rec;
    EXIT WHEN CV%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('C1=' || rec.c1 || ', C2=' || rec.c2 || ', c3=' || rec.c3 || ',
RowCount=' || cv%rowcount);
  END LOOP;
  CLOSE cv;
  DBMS_OUTPUT.PUT_LINE('After Close> CV%ISOPEN=' || CV%ISOPEN);
END;
/
After Open> CV%ISOPEN=TRUE
C1=1, C2=1, c3=1, RowCount=1
C1=2, C2=2, c3=2, RowCount=2
C1=3, C2=3, c3=3, RowCount=3
C1=4, C2=4, c3=4, RowCount=4
After Close> CV%ISOPEN=FALSE
Anonymous PL block executed.

```

호환성

SQL 표준에 정의되어 있지 않다.

참조

자세한 내용은 다음을 참조한다.

- **Cursor Variable Declaration**
- **FETCH Statement**
- **CLOSE Statement**

23.29 Procedure Call

기능

사용자 정의 procedure, built-in procedure 또는 nested procedure를 호출한다.

구문

```
<Procedure call> ::=
    proc_name [ ( expr { , expr } ... ) ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

- 사용자 정의 procedure를 호출할 경우, 해당 procedure에 대한 다음 권한 중 하나가 있어야 한다.
 - EXECUTE PROCEDURE
 - Procedure가 속한 스키마에 대해 (EXECUTE PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
 - EXECUTE ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

- Proc_name
 - 실행할 procedure의 이름으로써 다음과 같은 형태로 사용할 수 있다.

형태	구문	설명
단일 identifier	procedure_name	주어진 이름의 procedure를 호출한다. 검색 순서는 다음과 같다. 1. nested procedure 2. schema-level procedure
	label_name.procedure_name	nested procedure를 호출한다.

형태	구문	설명
identifier chain	schema_name.procedure_name	schema-level procedure를 호출한다.
	package_name.procedure_name	built-in procedure를 호출한다.

주어진 이름 (proc_name)으로 검색하여 최초로 발견된 procedure에 주어진 인자와 개수 및 타입이 적절하지 않으면 다른 procedure를 찾지 않고 에러를 발생시킨다.

설명

이전에 정의된 사용자 정의 procedure, built-in procedure 또는 nested procedure를 호출한다.

호출할 때 default 값이 정의된 인자값은 생략할 수 있다.

OUT 이나 IN-OUT으로 지정된 인자에 procedure 변수나 bind parameter (?, :V1 등)를 사용하면 반환된 값을 얻을 수 있다.

사용 예

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER )
  IS
  BEGIN
    PUT_LINE( 'A1 = ' || A1 );
  END;
BEGIN
  PROC1( 100 );
END;
/
A1 = 100
Anonymous PL block executed.
```

호환성

SQL 표준에는 <call statement>를 사용하도록 되어 있다.

23.30 Procedure Declaration and Definition

기능

Nested procedure를 정의한다.

구문

```
<nested procedure statement> ::=
    PROCEDURE proc_name [ ( { param_name [IN|OUT|INOUT] datatype [ { := | DEFAULT } init_
    expr ] } [, ...] ) ]
    { IS | AS } <item_declaration> BEGIN <pl_stmt_list> END
    ;
```

사용 범위 및 접근 권한

PSM declaration section에서 사용할 수 있다.

구문 규칙 및 파라미터

- Proc_name
 - 생성할 procedure의 이름이며, 스키마 내에서 고유한 이름이어야 한다.
 - Procedure 이름의 길이는 128 바이트보다 작아야 한다.
- Param_name
 - Procedure가 사용할 인자의 이름을 정의한다.
 - 각 인자의 이름은 procedure 내에서 고유한 이름이어야 한다.
- Bind_type
 - 각 인자의 bind type을 설정한다.
 - 표기하지 않을 경우 기본 타입은 'IN'이다.
- Item_declaration
 - Procedure 내부에서 사용될 로컬 변수등의 item을 선언한다.
 - PL block에서 선언될 수 있는 모든 item들을 선언할 수 있다.
- PL Stmt List
 - Procedure의 body 부분으로써 수행할 PL statement들을 나열한다.

설명

Nested procedure는 해당 procedure 내에서만 호출할 수 있는 subprogram이다.
그 외의 사용 방법은 schema-level procedure와 동일하다.

사용 예

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER )
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
  END;
BEGIN
  PROC1( 100 );
END;
/
A1 = 100
Anonymous PL block executed.
```

호환성

Schema-level procedure와 동일하다.

참조

자세한 내용은 **CREATE PROCEDURE**를 참조한다.

23.31 RAISE Statement

기능

사용자가 정의한 exception을 명시적으로 발생시킨다.

구문

```
<RAISE Statement> ::=
    RAISE <Exception-Name>
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Raise 시킬 exception name이 raise가 속한 PL block이나 상위 PL block의 DECLARE 절에 선언되어야 한다. 단, predefined exception은 선언없이 raise할 수 있다.
- Raise 문이 exception handler 내에서 사용되면 exception name을 생략할 수 있는데 이 경우, 직전 exception이 상위 블록으로 전파된다.

설명

Exception이 발생한 PL block에서 처리되지 못하면 상위 PL block으로 전파된다.

Raise 시킬 exception이 RAISE 구문을 포함하는 PL block 및 상위 PL block 내의 모든 exception handler에 존재하지 않을 경우 에러가 발생한다.

처리될 때까지 RAISE exception이 발생한 PL block부터 상위로 전파되며 하위 PL block의 exception handler로는 전파되지 못한다.

표 23-9 User exception 전파

Raise exception	Exception handler SCOPE	Exception handler	상위 전파여부
User exception without error code	동일 scope	X	"unhandled exception" 오류를 상위 scope으로 전파한다.
User exception with error code	상위 scope	X	User exception을 전파한다.
User exception with error code	동일 scope	X	User defined error code를 전파한다.
User exception with error code	상위 scope	X	User defined error code를 전파한다.

사용 예

```

gSQL> DECLARE
V1 INTEGER;
exception1 EXCEPTION;
exception100 EXCEPTION;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Step1');
    BEGIN
        RAISE exception1;
        DBMS_OUTPUT.PUT_LINE('Step2');
        EXCEPTION WHEN Exception100 THEN DBMS_OUTPUT.PUT_LINE('in Exception');
    END;
    DBMS_OUTPUT.PUT_LINE('Step3');
    EXCEPTION WHEN Exception1 THEN DBMS_OUTPUT.PUT_LINE('out Exception');
    DBMS_OUTPUT.PUT_LINE('Step4');
END;
/
Step1
out Exception
Step4
Anonymous PL block executed.

```

호환성

SQL 표준에는 <handler declaration>과 <condition declaration>은 기술되어 있지만 구문은 지원하지 않는다.

참조

자세한 내용은 다음을 참조한다.

- [Exception Handler](#)
- [Exception Declaration](#)

23.32 Record Variable Declaration

기능

DECLARE 영역에서 record type 변수를 선언한다.

구문

```
<declare record variable> ::=
    variable_name <recordType>
    ;
<recordType> ::=
    <tableName>%ROWTYPE
    | USER_DEFINED_DATA_TYPE
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Variable_name
 - 선언할 변수의 이름이다.
 - 변수 이름의 길이는 128 바이트보다 작아야 한다.
 - 하나의 scope (PL block body) 내에서 고유한 이름이어야 한다.
 - 중첩된 하위의 PL block body에서는 같은 이름의 변수를 선언할 수 있다.
 - 중복 선언된 변수를 명확히 사용하기 위해서는 scope_name.variable_name의 형태로 사용해야 한다.
 - Scope 이름을 명시하지 않고 중복 선언된 변수를 사용할 경우에는 자동으로 가장 가까운 scope의 변수가 사용된다.
- Record_type
 - %ROWTYPE 및 사용자가 정의한 recordType을 사용할 수 있다.
 - recordType 선언에 대한 자세한 내용은 **User-defined Record Type**을 참조한다.

설명

- 선언된 변수는 다음과 같은 특징이 있다.
 - 하나의 scope (PL block body) 내에서 고유한 이름이어야 한다.
 - 중첩된 하위의 PL block body에 동일한 이름의 변수를 선언할 수 있다.
 - 중복 선언된 변수를 명확히 사용하기 위해서는 scope_name.variable_name의 형태로 사용해야 한다.
 - Scope 이름을 명시하지 않고 중복 선언된 변수를 사용할 경우에는 자동으로 가장 가까운 scope의 변수가 사용된다.
- 선언된 변수는 해당 scope와 하위 scope에서 사용되며, embedded SQL에서와 달리 ':' 기호를 붙이지 않는다.
- 사용된 변수는 해당 SQL이나 PSM control statement에 대해 bind parameter (INOUT)로 작동한다.

사용 예

```
gSQL> DECLARE
  TYPE MY_REC1 IS RECORD ( F1 INTEGER, F2 VARCHAR(10) );
  V1 MY_REC1;
BEGIN
  V1.F1 := 1;
  V1.F2 := 'AAA';
  INSERT INTO T1 VALUES( V1.F1, V1.F2 );
END;
/
Anonymous PL block executed.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT * FROM T1;
I1 I2
-- ---
1 AAA
1 row selected.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.33 RETURN Statement

기능

Function이 반환할 값을 지정한 후 해당 function을 종료한다. Procedure는 반환값을 지정하지 않고 해당 procedure를 종료한다.

구문

```
<return statement> ::=
    RETURN [ return_value_expr ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Return_value_expr
 - 반환할 값의 표현식으로써 function일 경우에만 명시할 수 있다.

설명

현재 수행되고 있는 procedure/ function을 종료한다.

Function의 경우, RETURN 구문이 수행되지 않고 종료되거나, RETURN 구문이 return_value_expr를 가지지 않으면 오류가 발생한다.

사용 예

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )  
      RETURN INTEGER  
      IS  
        V1 INTEGER;  
      BEGIN  
        SELECT COUNT(*)  
          INTO V1  
          FROM T1  
          WHERE T1.I1 >= A1 AND T1.I1 <= A2;  
        RETURN V1;  
      END;  
      /
```

Function created.

호환성

SQL 표준에는 기술되어 있지만 conformance rule은 존재하지 않는다.

23.34 RETURNING INTO clause

기능

Insert/ update/ delete에서 처리된 데이터가 PSM 변수로 반환된다.

구문

```
<Insert, Delete Returning_into_clause> ::=
    [ RETURN | RETURNING ] { * | { <value_expression> [ [AS] alias_name ] } [, ...] INTO
    variable_name [, ...]
<Update returning into clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name ] } [,
    ...] } INTO Variable [, ...]
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

레코드 타입 변수는 다른 타입과 섞어서 사용할 수 없다.

설명

Insert/ update/ delete 문을 처리한 before/ after record를 returning into를 통해 저장한다.

사용 예

```
gSQL> DECLARE
      rec t1%ROWTYPE;
BEGIN
      INSERT INTO t1 VALUES (1, 2, 3) RETURNING * INTO rec ;
      UPDATE T1 SET ROW = rec RETURNING * INTO rec;
      DELETE FROM T1 RETURNING * INTO rec;
END;
/
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.35 %ROWTYPE Attribute

기능

변수를 선언할 때 특정 테이블 또는 커서나 커서 변수의 result set과 같은 구조와 타입으로 정의한다.

구문

```
<rowtype attribute> ::=
    <identifier chain> % ROWTYPE
```

사용 범위 및 접근 권한

- PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
- PL block의 declaration 영역에서만 사용할 수 있다.
- 변수를 선언할 때는 <data type> 부분에만 사용할 수 있다
- 레코드 타입의 필드를 선언할 때는 <data type> 부분에 사용할 수 없다. (complex data type은 지원하지 않는다.)

구문 규칙 및 파라미터

- Identifier chain은 다음과 같다.
 - 참조할 테이블, view, synonym의 이름이나 커서 또는 커서 변수의 이름

설명

- 참조 대상 검색 순서
 - 커서 혹은 커서 변수
 - Base table, view, 또는 synonym
- 참조 범위
 - Row attribute를 이용하여 참조할 경우 테이블 또는 커서의 result set에서 column 이름과 타입만 참조한

- 다.
- 따라서 NOT NULL constraint나 DEFAULT 값 설정 등은 참조하지 않는다.

사용 예

```
gSQL> CREATE TABLE T1 ( I1 INTEGER NOT NULL, I2 VARCHAR(10) );
Table created.
gSQL> INSERT INTO T1 VALUES( 123, '1234567890' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
    V1 T1%ROWTYPE;
BEGIN
    SELECT * INTO V1.I1, V1.I2 FROM T1;
    DBMS_OUTPUT.PUT_LINE( 'V1.I1 = ' || V1.I1 || ' V1.I2 = ' || V1.I2 );
END;
/
V1.I1 = 123 V1.I2 = 1234567890
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.36 Scalar Variable Declaration

기능

DECLARE 영역에서 scalar 변수를 선언한다.

구문

```
<declare scalar variable> ::=
    variable_name <data type> [ <variable initialize clause> ]
    ;
<variable initialize clause> ::=
    [ NOT NULL ] { DEFAULT | := } <value expression>
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Variable_name
 - 선언할 변수의 이름이다.
 - 변수 이름의 길이는 128 바이트보다 작아야 한다.
 - 하나의 scope (PL block body) 내에서 고유한 이름이어야 한다.
 - 중첩된 하위의 PL block body에 동일한 이름의 변수를 선언할 수 있다.
 - 중복 선언된 변수를 명확히 사용하기 위해서는 scope_name.variable_name의 형태로 사용해야 한다.
 - Scope 이름을 명시하지 않고 중복 선언된 변수를 사용할 경우에는 자동으로 가장 가까운 scope의 변수가 사용된다.
- Data_Type
 - GOLDILOCKS에서 제공하는 모든 built-in **Data Type**들을 사용할 수 있다.
- Value_expression
 - 변수에 지정할 초기값을 표현한다.
 - GOLDILOCKS가 지원하는 모든 상수 및 multi-row 함수들을 제외한 모든 표현식을 사용할 수 있다.

설명

- 선언된 변수는 다음과 같은 특징이 있다.
 - 하나의 scope (PL block body) 내에서 고유한 이름이어야 한다.
 - 중첩된 하위의 PL block body에 동일한 이름의 변수를 선언할 수 있다.
 - 중복 선언된 변수를 명확히 사용하기 위해서는 scope_name.variable_name의 형태로 사용해야 한다.
 - Scope 이름을 명시하지 않고 중복 선언된 변수를 사용할 경우에는 자동으로 가장 가까운 scope의 변수가 사용된다.
- 선언된 변수는 해당 scope와 하위 scope에서 사용되며, embedded SQL에서와 달리 ':' 기호를 붙이지 않는다.
- 사용된 변수는 해당 SQL이나 PSM control statement에 대해 bind parameter (INOUT)로 작동한다.

사용 예

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I2 VARCHAR(10) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
V1 INTEGER := 100;
V2 INTEGER := -100;
V3 VARCHAR(10) := 'ABC';
BEGIN
  IF V1 > 50 THEN
    INSERT INTO T1 VALUES ( V1, V3 );
  ELSE
    INSERT INTO T1 VALUES ( V2, V3 );
  END IF;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
  I1 I2
--- ---
100 ABC
1 row selected.
```


호환성

- <declare scalar variable> 구문은 SQL 표준과 비교하여 다음과 같은 차이가 있다.
 - SQL 표준의 <SQL variable declaration>은 PL block body 내 (BEGIN 이후)에서 변수를 선언하지만, GOLDILOCKS는 별도의 declaration section에서 선언한다.
 - SQL 표준에서는 한 번의 DECLARE 구문으로 여러 개의 동일한 타입 변수를 선언할 수 있지만, GOLDILOCKS에서는 한 statement로 한 개의 변수만 선언할 수 있다.
 - SQL 표준에서는 초기 값을 세팅할 때 DEFAULT 구문만 사용하지만, GOLDILOCKS는 assign 기호인 :=도 사용할 수 있다.

23.37 SELECT INTO Statement

기능

SELECT를 통해 한 개의 row를 반환 받는다.

구문

```
<select statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
            <table expression>
    ;
<select target list> ::=
    variable_name [, ...]
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

INTO 절을 제외한 사항은 select statement와 동일한 규칙을 따른다.

설명

- SELECT INTO는 한 개의 record를 반환받는 용도로 사용한다.
 - 결과가 0 건인 경우, "NO_DATA_FOUND" exception이 발생한다.
 - 결과가 두 건 이상인 경우, "TOO_MANY_ROWS" exception이 발생한다.
- PSM의 record type 변수를 통해 결과를 반환받을 수 있다.
 - Record type 변수를 사용할 경우 다른 타입의 변수와 섞어서 사용할 수 없다.

사용 예

```
gSQL> CREATE TABLE T1 (c1 VARCHAR(20), c2 VARCHAR(20));
Table created.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC');
2 rows created.
gSQL> DECLARE
    v1 VARCHAR(20);
    v2 VARCHAR(20);
BEGIN
    SELECT * INTO v1, v2 FROM T1 WHERE c1 = 'AAA';
    DBMS_OUTPUT.PUT_LINE('V1=' || v1 || ', v2=' || v2);
END;
/
V1=AAA, v2=BBB
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.38 SQLCODE Function

기능

PSM에서 수행된 직전 statement의 error code를 반환한다.

구문

```
<SQLCODE function> ::= SQLCODE
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다.

구문 규칙 및 파라미터

별도의 argument를 갖지 않는다.

설명

PL/ SQL에서 수행된 statement의 error code를 반환한다.

Error code를 할당하지 않은 user-defined exception의 경우, handler에 의해 처리되는 시점에 1로 반환된다.

Exception handler에 의해 오류 처리가 완료되면 0으로 반환된다.

사용 예

```
gSQL> DECLARE
V1 INTEGER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('SQLCODE=[' || SQLCODE || ']');
```

```
        DBMS_OUTPUT.PUT_LINE('SQLERRM=[ ' || SQLERRM || ' ]');  
END;  
/  
SQLCODE=[0]  
SQLERRM=[ [SUNJESOFT][PL/SQL][GOLDILOCKS]successful completion]  
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.39 SQLERRM Function

기능

PSM에서 수행된 직전 statement의 error message를 반환한다.

구문

```
<SQLERRM function> ::= SQLERRM
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다.

구문 규칙 및 파라미터

별도의 argument를 갖지 않는다.

설명

PL/ SQL에서 수행된 statement의 error message를 반환한다.

Error code를 할당하지 않은 user-defined exception의 경우, handler에 의해 처리되는 시점에 user-defined exception으로 반환된다.

Exception handler에 의해 오류 처리가 완료되면 *successful completion* 메시지를 출력한다.

사용 예

```
gSQL> DECLARE
V1 INTEGER;
BEGIN
```

```
        DBMS_OUTPUT.PUT_LINE('SQLCODE=[ ' || SQLCODE || ' ]');
        DBMS_OUTPUT.PUT_LINE('SQLERRM=[ ' || SQLERRM || ' ]');
END;
/
SQLCODE=[0]
SQLERRM=[[SUNJESOFT][PL/SQL][GOLDILOCKS]successful completion]
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.40 %TYPE Attribute

기능

변수를 선언할 때나 RECORD 타입의 특정 필드를 정의할 때 그 타입을 특정 테이블의 column 또는 다른 변수와 동일한 타입으로 정의한다.

구문

```
<type attribute> ::=
    <identifier chain> % TYPE
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)

PL block의 declaration 영역에서만 사용할 수 있다.

변수를 선언할 때나 레코드 타입의 필드를 선언할 때 <data type> 부분에만 사용할 수 있다.

구문 규칙 및 파라미터

- Identifier_chain
 - 참조할 테이블의 column 또는 기존에 선언된 변수 (혹은 변수의 필드)의 이름이다.

설명

참조 범위

참조된 객체 타입에 따른 참조 범위는 다음과 같다.

- 참조된 객체가 테이블의 column일 때
 - Column의 데이터 타입을 참조한다.
 - Column의 constraint (예: NOT NULL)는 참조하지 않는다.

- Column의 기본 초기값은 참조하지 않는다.
- 참조된 객체가 다른 변수 (또는 변수의 필드)일 때
 - 변수 (또는 필드)의 데이터 타입을 참조한다.
 - 변수 (또는 필드)의 constraint (NOT NULL)를 참조한다.
 - 변수 (또는 필드)의 기본 초기값은 참조하지 않는다.
- 참조 대상 객체의 검색 순서는 다음과 같다.
 1. 변수 (또는 필드) 이름
 2. Column 이름

NOT NULL Constraints 변수 참조

NOT NULL 속성의 변수를 참조할 때는 초기값을 참조하지 않으므로 반드시 새로운 초기값을 지정해야 한다.

현재 record type 변수의 필드에 대한 type attribute를 사용할 때 필드 초기값 설정을 지원하지 않으므로 NOT NULL 타입의 필드는 참조할 수 없다.

사용 예

```
gSQL> DECLARE
  V1 NUMBER(5,2) := 100.01;
  V2 V1%TYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  DBMS_OUTPUT.PUT_LINE( 'V2 = ' || V2 );
END;
/
V1 = 100.01
V2 =
Anonymous PL block executed.
```

호환성

SQL 표준에 정의되어 있지 않다.

23.41 UPDATE Statement Extension

기능

PSM에서 UPDATE 문의 갱신 대상 column을 나열하는 방식 외에 추가적으로 record type 변수를 이용하여 레코드를 갱신한다.

PSM에서 UPDATE (searched) RETURNING INTO 절에 record type 변수를 사용하여 결과를 저장한다.

구문

```

<update Extension statement : searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        <target-list>
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        [ <returning into clause> ]
    ;

<update statement: positioned> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        <Target-List>
        WHERE CURRENT OF cursor_name
    ;

<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]

<fetch limit clause> ::=
    <fetch first clause>
    | <limit clause>

<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]

<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }

<returning into clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name ] } [,
    ...] } INTO Variable [, ...]

<target-List> ::=
    SET <set clause> [, ...]
  
```

```

        | SET ROW = <psm_variable>
<set clause> ::=
    column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )

```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
 PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- UPDATE SET ROW 구문에서 사용할 <PSM_Variable>은 반드시 record type으로 선언된 변수이어야만 한다.
- UPDATE RETURNING INTO 구문에서 사용할 <Variable>은 record type이 아니어도 된다.
 - 단, record type으로 기술한 경우 다른 데이터 타입의 변수와 섞어서 사용하거나 두 개 이상의 record type 변수를 나열할 수 없다.

설명

PSM에서 record type 변수를 통해 레코드를 갱신하거나 RETURNING INTO의 결과를 저장한다.

사용 예

```

gSQL> CREATE TABLE T1 (C1 VARCHAR(20), C2 VARCHAR(20));
Table created.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC'), ('CCC', 'DDD');
3 rows created.
gSQL> DECLARE
    v1 t1%ROWTYPE;
    v2 t1%ROWTYPE;
BEGIN
    v1.c1 := '1';
    v1.c2 := '2';
    UPDATE T1 SET ROW = v1 WHERE c1 = 'AAA' RETURNING * INTO v2;

```

```

        DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT=' || SQL%ROWCOUNT );
        DBMS_OUTPUT.PUT_LINE('v2.c1=' || v2.c1 || ', v2.c2=' || v2.c2);
END;
/
SQL%ROWCOUNT=1
v2.c1=1, v2.c2=2
Anonymous PL block executed.
gSQL> SELECT * FROM T1 ORDER BY C1;
C1  C2
--- ---
1   2
BBB CCC
CCC DDD
3 rows selected.

```

호환성

SQL 표준에 정의되어 있지 않다.

23.42 WHILE LOOP Statement

기능

<search condition>이 TRUE 값을 반환하는 동안 내부의 statement들을 수행한다.

구문

```
<while loop statement> ::=
    WHILE <search condition>
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

사용 범위 및 접근 권한

PSM 내에서만 사용할 수 있다. (예: package, procedure, function)
PL block의 body 영역에서만 사용할 수 있다.

구문 규칙 및 파라미터

- Search condition은 다음과 같다.
 - while 루프를 계속 순환하는 조건 표현식이다.
 - 최종적으로 boolean 타입을 반환해야 한다.

설명

while loop 구문은 <search condition>의 평가 결과가 TRUE인 동안 내부의 statement list를 수행한다.

사용 예

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  WHILE V1 < 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
  END LOOP;
END;
/
V1 = 0
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
V1 = 6
V1 = 7
V1 = 8
V1 = 9
Anonymous PL block executed.
```

호환성

〈while loop statement〉 구문은 SQL 표준에 〈while statement〉로 정의되어 있다.

SQL 표준의 〈while statement〉는 loop 구문을 DO ... END WHILE로 수행하는 반면에 GOLDDILOCKS는 LOOP ... END LOOP로 수행한다.

참조

자세한 내용은 다음을 참조한다.

- **CONTINUE Statement**
- **EXIT Statement**
- **GOTO Statement**

24.

PSM SQL References

24.1 ALTER FUNCTION

기능

Function을 다시 compile 한다.

구문

```
<alter function statement> ::=  
    ALTER FUNCTION function_name COMPILE  
    ;
```

사용 범위 및 접근 권한

<alter function statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 function의 소유자
- Function이 속한 스키마에 대해 (ALTER PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

- Function name
 - Compile 할 function의 이름이다.
 - schema_name.func_name과 같이 function이 속한 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

지정된 schema-level function을 다시 compile 한다.

사용 예

```
gSQL> ALTER FUNCTION FUNC1 COMPILE;  
Function altered.
```

호환성

SQL 표준에서는 CREATE를 수행할 때 지정된 characteristic들을 변경하는 구문이고, GOLDILOCKS에서는 function의 plan을 다시 생성하는 구문이다.

표 24-1 SQL 표준 호환성

Feature ID	설명	지원 여부
F381	Extended schema manipulation	O

참조

자세한 내용은 다음을 참조한다.

- CREATE FUNCTION
- DROP FUNCTION

24.2 ALTER PROCEDURE

기능

Procedure를 다시 compile 한다.

구문

```
<alter procedure statement> ::=  
    ALTER PROCEDURE proc_name COMPILE  
    ;
```

사용 범위 및 접근 권한

<alter procedure statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 procedure의 소유자
- Procedure가 속한 스키마에 대해 (ALTER PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
- ALTER ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

- Proc name
 - Compile할 procedure의 이름이다.
 - schema_name.proc_name과 같이 procedure가 속한 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

지정된 schema-level procedure를 다시 compile 한다.

사용 예

```

gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER )
IS
BEGIN
    INSERT INTO T1 VALUES( A1 );
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
(1) at (5:15): ERR-17053: schema or table object does not exist
Procedure created.
gSQL> CALL PROC1(1);
ERR-HY000(17032): PSM compilation error :
(1) at (5:15): ERR-17053: schema or table object does not exist
gSQL> CREATE TABLE T1( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> ALTER PROCEDURE PROC1 COMPILE;
Procedure altered.
gSQL> COMMIT;
Commit complete.
gSQL> CALL PROC1(2);
Procedure Call complete.
gSQL> SELECT * FROM T1;
I1
--
2
1 row selected.

```

호환성

SQL 표준에서는 CREATE를 수행할 때 지정된 characteristic들을 변경하는 구문이고, GOLDDILOCKS에서는 procedure의 plan을 다시 생성하는 구문이다.

표 24-2 SQL 표준 호환성

Feature ID	설명	지원 여부
F381	Extended schema manipulation	O

참조

자세한 내용은 다음을 참조한다.

- CREATE PROCEDURE
- DROP PROCEDURE

24.3 CALL Statement

기능

Schema-level procedure나 function을 수행한다.

구문

```

<call statement> ::=
    <sql call statement> | <odbc procedure call escape sequence>
    ;
<sql call statement> ::=
    CALL proc_name [ ( value_expr [ , value_expr ] .. ) ] [ INTO { '?' | { host_param [
indicator_param ] } } ]
<odbc procedure call escape sequence> ::=
    '{' [ ? = ] CALL proc_name [ ( value_expr [ , value_expr ] .. ) ] '}'

```

사용 범위 및 접근 권한

<call statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다

- 해당 procedure에 대한 EXECUTE 권한
- Procedure가 속한 schema에 대해 (EXECUTE PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
- EXECUTE ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

- proc_name
 - 실행할 procedure/ function의 이름이다
 - Schema_name.Proc_name과 같이 procedure가 속한 schema를 포함할 수 있다.
- value_expr
 - Procedure에 전달한 인자값을 표현한다. '?'나 ':V1'과 같은 bind parameter를 사용할 수도 있다.

설명

명시된 인자들을 사용하여 schema-level SQL procedure나 function을 실행한다.

<sql call statement> 형식 중에 function은 INTO 절 다음에 host variable 표현이나 dynamic bind parameter (?)를 사용하여 결과값을 반환한다.

<odbc procedure call escape sequence> 형식은 ODBC/ JDBC 등에서 PROCEDURE를 호출하기 위한 표준 구문이며, GOLDILOCKS는 이 구문을 server에서 지원한다. (gsql 등의 tool에서도 사용 가능하다.) Function은 앞에 assign 표현(? =)을 사용하여 결과값을 반환한다.

사용 예

Call Procedure

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1
(
  A1 INTEGER
)
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('A1=' || A1);
END;
/
Procedure created.
gSQL> \var v1 INTEGER;
gSQL> \exec :v1 := 123;
gSQL> CALL PROC1(:v1);
A1=123
Procedure Call complete.
```

Call Function

```
CREATE OR REPLACE FUNCTION FUNC1
(
  A1 INTEGER
)
RETURN INTEGER
IS
```

```
BEGIN
    return A1;
END;
/
Function created.
gSQL> \var v1 INTEGER;
gSQL> \var v2 INTEGER;
gSQL> \exec :v1 := 123;
gSQL> CALL FUNC1(:v1) INTO :v2;
Procedure Call complete.
gSQL> \print v2;
V2
---
123
```

호환성

SQL 표준에서는 PROCEDURE에 대한 호출만 가능하여 [INTO] 절 이하를 정의하지 않고 있다.

24.4 CREATE FUNCTION

기능

Schema-level function을 정의한다.

구문

```

<create procedure statement> ::=
    CREATE [ OR REPLACE ]
        FUNCTION func_name [ ( { param_name [IN|OUT|INOUT] datatype [ { := | DEFAULT } init_
expr ] } [, ...] ) ]
        RETURN datatype
        [ <func_characteristics> ]
        { IS | AS } <item_declaration> BEGIN <pl_stmt_list> END
    ;
<func_characteristics> ::=
    DETERMINISTIC | AUTHID CURRENT_USER | AUTHID DEFINER

```

사용 범위 및 접근 권한

<create function statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- Function을 생성하기 위해 다음 권한 중 하나가 있어야 한다.
 - Function이 속한 스키마에 대해 (CREATE PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY PROCEDURE ON DATABASE
- OR REPLACE 절을 사용할 때 이미 function이 존재할 경우, 기존 function을 제거할 수 있는 다음 권한 중 하나가 있어야 한다.
 - 해당 function의 소유자
 - Function이 속한 스키마에 대해 (DROP PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
 - DROP ANY PROCEDURE ON DATABASE
- 구문을 수행한 사용자는 생성한 function의 소유자이다.

구문 규칙 및 파라미터

OR REPLACE

이미 function이 존재할 경우, 기존의 function을 대체한다.

FUNCTION NAME

생성할 function의 이름으로써 스키마 내에서 고유한 이름이어야 한다.

schema_name.func_name과 같이 function이 속할 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

Function 이름의 길이는 128 바이트보다 작아야 한다.

PARAM NAME

Function이 사용할 인자의 이름을 정의한다.

각 인자의 이름은 function 내에서 고유한 이름이어야 한다.

각 인자 이름의 길이는 128 바이트보다 작아야 한다.

하나의 function에 사용할 수 있는 인자의 최대 개수에는 제한이 없다.

Bind Type

각 인자의 bind type을 설정한다.

표기하지 않을 경우 기본 타입은 IN 이다.

Func_Characteristics

Function 수행 옵션을 정의한다. 같은 항목에 대해서는 한 번만 기술해야 한다.

- DETERMINISTIC: 동일한 인자값들이 입력될 경우 해당 function은 항상 동일한 결과값을 반환한다.
- AUTHID CURRENT_USER: Function을 실행하는 중에 수행되는 SQL들은 invoker의 authorization에 따라 해석되고 수행된다.
- AUTHID DEFINER: Function을 실행하는 중에 수행되는 SQL들은 definer의 authorization에 따라 해석되고 수행된다.

Item Declaration

Function 내부에서 사용될 로컬 변수등의 item을 선언한다.

PL block에서 선언될 수 있는 모든 item들을 선언할 수 있다.

PL Stmt List

Function의 body 부분으로써 수행할 PL statement들을 나열한다.

Function의 내부에서는 '?'나 ':V1'과 같은 bind parameter를 사용할 수 없다.

설명

Schema-level SQL function을 정의한다. 생성된 function은 모든 expression에서 호출될 수 있다.

Function의 정의는 INFORMATION_SCHEMA의 ROUTINES 테이블에서 확인할 수 있다. Function의 parameter들에 대한 정의는 INFORMATION_SCHEMA의 PARAMETERS 테이블에서 확인할 수 있다.

관련 객체들이 존재하지 않아 function이 일시적으로 불완전해졌을 경우, <alter function> 구문으로 plan을 다시 생성하려 시도해볼 수 있다.

생성된 function은 <drop function> 구문을 사용하여 제거할 수 있다.

Function의 최대 생성 개수에는 제한이 없으므로 저장 공간에 문제가 없는 한 계속 생성할 수 있다.

사용 예

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
RETURN INTEGER
IS
    V1 INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    RETURN V1;
END;
/
```

Function created.

호환성

SQL 표준에서는 OR REPLACE 절을 정의하지 않고 있다.

표 24-3 SQL 표준 호환성

Feature ID	설명	지원여부
T471	Result sets return value	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
S023	Basic structured types	X
S241	Transform functions	X
S024	Enhanced structured types	X
T571	Array-returning external SQL-invoked functions	X
T572	Multiset-returning external SQL-invoked functions	X
S201	SQL routines on arrays	X
S202	SQL-invoked routines on multisets	X
T323	Explicit security for external routines	X
S231	Structured type locators	X
S232	Array locators	X
S233	Multiset locators	X
T041	Basic LOB data type support	X
S027	Create method by specific method name	X
T041	Basic LOB data type support	X
T324	Explicit security for SQL routines	O
T326	Table functions	X
T651	SQL-schema statements in SQL routines	X
T652	SQL-dynamic statements in SQL routines	O
T653	SQL-schema statements in external routines	X
T654	SQL-dynamic statements in external routines	X
T655	Cyclically dependent routines	X
T272	Enhanced savepoint management	X
T522	Default values for IN parameters of SQL-invoked procedures	O
B121	Routine language Ada	X
B122	Routine language C	X
B123	Routine language COBOL	X
B124	Routine language Fortran	X
B125	Routine language MUMPS	X
B126	Routine language Pascal	X
B127	Routine language PL/I	X
B128	Routine language SQL	O
B129	Routine language Ada: VARCHAR and NUMERIC support	X

참조

자세한 내용은 다음을 참조한다.

- **DROP FUNCTION**
- **ALTER FUNCTION**

24.5 CREATE PROCEDURE

기능

Schema-level procedure를 정의한다.

구문

```

<create procedure statement> ::=
    CREATE [ OR REPLACE ]
        PROCEDURE proc_name [ ( { param_name [IN|OUT|INOUT] datatype [ { := | DEFAULT } init_
expr ] } [, ...] ) ]
        [ <proc_characteristics> ]
        { IS | AS } <item_declaration> BEGIN <pl_stmt_list> END
    ;
<proc_characteristics> ::=
    AUTHID CURRENT_USER | AUTHID DEFINER

```

사용 범위 및 접근 권한

<create procedure statement> 구문을 수행하려면 사용자가 다음 조건들을 만족해야 한다.

- Procedure를 생성하기 위해 다음 권한 중 하나가 있어야 한다.
 - Procedure가 속한 스키마에 대해 (CREATE PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
 - CREATE ANY PROCEDURE ON DATABASE
- OR REPLACE 절을 사용할 때 이미 procedure가 존재할 경우, 기존 procedure를 제거할 수 있는 다음 권한 중 하나가 있어야 한다.
 - 해당 procedure의 소유자
 - Procedure가 속한 스키마에 대해 (DROP PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
 - DROP ANY PROCEDURE ON DATABASE
- 구문을 수행한 사용자는 생성한 procedure의 소유자이다.

구문 규칙 및 파라미터

[OR REPLACE]

이미 procedure가 존재할 경우, 기존의 procedure를 대체한다.

PROC NAME

생성할 procedure의 이름으로써 스키마 내에서 고유한 이름이어야 한다.

schema_name.proc_name과 같이 procedure가 속할 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

Procedure 이름의 길이는 128 바이트보다 작아야 한다.

PARAM NAME

Procedure가 사용할 인자의 이름을 정의한다.

각 인자의 이름은 procedure 내에서 고유한 이름이어야 한다.

각 인자 이름의 길이는 128 바이트보다 작아야 한다.

하나의 procedure에 사용할 수 있는 인자의 최대 개수에는 제한이 없다.

Bind Type

각 인자의 bind type을 설정한다.

표기하지 않을 경우 기본 타입은 IN이다.

proc_characteristics

Procedure 수행 옵션을 정의한다. 같은 항목에 대해서는 한 번만 기술해야 한다.

- AUTHID CURRENT_USER: Procedure를 실행하는 중에 수행되는 SQL들은 invoker의 authorization에 따라 해석되고 수행된다.
- AUTHID DEFINER: Procedure를 실행하는 중에 수행되는 SQL들은 definer의 authorization에 따라 해석되고 수행된다.

item declaration

Procedure 내부에서 사용될 로컬 변수등의 item을 선언한다.

PL block에서 선언될 수 있는 모든 item들을 선언할 수 있다.

PL Stmt List

Procedure의 body 부분으로써 수행할 PL statement들을 나열한다.

Procedure의 내부에서는 '?'나 ':V1'과 같은 bind parameter를 사용할 수 없다.

설명

Schema-level SQL procedure를 정의한다. 생성된 procedure는 CALL 구문, anonymous block, 또는 다른 procedure/ function에서 호출될 수 있다.

Procedure의 정의는 INFORMATION_SCHEMA의 ROUTINES 테이블에서 확인할 수 있다. Procedure의 parameter들에 대한 정의는 INFORMATION_SCHEMA의 PARAMETERS 테이블에서 확인할 수 있다.

관련 객체들이 존재하지 않아 procedure가 일시적으로 불완전해졌을 경우, <alter procedure> 구문으로 plan을 다시 생성하려 시도해볼 수 있다.

생성된 procedure는 <drop procedure> 구문을 사용하여 제거할 수 있다.

Procedure의 최대 생성 개수에는 제한이 없으므로 저장 공간에 문제가 없는 한 계속 생성할 수 있다.

사용 예

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
   FROM T1
  WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
Procedure created.
```

호환성

SQL 표준에서는 다음과 같은 절을 정의하지 않고 있다.

표 24-4 SQL 표준 호환성

Feature ID	설명	지원여부
T471	Result sets return value	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
S023	Basic structured types	X
S241	Transform functions	X
S024	Enhanced structured types	X
T571	Array-returning external SQL-invoked functions	X
T572	Multiset-returning external SQL-invoked functions	X
S201	SQL routines on arrays	X
S202	SQL-invoked routines on multisets	X
T323	Explicit security for external routines	X
S231	Structured type locators	X
S232	Array locators	X
S233	Multiset locators	X
T041	Basic LOB data type support	X
S027	Create method by specific method name	X
T041	Basic LOB data type support	X
T324	Explicit security for SQL routines	O
T326	Table functions	X
T651	SQL-schema statements in SQL routines	X
T652	SQL-dynamic statements in SQL routines	O
T653	SQL-schema statements in external routines	X
T654	SQL-dynamic statements in external routines	X
T655	Cyclically dependent routines	X
T272	Enhanced savepoint management	X
T522	Default values for IN parameters of SQL-invoked procedures	O
B121	Routine language Ada	X
B122	Routine language C	X
B123	Routine language COBOL	X
B124	Routine language Fortran	X
B125	Routine language MUMPS	X
B126	Routine language Pascal	X
B127	Routine language PL/I	X
B128	Routine language SQL	O
B129	Routine language Ada: VARCHAR and NUMERIC support	X

참조

자세한 내용은 다음을 참조한다.

- DROP PROCEDURE
- ALTER PROCEDURE

24.6 DROP FUNCTION

기능

Function을 제거한다.

구문

```
<drop function statement> ::=  
    DROP FUNCTION [ IF EXISTS ] func_name  
    ;
```

사용 범위 및 접근 권한

<drop function statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 function의 소유자
- Function이 속한 스키마에 대해 (DROP PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

Function이 존재하지 않더라도 에러가 발생하지 않는다.

FUNC NAME

제거할 function의 이름이다.

schema_name.func_name과 같이 function이 속한 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

지정된 schema-level function을 제거한다.

사용 예

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1
RETURN INTEGER
IS
    V1 INTEGER;
BEGIN
    V1 := 10;
    RETURN V1;
END;
/
Function created.
COMMIT;
Commit complete.
gSQL> DROP FUNCTION FUNC1;
Function dropped.
```

호환성

SQL 표준은 다음과 같은 절을 정의하지 않고 있다.

표 24-5 SQL 표준 호환성

Feature ID	설명	지원 여부
F032	CASCADE drop behavior	X
S024	Enhanced structured types	X

참조

자세한 내용은 다음을 참조한다.

- CREATE FUNCTION

- **ALTER FUNCTION**

24.7 DROP PROCEDURE

기능

Procedure를 제거한다.

구문

```
<drop procedure statement> ::=  
    DROP PROCEDURE [ IF EXISTS ] proc_name  
    ;
```

사용 범위 및 접근 권한

<drop procedure statement> 구문을 수행하려면 사용자에게 다음 권한 중 하나가 있어야 한다.

- 해당 procedure의 소유자
- Procedure가 속한 스키마에 대해 (DROP PROCEDURE 또는 CONTROL SCHEMA) ON SCHEMA
- DROP ANY PROCEDURE ON DATABASE

구문 규칙 및 파라미터

IF EXISTS

Procedure가 존재하지 않더라도 에러가 발생하지 않는다.

PROC NAME

제거할 procedure의 이름이다.

schema_name.proc_name과 같이 procedure가 속한 스키마를 정의할 수 있다. schema_name을 생략할 경우, 구문을 수행하는 사용자의 기본 스키마 이름이 사용된다.

설명

지정된 schema-level procedure를 제거한다.

사용 예

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
  FROM T1
  WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
Procedure created.
COMMIT;
Commit complete.
gSQL> DROP PROCEDURE PROC1;
Procedure dropped.
```

호환성

SQL 표준은 다음과 같은 절을 정의하지 않고 있다.

표 24-6 SQL 표준 호환성

Feature ID	설명	지원 여부
F032	CASCADE drop behavior	X
S024	Enhanced structured types	X

참조

자세한 내용은 다음을 참조한다.

- CREATE PROCEDURE
- ALTER PROCEDURE

파트 V.

Developer Manual

25. ODBC	2,367
25.1 GOLDILOCKS ODBC Driver 개요	2,368
GOLDILOCKS ODBC Driver 개념	2,368
ODBC 구성요소 개요	2,368
GOLDILOCKS ODBC Driver 사용	2,369
25.2 데이터 원본 구성	2,371
UNIX에서 DSN 구성	2,371
Windows에서 DSN 구성	2,377
25.3 GLOBAL CONNECTION	2,380
특징	2,380
설정	2,381
GLOBAL CONNECTION 처리 과정	2,382
제약 사항	2,384
25.4 카탈로그 함수	2,385
카탈로그 데이터 사용	2,385
ODBC 카탈로그 함수	2,386
25.5 ODBC API References	2,390
SQLAllocConnect	2,390
SQLAllocEnv	2,391
SQLAllocHandle	2,392
SQLAllocStmt	2,395
SQLBindCol	2,396
SQLBindParameter	2,403
SQLBrowseConnect	2,417
SQLBulkOperations	2,418
SQLCancel	2,419
SQLCancelHandle	2,420
SQLCloseCursor	2,421
SQLColAttribute	2,423
SQLColAttributes	2,428
SQLColumnPrivileges	2,429
SQLColumns	2,432
SQLConnect	2,436
SQLCopyDesc	2,438
SQLDescribeCol	2,439
SQLDescribeParam	2,442
SQLDisconnect	2,444
SQLDriverConnect	2,446
SQLEndTran	2,450

SQLError	2,452
SQLExecDirect	2,453
SQLExecute	2,456
SQLExtendedFetch	2,459
SQLFetch	2,460
SQLFetchScroll	2,467
SQLForeignKeys	2,473
SQLFreeConnect	2,478
SQLFreeEnv	2,479
SQLFreeHandle	2,480
SQLFreeStmt	2,482
SQLGetConnectAttr	2,484
SQLGetConnectOption	2,487
SQLGetCursorName	2,488
SQLGetData	2,490
SQLGetDescField	2,496
SQLGetDescRec	2,512
SQLGetDiagField	2,516
SQLGetDiagRec	2,524
SQLGetEnvAttr	2,527
SQLGetFunctions	2,531
SQLGetGroupCount	2,536
SQLGetGroupIDs	2,538
SQLGetGroupName	2,540
SQLGetInfo	2,542
SQLGetStmtAttr	2,553
SQLGetStmtOption	2,565
SQLGetSuitableGroupID	2,566
SQLGetTypeInfo	2,568
SQLMoreResults	2,575
SQLNativeSql	2,579
SQLNumParams	2,580
SQLNumResultCols	2,583
SQLParamData	2,586
SQLParamOptions	2,588
SQLPrepare	2,589
SQLPrimaryKeys	2,594
SQLProcedureColumns	2,599
SQLProcedures	2,606

SQLPutData	2,611
SQLRowCount	2,616
SQLSetConnectAttr	2,619
SQLSetConnectOption	2,627
SQLSetCursorName	2,628
SQLSetDescField	2,631
SQLSetDescRec	2,654
SQLSetEnvAttr	2,659
SQLSetParam	2,663
SQLSetPos	2,664
SQLSetScrollOptions	2,676
SQLSetStmtAttr	2,677
SQLSetStmtOption	2,691
SQLSpecialColumns	2,692
SQLStatistics	2,699
SQLTablePrivileges	2,706
SQLTables	2,712
SQLTransact	2,718
25.6 XA API References	2,719
개요	2,719
XA 인터페이스	2,720
사용 예	2,731
26. JDBC	2,737
26.1 GOLDILOCKS JDBC 개요	2,738
GOLDILOCKS JDBC 소개	2,738
특징	2,738
버전 및 지원 여부	2,740
사용 예	2,741
26.2 기능 명세	2,743
연결	2,743
데이터 조작	2,747
데이터 조회	2,751
ResultSet 스크롤	2,752
추가 타입 사용	2,754
로깅	2,758
Plan Text 조회	2,760
Connection Failover	2,761
Direct Attach 모드 접속	2,763
26.3 JDBC API References	2,765

Array	2,765
Blob	2,767
CallableStatement	2,769
Clob	2,787
CommonDataSource	2,789
Connection	2,794
ConnectionPoolDataSource	2,804
DatabaseMetaData	2,805
DataSource	2,838
Driver	2,839
NClob	2,841
ParameterMetaData	2,843
PooledConnection	2,846
PreparedStatement	2,848
Ref	2,861
ResultSet	2,862
ResultSetMetaData	2,899
RowId	2,904
RowSet	2,905
RowSetMetaData	2,915
Savepoint	2,918
SQLData	2,919
SQLXML	2,920
Statement	2,922
Struct	2,932
XAConnection	2,933
XADataSource	2,934
XAResource	2,935
GoldilocksInterval	2,937
GOLDILOCKS Type	2,947
타입 변환	2,947
27. Embedded SQL	2,953
27.1 Precompiler	2,954
개요	2,954
Building Application	2,955
Precompiler Options	2,959
27.2 Embedded SQL	2,964
Preprocessing	2,964
연결	2,970

Transaction	2,972
Host Variables and Datatypes	2,974
Embedded SQL	3,025
Options	3,046
Host Array	3,047
Handling Run-time Error	3,064
27.3 Advanced Topic	3,077
Embedded Dynamic SQL	3,077
Multi-threaded Application	3,091
C++ Application	3,102
XA	3,103
27.4 Embedded SQL Reference	3,117
EXEC SQL AT	3,118
EXEC SQL ATOMIC INSERT	3,119
EXEC SQL AUTOCOMMIT	3,121
EXEC SQL BEGIN DECLARE SECTION	3,122
EXEC SQL COMMIT RELEASE	3,123
EXEC SQL CONNECT	3,124
EXEC SQL CONTEXT ALLOCATE	3,126
EXEC SQL CONTEXT FREE	3,128
EXEC SQL CONTEXT USE	3,130
EXEC SQL DISCONNECT	3,132
EXEC SQL END DECLARE SECTION	3,133
EXEC SQL FOR	3,134
EXEC SQL GET GROUPID INTO	3,135
EXEC SQL INCLUDE	3,137
EXEC SQL INCLUDE SQLCA	3,138
EXEC SQL OPTION	3,139
EXEC SQL ROLLBACK RELEASE	3,140
EXEC SQL WHENEVER	3,141
28. PDO	3,143
28.1 PDO 개요	3,144
28.2 설치/구성	3,144
요구 사항	3,144
설치	3,144
28.3 사용	3,146
데이터 원본 이름 (DSN)	3,146
28.4 사용 예	3,147
29. PyDBC	3,157

29.1	GOLDILOCKS PyDBC	3,158
	개요	3,158
	버전 체계	3,158
	설치	3,158
	사용 예	3,159
29.2	API Reference	3,161
	pygoldilocks 모듈	3,161
	Connection	3,164
	Cursor	3,166
	Row	3,174
29.3	Exception	3,176
29.4	Data Type	3,177
	Python 매개 변수를 GOLDILOCKS로 전달	3,177
	GOLDILOCKS로부터 전달받는 SQL 값	3,178
30.	Hibernate	3,181
30.1	개요	3,182
30.2	연동	3,182
	다운로드	3,182
	GoldilocksDialect 클래스 연동	3,182
30.3	사용 예	3,183
	설정	3,183
	Application 예제	3,185

25.

ODBC

25.1 GOLDILOCKS ODBC Driver 개요

GOLDILOCKS ODBC Driver 개념

Open Database Connectivity (ODBC)는 데이터베이스 Application Programming Interface (API)에 대한 스펙이다. Microsoft ODBC 3.0 버전은 International Standards Organization/ International Electromechanical Commission (ISO/ IEC)과 X/ open의 콜 레벨 인터페이스 (CLI) 권장 스펙을 기반으로 한다. ODBC는 C 라이브러리 함수를 사용하여 SQL 문을 지원한다. 응용 프로그램에서 이 함수를 호출하여 ODBC 기능을 구현한다.

ODBC 아키텍처에는 다음과 같은 기능을 수행하는 네 가지 구성 요소가 있다.

구성요소	기능
응용 프로그램	ODBC 데이터 원본과 통신하는 ODBC 함수를 호출하고, SQL 문을 전송하여 결과 집합을 처리한다.
드라이버 관리자	응용 프로그램과 응용 프로그램에서 사용하는 모든 ODBC 드라이버 사이의 통신을 관리한다.
드라이버	응용 프로그램으로부터의 모든 ODBC 호출을 처리하고, 데이터 원본에 연결하여 응용 프로그램에서 데이터 원본으로 SQL 문을 전달한 후 응용 프로그램에 결과를 반환한다. 필요한 경우 드라이버가 응용 프로그램에서 전달된 ODBC SQL을 데이터 원본에서 사용하는 기본 SQL로 변환한다.
데이터 원본	드라이버가 DBMS의 데이터에 접근하는데 필요한 모든 정보를 포함한다.

ODBC 응용 프로그램을 사용하여 다음과 같은 작업을 수행할 수 있다.

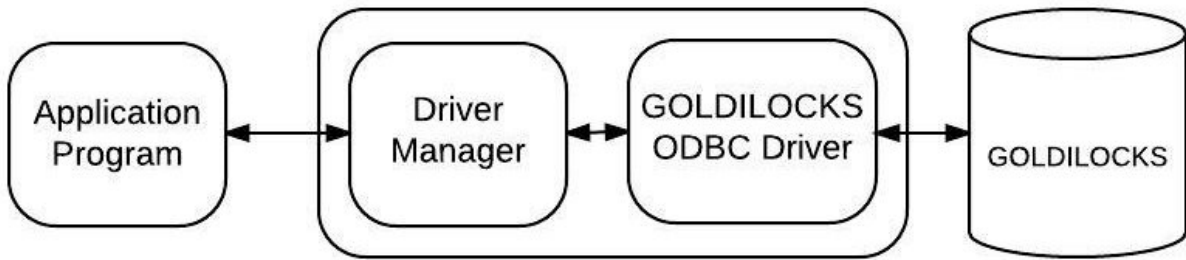
- 데이터 원본과 연결
- 데이터 원본에 SQL 문 전송
- 데이터 원본에서 SQL 문의 결과 처리
- 오류 및 메시지 처리
- 데이터 원본에 대한 연결 종료

ODBC 구성요소 개요

드라이버 관리자를 포함하는 GOLDILOCKS ODBC Driver

다음은 드라이버 관리자가 시스템에 포함된 소프트웨어 아키텍처이다. 이 경우, 응용 프로그램은 드라이버 관리자 라이브러리로 링크해야 한다.

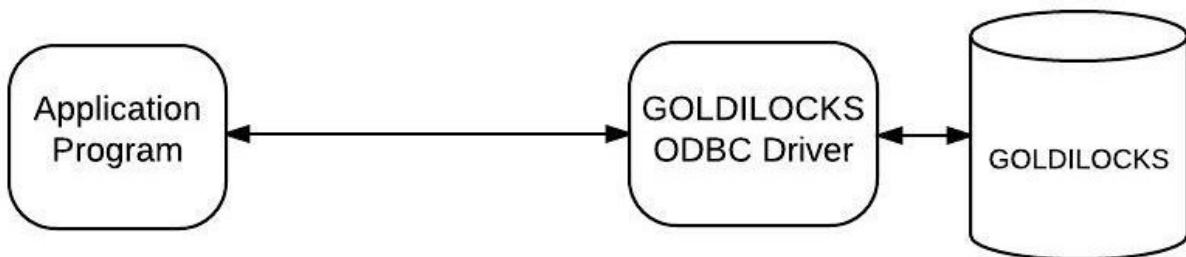
그림 1 드라이버 관리자를 포함하는 GOLDILOCKS ODBC driver



드라이버 관리자를 포함하지 않는 GOLDILOCKS ODBC Driver

다음은 응용 프로그램이 드라이버 관리자를 포함하지 않고 GOLDILOCKS ODBC driver를 사용하는 아키텍처이다. 이 경우, 응용 프로그램은 GOLDILOCKS ODBC driver 라이브러리로 링크해야 한다.

그림 2 드라이버 관리자를 포함하지 않는 GOLDILOCKS ODBC driver



GOLDILOCKS ODBC Driver 사용

헤더 파일

GOLDILOCKS ODBC driver를 실행하려면 \$GOLDILOCKS_HOME/include에 설치된 goldilocks.h 파일을 포함해야 한다. 이 파일은 GOLDILOCKS ODBC driver의 상수와 유형을 정의하고, GOLDILOCKS ODBC driver 함수의 프로토 타입을 제공한다.

라이브러리

드라이버 관리자를 사용하지 않는 응용 프로그램은 GOLDILOCKS ODBC driver 라이브러리의 정적 또는 공유 버전 에 링크해야 한다.

UNIX

표 25-1 GOLDILOCKS UNIX ODBC driver 라이브러리

파일 이름	설명
libgoldilocks.a	DA와 CS가 포함된 라이브러리의 정적 버전
libgoldilocks.a	DA 전용 라이브러리의 정적 버전
libgoldilocksas.so	DA 전용 라이브러리의 공유 버전
libgoldilocksc.a	CS 전용 라이브러리의 정적 버전
libgoldilockscs-ul32.so	SQLLEN을 4바이트로 인식하는 64 비트 CS 전용 라이브러리의 공유 버전
libgoldilockscs-ul64.so	SQLLEN을 8바이트로 인식하는 64 비트 CS 전용 라이브러리의 공유 버전
libgoldilockscs.so	32 비트 CS 전용 라이브러리의 공유 버전
libgoldilockss.so	DA와 CS가 포함된 라이브러리의 공유 버전

Windows

GOLDILOCKS Windows ODBC driver 라이브러리는 CS 라이브러리만 제공한다.

표 25-2 GOLDILOCKS Windows ODBC driver 라이브러리

파일 이름	설명
goldilockscs-ul64.dll	SQLLEN을 8바이트로 인식하는 64 비트 CS 전용 라이브러리의 공유 버전
goldilockscs.dll	32 비트 CS 전용 라이브러리의 공유 버전
goldilockssetup32.dll	32 비트 ODBC driver manager를 위한 설정 라이브러리
goldilockssetup64.dll	64 비트 ODBC driver manager를 위한 설정 라이브러리

25.2 데이터 원본 구성

UNIX에서 DSN 구성

odbcinst.ini 파일

odbcinst.ini 파일은 설치된 ODBC 드라이버에 대한 설정 파일이다.

- unixODBC

```
% odbcinst -j
unixODBC 2.3.4
DRIVERS.....: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /home/goldilocks/.odbc.ini
SQLULEN Size.....: 8
SQLLEN Size.....: 8
SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcinstini
/etc/odbcinst.ini
```

ODBC 드라이버 스펙

odncinst.ini 파일의 ODBC 드라이버 스펙 섹션은 드라이버의 속성값과 목록을 기술한다. 설치된 각 드라이버에는 드라이버 이름 아래 등록 정보 섹션이 있다.

```
[driver_name]
Description = driver_description
Driver = driver_library_path
Setup = setup_library_path
FileUsage = file_usage
```

드라이버 스펙 섹션의 키워드는 다음 표와 같다.

키워드	설명
Description	드라이버를 설명하는 문자열

키워드	설명
Driver	드라이버 라이브러리 경로
Setup	설치 라이브러리 경로
FileUsage	파일 기반 드라이버가 DSN에서 파일을 직접 처리하는 방법을 표시하는 문자

다음은 GOLDILOCKS ODBC 드라이버 스펙에 대한 정보를 보여주는 예이다.

```
[GOLDILOCKS ODBC Driver]
Description= GOLDILOCKS ODBC Driver
Driver = /home/goldilocks/home/lib/libgoldilockscs-ul64.so
Setup = /home/goldilocks/home/lib/libgoldilockscs-ul64.so
FileUsage = 0
```

odbc.ini 파일

odbc.ini 파일은 응용 프로그램이 연결하는 DSN에 대한 설정 파일이며, 사용자 DSN과 시스템 DSN으로 나뉜다. 일반적으로 사용자 DSN 파일은 ~/.odbc.ini 파일이고 시스템 DSN 파일은 /etc/odbc.ini 파일이다.

- unixODBC

```
% odbcinst -j
unixODBC 2.3.4
DRIVERS.....: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /home/goldilocks/.odbc.ini
SQLULEN Size.....: 8
SQLLEN Size.....: 8
SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcini
/etc/odbc.ini
```

데이터 원본 스펙

odbc.ini 파일의 데이터 원본 스펙 섹션은 DSN을 설명한다.

```
[data_source_name]
Driver = driver_name
PROTOCOL = protocol_type
```

```

CS_MODE = {default | dedicated | shared}
HOST = host_address
PORT = port_no
CHARSET = {SQL_ASCII | UTF8 | UHC | GB18030}
TCP_NODELAY = {0 | 1}
ALTERNATE_SERVERS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)
CONNECTION_RETRY_COUNT = retry_count
CONNECTION_RETRY_DELAY = retry_delay
FAILOVER_TYPE = {CONNECTION | SESSION}
FAILOVER_GRANULARITY = {0 | 1 | 2}
DATE_FORMAT = date_format_string
TIME_FORMAT = time_format_string
TIME_WITH_TIME_ZONE_FORMAT = timetz_format_string
TIMESTAMP_FORMAT = timestamp_format_string
TIMESTAMP_WITH_TIME_ZONE_FORMAT = timestamptz_format_string
CHAR_LENGTH_UNITS = {BYTE | OCTETS | CHAR | CHARACTERS}
ENABLE_SQLDESCRIBEPARAM = {0 | 1}
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = {0 | 1}
USE_TARGETTYPE = {0 | 1 | 2}
LOCATOR_DSN = locator_dsn_name
LOCATOR_SERVICE = locator_service_name
LOCALITY_AWARE_TRANSACTION = {0 | 1}
LOCALITY_GROUP_POLICY = {0 | 1 | 2}
LOCALITY_GROUP_PATH = group_name1, group_name2, group_name3
LOCALITY_MEMBER_POLICY = {0 | 1 | 2 | 3 | 4}
LOCALITY_MEMBER_PATH = member_name1, member_name2, member_name3
DB_HOME = database_home_path
PACKET_COMPRESSION_THRESHOLD = packet_compression_threshold
[locator_dsn_name]
FILE = location_file_name
HOST = IP address(v4)
PORT = locator_port
CONNECTION_TIMEOUT = second
ALTERNATE_LOCATORS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)

```

데이터 원본 스펙 섹션의 키워드는 다음 표와 같다.

표 25-3 데이터 원본 스펙 섹션의 키워드

키워드	설명
data_source_name	데이터 원본 섹션에 지정된 데이터 원본이다.
Driver	odbcinst.ini에 설치된 드라이버 이름이다.

키워드	설명
PROTOCOL	연결 방식으로써 DA 또는 TCP 이다.
CS_MODE	Dedicated 모드로 접속할지 shared 모드로 접속할지를 설정한다. 이 설정을 사용하지 않으면 listener의 configuration (DEFAULT_CS_MODE)에 따라서 모드가 결정된다.
HOST	호스트의 IP 주소이다.
PORT	연결 포트 번호이다.
TCP_NODELAY	socket TCP_NODELAY 옵션이다.
UID	사용자 ID 이다.
PWD	사용자 password 이다.
CHARSET	Client character set 이다.
ALTERNATE_SERVERS	Failover가 발생할 경우, 연결을 시도하는 서버 리스트이며 각 서버는 콤마 (,) 로 구분한다. Failover 기능을 사용하지 않을 경우, ALTERNATE_SERVERS를 공백으로 설정한다.
CONNECTION_RETRY_COUNT	연결에 실패할 경우, 서버에 접속을 시도하는 횟수이다.
CONNECTION_RETRY_DELAY	연결에 실패할 경우, 서버에 접속을 시도하는 간격이다. (단위: 초)
FAILOVER_TYPE	- CONNECTION: 연결에 실패할 경우, ALTERNATE_SERVERS로 연결한다. - SESSION: 연결에 실패하거나 statement 동작 중 연결이 끊어졌을 경우, ALTERNATE_SERVERS로 연결한 후에 statement를 복원한다. 연결이 끊어질 때 진행 중인 트랜잭션이 없었을 경우, failover 후에 진행 중인 statement를 수행한다.
FAILOVER_GRANULARITY	0: Failover 진행 중에 에러가 발생해도 failover를 계속 진행한다. 1: Failover 진행 중에 SQLExeecute(), SQLExecDirect()를 제외한 에러가 발생할 경우 failover에 실패한다. 2: Failover 진행 중에 에러가 발생할 경우 failover에 실패한다.
DATE_FORMAT	DATE 타입 형식 문자열이다.
TIME_FORMAT	TIME 타입 형식 문자열이다.
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE 타입 형식 문자열이다.
TIMESTAMP_FORMAT	TIMESTAMP 타입 형식 문자열이다.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE 타입 형식 문자열이다.
CHAR_LENGTH_UNITS	SQLBindParameter()에서 ParameterType이 SQL_CHAR, SQL_VARCHAR일 경우, ColumnSize의 단위이다. - BYTE, OCTETS: 바이트 단위이다. - CHAR, CHARACTERS: 문자 단위이다.
ENABLE_SQLDESCRIBEPARAM	SQLDescribeParam() 동작 여부를 결정한다. 0: SQLDescribeParam()를 지원하지 않는다. 1: 모든 parameter에 대해 SQL_VARCHAR를 반환한다.
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK	SQLBindParameter()의 ColumnSize와 DecimalDigits를 검사할지 여부를 결정한다. 0: ColumnSize와 DecimalDigits를 검사하지 않는다. 1: ColumnSize와 DecimalDigits를 검사한다.

키워드	설명
USE_TARGETTYPE	통신으로 column 타입을 수신할 때 함께 수신할 타입 정보를 설정한다. 0: Column 타입만 수신한다. 1: Column 타입과 column 이름을 수신한다. 2: Column 타입과 column의 모든 정보를 수신한다.
LOCATOR_DSN	Location 정보를 지정하는 Data Source Name (DSN)이다.
LOCATOR_SERVICE	Service hint, glocator로부터 접속 정보를 얻는다.
LOCALITY_AWARE_TRANS ACTION	GLOBAL CONNECTION 사용 여부이다. 0: GLOBAL CONNECTION을 사용하지 않는다. 1: GLOBAL CONNECTION을 사용한다.
LOCALITY_GROUP_POLICY	GLOBAL CONNECTION을 사용할 때 선택할 수 있는 그룹이 없거나 둘 이상의 그룹을 선택할 수 있을 경우, 그룹 선택 방법을 설정한다. 0: 임의로 선택한다. 1: LOCALITY_GROUP_PATH 설정에 존재하는 그룹을 순서대로 선택한다. LOCALITY_GROUP_PATH에 있는 모든 그룹을 사용할 수 없는 경우에는 임의의 그룹을 선택한다. 2: 순서대로 선택한다. 매번 드라이버에 연결된 그룹 순서대로 선택한다.
LOCALITY_GROUP_PATH	GLOBAL CONNECTION을 사용할 때 선택할 수 있는 그룹이 한 개가 아니었을 때 선택한 그룹의 목록을 지정한다. 각 그룹은 콤마 (,)로 구분한다. 예: G1,G2,G3
LOCALITY_MEMBER_POLICY	GLOBAL CONNECTION을 사용할 때 선택된 그룹 내의 멤버를 선택하는 방법을 결정한다. 0: DML : MASTER / SELECT : MASTER 1: DML : ANY / SELECT : ANY 2: DML : MASTER / SELECT : ANY 3: DML : MASTER / SELECT : SLAVE 4: LOCALITY_MEMBER_PATH 설정에 존재하는 멤버를 순서대로 선택한다. LOCALITY_MEMBER_PATH에 존재하는 모든 멤버들을 사용할 수 없는 경우에는 선택된 그룹의 MASTER를 선택한다.
LOCALITY_MEMBER_PATH	GLOBAL CONNECTION을 사용할 때 선택된 그룹에서 사용할 멤버들의 목록을 지정한다. 각 멤버는 콤마 (,)로 구분한다. 예: G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
DB_HOME	데이터베이스의 홈 디렉토리를 설정한다. 기본값은 \$GOLDILOCKS_HOME 환경 변수를 사용한다.
PACKET_COMPRESSION_THRESHOLD	서버로 보낼 통신 데이터의 크기가 PACKET_COMPRESSION_THRESHOLD 보다 클 경우 통신 데이터를 압축한다. 설정값의 범위는 32 ~ 2113929216 이다.

표 25-4 Location

키워드	설명
FILE	Location file name
HOST	glocator ip address
PORT	glocator port number
CONNECTION_TIMEOUT	Connection timeout with glocator (second)
ALTERNATE_LOCATORS	glocator로부터 응답을 받지 못한 경우, ALTERNATE_LOCATORS를 이용하여 접속 정

키워드	설명
	보를 얻는다.

노트

- LOCATOR_DSN에 FILE과 HOST, PORT 속성이 모두 설정된 경우, FILE 속성을 우선적으로 적용한다. FILE에 대한 자세한 내용은 **Location File**을 참조한다.
- LOCATOR_SERVICE 속성은 LOCATOR_SERVICE에 속한 서버에 접속할 수 있게 한다. 접속한 서버 이외의 서버는 ALTERNATE_SERVERS가 된다.
FAILOVER_TYPE이 설정되지 않았다면 FAILOVER_TYPE은 session이 된다.
FAILOVER_GRANULARITY이 설정되지 않았다면 FAILOVER_GRANULARITY은 1이 된다.
자세한 내용은 **glocator**와 **gloctl**를 참조한다.

다음은 GOLDDILOCKS DSN을 구성하는 예이다.

```
[GOLDDILOCKS]
Driver = GOLDDILOCKS ODBC Driver
PROTOCOL = TCP
CS_MODE = SHARED
HOST = 192.168.0.10
PORT = 22581
CHARSET = UTF8
TCP_NODELAY = 1
ALTERNATE_SERVERS = (HOST=192.168.0.11:PORT=22581,HOST=192.168.0.12:PORT=22581)
CONNECTION_RETRY_COUNT = 3
CONNECTION_RETRY_DELAY = 1
FAILOVER_TYPE = SESSION
FAILOVER_GRANULARITY = 0
DATE_FORMAT = SYYYY-MM-DD
TIME_FORMAT = HH24:MI:SS.FF6
TIME_WITH_TIME_ZONE_FORMAT = HH24:MI:SS.FF6 TZH:TZM
TIMESTAMP_FORMAT = SYYYY-MM-DD HH24:MI:SS.FF6
TIMESTAMP_WITH_TIME_ZONE_FORMAT = SYYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM
CHAR_LENGTH_UNITS = CHARACTERS
```

```

ENABLE_SQLDESCRIBEPARAM = 1
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = 1
USE_TARGETTYPE = 0
PACKET_COMPRESSION_THRESHOLD = 2113929216
LOCALITY_AWARE_TRANSACTION = 0
LOCALITY_GROUP_POLICY = 0
LOCALITY_GROUP_PATH = G1,G2,G3
LOCALITY_MEMBER_POLICY = 0
LOCALITY_MEMBER_PATH = G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
LOCATOR_DSN = LOCATOR
LOCATOR_SERVICE = S1
[LOCATOR]
FILE = /home/test/.location.ini
HOST = 127.0.0.1
PORT = 42581
ALTERNATE_LOCATORS=(HOST=127.0.0.1:PORT=42582,HOST=127.0.0.1:PORT=42583)

```

Windows에서 DSN 구성

Windows에서는 ODBC 데이터 원본 관리자를 통해 DSN을 추가하거나 설정할 수 있다.

그림 3 Creating new data source

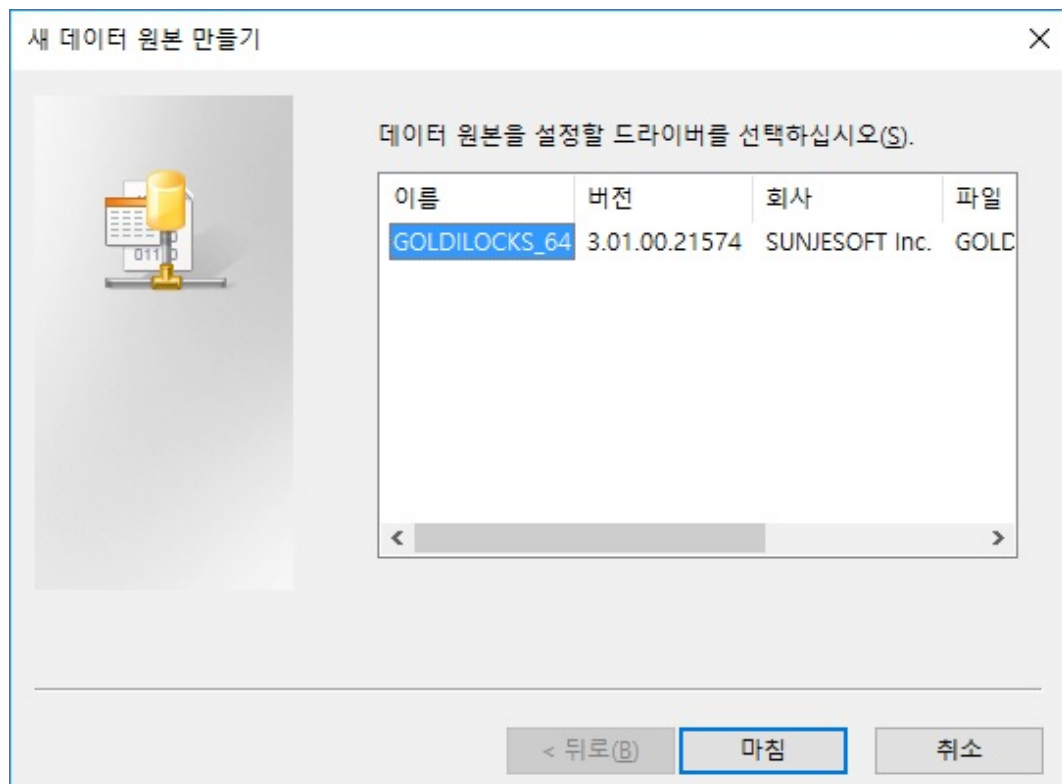


그림 4 ODBC driver configuration

GOLDILOCKS Driver Configuration

Connection

DSN

GOLDILOCKS

HOST

192.168.0.10

PORT

22581

UID

TEST

CS Mode

☒ Default

☐ Dedicated

☐ Shared

Failover

Connection Retry

Count

3

Delay

1

Type

☐ Connection

☒ Session

Granularity

☒ Non-atomic

☐ Atomic

AlternateServers

2.168.0.11:PORT=22581,HOST=192.168.0.12:PORT=22581

Format

DATE

YYYY-MM-DD HH24:MI:SS

TIME

HH24:MI:SS.FF6

TIME WITH TIME ZONE

HH24:MI:SS.FF6 TZh:TzM

TIMESTAMP

YYYY-MM-DD HH24:MI:SS.FF6

TIMESTAMP WITH TIME ZONE

YYYY-MM-DD HH24:MI:SS.FF6 TZh:TzM

Character Length Unit

☒ Octets

☐ Characters

OK

Cancel

각 항목은 다음과 같다.

표 25-5 DSN 구성 키워드

키워드	설명
DSN	데이터 원본 이름이다.
HOST	호스트의 IP 주소이다.
PORT	연결 포트 번호이다.
UID	사용자 ID 이다.
CS_MODE	Dedicated 모드로 접속할지 shared 모드로 접속할지를 설정한다. 설정하지 않을 경우, listener의 configuration (DEFAULT_CS_MODE)에 따라 default 모드가 결정된다.
ALTERNATE_SERVERS	Failover가 발생할 경우, 연결을 시도하는 서버 목록으로써 각 서버는 콤마 (,)로 구분한다. Failover 기능을 사용하지 않을 경우, ALTERNATE_SERVERS를 공백으로 설정한다.

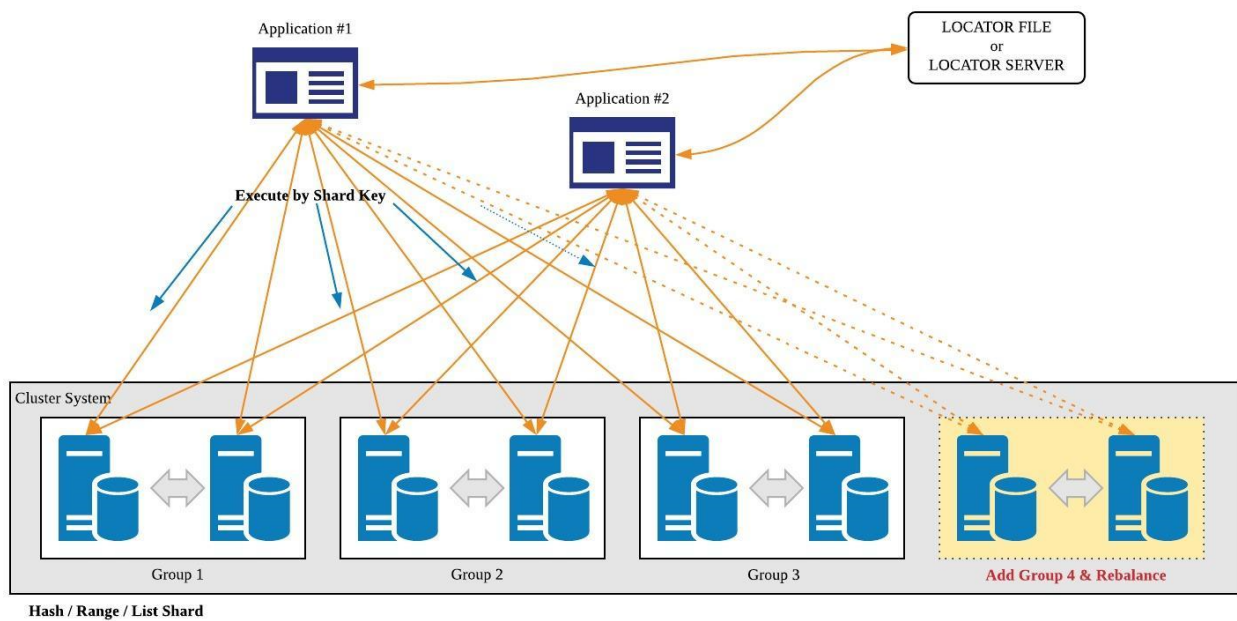
키워드	설명
CONNECTION_RETRY_COUNT	연결에 실패할 경우, 서버에 접속을 시도하는 횟수이다.
CONNECTION_RETRY_DELAY	연결에 실패할 경우, 서버에 접속을 시도하는 간격이다. (단위: 초)
FAILOVER_TYPE	• CONNECTION: 연결에 실패할 경우, ALTERNATE_SERVERS로 연결한다. • SESSION: 연결에 실패하거나 statement 동작 중 연결이 끊어졌을 경우, ALTERNATE_SERVERS로 연결한 후에 statement를 복원한다. 연결이 끊어질 때 진행 중인 트랜잭션이 없었을 경우, failover 후에 진행 중인 statement를 수행한다.
FAILOVER_GRANULARITY	• Non-atomic: Failover 진행 중에 에러가 발생해도 failover를 계속 진행한다. • Atomic: Failover 진행 중에 에러가 발생할 경우 failover에 실패한다.
DATE_FORMAT	DATE 타입 형식 문자열이다.
TIME_FORMAT	TIME 타입 형식 문자열이다.
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE 타입 형식 문자열이다.
TIMESTAMP_FORMAT	TIMESTAMP 타입 형식 문자열이다.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE 타입 형식 문자열이다.
CHAR_LENGTH_UNITS	SQLBindParameter()에서 ParameterType이 SQL_CHAR, SQL_VARCHAR일 경우 Column Size의 단위이다. • BYTE, OCTETS: 바이트 단위 • CHAR, CHARACTERS: 문자 단위

25.3 GLOBAL CONNECTION

클러스터 환경에서 응용 프로그램이 질의 처리에 적합한 노드를 선택해 수행할 수 있는 GLOBAL CONNECTION 기능을 지원한다.

특징

그림 5 GLOBAL CONNECTION

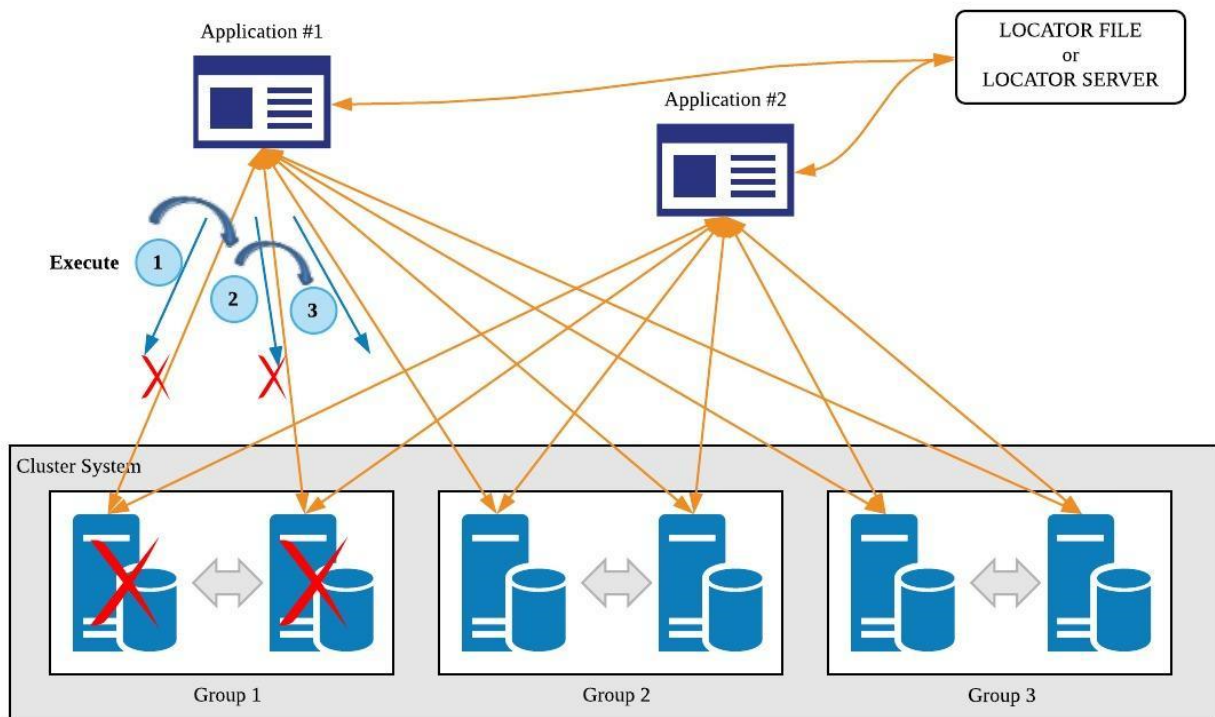


Hash, range, list sharding 방식 모두 global connection을 사용할 수 있는데 이 때 응용 프로그램을 변경할 필요는 없다.

GLOBAL CONNECTION을 사용할 경우 응용 프로그램이 클러스터 시스템의 모든 노드에 접속한다. SQL을 수행할 때 주어진 shard key 값에 따라 적합한 노드를 자동으로 선택하여 실행한다.

또한 online scale-out이 수행될 때, 사용자가 새로운 노드를 추가로 고려할 필요없이 응용 프로그램이 새 노드에 자동으로 접속해 해당 노드를 운영할 수 있다.

그림 6 GLOBAL CONNECTION HA (high availability)



SQL을 수행할 때 선택한 노드에 장애가 발생하면 같은 그룹의 다른 노드를 통해 SQL을 수행한다. 선택된 그룹의 모든 노드에 장애가 발생하면 다른 그룹을 통해 SQL을 수행한다.

장애가 발생한 노드를 복구하면 online 상태에서 해당 노드에 자동으로 다시 접속한다. 뿐만 아니라, 사용자가 다음 구문을 사용하여 응용 프로그램으로 하여금 다시 접속을 수행하게 할 수도 있다.

ALTER SYSTEM RECONNECT GLOBAL CONNECTION

설정

GLOBAL CONNECTION을 사용하려면 LOCALITY_AWARE_TRANSACTION 속성값과 함께 LOCATOR 파일이나 LOCATOR 서버를 설정해야 한다.

- DSN을 사용할 때의 .odbc.ini

```
[GOLDILOCKS]
HOST = 192.168.0.1
PORT = 22581
UID = TEST
PWD = test
LOCALITY_AWARE_TRANSACTION = 1
```

```

LOCATOR_DSN = LOCATOR
[LOCATOR]
FILE = /home/goldilocks/.location.ini

```

- 연결 문자열을 사용할 때

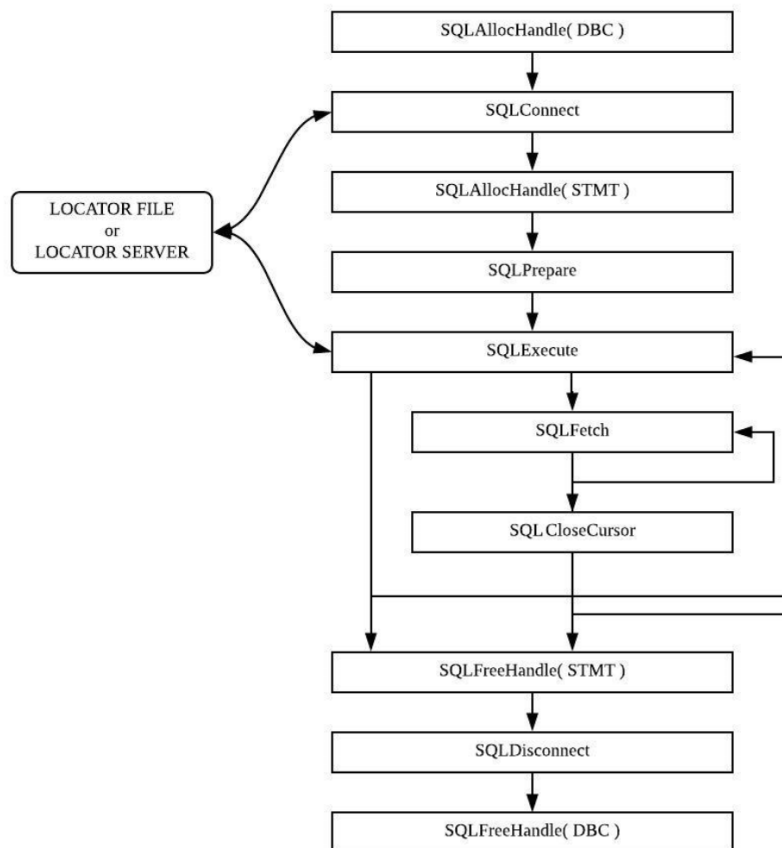
```

SQLDriverConnect( dbc,
                  NULL,
                  (SQLCHAR*)"HOST=192.168.0.1;PORT=22581;UID=TEST;PWD=test;LOCALITY_AWARE_
TRANSACTION=1;LOCATOR_HOST=192.168.0.2;LOCATOR_PORT=42581",
                  SQL_NTS,
                  NULL,
                  0,
                  NULL,
                  SQL_DRIVER_NOPROMPT );

```

GLOBAL CONNECTION 처리 과정

그림 7 Basic steps of applying GLOBAL CONNECTION



1. SQLAllocHandle (DBC)

Connection handle을 할당한다.

2. SQLConnect

사용자로부터 서버 정보를 입력 받은 서버로 연결하여 클러스터 시스템의 정보를 획득한 후, LOCATOR 파일 또는 LOCATOR 서버를 통해 클러스터 시스템 정보를 구축하며, 클러스터 시스템의 모든 노드에 접속한다.

3. SQLAllocHandle(STMT)

연결된 모든 노드에 각각 statement를 할당한다.

4. SQLPrepare

연결된 모든 노드에서 SQL을 실행할 준비를 한다.

5. SQLExecute

클러스터 시스템 정보가 구축되어 있지 않으면 응용 프로그램이 LOCATOR 파일이나 LOCATOR 서버를 통해 클러스터 시스템 정보를 구축하며, 클러스터 시스템의 모든 노드에 접속한다.

클러스터에 노드를 추가하여 새로운 노드에 접속한 경우, 다른 노드의 모든 statement들을 해당 노드에 동일하게 생성한 후에 SQL을 실행할 준비까지 한다.

Sharding key 정보가 이미 구축되어 있으면 응용 프로그램이 sharding key 정보를 이용하여 적합한 노드를 선택해 질의를 수행한다.

Sharding key 정보가 구축되어 있지 않으면 응용 프로그램이 임의의 서버로부터 해당 SQL의 sharding key 정보를 구축한 후, 적합한 노드를 선택하여 질의를 수행한다.

선택된 노드에 장애가 발생하면 해당 노드를 제외하고 적합한 노드를 다시 선택하여 질의를 수행한다.

SQLExecute 후에 sharding 정보가 변경되면 구축된 sharding key 정보를 삭제한다.

SQLExecute 후에 클러스터 노드가 추가/삭제되는 등 클러스터 시스템 정보가 변경되면 구축된 클러스터 시스템 정보를 삭제한다.

6. SQLFetch

SQL이 수행된 노드로부터 데이터를 가지고 온다.

7. SQLCloseCursor

SQL이 수행된 노드에서 커서를 닫는다.

8. SQLFreeHandle(STMT)

연결된 모든 노드에서 statement를 해제한다.

9. SQLDisconnect

모든 노드와의 접속을 해제한다.

10. SQLFreeHandle(DBC)

Connection handle을 해제한다.

제약 사항

Global connection을 사용할 때는 SQL 구문에 session dependancy를 가지는 객체 (session dependent object) 나 구문 (session dependent clause) 또는 함수 (session dependent function)를 사용할 수 없다.

또한 다수의 클러스터 노드에 접근하는 SQL들이 하나의 트랜잭션 안에 존재할 경우, 트랜잭션 내의 SQL들은 첫 번째 SQL의 sharding key가 선택한 클러스터 노드를 공통적으로 사용한다.

SQLExecDirect는 임의의 노드에서 수행된다.

Session Dependent Object

SQL 구문에서 session dependent object를 사용하는 경우 global connection을 지원하지 않는다.

- Global temporary table

Session Dependent Clause

SQL 구문에서 session dependent clause를 사용하는 경우 global connection을 지원하지 않는다.

- @domain 관련 구문 전체

Session Dependent Function과 Pseudo Column

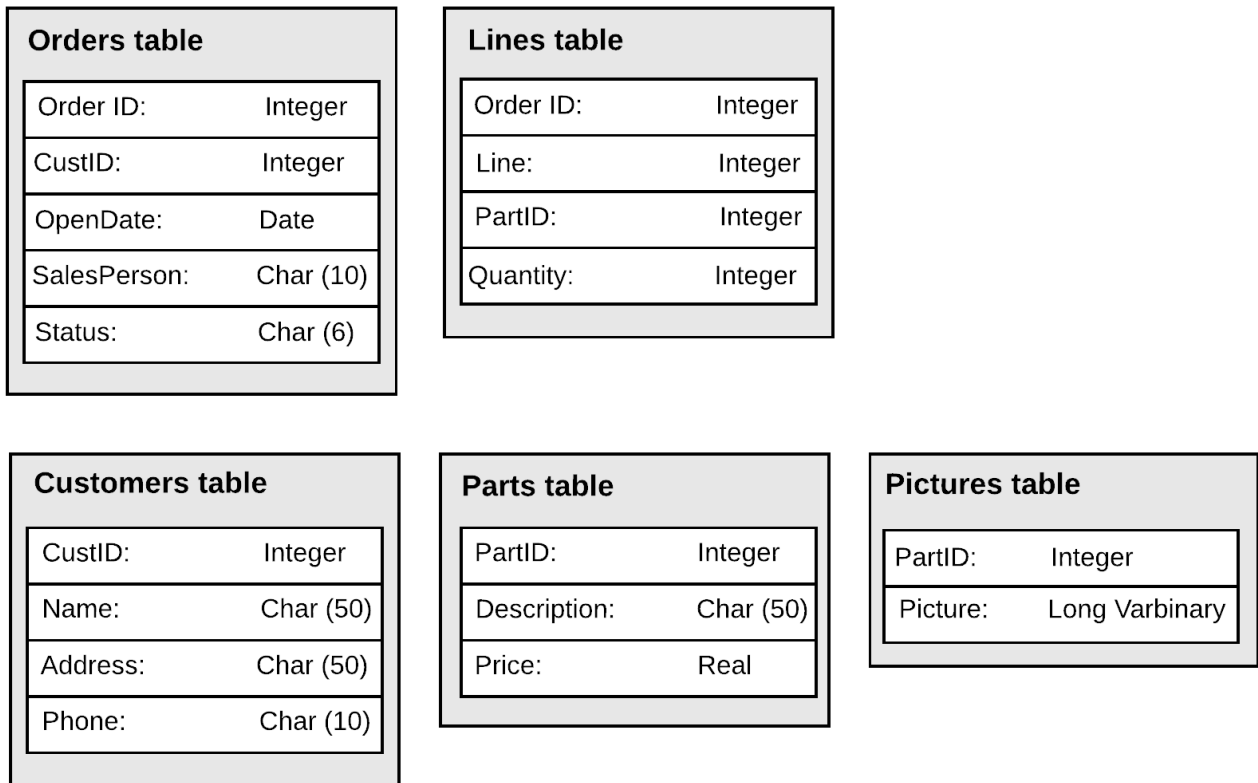
SQL 구문에서 session dependent 정보를 사용하는 경우 global connection을 지원하지 않는다.

- CURRVAL(sequence), sequence.CURRVAL
- UUID()
- VERSION()
- SESSION_ID()
- SESSION_SERIAL()
- USER_ID()
- LAST_IDENTITY_VALUE()
- STATEMENT_VIEW_SCN()
- STATEMENT_VIEW_SCN_GCN()
- STATEMENT_VIEW_SCN_DCN()
- STATEMENT_VIEW_SCN_LCN()
- LOCAL_GROUP_ID()
- LOCAL_MEMBER_ID()
- LOCAL_GROUP_NAME()
- LOCAL_MEMBER_NAME()

25.4 카탈로그 함수

모든 데이터베이스들은 데이터를 어떻게 저장하는지에 대한 구조체를 가지고 있다. 예를 들어 간단한 sales order 데이터베이스는 다음 그림과 같은 구조체를 가질 것이고 이 때, ID column들은 테이블들을 연결하는데 사용된다.

그림 8 Sales orders 구조체



이 구조체는 데이터베이스 카탈로그라고 불리는 시스템 테이블들의 set 내에 권한 등과 같은 다른 정보와 함께 저장되어 있다. 이것은 데이터 디렉터리라고도 불린다.

응용 프로그램은 카탈로그 함수들을 호출하여 이 구조체를 발견할 수 있다. 카탈로그 함수들은 결과 집합 안에 정보를 반환하고 일반적으로 카탈로그 안에 테이블에 대한 SELECT 명령문들을 통해 구현된다.

카탈로그 데이터 사용

응용 프로그램은 다양한 방법으로 카탈로그 데이터를 사용하는데 다음은 몇 가지 공통된 사용법이다.

SQL 명령문들을 실행 시점에 구성

주문 입력 응용 프로그램 같은 수직 응용 프로그램은 하드 코딩된 SQL 명령문들을 가지고 있다. 응용 프로그램에서 사용된 테이블들과 column들은 사전에 고정되어 있고 응용 프로그램은 이런 테이블들에 접근한다. 예를 들어 주문 입력 응용 프로그램은 시스템에 새로운 주문들을 추가하기 위해 매개 변수화 된 하나의 명령문을 가지고 있다.

ODBC를 사용하여 데이터를 회수하는 스프레드시트 프로그램 같은 일반적인 응용 프로그램은 때에 따라 사용자로부터의 입력을 기반으로 실행 시점에 SQL 명령문들을 구성한다. 이러한 응용 프로그램은 사용자에게 테이블들과 column들을 사용하기 위한 형식을 요구할 수 있다. 이 때, 사용자가 선택한 테이블들이나 column들의 목록을 응용 프로그램에서 보여주면 사용자가 보다 쉽게 사용할 수 있다. 이런 목록들을 구축하기 위해 응용 프로그램은 SQLTables와 SQLColumns 카탈로그 함수들을 호출할 것이다.

개발 중 SQL 명령문 구성

일반적으로 응용 프로그램 개발 환경들은 프로그램을 개발하는 동안 개발자가 데이터베이스 쿼리를 생성하는 것을 허용한다. 그 후, 쿼리들은 응용 프로그램에 하드코딩되어 내장된다.

또한 이러한 환경에서는 SQLTables와 SQLColumns를 사용하여 개발자가 선택한 것들의 목록을 만들 수 있다. 이런 환경들은 SQLPrimaryKeys와 SQLForeignKeys를 사용하여 자동적으로 선택된 테이블들간의 관계를 알아내고 보여 준다. 그리고 SQLStatistics를 사용하여 인덱스 필드들을 알아내고 강조하여 개발자가 쿼리를 효과적으로 생성할 수 있게 한다.

커서 구성

스크롤 커서를 제공하는 응용 프로그램, 드라이버, 미들웨어는 SQLSpecialColumns를 사용하여 row를 유일하게 식별하는 column(s)을 알아낸다. 프로그램은 회수된 각 row에 대한 이런 column들의 값들을 포함하는 keyset을 구축할 수 있다. 응용 프로그램은 이러한 값들을 사용하여 row를 향해 뒤로 스크롤하여 row에 대한 가장 최근의 데이터를 회수한다.

ODBC 카탈로그 함수

ODBC는 다음과 같은 카탈로그 함수들을 포함한다.

표 25-6 ODBC 카탈로그 함수

함수	설명
SQLTables	데이터 소스에 있는 카탈로그, 스키마, 테이블 또는 테이블 형식들의 목록을 반환한다.
SQLColumns	한 개 이상의 테이블들의 column들 목록을 반환한다.
SQLStatistics	한 개 테이블에 대한 통계 목록을 반환한다. 또한 그 테이블에 연결된 인덱스들의 목록도 반환한다.
SQLSpecialColumns	한 개 테이블에서 유일하게 row를 식별하는 column들의 목록을 반환한다. 또한 그 테이블 내 column들의 목록을 반환하는데 그것들은 자동으로 갱신된다.
SQLPrimaryKeys	한 개 테이블의 primary key를 구성하는 column들의 목록을 반환한다.

함수	설명
SQLForeignKeys	한 개 테이블 내의 foreign key들의 목록 또는 그 테이블을 참조하는 다른 테이블 내의 foreign key들의 목록을 반환한다.
SQLTablePrivileges	한 개 이상의 테이블들과 관련된 권한들의 목록을 반환한다.
SQLColumnPrivileges	한 개 테이블에 하나 이상의 column들과 관련된 권한들의 목록을 반환한다.
SQLProcedures	드라이버에서 지원하지 않는다.
SQLProcedureColumns	드라이버에서 지원하지 않는다.
SQLGetTypeInfo	데이터 소스에서 지원하는 SQL 데이터 형식들의 목록을 반환한다. 이러한 데이터 형식들은 일반적으로 CREATE TABLE과 ALTER TABLE 명령문들에 사용된다.

카탈로그 함수의 데이터 반환

각 카탈로그 함수는 결과 집합처럼 데이터를 반환한다. 이 결과 집합은 다른 어떤 결과 집합과도 다르지 않다. 이것은 보통 드라이버에 하드코딩되어 있거나, 데이터 소스의 procedure 안에 저장된 매개 변수화 된 SELECT 명령문처럼 미리 정의되어 있는 것에 의해 생성된다.

각 카탈로그 함수에 대한 결과 집합은 이 사용 설명서 내의 해당 함수에 대한 참조 문단에 기술되어 있다. 결과 집합은 리스팅된 column들 외에도 추가적으로 드라이버에서 명시한 column들을 마지막으로 선택된 column 이후에 포함할 수 있다. 이러한 column들은 드라이버 사용 설명서에 설명되어 있다.

응용 프로그램은 결과 집합의 마지막을 기준으로 드라이버에서 명시한 column을 바인딩한다. 그들은 드라이버에서 명시한 column의 수를 요구되는 column 이후에 발생할 column의 수보다 작은 마지막 column의 수처럼 계산한다. 이것은 새로운 column을 미래의 ODBC 버전이나 드라이버에 추가할 때 응용 프로그램을 변경해야 하는 수고를 덜어 준다. 이 스키마를 작동시키려면 오래된 드라이버에 명시된 column들의 앞에 새로운 드라이버에 명시된 column들을 추가하여 column의 숫자가 결과 집합의 마지막을 기준으로 변경되는 것을 방지해야 한다.

그것들이 특수 문자들을 포함하더라도 결과 집합 내에 반환되는 식별자들은 인용하지 않는다.

예를 들어 (드라이버에서 명시하고 SQLGetInfo를 통해 반환되는) 식별자 인용문자가 double quote (")인 Accounts Payable 테이블이 Customer Name이라는 이름의 column을 포함할 경우, 이 column에 대해 SQLColumns가 반환하는 row에서 TABLE_NAME column의 값은 Accounts Payable이지 "Accounts Payable"이 아니며 COLUMN_NAME column의 값은 Customer Name 이지 "Customer Name" 이 아니다.

응용 프로그램은 Accounts Payable 테이블에서 고객들의 이름을 회수하기 위해 다음과 같이 인용할 것이다.

```
SELECT "Customer Name" FROM "Accounts Payable"
```

카탈로그 함수는 사용자 이름과 암호를 기반으로 연결된 SQL-like 승인 모델을 기반으로 하고 권한을 가진 사용자에게만 데이터를 반환한다. 이 모델에 맞지 않는 각 파일들의 암호 보호는 드라이버에서 정의한다.

카탈로그 함수들이 반환하는 결과 집합은 대부분 갱신할 수 없고 응용 프로그램이 이 결과 집합들 내의 데이터를 갱신한다고해도 데이터베이스의 구조체를 변경할 수 있는 것은 아니다.

카탈로그 함수의 인자

패턴값 인자

SQLTables의 TableName 인자처럼, 카탈로그 함수들의 몇몇 인자들은 검색 패턴을 받아들인다. SQL_ATTR_METADATA_ID 명령문 속성이 SQL_FALSE로 설정되어 있을 경우, 이 인자들이 검색 패턴을 받아들인다. 이것들은 이 속성이 SQL_TRUE로 설정되어 있을 경우, 검색 패턴을 받아들이지 않는 식별자 인자들이다.

검색 패턴 문자들은 다음과 같은 특징을 갖는다.

- 언더스코어 (_)는 어떠한 하나의 문자를 대표한다.
- 퍼센트 사인 (%)은 0 개 이상의 어떠한 문자를 대표한다.
- 이스케이프 문자는 드라이버에서 명시되고 퍼센트 사인, 언더스코어, 이스케이프 문자를 있는 그대로 포함하기 위해 사용된다. 이스케이프 문자가 특수 문자가 아닌 것 앞에 있을 경우, 이스케이프 문자는 특별한 의미를 가지지 않지만 특수 문자 앞에 있을 경우의 이스케이프 문자는 특수 문자이다. 예를 들어 "Wa" 는 "W" d와 "a" 처럼 두 개의 문자로 취급되지만 "W%" 는 특별하지 않은 하나의 "%" 로 취급될 뿐이다.

이스케이프 문자는 SQLGetInfo에서 SQL_SEARCH_PATTERN_ESCAPE 옵션을 사용하여 회수된다. 검색 패턴을 받아들이는 인자에 문자를 있는 그대로 받아들이게 하려면 언더스코어, 퍼센트 사인 또는 이스케이프 문자보다 앞에 있어야 한다.

다음 표는 검색 패턴을 사용하는 예이다.

검색 패턴	설명
%A%	A를 포함하는 모든 식별자이다.
ABC_	ABC로 시작하는 모든 네 개의 문자이다.
ABCW_	이스케이프 문자를 백슬러쉬 (\)로 가정하고 식별자는 ABC_ 이다.
WW%	이스케이프 문자를 백슬러쉬 (\)로 가정하고 식별자는 백슬러쉬 (\)로 시작한다.

주의

검색 패턴을 받아들이는 인자들에 이스케이프 검색 패턴 문자들을 사용할 때는 각별히 유의해야 한다. 이것은 일반적으로 식별자로 사용되는 언더스코어 (_)에 대해 특히나 TRUE이다.

응용 프로그램에서 하나의 카탈로그 함수에서 회수한 값을 그대로 다른 카탈로그 함수의 검색 패턴 인자에 넘기는 실수가 많이 일어난다.

예를 들어, 응용 프로그램이 SQLTables에 대한 결과 집합에서 이름이 MY_TABLE인 테이블을 회수하여 MY_TABLE에 있는 column들의 리스트를 회수하기 위해 SQLColumns에 넘겼을 경우, 응용 프로그램이 MY_TABLE에 대한 column들을 얻는 것이 아니라 MY_TABLE, MY1TABLE, MY2TABLE과 같이 검색 패턴 MY_TABLE에 매칭되는 모든 테이블에 대한 column을 얻게 된다.

노트

ODBC 2.x 드라이버는 SQLTables의 CatalogName 인자에 검색 패턴을 지원하지 않는다. ODBC 3.x 드라이버는 SQL_ATTR_ODBC_VERSION 환경 속성을 SQL_OV_ODBC3로 설정한 경우, 이 인자에 검색 패턴을 받아들인다. 이 속성을 SQL_OV_ODBC2로 설정하면 검색 패턴을 받아들이지 않는다.

검색 패턴 인자에 NULL 포인터를 넘기더라도 그 인자에 검색을 강요하지 않는다. NULL 포인터와 검색 패턴 % (아무 문자)는 같은 의미이다. 그러나 길이가 0 인 검색 패턴은 빈 문자열 ("")과 매칭된다.

25.5 ODBC API References

SQLAllocConnect

적합성

도입된 버전: ODBC 1.0

개요

SQLAllocConnect 함수는 ODBC 3.x에서 SQLAllocHandle 함수로 대체되었다.
자세한 내용은 **SQLAllocHandle** 함수를 참조한다.

구문

```
SQLRETURN SQLAllocConnect(  
    SQLHENV  EnvironmentHandle,  
    SQLHDBC * ConnectionHandlePtr);
```

인자

EnvironmentHandle

[입력] 환경 핸들이다.

ConnectionHandlePtr

[출력] 새로 할당 받을 연결 핸들의 포인터이다.

SQLAllocEnv

적합성

도입된 버전: ODBC 1.0

개요

SQLAllocEnv 함수는 ODBC 3.x에서 SQLAllocHandle 함수로 대체되었다.
자세한 내용은 **SQLAllocHandle** 함수를 참조한다.

구문

```
SQLRETURN SQLAllocEnv(  
    SQLHENV * EnvironmentHandlePtr);
```

인자

EnvironmentHandlePtr

[출력] 새로 할당받을 환경 핸들의 포인터이다.

SQLAllocHandle

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLAllocHandle은 환경, 연결, 또는 명령문 핸들을 할당한다.

구문

```
SQLRETURN SQLAllocHandle(  
    SQLSMALLINT    HandleType,  
    SQLHANDLE       InputHandle,  
    SQLHANDLE *     OutputHandlePtr);
```

인자

HandleType

[입력] SQLAllocHandle이 할당할 핸들의 유형으로 SQL_HANDLE_DBC, SQL_HANDLE_ENV, SQL_HANDLE_STMT 중 하나여야 한다.

InputHandle

[입력] HandleType이 SQL_HANDLE_ENV이면 SQL_NULL_HANDLE 이다.

HandleType이 SQL_HANDLE_DBC이면 환경 핸들이어야 한다.

SQL_HANDLE_STMT 이면 연결 핸들이어야 한다.

OutputHandlePtr

[출력] 새로 할당받을 핸들의 포인터이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR.

진단

SQLSTATE	Error	설명
08003	Connection not open	연결되어 있지 않은 상태에서 HandleType 인자가 SQL_HANDLE_STMT, SQL_HANDLE_DESC이다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	OutputHandlePtr 인자가 null 포인터이다.
HY010	Function sequence error	HandleType 인자가 SQL_HANDLE_DBC 이며, SQL_ODBC_VERSION 속성을 설정하는 SQLSetEnvAttr을 호출하지 않았다.
HY014	Limit on the number of handles exceeded	할당 핸들 개수를 제한한다.
HY092	Invalid attribute/option identifier	HandleType 인자가 SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, SQL_HANDLE_DESC이 아니다.
IM001	Driver does not support this function	HandleType 인자가 SQL_HANDLE_DESC이다.

설명

SQLAllocHandle은 환경, 연결, 명령문, 설명자를 위한 핸들을 할당하는데 사용된다. *OutputHandlePtr을 사용하여 SQLAllocHandle을 사용하는 경우 드라이버가 해당 핸들에 정보를 덮어쓴다. 드라이버 관리자는 *OutputHandlePtr에 포함된 핸들이 이미 사용되었는지 여부를 확인할 수 없고, 핸들이 덮어쓴 이전 정보도 알 수 없다.

환경 핸들 할당

환경 핸들은 연결 핸들이 유효한지, 활성화되어 있는지와 같은 전역 정보를 제공한다.

환경 핸들을 요청하려면 응용 프로그램은 HandleType이 SQL_HANDLE_ENV, InputHandle이 SQL_NULL_HANDLE 인 SQLAllocHandle을 호출한다. 드라이버는 환경 정보를 위한 메모리를 할당하고, 다시 *OutputHandle 인자로 할당된 핸들을 전달한다. 응용 프로그램은 환경 핸들 인자를 필요로 하는 호출에 *OutputHandle 값을 전달한다.

환경 핸들을 할당한 후, 응용 프로그램은 SQLSetEnvAttr을 호출하여 SQL_ATTR_ODBC_VERSION 속성을 설정해야 한다. 만약 연결 핸들을 할당하기 위해 SQLAllocHandle을 호출할 때 이 속성이 설정되지 않은 경우, SQLSTATE HY010 (Function sequence error)가 반환된다.

연결 핸들 할당

연결 핸들은 유효한 명령문인지, 접속되어 있는 설명자 핸들인지 여부 및 트랜잭션이 현재 열려 있는지와 같은 정보를 제공한다.

연결 핸들을 요청하기 위해 응용 프로그램은 HandleType이 SQL_HANDLE_DBC인 SQLAllocHandle을 호출한다. I

InputHandle 인자는 SQLAllocHandle을 호출해 반환된 환경 핸들로 설정한다. 드라이버는 연결 정보를 위한 메모리를 할당하고, 다시 *OutputHandle 인자로 할당된 핸들을 전달한다. 응용 프로그램은 연결 핸들 인자를 필요로 하는 호출에 *OutputHandle 값을 전달한다.

연결 핸들을 할당하는 SQLAllocHandle을 호출하기 전에 SQL_ATTR_ODBC_VERSION 환경 속성이 설정되어 있지 않은 경우, SQLSTATE HY010 (Function sequence error)가 반환된다.

명령문 핸들 할당

명령문 핸들은 에러 메시지, 커서 이름 및 SQL 문 처리 상태와 같은 정보를 제공한다.

명령문 핸들을 요청하기 위해 응용 프로그램이 데이터 소스에 연결한 다음, SQL 문을 전송하기 전에 SQLAllocHandle을 호출한다. 이 호출에서 HandleType은 SQL_HANDLE_STMT로 설정해야 하고, InputHandle은 SQLAllocHandle을 호출해 반환된 연결 핸들로 설정한다. 드라이버는 명령문 정보를 위한 메모리를 할당하고, 다시 *OutputHandle 인자로 할당된 핸들을 전달한다. 응용 프로그램은 명령문 핸들 인자를 필요로 하는 호출에 *OutputHandle 값을 전달한다.

명령문 핸들이 할당되면, 드라이버가 자동으로 네 개의 설명자 집합을 할당하고, 이 설명자 핸들들을 SQL_ATTR_APP_ROW_DESC, SQL_ATTR_APP_PARAM_DESC, SQL_ATTR_IMP_ROW_DESC 및 SQL_ATTR_IMP_PARAM_DESC 명령문 속성에 할당한다. 이를 암묵적 설명자 할당이라고 한다.

SQLAllocStmt

적합성

도입된 버전: ODBC 1.0

개요

SQLAllocStmt 함수는 ODBC 3.x에서 SQLAllocHandle 함수로 대체되었다.
자세한 내용은 **SQLAllocHandle** 함수를 참조한다.

구문

```
SQLRETURN SQLAllocStmt(  
    SQLHDBC      ConnectionHandle,  
    SQLHSTMT * StatementHandlePtr);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

StatementHandlePtr

[출력] 새로 할당받을 명령문 핸들의 포인터이다.

SQLBindCol

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLBindCol은 응용 프로그램의 데이터 버퍼를 결과 집합의 column들에 바인딩한다.

구문

```
SQLRETURN SQLBindCol(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLSMALLINT    TargetType,
    SQLPOINTER     TargetValuePtr,
    SQLLEN         BufferLength,
    SQLLEN *       StrLen_or_Ind);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ColumnNumber

[입력] 바인드 할 결과 집합의 column 번호. Column은 1부터 시작하는 오름차순으로 번호가 매겨져 있다.

TargetType

[입력] *TargetValuePtr 버퍼의 C 데이터 형식의 식별자. SQLFetch, SQLFetchScroll 및 SQLSetPos를 사용하여 데이터를 검색할 때, 드라이버는 이 타입으로 데이터를 변환한다.

TargetType 인자가 interval 데이터 타입일 경우, 기본값은 interval leading precision (2), interval second s precision (6) 이며, ARD의 SQL_DESC_DATETIME_INTERVAL_PRECISION 및 SQL_DESC_PRECISION 필드 각각에 설정된다. TargetType 인자가 SQL_C_NUMERIC일 경우, 기본값은 precision (38), scale (0)이며, ARD의 SQL_DESC_PRECISION 및 SQL_DESC_SCALE 필드 각각에 설정된다. 만약 기본 precision 및 scale이 적절하지 않은 경우, 응용 프로그램은 SQLSetDescField 또는 SQLSetDescRec를 호출하여 해당 설명자 필드를 명시적으로 설정해야 한다.

TargetValuePtr

[지연된 입력/출력] Column에 바인딩 할 데이터 버퍼 포인터. SQLFetch 및 SQLFetchScroll은 이 버퍼에 데이터를 반환한다.

TargetValuePtr이 null 포인터일 경우, 드라이버는 column에 대한 데이터 버퍼 바인딩을 해제한다. 응용 프로그램은 SQL_UNBIND 옵션으로 SQLFreeStmt를 호출하여 모든 column의 바인딩을 해제할 수 있다. 응용 프로그램은 TargetValuePtr 인자를 null 포인터로 하여, SQLBindCol을 호출해 column에 대한 데이터 버퍼 바인딩을 해제할 수 있지만 StrLen_or_IndPtr 인자가 유효한 값이면 column에 대한 길이/ 지시자 버퍼는 여전히 바인딩 되어 있다.

BufferLength

[입력] *TargetValuePtr 버퍼의 바이트 단위 길이

드라이버는 문자나 이진 데이터와 같은 가변 길이 데이터를 반환할 때, *TargetValuePtr 버퍼의 끝을 넘어 쓰지 않도록 하기 위해 BufferLength를 사용한다. *TargetValuePtr에 문자 데이터를 반환할 때 드라이버는 null 종료 문자도 계산하는 것에 주의해야 한다. 따라서 *TargetValuePtr은 null 종료 문자를 위한 공간을 포함해야 하며, 그렇지 않으면 드라이버가 데이터를 자를 수 있다.

드라이버는 정수 또는 날짜 구조체와 같은 고정 길이 데이터를 반환할 때, 버퍼가 데이터를 저장하기에 충분히 큰 것으로 가정해 BufferLength를 무시한다. 그러므로 응용 프로그램은 고정 길이 데이터에 충분히 큰 버퍼를 할당하는 것이 중요하며, 그렇지 않으면 드라이버는 버퍼의 끝을 넘어 쓸 수 있다.

StrLen_or_IndPtr

[지연된 입력/ 출력] Column에 바인딩 할 길이/ 지시자 버퍼에 대한 포인터. SQLFetch 및 SQLFetchScroll은 이 버퍼에 값을 반환한다.

SQLFetch 및 SQLFetchScroll은 길이/ 지시자 버퍼에 반환 가능한 데이터의 길이, SQL_NO_TOTAL, SQL_NULL_DATA를 반환한다.

지시자 버퍼와 길이 버퍼가 별도의 버퍼일 경우, 길이 버퍼는 모든 값을 반환할 수 있는 반면, 지시자 버퍼는 오직 SQL_NULL_DATA만 반환할 수 있다.

StrLen_or_IndPtr이 null 포인터인 경우, 길이/ 지시자 값은 사용되지 않으며, null 데이터를 가져올 때 에러가 발생한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR.

진단

SQLSTATE	Error	설명
07006	Restricted data type attribute violation	ColumnNumber 인자가 0이고 TargetType 인자가 SQL_C_BOOKMARK 또는 SQL_C_VARBOOKMARK가 아니다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
	Memory allocation	

SQLSTATE	Error	설명
HY001	n error	메모리 할당 에러이다.
HY003	Invalid application buffer type	TargetType 인자값이 유효한 데이터 타입이 아니다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	BufferLength 인자값이 0보다 작다.
HYC00	Optional feature not implemented	드라이버가 해당 column의 SQL 데이터 타입과 TargetType 인자값 및 변환을 지원하지 않는다. ColumnNumber 인자값이 0이고, 드라이버는 책갈피를 지원하지 않는다.

설명

SQLBindCol은 응용 프로그램의 데이터 버퍼 및 길이/지시자 버퍼에 결과 집합의 column을 바인딩 하는데 사용된다. 응용 프로그램에서 데이터를 가져오기 위해 SQLFetch와 SQLFetchScroll을 호출하면, 드라이버는 지정된 버퍼에 바인딩된 column 데이터를 반환한다.

응용 프로그램은 column을 바인딩하지 않고, SQLGetData를 호출하여 데이터를 검색할 수 있다.

Column 바인딩

응용 프로그램은 column을 바인딩하기 위해 SQLBindCol을 호출하여, column 번호, 타입, 주소 및 데이터 버퍼의 길이와 길이/지시자 버퍼의 주소를 전달한다.

응용 프로그램에서 SQLBindCol 할 때 버퍼를 바인딩하지만, 드라이버는 SQLFetch와 SQLFetchScroll을 호출할 때 접근하므로, 이들 버퍼는 지연되어 사용된다. 따라서 응용 프로그램은 SQLBindCol에서 설정한 포인터들이 데이터를 반환할 때까지 유효하도록 해야 한다. 만약 응용 프로그램이 버퍼 해제처럼 이 포인터를 유효하지 않게 한 후에 호출한다면, 결과가 올바르지 않다.

바인딩은 새로운 바인딩으로 대체되거나, column 바인딩을 해제하거나, 명령문이 해제될 때까지 유지된다.

Column 바인딩 해제

한 개 column의 바인딩만 해제하려면, 응용 프로그램에서 해제할 column의 번호를 ColumnNumber로 설정하고, TargetValue를 null 포인터로 설정하여 SQLBindCol을 호출하면 된다. 만약 ColumnNumber가 바인딩 해제된 column 번호일 경우, SQLBindCol은 계속 SQL_SUCCESS를 반환한다.

모든 column의 바인딩을 해제하려면, 응용 프로그램이 option인 SQL_UNBIND를 설정하여 SQLFreeStmt를 호출하면 된다. 또한 ARD의 SQL_DESC_COUNT 필드를 0으로 설정하여 모든 column의 바인딩을 해제할 수도 있다.

Column 재 바인딩

응용 프로그램은 바인딩을 변경하기 위해 다음 두 연산 중 하나를 수행할 수 있다.

- SQLBindCol을 호출하여 이미 바인딩된 column에 대해 새로운 바인딩을 명시한다.

- SQLBindCol을 호출하여 명시된 버퍼 주소에 offset을 추가한다. 자세한 정보는 다음 문단인 **바인딩 Offset**을 참조한다.

바인딩 Offset

바인딩 offset은 (TargetValuePtr 및 StrLen_or_IndPtr에 명시된) 데이터 및 길이/ 지시자 버퍼가 역참조 되기 전, 주소에 추가되는 값이다.

바인딩 Offset을 이용하는 것은 기본적으로 SQLBindCol을 호출하여 column을 다시 바인딩하는 것과 같은 효과가 있다. 차이점이 있다면 새로 SQLBindCol을 호출할 경우, 데이터 및 길이/ 지시자 버퍼의 새로운 주소를 명시하는 반면에 바인딩 offset은 주소를 갱신하지 않고 주소에 offset만 추가한다는 것이다. 응용 프로그램은 원하면 언제든지 새로운 offset을 명시할 수 있고, 이 offset은 항상 기본 바인딩 주소에 추가된다. 특별히 offset을 0으로 설정하거나 명령문 속성을 NULL 포인터로 설정하면 드라이버는 기본 바인딩 주소를 사용한다.

기본 바인딩 주소와 offset의 합은 반드시 유효한 주소여야 하지만, 추가될 offset 주소가 반드시 유효할 필요는 없다.

바인딩 배열

Row 집합 크기 (SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성의 값)가 1 보다 크면 응용 프로그램은 한 개의 버퍼 대신 버퍼 배열을 바인딩한다.

응용 프로그램은 다음과 같은 두 가지 방법을 사용하여 배열을 바인딩 할 수 있다.

- 각 column에 배열을 바인딩한다. 이는 각 데이터 구조체 (배열)가 하나의 column에 대한 데이터를 포함하기 때문에 **Column 방향 바인딩** 처럼 참조한다.
- Row 데이터 전체를 포함할 수 있는 구조체를 정의하고 이 구조체들의 배열을 바인딩한다. 이것은 각 데이터 구조체가 row 하나에 대한 데이터를 포함하기 때문에 **Row 방향 바인딩** 처럼 참조한다.

각 버퍼 배열은 적어도 row 집합 크기 만큼의 원소들을 가지고 있어야 한다.

Column 방향 바인딩

Column 방향 바인딩에서 응용 프로그램은 독립된 데이터 및 길이/지시자 배열을 각 column에 바인드한다.

Column 방향 바인딩을 사용하기 위해 응용 프로그램은 우선 SQL_ATTR_ROW_BIND_TYPE 명령문 속성을 SQL_BIND_BY_COLUMN으로 설정한다 (기본값). 바인딩 될 각 column에 대해 응용 프로그램은 다음 과정들을 수행한다.

1. 데이터 버퍼 배열을 할당한다.
2. 길이/ 지시자 버퍼들의 배열을 할당한다.

노트

Column 방향 바인딩이 사용될 때 응용 프로그램이 설명자들에 직접 기록할 경우, 독립된 배열들은 길이와 지시자 데이터에 사용될 수 있다.

3. 다음 인자들과 함께 SQLBindCol을 호출한다.

- TargetType은 데이터 버퍼 배열에 있는 각 인자의 형식이다.
- TargetValuePtr은 데이터 버퍼 배열의 주소이다.
- BufferLength는 데이터 버퍼 배열에 있는 각 원소의 크기이다. BufferLength 인자는 데이터가 고정 길이 데이터인 경우 무시된다.
- StrLen_or_IndPtr은 길이/ 지시자 배열의 주소이다.

Row 방향 바인딩

Row 방향 바인딩에서 응용 프로그램은 바인딩 될 각 column에 대한 데이터 및 길이/ 지시자 버퍼를 포함하는 구조체를 정의한다.

Row 방향 바인딩을 사용하기 위해 응용 프로그램은 다음 과정들을 수행한다.

1. (데이터 및 길이/ 지시자 버퍼 모두를 포함하는) 데이터의 한 개 row를 포함할 수 있는 구조체를 정의하고 이 구조체들의 배열을 할당한다.

노트

Row 방향 바인딩이 사용될 때 응용 프로그램이 설명자들에 직접 기록할 경우, 독립된 필드들은 길이와 지시자 데이터에 사용될 수 있다.

2. SQL_ATTR_ROW_BIND_TYPE 명령문 속성은 데이터의 한 개 row를 포함하는 구조체의 크기나 바인딩 될 결과 column들에 대한 버퍼의 인스턴스 크기를 설정한다. 해당 길이들은 반드시 바인딩 된 모든 column의 공간과 구조체 또는 버퍼의 패딩을 포함해야 하며 바인딩 된 column의 주소가 명시된 길이만큼 증가했을 때 다음 row의 같은 column의 시작 위치를 가리킬 것을 보장해야 한다. ANSI C에서는 sizeof 연산자를 사용하여 이를 보장한다.

3. 바인딩 될 각 column에 대해 다음 인자들과 함께 SQLBindCol 을 호출한다.

- TargetType은 column에 바인딩 될 데이터 버퍼 멤버의 형식이다.
- TargetValuePtr은 첫 번째 배열 인자에 있는 데이터 버퍼 멤버의 주소이다.
- BufferLength는 데이터 버퍼 멤버의 크기이다.
- StrLen_or_IndPtr은 바인딩 될 길이/ 지시자 멤버의 주소이다.

버퍼 주소

버퍼 주소는 데이터 또는 길이/ 지시자 버퍼의 실제 주소이다. 드라이버는 (데이터를 회수할 때처럼) 버퍼에 기록하기 전에 버퍼 주소를 계산한다. 이것은 다음 공식을 사용하여 계산하는데 이 공식은 TargetValuePtr 및 StrLen_or_IndPtr 인자들에 명시된 주소, 바인딩 offset, row 번호를 사용한다.

$\text{Bound Address} + \text{Binding Offset} + ((\text{Row Number} - 1) \times \text{Element Size})$

공식들의 변수들은 다음과 같이 정의한다.

표 25-7 공식의 변수

변수	설명
Bound address	데이터 버퍼들의 주소는 SQLBindCol의 TargetValuePtr 인자에 명시되어 있다. 길이/ 지시자 버퍼 주소는 SQLBindCol의 StrLen_of_IndPtr 인자에 명시되어 있다. 바인딩 주소가 0이면 계산된 주소가 0이 아니어도 데이터 값은 반환되지 않는다.
Binding offset	Row 방향 바인딩이 사용될 경우, SQL_ATTR_ROW_BIND_OFFSET_PTR 명령문 속성과 함께 명시된 주소에 값이 저장된다. Column 방향 바인딩이 사용되었거나 SQL_ATTR_ROW_BIND_OFFSET_PTR 명령문 속성이 NULL 포인터일 경우, 바인딩 offset은 0이다.
Row number	1-based인 row 집합의 row 번호이다. 단일 row를 회수할 경우의 row 번호는 기본적으로 1이다.
Element size	바인딩 배열 안의 원소 크기이다. Column 방향 바인딩이 사용될 경우, 길이/ 지시자 버퍼에 대해 sizeof (SQLLEN)이다. 데이터 버퍼들에 대해 가변 길이 데이터 형식일 경우 element size는 SQLBindCol의 BufferLength 인자값이고, 고정 길이 데이터 형식일 경우 element size는 데이터 형식의 크기이다. Row 방향 바인딩이 사용될 경우, 데이터 및 길이/ 지시자 버퍼 모두 SQL_ATTR_ROW_BIND_TYPE 명령문 속성값이다.

설명자들과 SQLBindCol

본 절에서는 SQLBindCol이 어떻게 설명자들과 소통하는지 설명한다.

주의

하나의 명령문에 대해 SQLBindCol을 호출하면 다른 명령문에 영향을 줄 수 있다. 이는 명령문과 관련된 ARD가 명시적으로 할당되어 있고, 다른 명령문들과 관련되어 있을 때 발생한다. SQLBindCol은 설명자를 수정하기 때문에, 설명자에 대한 수정은 설명자와 관련된 모든 명령문들에 적용된다. 의도한 동작이 아니라면 응용 프로그램이 SQLBindCol을 호출하기 전에 다른 명령문들로부터 이 설명자와의 관련성을 해제해야 한다.

인자 맵핑

개념상, SQLBindCol은 다음 순서대로 수행된다.

1. SQLGetStmtAttr를 호출하여 ARD 핸들을 획득한다.
2. SQLGetDescField를 호출하여 SQL_DESC_COUNT 필드의 설명자를 얻고, ColumnNumber 인자의 값이 SQ

L_DESC_COUNT의 값을 초과하면 SQLSetDescField를 호출하여 SQL_DESC_COUNT 값을 ColumnNumber 값까지 증가시킨다.

3. SQLSetDescField를 여러 번 호출하여 ARD의 다음 필드들의 값을 설정한다.

- SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE을 targetType 값으로 설정한다.
 - targetType이 datetime 또는 interval 서브 형식의 함축된 식별자들 중 하나인 경우를 제외한 SQL_DESC_TYPE을 각각 SQL_DATETIME 또는 SQL_INTERVAL로 설정한다. SQL_DESC_CONCISE_TYPE은 함축된 식별자로 설정하며, SQL_DESC_DATETIME_INTERVAL_CODE는 해당 datetime 또는 interval 서브 코드로 설정한다.
- 하나 이상의 SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE 및 SQL_DESC_DATETIME_INTERVAL_PRECISION을 targetType에 대해 적절하게 설정한다.
- SQL_DESC_OCTET_LENGTH 필드는 BufferLength 값으로 설정한다.
- SQL_DESC_DATA_PTR 필드는 TargetValue 값으로 설정한다.
- SQL_DESC_INDICATOR_PTR 필드 또한 StrLen_or_Ind 값으로 설정한다.
- SQL_DESC_OCTET_LENGTH_PTR 필드는 StrLen_or_Ind 값으로 설정한다.

StrLen_or_Ind 인자가 참조하는 변수는 지시자와 길이 정보에 사용된다. 회수할 때 column의 값이 NULL 인 경우, SQL_NULL_DATA를 저장한다. 그렇지 않으면 해당 변수의 데이터 길이를 저장한다.

NULL 포인터를 입력하면 회수할 때의 column 값이 NULL인 경우 SQL_NULL_DATA를 반환할 방법이 없기 때문에 회수 연산에 실패한다.

SQLBindCol이 실패하면 ARD 내에 설정하려 했던 설명자 필드들의 내용은 정의되지 않고, ARD의 SQL_DESC_COUNT 필드는 변경되지 않는다.

COUNT 필드의 암묵적 초기화

SQLBindCol은 ColumnNumber가 SQL_DESC_COUNT 값을 증가시키는 경우에만 SQL_DESC_COUNT를 ColumnNumber 값으로 설정한다. TargetValuePtr 값이 NULL 포인터이고 ColumnNumber 값이 (가장 높은 바인딩 column을 바인딩 해제 했을 때) SQL_DESC_COUNT와 같다면, SQL_DESC_COUNT는 남아 있는 바인딩 column 중에 가장 높은 번호로 설정된다.

SQL_DEFAULT에 대한 주의 사항

응용 프로그램은 column 데이터를 성공적으로 검색하기 위해 응용 프로그램 버퍼에서 데이터의 길이와 시작 지점을 올바르게 설정해야 한다. 응용 프로그램이 targetType을 명시적으로 정의하면 응용 프로그램의 오류는 쉽게 발견된다.

하지만 응용 프로그램이 targetType에 SQL_DEFAULT를 정의할 때 SQLBindCol은 메타 데이터를 갱신하거나 다른 column에 대해 코드를 적용함으로써, 응용 프로그램에서 목표로 하는 데이터 형식과 다른 데이터 형식의 column에 적용될 수 있다. 이러한 경우 응용 프로그램이 회수된 column 데이터의 시작 또는 길이를 항상 결정할 수 있는 것은 아니다. 이것은 보고되지 않은 데이터 오류 또는 메모리 침범을 유발할 수 있다.

SQLBindParameter

적합성

도입된 버전: ODBC 2.0

표준 준수: ODBC

개요

SQLBindParameter는 SQL 문의 매개 변수 마커에 버퍼를 바인딩한다.

구문

```
SQLRETURN SQLBindParameter(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   ParameterNumber,
    SQLSMALLINT    InputOutputType,
    SQLSMALLINT    ValueType,
    SQLSMALLINT    ParameterType,
    SQLULEN        ColumnSize,
    SQLSMALLINT    DecimalDigits,
    SQLPOINTER     ParameterValuePtr,
    SQLLEN         BufferLength,
    SQLLEN *       StrLen_or_IndPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ParameterNumber

[입력] 1부터 순차적으로 증가하는 매개 변수 번호이다.

InputOutputType

[입력] 매개 변수의 타입이다.

ValueType

[입력] 매개 변수의 C 데이터 타입이다.

ParameterType

[입력] 매개 변수의 SQL 데이터 타입이다.

ColumnSize

[입력] 해당 매개 변수 마커의 column 또는 표현식의 크기이다.

DecimalDigits

[입력] 해당 매개 변수 마커의 column 또는 표현식의 소수점 자리 수이다.

ParameterValuePtr

[지연된 입력] 매개 변수의 데이터 버퍼 포인터이다.

BufferLength

[입력/출력] ParameterValuePtr 버퍼의 바이트 길이이다.

StrLen_or_IndPtr

[지연된 입력] 매개 변수의 길이/ 지시자에 대한 포인터이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

표 25-8 SQLSTATE values

SQLSTATE	Error	설명
07006	Restricted data type attribute violation	ValueType 인자 데이터 타입을 ParameterType 인자 데이터 타입으로 변환할 수 없다.
07009	Invalid descriptor index	ParameterNumber 인자값이 1보다 작다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY003	Invalid application buffer type	ValueType 인자값이 유효한 C 데이터 타입이 아니다.
HY004	Invalid SQL data type	ParameterType 인자값이 유효한 SQL 데이터 타입이 아니다.
HY009	Invalid argument value	ParameterValuePtr 인자 및 StrLen_or_IndPtr 인자가 null 포인터이며, InputOutputType 인자가 SQL_PARAM_OUTPUT이 아니다. InputOutputType 인자가 SQL_PARAM_OUTPUT이며, ParameterValuePtr 인자가 null 포인터, C 타입이 문자나 이진이며, BufferLength가 0보다 크다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY021	Inconsistent descriptor information	무결성 검사를 할 때 설명자 정보에 일관성이 없다.

SQLSTATE	Error	설명
	n	
HY090	Invalid string or buffer length	BufferLength 값이 0보다 작다.
HY104	Invalid precision or scale value	ColumnSize 및 DecimalDigits에 지정된 값이 ParameterType 인자의 SQL 데이터 지원 범위를 벗어났다.
HY105	Invalid parameter type	InputOutputType 인자값이 유효하지 않다.
HYC00	Optional feature not implemented	드라이버가 ValueType 인자값과 ParameterType 인자값의 변환을 지원하지 않는다. ParameterType 인자값은 유효하지만 드라이버가 지원하지 않는다.

설명

응용 프로그램은 SQL 문의 각 매개 변수 마커를 바인딩하기 위해 SQLBindParameter를 호출한다. 응용 프로그램에서 SQLBindParameter를 다시 호출하거나, SQL_RESET_PARAMS 옵션으로 SQLFreeStmt를 호출하거나, SQLSetDescField를 호출하여 APD의 SQL_DESC_COUNT 헤더 필드를 0으로 설정할 때까지 바인딩은 유효하다.

ParameterNumber 인자

SQLBindParameter를 호출할 때 ParameterNumber가 SQL_DESC_COUNT의 값보다 큰 경우, SQL_DESC_COUNT 값을 ParameterNumber로 증가시키기 위해 SQLSetDescField를 호출한다.

InputOutputType 인자

InputOutputType 인자는 매개 변수 타입을 지정한다. 이 인자는 IPD의 SQL_DESC_PARAMETER_TYPE 필드를 설정한다.

InputOutputType 인자는 다음 값들 중 하나이다.

- SQL_PARAM_INPUT: Procedure나 SELECT INTO 구문이 아닌 SQL 문에 매개 변수를 표시한다. 예를 들어 INSERT INTO Employee VALUES (?, ?, ?) 의 매개 변수는 입력 매개 변수이다.
 - 명령문이 실행되면 드라이버가 매개 변수 데이터를 전송하며, 이 때 *ParameterValuePtr 버퍼가 유효한 입력값을 포함하거나, *StrLen_or_IndPtr 버퍼가 SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_LEN_DATA_AT_EXEC 매크로의 결과를 포함해야 한다.
- SQL_PARAM_INPUT_OUTPUT: Procedure의 입력/ 출력 매개 변수를 표시한다.
 - 명령문이 실행되면 드라이버가 매개 변수 데이터를 전송하며, 이 때 *ParameterValuePtr 버퍼가 유효한 입력값을 포함하거나, *StrLen_or_IndPtr 버퍼가 SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_LEN_DATA_AT_EXEC 매크로의 결과를 포함해야 한다.
 - 명령문이 수행된 후 드라이버는 응용 프로그램의 매개 변수에 데이터를 반환한다. 만약 입력/ 출력 매개 변수에 값을 반환할 수 없다면, 드라이버가 *StrLen_or_IndPtr 버퍼에 SQL_NULL_DATA를 설정한다.
- SQL_PARAM_OUTPUT: 매개 변수는 procedure의 반환값이나 procedure, SELECT INTO 구문의 출력 매개 변수를 표시한다. 예를 들어 SELECT ID INTO ? FROM Employee WHERE NAME = 'Paul'는 ID를 반환하는

출력 매개 변수이다.

- 명령문이 수행된 후 응용 프로그램의 ParameterValuePtr과 StrLen_or_IndPtr 인자가 null 포인터가 아니라면 드라이버가 매개 변수에 데이터를 반환하고, 그렇지 않으면 출력 결과를 폐기한다.
- 만약 출력 매개 변수에 값을 반환할 수 없다면, 드라이버가 *StrLen_or_IndPtr 버퍼에 SQL_NULL_DATA를 설정한다.

ValueType 인자

ValueType 인자는 매개 변수의 C 데이터 타입을 지정한다. 이 인자는 APD의 SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE 필드값을 설정한다.

ValueType 인자가 interval 데이터 타입 중 하나인 경우,

- APD의 ParameterNumber 레코드의 SQL_DESC_TYPE 필드는 SQL_INTERVAL로 설정된다.
- SQL_DESC_CONCISE_TYPE 필드는 concise interval 데이터 타입으로 설정된다.
- SQL_DESC_DATETIME_INTERVAL_CODE 필드는 특정 interval 데이터의 서브 코드로 설정된다.
- interval leading precision의 기본값은 (2), interval seconds precision의 기본값은 (6)으로 APD의 SQL_DESC_DATETIME_INTERVAL_PRECISION과 SQL_DESC_PRECISION 필드 각각에 설정된다.
- 만약 기본 precision과 scale이 적절하지 않은 경우, 응용 프로그램은 SQLSetDescField나 SQLSetDescRec를 호출하여 해당 설명자 필드를 명시적으로 설정해야 한다.

ValueType 인자가 datetime 데이터 타입 중 하나인 경우,

- APD의 ParameterNumber 레코드의 SQL_DESC_TYPE 필드는 SQL_DATETIME으로 설정된다.
- SQL_DESC_CONCISE_TYPE 필드는 concise date C 데이터 타입으로 설정된다.
- SQL_DESC_DATETIME_INTERVAL_CODE 필드는 특정 datetime 데이터의 서브 코드로 설정된다.

ValueType 인자가 SQL_C_NUMERIC 일 경우,

- precision의 기본값은 (38), scale의 기본값은 (0)이며, APD의 SQL_DESC_PRECISION과 SQL_DESC_SCALE 필드 각각에 설정된다.
- 만약 기본 precision 및 scale이 적절하지 않은 경우, 응용 프로그램은 SQLSetDescField나 SQLSetDescRec를 호출하여 해당 설명자 필드를 명시적으로 설정해야 한다.

ParameterType 인자

ParameterType 인자는 매개 변수의 SQL 데이터 타입을 지정한다. 이 인자는 IPD의 SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE 필드값을 설정한다.

ParameterType 인자가 datetime 데이터 타입 중 하나인 경우,

- IPD의 SQL_DESC_TYPE 필드는 SQL_DATETIME으로 설정된다.
- SQL_DESC_CONCISE_TYPE 필드는 concise datetime SQL 데이터 타입으로 설정된다.
- SQL_DESC_DATETIME_INTERVAL_CODE 필드는 해당 datetime의 서브 코드로 설정된다.

ParameterType 인자가 interval 데이터 타입 중 하나인 경우,

- IPD의 SQL_DESC_TYPE 필드는 SQL_INTERVAL로 설정된다.
- SQL_DESC_CONCISE_TYPE 필드는 concise SQL interval 데이터 타입으로 설정된다.
- SQL_DESC_DATETIME_INTERVAL_CODE 필드는 해당 interval의 서브 코드로 설정된다.
- IPD의 SQL_DESC_DATETIME_INTERVAL_PRECISION 필드에는 interval leading precision이, SQL_DESC_PRECISION 필드에는 interval second precision이 설정된다.
- SQL_DESC_DATETIME_INTERVAL_PRECISION과 SQL_DESC_PRECISION의 기본값이 적절하지 않은 경우, 응용 프로그램이 SQLSetDescField를 호출하여 이들을 설정한다.

ParameterType 인자가 SQL_NUMERIC 일 경우,

- precision의 기본값은 (38), scale의 기본값은 (0)이고 IPD의 SQL_DESC_PRECISION과 SQL_DESC_SCALE 필드 각각에 설정된다.
- 만약 기본 precision과 scale이 적절하지 않은 경우, 응용 프로그램은 SQLSetDescField나 SQLSetDescRec를 호출하여 해당 설명자 필드를 명시적으로 설정해야 한다.

ColumnSize 인자

ColumnSize 인자는 해당 매개 변수 마커의 column 또는 표현식의 크기이다. 이 인자는 ParameterType 인자의 SQL 데이터 타입에 따라 IPD의 다른 필드를 설정한다.

- ParameterType이 SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, concise SQL datetime 또는 interval 데이터 타입일 경우, IPD의 SQL_DESC_LENGTH 필드에 ColumnSize의 값을 설정한다.
- ParameterType이 SQL_DECIMAL, SQL_NUMERIC, SQL_FLOAT, SQL_REAL, SQL_DOUBLE일 경우, IPD의 SQL_DESC_PRECISION 필드에 ColumnSize의 값을 설정한다.
- 그 외 데이터 타입일 경우, ColumnSize 인자는 무시된다.

DecimalDigits 인자

- ParameterType이 SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_INTERVAL_SECOND, SQL_INTERVAL_DAY_TO_SECOND, SQL_INTERVAL_HOUR_TO_SECOND, SQL_INTERVAL_MINUTE_TO_SECOND일 경우, IPD의 SQL_DESC_PRECISION 필드에 DecimalDigits의 값을 설정한다.
- ParameterType이 SQL_NUMERIC, SQL_DECIMAL일 경우, IPD의 SQL_DESC_SCALE 필드에 DecimalDigits의 값을 설정한다.
- 그 외 데이터 타입일 경우, DecimalDigits 인자는 무시된다.

ParameterValuePtr 인자

ParameterValuePtr 인자는 SQLExecute, SQLExecDirect를 호출할 때 매개 변수에 대한 실제 데이터를 가리킨다. 데이터는 반드시 ValueType으로 지정된 형식이어야 한다. 이 인자는 APD의 SQL_DESC_DATA_PTR 필드를 설정한다.

*StrLen_or_IndPtr이 SQL_LEN_DATA_AT_EXEC (length) 매크로의 결과이거나 SQL_DATA_AT_EXEC일 경우, ParameterValuePtr은 매개 변수와 관련된 응용 프로그램이 정의한 포인터 값이며, 이는 SQLParamData를 수행할 때 응용 프로그램에 반환된다. 예를 들어, ParameterValuePtr은 매개 변수 번호, 데이터 포인터, 응용 프로그램이 입

력 매개 변수를 바인딩하는데 사용하는 구조체에 대한 포인터와 같이 0이 아닌 토큰일 수 있다.

InputOutputType 인자가 SQL_PARAM_INPUT_OUTPUT 또는 SQL_PARAM_OUTPUT일 경우, ParameterValuePtr은 출력값이 저장되는 버퍼 포인터이어야 한다.

SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1보다 큰 경우, ParameterValuePtr은 배열을 가리킨다. 단일 SQL 문은 전체 배열의 입력 또는 입력/ 출력 매개 변수 입력값을 처리하고, 입력/ 출력 또는 출력 매개 변수 출력값의 배열을 반환한다.

BufferLength 인자

문자 및 이진 C 데이터에 대해, SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1일 경우, *ParameterValuePtr 버퍼의 길이를 지정하며, SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1보다 클 경우, *ParameterValuePtr 배열 요소의 길이를 지정한다.

입력할 때와 출력할 때 모두, BufferLength는 *ParameterValuePtr 배열에서 위치를 결정하는데 사용된다. 이 인자는 APD의 SQL_DESC_OCTET_LENGTH 필드를 설정한다.

입력/ 출력 또는 출력 매개 변수의 경우, 출력을 자를지 여부를 결정하는데 BufferLength가 사용된다.

- 문자 C 데이터의 경우, 반환할 수 있는 데이터의 바이트 길이가 BufferLength보다 크거나 같으면, *ParameterValuePtr의 데이터가 BufferLength - 1 바이트로 잘리고, null로 종료된다.
- 이진 C 데이터의 경우, 반환할 수 있는 데이터의 바이트 길이가 BufferLength보다 크거나 같으면, *ParameterValuePtr의 데이터가 BufferLength 바이트로 잘린다.
- 그 외 C 데이터 타입의 경우, BufferLength 인자는 무시된다.

StrLen_or_IndPtr 인자

StrLen_or_IndPtr 인자는 SQLExecute나 SQLExecDirect를 호출할 때 다음 중 하나를 포함한다. (StrLen_or_IndPtr은 APD의 SQL_DESC_OCTET_LENGTH_PTR과 SQL_DESC_INDICATOR_PTR을 설정한다.)

- *ParameterValuePtr에 저장된 매개 변수 길이이다. 문자나 이진 C 데이터 외에는 모두 무시된다.
- SQL_NTS: 매개 변수 값은 null로 종료되는 문자열이다.
- SQL_NULL_DATA: 매개 변수 값은 NULL이다.
- SQL_LEN_DATA_AT_EXEC(length) 매크로의 결과: 매개 변수 데이터는 SQLPutData를 수행할 때 전송된다.
 - 만약 ParameterType 인자가 SQL_LONGVARBINARY, SQL_LONGVARCHAR 또는 긴 데이터 타입이고, SQLGetInfo의 SQL_NEED_LONG_DATA_LEN 정보가 Y를 반환할 경우, length는 매개 변수에 의해 전송되는 데이터 바이트 수가 된다.
 - SQLGetInfo의 SQL_NEED_LONG_DATA_LEN 정보가 N일 경우, length는 음수가 아닌 값이어야 하고, 이는 무시된다.
 - 예를 들어, SQLPutData를 여러 번 호출해 10,000 바이트의 SQL_LONGVARCAHR 매개 변수 데이터를 전송하려고 할 경우, *StrLen_or_IndPtr은 SQL_LEN_DATA_AT_EXEC(10000)으로 설정하면 된다.
- SQL_DATA_AT_EXEC: 매개 변수 데이터는 SQLPutData를 수행할 때 전송된다.

StrLen_or_IndPtr이 null 포인터인 경우, 드라이버는 모든 입력 매개 변수 값이 null이 아니고, 문자와 이진 데이터가 null로 종료된다고 간주한다. InputOutputType이 SQL_PARAM_OUTPUT이고, ParameterValuePtr과 StrLen_or

_IndPtr이 모두 null 포인터인 경우, 드라이버가 출력값을 폐기한다.

InputOutputType 인자가 SQL_PARAM_INPUT_OUTPUT, SQL_PARAM_OUTPUT일 경우, StrLen_or_IndPtr은 SQL_NULL_DATA, *ParameterValuePtr에 (문자 데이터의 null 종료 바이트를 제외한) 반환 가능한 바이트 수, SQL_NO_TOTAL (반환될 바이트 수를 판단할 수 없는 경우)를 가리킨다.

SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1보다 큰 경우, StrLen_or_IndPtr은 SQLLEN 값의 배열을 가리킨다.

매개 변수 값 전달

응용 프로그램은 *ParameterValuePtr 버퍼나 복수의 SQLPutData를 호출하여 매개 변수 값을 전달할 수 있다. SQLPutData를 통해 전달된 매개 변수들은 data-at-execution 매개 변수라고 알려져 있다. 이것들은 일반적으로 SQL_LONGVARIABLE와 SQL_LONGVARCHAR 매개 변수들의 데이터를 보내는데 사용되고 다른 매개 변수들과 섞어서 사용할 수도 있다.

매개 변수 값들을 전달하려면 응용 프로그램이 다음 과정을 수행해야 한다.

1. 매개 변수들의 값 (ParameterValuePtr 인자)과 길이/ 지시자 (StrLen_or_IndPtr 인자)의 버퍼들을 바인드하기 위해 각각의 매개 변수에 SQLBindParameter를 호출한다. data-at-execution 매개 변수들의 경우, ParameterValuePtr은 매개 변수 번호나 데이터 포인터와 같은 응용 프로그램에서 정의된 포인터 값이다. 이 값은 추후에 반환될 것이고 매개 변수를 식별하는데 사용될 수 있다.
2. 입력 및 입력/ 출력 매개 변수들 값을 *ParameterValuePtr과 *StrLen_or_IndPtr 버퍼들에 설정한다.
 - 일반 매개 변수들의 경우, 응용 프로그램이 *ParameterValuePtr 버퍼에 매개 변수 값을 입력하고 *StrLen_or_IndPtr 버퍼에 그 값의 길이를 입력한다.
 - data-at-execution 매개 변수들의 경우, 응용 프로그램이 *StrLen_or_IndPtr 버퍼에 SQL_LEN_DATA_AT_EXEC(length) 매크로 (ODBC 2.0 드라이버를 호출할 때)의 결과를 입력한다.
3. SQLExecute나 SQLExecDirect를 호출하여 SQL 명령문을 실행한다.
 - data-at-execution 매개 변수가 없을 경우, 프로세스가 완료된다.
 - data-at-execution 매개 변수가 있을 경우, 함수가 SQL_NEED_DATA를 반환한다.
4. SQLParamData를 호출하여 처리할 첫 번째 data-at-execution 매개 변수를 위해 SQLBindParameter의 ParameterValuePtr 인자에 명시된 응용 프로그램의 정의값을 회수한다. SQLParamData는 SQL_NEED_DATA를 반환한다.

노트

Data-at-execution 매개 변수들은 data-at-execution column들과 유사하지만 SQLParamData에서 반환되는 값이 다르다.

Data-at-execution 매개 변수들은 명령문이 SQLExecDirect 또는 SQLExecute와 함께 실행될 때 SQLPutD

ata를 통해 보내질 SQL 명령문의 매개 변수들이다. 그것들은 SQLBindParameter에 바인딩된다.

SQLParamData에서 반환된 값은 SQLBindParameter의 ParameterValuePtr 인자에 전달된 포인터 값이다. data-at-execution column들은 row가 SQLBulkOperations로 갱신 또는 추가되거나 SQLSetPos를 통해 갱신될 때 보내질 row 집합의 column들이다. 그것들은 SQLBindCol에 바인딩된다. SQLParamData에서 반환된 값은 처리될 *TargetValuePtr (SQLBindCol을 호출하여 설정한) 버퍼 안의 row 주소이다.

5. 한 번 이상 SQLPutData를 호출하여 매개 변수의 데이터를 전송한다. 데이터 값이 SQLPutData의 *Parameter ValuePtr 버퍼에 명시된 것보다 클 경우, 한 번 이상 호출해야 한다. 문자, 이진 또는 데이터 소스 명시 데이터 형식으로 된 문자 C 데이터 또는 이진 C 데이터를 전송할 때만 같은 매개 변수에 대해 SQLPutData를 여러 번 호출할 수 있다.
6. SQLParamData를 다시 호출하여 매개 변수를 위한 모든 데이터가 전송됐다는 신호를 보낸다.
 - 하나 이상의 data-at-execution 매개 변수들이 있을 경우, SQLParamData가 SQL_NEED_DATA를 반환하고 다음 data-at-execution 매개 변수를 위한 응용 프로그램 정의값을 처리한다. 응용 프로그램은 4번과 5번 과정을 반복한다.
 - Data-at-execution 매개 변수가 없을 경우, 프로세스가 완료된다. 명령문이 성공적으로 실행된 경우, SQLParamData는 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환한다. 실행에 실패할 경우, SQL_ERROR를 반환한다. 이 시점에서 SQLParamData는 모든 SQLSTATE에 의해 반환될 수 있고 명령문을 실행하는데 사용되는 SQLExecDirect나 SQLExecute 함수에 의해서도 반환될 수 있다.
 - 입력/ 출력 또는 출력 매개 변수들의 출력값들은 응용 프로그램이 명령문에 의해 생성된 모든 결과 집합들을 회수한 후에 *ParameterValuePtr과 *StrLen_or_IndPtr 버퍼들에서 사용할 수 있다.

SQLExecute나 SQLExecDirect를 호출하여 명령문을 SQL_NEED_DATA 상태로 만든다. 이 시점에서 응용 프로그램은 SQLCancel, SQLGetDiagField, SQLGetDiagRec, SQLGetFunctions, SQLParamData 또는 SQLPutData만 명령문 또는 명령문과 관련된 연결 핸들과 함께 호출할 수 있다.

만약 위와 다른 함수를 명령문 또는 명령문과 관련된 연결 핸들에 대해 호출할 경우, 함수가 SQLSTATE HY010(Function sequence error)를 반환한다. SQLParamData 또는 SQLPutData가 오류를 발생시키거나 SQLParamData가 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환하거나, 명령문이 취소되었을 경우, 명령문은 SQL_NEED_DATA 상태에서 벗어난다.

드라이버가 아직 data-at-execution 매개 변수들을 위한 데이터를 필요로 하는 동안 응용 프로그램이 SQLCancel을 호출할 경우, 드라이버는 명령문 실행을 취소한다. 이 때, 응용 프로그램은 SQLExecute나 SQLExecDirect를 다시 호출할 수 있다.

매개 변수 배열 사용

응용 프로그램은 다음과 같은 두 가지 방법을 사용하여 매개 변수 마커들과 함께 명령문을 준비하고 매개 변수 배열 안에 전달한다.

- 드라이버가 배열 처리 능력에 의존하여 처리한다. 이 경우, 매개 변수와 함께하는 모든 명령문이 하나의 단위로

간주된다. 배열 처리 능력을 지원하는 데이터 소스의 예로는 Oracle이 있다.

- 드라이버가 SQL 명령문의 배치를 생성하는 방법도 있다. SQL 명령문 하나당 매개 변수 배열 안의 매개 변수 집합 한 개를 매칭한 배치를 실행한다. 매개 변수 배열은 UPDATE WHERE CURRENT OF 명령문에는 사용할 수 없다.

매개 변수 배열이 처리되었을 때 각각의 결과 집합/ row 개수들을 매개 변수 집합당 하나씩 사용하거나 통째로 하나를 사용할 수도 있다. SQLGetInfo에서 SQL_PARAM_ARRAY_ROW_COUNTS 옵션은 row 개수를 각각의 매개 변수 집합 (SQL_PARC_BATCH)에 사용할 수 있는지 또는 단 한 개의 row 개수만 사용할 수 있는지 (SQL_PARC_NO_BATCH) 여부를 나타낸다.

SQLGetInfo에서 SQL_PARAM_ARRAY_SELECTS 옵션은 결과 집합을 각각의 매개 변수 집합 (SQL_PAS_BATCH)에 사용할 수 있는지 또는 단 하나의 결과 집합 (SQL_PAS_NO_BATCH)에만 사용할 수 있는지 여부를 나타낸다. 드라이버가 result set-generating 명령문이 매개 변수 배열과 함께 실행되는 것을 허용하지 않을 경우, SQL_PARAM_ARRAY_SELECTS는 SQL_PAS_NO_SELECT를 반환한다.

자세한 정보는 **SQLGetInfo** 를 참조한다.

매개 변수 배열을 지원하기 위해 SQL_ATTR_PARAMSET_SIZE 명령문 속성은 매개 변수당 값의 개수를 명시한다. 필드가 1보다 클 경우, APD의 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR과 SQL_DESC_OCTET_LENGTH_PTR 필드들은 배열을 가리켜야 한다. 각 배열의 원소수는 SQL_ATTR_PARAMSET_SIZE의 값과 같다.

APD의 SQL_DESC_ROWS_PROCESSED 필드는 오류를 포함하여 처리된 매개 변수 집합의 개수를 포함하는 버퍼를 가리킨다. 처리된 각 매개 변수 집합처럼 드라이버는 새로운 값을 버퍼에 저장한다. NULL 포인터의 경우 어떤 수도 반환하지 않는다.

매개 변수 배열이 사용됐을 때, 함수 설정으로 SQL_ERROR가 반환되더라도 APD의 SQL_DESC_ROWS_PROCESSED_PTR 필드가 가리키는 값은 생성된다. SQL_NEED_DATA가 반환되면 APD의 SQL_DESC_ROWS_PROCESSED_PTR 필드가 가리키는 값은 처리될 매개 변수들의 값으로 설정된다.

Column 방향 매개 변수 바인딩

Column 방향 바인딩에서 응용 프로그램은 분리된 매개 변수 및 길이/ 지시자 배열들을 각각의 매개 변수에 바인딩한다.

Column 방향 바인딩을 사용하기 위해 응용 프로그램은 제일 먼저 SQL_ATTR_PARAM_BIND_TYPE 명령문 속성을 SQL_PARAM_BIND_BY_COLUMN으로 설정한다. (이것이 기본값이다.) 각 column을 바인딩하기 위해 응용 프로그램이 다음 절차를 수행한다.

1. 매개 변수 버퍼 배열을 할당한다.
2. 길이/ 지시자 버퍼 배열을 할당한다.

노트

Column 방향 바인딩이 사용될 때 응용 프로그램이 설명자들에 직접 기록할 경우, 독립된 배열은 길이와 지시자 데이터에 사용될 수 있다.

3. 다음 인자와 함께 `SQLBindParameter`를 호출한다.

- `ValueType`은 매개 변수 버퍼 배열 안에 있는 단일 원소의 C 형식이다.
- `ParameterType`은 매개 변수의 SQL 형식이다.
- `ParameterValuePtr`은 매개 변수 버퍼 배열의 주소이다.
- `BufferLength`는 매개 변수 버퍼 배열 안에 있는 단일 원소의 크기이다. `BufferLength` 인자는 데이터가 고정길 이 데이터 타입일 때는 무시된다.

Row 방향 매개 변수 바인딩

Row 방향 바인딩에서 응용 프로그램은 바인딩 될 각 매개 변수에 대해 매개 변수와 길이/ 지시자 버퍼를 포함하는 구조체를 정의한다.

Row 방향 바인딩을 사용하기 위해 응용 프로그램이 다음 절차를 수행해야 한다.

1. (매개 변수와 길이/ 지시자 버퍼들을 포함하는) 한 개의 매개 변수 집합을 유지하는 구조체를 정의하고 이 구조체들의 배열을 할당한다.

노트

Row 방향 바인딩이 사용될 때 응용 프로그램이 설명자에 직접 기록할 경우, 별도의 필드들은 길이와 지시자 데이터로 사용될 수 있다.

2. `SQL_ATTR_PARAM_BIND_TYPE` 명령문 속성은 단일 매개 변수 집합을 포함하는 구조체의 크기 또는 바인딩 될 매개 변수 버퍼의 인스턴스 크기를 설정한다. 해당 길이는 바인딩된 매개 변수의 모든 공간을 포함해야 한다. 또한 이 길이는 바인딩 매개 변수와 구조체의 패딩을 위한 공간을 모두 포함하거나 바인딩된 매개 변수 주소가 명시된 길이만큼 증가할 때 결과가 반드시 다음 매개 변수의 시작을 가리키도록 버퍼링해야 한다. ANSI C에서는 `sizeof` 연산자를 사용하여 이를 보장한다.

3. 바인딩 될 각 매개 변수에 다음 인자들과 함께 `SQLBindParameter`를 호출한다.

- `ValueType`은 column에 바인딩 될 매개 변수 버퍼 멤버의 형식이다.
- `ParameterType`은 매개 변수 SQL 형식이다.
- `ParameterValuePtr`은 첫 번째 배열 원소에 있는 매개 변수 버퍼 멤버의 주소이다.
- `BufferLength`는 매개 변수 버퍼 멤버의 크기이다.
- `StrLen_or_IndPtr`은 바인딩 될 길이/ 지시자 멤버의 주소이다.

오류 정보

드라이버가 배치와 같은 매개 변수 배열을 실행하지 않을 경우, (`SQL_PARAM_ARRAY_ROW_COUNTS` 옵션은 `SQL_PARC_NO_BATCH`와 같다.) 오류 상황은 한 개의 명령문이 실행된 것처럼 다뤄진다.

배치를 실행하는 경우 응용 프로그램은 SQL 명령문의 매개 변수 또는 매개 변수 배열의 매개 변수가 SQLExecDirect 또는 SQLExecute가 오류를 반환하는 원인인지 여부를 파악하기 위해 IPD의 SQL_DESC_ARRAY_STATUS_PTR 헤더 필드를 사용할 수 있다. 이 필드는 매개 변수 값의 각 row에 대한 상태 정보를 포함하고 있다. 이 필드가 오류가 발생된 것을 나타내고 있다면 진단 데이터 구조체 안에 있는 필드들은 실패한 row와 매개 변수를 나타낸다. 배열 안에 있는 column들의 개수는 APD 안에 있는 SQL_DESC_ARRAY_SIZE 헤더 필드에 의해 정의될 것이고 이는 SQL_ATTR_PARAMSET_SIZE 명령문 속성에 의해 설정될 수 있다.

노트

APD에 있는 SQL_DESC_ARRAY_STATUS_PTR 헤더 필드는 매개 변수들을 무시하는데 사용된다. 매개 변수 무시에 대한 자세한 내용은 **매개 변수 집합 무시** 를 참조한다.

SQLExecute 또는 SQLExecDirect가 SQL_ERROR를 반환할 때 IPD 안의 SQL_DESC_ARRAY_STATUS_PTR 필드가 가리키는 배열 안에 있는 원소들은 SQL_PARAM_ERROR, SQL_PARAM_SUCCESS, SQL_PARAM_SUCCESS_WITH_INFO, SQL_PARAM_UNUSED 또는 SQL_PARAM_DIAG_UNAVAILABLE을 포함한다.

배열 안에 있는 각 원소들을 위해 진단 데이터 구조체는 하나 이상의 상태 레코드를 포함한다. 구조체의 SQL_DIAG_ROW_NUMBER 필드는 오류를 야기한 매개 변수 값의 row 번호를 나타낸다. 만약 오류를 야기한 매개 변수 row 안에 있는 특정한 매개 변수를 알아낼 수 있다면 매개 변수 번호는 SQL_DIAG_COLUMN_NUMBER 필드에 입력된다.

SQLExecute나 SQLExecDirect가 이전 매개 변수 실행을 강제로 취소하여 오류가 발생하고 매개 변수가 사용되지 않은 경우 SQL_PARAM_UNUSED가 설정된다. 예를 들어 50 개의 매개 변수가 있고 SQLExecute 또는 SQLExecDirect 취소를 야기하는 40 번째 매개 변수 집합을 실행하는 동안 오류가 발생한 경우, 41에서 50까지의 매개 변수에 대한 상태 배열에 SQL_PARAM_UNUSED가 설정된다.

SQL_PARAM_DIAG_UNAVAILABLE은 드라이버가 매개 변수 배열을 단일한 것으로 취급할 경우 입력된다. 따라서 각각의 매개 변수 수준 오류 정보를 생성하지는 않는다.

한 개의 매개 변수 집합을 처리할 때 일부 오류들은 배열 안에서 추후에 매개 변수 집합 처리를 정지시킨다. 다른 오류들은 추후의 매개 변수들 처리에 영향을 미치지 않는다. 처리를 정지시키는 오류들은 드라이버에서 정의된다. 처리가 정지되지 않을 경우, 배열 안의 모든 매개 변수들은 처리되고 SQL_SUCCESS_WITH_INFO가 오류의 결과처럼 반환되며, SQL_ATTR_PARAMS_PROCESSED_PTR에 의해 정의된 버퍼는 오류를 포함하여 처리된 전체 매개 변수 집합의 개수로 설정된다.

노트

매개 변수 배열 처리에서 오류가 발생할 때 ODBC2.x에서와 ODBC 3.x에서의 ODBC의 행동들은 각각 다르다.

ODBC 2.x에서 함수는 SQL_ERROR를 반환하고 처리를 중단한다. SQLParamOptions의 pirow 인자가 가리키는 버퍼는 오류 row의 개수를 포함한다.

ODBC 3.x에서 함수는 SQL_SUCCESS_WITH_INFO를 반환하는데 처리는 중단될 수도 있고 지속될 수도 있다. 지속될 경우, SQL_ATTR_PARAMS_PROCESSED_PTR에 명시된 버퍼는 오류를 포함하여 처리된 모든 매개 변수의 값으로 설정된다. 이런 변경은 기존의 응용 프로그램에 문제가 발생하게 할 수 있다.

SQLExecute나 SQLExecDirect가 매개 변수 배열 안에 있는 모든 매개 변수 집합들의 처리가 완료되기 전에 SQL_ERROR나 SQL_NEED_DATA 등을 반환할 때 상태 배열은 이미 처리된 매개 변수들의 상태들을 포함한다. IPD 안에 있는 SQL_DESC_ROWS_PROCESSED_PTR 필드가 가리키는 위치는 매개 변수 배열 안에서 SQL_ERROR 또는 SQL_NEED_DATA 오류 코드를 야기하는 row 번호를 포함한다. 매개 변수들의 배열이 SELECT 명령문에 보내질 때 상태 배열 값을 사용할 수 있는지 여부는 드라이버에서 정의된다. 이는 명령문이 이미 실행되었거나 결과 집합 등을 회수한 후에 사용할 수 있다.

매개 변수 집합 무시

APD의 SQL_DESC_ARRAY_STATUS_PTR 필드는 SQL 명령문 안에서 무시해야 할 바인딩 매개 변수 집합을 나타내는데 사용된다. 수행 중에 드라이버가 직접 하나 이상의 매개 변수 집합들을 무시하기 위해 응용 프로그램이 다음 절차를 수행하여야 한다.

1. SQLSetDescField를 호출하여 APD의 SQL_DESC_ARRAY_STATUS_PTR 헤더 필드가 상태정보를 포함하는 SQLUSMALLINT 값들의 배열을 가리키도록 한다. 또한 이 필드는 SQLSetStmtAttr에서 Attribute 인자의 SQL_ATTR_PARAM_OPERATION_PTR을 통해 설정할 수 있는데 이는 응용 프로그램인 설명자 핸들을 얻지 않고 필드를 설정할 수 있도록 허용한다.
2. APD의 SQL_DESC_ARRAY_STATUS_PTR이 정의한 배열의 각 원소를 다음 두 값 중 하나로 설정한다.
 - SQL_PARAM_IGNORE: 해당 row를 명령문 실행에서 제외한다.
 - SQL_PARAM_PROCEED: 해당 row를 명령문 실행에 포함한다.
3. SQLExecDirect나 SQLExecute를 호출하여 준비된 (prepared) 명령문을 실행한다. APD의 SQL_DESC_ARRAY_STATUS_PTR이 정의한 배열에 다음 규칙들을 적용한다.
 - 기본적으로 포인터는 NULL로 설정한다.
 - 포인터가 NULL이면 모든 원소가 SQL_ROW_PROCEED로 설정되었을 때처럼 모든 매개 변수 집합이 사용된다.
 - 원소를 SQL_PARAM_PROCEED로 설정한다고 해서 연산이 반드시 특정한 매개 변수 집합을 사용하는 것은 아니다.
 - SQL_PARAM_PROCEED는 헤더 파일에 0으로 정의되어 있다.

응용 프로그램은 APD의 SQL_DESC_ARRAY_STATUS_PTR 필드가 IRD의 SQL_DESC_ARRAY_STATUS_PTR 필드와 동일한 배열을 참조하도록 설정할 수 있다. 이는 매개 변수들을 row 데이터에 바인딩 할 때 매우 유용하다. 매개 변수들은 row 데이터의 상태에 따라 무시될 수 있다.

SQL_PARAM_IGNORE와 더불어 다음의 상태 코드들은 SQL 명령문에서 매개 변수 설정을 무시하도록 한다.

- SQL_ROW_DELETED
- SQL_ROW_UPDATED
- SQL_ROW_ERROR

SQL_PARAM_PROCEED와 더불어 다음의 상태 코드들은 SQL 명령문에서 매개 변수 설정이 처리되도록 한다.

- SQL_ROW_SUCCESS
- SQL_ROW_SUCCESS_WITH_INFO
- SQL_ROW_ADDED

매개 변수 재 바인딩

응용 프로그램은 많은 매개 변수를 포함할 수 있지만 몇몇의 매개 변수만 사용하는 SQLExecDirect나SQLExecute를 호출하려는 버퍼 영역 구성을 가지고 있는 경우, 매개 변수의 재 바인딩이 특히 유용하다. 버퍼 영역의 남은 공간은 offset을 통해 기존 바인딩을 수정하여 다음 매개 변수 설정하는데 사용될 수 있다.

APD에 있는 SQL_DESC_BIND_OFFSET_PTR 헤더 필드는 바인딩 offset을 가리킨다. 필드가 NULL이 아니면 드라이버는 포인터를 역참조하고, SQL_DESC_DATA_PTR, SQL_INDICATOR_PTR의 값이 없고 SQL_DESC_OCTET_LENGTH_PTR 필드가 NULL 포인터일 경우, 실행 시간에 설명자 레코드들 안의 필드들에 역참조 값을 추가한다.

offset은 재 바인딩 후에 유효하다. SQL_DESC_BIND_OFFSET_PTR 필드는 offset 그 자체라기보다는 offset에 대한 포인터이기 때문에 응용 프로그램은 설명자 필드를 변경하기 위해 SQLSetDescField나 SQLSetDescRec를 호출하지 않고 직접 offset을 변경할 수 있다. 기본적으로 포인터는 NULL이다.

ARD의 SQL_DESC_BIND_OFFSET_PTR 필드는 SQLSetDescField를 호출하거나 SQLSetStmtAttr의 attribute 인자에 SQL_ATTR_PARAM_BIND_OFFSET_PTR을 통해 설정할 수 있다. offset 바인딩은 언제나 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR과 SQL_DESC_OCTET_LENGTH_PTR 필드에 직접 값을 추가한다. offset이 다른 값으로 변경될 경우, 각 설명자 필드에 새로운 값이 계속해서 직접 추가된다. 새로운 offset은 이전의 offset에 합하여 추가되지 않는다.

설명자

매개 변수가 어떻게 바인딩 되는지는 APD와 IPD의 필드들이 결정한다. SQLBindParameter 안의 인자들은 설명자 필드들을 설정하는데 사용된다. 응용 프로그램은 설명자 핸들을 얻지 않고도 SQLBindParameter를 호출할 수 있기 때문에 SQLBindParameter를 사용하는 편이 더 효과적이지만, SQLSetDescField 함수를 사용하여 필드들을 설정할 수도 있다.

주의

하나의 명령문에 대해 SQLBindParameter를 호출하면 다른 명령문에 영향을 줄 수 있다. 이는 명령문과 관련된 ARD가 명시적으로 할당되어 있고, 이것이 다른 명령문들과 관련되어 있을 때 발생한다. SQLBindParameter는 APD의 필드들을 수정하기 때문에, 필드들에 대한 수정은 이 설명자와 관련된 모든 명령문들에 적용된다. 의도한 동작이 아니라면 응용 프로그램이 SQLBindParameter를 호출하기 전에 다른 명령문들로부터 이 설명자와의 관련성을 해제해야 한다.

개념상 SQLBindParameter는 다음 절차를 수행해야 한다.

1. SQLGetStmtAttr을 호출하여 APD 핸들을 획득한다.
2. SQLGetDescField를 호출하여 APD의 SQL_DESC_COUNT 필드를 얻고, ColumnNumber 값이 SQL_DESC_COUNT 값을 초과하면 SQLSetDescField를 호출하여 SQL_DESC_COUNT 값을 ColumnNumber로 증가시킨다.
3. SQLSetDescField를 여러 번 호출하여 APD의 다음 필드들 값을 설정한다.
 - SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE을 ValueType 값으로 설정한다.
 - ValueType이 datetime이나 interval 서브형식의 함축된 식별자들 중 하나일 경우, 이를 제외시킨다. SQL_DESC_TYPE을 SQL_DATETIME이나 SQL_INTERVAL로 각각 설정하고 SQL_DESC_CONCISE_TYPE은 함축된 식별자로, SQL_DESC_DATETIME_INTERVAL_CODE는 datetime이나 interval 서브코드에 대응하여 설정한다.
 - SQL_DESC_OCTET_LENGTH 필드는 BufferLength 값으로 설정한다.
 - SQL_DESC_DATA_PTR 필드는 ParameterValue 값으로 설정한다.
 - SQL_DESC_OCTET_LENGTH_PTR 필드는 StrLen_or_Ind 값으로 설정한다.
 - SQL_DESC_INDICATOR_PTR 필드도 StrLen_or_Ind 값으로 설정한다.
 - StrLen_or_Ind는 매개 변수 값의 지시자 정보와 길이 둘 다 명시한다.
4. SQLGetStmtAttr을 호출하여 IPD 핸들을 얻는다.
5. SQLGetDescField를 호출하여 IPD의 SQL_DESC_COUNT 필드를 얻고 ColumnNumber 값이 SQL_DESC_COUNT 값을 초과할 경우, SQLSetDescField를 호출하여 SQL_DESC_COUNT 값을 ColumnNumber 값까지 증가시킨다.
6. SQLSetDescField를 여러 번 호출하여 IPD의 다음 필드들 값을 설정한다.
 - SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE을 ParameterType 값으로 설정한다.
 - ParameterType이 datetime이나 interval 서브형식의 함축된 식별자들 중 하나일 경우, 이를 제외시킨다. SQL_DESC_TYPE을 SQL_DATETIME이나 SQL_INTERVAL로 각각 설정하고 SQL_DESC_CONCISE_TYPE은 함축된 식별자로, SQL_DESC_DATETIME_INTERVAL_CODE는 datetime 또는 interval 서브코드에 대응하여 설정한다.
 - 하나 이상의 SQL_DESC_LENGTH, SQL_DESC_PRECISION과 SQL_DESC_DATETIME_INTERVAL_PRECISION을 ParameterType에 대해 적절하게 설정한다.
 - SQL_DESC_SCALE을 DecimalDigits 값으로 설정한다.

SQLBindParameter를 호출하는 데 실패할 경우, APD 안에 설정하려던 설명자 필드들의 내용은 정의되지 않고 APD의 SQL_DESC_COUNT 필드는 변경되지 않는다. 또한, IPD 안의 적절한 레코드의 SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE과 SQL_DESC_TYPE 필드들은 정의되지 않고 IPD의 SQL_DESC_COUNT 필드도 변경되지 않는다.

SQLBrowseConnect

지원되지 않는다.

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLBrowseConnect는 데이터 소스에 연결하는데 필요한 속성과 속성값을 발견하고, 반복적 나열 방법을 지원한다.

구문

```
SQLRETURN SQLBrowseConnect(  
    SQLHDBC          ConnectionHandle,  
    SQLCHAR *        InConnectionString,  
    SQLSMALLINT       StringLength1,  
    SQLCHAR *        OutConnectionString,  
    SQLSMALLINT       BufferLength,  
    SQLSMALLINT *     StringLength2Ptr);
```

SQLBulkOperations

지원되지 않는다.

적합성

도입된 버전: ODBC 3.0

표준 준수: ODBC

개요

SQLBulkOperations는 대량 삽입 및 북마크를 통한 갱신, 삭제, 가져오기 같은 대량 북마크 연산을 수행한다.

구문

```
SQLRETURN SQLBulkOperations(  
    SQLHSTMT      StatementHandle,  
    SQLUSMALLINT  Operation);
```

SQLCancel

지원되지 않는다.

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLCancel은 처리 중인 명령문을 취소한다.

처리 중인 연결이나 명령문을 취소하려면 **SQLCancelHandle** 함수를 사용한다.

구문

```
SQLRETURN SQLCancel(  
    SQLHSTMT    StatementHandle);
```

SQLCancelHandle

지원되지 않는다.

적합성

도입된 버전: ODBC 3.8

표준 준수: 없음

개요

SQLCancelHandle은 연결이나 명령문의 처리를 취소한다.

구문

```
SQLRETURN SQLCancelHandle(  
    SQLSMALLINT  HandleType,  
    SQLHANDLE     Handle);
```

SQLCloseCursor

적합성

도입된 버전: ODBC 3.0
표준 준수: ISO 92

개요

SQLCloseCursor는 명령문에서 열린 커서를 닫고, 남아 있는 결과들을 폐기한다.

구문

```
SQLRETURN SQLCloseCursor(  
    SQLHSTMT      StatementHandle);
```

인자

StatementHandle
[입력] 명령문 핸들이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
24000	Invalid cursor state	명령문에 열린 커서가 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.

설명

SQLCloseCursor는 열린 커서가 없을 경우 SQLSTATE 24000 (Invalid cursor state)를 반환한다. SQLCloseCursor 호출은 SQL_CLOSE 옵션으로 SQLFreeStmt를 호출하는 것과 같지만, 열린 커서가 없을 경우 SQLCloseCursor는 SQLSTATE 24000 (Invalid cursor state)을 반환하는 반면에 SQL_CLOSE 옵션으로 SQLFreeStmt를 호출하면 응용 프로그램에 아무런 영향을 미치지 않는다.

SQLColAttribute

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLColAttribute는 결과 집합 column에 대한 설명자 정보를 반환한다. 설명자 정보는 문자열이나 정수값으로 반환된다.

구문

```
SQLRETURN SQLColAttribute (
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ColumnNumber,
    SQLUSMALLINT  FieldIdentifier,
    SQLPOINTER     CharacterAttributePtr,
    SQLSMALLINT    BufferLength,
    SQLSMALLINT *  StringLengthPtr,
    SQLLEN *       NumericAttributePtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ColumnNumber

[입력] IRD에서 필드값을 검색해야 할 레코드 번호이다. 이는 1부터 시작하며 순차적으로 증가하는 결과 데이터의 column 번호와 대응한다. Column은 임의의 순서로 기술될 수 있다.

ColumnNumber에 0번 column을 지정할 수 있지만, SQL_DESC_TYPE과 SQL_DESC_OCTET_LENGTH를 제외한 나머지 경우에는 정의되지 않은 값을 반환한다.

FieldIdentifier

[입력] 설명자 핸들이다. 이는 IRD에서 조회할 필드를 정의한다. (예: SQL_COLUMN_TABLE_NAME)

CharacterAttributePtr

[출력] IRD의 ColumnNumber row의 FieldIdentifier 필드값이 문자열인 경우, 필드값을 반환하는 버퍼 포인터이며, 필드값이 문자열이 아닌 경우에는 사용되지 않는다.

CharacterAttributePtr이 NULL인 경우, StringLengthPtr은 반환 가능한 전체 바이트 수 (null 종료 문자 제

외)를 반환한다.

BufferLength

[입력] FieldIdentifier가 ODBC에 정의된 필드이고, CharacterAttributePtr이 문자열이나 이진 버퍼를 가리킬 경우, 이는 *CharacterAttributePtr의 길이이다. FieldIdentifier가 ODBC에 정의된 필드이고, *CharacterAttributePtr이 정수일 경우, 이 인자는 무시된다.

StringLengthPtr

[출력] *CharacterAttributePtr에서 반환 가능한 총 바이트 수 (문자 데이터에 대한 null 종료 바이트 제외)를 반환하는 포인터이다.

문자 데이터의 경우, 반환할 수 있는 바이트의 수가 BufferLength보다 크거나 같은 경우, *CharacterAttributePtr의 설명 정보는 BufferLength - 1 길이로 잘리고, 드라이버에 의해 null 종료된다.

다른 데이터 타입의 경우, BufferLength 값은 무시된다.

NumericAttributePtr

[출력] IRD의 ColumnNumber row의 FieldIdentifier 필드값이 SQL_DESC_COLUMN_LENGTH 같은 숫자인 경우, 필드의 값을 반환하는 버퍼 포인터이며, 필드값이 숫자가 아닌 경우에는 사용되지 않는다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	*CharacterAttributePtr 버퍼가 전체 문자열을 반환할만큼 충분히 크지 않아 문자열이 잘렸으며, 잘리지 않는 문자열의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07005	Prepared statement not a cursor-specification	명령문은 결과 집합을 반환하지 않았고, FieldIdentifier는 SQL_DESC_COUNT가 아니다. 설명할 column이 없다.
07009	Invalid descriptor index	ColumnNumber 인자값이 결과 집합의 column 수보다 크다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	이 함수가 SQLPrepre, SQLExecDirect, 카탈로그 함수 이전에 호출되었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	*CharacterAttributePtr이 문자열이며, BufferLength가 0보다 작지만 SQL_NTS가 아니다.
HY091	Invalid descriptor field identifier	FieldIdentifier 인자의 값이 정의된 값이 아니다.

설명

SQLColAttribute는 *NumericAttributePtr이나 *CharacterAttributePtr에 정보를 반환한다. 정수 정보는 SQLLEN 값으로 *NumericAttributePtr에 반환된다. 다른 모든 형식은 *CharacterAttributePtr에 반환된다. *NumericAttributePtr에 정보가 반환될 때, 드라이버는 CharacterAttributePtr, BufferLength, StringLengthPtr을 무시한다. *CharacterAttributePtr에 정보가 반환될 때, 드라이버는 NumericAttributePtr을 무시한다.

SQLColAttribute는 IRD의 설명 필드에서 값을 반환한다. SQLColAttribute로 반환된 FieldIdentifier 값은 적절한 IRD 핸들로 SQLGetDescField를 호출하여 얻을 수 있다.

다음 표는 SQLColAttribute로 반환되는 설명자 유형이다. NumericAttributePtr 타입은 SQLLEN* 이다.

FieldIdentifier	정보 반환	설명
SQL_DESC_AUTO_UNIQUE (ODBC 1.0)	NumericAttributePtr	- SQL_TRUE: 자동 증가 column이다. - SQL_FALSE: 자동 증가 column이 아니거나 숫자형 타입이 아니다.
SQL_DESC_BASE_COLUMN_NAME (ODBC 3.0)	CharacterAttributePtr	결과 집합 column에 대한 기본 column 이름이다. 기본 column 이름이 존재하지 않을 경우 (표현식 column의 경우), 이 변수는 빈 문자열을 포함한다. 이 정보는 IRD의 읽기 전용 필드인 SQL_DESC_BASE_COLUMN_NAME 레코드 필드에서 반환된다.
SQL_DESC_BASE_TABLE_NAME (ODBC 3.0)	CharacterAttributePtr	Column이 포함된 기본 테이블의 이름이다. 기본 테이블 이름을 정의할 수 없거나 해당되지 않을 경우, 이 변수는 빈 문자열을 포함한다. 이 정보는 IRD의 읽기 전용 필드인 SQL_DESC_BASE_TABLE_NAME 레코드 필드에서 반환된다.
SQL_DESC_CASE_SENSITIVE (ODBC 1.0)	NumericAttributePtr	- SQL_TRUE: Column이 정렬과 비교를 위해 대소문자를 구분한다. - SQL_FALSE: Column이 정렬과 비교를 위해 대소문자를 구분하지 않거나, 문자가 아니다.
SQL_DESC_CATALOG_NAME (ODBC 2.0)	CharacterAttributePtr	Column이 포함된 테이블의 카탈로그이다.
SQL_DESC_CONCISE_TYPE (ODBC 1.0)	NumericAttributePtr	Concise 데이터 타입이다. Datetime이나 interval 데이터의 경우, SQL_TYPE_TIME, SQL_INTERVAL_YEAR 같은 concise 데이터를 반환한다. 이 정보는 IRD의 SQL_DESC_CONCISE_TYPE 레코드 필드에서 반환된다.
SQL_DESC_COUNT (ODBC 1.0)	NumericAttributePtr	결과 집합에서 사용할 수 있는 column의 개수이다. 결과 집합에 column이 없는 경우 0을 반환한다. ColumnNumber 인자는 무시된다. 이 정보는 IRD의 SQL_DESC_COUNT 헤더 필드에서 반환된다.
SQL_DESC_DISPLAY_SIZE (ODBC 1.0)	NumericAttributePtr	Column을 표시하는데 필요한 최대 문자 개수이다.
SQL_DESC_FIXED		

FieldIdentifier	정보 반환	설명
D_PREC_SCALE (ODBC 1.0)	NumericAttributePtr	- SQL_TRUE: Column이 고정 precision이고 0이 아닌 scale을 갖는다. - SQL_FALSE: Column이 고정 precision이 아니고 0이 아닌 scale을 갖는다.
SQL_DESC_LABEL (ODBC 2.0)	CharacterAttributePtr	Column 레이블 또는 제목이다. 예를 들어, EmpName이라는 column 이름이 employee name 또는 별칭으로 표시될 수 있다. 레이블이 없는 경우, column 이름이 반환된다. Column에 레이블과 이름이 지정되어 있지 않는 경우, 빈 문자열이 반환된다.
SQL_DESC_LENGTH (ODBC 3.0)	NumericAttributePtr	문자열이나 이진 데이터 타입의 최대 또는 실제 길이 값이다. 고정 길이 데이터 타입은 최대 문자 길이이고, 가변 길이 데이터 타입은 실제 문자 길이이다. 이 값은 항상 문자열 끝의 null 종료 바이트를 제외한다. 이 정보는 IRD의 SQL_DESC_LENGTH 레코드 필드에서 반환된다.
SQL_DESC_LITERAL_PREFIX (ODBC 3.0)	CharacterAttributePtr	이 VARCHAR(128) 레코드 필드는 드라이버가 데이터 타입의 접두사를 인식하는 문자(열)를 포함한다. 접두사가 적용되지 않는 데이터 타입은 빈 문자열을 포함한다.
SQL_DESC_LITERAL_SUFFIX (ODBC 3.0)	CharacterAttributePtr	이 VARCHAR(128) 레코드 필드는 드라이버가 데이터 타입의 접미사를 인식하는 문자(열)를 포함한다. 접미사가 적용되지 않는 데이터 타입은 빈 문자열을 포함한다.
SQL_DESC_LOCAL_TYPE_NAME (ODBC 3.0)	CharacterAttributePtr	이 VARCHAR(128) 레코드 필드는 데이터 타입의 일반 이름과 다를 수 있는 데이터 유형의 지역화 (모국어) 이름을 포함한다. 지역화 된 이름이 없는 경우, 빈 문자열이 반환된다. 이 필드는 표시만을 목적으로 한다. 문자열의 character set은 로케일에 따라 다르며, 기본값은 일반적으로 서버의 character set이다.
SQL_DESC_NAME (ODBC 3.0)	CharacterAttributePtr	Column 별칭이 적용될 경우, column의 별칭이다. Column 별칭이 적용되지 않는 경우, column 이름이 반환된다. 두 경우 모두 SQL_DESC_UNNAMED는 SQL_NAMED로 설정된다. 만약 column 이름이나 별칭이 없는 경우 빈 문자열이 반환되고, SQL_DESC_UNNAMED는 SQL_UNNAMED가 설정된다. 이 정보는 IRD의 SQL_DESC_NAME 레코드 필드에서 반환된다.
SQL_DESC_NULLABLE (ODBC 3.0)	NumericAttributePtr	- SQL_NULLABLE: Column이 NULL 값을 가질 수 있다. - SQL_NO_NULLS: Column이 NULL 값을 가질 수 없다. - SQL_NULLABLE_UNKNOWN: Column이 NULL 값을 허용하는지 여부를 알 수 없다. 이 정보는 IRD의 SQL_DESC_NULLABLE 레코드 필드에서 반환된다.
SQL_DESC_NUM_PREC_RADIX (ODBC 3.0)	NumericAttributePtr	SQL_DESC_TYPE 필드 데이터 타입이 approximate numeric 데이터 타입이면 SQL_DESC_PRECISION 필드가 비트의 수를 포함하고 있기 때문에 이 필드는 2를 포함한다. SQL_DESC_TYPE 필드 데이터 타입이 exact numeric 데이터 타입이면 SQL_DESC_PRECISION 필드가 10진수 자리의 수를 포함하고 있기 때문에 이 필드는 10을 포함한다. 이 필드는 숫자가 아닌 모든 데이터 타입에 대해 0으로 설정된다.
SQL_DESC_OCTET_LENGTH (ODBC 3.0)	NumericAttributePtr	문자열이나 이진 데이터 타입의 바이트 길이이다. 고정 길이 문자나 이진 타입의 경우, 실제 바이트 길이이다. 가변 길이 문자나 이진 데이터 타입의 경우, 바이트 단위의 최대 길이이다. 이 값은 null 종료를 포함하지 않는다. 이 정보는 IRD의 SQL_DESC_OCTET_LENGTH 레코드 필드에서 반환된다.

FieldIdentifier	정보 반환	설명
SQL_DESC_PRECISION (ODBC 3.0)	NumericAttribute Ptr	숫자 데이터 타입에 적용할 수 있는 precision이다. SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP와 시간 간격을 나타내는 모든 interval 데이터 타입의 경우, 이 값은 적용 가능한 fractional seconds precision이다. 이 값은 IRD의 SQL_DESC_PRECISION 레코드 필드에서 반환된다.
SQL_DESC_SCALE (ODBC 3.0)	NumericAttribute Ptr	숫자 데이터 타입에 적용할 수 있는 scale이다. DECIMAL과 NUMERIC 데이터 타입의 경우, 정의된 scale이며, 다른 모든 데이터 타입에는 정의하지 않는다. 이 값은 IRD의 SQL_DESC_SCALE 레코드 필드에서 반환된다.
SQL_DESC_SCHEMA_NAME (ODBC 2.0)	CharacterAttributePtr	Column을 포함하는 테이블의 스키마이다.
SQL_DESC_SEARCHABLE (ODBC 1.0)	NumericAttribute Ptr	- SQL_PRED_NONE: Column을 WHERE 절에 사용할 수 없다. (ODBC 2.x의 SQL_UNSEARCHABLE과 동일하다.) - SQL_PRED_CHAR: Column을 WHERE 절에 사용할 수 있지만 오직 LIKE만 사용할 수 있다. (ODBC 2.x의 SQL_LIKE_ONLY와 동일하다.) - SQL_PRED_BASIC: LIKE를 제외한 모든 비교 연산자를 WHERE 절에 사용할 수 있다. (ODBC 2.x의 SQL_EXCEPT_LIKE와 동일하다.) - SQL_PRED_SEARCHABLE: Column은 비교 연산자와 함께 WHERE 절에 사용할 수 있다.
SQL_DESC_TABLE_NAME (ODBC 2.0)	CharacterAttributePtr	Column을 포함하는 테이블의 이름이다. 테이블 이름을 확인할 수 없는 경우, 빈 문자열을 반환한다.
SQL_DESC_TYPE (ODBC 3.0)	NumericAttribute Ptr	SQL 데이터 타입을 지정하는 숫자값이다. Datetime과 interval 데이터 타입의 경우, SQL_DATETIME 또는 SQL_INTERVAL과 같은 verbose 데이터 타입을 반환한다. 이 정보는 IRD의 SQL_DESC_TYPE 레코드 필드에서 반환된다.
SQL_DESC_TYPE_NAME (ODBC 1.0)	CharacterAttributePtr	데이터 소스에 종속적인 데이터 타입 이름이다. (예: "CHARACTER", "CHARACTER VARYING", "CHARACTER LONG VARYING")
SQL_DESC_UNNAMED (ODBC 3.0)	NumericAttribute Ptr	SQL_NAMED 또는 SQL_UNNAMED이다. IRD의 SQL_DESC_NAME 필드에 column 별칭 또는 column 이름이 포함된 경우, SQL_NAMED가 반환되며, column 이름이나 별칭이 없으면 SQL_UNNAME가 반환된다. 이 정보는 IRD의 SQL_DESC_UNNAMED 레코드 필드에서 반환된다.
SQL_DESC_UNSIGNED (ODBC 1.0)	NumericAttribute Ptr	- SQL_TRUE: Column이 unsigned나 숫자가 아니다. - SQL_FALSE: Column이 signed이다.
SQL_DESC_UPDATABLE (ODBC 1.0)	NumericAttribute Ptr	Column은 SQL_ATTR_READONLY, SQL_ATTR_WRITE, SQL_ATTR_READWRITE_UNKNOWN 값을 가질 수 있다.

SQLColAttributes

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.0의 SQLColAttributes 함수는 ODBC 3.x에서 SQLColAttribute 함수로 대체되었다.
자세한 내용은 **SQLColAttribute** 함수를 참조한다.

SQLColumnPrivileges

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLColumnPrivileges는 지정된 테이블에 대한 column과 관련 권한의 목록을 결과 집합으로 반환한다.

구문

```
SQLRETURN SQLColumnPrivileges(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3,
    SQLCHAR *      ColumnName,
    SQLSMALLINT    NameLength4);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] 카탈로그 이름이다. CatalogName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, CatalogName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, CatalogName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength1

[입력] *CatalogName의 길이이다.

SchemaName

[입력] 스키마 이름이다. SchemaName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, SchemaName은 대소문자

를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, SchemaName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength2

[입력] *SchemaName의 길이이다.

TableName

[입력] 테이블 이름이다. 이 인자는 null 포인터가 될 수 없다. TableName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, TableName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, TableName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength3

[입력] *TableName의 길이이다.

ColumnName

[입력] Column 이름에 대한 문자열 검색 패턴이다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, ColumnName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, ColumnName은 대소문자를 구분하며, 문자 그대로 처리되는 패턴 값이다.

NameLength4

[입력] *ColumnName의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 호출해 열린 커서가 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	TableName 인자가 null 포인터이다. SQL_ATTR_METADATA_ID 명령문 속성값이 SQL_TRUE이며, SchemaName 또는 ColumnName이 null 포인터이다.
	Function sequenc	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며,

SQLSTATE	Error	설명
HY010	e error	모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	이름 길이 인자 중 하나의 값이 0보다 작지만 SQL_NTS가 아니다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

SQLColumnPrivileges는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, PRIVILEGE로 정렬된 표준 결과 집합을 반환한다.

결과 집합의 column은 다음 표와 같다.

Column 이름	Column 번호	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	VARCHAR	카탈로그 식별자이다.
TABLE_SCHEM (ODBC 1.0)	2	VARCHAR	스키마 식별자이다.
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	테이블 식별자이다.
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	Column 이름이다. Column 이름이 없을 경우, 빈 문자열을 반환한다.
GRANTOR (ODBC 1.0)	5	VARCHAR	권한을 부여한 사용자의 이름이다.
GRANTEE (ODBC 1.0)	6	VARCHAR not NULL	권한이 부여된 사용자의 이름이다.
PRIVILEGE (ODBC 1.0)	7	VARCHAR not NULL	Column 권한 식별자로서 다음 중 하나가 될 수 있다. • SELECT: 권한을 부여 받은 사용자는 column의 데이터를 검색할 수 있다. • INSERT: 권한을 부여 받은 사용자는 관련 테이블의 column에 데이터를 추가할 수 있다. • UPDATE: 권한을 부여 받은 사용자는 column 데이터를 갱신할 수 있다. • REFERENCES: 권한을 부여 받은 사용자는 제약 조건 (예: unique, referential, table check 제약 조건)에서 column을 참조할 수 있다.
IS_GRANTABLE (ODBC 1.0)	8	VARCHAR	권한을 부여 받은 사용자가 다른 사용자에게 권한을 부여할 수 있는지 여부로서 "YES" 또는 "NO" 이다.

SQLColumns

적합성

도입된 버전: ODBC 1.0

표준 준수: Open Group

개요

SQLColumns는 지정된 테이블의 column 이름 목록을 결과 집합으로 반환한다.

구문

```
SQLRETURN SQLColumns(  
    SQLHSTMT      StatementHandle,  
    SQLCHAR *      CatalogName,  
    SQLSMALLINT    NameLength1,  
    SQLCHAR *      SchemaName,  
    SQLSMALLINT    NameLength2,  
    SQLCHAR *      TableName,  
    SQLSMALLINT    NameLength3,  
    SQLCHAR *      ColumnName,  
    SQLSMALLINT    NameLength4);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] 카탈로그 이름이다. CatalogName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, CatalogName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, CatalogName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength1

[입력] *CatalogName의 길이이다.

SchemaName

[입력] 스키마 이름에 대한 문자열 검색 패턴이다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, SchemaName은 대소문자

를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, SchemaName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength2

[입력] *SchemaName의 길이이다.

TableName

[입력] 테이블 이름에 대한 문자열 검색 패턴이다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, TableName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, TableName은 대소문자를 구분하며, 문자 그대로 처리되는 일반 인자이다.

NameLength3

[입력] *TableName의 길이이다.

ColumnName

[입력] Column 이름에 대한 문자열 검색 패턴이다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, ColumnName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, ColumnName은 대소문자를 구분하며, 문자 그대로 처리되는 패턴값이다.

NameLength4

[입력] *ColumnName의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 호출해 열린 커서가 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	TableName 인자가 null 포인터이다. SQL_ATTR_METADATA_ID 명령문 속성값이 SQL_TRUE이며, SchemaName이나 ColumnName이 null 포인터이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
	Invalid string or b	

SQLSTATE	Error	설명
HY090	uffer length	이름 길이 인자 중 하나의 값이 0보다 작지만 SQL_NTS가 아니다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

이 함수는 일반적으로 데이터 소스의 카탈로그에서 테이블 또는 테이블의 column들에 대한 정보를 검색하는 명령문을 실행하기 전에 사용된다. SQLColumns는 SQLTables가 반환한 모든 데이터 유형을 검색하는데 사용된다. 반면에 SQLColAttribute와 SQLDescribeCol은 결과 집합의 column을 설명하고, SQLNumResultCols는 결과 집합의 column 개수를 반환한다.

SQLColumns는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, ORDINAL_POSITION로 정렬된 표준 결과 집합을 반환한다.

다음 표는 결과 집합의 column이다.

Column 이름	Column 번호	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	VARCHAR	카탈로그 이름이다.
TABLE_SCHEMA (ODBC 1.0)	2	VARCHAR	스키마 이름이다.
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	테이블 이름이다.
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	Column 이름이다. Column 이름이 없을 경우, 빈 문자열을 반환한다.
DATA_TYPE (ODBC 1.0)	5	SMALLINT not NULL	SQL 데이터 타입이다. Datetime이나 interval 데이터 타입의 경우, 이 column은 SQL_TYPE_DATE, SQL_INTERVAL_YEAR_TO_MONTH 같은 concise 데이터 타입을 반환한다.
TYPE_NAME (ODBC 1.0)	6	VARCHAR not NULL	데이터 소스에 종속된 데이터 타입의 이름이다. (예: "CHARACTER", "CHARACTER VARYING", "CHARACTER LONG VARYING")
COLUMN_SIZE (ODBC 1.0)	7	INTEGER	DATA_TYPE이 SQL_CHAR 또는 SQL_VARCHAR인 경우, 이 column은 column의 최대 길이 문자수를 포함한다. Datetime 데이터 column의 경우, 값을 문자로 변환할 때 필요한 문자의 개수이다. 숫자 데이터 타입의 경우, NUM_PREC_RADIX column에 따른 총 자릿수 또는 column에 허용되는 비트 수이다. Interval 데이터 타입의 경우, interval leading precision에 의해 표현될 때 필요한 문자의 개수이다.
BUFFER_LEN			SQL_C_DEFAULT가 지정된 경우, SQLGetData, SQLFetch, SQLFetchScroll

Column 이름	Column 번호	데이터 타입	설명
GTH (ODBC 1.0)	8	INTEGER	로 전송되는 데이터의 바이트 길이이다.
DECIMAL_DIGITS (ODBC 1.0)	9	SMALLINT	양수의 경우 소숫점 오른쪽의 유효 자릿수이다. 음수의 경우 소숫점 왼쪽의 유효 자릿수이다. SQL_TYPE_TIME이나 SQL_TYPE_TIMESTAMP의 경우, 이 column은 fractional seconds의 자릿수이다. Second를 포함하는 interval 데이터 타입의 경우, 이 column은 소숫점 오른쪽에 있는 숫자의 개수 (fractional seconds)이다. DECIMAL_DIGITS를 적용할 수 없는 데이터 타입은 NULL을 반환한다.
NUM_PREC_RADIX (ODBC 1.0)	10	SMALLINT	숫자형 데이터 타입의 경우, 2나 10이다. 2인 경우, COLUMN_SIZE와 DECIMAL_DIGITS는 column이 허용하는 비트 수이다. 10인 경우, COLUMN_SIZE와 DECIMAL_DIGITS는 column이 허용하는 자릿수이다. NUM_PREC_RADIX를 적용할 수 없는 데이터 타입은 NULL을 반환한다.
NULLABLE (ODBC 1.0)	11	SMALLINT not NULL	- SQL_NO_NULLS: Column이 NULL 값을 가질 수 없다. - SQL_NULLABLE: Column이 NULL 값을 가질 수 있다. - SQL_NULLABLE_UNKNOWN: Column이 NULL 값을 허용하는지 여부를 알 수 없다.
REMARKS (ODBC 1.0)	12	VARCHAR	Column에 대한 설명이다.
COLUMN_DEFINITION (ODBC 3.0)	13	VARCHAR	Column의 기본값이다. 이 값이 quote로 묶여 있을 경우, 이 column은 문자열로 해석해야 한다.
SQL_DATA_TYPE (ODBC 3.0)	14	SMALLINT not NULL	IRD의 SQL_DESC_TYPE 레코드 필드의 SQL 데이터 타입이다. 이 column은 datetime과 interval 데이터 타입을 제외하고는 DATA_TYPE과 같다. Datetime 또는 interval 데이터 타입의 경우, 이 column이 SQL_DATE, SQL_INTERVAL과 같은 nonconcise 데이터 타입을 반환하며, SQL_DATETIME_SUB column으로 특정 데이터 타입을 결정할 수 있다.
SQL_DATETIME_SUB (ODBC 3.0)	15	SMALLINT	Datetime과 interval 데이터 타입의 하위 타입 코드이다. 다른 타입의 경우, NULL을 반환한다.
CHAR_OCTET_LENGTH (ODBC 3.0)	16	INTEGER	문자 또는 이진 데이터 타입 column의 바이트 단위 최대 길이이다. 다른 타입의 경우, NULL을 반환한다.
ORDINAL_POSITION (ODBC 3.0)	17	INTEGER not NULL	테이블에서 column의 위치이다.
IS_NULLABLE (ODBC 3.0)	18	VARCHAR	"NO": Column이 NULL을 포함할 수 없다. "YES": Column이 NULL을 포함할 수 있다. - NULL 허용 여부를 알 수 없는 경우, 길이가 0인 문자열을 반환한다.

SQLConnect

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLConnect는 드라이버와 데이터 소스에 대한 연결을 설정한다. 연결 핸들은 상태, 트랜잭션 상태, 에러 정보를 포함하는 연결에 관한 모든 정보를 참고한다.

구문

```
SQLRETURN SQLConnect(  
    SQLHDBC          ConnectionHandle,  
    SQLCHAR *        ServerName,  
    SQLSMALLINT       NameLength1,  
    SQLCHAR *        UserName,  
    SQLSMALLINT       NameLength2,  
    SQLCHAR *        Authentication,  
    SQLSMALLINT       NameLength3);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

ServerName

[입력] 데이터 소스 이름이다.

NameLength1

[입력] *ServerName의 길이이다.

UserName

[입력] 사용자 식별자이다.

NameLength2

[입력] *UserName의 길이이다.

Authentication

[입력] 인증 문자열 (일반적으로 암호)이다.

NameLength3

[입력] *Authentication의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

진단

SQLSTATE	Error	설명
08001	Client unable to establish connection	드라이버는 데이터 소스와의 연결을 설정할 수 없다.
08002	Connection name in use	지정된 ConnectionHandle이 이미 데이터 소스와 연결되어 있다.
08004	Server rejected the connection	설정값 한계 상황에서 데이터 소스가 연결 설정을 거부했다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
28000	Invalid authorization specification	UserName이나 authentication 인자값이 올바르지 않다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY090	Invalid string or buffer length	NameLength1, NameLength2 또는 NameLength3의 값이 0보다 작지만 SQL_NTS가 아니다.
HYT00	Timeout expired	데이터 소스에 연결하기 전에 로그인 제한시간이 만료되었다. 이 제한 시간은 SQLSetConnectAttr의 SQL_ATTR_LOGIN_TIMEOUT을 통해 설정할 수 있다.

설명

드라이버는 사용자 DSN 정보를 \$ODBCINI 환경 변수에 설정된 파일, \$HOME/.odbc.ini, /home/.odbc.ini 파일 순서로 검색한다. 만약 사용자 DSN에 입력된 DSN이 없을 경우, 시스템 DSN인 \$ODBCSYSINI/odbc.ini, /etc/odbc.ini 파일 순서로 DSN 정보를 검색한다.

SQLCopyDesc

지원되지 않는다.

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLCopyDesc는 설명자 정보를 하나의 설명자 핸들에서 다른 설명자 핸들로 복사한다.

구문

```
SQLRETURN SQLCopyDesc(  
    SQLHDESC    SourceDescHandle,  
    SQLHDESC    TargetDescHandle);
```


SQLDescribeCol

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLDescribeCol은 결과 집합 column에서 하나의 column 이름, 타입, column 크기, 소수 자릿수, null 허용 정보를 반환한다. 이 정보는 IRD의 필드들에서 사용할 수 있다.

구문

```
SQLRETURN SQLDescribeCol(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLCHAR *      ColumnName,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   NameLengthPtr,
    SQLSMALLINT *   DataTypePtr,
    SQLULEN *       ColumnSizePtr,
    SQLSMALLINT *   DecimalDigitsPtr,
    SQLSMALLINT *   NullablePtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ColumnNumber

[입력] 1부터 순차적으로 증가하는 결과 집합의 column 번호이다.

ColumnName

[출력] Column 이름을 반환하는, null로 종료되는 버퍼에 대한 포인터이다. 이 값은 IRD의 SQL_DESC_NAME 필드에서 읽을 수 있다. Column 이름이 없거나 column 이름을 확인할 수 없는 경우, 드라이버는 빈 문자열을 반환한다.

ColumnName이 null인 경우, NameLengthPtr은 반환 가능한 전체 바이트 수 (null 종료 문자 제외)를 반환한다.

BufferLength

[입력] *ColumnName의 길이이다.

NameLengthPtr

[출력] *ColumnName에 반환 가능한 전체 바이트 수 (null 종료 문자 제외)를 반환하는 버퍼의 포인터이다. 반환 가능한 길이가 BufferLength보다 크거나 같다면, *ColumnName은 BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘린다.

DataTypePtr

[출력] Column의 SQL 타입을 반환할 버퍼의 포인터이다. 이 값은 IRD의 SQL_DESC_CONCISE_TYPE 필드에서 읽을 수 있다.

ColumnSizePtr

[출력] 데이터 소스의 column 크기를 반환하는 버퍼의 포인터이다.

DecimalDigitsPtr

[출력] 데이터 소스의 소수 자릿수를 반환하는 버퍼의 포인터이다.

NullablePtr

[출력] Column이 null 값을 허용하는지 여부를 반환하는 버퍼의 포인터이다. 이 값은 IRD의 SQL_DESC_NULLABLE 필드에서 읽을 수 있다. 이는 다음 값 중의 하나이다.

- SQL_NO_NULLS: Column이 NULL 값을 허용하지 않는다.
- SQL_NULLABLE: Column이 NULL 값을 허용한다.
- SQL_NULLABLE_UNKNOWN: Column이 NULL 값을 허용하는지 여부를 드라이버가 결정할 수 없다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	*ColumnName이 전체 column 이름을 반환하는데 충분한 크기가 아니어서 column 이름이 잘렸다. 잘리지 않는 column의 길이는 *NameLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07005	Prepared statement not a cursor-specification	명령문은 결과 집합을 반환하지 않으므로 설명할 column이 없다.
07009	Invalid descriptor index	ColumnNumber 인자의 값이 결과 집합의 column 수보다 크다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
	Memory allocation failure	

SQLSTATE	Error	설명
HY001	n failure	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	BufferLength 인자값이 0보다 작다.

설명

일반적으로 응용 프로그램은 SQLPrepare를 호출한 후에 관련된 SQLExecute 호출 전후로 SQLDescribeCol을 호출한다. 또한 응용 프로그램은 SQLExecDirect를 호출한 후에 SQLDescribeCol을 호출할 수 있다.

SQLDescribeParam

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLDescribeParam은 준비된 SQL 문과 관련된 매개 변수 마커의 설명을 반환한다. 이 정보는 IPD의 필드들에서 사용할 수 있다.

구문

```
SQLRETURN SQLDescribeParam(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ParameterNumber,
    SQLSMALLINT *  DataTypePtr,
    SQLULEN *      ParameterSizePtr,
    SQLSMALLINT *  DecimalDigitsPtr,
    SQLSMALLINT *  NullablePtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ParameterNumber

[입력] 1부터 시작해 순차적으로 증가하는 매개 변수 마커 번호이다.

DataTypePtr

[출력] 매개 변수의 SQL 타입을 반환하는 버퍼의 포인터이다. 이 값은 IPD의 SQL_DESC_CONCISE_TYPE 레코드 필드에서 읽을 수 있다.

ParameterSizePtr

[출력] 해당 매개 변수 마커에 해당하는 column 또는 표현식의 크기를 반환하는 버퍼의 포인터이다.

DecimalDigitsPtr

[출력] 해당 매개 변수 마커에 해당하는 column 또는 표현식의 소수 자릿수를 반환하는 버퍼의 포인터이다.

NullablePtr

[출력] 매개 변수가 NULL을 허용하는지 여부를 나타내는 버퍼의 포인터이다. 이 값은 IPD의 SQL_DESC_NULLABLE 필드에서 읽을 수 있다. 이는 다음 값 중의 하나이다.

- SQL_NO_NULLS: 매개 변수가 NULL 값을 허용하지 않는다.
- SQL_NULLABLE: 매개 변수가 NULL 값을 허용한다.
- SQL_NULLABLE_UNKNOWN: 매개 변수가 NULL 값을 허용하는지 여부를 드라이버가 결정할 수 없다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
07009	Invalid descriptor index	ParameterNumber 인자의 값이 1보다 작다. ParameterNumber 인자의 값이 관련된 SQL 문의 매개 변수 개수보다 크다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLPrepare나 SQLExecDirect보다 이 함수가 먼저 호출되었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.

설명

드라이버가 준비된 SQL 문으로는 정확한 매개 변수의 정보를 줄 수 없어, 무조건 *DataTypePtr에 SQL_VARCHAR, *ParameterSizePtr에 4000, *DecimalDigitsPtr에 0, *NullablePtr에 SQL_NULLABLE을 반환한다.

SQLDisconnect

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLDisconnect는 특정 연결 핸들과 관련된 연결을 닫는다.

구문

```
SQLRETURN SQLDisconnect(  
    SQLHDBC      ConnectionHandle);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

진단

SQLSTATE	Error	설명
08003	Connection not open	ConnectionHandle 인자의 연결이 열려 있지 않다.
25000	Invalid transaction state	ConnectionHandle 인자의 연결에 트랜잭션이 진행 중이다. 트랜잭션은 활성화 상태로 유지된다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.

설명

응용 프로그램이 완료되지 않은 트랜잭션이 있는 연결 핸들로 SQLDisconnect를 호출하는 경우, 드라이버가 SQLSTATE 25000 (invalid transaction state)을 반환하며, 트랜잭션은 변경되지 않고, 연결은 열려 있다. 완료되지 않은 트랜잭션이란 SQLEndTran으로 commit 또는 rollback 되지 않은 트랜잭션이다.

응용 프로그램이 모든 명령문을 해제하기 전에 SQLDisconnect를 호출하면, 드라이버가 데이터 소스에서 연결을 해제한 후에 모든 명령문을 삭제하고 연결 핸들에 명시적으로 할당된 설명자를 삭제한다.

SQLDriverConnect

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLDriverConnect는 SQLConnect를 대체할 수 있고 SQLConnect의 세 개 인자보다 더 많은 연결 정보를 필요로 하는 데이터 소스를 지원한다.

SQLDriverConnect는 데이터 소스의 이름, 하나 이상의 사용자, 하나 이상의 암호 및 데이터 소스가 요구하는 다른 정보를 포함하는 연결 문자열을 이용해 연결을 설정한다.

연결이 설정되면, SQLDriverConnect는 완성된 연결 문자열을 반환한다. 응용 프로그램은 다음 연결을 요청할 때 이 문자열을 사용할 수 있다.

구문

```
SQLRETURN SQLDriverConnect(
    SQLHDBC          ConnectionHandle,
    SQLHWND          WindowHandle,
    SQLCHAR *         InConnectionString,
    SQLSMALLINT       StringLength1,
    SQLCHAR *         OutConnectionString,
    SQLSMALLINT       BufferLength,
    SQLSMALLINT *     StringLength2Ptr,
    SQLUSMALLINT      DriverCompletion);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

WindowHandle

[입력] 윈도우 핸들이다. 응용 프로그램은 상위 윈도우의 핸들을 전달하거나 null 포인터를 전달하며, null 포인터의 경우, SQLDriverConnect가 대화 상자를 표시하지 않는다.

InConnectionString

[입력] 전체 연결 문자열, 부분 연결 문자열 또는 빈 문자열이다.

StringLength1

[입력] *InConnectionString의 길이이다.

OutConnectionString

[출력] 완성된 연결 문자열의 버퍼 포인터이다. 대상 데이터 소스에 성공적으로 연결하면, 이 버퍼가 완성된 연결 문자열을 포함한다. 응용 프로그램은 적어도 1,024 문자 버퍼를 할당해야 한다.

OutConnectionString이 NULL인 경우, StringLength2Ptr에는 반환 가능한 총 문자수 (null 종료 문자 제외)를 반환한다.

BufferLength

[입력] *OutConnectionString의 길이이다.

StringLength2Ptr

[출력] *OutConnectionString에 반환 가능한 전체 문자 수 (null 종료 문자 제외)를 반환하는 버퍼의 포인터이다. 반환 가능한 길이가 BufferLength보다 크거나 같다면, *OutConnectionString은 BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘린다.

DriverCompletion

[입력] 드라이버가 더 많은 정보를 요청해야 하는지 여부를 나타내는 플래그로써 SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, 또는 SQL_DRIVER_NOPROMPT이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, SQL_S
TILL_EXECUTING

진단

SQLSTATE	Error	설명
01004	String data, right truncated	*OutConnectionString 버퍼가 전체 연결 문자열을 반환하는데 충분한 크기가 아니라서 연결 문자열이 잘렸다. 잘리지 않은 연결 문자열의 길이는 *StringLength2Ptr로 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08001	Client unable to establish connection	드라이버는 데이터 소스와 연결을 설정할 수 없다.
08002	Connection name in use	지정된 ConnectionHandle이 이미 데이터 소스와 연결되어 있다.
08004	Server rejected the connection	설정값 한계 상황에서, 데이터 소스는 연결 설정을 거부했다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.

SQLSTATE	Error	설명
28000	Invalid authorization specification	연결 문자열의 사용자 식별자 및 인증 문자열이 올바르지 않다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY090	Invalid string or buffer length	StringLength1 인자값이 0보다 작지만 SQL_NTS가 아니다. BufferLength 인자값이 0보다 작다.
HY110	Invalid driver completion	DriverCompletion 인자값이 SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, SQL_DRIVER_NOPROMPT가 아니다.
HYC00	Optional feature not implemented	드라이버가 응용 프로그램이 요구하는 ODBC 동작을 지원하지 않는다.
HYT00	Timeout expired	데이터 소스에 연결하기 전에 로그인 제한시간이 만료되었다. 이 제한 시간은 SQLSetConnectAttr의 SQL_ATTR_LOGIN_TIMEOUT을 통해 설정할 수 있다.

설명

연결 문자열의 구문은 다음과 같다.

```
connection-string ::= empty-string[;] | attribute[;] | attribute; connection-string
empty-string ::= attribute ::= attribute-keyword=attribute-value | DRIVER=[{}attribute-value{}]
attribute-keyword ::= DSN | PROTOCOL | CS_MODE | HOST | PORT | TCP_NODELAY | UID | PWD |
ALTERNATE_SERVERS | FAILOVER_TYPE | FAILOVER_GRANULARITY | DATE_FORMAT | TIME_FORMAT | TIME_
WITH_TIME_ZONE_FORMAT | TIMESTAMP_FORMAT | TIMESTAMP_WITH_TIME_ZONE_FORMAT | CHAR_LENGTH_UNITS
| CONN_NAME
attribute-value ::= character-string
```

character-string은 0개 이상의 문자이다. attribute-keyword는 대소문자를 구분하지 않지만 attribute-value는 대소문자를 구분한다. DSN 키워드의 값은 공백만으로 구성되지 않는다.

attribute-keyword는 다음 표와 같다.

키워드	설명
DSN	데이터 소스 이름이다.
PROTOCOL	연결 방식 (DA/ TCP) 이다.
CS_MODE	Dedicated 모드로 접속할지 shared 모드로 접속할지를 설정한다. 이 설정을 사용하지 않을 경우, listener의 configuration (DEFAULT_CS_MODE)에 따라서 모드가 결정된다.
HOST	호스트의 IP 주소이다.
PORT	연결 포트 번호이다.
TCP_NODELAY	socket TCP_NODELAY 옵션이다.
UID	사용자 ID 이다.

키워드	설명
PWD	사용자 ID에 대한 암호이다. 사용자 ID의 암호가 없을 경우 빈 문자열 (PWD=;)이다.
ALTERNATE_SERVERS	Failover가 발생할 경우, 연결을 시도하는 서버 리스트로써 각 서버는 콤마 (,)로 구분한다. Failover 기능을 사용하지 않을 경우, ALTERNATE_SERVERS를 설정하지 않는다.
FAILOVER_TYPE	• CONNECTION: 연결에 실패할 경우, ALTERNATE_SERVERS로 연결한다. • SESSION: 연결에 실패하거나 statement 동작 중에 연결이 끊어졌을 경우, ALTERNATE_SERVERS로 연결한 후에 statement를 복원한다. 연결이 끊어질 때 진행 중인 트랜잭션이 없었을 경우, failover 후에 진행 중인 statement를 수행한다.
FAILOVER_GRANULARITY	• 0: Failover 진행 중에 에러가 발생해도 failover를 계속 진행한다. • 1: Failover 진행 중에 SQLExeecute(), SQLExecDirect()를 제외한 에러가 발생할 경우 failover에 실패한다. • 2: Failover 진행 중에 에러가 발생할 경우 failover에 실패한다.
DATE_FORMAT	DATE 타입 형식 문자열이다.
TIME_FORMAT	TIME 타입 형식 문자열이다.
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE 타입 형식 문자열이다.
TIMESTAMP_FORMAT	TIMESTAMP 타입 형식 문자열이다.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE 타입 형식 문자열이다.
CHAR_LENGTH_UNITS	SQLBindParameter()에서 ParameterType이 SQL_CHAR, SQL_VARCHAR 일 경우 ColumnSize의 단위이다. • BYTE, OCTETS: 바이트 단위 • CHAR, CHARACTERS: 문자 단위
CONN_NAME	XA에서 사용되는 connection name이다. 여기에 지정된 이름은 embedded SQL 프로그램에서만 유효하며, 그 외에는 무시된다.

SQLEndTran

적합성

도입된 버전: ODBC 3.0
표준 준수: ISO 92

개요

SQLEndTran은 연결과 관련된 모든 명령문의 활성 작업에 대해 commit 또는 rollback 작업을 요청한다.

구문

```
SQLRETURN SQLEndTran(  
    SQLSMALLINT  HandleType,  
    SQLHANDLE     Handle,  
    SQLSMALLINT  CompletionType);
```

인자

HandleType

[입력] 핸들 식별자이다. 환경 핸들일 경우 SQL_HANDLE_ENV이어야 하고, 연결 핸들일 경우 SQL_HANDLE_DBC이어야 한다.

Handle

[입력] 트랜잭션 범위를 나타내는 HandleType의 핸들이다.

CompletionType

[입력] SQL_COMMIT 또는 SQL_ROLLBACK이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

진단

SQLSTATE	Error	설명
08003	Connection not open	HandleType이 SQL_HANDLE_DBC이고, handle은 연결 상태가 아니다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.

SQLSTATE	Error	설명
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY012	Invalid transaction operation code	CompletionType 인자값이 SQL_COMMIT나 SQL_ROLLBACK가 아니다.
HY092	Invalid attribute/option identifier	HandleType 인자값이 SQL_HANDLE_ENV나 SQL_HANDLE_DBC가 아니다.

설명

CompletionType이 SQL_COMMIT일 경우, SQLEndTran은 연결과 관련된 모든 명령문에 대한 활성 작업에 commit을 요청한다. CompletionType이 SQL_ROLLBACK일 경우, SQLEndTran은 연결과 관련된 모든 명령문에 대한 활성 작업에 rollback을 요청한다. 활성화된 트랜잭션이 없을 경우, SQLEndTran은 데이터 소스에 영향을 미치지 않고, SQL_SUCCESS를 반환한다.

드라이버가 수동 commit 모드 (SQLSetConnectAttr을 호출하여 SQL_ATTR_AUTOCOMMIT 속성을 SQL_AUTOCOMMIT_OFF로 설정)일 경우, SQL 명령문이 현재 데이터 소스에 대해 실행되면 새 트랜잭션이 암시적으로 시작된다.

SQLEndTran은 commit 할 때 연결과 관련된 열린 커서에 영향을 미치지 않는다. 커서는 SQLEndTran을 호출하기 전에 가리키는 row에 남아 있다.

SQLEndTran은 rollback 할 때 모든 명령문에서 열려 있는 모든 커서를 닫는다. SQLEndTran은 명령문을 준비 상태로 두며, 응용 프로그램은 SQLPrepare를 호출하지 않고, SQLExecute를 호출할 수 있다.

활성화된 트랜잭션이 없을 경우, SQLEndTran은 SQL_SUCCESS를 반환한다.

드라이버가 자동 commit 모드일 경우, SQLEndTran은 CompletionType에 관계없이 항상 SQL_SUCCESS를 반환한다.

SQLException

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLException은 에러나 상태 정보를 반환한다.

SQLExecDirect

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLExecDirect는 명령문에 매개 변수가 있을 경우, 매개 변수 마커의 현재 값을 사용하여 명령문을 수행한다. SQLExecDirect는 한 번만 실행되는 명령문을 위한 가장 빠른 방법이다.

구문

```
SQLRETURN SQLExecDirect(  
    SQLHSTMT      StatementHandle,  
    SQLCHAR *      StatementText,  
    SQLINTEGER     TextLength);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

StatementText

[입력] 실행된 SQL 명령문이다.

TextLength

[입력] *StatementText의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	입력/ 출력 또는 출력 매개 변수에 반환된 문자열이나 이진 데이터가 잘렸다. 문자열이 잘린 경우, 오른쪽이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)

SQLSTATE	Error	설명
01S02	Option value changed	지정된 명령문 속성값이 작업을 실행하기에 적합하지 않아 일시적으로 비슷한 값으로 대체되었다. (SQLGetStmtAttr을 호출하여 일시적으로 변경된 값을 확인할 수 있다.) 이 대체값은 커서가 닫힐 때까지 유효하며, 커서가 닫힐 때 이전 값으로 변경된다. 변경할 수 있는 명령문 속성은 다음과 같다. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	SQLBindParameter의 ValueType 인자에 의해 식별되는 데이터 값을 SQLBindParameter의 ParameterType 인자에 의해 식별되는 데이터 형식으로 변환할 수 없다. SQL_PARAM_INPUT_OUTPUT이나 SQL_PARAM_OUTPUT 매개 변수에 반환된 데이터 값을 SQLBindParameter의 ValueType 인자로 식별된 데이터 형식으로 변환할 수 없다. (하나 이상의 row가 성공적으로 반환될 경우, 함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07007	Restricted parameter value violation	매개 변수 타입이 SQL_PARAM_INPUT_OUTPUT이고, SQLBindParameter의 *StrLen_or_IndPtr이 SQL_NULL_DATA, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC(len), 또는 SQL_DATA_AT_EXEC가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22001	String data, right truncation	문자열이나 이진 데이터가 잘렸다.
22002	Indicator variable required but not supplied	SQLBindParameter의 StrLen_or_IndPtr이 null 포인터인 출력 매개 변수에 NULL 데이터가 바인딩 되었다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 통해 StatementHandle에 커서를 위치시켰다. 커서는 열려있지만 StatementHandle에 위치하지 않고 있다. *StatementText는 positioned update 또는 delete 명령문이고, 커서는 결과 집합의 시작 전이나 끝 이후에 위치하고 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	*StatementText가 null 포인터이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
		TextLength 인자값이 0보다 작지만 SQL_NTS가 아니다.

SQLSTATE	Error	설명
HY090	Invalid string or buffer length	SQLBindParameter로 설정한 매개 변수 값이 null 포인터이고, 매개 변수 길이는 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC_OFFSET 이하의 값이 아니다. SQLBindParameter로 설정한 매개 변수 값이 null 포인터가 아니고, C 데이터 타입이 SQL_C_BINARY 또는 SQL_C_CHAR이며, 매개 변수 길이는 0보다 작지만 SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC_OFFSET 이하의 값이 아니다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

응용 프로그램은 SQLExecDirect를 호출하여 SQL 명령문을 데이터 소스로 보낸다.

응용 프로그램은 SQL 명령문에 한 개 이상의 매개 변수 표시 문자를 포함할 수 있다. 매개 변수 마커를 포함하려면, 응용 프로그램이 SQL 명령문의 적절한 위치에 물음표 (?)를 포함해야 한다.

SQL 명령문이 SELECT문이고, 응용 프로그램이 SQLSetCursorName으로 커서를 연결하는 경우, 드라이버는 지정된 커서를 사용한다. 응용 프로그램이 명령문과 커서를 연결하지 않은 경우, 드라이버는 커서 이름을 생성한다.

데이터 소스가 수동 commit 모드이고 트랜잭션이 아직 시작되지 않은 경우, 드라이버는 SQL 명령문을 전송하기 전에 트랜잭션을 시작한다.

SQLExecDirect가 data-at-execution 매개 변수를 발견하면 SQL_NEED_DATA를 반환한다. 응용 프로그램은 SQL ParamData와 SQLPutData를 사용하여 데이터를 전송한다.

SQLExecDirect가 검색 조건이 있는 갱신, 삽입, 삭제 명령문을 수행했으나 데이터 소스에 갱신된 row가 없을 경우, SQLExecDirect 호출은 SQL_NO_DATA를 반환한다.

SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1보다 크고, SQL 명령문에 최소 하나의 매개 변수 표시 문자가 포함된 경우, SQLExecDirect는 SQLBindParameter의 ParameterValuePtr 인자가 가리키는 배열에서 매개 변수 집합에 대해 SQL 명령문을 한 번씩 수행한다.

SQLExecute

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLExecute는 명령문에 매개 변수가 있을 경우, 매개 변수 마커의 현재 값을 사용하여 준비된 명령문을 수행한다.

구문

```
SQLRETURN SQLExecute(  
    SQLHSTMT      StatementHandle);
```

인자

StatementHandle
[입력] 명령문 핸들이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	입력/ 출력 또는 출력 매개 변수에 반환된 문자열이나 이진 데이터가 잘렸다. 문자열이 잘린 경우, 오른쪽이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	지정된 명령문 속성값이 작업을 실행하기에 적합하지 않아 일시적으로 비슷한 값으로 대체되었다. (SQLGetStmtAttr을 호출하여 일시적으로 변경된 값을 확인할 수 있다.) 이 대체값은 커서가 닫힐 때까지 유효하며, 커서가 닫힐 때 이전 값으로 변경된다. 변경할 수 있는 명령문 속성은 다음과 같다. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUE

SQLSTATE	Error	설명
		RY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	SQLBindParameter의 ValueType 인자에 의해 식별된 데이터 값을 SQLBindParameter의 ParameterType 인자에 의해 식별되는 데이터 형식으로 변환할 수 없다. SQL_PARAM_INPUT_OUTPUT이나 SQL_PARAM_OUTPUT 매개 변수에 반환된 데이터 값을 SQLBindParameter의 ValueType 인자로 식별된 데이터 형식으로 변환할 수 없다. (하나 이상의 row가 성공적으로 반환될 경우, 함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07007	Restricted parameter value violation	매개 변수 타입이 SQL_PARAM_INPUT_OUTPUT이고, SQLBindParameter의 *StrLen_or_IndPtr이 SQL_NULL_DATA, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC(len), 또는 SQL_DATA_AT_EXEC가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22001	String data, right truncation	문자열이나 이진 데이터가 잘렸다.
22002	Indicator variable required but not supplied	SQLBindParameter의 StrLen_or_IndPtr이 null 포인터인 출력 매개 변수에 NULL 데이터가 바인딩 되었다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 통해 StatementHandle에 커서를 위치시켰다. 커서는 열려있지만 StatementHandle에 위치하지 않고 있다. *StatementText는 positioned update 또는 delete 명령문이고, 커서는 결과 집합의 시작 전이나 끝 이후에 위치하고 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다. StatementHandle이 준비되지 않았다.
HY090	Invalid string or buffer length	TextLength 인자값이 0보다 작지만 SQL_NTS가 아니다. SQLBindParameter로 설정한 매개 변수 값이 null 포인터이고, 매개 변수 길이는 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC_OFFSET 이하의 값이 아니다. SQLBindParameter로 설정한 매개 변수 값이 null 포인터가 아니고, C 데이터 타입이 SQL_C_BINARY 또는 SQL_C_CHAR이며, 매개 변수 길이는 0보다 작지만 SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, SQL_LEN_DATA_AT_EXEC_OFFSET 이하의 값이 아니다.

SQLSTATE	Error	설명
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

SQLExecute는 SQLPrepare에 의해 준비된 명령문을 실행한다. 응용 프로그램은 SQLExecute 호출 결과를 버린 후, 새 매개 변수 값을 사용하여 SQLExecute를 다시 호출할 수 있다.

SELECT 명령문을 한 번 이상 실행하려면, 응용 프로그램이 SELECT 명령문을 다시 실행하기 전에 반드시 SQLCloseCursor를 호출해야 한다.

데이터 소스가 수동 commit 모드이고 트랜잭션이 아직 시작되지 않은 경우, 드라이버는 SQL 명령문을 전송하기 전에 트랜잭션을 시작한다.

SQLExecute가 data-at-execution 매개 변수를 발견하면 SQL_NEED_DATA를 반환한다. 응용 프로그램은 SQLParamData와 SQLPutData를 사용하여 데이터를 전송한다.

SQLExecute가 검색 조건이 있는 갱신, 삽입, 삭제 명령문을 수행했으나 데이터 소스에 갱신된 row가 없을 경우, SQLExecute 호출은 SQL_NO_DATA를 반환한다.

SQL_ATTR_PARAMSET_SIZE 명령문 속성값이 1보다 크고, SQL 명령문에 최소 하나의 매개 변수 표시 문자가 포함된 경우, SQLExecute는 SQLBindParameter의 ParameterValuePtr 인자가 가리키는 배열에서 매개 변수 집합에 대해 SQL 명령문을 한 번씩 수행한다.

SQLExtendedFetch

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLExtendedFetch는 결과 집합으로부터 지정된 데이터 집합을 가져와서 바인딩된 모든 column에 반환한다.

구문

```
SQLRETURN SQLExtendedFetch(  
    SQLHSTMT          StatementHandle,  
    SQLUSMALLINT       FetchOrientation,  
    SQLLEN             FetchOffset,  
    SQLULEN *          RowCountPtr,  
    SQLUSMALLINT *     RowStatusArray);
```

SQLFetch

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLFetch는 결과 집합에서 다음 row 집합을 가져와서 바인딩 된 모든 column에 반환한다.

구문

```
SQLRETURN SQLFetch(  
    SQLHSTMT      StatementHandle);
```

인자

StatementHandle
[입력] 명령문 핸들이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	Column에 대해 반환된 문자열이나 이진 데이터가 잘렸다. 문자열이 잘린 경우, 오른쪽이 잘렸다.
01S07	Fractional truncation	Column에 대해 반환된 데이터가 잘렸다. 숫자 데이터 타입이 잘린 경우, 소수 부분이 잘렸다. 기간 구성 요소를 포함하는 time, timestamp, interval 데이터 타입이 잘린 경우, 시간의 소수 부분이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	결과 집합에 있는 column 데이터 값을 SQLBindCol의 TargetType이 지정한 데이터 타입으로 변환할 수 없다.
	Communication I	

SQLSTATE	Error	설명
08S01	link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22002	Indicator variable required but not supplied	SQLBindCol의 StrLen_or_IndPtr (또는 SQLSetDescField, SQLSetDescRec로 설정된 SQL_DESC_INDICATOR_PTR)이 null 포인터인 column에 NULL 데이터를 가져왔다.
22003	Numeric value out of range	하나 이상의 column에서 반환된 숫자값의 (소수가 아닌) 정수 부분이 잘렸다.
22007	Invalid datetime format	결과 집합의 문자열이 유효한 date, time, timestamp 형식이 아니다.
22012	Division by zero	0으로 나눈 산술식의 결과값이 반환되었다.
22015	Interval field overflow	Exact numeric 또는 interval SQL 데이터 타입을 interval C 타입으로 지정할 경우, 선행 필드의 유효 숫자가 손실되었다. SQL 타입의 값을 interval C 타입으로 표현할 수 없다.
22018	Invalid character value for cast specification	C 버퍼의 character set으로 표현할 수 없는 문자를 결과 집합의 문자 column이 포함하였다. C 타입은 exact 또는 approximate numeric, datetime, interval 데이터 타입이고 SQL 타입은 문자 데이터 형식일 때, C 타입에 바인딩 되는 column의 값이 유효한 문자가 아니다.
24000	Invalid cursor state	StatementHandle이 실행된 상태이지만 StatementHandle과 관련된 결과 집합이 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	지정된 StatementHandle이 실행 상태가 아니다. 이 함수는 SQLExecDirect, SQLExecute, 카탈로그 함수를 호출하지 않고 호출되었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환 되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한 시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

SQLFetch는 결과 집합에서 다음 데이터 집합을 반환한다. SQLFetch는 결과 집합이 생성되고 커서를 닫기 전까지, 결과 집합이 존재하는 동안만 호출할 수 있다. Column이 바인딩 되어 있을 경우, column에 데이터를 반환한다. 응용 프로그램이 row 상태 배열에 대한 포인터 또는 가져온 row 개수를 반환하는 버퍼를 지정할 경우, SQLFetch는 이 정보도 반환한다. SQLFetch와 SQLFetchScroll은 섞어서 호출할 수 있다.

커서의 위치

결과 집합이 생성되면, 커서는 결과 집합의 시작 전에 위치한다. SQLFetch는 다음 row 집합을 가져온다. 이는 Fetch Orientation이 SQL_FETCH_NEXT로 설정된 SQLFetchScroll을 호출하는 것과 같다.

SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성은 row 집합의 row 개수를 지정한다. SQLFetch로 가져오는 row 집합이 결과 집합의 끝과 겹치면, SQLFetch는 부분 row 집합을 반환한다. 이는 S를 가져오는 row 집합의 시작 row, R을 row 집합의 크기, L을 결과 집합의 마지막 row라고 할 때, $S + R - 1$ 이 L보다 큰 경우이며, row 집합의 처음 $L - S + 1$ 개의 row만 유효하다. 나머지 row는 모두 비어 있고, SQL_ROW_NOROW 상태가 된다.

SQLFetch를 반환한 후의 현재 row는 row 집합의 첫 번째 row이다.

다음 표의 규칙은 이 세션의 두 번째 표에 있는 조건에 따라 SQLFetch를 호출한 후의 커서의 위치를 설명한다.

상태	새 row 집합의 첫 번째 row
Before start	1
$\text{CurrRowsetStart} \leq \text{LastResultRow} - \text{RowsetSize}$ ^[1]	$\text{CurrRowsetStart} + \text{RowsetSize}$ ^[2]
$\text{CurrRowsetStart} > \text{LastResultRow} - \text{RowsetSize}$ ^[1]	After end
After end	After end

^[1]Fetch 하는 도중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 이 fetch 이전에 사용된 row 집합의 크기이다.

^[2]Fetch 하는 도중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 새로운 fetch에 사용된 row 집합의 크기이다.

표기법	의미
Before start	블록 커서는 결과 집합의 시작 전에 위치한다. 새로운 row 집합의 첫 번째 row가 결과 집합의 시작 전에 위치할 경우, SQLFetch는 SQL_NO_DATA를 반환한다.
After end	블록 커서는 결과 집합의 끝 이후에 위치한다. 새로운 row 집합의 첫 번째 row가 결과 집합의 끝 이후에 위치할 경우, SQLFetch는 SQL_NO_DATA를 반환한다.
CurrRowsetStart	현재 row 집합의 첫 번째 row 번호이다.
LastResultRow	결과 집합의 마지막 row 번호이다.
RowsetSize	Row 집합의 크기이다.

다음은 결과 집합에 100 개의 row가 있고, row 집합의 크기가 5인 예이다. 다음 표는 SQLFetch가 다른 시작 위치에 대해 반환한 row 집합과 반환 코드를 보여준다.

현재 row 집합	반환 코드	새로운 row 집합	Fetch된 row 개수
Before start	SQL_SUCCESS	1 to 5	5
1 to 5	SQL_SUCCESS	6 to 10	5
52 to 56	SQL_SUCCESS	57 to 61	5
91 to 95	SQL_SUCCESS	96 to 100	5
		98 to 100	

현재 row 집합	반환 코드	새로운 row 집합	Fetch된 row 개수
93 to 97	SQL_SUCCESS	Row 상태 배열의 4, 5 row는 SQL_ROW_NOROW로 설정된다.	3
96 to 100	SQL_NO_DATA	없음	0
99 to 100	SQL_NO_DATA	없음	0
After end	SQL_NO_DATA	없음	0

바인딩 column에서 데이터 반환

SQLFetch가 각 row를 반환하는 것처럼, column에 바인딩 된 버퍼 안의 각 바인딩 column에 데이터를 입력한다. 바인딩 된 column이 없으면 SQLFetch는 아무 데이터도 반환하지 않고 커서를 앞으로 이동시키지도 않는다. 데이터는 SQLGetData를 통해 계속 회수될 수 있다. 커서가 다중 row 커서 (SQL_ATTR_ROW_ARRAY_SIZE 가 1 보다 클 때)일 경우, SQLGetData는 SQLGetInfo의 InfoType을 SQL_FETCHDATA_EXTENSIONS로 설정했을 때와 SQL_GD_BLOCK이 반환됐을 때 호출할 수 있다.

자세한 내용은 **SQLGetData** 를 참조한다.

SQLFetch는 row 속의 각 바인딩 column에 대해 다음을 수행한다.

1. 데이터가 NULL인 경우 길이/ 지시자 버퍼를 SQL_NULL_DATA로 설정하고 다음 column으로 진행한다. 데이터가 NULL일 때 길이/ 지시자 버퍼를 바인딩하지 않을 경우, SQLFetch는 row와 다음으로 이행할 row에 대해 SQLSTATE 22002 (indicator variable required but not supplied)를 반환한다. 길이/ 지시자의 주소 결정에 대한 자세한 내용은 SQLBindCol의 **버퍼 주소**를 참조한다. Column의 데이터가 NULL이 아닐 경우, SQLFetch는 2번 과정을 수행한다.
2. SQL_ATTR_MAX_LENGTH 명령문 속성을 0이 아닌 값으로 설정하고 column이 문자 또는 바이너리 데이터를 포함할 경우, 데이터는 SQL_ATTR_MAX_LENGTH 바이트 길이로 잘린다.

노트

SQL_ATTR_MAX_LENGTH 명령문 속성은 네트워크 트래픽 절감을 목적으로 한다. 이것은 일반적으로 데이터 소스에서 구현되며 데이터가 네트워크에서 반환되기 전에 데이터를 자른다. 드라이버와 데이터 소스들이 이것을 지원하도록 요구하지 않으므로 데이터가 특정 크기로 잘리도록 보장하려면 응용 프로그램이 SQLBindCol의 cbValueMax 인자에 그 크기를 명시하고 버퍼를 생성해야 한다.

3. 데이터를 SQLBindCol의 TargetType에 명시된 형식으로 변환한다.
4. 데이터가 문자나 바이너리 같은 가변 길이 데이터 형식으로 변환된 경우, SQLFetch는 데이터의 길이가 데이터 버퍼의 길이를 초과하는지 확인한다. (NULL 종료문자를 포함하는) 문자 데이터의 길이가 데이터 버퍼의 길이를 초과할 경우, SQLFetch가 데이터를 NULL 종료문자 길이보다 작은 데이터버퍼 길이로 자른다. 이 때 데이터는 NULL로 종료한다. 바이너리 데이터의 길이가 데이터 버퍼길이를 초과할 경우, SQLFetch는 그것을 데이터버퍼 길이로 자른다. 데이터 버퍼의 길이는 SQLBindCol의 BufferLength에 명시된다. SQLFetch는 고정길이 데이터 형식으로 변환되는 데이터는 절대 자르지 않는데 데이터 버퍼의 길이는 항상 데이터 형식의 길이와 같기 때문이다.

다.

5. 데이터 버퍼에 변환된 (가능하다면 절삭된) 데이터를 입력한다. 데이터 버퍼의 주소 결정에 대한 자세한 내용은 SQLBindCol의 를 참조한다.
6. 길이/ 지시자 버퍼에 데이터의 길이를 입력한다. 같은 버퍼에 지시자 포인터와 길이 포인터가 모두 설정될 경우, (SQLBindCol을 호출하여) 버퍼에 유효한 데이터의 길이가 기록되고, NULL 데이터의 경우 버퍼에 SQL_NULL_DATA가 기록된다. 길이/ 지시자 버퍼가 바인딩 되지 않았을 경우, SQLFetch는 길이를 반환하지 않는다.
 - 문자 또는 바이너리 데이터의 경우 버퍼 크기가 매우 작기 때문에 이것은 데이터를 변환한 후 자르기 전의 데이터 길이이다. 매우 긴 데이터일 경우, 드라이버가 변환 후 데이터 길이를 결정할 수 없을 때, 길이를 SQL_NO_TOTAL로 설정한다. 데이터가 SQL_ATTR_MAX_LENGTH 명령문 속성에 의해 잘린 경우, 실제 길이 대신 길이/ 지시자 버퍼에 이 속성값을 입력하는데 이 속성이 변환 전 서버에서 데이터를 자르도록 설계되어 있기 때문이다. 따라서 드라이버가 실제 길이를 계산할 수 있는 방법이 없다.
 - 다른 모든 데이터 형식들의 경우, 이것은 변환 후 데이터의 길이이다.
7. 데이터가 변환되는 동안 유효 숫자의 손실없이 잘렸다면 (예를 들어 실수 1.234가 잘려서 1로 변환된 경우) SQLFetch는 SQLSTATE 01S07 (fractional truncation)과 SQL_SUCCESS_WITH_INFO를 반환한다. 데이터 버퍼의 길이가 너무 작아서 데이터가 잘릴 경우, (예를 들어 문자열 "abcdef"가 4 바이트의 버퍼에 들어 가는 경우) SQLFetch는 SQLSTATE 01004 (data truncated)와 SQL_SUCCESS_WITH_INFO를 반환한다. 데이터가 SQL_ATTR_MAX_LENGTH 명령문 속성에 의해 잘린 경우, SQLFetch는 SQL_SUCCESS를 반환하고 SQLSTATE 01S07 (fractional truncation)이나 SQLSTATE 01004 (data truncated)를 반환하지 않는다. 데이터가 변환되는 동안 유효숫자가 잘린 경우, (예를 들어 100,000를 넘는 SQL_INTEGER 값이 SQL_C_TINYINT로 변환되는 경우) SQLFetch는 SQLSTATE 22003 (numeric value out of range)와 (row 집합 크기가 1일 경우) SQL_ERROR 또는 (row 집합 크기가 1보다 큰 경우) SQL_SUCCESS_WITH_INFO를 반환한다.

만약 SQLFetch, SQLFetchScroll의 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 바인딩 데이터 버퍼와 길이/ 지시자 버퍼의 내용은 정의되지 않는다.

Row 상태 배열

Row 상태 배열은 row 집합 각각의 상태를 반환하는데 사용된다. 이 배열의 주소는 SQL_ATTR_ROW_STATUS_PTR 명령문 속성으로 지정된다. 이 배열은 SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성에 의해 지정된 수만큼 응용 프로그램에서 요소를 할당해야 한다. 이 값은 SQLFetch, SQLFetchScroll, SQLBulkOperations 또는 SQLSetPos에 의해 설정된다. SQL_ATTR_ROW_STATUS_PTR 명령문 속성의 값이 null 포인터인 경우, 이 함수는 row의 상태를 반환하지 않는다.

SQLFetch, SQLFetchScroll에서 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, row 상태 버퍼의 내용은 정의되지 않는다.

다음 값이 row 상태 배열에 반환된다.

Row 상태 배열 값	설명
SQL_ROW_SUCCESS	Row를 성공적으로 가져왔으며 결과 집합에서 마지막으로 가지고 온 후 갱신되지 않았다.
SQL_ROW_SUCCESS_WITH_INFO	Row를 성공적으로 가져왔으며 결과 집합에서 마지막으로 가지고 온 후 갱신되지 않았다. 그러나 row에 관해 경고가 반환되었다.

Row 상태 배열 값	설명
SQL_ROW_ERROR	Row를 가지고 오는 동안 에러가 발생하였다.
SQL_ROW_UPDATED	Row를 성공적으로 가져왔으며 결과 집합에서 마지막으로 가지고 온 후에 갱신되었다. Row를 다시 가지고 오거나 SQLSetPos에 의해 다시 수행된 경우, 상태는 새로운 row의 상태로 갱신된다.
SQL_ROW_DELETED	결과 집합에서 row를 마지막으로 갖고 온 후에 삭제되었다.
SQL_ROW_NOROW	Row 집합이 결과 집합의 끝과 겹쳤고 row가 없음을 반환한다.

Row 반입 버퍼

Row 반입 버퍼는 가져온 row의 개수를 반환하는데 사용된다. 데이터를 가져올 때 에러가 발생하여 데이터가 없는 row도 포함한다. 즉, row 상태 배열에서 SQL_ROW_NOROW 값이 아닌 row의 개수이다. 이 버퍼의 주소는 SQL_ATTR_ROWS_FETCHED_PTR 명령문 속성으로 지정한다. 응용 프로그램이 이 버퍼를 할당하며, SQLFetch, SQLFetchScroll이 이 버퍼를 설정한다. SQL_ATTR_ROWS_FETCHED_PTR 명령문 속성값이 null 포인터인 경우, 이 함수는 가져온 row 개수를 반환하지 않는다. 결과 집합의 현재 row 개수를 확인하기 위해 응용 프로그램이 SQL_ATTR_ROW_NUMBER 속성으로 SQLGetStmtAttr을 호출할 수 있다.

SQLFetch, SQLFetchScroll에서 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, row 반입 버퍼의 내용은 정의되지 않는다. SQL_NO_DATA를 반환할 경우, row 반입 버퍼의 값은 0으로 설정된다.

에러 처리

에러와 경고는 개별 row 또는 함수 전체에 적용할 수 있다.

함수 전체에 대한 에러와 경고

SQLSTATE HYT00 (timeout expired)나 SQLSTATE 24000 (invalid cursor state)처럼 전체 함수에 적용되는 에러의 경우, SQLFetch가 SQL_ERROR와 해당 SQLSTATE를 반환한다. Row 버퍼의 내용은 정의되지 않고, 커서의 위치는 변경되지 않는다.

전체 함수에 적용되는 경고의 경우, SQLFetch가 SQL_SUCCESS_WITH_INFO와 해당 SQLSTATE를 반환한다. 전체 함수에 적용되는 경고는 개별 row에 적용되는 상태를 기록하기 전에 반환된다.

개별 row에 대한 에러와 경고

SQLSTATE 22012 (division by zero)와 같은 에러나 SQLSTATE 01004 (data truncated)와 같은 경고는 개별 row에 적용된다.

SQLFetch는 다음을 수행한다.

- 에러에 대한 SQL_ROW_ERROR 또는 경고에 대한 SQL_ROW_SUCCESS_WITH_INFO를 row 상태 배열의 해당 요소에 설정한다.
- 에러 또는 경고에 대한 SQLSTATE가 포함된 레코드를 하나 이상 추가한다.
- 상태 레코드에 row와 column 번호 필드를 설정한다. SQLFetch가 row 또는 column의 번호를 확인할 수 없는 경우, 번호를 각각 SQL_ROW_NUMBER_UNKNOWN 또는 SQL_COLUMN_NUMBER_UNKNOWN으로 설정한다.

정한다. 상태 레코드가 특정 column에 적용되지 않을 경우, SQLFetch는 column 번호를 SQL_NO_COLUMN_NUMBER로 설정한다.

SQLFetch는 row 집합의 모든 row (상태가 SQL_ROW_NOROW인 row는 포함하지 않음)에 에러가 발생했을 경우 SQL_ERROR를 반환하며, 일부 row에 에러가 발생했을 경우 SQL_SUCCESS_WITH_INFO를 반환한다. Row 집합의 크기가 1인 경우, row에 에러가 발생하면 SQLFetch는 SQL_ERROR를 반환한다.

설명자와 SQLFetch

SQLFetch는 다음 설명자 필드를 사용한다.

설명자 필드	설명자	필드의 위치	설정
SQL_DESC_ARRAY_SIZE	ARD	헤더	SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성
SQL_DESC_ARRAY_STATUS_PTR	IRD	헤더	SQL_ATTR_ROW_STATUS_PTR 명령문 속성
SQL_DESC_BIND_OFFSET_PTR	ARD	헤더	SQL_ATTR_ROW_BIND_OFFSET_PTR 명령문 속성
SQL_DESC_BIND_TYPE	ARD	헤더	SQL_ATTR_ROW_BIND_TYPE 명령문 속성
SQL_DESC_COUNT	ARD	헤더	SQLBindCol의 ColumnNumber 인자
SQL_DESC_DATA_PTR	ARD	레코드	SQLBindCol의 TargetValuePtr 인자
SQL_DESC_INDICATOR_PTR	ARD	레코드	SQLBindCol의 StrLen_or_IndPtr 인자
SQL_DESC_OCTET_LENGTH	ARD	레코드	SQLBindCol의 BufferLength 인자
SQL_DESC_OCTET_LENGTH_PTR	ARD	레코드	SQLBindCol의 StrLen_or_IndPtr 인자
SQL_DESC_ROWS_PROCESSED_PTR	IRD	레코드	SQL_ATTR_ROWS_FETCHED_PTR 명령문 속성
SQL_DESC_TYPE	ARD	레코드	SQLBindCol의 TargetType 인자

모든 설명자 필드는 SQLSetDescField를 통해 설정할 수 있다.

길이와 지시자 버퍼의 분리

응용 프로그램은 길이와 지시자 값을 저장하는데 단일 버퍼 또는 두 개의 버퍼를 바인딩 할 수 있다. 응용 프로그램이 SQLBindCol을 호출할 경우, StrLen_or_IndPtr 인자로 전달된 동일한 주소에, ARD의 SQL_DESC_OCTET_LENGTH_PTR과 SQL_DESC_INDICATOR_PTR 필드를 설정한다. 응용 프로그램은 SQLSetDescField 또는 SQLSetDescRec를 호출하여 두 필드의 주소를 다르게 설정할 수 있다.

SQLFetch는 응용 프로그램이 별도의 길이와 지시자 버퍼를 지정했는지 판단한다. 데이터가 NULL이 아닌 경우, SQLFetch는 지시자 버퍼를 0으로 설정하고, 길이 버퍼에 길이를 반환한다. 데이터가 NULL인 경우, SQLFetch는 지시자 버퍼를 SQL_NULL_DATA로 설정하고, 길이 버퍼는 수정하지 않는다.

SQLFetchScroll

적합성

도입된 버전: ODBC 3.0
표준 준수: ISO 92

개요

SQLFetchScroll은 결과 집합에서 지정된 데이터 집합을 가져와서 바인딩 된 모든 column에 반환한다.

구문

```
SQLRETURN SQLFetchScroll(  
    SQLHSTMT      StatementHandle,  
    SQLSMALLINT    FetchOrientation,  
    SQLLEN         FetchOffset);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

FetchOrientation

[입력] Fetch 타입: SQL_FETCH_NEXT, SQL_FETCH_PRIOR, SQL_FETCH_FIRST, SQL_FETCH_LAST, SQL_FETCH_ABSOLUTE, SQL_FETCH_RELATIVE, SQL_FETCH_BOOKMARK

FetchOffset

[입력] 가져올 row의 개수이다. 이 인자의 해석은 FetchOrientation의 인자값에 따라 달라진다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	Column에 대해 반환된 문자열이나 이진 데이터가 잘렸다. 문자열이 잘린 경우, 오른쪽이 잘렸다.

SQLSTATE	Error	설명
01S07	Fractional truncation	Column에 대해 반환된 데이터가 잘렸다. 숫자 데이터 타입이 잘린 경우, 소수 부분이 잘렸다. 기간 구성 요소를 포함하는 time, timestamp, interval 데이터 타입이 잘린 경우, 시간의 소수 부분이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	결과 집합에 있는 column 데이터 값을 SQLBindCol의 TargetType이 지정한 데이터 타입으로 변환할 수 없다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22002	Indicator variable required but not supplied	SQLBindCol의 StrLen_or_IndPtr (또는 SQLSetDescField, SQLSetDescRec로 설정된 SQL_DESC_INDICATOR_PTR)이 null 포인터인 column에 NULL 데이터를 가져왔다.
22003	Numeric value out of range	하나 이상의 column에서 반환한 숫자의 (소수가 아닌) 정수 부분이 잘렸다.
22007	Invalid datetime format	결과 집합의 문자열이 유효한 date, time, timestamp 형식이 아니다.
22012	Division by zero	0으로 나눈 산술식의 결과값이 반환되었다.
22015	Interval field overflow	Exact numeric 또는 interval SQL 데이터 타입을 interval C 타입에 지정할 때, 선행 필드의 유효 숫자가 손실되었다. SQL 타입의 값을 interval C 타입에 표현할 수 없다.
22018	Invalid character value for cast specification	C 버퍼의 character set으로 표현할 수 없는 문자가 결과 집합의 문자 column에 포함되었다. C 타입은 exact 또는 approximate numeric, datetime, interval 데이터 타입이고, SQL 타입은 문자 데이터 형식일 때, C 타입에 바인딩 되는 column의 값이 유효한 문자가 아니다.
24000	Invalid cursor state	StatementHandle이 실행된 상태이지만, StatementHandle과 관련된 결과 집합이 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	지정된 StatementHandle이 실행 상태가 아니다. 이 함수는 SQLExecDirect, SQLExecute, 카탈로그 함수를 호출하지 않고 호출되었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY106	Fetch type out of range	FetchOrientation 인자에 지정된 값이 유효하지 않다. SQL_ATTR_CURSOR_TYPE 명령문 속성값이 SQL_CURSOR_FORWARD_ONLY이며, FetchOrientation 인자값이 SQL_FETCH_NEXT가 아니다. SQL_ATTR_CURSOR_SCROLLABLE 명령문 속성값이 SQL_NONSCROLLABLE이

SQLSTATE	Error	설명
		며, FetchOrientation 인자값이 SQL_FETCH_NEXT가 아니다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한 시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

SQLFetchScroll은 결과 집합에서 특정 row 집합을 반환한다. Row 집합은 절대 또는 상대 위치, 또는 북마크 위치로 지정할 수 있다. SQLFetchScroll은 결과 집합이 생성되고 커서를 닫기 전까지, 결과 집합이 존재하는 동안만 호출할 수 있다. Column이 바인딩 되어 있을 경우, column에 데이터를 반환한다. 응용 프로그램이 row 상태 배열에 대한 포인터 또는 가져온 row의 수를 반환하는 버퍼를 지정할 경우, SQLFetchScroll은 이 정보도 반환한다. SQLFetch와 SQLFetchScroll은 섞어서 호출할 수 있다.

커서의 위치

결과 집합이 생성되면, 커서는 결과 집합의 시작 전에 위치한다. SQLFetchScroll은 다음 표와 같이 FetchOrientation과 FetchOffset 인자에 따라 블록 커서를 위치시킨다. 새로운 row 집합의 시작을 결정하는 규칙은 다음과 같다.

FetchOrientation	의미
SQL_FETCH_NEXT	다음 row 집합을 반환하며 SQLFetch 호출과 동일하다. SQLFetchScroll은 FetchOffset 값을 무시한다.
SQL_FETCH_PRIOR	이전 row 집합을 반환한다. SQLFetchScroll은 FetchOffset 값을 무시한다.
SQL_FETCH_RELATIVE	현재 row 집합의 시작 부분에 FetchOffset의 row 집합을 반환한다.
SQL_FETCH_ABSOLUTE	FetchOffset에서 시작하는 row 집합을 반환한다.
SQL_FETCH_FIRST	결과 집합의 첫 번째 row 집합을 반환한다. SQLFetchScroll은 FetchOffset 값을 무시한다.
SQL_FETCH_LAST	결과 집합의 마지막 전체 row 집합을 반환한다. SQLFetchScroll은 FetchOffset 값을 무시한다.
SQL_FETCH_BOOKMARK	SQL_ATTR_FETCH_BOOKMARK_PTR 명령문 속성이 지정한 북마크에서 FetchOffset의 row 집합을 반환한다.

SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성은 row 집합의 row 개수를 지정한다. SQLFetchScroll로 가져오는 row 집합이 결과 집합의 끝과 겹치면, SQLFetchScroll은 부분 row 집합을 반환한다. 이는 S를 가져오는 row 집합의 시작 row, R을 row 집합의 크기, L을 결과 집합의 마지막 row라고 할 때, $S + R - 1$ 이 L보다 큰 경우이며, row 집합의 처음 $L - S + 1$ 개의 row만 유효하다. 나머지 row는 모두 비어 있고, SQL_ROW_NOROW 상태가 된다.

SQLFetchScroll을 반환한 후의 현재 row는 row 집합의 첫 번째 row이다.

커서 위치 규칙

다음 섹션에서는 FetchOrientation 각각에 대한 정확한 규칙을 설명한다. 이러한 규칙은 다음과 같은 표기법을 사용한다.

표기법	의미
Before start	블록 커서는 결과 집합의 시작 전에 위치한다. 새로운 row 집합의 첫 번째 row가 결과 집합의 시작 전이면, SQLFetchScroll은 SQL_NO_DATA를 반환한다.
After end	블록 커서는 결과 집합의 끝 이후에 위치한다. 새로운 row 집합의 첫 번째 row가 결과 집합의 끝 이후이면, SQLFetchScroll은 SQL_NO_DATA를 반환한다.
CurrRowsetStart	현재 row 집합의 첫 번째 row 번호이다.
LastResultRow	결과 집합의 마지막 row 번호이다.
RowsetSize	Row 집합의 크기이다.
FetchOffset	FetchOffset 인자의 값이다.
BookmarkRow	SQL_ATTR_FETCH_BOOKMARK_PTR 명령문 속성이 지정한 북마크에 해당하는 row이다.

SQL_FETCH_NEXT

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
Before start	1
$\text{CurrRowsetStart} + \text{RowsetSize}[1] \leq \text{LastResultRow}$	$\text{CurrRowsetStart} + \text{RowsetSize}[1]$
$\text{CurrRowsetStart} + \text{RowsetSize}[1] > \text{LastResultRow}$	After end
After end	After end

[1] Fetch 하는 도중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 이 fetch 이전에 사용된 row 집합의 크기이다.

SQL_FETCH_PRIOR

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
Before start	Before start
$\text{CurrRowsetStart} = 1$	Before start
$1 < \text{CurrRowsetStart} \leq \text{RowsetSize}[1]$	1
$\text{CurrRowsetStart} > \text{RowsetSize}[1]$	$\text{CurrRowsetStart} - \text{RowsetSize}[1]$
After end AND $\text{LastResultRow} < \text{RowsetSize}[1]$	1
After end AND $\text{LastResultRow} \geq \text{RowsetSize}[1]$	$\text{LastResultRow} - \text{RowsetSize} + 1[1]$

[1] Fetch 하는 중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 새로운 fetch에 사용된 row 집합의 크기이다.

SQL_FETCH_RELATIVE

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
(Before start AND FetchOffset > 0) OR (After end AND FetchOffset < 0)	__ [1]
BeforeStart AND FetchOffset <= 0	Before start
CurrRowsetStart = 1 AND FetchOffset < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND Fetch Offset > RowsetSize[2]	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND Fetch Offset <= RowsetSize[2]	1
1 <= CurrRowsetStart + FetchOffset <= LastResultRow	CurrRowsetStart + FetchOffset
CurrRowsetStart + FetchOffset > LastResultRow	After end
After end AND FetchOffset >= 0	After end

[1] SQLFetchScroll은 FetchOrientation을 SQL_FETCH_ABSOLUTE로 설정하여 호출한 경우와 동일한 row 집합을 반환한다.

[2] Fetch 하는 중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 새로운 fetch에 사용된 row 집합의 크기이다.

SQL_FETCH_ABSOLUTE

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
FetchOffset < 0 AND FetchOffset <= LastResultRow	LastResultRow + FetchOffset + 1
FetchOffset < 0 AND FetchOffset > LastResultRow AND FetchOffset > RowsetSize[1]	Before start
FetchOffset < 0 AND FetchOffset > LastResultRow AND FetchOffset <= RowsetSize[1]	1
FetchOffset = 0	Before start
1 <= FetchOffset <= LastResultRow	FetchOffset
FetchOffset > LastResultRow	After end

[1] Fetch 하는 중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 새로운 fetch에 사용된 row 집합의 크기이다.

SQL_FETCH_FIRST

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
Any	1

SQL_FETCH_LAST

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
RowsetSize[1] ≤ LastResultRow	LastResultRow - RowsetSize + 1[1]
RowsetSize[1] > LastResultRow	1

[1] Fetch 하는 중에 row 집합 크기가 변경되는 경우, row 집합의 크기는 새로운 fetch에 사용된 row 집합의 크기이다.

SQL_FETCH_BOOKMARK

다음과 같은 규칙이 적용된다.

상태	새로운 row 집합의 첫 번째 row
BookmarkRow + FetchOffset < 1	Before start
1 ≤ BookmarkRow + FetchOffset ≤ LastResultRow	BookmarkRow + FetchOffset
BookmarkRow + FetchOffset > LastResultRow	After end

버퍼 주소

SQLFetchScroll은 데이터의 주소와 길이/ 지시자 버퍼의 주소를 SQLFetch와 동일하게 결정한다. 자세한 내용은 SQLBindCol의 **버퍼 주소**를 참조한다.

Row 상태 배열

SQLFetchScroll은 row 상태 배열을 SQLFetch와 동일하게 설정한다. 자세한 내용은 SQLFetch의 **Row 상태 배열**을 참조한다.

Row 반입 버퍼

SQLFetchScroll은 가져온 row의 개수를 SQLFetch와 동일하게 반환한다. 자세한 내용은 SQLFetch의 **Row 반입 버퍼**를 참조한다.

에러 처리

SQLFetchScroll은 에러와 경고를 SQLFetch와 동일하게 반환한다. 자세한 내용은 SQLFetch의 **에러 처리**를 참조한다.

SQLForeignKeys

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLForeignKeys는 다음을 반환한다.

- 지정된 테이블 (다른 테이블의 기본 키를 참조하도록 지정된 테이블의 column)의 외래 키 목록
- 지정된 테이블의 기본 키를 참조하는 다른 테이블의 외래 키 목록

드라이버는 각 목록을 결과 집합으로 반환한다.

구문

```
SQLRETURN SQLForeignKeys(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     PKCatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     PKSchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     PKTableName,
    SQLSMALLINT   NameLength3,
    SQLCHAR *     FKCatalogName,
    SQLSMALLINT   NameLength4,
    SQLCHAR *     FKSchemaName,
    SQLSMALLINT   NameLength5,
    SQLCHAR *     FKTableName,
    SQLSMALLINT   NameLength6);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

PKCatalogName

[입력] 기본 키 테이블 카탈로그 이름이다. PKCatalogName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, PKCatalogName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, PKCatalogName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength1

[입력] *PKCatalogName의 길이이다.

PKSchemaName

[입력] 기본 키 테이블 스키마 이름이다. PKSchemaName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, PKSchemaName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, PKSchemaName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength2

[입력] *PKSchemaName의 길이이다.

PKTableName

[입력] 기본 키 테이블 이름이다. PKTableName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, PKTableName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, PKTableName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength3

[입력] *PKTableName의 길이이다.

FKCatalogName

[입력] 외래 키 테이블 카탈로그 이름이다. FKCatalogName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, FKCatalogName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, FKCatalogName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength4

[입력] *FKCatalogName의 길이이다.

FKSchemaName

[입력] 외래 키 테이블 스키마 이름이다. FKSchemaName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, FKSchemaName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, FKSchemaName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength5

[입력] *FKSchemaName의 길이이다.

FKTableName

[입력] 외래 키 테이블 이름이다. FKTableName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되어 있는 경우, FKTableName은 대소문자를 구분하지 않는 식별자로 취급된다. SQL_FALSE일 경우, FKTableName은 대소문자를 구분하며 문자 그대로 처리되는 일반 인자이다.

NameLength6

[입력] *FKTableName의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 호출해 열린 커서가 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	PKTableName과 FKTableName 인자가 null 포인터이다. SQL_ATTR_METADATA_ID 명령문 속성값이 SQL_TRUE이며, PKSchemaName, FKSchemaName, PKTableame 또는 FKTableame 인자가 null 포인터이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	이름 길이 인자 중 하나의 값이 0보다 작지만 SQL_NTS가 아니다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한 시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

설명

*PKTableName이 테이블 이름을 포함하는 경우, SQLForeignKeys는 지정된 테이블의 기본 키와 이를 참조하는 모든 외래 키가 포함된 결과 집합을 반환한다. 다른 테이블의 외래 키 목록은 지정된 테이블의 unique 제약 조건을 가리키는 외래 키를 포함하지 않는다.

*FKTableName이 테이블 이름을 포함하는 경우, SQLForeignKeys는 다른 테이블의 기본 키를 가리키는 지정된 테이블의 외래 키가 포함된 결과 집합과, 그들이 참조하는 다른 테이블의 기본 키를 반환한다.

*PKTableName과 *FKTableName 모두 테이블 이름을 포함하는 경우, SQLForeignKeys는 *PKTableName에 지정된 테이블의 기본 키를 참조하는 *FKTableName에 지정된 테이블의 외래 키를 반환한다. 이 키는 하나여야 한다.

SQLForeignKeys는 표준 결과 집합을 반환한다. 기본 키와 관련된 외래 키를 요청하는 경우, 결과 집합은 FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, KEY_SEQ로 정렬된다. 외래 키와 관련된 기본 키를 요청하는 경우, PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_NAME, KEY_SEQ로 정렬된다.

다음 표는 결과 집합의 column이다.

Column 이름	Column 번호	데이터 타입	설명
PKTABLE_CAT (ODBC 1.0)	1	VARCHAR	기본 키 테이블의 카탈로그 이름이다.
PKTABLE_SCHEM (ODBC 1.0)	2	VARCHAR	기본 키 테이블의 스키마 이름이다.
PKTABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	기본 키 테이블의 이름이다.
PKCOLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	기본 키 column 이름이다. 이름이 없는 column에 대해서는 빈 문자열을 반환한다.
FKTABLE_CAT (ODBC 1.0)	5	VARCHAR	외래 키 테이블의 카탈로그 이름이다.
FKTABLE_SCHEM (ODBC 1.0)	6	VARCHAR	외래 키 테이블의 스키마 이름이다.
FKTABLE_NAME (ODBC 1.0)	7	VARCHAR not NULL	외래 키 테이블의 이름이다.
FKCOLUMN_NAME (ODBC 1.0)	8	VARCHAR not NULL	외래 키 column 이름이다. 이름이 없는 column에 대해서는 빈 문자열을 반환한다.
KEY_SEQ (ODBC 1.0)	9	SMALLINT not NULL	1부터 시작하는 키의 column 순차 번호이다.
UPDATE_RULE			SQL 연산이 UPDATE일 때, 외래 키에 적용하는 동작이다. (참조된 테이블은 기본 키가 있는 테이블, 참조 테이블은 외래 키가 있는 테이블이다.) - SQL_CASCADE: 참조되는 테이블의 기본 키가 변경되면, 참조 테이블의 외래 키도 변경된다. - SQL_NO_ACTION: 참조되는 테이블의 기본 키를 업데이트 할 때 참조하는 테이블의 row가 참조된 테이블에 대응하지 않는 경우가 발생하면 업

Column 이름	Column 번호	데이터 타입	설명
LE (ODBC 1.0)	10	SMALLINT	데이트가 거부된다. 참조하는 테이블의 외래 키 업데이트가 참조 테이블의 기본 키 값으로 존재하지 않을 경우, 업데이트가 거부된다. - SQL_SET_NULL: 참조 테이블에서 하나 이상 row의 기본 키 중 하나 이상의 구성 요소를 변경하여 업데이트 할 경우, 기본 키의 변경된 구성 요소에 대응하는 참조 테이블에서 외래 키의 구성 요소로 참조하는 테이블과 일치하는 모든 row를 NULL로 설정한다. - SQL_SET_DEFAULT: 참조 테이블에서 하나 이상 row의 기본 키 중 하나 이상의 구성 요소를 변경하여 업데이트 할 경우, 기본 키의 변경된 구성 요소에 대응하는 참조 테이블에서 외래 키의 구성 요소로 참조하는 테이블의 일치하는 모든 row를 기본값으로 설정한다.
DELETE_RULE (ODBC 1.0)	11	SMALLINT	SQL 연산이 DELETE일 때, 외래 키에 적용하는 동작이다. (참조된 테이블은 기본 키가 있는 테이블, 참조 테이블은 외래 키가 있는 테이블이다.) - SQL_CASCADE: 참조되는 테이블의 기본 키가 삭제되면, 참조 테이블의 외래 키도 삭제된다. - SQL_NO_ACTION: 참조되는 테이블의 기본 키를 삭제할 때 참조하는 테이블의 row가 참조된 테이블에 대응하지 않으면 업데이트가 거부된다. - SQL_SET_NULL: 참조되는 테이블에서 하나 이상의 row가 삭제되면 참조 테이블 외부 키의 각 구성 요소가 참조하는 테이블의 일치하는 모든 row를 NULL로 설정한다. - SQL_SET_DEFAULT: 참조되는 테이블에서 하나 이상의 row가 삭제되면 참조 테이블 외부 키의 각 구성 요소가 참조하는 테이블의 일치하는 모든 row를 기본값으로 설정한다.
FK_NAME (ODBC 2.0)	12	VARCHAR	외래 키 이름이다.
PK_NAME (ODBC 2.0)	13	VARCHAR	기본 키 이름이다.
DEFERRABILITY (ODBC 3.0)	14	SMALLINT	SQL_INITIALLY_DEFERRED, SQL_INITIALLY_IMMEDIATE, SQL_NOT_DEFERRABLE

SQLFreeConnect

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLFreeConnect 함수는 ODBC 3.x에서 SQLFreeHandle 함수로 대체되었다.
자세한 내용은 **SQLFreeHandle** 함수를 참조한다.

SQLFreeEnv

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLFreeEnv 함수는 ODBC 3.x에서 SQLFreeHandle 함수로 대체되었다.
자세한 내용은 **SQLFreeHandle** 함수를 참조한다.

SQLFreeHandle

적합성

도입된 버전: ODBC 3.0
표준 준수: ISO 92

개요

SQLFreeHandle은 지정된 환경, 연결, 명령문, 설명자 핸들과 관련된 자원을 해제한다.

구문

```
SQLRETURN SQLFreeHandle(  
    SQLSMALLINT    HandleType,  
    SQLHANDLE       Handle);
```

인자

HandleType

[입력] SQLFreeHandle로 해제될 핸들 타입이다. SQL_HANDLE_DBC, SQL_HANDLE_DESC, SQL_HANDLE_ENV, SQL_HANDLE_STMT 중 하나여야 한다.
HandleType이 이 값들 중 하나가 아닐 경우, SQLFreeHandle은 SQL_INVALID_HANDLE을 반환한다.

Handle

[입력] 해제될 핸들이다.

반환

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE
SQLFreeHandle이 SQL_ERROR를 반환할 경우, 핸들은 여전히 유효하다.

진단

SQLSTATE	Error	설명
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
		HandleType 인자가 SQL_HANDLE_ENV이고, 적어도 하나의 연결이 할당되거나

SQLSTATE	Error	설명
HY010	Function sequence error	접속 상태에 있었다. HandleType 인자가 SQL_HANDLE_ENV인 SQLFreeHandle을 호출하기 전에, SQLDisconnect와 HandleType 인자가 SQL_HANDLE_DBC인 SQLFreeHandle을 호출해야 한다. HandleType 인자가 SQL_HANDLE_DBC이고, 함수가 SQLDisconnect를 호출하기 전에 호출되었다. HandleType 인자가 SQL_HANDLE_STMT이고, SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.

설명

SQLFreeHandle은 환경, 연결, 명령문, 설명자 핸들을 해제하는데 사용된다.

핸들이 해제된 후, 응용 프로그램은 해제된 핸들을 사용해서는 안된다.

환경 핸들 해제

HandleType이 SQL_HANDLE_ENV인 SQLFreeHandle을 호출하기 전에, 응용 프로그램이 환경으로부터 할당된 모든 연결에 대해, HandleType이 SQL_HANDLE_DBC인 SQLFreeHandle을 호출해야 한다. 그렇지 않으면, SQLFreeHandle은 SQL_ERROR를 반환하고 환경과 활성화된 연결은 유효하게 남아 있다.

연결 핸들 해제

HandleType이 SQL_HANDLE_DBC인 SQLFreeHandle을 호출하기 전에, 핸들에 연결이 있는 경우, 응용 프로그램은 SQLDisconnect를 호출해야 한다. 그렇지 않으면, SQLFreeHandle은 SQL_ERROR를 반환하고 연결은 유효하게 남아 있다.

명령문 핸들 해제

HandleType이 SQL_HANDLE_STMT인 SQLFreeHandle은 HandleType이 SQL_HANDLE_STMT인 SQLAllocHandle을 호출하여 할당된 모든 자원을 해제한다. 응용 프로그램이 결과가 남아 있는 명령문으로 SQLFreeHandle을 호출하면 남아 있는 결과는 삭제된다. 응용 프로그램이 명령문 핸들을 해제할 때, 드라이버는 명령문과 관련된 네 개의 자동 할당 설명자도 해제한다.

SQLDisconnect는 연결에 대해 열려있는 모든 명령문과 설명자를 자동으로 삭제한다.

SQLFreeStmt

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLFreeStmt는 지정된 명령문과 관련된 처리를 중단하고, 열린 커서를 닫고, 남아 있는 결과를 버리거나 선택적으로 명령문 핸들과 관련된 모든 자원을 해제한다.

구문

```
SQLRETURN SQLFreeStmt(  
    SQLHSTMT      StatementHandle,  
    SQLUSMALLINT  Option);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

Option

[입력] 다음 옵션 중 하나이다.

- SQL_CLOSE: StatementHandle과 관련된 커서를 닫고, 남아 있는 결과를 삭제한다. 응용 프로그램은 같거나 다른 매개 변수 값으로 다시 SELECT 문을 실행하여 커서를 나중에 다시 열 수 있다. 커서가 열려 있지 않은 경우, 이 옵션은 응용 프로그램에 영향을 미치지 않는다. SQLCloseCursor 역시 커서를 닫는다.
- SQL_DROP: 이 옵션을 더 이상 사용하지 않는다.
- SQL_UNBIND: 지정된 StatementHandle에 대해 SQLBindCol로 바인딩 된 모든 column 버퍼를 해제하고, ARD의 SQL_DESC_COUNT 필드를 0으로 설정한다.
- SQL_RESET_PARAMS: 지정된 StatementHandle에 대해 SQLBindParameter로 설정한 모든 매개 변수 버퍼를 해제하고, APD의 SQL_DESC_COUNT 필드를 0으로 설정한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY092	Option type out of range	Option 인자의 값이 SQL_CLOSE, SQL_DROP, SQL_UNBIND, SQL_RESET_PARAMS가 아니다.

설명

SQL_CLOSE 옵션으로 SQLFreeStmt를 호출하는 것은 SQLCloseCursor를 호출하는 것과 같다. 하지만 열린 커서가 없을 때 SQL_CLOSE 옵션으로 SQLFreeStmt를 호출하면 응용 프로그램에 아무런 영향을 미치지 않는 반면에 SQLCloseCursor는 SQLSTATE 24000 (invalid cursor state)을 반환한다.

SQLGetConnectAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetConnectAttr은 연결 속성의 현재 설정을 반환한다.

구문

```
SQLRETURN SQLGetConnectAttr(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

Attribute

[입력] 검색할 속성이다.

ValuePtr

[출력] Attribute로 지정된 속성의 현재 설정 값을 반환할 메모리의 포인터이다.

ValuePtr이 null일 경우, StringLengthPtr은 반환 가능한 전체 바이트 수 (null 종료 문자 제외)를 반환한다.

BufferLength

[입력] Attribute가 ODBC에 정의된 필드이고, ValuePtr이 문자열이나 이진 버퍼를 가리킬 경우, 이 인자는 * ValuePtr의 길이여야 한다. Attribute가 ODBC에 정의된 필드이고, *ValuePtr이 정수일 경우, 이 인자는 무시된다.

StringLengthPtr

[출력] *ValuePtr에서 반환 가능한 총 바이트 수 (문자 데이터에 대한 null 종료 바이트 제외)를 반환하는 포인터이다.

문자 데이터의 경우, 반환할 수 있는 바이트의 수가 BufferLength 보다 크거나 같은 경우, *ValuePtr은 Buff

erLength에서 1을 뺀 만큼의 길이로 잘리고, 드라이버가 null 종료한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	BufferLength에서 null 종료 문자 크기만큼 잘린 데이터가 *ValuePtr에 반환되었다. 잘리지 않는 문자열의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08003	Connection not open	연결 상태에서 요구할 수 있는 attribute 값이 지정되었다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY090	Invalid string or buffer length	*ValuePtr이 문자열이고 BufferLength가 0보다 작지만 SQL_NTS가 아니다.
HY092	Invalid attribute/option identifier	Attribute 인자값이 유효하지 않다.
HYC00	Optional feature not implemented	Attribute 인자값이 유효하지만 드라이버가 지원하지 않는다.

설명

Attribute가 문자열을 반환하는 속성을 지정하는 경우, ValuePtr에는 문자열 버퍼에 대한 포인터가 있어야 한다. null 종료 문자를 포함하여 반환되는 문자열의 최대 길이는 BufferLength 바이트이다.

Attribute	ValuePtr 내용
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	SQLINTEGER 값이다. SQL_MODE_READ_ONLY는 갱신을 요구하지 않는 연결을 나타내는 지시자로 사용된다. 이 모드는 드라이버 또는 데이터 소스에 대한 적절한 잠금 계획, 트랜잭션 관리, 최적화에 사용될 수 있다. 기본값은 SQL_MODE_READ_WRITE이다.
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	자동 commit 또는 수동 commit 사용 여부를 지정하는 SQLINTEGER 값이다. SQL_AUTOCOMMIT_ON: 이 값이 기본값이다. 드라이버는 자동 commit 모드를 사용한다. 각 명령문 수행 후에 바로 commit 된다. 수동 commit 모드에서 자동 commit 모드로 변경하기 위해 SQL_ATTR_AUTOCOMMIT를 SQL_AUTOCOMMIT_ON으로 설정할 경우, 연결에 대해 열린 트랜잭션은 commit 된다.

Attribute	ValuePtr 내용
	SQL_AUTOCOMMIT_OFF: 드라이버는 수동 commit 모드를 사용하고, 응용 프로그램은 명시적으로 SQLEndTrans으로 commit 하거나 rollback 해야 한다.
SQL_ATTR_CHARACTER_SET	드라이버의 character set 문자열이다.
SQL_ATTR_DATABASE_CHARACTER_SET	데이터 소스의 character set 문자열이다.
SQL_ATTR_DATE_FORMAT	드라이버의 DATE 형식 문자열이다.
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	SQLINTEGER 값으로 로그인 요청을 기다리는 시간 (초)이다. ValuePtr이 0일 경우, 타임아웃은 사용되지 않으며 연결 시도는 무기한 대기한다.
SQL_ATTR_METADATA_ID (ODBC 3.0)	카탈로그 함수의 문자열 인자를 결정하는 SQLINTEGER 값이다. SQL_TRUE이면 카탈로그 함수의 문자열 인자는 식별자로 취급되며, 대소문자를 구분하지 않는다. 구분 기호로 구분되지 않는 문자열의 경우, 드라이버는 모든 앞뒤 공백을 제거하고, 문자열은 대문자로 구성된다. 구분 기호로 구분된 문자열의 경우, 앞뒤 공백을 제거하고, 구분 문자 사이의 문자 그대로 유지한다. 이런 인자 중 하나가 null 포인터로 설정되어 있는 경우, 함수는 SQL_ERROR와 SQLSTATE HY009 (invalid use of null pointer)를 반환한다. SQL_FALSE이면 카탈로그 함수의 문자열 인자는 식별자로 취급되지 않는다. 문자열 인자에 따라 문자열 검색 패턴을 포함할 수도 있고 포함하지 않을 수도 있다. 기본값은 SQL_FALSE이다. SQL_ATTR_METADATA_ID는 명령문 level에서도 설정할 수 있다.
SQL_ATTR_TIMESTAMP_FORMAT	드라이버의 TIMESTAMP 형식 문자열이다.
SQL_ATTR_TIMESTAMP_WITH_TIMEZONE_FORMAT	드라이버의 TIMESTAMP WITH TIME ZONE 형식 문자열이다.
SQL_ATTR_TIMEZONE	드라이버의 시간대 문자열이다.
SQL_ATTR_TIME_FORMAT	드라이버의 TIME 형식 문자열이다.
SQL_ATTR_TIME_WITH_TIMEZONE_FORMAT	드라이버의 TIME WITH TIME ZONE 형식 문자열이다.
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	현재 연결에 대한 isolation level을 설정한 32 비트 마스크이다.

SQLGetConnectOption

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLGetConnectOption 함수는 ODBC 3.x에서 SQLGetConnectAttr 함수로 대체되었다.
자세한 내용은 [SQLGetConnectAttr](#) 함수를 참조한다.

SQLGetCursorName

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLGetCursorName은 지정된 명령문과 관련된 커서 이름을 반환한다.

구문

```
SQLRETURN SQLGetCursorName(  
    SQLHSTMT      StatementHandle,  
    SQLCHAR *      CursorName,  
    SQLSMALLINT    BufferLength,  
    SQLSMALLINT *   NameLengthPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CursorName

[출력] 커서 이름을 반환할 버퍼 포인터이다.

CursorName이 NULL일 경우, NameLengthPtr은 반환 가능한 바이트 길이 (null 종료 문자 제외)를 반환한다.

BufferLength

[입력] *CursorName의 길이이다.

NameLengthPtr

[출력] *CursorName에 반환 가능한 총 바이트 길이 (null 종료 문자 제외)를 위한 메모리 포인터이다. 반환할 수 있는 바이트 수보다 크거나 같은 경우, *sCursorName은 BufferLength에서 1을 뺀 만큼의 길이로 잘린다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	*CursorName 버퍼가 전체 커서 이름을 반환하기에 충분히 크지 않아서 커서 이름이 잘렸다. 잘리지 않는 커서 이름의 길이는 *NameLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	BufferLength 인자의 값이 0보다 작다.

설명

커서 이름은 positioned update와 positioned delete 구문 (예: UPDATE table-name ...WHERE CURRENT OF cursor-name)에서만 사용된다. 응용 프로그램에서 SQLSetCursorName으로 커서 이름을 설정하지 않은 경우, 드라이버는 SQL_CUR로 시작하는 커서 이름을 생성한다.

SQLGetCursorName은 명령문이 할당되거나 준비 상태에 있을 경우, 커서 이름을 재설정 할 수 있다.

명시적 또는 암시적으로 설정된 커서 이름은 관련된 명령문이 HandleType이 SQL_HANDLE_STMT인 SQLFreeHandle을 호출해 삭제할 때까지 유효하다.

SQLGetData

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLGetData는 결과 집합에서 하나의 column 데이터를 검색하며 가변 길이 데이터를 검색하기 위해 여러 번 호출 될 수 있다.

구문

```
SQLRETURN SQLGetData(  
    SQLHSTMT      StatementHandle,  
    SQLUSMALLINT   Col_or_Param_Num,  
    SQLSMALLINT    TargetType,  
    SQLPOINTER     TargetValuePtr,  
    SQLLEN          BufferLength,  
    SQLLEN *        StrLen_or_IndPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

Col_or_Param_Num

[입력] 반환되는 데이터에서 column 데이터를 검색하기 위한 column의 번호이다. 결과 집합 column은 1부터 시작해 증가하는 순서로 번호가 매겨진다.

TargetType

[입력] *TargetValuePtr 버퍼의 C 데이터 타입 식별자이다.

TargetType이 SQL_ARD_TYPE 일 경우, 드라이버는 ARD의 SQL_DESC_CONCISE_TYPE 필드에 지정된 타입 식별자를 사용한다.

TargetValuePtr

[출력] 데이터가 반환되는 버퍼의 포인터이다.

TargetValuePtr은 NULL이 될 수 없다.

BufferLength

[입력] *TargetValuePtr 버퍼의 바이트 길이이다.

드라이버는 문자나 이진 데이터와 같은 가변 길이 데이터를 반환할 때, *TargetValuePtr 버퍼의 끝을 넘어 쓰지 않도록 하기 위해 BufferLength를 사용한다. *TargetValuePtr에 문자 데이터를 반환할 때 드라이버가 null 종료 문자도 계산하는 것에 주의해야 한다. 따라서 *TargetValuePtr은 null 종료 문자를 위한 공간을 포함해야 하며, 그렇지 않으면 드라이버가 데이터를 자를 수 있다.

드라이버는 정수 또는 날짜 구조체와 같은 고정 길이 데이터를 반환할 때, 버퍼가 데이터를 저장하기에 충분히 큰 것으로 가정해 BufferLength를 무시한다. 그러므로 응용 프로그램은 고정 길이 데이터에 충분히 큰 버퍼를 할당하는 것이 중요하며, 그렇지 않으면 드라이버는 버퍼의 끝을 넘어 쓸 수 있다.

SQLGetData는 BufferLength가 0보다 작을 경우, SQLSTATE HY090 (invalid string or buffer length)를 반환한다.

StrLen_or_IndPtr

[출력] 길이 또는 지시자 값을 반환하는 버퍼 포인터이다. 이 인자가 null 포인터일 경우, 길이/ 지시자 값은 반환되지 않으며, NULL 데이터를 가져올 때 에러가 발생한다.

SQLGetData는 길이/ 지시자 버퍼에 반환 가능한 데이터의 길이와 SQL_NO_TOTAL, SQL_NULL_DATA를 반환할 수 있다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	Col_or_Param_Num으로 지정된 column에 대한 모든 데이터는 단일 함수 호출로 검색할 수 없다. SQL_NO_TOTAL이나 SQLGetData를 호출하기 전에 지정된 column에 남아 있는 데이터의 길이를 *StrLen_or_IndPtr에 반환한다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S07	Fractional truncation	하나 이상의 column에 대해 반환되는 데이터가 잘렸다. 숫자 데이터 타입이 잘린 경우, 소수 부분이 잘렸다. 기간 구성 요소를 포함하는 time, timestamp, interval 데이터 타입이 잘린 경우, 시간의 소수 부분이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	결과 집합에 있는 column 데이터의 값을 TargetType 인자가 지정한 C 데이터 타입으로 변환할 수 없다.
07009	Invalid descriptor index	Col_or_Param_Num 인자값이 결과 집합의 column의 개수보다 크다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.

SQLSTATE	Error	설명
22002	Indicator variable required but not supplied	StrLen_or_IndPtr이 null 포인터이며, 반환된 데이터가 NULL이다.
22003	Numeric value out of range	하나 이상의 column에서 반환된 숫자값의 (소수가 아닌) 정수 부분이 잘렸다.
22007	Invalid datetime format	결과 집합의 문자열이 유효한 date, time, timestamp 형식이 아니다.
22012	Division by zero	0으로 나눈 산술식의 결과값이 반환되었다.
22015	Interval field overflow	Exact numeric 또는 interval SQL 데이터 타입에서 interval C 타입으로 지정했을 때, 선행 필드의 유효 숫자가 손실되었다. SQL 타입의 값을 interval C 타입에 표현할 수 없다.
22018	Invalid character value for cast specification	C 버퍼의 character set으로 표현할 수 없는 문자가 결과 집합의 문자 column에 포함되었다. C 타입이 exact 또는 approximate numeric, datetime, interval 데이터 타입이고, SQL 타입이 문자 데이터 형식일 때, C 타입에 바인딩 되는 column의 값이 유효한 문자가 아니다.
24000	Invalid cursor state	SQLFetch 또는 SQLFetchScroll을 호출하지 않고, 함수를 호출하였다. StatementHandle이 실행된 상태이지만, StatementHandle과 관련된 결과 집합이 없다. SQLFetch 또는 SQLFetchScroll을 호출하여 커서가 열렸지만, 커서가 결과 집합의 시작 전, 또는 끝 이후를 가리키고 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY003	Program type out of range	TargetType 인자값이 유효하지 않다.
HY009	Invalid use of null pointer	TargetValuePtr 인자가 null 포인터이다.
HY010	Function sequence error	지정된 StatementHandle이 실행된 상태가 아니다. 이 함수는 SQLExecDirect, SQLExecute, 카탈로그 함수를 호출하지 않고 호출되었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다.
HY090	Invalid string or buffer length	BufferLength 인자값이 0보다 작다.

설명

SQLGetData는 지정된 column의 데이터를 반환하며, SQLFetch 또는 SQLFetchScroll의 결과 집합에서 하나 이상의 row를 가져온 후에만 호출할 수 있다. 응용 프로그램 제한으로 인해 SQLGetData를 한 번만 호출하여 반환하기에 가변 길이 데이터가 너무 큰 경우, SQLGetData가 부분 검색을 수행할 수 있다.

SQLGetData 사용

TargetType 인자가 interval 데이터 타입일 경우, 기본값 interval leading precision (2), interval seconds precision (6)이 ARD의 SQL_DESC_DATETIME_INTERVAL_PRECISION 및 SQL_DESC_PRECISION 필드 각각에 설정된다. TargetType 인자가 SQL_C_NUMERIC일 경우, 기본값 precision (38), scale (0)이 ARD의 SQL_DESC_PRECISION 및 SQL_DESC_SCALE 필드 각각에 설정된다. 만약 기본 precision과 scale이 적절하지 않은 경우, 응용 프로그램은 SQLSetDescField나 SQLSetDescRec를 호출하여 해당 설명자 필드를 명시적으로 설정해야 한다.

가변 길이 데이터의 부분 검색

SQLGetData는 SQL 데이터 타입이 SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_WCHAR, SQL_WVARCHAR, SQL_WLONGVARCHAR, SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY인 가변 길이 데이터를 부분 검색할 수 있다.

Column의 데이터를 부분적으로 검색하기 위해 응용 프로그램은 같은 column에 대해 SQLGetData를 연속적으로 여러 번 호출한다. 각 호출에서 SQLGetData는 데이터의 다음 부분을 반환한다. 응용 프로그램은 문자 데이터의 중간 부분 null 종료 문자 제거에 주의하면서 부분을 재조립해야 한다. 반환될 데이터가 더 있을 경우, SQLGetData는 SQL_SUCCESS_WITH_INFO와 SQLSTATE 01004 (data truncated)를 반환하며, 데이터의 마지막 부분을 반환할 경우, SQL_SUCCESS를 반환한다.

SQLGetData는 고정 길이 데이터의 부분을 반환하는데 사용할 수 없다. 고정 길이 데이터를 포함하는 column에 대해 SQLGetData를 한 번 이상 호출한 경우, 처음 호출 이후로는 SQL_NO_DATA를 반환한다.

SQLGetData로 데이터 검색

지정된 column에 대한 데이터를 반환하기 위해, SQLGetData는 다음 절차를 수행한다.

1. Column에 모든 데이터가 반환되었을 경우, SQL_NO_DATA가 반환된다.
2. 데이터가 NULL이면 *StrLen_or_IndPtr을 SQL_NULL_DATA로 설정한다. 데이터가 NULL이고 *StrLen_or_IndPtr이 null 포인터이면, SQLGetData는 SQLSTATE 22002 (indicator variable required but not supplied)를 반환한다.

Column의 데이터가 NULL이 아니면, SQLGetData는 3번 절차를 진행한다.

3. SQL_ATTR_MAX_LENGTH 명령문 속성이 0이 아닌 값으로 설정되거나, column이 문자 또는 이진 데이터를 포함하고 SQLGetData가 column에 대해 이전에 호출된 적이 없으면 데이터는 SQL_ATTR_MAX_LENGTH 바이트 길이로 잘린다.

노트

SQL_ATTR_MAX_LENGTH 명령문 속성은 네트워크 트래픽 절감을 목적으로 한다. 이것은 일반적으로 데이터 소스에 의해 구현되며 데이터가 네트워크에서 반환되기 전에 데이터를 자른다. 드라이버와 데이터 소스들이 이것을 지원하도록 요구하지 않으므로 데이터가 특정 크기로 잘리도록 보장하려면 응용 프로그램이 그 크기의 버퍼를 할당하고 BufferLength 인자의 크기를 지정해야 한다.

4. TargetType에 지정된 타입으로 데이터를 변환한다. 데이터 타입에 맞는 기본 정밀도와 스케일 값이 데이터에 주어진다. TargetType이 SQL_ARD_TYPE이라면, ARD의 SQL_DESC_CONCISE_TYPE 필드에 있는 데이터 타입이 사용되고, 데이터는 SQL_DESC_CONCISE_TYPE 필드의 데이터 타입에 따라, ARD의 SQL_DESC_DATETIME_INTERVAL_PRECISION, SQL_DESC_PRECISION 그리고 SQL_DESC_SCALE 필드의 정밀도와 스케일을 제공한다. 모든 기본 정밀도 또는 스케일이 적절하지 않다면, 응용 프로그램은 SQLSetDescField 또는 SQLSetDescRec를 호출하여 적절한 설명자 필드를 명시적으로 설정해야 한다.
5. 데이터가 문자나 바이너리 같은 가변 길이 데이터 타입으로 전환되면, SQLGetData는 데이터의 길이가 BufferLength를 초과하였는지 검사한다. (null 종료 문자 포함) 문자 데이터의 길이가 BufferLength를 초과하면, SQLGetData는 null 종료 문자를 제외한 BufferLength 만큼 데이터를 자른다. 바이너리 데이터의 길이가 데이터 버퍼의 길이를 초과하면 SQLGetData는 BufferLength 바이트 길이로 자른다.
제공되는 데이터 버퍼에 null 종료 문자를 저장하지 못한다면, SQLGetData는 SQL_SUCCESS_WITH_INFO와 SQLSTATE 01004를 반환한다.
SQLGetData는 고정 길이 데이터 타입으로 변환되는 데이터를 자르지 않는다. 이 경우, *TargetValuePtr의 길이는 항상 데이터 타입의 사이즈로 간주된다.
6. *TargetValuePtr에 변환된 데이터를 저장한다. SQLGetData는 out of line인 데이터를 반환할 수 없음에 유의한다.
7. *StrLen_or_IndPtr에 데이터의 길이를 저장한다. StrLen_or_IndPtr이 null 포인터일 경우, SQLGetData는 길이를 반환하지 않는다.
 - 문자 또는 바이너리 데이터의 경우, 이것은 변환된 후 BufferLength로 잘리기 전의 데이터 길이이다. 매우 긴 데이터일 경우, 드라이버가 변환된 후의 데이터 길이를 확인할 수 없다면 SQL_SUCCESS_WITH_INFO를 반환하고 길이를 SQL_NO_TOTAL로 설정한다. (SQLGetData의 마지막 호출은 반드시 데이터의 길이를 0 또는 SQL_NO_TOTAL이 아닌 길이값을 반환해야 한다.) 데이터가 SQL_ATTR_MAX_LENGTH 명령문 속성에 의해 잘린 경우, 이 속성값은 *StrLen_or_IndPtr에 저장된다. 이것은 이 속성이 변환되기 전에 서버에서 데이터를 전달하도록 설계되었기 때문이고 드라이버는 실제 길이가 무엇인지 알아낼 수 없다. 이것은 동일한 column에 대해 SQLGetData를 연속적으로 여러 번 호출할 때 현재 호출의 시작에서 이용할 수 있는 데이터 길이이다. 즉, 길이는 각각의 후속 호출로 인해 감소한다.
 - 다른 모든 데이터 타입의 경우 이것은 변환 이후의 데이터 길이이다. 즉, 데이터가 변환되는 타입의 크기이다.
8. 데이터 변환 중에 기본값이 손실되지 않고 데이터가 잘리거나 (예: 실수 1.234가 변환될 때 정수 1로 잘림) BufferLength가 작아서 잘리는 경우 (예: "abcdef"가 4 바이트 길이 버퍼에 저장됨), SQLGetData는 SQLSTATE 01004 (data truncated)와 SQL_SUCCESS_WITH_INFO를 반환한다. SQL_ATTR_MAX_LENGTH 명령문 속성 때문에 기본값이 손실되지 않고 데이터가 잘리면, SQLGetData는 SQL_SUCCESS를 반환하고 SQLSTATE 010

04 (data truncated)를 반환하지 않는다.

SQLGetData가 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않는 경우, (SQLGetData가 바인딩된 column에 대해 호출된 경우) 바인딩 된 데이터 버퍼의 내용과 길이/ 지시자 버퍼는 정의되지 않는다.

SQLGetData를 연속적으로 호출하면 요청된 마지막 column부터 데이터를 검색한다. 이전 offset은 유효하지 않게 된다.

예를 들어서, 다음과 같은 순서로 수행될 때,

SQLGetData(icol=n), SQLGetData(icol=m), SQLGetData(icol=n)

두 번째 호출 SQLGetData(icol=n)은 n column부터 데이터 검색을 시작한다. 데이터의 모든 offset은 이전 SQLGetData 호출 때문에 더 이상 유효하지 않다.

SQLGetData와 설명자

SQLGetData는 어떤 설명자 필드와도 직접 상호 작용하지 않는다.

TargetType이 SQL_ARD_TYPE인 경우, ARD의 SQL_DESC_CONCISE_TYPE 필드의 데이터 타입을 사용한다. TargetType이 SQL_ARD_TYPE 또는 SQL_C_DEFAULT인 경우, SQL_DESC_CONCISE_TYPE 필드의 데이터 타입에 따라 ARD의 SQL_DESC_DATETIME_INTERVAL_PRECISION, SQL_DESC_PRECISION 및 SQL_DESC_SCALE 필드의 precision과 scale이 주어진다.

SQLGetDescField

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetDescField는 설명자 레코드의 단일 필드의 현재 설정 또는 값을 반환한다.

구문

```
SQLRETURN SQLGetDescField(  
    SQLHDESC      DescriptorHandle,  
    SQLSMALLINT    RecNumber,  
    SQLSMALLINT    FieldIdentifier,  
    SQLPOINTER     ValuePtr,  
    SQLINTEGER     BufferLength,  
    SQLINTEGER *    StringLengthPtr);
```

인자

DescriptorHandle

[입력] 설명자 핸들이다.

RecNumber

[입력] 응용 프로그램이 찾고자 하는 정보의 설명자 레코드이다. FieldIdentifier 인자가 헤더 필드일 경우, RecNumber는 무시된다. RecNumber가 SQL_DESC_COUNT 보다 작거나 같고, row에 column이나 매개 변수에 대한 데이터가 포함되지 않은 경우, SQLGetDescField는 필드의 기본값을 반환한다.

FieldIdentifier

[입력] 값이 반환되는 설명자 필드이다.

ValuePtr

[출력] 설명자 정보를 반환하는 버퍼의 포인터이다. 데이터 타입은 FieldIdentifier 값에 의존적이다.

ValuePtr이 정수 타입일 경우, 응용 프로그램은 0으로 초기화 된 SQLULEN 버퍼를 사용해야 한다.

ValuePtr이 null일 경우, StringLengthPtr은 반환 가능한 전체 바이트 수 (null 종료 문자 제외)를 반환한다.

BufferLength

[입력] FieldIdentifier가 ODBC에 정의된 필드이고, ValuePtr이 문자열이나 이진 버퍼를 가리킬 경우, 이 인자는 *ValuePtr의 길이여야 한다. FieldIdentifier가 ODBC에 정의된 필드이고, *ValuePtr이 정수일 경우, 이 인자는 무시된다.

StringLengthPtr

[출력] *ValuePtr에서 반환 가능한 총 바이트 수 (문자 데이터에 대한 null 종료 바이트 제외)를 반환하는 포인터이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE

RecNumber가 현재 설명자 레코드의 수보다 큰 경우, SQL_NO_DATA가 반환된다.

DescriptorHandle이 IRD 핸들이고 명령문이 준비 또는 실행 상태에 있지만 관련된 커서가 없을 경우, SQL_NO_DATA를 반환한다.

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	*ValuePtr의 버퍼 길이가 설명자 필드값의 길이보다 작아서 잘렸다. 남아 있는 설명자 필드의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07009	Invalid descriptor index	RecNumber 인자가 0으로 설정되었고, SQL_ATTR_USE_BOOKMARKS 상태 속성이 SQL_UB_OFF로 설정되었으며 DescriptorHandle 인자가 IRD 핸들이었다. FieldIdentifier 인자가 레코드 필드이었고, RecNumber 인자가 0이었으며, DescriptorHandle 인자가 IPD 핸들이었다. RecNumber 인자가 0보다 작았다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY007	Associated statement is not prepared	DescriptorHandle이 IRD 핸들과 관련되어 있으며 관련된 상태 핸들이 준비나 실행 단계에 있지 않다.
HY010	Function sequence error	호출된 상태에서 여전히 수행 중인 비동기적 수행 함수에 대한 StatementHandle과 관련된 DescriptorHandle이다. SQLExecute, SQLExecDirect, SQLBulkOperations, SQLSetPos가 호출되었고, SQL_NEED_DATA를 반환하는 StatementHandle과 관련된 DescriptorHandle이다.

SQLSTATE	Error	설명
		DescriptorHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수가 호출되었고, SQLGetDescField가 호출되었을 때도 이 비동기 수행 함수가 여전히 실행되고 있었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY021	Inconsistent descriptor information	SQL_DESC_TYPE과 SQL_DESC_DATETIME_INTERVAL_CODE 필드가 ODBC SQL 타입, 특정 드라이버의 (IPD를 위한) SQL 타입 또는 (APD 또는 ARD를 위한) ODBC C 타입에 유효한 형식이 아니다.
HY090	Invalid string or buffer length	*ValuePtr이 문자열이고, BufferLength가 0보다 작다.
HY091	Invalid descriptor field Identifier	FieldIdentifier가 ODBC에서 정의된 필드가 아니고 구현 정의된 값이 아니다. FieldIdentifier가 DescriptorHandle에 대해 정의되지 않았다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

응용 프로그램은 설명자 레코드의 단일 필드값을 반환하기 위해 SQLGetDescField를 호출할 수 있다. SQLGetDescField 호출은 헤더 필드와 레코드 필드, 북마크 필드를 포함한 모든 설명자 타입의 모든 필드 설정을 반환할 수 있다. 응용 프로그램은 SQLGetDescField를 반복 호출하여 동일하거나 다른 설명자에서 임의의 순서로 다수의 필드 설정을 가져올 수 있다. SQLGetDescField는 드라이버 정의 설명자 필드를 반환하기 위해 호출될 수 있다.

성능상의 이유로 응용 프로그램은 명령문을 실행하기 전에 IRD를 위해 SQLGetDescField를 호출하면 안된다.

이름, 데이터 타입, column 또는 매개 변수 데이터 크기에 대한 다수의 필드 설정은 한 번의 SQLGetDescRec 호출로 검색할 수 있다. SQLGetStmtAttr은 명령문 속성인 설명자 헤더 안의 단일 필드 설정을 반환하기 위해 호출될 수 있다. SQLColAttribute와 SQLDescribeCol, SQLDescribeParam은 레코드나 북마크 필드들을 반환한다.

응용 프로그램이 특정한 설명자 타입에 대해 정의되지 않은 필드값을 검색하기 위해 SQLGetDescField를 호출하였을 때, 함수는 SQL_SUCCESS를 반환하지만 반환되는 필드의 값은 정의하지 않는다. 예를 들어, APD/ ARD의 SQL_DESC_NAME이나 SQL_DESC_NULLABLE 필드에 대해 SQLGetDescField를 호출하면 SQL_SUCCESS를 반환하지만 필드값은 정의하지 않는다.

응용 프로그램이 특정 설명자 타입에 대해 정의되어 있지만 기본값이나 어떠한 설정도 되어 있지 않은 필드값을 검색하기 위해 SQLGetDescField를 호출할 경우, 함수는 SQL_SUCCESS를 반환하지만 반환되는 필드값은 정의되지 않는다.

헤더 필드

각 설명자는 다음 필드들로 구성된다.

SQL_DESC_ALLOC_TYPE[All]

읽기 전용 SQLSMALLINT인 이 헤더 필드는 서술자가 드라이버에 의해 자동으로 할당됐는지 아니면 응용 프로그램에 의해 명백하게 할당되었는지 여부를 명시한다. 응용 프로그램은 이 필드를 얻을 수는 있지만 수정할 수는 없다. 드라이버가 설명자를 자동으로 할당했을 때 필드는 SQL_DESC_ALLOC_AUTO로 설정된다.

SQL_DESC_ARRAY_SIZE[Application descriptors]

ARD에 있고 SQLULEN인 이 헤더 필드는 row 집합의 row 개수를 명시한다. 이것은 SQLFetch, SQLFetchScroll, SQLBulkOperations 또는 SQLSetPos를 호출하여 생기는 연산들이 반환하는 row들의 개수이다.

APD에 있고 SQLULEN인 이 헤더 필드는 각 매개 변수 값의 개수를 명시한다.

이 필드의 기본값은 1이다. SQL_DESC_ARRAY_SIZE가 1보다 크면 APD 또는 ARD의 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR 및 SQL_DESC_OCTET_LENGTH_PTR은 배열을 가리킨다. 각 배열의 상수는 이 필드의 값과 같다.

ARD에 있는 이 필드는 SQL_ATTR_ROW_ARRAY_SIZE 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다. APD의 필드 역시 SQL_ATTR_PARAMSET_SIZE 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다.

SQL_DESC_ARRAY_STATUS_PTR[All]

각 설명자 형식에 대한 SQLUSMALLINT* 인 헤더 필드는 SQLUSMALLINT 값의 배열을 가리킨다. 이 배열에는 row 상태 배열 (IRD), 매개 변수 상태 배열 (IPD), row 연산 배열 (ARD), 매개 변수 연산 배열(APD)이 있다.

IRD에 있는 이 헤더 필드는 SQLBulkOperations, SQLFetch, SQLFetchScroll, 또는 SQLSetPos를 호출한 후의 상태값을 포함하는 row 상태 배열을 가리킨다. 응용 프로그램은 SQLUSMALLINT 배열을 할당하고 이 필드가 배열을 가리키게 해야 한다. 이 필드는 기본적으로 NULL 포인터이다. 드라이버는 SQL_DESC_ARRAY_STATUS_PTR 필드를 NULL 포인터로 설정하지 않는 한 배열을 생성할 것이다.

주의

응용 프로그램이 IRD의 SQL_DESC_ARRAY_STATUS_PTR 필드가 가리키는 row 상태 배열의 요소들을 설정할 경우, 드라이버의 행동은 정의되지 않는다.

SQLBulkOperations, SQLFetch, SQLFetchScroll 또는 SQLSetPos를 호출하면 배열이 초기화되어 생성된다. 이 호출이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 이 필드가 가리키는 배열의 내용은 정의되지 않는다. 배열의 요소들은 다음 값들을 포함할 수 있다.

- SQL_ROW_SUCCESS: Row가 성공적으로 회수되었고, 마지막으로 회수된 이후로 아무 변화가 없었다.
- SQL_ROW_SUCCESS_WITH_INFO: Row가 성공적으로 회수되었고, 마지막으로 회수된 이후로 아무 변화가 없었다. 하지만 row에 대해 경고가 반환되었다.

- SQL_ROW_ERROR: Row를 회수하는 동안 오류가 발생했다.
- SQL_ROW_UPDATED: Row가 성공적으로 회수되었고, 마지막으로 회수된 이후로 갱신되었다. Row가 다시 회수되었을 경우, 상태는 SQL_ROW_SUCCESS이다.
- SQL_ROW_DELETED: 마지막으로 회수된 이후로 row가 삭제되었다.
- SQL_ROW_ADDED: SQLBulkOperations가 row를 입력했다. 다시 row가 회수되었을 경우, 상태는 SQL_ROW_SUCCESS이다.
- SQL_ROW_NOROW: Row 집합이 결과 집합의 마지막과 겹쳐진다. 그리고 row 상태 배열의 요소에 대응하여 반환된 row가 없다.

IRD의 이 필드는 SQL_ATTR_ROW_STATUS_PTR 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다.

IRD의 SQL_DESC_ARRAY_STATUS_PTR 필드는 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO가 반환된 이후에만 유효하다. 반환 코드가 이것들 중 하나가 아닐 경우, SQL_DESC_ROWS_PROCESSED_PTR이 가리키는 것은 정의되지 않는다.

IPD에 있는 이 헤더 필드는 SQLExecute나 SQLExecDirect를 호출한 후에 매개 변수 각각의 상태 정보를 포함하는 매개 변수 상태 배열을 가리킨다. 만약 SQLExecute나 SQLExecDirect를 호출하고 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 이 필드가 가리키는 배열의 내용은 정의되지 않는다. 응용 프로그램은 SQLUSMALLINT 배열을 할당하고 이 필드가 배열을 가리키게 해야 한다. 드라이버는 SQL_DESC_ARRAY_STATUS_PTR 필드를 NULL 포인터로 설정하지 않는 한, 배열을 생성할 것이다.

배열의 요소는 다음 값들을 포함한다.

- SQL_PARAM_SUCCESS: 이 매개 변수 집합에 대한 SQL 명령문이 성공적으로 실행되었다.
- SQL_PARAM_SUCCESS_WITH_INFO: 이 매개 변수 집합에 대한 SQL 명령문이 성공적으로 실행되었지만 진단 데이터 구조체에 사용 가능한 경고 정보가 있다.
- SQL_PARAM_ERROR: 이 매개 변수 집합을 처리하는 도중에 오류가 발생했다. 추가적인 오류 정보는 진단 데이터 구조체에 있다.
- SQL_PARAM_UNUSED: 이전의 몇몇 매개 변수를 처리하던 도중에 오류가 발생하였거나 APD의 SQL_DESC_ARRAY_STATUS_PTR 필드가 명시한 배열의 매개 변수 집합에 SQL_PARAM_IGNORE가 설정되었기 때문에 사용되지 않았다.
- SQL_PARAM_DIAG_UNAVAILABLE: 진단 정보를 사용할 수 없다. 예를 들어, 드라이버는 매개 변수의 배열을 단일한 것으로 취급하여 오류 정보의 level을 생성하지 않는다.

IPD에 있는 이 필드는 SQL_ATTR_PARAM_STATUS_PTR 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다.

ARD에서 해당 필드는 SQLSetPos 연산에서 해당 row를 무시할지 여부를 설정하기 위해 응용 프로그램으로부터 설정된 값들에 대한 row 연산 배열을 가리킨다. 배열의 요소들은 다음 값들을 포함할 수 있다.

- SQL_ROW_PROCEED: Row는 SQLSetPos를 이용한 벌크 연산에 포함된다. (이 설정은 연산이 해당 row에서 발생한다고 보장하지 않는다. Row가 IRD row 상태 배열의 SQL_ROW_ERROR 상태를 가지고 있을 경우, 드라이버는 row에 대한 연산을 이행할 수 없다.)
- SQL_ROW_IGNORE: Row는 SQLSetPos를 이용한 벌크 연산에서 제외된다.

배열의 요소가 설정되어 있지 않은 경우 모든 row는 벌크 연산에 포함된다. ARD의 SQL_DESC_ARRAY_STATUS_

PTR 필드 안의 값이 NULL 포인터이면 모든 row가 벌크 연산에 포함된다. 포인터가 유효한 배열을 가리키고 있고 배열의 모든 요소가 SQL_ROW_PROCEED인 것으로 해석된다. 배열의 모든 요소를 SQL_ROW_IGNORE로 설정할 경우, 무시된 row에 대한 row 상태 배열 안의 값은 변경되지 않는다.

ARD 안에 있는 이 필드 역시 SQLSetStmtAttr을 SQL_ATTR_ROW_OPERATION_PTR 속성과 함께 호출하여 설정할 수 있다.

APD에 있는 이 헤더필드는 SQLExecute나 SQLExecDirect가 호출될 때 이 매개 변수 집합을 무시할지 여부를 나타내기 위해 응용 프로그램이 설정할 수 있는 값의 매개 변수 연산 배열을 가리킨다. 배열에 있는 요소들은 다음 값들을 포함할 수 있다.

- SQL_PARAM_PROCEED: 매개 변수 집합은 SQLExecute나 SQLExecDirect 호출에 포함된다.
- SQL_PARAM_IGNORE: 매개 변수 집합은 SQLExecute나 SQLExecDirect 호출에 포함되지 않는다.

배열 요소가 설정되지 않은 경우, 배열의 모든 매개 변수 집합들은 SQLExecute나 SQLExecDirect를 호출하기 위해 사용된다. APD의 SQL_DESC_ARRAY_STATUS_PTR 필드값이 NULL 포인터일 경우, 모든 매개 변수 집합이 사용된다. 포인터가 유효한 배열이거나 모든 요소가 SQL_PARAM_PROCEED인 배열을 가리키는 것과 동일하게 해석된다.

APD에 있는 이 필드 역시 SQL_ATTR_PARAM_OPERATION_PTR 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다.

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

SQLENUM* 인 이 헤더 필드는 바인딩의 offset을 가리킨다. 이것은 기본적으로 NULL 포인터로 설정된다. 이 필드가 NULL 포인터가 아닐 경우, 드라이버는 포인터와 회수 시점 설명자 레코드 (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR 및 SQL_DESC_OCTET_LENGTH_PTR)의 NULL 아닌 값을 가지는 지연된 필드 각각의 값을 역참조하고 바인딩될 때 새로운 포인터의 값을 사용한다.

바인딩 offset은 항상 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR 및 SQL_DESC_OCTET_LENGTH_PTR 필드들에 직접적으로 추가된다. 만약 offset이 다른 값으로 갱신되었다면 새로운 값은 각 설명자 필드값으로 직접적으로 계속 추가된다. 새로운 offset은 필드의 이전 offset 값에 추가되는 것이 아니다.

이 필드는 지연된 필드이다. 이 필드는 설정되는 시점에 사용되지 않고 데이터 버퍼들의 주소를 확인해야 할 때 드라이버에 의해 나중에 사용된다.

ARD 안의 이 필드 또한 SQL_ATTR_ROW_BIND_OFFSET_PTR 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다. ARD에 있는 이 필드는 SQL_ATTR_PARAM_BIND_OFFSET_PTR 속성과 SQLSetStmtAttr을 호출하여 설정할 수 있다.

자세한 내용은 SQLFetchScroll이나 SQLBindParameter를 참조한다.

SQL_DESC_BIND_TYPE[Application descriptors]

SQLINTEGER인 이 헤더 필드는 바인딩의 방향을 설정하는데 사용된다.

ARD에 있는 이 필드는 SQLFetchScroll 또는 SQLFetch가 관련된 명령문 핸들에 호출되었을 때의 바인딩 방향을 명시한다.

Column들에 대한 column 방향 바인딩을 선택하려면 이 필드를 SQL_BIND_BY_COLUMN (기본값) 으로

설정한다.

ARD에 있는 이 필드 역시 SQL_ATTR_ROW_BIND_TYPE 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다.

이 필드는 동적 매개 변수에 사용될 바인딩 방향을 명시한다.

매개 변수에 대한 column 방향 바인딩을 선택하려면 이 필드를 SQL_BIND_BY_COLUMN (기본값)으로 설정한다.

이 필드 역시 SQL_ATTR_PARAM_BIND_TYPE 속성을 사용한 SQLSetStmtAttr을 호출하여 설정할 수 있다.

SQL_DESC_COUNT[All]

SQLSMALLINT인 이 헤더 필드는 데이터를 포함하는 최상위 레코드의 1-based 인덱스를 명시한다. 드라이버가 설명자에 데이터 구조체를 설정할 때 중요한 레코드의 수가 얼마나 되는지 보여주려면 반드시 SQL_DESC_COUNT를 설정해야 한다. 응용 프로그램이 데이터 구조체의 인스턴스를 할당할 때 레코드를 위해 얼마나 많은 공간을 예약해야 할지 명시할 필요는 없다. 응용 프로그램이 레코드들의 내용물들을 명시하는 것처럼, 드라이버는 설명자 핸들이 충분한 크기의 데이터 구조체를 나타내도록 보장하기 위해 필요하여 요청된 작업을 수행한다.

SQL_DESC_COUNT가 바인딩 된 모든 데이터 column 또는 모든 매개 변수의 개수가 아닌 최상위 레코드의 개수이다. 최상위 column 또는 매개 변수의 바인딩이 해제될 경우, SQL_DESC_COUNT는 다음 최상위 column 또는 매개 변수의 수로 바뀐다. 최상위 column 또는 매개 변수 보다 작은 수의 column 또는 매개 변수의 바인딩이 해제될 경우, (TargetValuePtr 인자를 NULL 포인터로 설정하고 SQLBindCol을 호출하거나 ParameterValuePtr 인자를 NULL 포인터로 설정하여 SQLBindParameter를 호출하는 경우) SQL_DESC_COUNT는 변경되지 않는다. 추가되는 column 또는 매개 변수가 데이터를 포함하는 최상위 레코드보다 큰 수로 바인딩될 경우, 드라이버가 자동으로 SQL_DESC_COUNT 필드의 값을 증가시킨다. SQL_UNBIND 옵션으로 SQLFreeStmt를 호출하여 모든 column들에 대한 바인딩이 해제될 경우, ARD와 IRD 안에 있는 SQL_DESC_COUNT 필드는 0으로 설정된다. SQL_RESET_PARAMS 옵션으로 SQLFreeStmt가 호출되면, APD와 IPD에 있는 SQL_DESC_COUNT 필드들은 0으로 설정된다.

SQL_DESC_COUNT 값은 응용 프로그램에서 SQLSetDescField를 호출하여 명시적으로 설정할 수 있다. SQL_DESC_COUNT 값이 명시적으로 감소할 경우, SQL_DESC_COUNT의 새로운 값보다 큰 값을 가지는 모든 레코드들이 효과적으로 삭제된다. ARD의 SQL_DESC_COUNT 필드값이 명시적으로 0으로 설정되면, 바인딩된 북마크 column을 제외한 모든 버퍼가 해제된다. ARD의 이 필드의 레코드 수는 바인딩된 북마크 column을 포함하지 않는다. 북마크 column의 바인딩을 해제하는 유일한 방법은 SQL_DESC_DATA_PTR 필드를 NULL 포인터로 설정하는 것이다.

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

IRD의 SQLULEN*인 이 헤더 필드는 SQLFetch나 SQLFetchScroll이 호출된 후에 회수된 row들의 개수, SQLBulkOperations나 SQLSetPos를 호출하여 이행된 벌크 연산의 영향을 받은 row들의 개수와 오류 row들의 개수를 포함하는 버퍼를 가리킨다.

IPD의 SQLINTEGER*인 이 헤더 필드는 처리된 매개 변수 집합과 오류 개수를 포함하는 버퍼를 가리킨다. NULL 포인터일 경우, 개수를 반환하지 않는다.

SQL_DESC_ROWS_PROCESSED_PTR은 SQLFetch나 SQLFetchScroll(IRD)을 호출한 후 또는 SQLExecute, SQLExecDirect나 SQLParamData(IPD)를 호출하여 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를

반환받은 후에만 유효하다. 만약 위의 함수들이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않으며 SQL_NO_DATA를 반환할 때까지 버퍼의 값은 0으로 설정된다.

ARD에 있는 이 필드 또한 SQL_ATTR_ROWS_FETCHED_PTR 속성과 함께 SQLSetStmtAttr을 호출하여 설정할 수 있다. APD의 경우 SQL_ATTR_PARAMS_PROCESSED_PTR 속성과 함께 호출하여 설정할 수 있다.

이 필드가 가리키는 버퍼는 응용 프로그램에 의해 할당된 것이다. 이는 드라이버가 설정한 지연된 출력 버퍼이다. 기본적으로 NULL 포인터가 설정된다.

레코드 필드

각 설명자는 설명자의 형식에 의존하여 column 데이터 또는 동적 매개 변수 중 하나를 정의한 필드들로 구성된 하나 이상의 레코드를 포함한다. 각 레코드는 단일 column 또는 매개 변수의 완전한 명세이다.

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs]

읽기전용 SQLINTEGER인 이 레코드 필드는 column이 자동 증가 column이면 SQL_TRUE를, 그렇지 않으면 SQL_FALSE 값을 가진다. 자동 증가 column인 경우 반드시 읽기전용일 필요는 없다.

SQL_DESC_BASE_COLUMN_NAME[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 결과 집합 column의 기본 column 이름을 포함한다. 기본 column 이름이 존재하지 않을 경우, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_TABLE_NAME[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 결과 집합 column의 기본 테이블 이름을 포함한다. 기본 테이블 이름이 정의될 수 없거나 사용될 수 없다면, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_CASE_SENSITIVE[Implementation descriptors]

읽기전용 SQLINTEGER인 이 레코드 필드는 column 또는 매개 변수를 정렬하고 비교할 때 대소문자를 구분할 경우 SQL_TRUE를, 대소문자를 구분하지 않거나 문자가 아닌 column인 경우에는 SQL_FALSE를 포함한다.

SQL_DESC_CATALOG_NAME[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 column이 포함된 기본 테이블의 카탈로그를 포함한다. Column이 표현식이거나 view의 일부일 경우, 반환값은 드라이버에 따라 다르다. 데이터 소스가 카탈로그를 지원하지 않거나 카탈로그를 확인할 수 없을 경우, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_CONCISE_TYPE[All]

SQLSMALLINT인 이 헤더 필드는 datetime과 interval 데이터 형식을 포함하는 모든 데이터 형식에 대해 간결한 데이터 형식을 지정한다.

SQL_DESC_CONCISE_TYPE, SQL_DESC_TYPE과 SQL_DESC_DATETIME_INTERVAL_CODE 필드의 값은 상호 의존적이다. 필드들 중에 하나의 time이 설정되면 다른 것들도 설정되어야 한다. SQL_DESC_CONCISE_TYPE은 SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 설정할 수 있다. SQL_DESC_TYPE은 SQLSetDescField나 SQLSetDescRec를 호출하여 설정할 수 있다.

SQL_DESC_CONCISE_TYPE을 interval 또는 datetime 데이터 형식을 제외한 간결한 데이터 형식으로 설정할 경우, SQL_DESC_TYPE 필드는 같은 값으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 0으로 설정된다.

SQL_DESC_CONCISE_TYPE을 간결한 datetime 또는 interval 데이터 형식으로 설정한 경우, SQL_DESC_TYPE 필드는 상세한 형식 (SQL_DATETIME 또는 SQL_INTERVAL)으로 설정하고, SQL_DESC_DATETIME_INTERVAL_CODE 필드는 적절한 서브코드로 설정한다.

SQL_DESC_DATA_PTR [Application descriptors 및 IPD]

이 SQLPOINTER 레코드 필드는 매개변수 값 (APD의 경우) 또는 column 값 (ARD의 경우)을 저장할 변수의 주소를 가리킨다. 이 필드는 deferred field로써, 값을 설정하는 시점이 아니라 나중에 driver가 실제 데이터를 가져올 때 사용된다.

ARD의 SQL_DESC_DATA_PTR 필드에 지정된 column은 SQLBindCol을 호출할 때 TargetValuePtr 인수가 null pointer이거나, SQLSetDescField 또는 SQLSetDescRec를 호출하여 ARD의 SQL_DESC_DATA_PTR 필드를 null pointer로 설정할 경우 unbound 된다. 그러나 다른 필드들은 SQL_DESC_DATA_PTR 필드가 null pointer로 설정되더라도 영향을 받지 않는다.

SQLFetch 또는 SQLFetchScroll을 호출하여 이 필드가 가리키는 버퍼가 채워질 때 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO가 반환되지 않으면 버퍼의 내용은 undefined 상태이다.

APD, ARD 또는 IPD의 SQL_DESC_DATA_PTR 필드가 설정될 때마다 driver는 SQL_DESC_TYPE 필드에 유효한 ODBC C 데이터 타입이나 driver-specific 데이터 타입이 설정되었는지, 그리고 데이터 타입과 관련된 다른 필드들이 일관성을 유지하는지 (consistency) 확인한다. 특히 IPD의 SQL_DESC_DATA_PTR 필드는 오직 consistency 검사를 수행하기 위한 목적으로만 사용된다. 즉, 애플리케이션이 IPD의 SQL_DESC_DATA_PTR 필드를 설정한 후, 나중에 이 필드에 대해 SQLGetDescField를 호출한다고 하더라도, 반드시 이전에 설정한 값이 반환되는 것은 아니다.

SQL_DESC_DATETIME_INTERVAL_CODE[All]

SQLSMALLINT인 이 레코드 필드는 SQL_DESC_TYPE 필드가 SQL_DATETIME 또는 SQL_INTERVAL일 때 datetime 또는 interval 데이터 형식을 명시하기 위한 서브코드를 포함하고 있다. 이것은 SQL과 C 모두 마찬가지이다. 코드는 datetime 형식들의 "TYPE" 또는 "C_TYPE", interval 형식들의 "INTERVAL" 또는 "C_INTERVAL"을 대체한 "CODE"를 포함하는 데이터 형식 이름을 포함한다.

응용 프로그램 설명자의 SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE을 SQL_C_DEFAULT로 설정하고 설명자가 명령문 핸들과 관련되어 있지 않을 경우, SQL_DESC_DATETIME_INTERVAL_CODE의 내용은 정의되지 않는다.

이 필드는 다음 테이블에 나열된 datetime 데이터 형식들을 설정할 수 있다.

Datetime type	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIME_WITH_TIMEZONE/ SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_CODE_TIME_WITH_TIMEZONE
SQL_TYPE_TIMESTAMP SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE/ SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_CODE_TIMESTAMP_WITH_TIMEZONE

이 필드는 다음 테이블에 나열된 interval 데이터 형식들을 설정할 수 있다.

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MINUTE_TO_SECOND
SQL_INTERVAL_MONTH SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[All]

SQLINTEGER인 이 레코드 필드는 SQL_DESC_TYPE 필드가 SQL_INTERVAL일 경우, interval leading precision을 포함한다. SQL_DESC_DATETIME_INTERVAL_CODE 필드를 interval 데이터 형식으로 설정할 때, 이 필드는 기본 interval leading precision을 설정한다.

SQL_DESC_DISPLAY_SIZE[IRDs]

읽기전용 SQLLEN인 이 레코드 필드는 column으로부터 데이터를 보여주기 위해 필요한 최대 문자의 개수를 포함한다.

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors]

읽기전용 SQLSMALLINT인 이 레코드 필드는 column이 exact numeric column이고 0이 아닌 scale 과 고정된 precision을 가지고 있을 경우 SQL_TRUE를, 그렇지 않다면 SQL_FALSE를 설정한다.

SQL_DESC_INDICATOR_PTR[Application descriptors]

ARD에 있고 SQLLEN*인 이 레코드 필드는 지시자 변수를 나타낸다. 이 변수는 column 값이 NULL일 경우 SQL_NULL_DATE를 포함한다. APD의 경우, 동적 인자들을 NULL로 명시하기 위해 표시자 변수를 SQL_NULL_DATA로 설정한다. 그렇지 않으면 변수는 0이다.

ARD의 SQL_DESC_INDICATOR_PTR 필드가 NULL 포인터일 경우, 드라이버는 column의 NULL 여부에 대해 반환되는 정보를 막는다. Column이 NULL이고 SQL_DESC_INDICATOR_PTR이 NULL 포인터일 경우, 드라이버가 SQLFetch 또는 SQLFetchScroll을 호출한 후에 버퍼를 생성하려고 할 때 SQLSTATE 22002 (indicator variable required but not supplied)가 반환된다. SQLFetch 또는 SQLFetchScroll 호출이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않는다.

SQL_DESC_INDICATOR_PTR 필드는 SQL_DESC_OCTET_LENGTH_PTR이 나타내는 필드를 설정할지 여부를 결정한다. Column의 데이터 값이 NULL일 경우 드라이버는 지시자 변수를 SQL_NULL_DATA로 설정한다. 이 때 SQL_DESC_OCTET_LENGTH_PTR이 나타내는 필드는 설정하지 않는다. 데이터를 회수하는 동안 NULL 값을 받지 않을 경우, SQL_DESC_INDICATOR_PTR이 나타내는 버퍼를 0으로 설정하고 SQL_DESC_OCTET_LENGTH_PTR이 나타내는 버퍼는 데이터의 길이를 설정한다.

APD의 SQL_DESC_INDICATOR_PTR 필드가 NULL 포인터일 경우, 응용 프로그램은 인자들을 NULL로 명시하기 위해 이 설명자 레코드를 사용할 수 없다.

이 필드는 지연된 필드이다. 이 필드는 설정될 때는 사용되지 않지만 드라이버가 NULL 가능성 (ARD) 을 나타내거나 NULL 가능성 (APD)을 결정할 때 사용된다.

SQL_DESC_LABEL[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 column의 라벨이나 표지를 포함한다. Column이 라벨을 가지지 않을 경우, 이 변수는 column의 이름을 포함한다. Column이 명명되지 않거나 라벨을 사용할 수 없을 경우, 이 변수는 빈 문자열을 포함한다.

SQL_DESC_LENGTH[AII]

SQLULEN인 이 레코드 필드는 문자열이나 바이트 단위인 이진 데이터의 최대 길이 또는 실제 길이이다. 이것은 고정 길이 데이터 형식의 최대 길이이거나 가변 길이 데이터 형식의 실제 길이이다. 이 값은 항상 문자열의 끝에 있는 NULL 종료 문자를 제외한다. 형식이 SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP 또는 SQL interval 데이터 형식인 값의 경우, 이 필드는 datetime 또는 interval 값이 문자열로 재표현될 때의 문자들의 길이를 가진다.

이 필드의 값은 ODBC 2.x에 정의된 "length"의 값과는 다른 것이다.

SQL_DESC_LITERAL_PREFIX[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 문자 또는 드라이버가 이 데이터 형식의 문자를 접두사로 인식하는 문자들을 포함한다. 이 변수는 문자 접두사를 변환할 수 없는 데이터 형식에 대한 빈 문자열을 포함한다.

SQL_DESC_LITERAL_SUFFIX[IRDs]

읽기 전용 SQLCHAR*인 이 레코드 필드는 문자 또는 드라이버가 접미사로 인식하는 문자들을 포함한다. 이 변수는 문자 접미사를 적용할 수 없는 데이터 형식에 대한 빈 문자열을 포함한다.

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors]

읽기전용 SQLCHAR*인 이 레코드 필드는 데이터 형식에 지역화된 이름을 포함하는데 이는 데이터 형식의 정규 이름과 다를 수 있다. 지역화된 이름이 없을 경우, 빈 문자열이 반환된다. 이 필드는 보여주기만을 목적으로

하는 필드이다.

SQL_DESC_NAME[Implementation descriptor]

SQLCHAR*인 이 레코드 필드는 row 설명자 내에서 필드의 별칭을 포함한다. Column의 별칭이 적용되지 않을 경우, column 이름이 반환된다. 경우에 따라 드라이버가 SQL_DESC_NAME 필드를 설정할 때 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. Column의 이름 또는 column의 별칭이 없을 경우, 드라이버가 SQL_DESC_NAME 필드의 빈 문자열을 반환하고 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정한다.

응용 프로그램은 IPD의 SQL_DESC_NAME 필드를 매개 변수 이름이나 별칭으로 설정하여 stored procedure 매개 변수를 이름으로 명시할 수 있게 한다. IRD의 SQL_DESC_NAME 필드는 읽기전용 필드이다. 응용 프로그램이 이 필드를 설정하려 할 경우, SQLSTATE HY091 (invalid descriptor field identifier)를 반환한다.

IPD의 경우 드라이버가 명명된 매개 변수를 지원하지 않는다면 이 필드는 정의되지 않는다. 드라이버가 명명된 매개 변수를 지원하고 이것이 매개 변수를 설명할 수 있다면 매개 변수 이름은 이 필드에 반환된다.

SQL_DESC_NULLABLE[Implementation descriptors]

IRD의 경우, 읽기전용 SQLSMALLINT인 이 레코드 필드의 column이 NULL 값을 가질 수 있으면 SQL_NULLABLE이고 NULL 값을 가질 수 없으면 SQL_NO_NULLS이며 column이 NULL 값을 허용하는 지 여부를 알 수 없다면 SQL_NULLABLE_UNKNOWN이다. 이 필드는 결과 집합의 column을 위해 특별히 존재한다.

IPD의 경우, 이 필드는 동적 매개 변수가 항상 NULL 가능하고 응용 프로그램에 의해 설정될 수 없기 때문에 항상 SQL_NULLABLE로 설정한다.

SQL_DESC_NUM_PREC_RADIX[All]

SQLINTEGER인 이 필드는 SQL_DESC_TYPE 필드가 approximate numeric 데이터 형식일 때 2 값을 가진다. 왜냐하면 SQL_DESC_PRECISION 필드가 비트 수를 포함하기 때문이다. 이 필드는 SQL_DESC_TYPE 필드가 exact numeric 데이터 형식을 가지는 경우 SQL_DESC_PRECISION 필드가 소수점 자리의 수를 포함하기 때문에 10 값을 가진다. 이 필드는 숫자가 아닌 데이터 형식에는 0을 설정한다.

SQL_DESC_OCTET_LENGTH[All]

SQLLEN인 이 레코드 필드는 문자열이나 이진 데이터 형식의 바이트 단위 길이를 포함한다. 고정 길이의 문자 또는 이진 형식들의 경우 바이트 단위의 실제 길이이다. 가변 길이의 문자 또는 이진 데이터 형식들의 경우 바이트 단위의 최대 길이이다. 이 값은 구현 설명자들에 대해 NULL 종료 문자를 위한 공백을 포함하지 않고 응용 프로그램 설명자들에 대해서는 NULL 종료 문자를 위한 공백을 포함한다. 응용 프로그램 데이터의 경우 이 필드는 버퍼의 크기를 포함한다. APD들의 경우 이 필드는 출력 또는 입력/ 출력 매개 변수들에 대해서만 정의된다.

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

SQLLEN*인 이 레코드 필드는 동적 인자 (매개 변수 설명자) 또는 바인딩된 column 값 (row 설명자)의 바이트 단위 전체 길이를 포함하는 변수를 가리킨다.

APD의 경우 이 값은 문자열 및 이진을 제외한 모든 인자들에 대해 무시된다. 이 필드가 SQL_NTS일 경우, 동적 인자는 NULL로 종료되어야 한다. 실행시점에 데이터 매개 변수가 될 바인딩된 매개 변수라는 것을 나타내기 위해 응용 프로그램은 실행할 때 SQL_DATA_AT_EXEC 또는 SQL_LEN_DATA_AT_EXEC 매크로의 결과를 포함할 변수를 APD의 레코드에 있는 이 필드에 설정한다. 이런 필드가 하나 이상일 경우, SQL_DESC_DATA_PTR은 응용 프로그램이 요구하는 매개 변수를 결정하는데 도움이 되는 식별값으로 설정될 수 있다.

ARD의 OCTET_LENGTH_PTR 필드가 NULL 포인터일 경우, 드라이버는 column의 길이 정보를 반환하지 않는다. APD의 SQL_DESC_OCTET_LENGTH_PTR이 NULL 포인터일 경우, 드라이버는 문자열과 이진 값들이 NULL로 종료되었다고 간주한다. (이진값들은 NULL로 종료되지 않아야 하지만 데이터 끝감을 피하기 위해 길이를 부여해야 한다.)

이 필드가 가리키는 버퍼를 채울 SQLFetch 또는 SQLFetchScroll이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않는다. 이 필드는 지연된 필드다. 이 필드는 당장 사용되지는 않지만 추후 드라이버가 데이터의 octet 길이를 표시하거나 결정할 때 사용된다.

SQL_DESC_PARAMETER_TYPE[IPDs]

SQLSMALLINT인 이 레코드 필드는 입력 매개 변수에 대해 SQL_PARAM_INPUT으로, 입력/출력 매개 변수에 대해 SQL_PARAM_INPUT_OUTPUT으로, 출력 매개 변수에 대해 SQL_PARAM_OUTPUT으로, 스트림된 입력/출력 매개 변수에 대해 SQL_PARAM_INPUT_OUTPUT_STREAM으로, 스트림된 출력 매개 변수에 대해 SQL_PARAM_OUTPUT_STREAM으로 설정된다. 이것은 기본적으로 SQL_PARAM_INPUT으로 설정된다.

SQL_DESC_PRECISION[All]

SQLSMALLINT인 이 레코드 필드는 exact numeric 형식에 대해 유효 정수 개수를 포함하고 approximate numeric일 경우, mantissa (이진 정밀도)의 비트 수를 포함한다. 또는 SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_INTERVAL_SECOND 데이터 형식의 fractional 초 부분의 유효 정수의 개수를 포함한다. 이 필드는 다른 모든 데이터 형식에는 정의되지 않는다.

이 필드의 값은 ODBC 2.x에 정의된 "precision" 값과는 다르다.

SQL_DESC_ROWVER[Implementation descriptors]

SQLSMALLINT인 이 레코드 필드는 row가 갱신 (예를 들어 SQL server의 "timestamp") 되었을 때 DBMS에 의해 column이 자동으로 수정되는지 여부를 나타낸다. 이 레코드 필드의 값은 column이 row 버저닝 column이면 SQL_TRUE로, 그렇지 않다면 SQL_FALSE로 설정된다. 이 column 속성은 column의 자동 갱신 여부를 결정하기 위해 SQL_ROWVER의 IdentifierType 인자로 SQLSpecialColumns를 호출하는 것과 유사하다.

SQL_DESC_SCALE[All]

SQLSMALLINT인 이 레코드 필드는 decimal과 numeric 데이터 형식의 정의된 소수점 자릿수를 포함한다. 이 필드는 다른 모든 데이터 형식들에는 정의되지 않는다.

이 필드의 값은 ODBC 2.x에 정의된 "scale" 값과 다르다.

SQL_DESC_SCHEMA_NAME[IRDs]

읽기전용 SQLCHAR*인 이 레코드 필드는 column을 포함하는 기본 테이블의 스키마 이름을 포함한다. 반환 값은 column이 표현식이거나 view의 일부일 경우, 드라이버에 따라 다르다. 데이터 소스가 스키마를 지원하지 않거나 스키마 이름을 확인할 수 없을 경우, 이 변수는 빈 문자열을 포함한다.

SQL_DESC_SEARCHABLE[IRDs]

읽기 전용 SQLSMALLINT인 이 레코드 필드는 다음 값들 중 하나를 설정한다.

- Column이 WHERE 절에 사용될 수 없는 경우, SQL_PRED_NONE이다. (이것은 ODBC 2.X의 SQL_UNSEARCHABLE과 같다.)
- Column이 WHERE 절에서 오직 LIKE 조건에만 함께 사용될 수 있는 경우에는 SQL_PRED_CHAR이다.
- Column이 WHERE 절에서 LIKE를 제외한 모든 비교 연산자들과 함께 사용될 수 있는 경우에는, SQL_PRED

D_BASIC이다. (이것은 ODBC 2.x의 SQL_EXCEPT_LIKE 값과 같다.)

- Column이 WHERE 절의 어떤 비교 연산자들과도 사용될 수 있는 경우에는, SQL_PRED_SEARCHABLE이다.

SQL_DESC_TABLE_NAME[IRDS]

읽기전용 SQLCHAR*인 이 레코드 필드는 이 column을 포함하는 기본 테이블의 이름을 포함한다. Column이 표현식이거나 view의 일부일 경우, 반환값은 드라이버에 따라 다르다.

SQL_DESC_TYPE[All]

SQLSMALLINT인 이 레코드 필드는 datetime과 interval 데이터 형식들을 제외한 모든 데이터 형식들에 대해 축약된 SQL 또는 C 데이터 형식을 포함한다. 이 필드는 datetime과 interval 데이터 형식들에 대해 SQL_DATETIME이나 SQL_INTERVAL을 명시한다.

이 필드가 SQL_DATETIME이나 SQL_INTERVAL을 포함할 때마다, SQL_DESC_DATETIME_INTERVAL_CODE 필드는 반드시 함축 형식의 적절한 서브코드를 포함해야 한다. Datetime 데이터 형식들에 대해 SQL_DESC_TYPE은 SQL_DATETIME을 포함하고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 datetime 데이터 형식을 명시한 서브코드를 포함한다. Interval 데이터 형식들에 대해 SQL_DESC_TYPE은 SQL_INTERVAL을 포함하고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 interval 데이터 형식을 명시하는 서브코드를 포함한다.

SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE 필드의 값은 상호 의존적이다. 필드 중 하나를 설정하면 다른 필드도 설정해야 한다. SQL_DESC_TYPE은 SQLSetDescField나 SQLSetDescRec를 호출하여 설정할 수 있다. SQL_DESC_CONCISE_TYPE은 SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 설정할 수 있다.

SQL_DESC_TYPE이 interval 또는 datetime 데이터 형식이 아닌 함축된 데이터 형식으로 설정된 경우, SQL_DESC_CONCISE_TYPE 필드는 같은 값으로 설정하고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 0으로 설정한다.

SQL_DESC_TYPE이 datetime 또는 interval 데이터 형식의 긴 데이터 형식 (SQL_DATETIME 또는 SQL_INTERVAL)으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드를 적절한 서브코드로 설정한다면 SQL_DESC_CONCISE_TYPE 필드는 함축된 데이터 형식에 대응하는 값으로 설정된다. SQL_DESC_TYPE을 함축된 datetime 또는 interval 형식들 중 하나로 설정하면 SQLSTATE HY021 (Inconsistent descriptor information)을 반환한다.

SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 SQL_DESC_TYPE 필드를 설정할 때 다음 필드들은 다음 표의 기본값들로 설정한다. 같은 레코드의 남은 필드값들은 정의되지 않는다.

SQL_DESC_TYPE 값	다른 필드들은 암묵적으로 설정
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH는 1로 설정된다. SQL_DESC_PRECISION은 0으로 설정된다.
SQL_DATETIME	SQL_DESC_DATETIME_INTERVAL_CODE가 SQL_CODE_DATE 또는 SQL_CODE_TIME으로 설정될 때 SQL_DESC_PRECISION은 0으로 설정된다. SQL_DESC_TIMESTAMP로 설정될 때는 SQL_DESC_PRECISION이 6으로 설정된다.

SQL_DESC_TYPE 값	다른 필드들은 암묵적으로 설정
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE은 0으로 설정된다. SQL_DESC_PRECISION은 각 데이터 형식의 구현 정의된 정밀도를 설정한다.
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION은 SQL_FLOAT에 구현 정의된 기본 정밀도로 설정된다.
SQL_INTERVAL	SQL_DESC_DATETIME_INTERVAL_CODE를 interval 데이터 형식으로 설정할 때 SQL_DESC_DATETIME_INTERVAL_PRECISION은 2 (기본 interval leading precision)로 설정된다. Interval이 초 부분을 가지고 있을 때 SQL_DESC_PRECISION은 6 (기본 interval seconds precision)으로 설정된다.

응용 프로그램이 SQLSetDescRec가 아닌 SQLSetDescField를 호출하여 설명자 필드를 설정할 때 응용 프로그램은 반드시 데이터 형식을 가장 먼저 정의해야 한다. 이 때 이전 테이블에 표시된 다른 필드들은 암묵적으로 설정된다. 어떤 값도 암묵적으로 설정할 수 없다면 응용 프로그램은 명백한 설정을 위해 SQLSetDescField나 SQLSetDescRec를 호출할 수 있다.

SQL_DESC_TYPE_NAME[Implementation descriptors]

읽기전용 SQLCHAR*인 이 레코드 필드는 데이터 소스에 의존적인 형식 이름 ("CHAR", "VARCHAR" 등)을 포함한다. 데이터 형식 이름이 unknown이면 변수는 빈 문자열을 포함한다.

SQL_DESC_UNNAMED[Implementation descriptors]

SQL_DESC_NAME 필드를 설정하면, row 설명자에 존재하며 SQLSMALLINT인 이 레코드 필드는 드라이버에 의해 SQL_NAME이나 SQL_UNNAMED 중 하나로 설정된다. SQL_DESC_NAME 필드가 column의 별칭을 포함하거나 column의 별칭이 적용되지 않을 경우 드라이버가 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. 응용 프로그램이 IPD의 SQL_DESC_NAME 필드를 매개 변수 이름 또는 별칭으로 설정하면 드라이버는 IPD의 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. Column의 이름이나 별칭이 없을 경우, 드라이버는 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정한다.

응용 프로그램 IPD의 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정할 수 있다. 응용 프로그램이 IPD의 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정하려고 시도하면 드라이버는 SQLSTATE HY091 (invalid descriptor field identifier)를 반환한다. IRD의 SQL_DESC_UNNAMED 필드는 읽기전용이다. 응용 프로그램이 설정하려고 시도하면 SQLSTATE HY091 (Invalid descriptor field identifier)이 반환된다.

SQL_DESC_UNSIGNED[Implementation descriptors]

읽기전용 SQLSMALLINT인 이 레코드 필드는 column 형식이 unsigned거나 non-numeric일 경우 SQL_TRUE로 설정되고, column 형식이 signed일 경우, SQL_FALSE로 설정된다.

SQL_DESC_UPDATABLE[IRDs]

읽기전용 SQLSMALLINT인 이 레코드 필드는 다음 값들 중 하나로 설정된다.

- 결과 집합 column이 읽기전용이면 SQL_ATTR_READ_ONLY이다.
- 결과 집합 column에 읽고 쓸 수 있으면 SQL_ATTR_WRITE이다.
- 결과 집합 column에 대한 갱신가능 여부를 알 수 없으면 SQL_ATTR_READWRITE_UNKNOWN이다.

SQL_DESC_UPDATABLE 결과 집합 안에 있는 (기본 테이블에 있는 column이 아닌) column의 갱신가능성

을 설명한다. 이 결과 집합 column들이 기반으로 하고 있는 기본 테이블 안에 있는 column의 갱신가능성은 이 필드 안에 있는 값과 다를 수 있다. Column의 갱신여부는 데이터 형식, 사용자 권한 그리고 결과 집합 자체의 정의를 기반으로 한다. Column의 갱신여부가 불확실할 경우, SQL_ATTR_READWRITE_UNKNOWN이 반환되어야 한다.

SQL_DESC_CHAR_LENGTH_UNITS[Implementation descriptors]

SQLSMALLINT인 이 레코드 필드는 SQL 형식이 SQL_CHAR, SQL_VARCHAR나 SQL_LONGVARCHAR인 column의 길이 단위를 나타낸다.

- SQL_CLU_CHARACTERS: 길이 단위가 CHARACTER이다. 예를 들어 "문자열"이라는 데이터는 인코딩 방식이 UHC (Unified Hangul Code)일 경우 그 길이가 3이다.
- SQL_CLU_OCTETS: 길이 단위가 OCTETS이다. 예를 들어 "문자열"이라는 데이터는 인코딩 방식이 UHC일 경우, 그 길이가 6이다.
- SQL_CLU_NONE: 길이 단위가 정의되어 있지 않다. 위에 열거된 SQL 형식을 제외한 SQL 형식일 경우 반환되는 값이다.

SQLGetDescRec

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetDescRec는 설명자 레코드의 다수 필드에 대한 현재 설정 또는 값을 반환한다. 반환된 필드는 이름, 데이터 타입, column의 크기 또는 인자 데이터에 대해 설명한다.

구문

```
SQLRETURN SQLGetDescRec(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT    RecNumber,
    SQLCHAR *      Name,
    SQLSMALLINT    BufferLength,
    SQLSMALLINT *   StringLengthPtr,
    SQLSMALLINT *   TypePtr,
    SQLSMALLINT *   SubTypePtr,
    SQLLEN *        LengthPtr,
    SQLSMALLINT *   PrecisionPtr,
    SQLSMALLINT *   ScalePtr,
    SQLSMALLINT *   NullablePtr);
```

인자

DescriptorHandle

[입력] 설명자 핸들이다.

RecNumber

[입력] 응용 프로그램이 찾고자 하는 정보의 설명자 레코드이다. 설명자 레코드는 레코드 숫자 0을 북마크 레코드로 하여 1부터 번호가 정해진다. RecNumber가 SQL_DESC_COUNT보다 작거나 같지만, row에 column이나 매개 변수에 대한 데이터가 포함되지 않은 경우, SQLGetDescField는 필드의 기본값을 반환한다.

Name

[출력] 설명자 레코드를 위한 SQL_DESC_NAME을 반환하는 버퍼의 포인터이다. Name이 NULL이면 StringLengthPtr은 name의 버퍼 길이 (null 종료 문자를 포함한 길이)를 반환한다.

BufferLength

[입력] Name 버퍼 길이이다.

StringLengthPtr

[출력] *Name 버퍼에서 null 종료 문자를 제외하고 반환 가능한 문자의 개수를 반환할 포인터이다. 문자의 개수가 BufferLength보다 크거나 같으면 *Name의 데이터는 BufferLength에서 null 종료 문자 길이만큼 뺀 길이로 잘리고 null로 종료된다.

TypePtr

[출력] 설명자 레코드를 위한 SQL_DESC_TYPE 필드의 값을 반환할 버퍼의 포인터이다.

SubTypePtr

[출력] SQL_DATETIME 또는 SQL_INTERVAL 타입을 갖는 레코드를 위한 SQL_DESC_DATETIME_INTERVAL_CODE 필드의 값을 반환할 버퍼의 포인터이다.

LengthPtr

[출력] 설명자 레코드를 위한 SQL_DESC_OCTET_LENGTH 필드의 값을 반환할 버퍼의 포인터이다.

PrecisionPtr

[출력] 설명자 레코드를 위한 SQL_DESC_PRECISION 필드의 값을 반환할 포인터이다.

ScalePtr

[출력] 설명자 레코드를 위한 SQL_DESC_SCALE 필드의 값을 반환할 포인터이다.

NullablePtr

[출력] 설명자 레코드를 위한 SQL_DESC_NULLABLE 필드의 값을 반환할 포인터이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE

RecNumber가 현재 설명자 레코드의 수보다 큰 경우, SQL_NO_DATA가 반환된다.

DescriptorHandle이 IRD 핸들이고 명령문이 준비 또는 실행 상태에 있지만 관련된 커서가 없을 경우, SQL_NO_DATA를 반환한다.

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	*ValuePtr의 버퍼 길이가 설명자 필드의 값의 길이보다 작아서 잘렸다. 남아 있는 설명자 필드의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
		fieldIdentifier 인자가 레코드 필드이고, RecNumber 인자값이 0으로 설정되었으며 DescriptorHandle 인자가 IPD 핸들이었다. RecNumber 인자가 0으로 설정되고, SQL_ATTR_USE_BOOKMARKS

SQLSTATE	Error	설명
07009	Invalid descriptor index	상태 속성이 SQL_UB_OFF로 설정되었으며, DescriptorHandle 인자가 I RD 핸들이었다. RecNumber 인자가 0보다 작았다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY007	Associated statement is not prepared	DescriptorHandle이 IRD 핸들과 연관되어 있고 관련된 명령문 핸들이 준비나 실행 단계에 있지 않다.
HY010	Function sequence error	DescriptorHandle과 관련된 StatementHandle이 함수를 비동기적으로 수행하고, 호출되었을 때도 여전히 이 함수가 실행되고 있었다. SQLExecute, SQLExecDirect를 호출한 후에 SQL_NEED_DATA가 반환되었으며, 모든 data-at-execution 변수를 보내기 전에 이 함수를 호출했다. DescriptorHandle과 관련된 연결 핸들에 대한 함수가 호출되었는데, SQLGetDescRec이 호출되었을 때 이 함수가 여전히 수행 중이었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

응용 프로그램은 단일 column 또는 매개 변수에 대해 다음과 같은 설명자 필드의 값을 검색하기 위해 SQLGetDescRec를 호출할 수 있다.

- SQL_DESC_NAME
- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_NULLABLE

SQLGetDescRec는 헤더 필드의 값을 검색하지 않는다.

응용 프로그램은 null 포인터 필드에 대응하는 인자를 설정하여 필드 설정에 대한 반환을 막을 수 있다.

응용 프로그램이 SQLGetDescRec를 호출하여 특정 설명자 타입에 대해 정의되지 않은 필드값을 검색하였을 때, 함수는 SQL_SUCCESS를 반환하지만 필드를 위한 반환값은 정의하지 않는다. 예를 들어, APD 또는 ARD의 SQL_DESC_NAME이나 SQL_DESC_NULLABLE 필드에 대해 SQLGetDescRec를 호출할 경우, SQL_SUCCESS가 반환되지만 필드값은 정의되지 않는다.

응용 프로그램이 특정 설명자 타입으로 정의되었지만 기본값으로 설정되지 않은 필드의 값을 검색하기 위해 SQLGetDescRec를 호출하였을 때, 함수는 SQL_SUCCESS를 반환하지만 필드값은 정의하지 않는다. 자세한 내용은 SQLSetDescField의 **설명자 필드의 초기화**를 참조한다.

SQLGetDescField를 호출하면 필드값들을 각각 별도로 검색할 수 있다. 설명자 헤더 또는 레코드 안의 필드에 대한 자세한 내용은 **SQLSetDescField**를 참조한다.

SQLGetDiagField

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetDiagField는 에러, 경고, 상태 정보를 포함하는 진단 데이터 구조에서 레코드 필드의 현재 값을 반환한다.

구문

```
SQLRETURN SQLGetDiagField(
    SQLSMALLINT      HandleType,
    SQLHANDLE         Handle,
    SQLSMALLINT      RecNumber,
    SQLSMALLINT      DiagIdentifier,
    SQLPOINTER        DiagInfoPtr,
    SQLSMALLINT      BufferLength,
    SQLSMALLINT *     StringLengthPtr);
```

인자

HandleType

[입력] 진단이 필요한 핸들 타입에 대한 식별자이다. 식별자는 다음 중 하나이어야 한다.

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

[입력] HandleType이 가리키는 타입의 진단 데이터 구조를 위한 핸들이다. HandleType이 SQL_HANDLE_ENV이면, handle은 공유 또는 비공유 환경 핸들일 수 있다.

RecNumber

[입력] 응용 프로그램이 찾는 정보의 상태 레코드를 나타낸다. 상태 레코드 번호는 1부터 시작한다. DiagIdentifier 인자가 진단 헤더의 필드를 가리키면, RecNumber는 무시된다. 그렇지 않으면 RecNumber는 0보다 커야 한다.

DiagIdentifier

[입력] 값이 반환되는 진단 필드를 가리킨다. 자세한 내용은 설명 섹션의 **DiagIdentifier 인자**를 참조한다.

DiagInfoPtr

[출력] 진단 정보가 반환되는 버퍼의 포인터이다. 데이터 타입은 DiagIdentifier의 값에 좌우된다. 일부 드라이버는 버퍼의 32 비트 또는 16 비트의 하위 비트만 기록하고 상위 비트는 그대로 남겨두기 때문에 DiagInfoPtr이 정수 타입일 경우, 응용 프로그램은 SQLULEN의 버퍼를 사용해야하고 함수를 호출하기 전에 값을 0으로 초기화해야한다. DiagInfoPtr이 NULL이면, StringLengthPtr은 DiagInfoPtr이 가리키는 버퍼에서 null 종료 문자를 제외한 바이트의 총 숫자를 반환한다.

BufferLength

[입력] DiagIdentifier가 ODBC 정의된 진단이고 DiagInfoPtr이 문자열이나 바이너리 버퍼를 가리키면, 이 인자는 *DiagInfoPtr의 길이어야 한다. DiagIdentifier가 ODBC 정의된 진단 필드이고 *DiagInfoPtr이 숫자일 경우, BufferLength는 무시된다. (SQLGetDiagFieldW를 호출할 때) *DiagInfoPtr의 값이 unicode 문자열이면 BufferLength 인자는 반드시 짝수이어야 한다. DiagIdentifier가 드라이버가 정의한 필드일 경우, 응용 프로그램이 BufferLength 인자를 설정하여 드라이버 관리자의 필드 특성을 가리킨다. BufferLength는 다음과 같은 값을 가질 수 있다.

- DiagInfoPtr이 문자열 포인터일 경우, BufferLength는 문자열의 길이 또는 SQL_NTS이다.
- DiagInfoPtr이 바이너리 버퍼의 포인터일 경우, 응용 프로그램은 SQL_LENG_BINARY_ATTR (length) 매크로의 결과를 BufferLength에 저장한다. 음수는 BufferLength에 저장된다.
- DiagInfoPtr이 문자열이나 바이너리 버퍼의 포인터가 아닐 경우, BufferLength는 SQL_IS_POINTER 값을 가져야 한다.
- *DiagInfoPtr이 고정 길이 데이터 타입일 경우, BufferLength는 SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, 또는 SQL_IS_USMALLINT 중에 적절한 하나이다.

StringLengthPtr

[출력] 문자 데이터에 대해 *DiagInfoPtr에 반환되는 null 종료문자를 제외한 바이트의 총 길이를 반환할 버퍼의 포인터이다. 반환에 사용되는 바이트의 길이가 BufferLength 보다 크거나 같으면, *DiagInfoPtr 안의 텍스트는 BufferLength에서 null 종료 문자를 뺀 만큼 잘린다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_NO_DATA

진단

SQLGetDiagField는 진단 레코드를 게시하지 않는다. 다음과 같은 반환값의 결과를 사용한다.

- SQL_SUCCESS: 함수가 성공적으로 진단 정보를 반환했다.
- SQL_SUCCESS_WITH_INFO: *DiagInfoPtr이 요청된 진단 필드의 반환값을 저장하기에 작아서 잘렸다. 잘렸는지 여부를 확인하려면 응용 프로그램이 BufferLength를 *StringLengthPtr에 기록된 실제 바이트 길이와 비교해야 한다.

- SQL_INVALID_HANDLE: Handle이 HandleType이 가리키는 타입에 대해 유효하지 않다.
- SQL_ERROR: 다음 중 하나가 발생했다.
 - DiagIdentifier 인자가 유효한 값이 아니다.
 - DiagIdentifier 인자가 SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FUNCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, SQL_DIAG_ROW_COUNT 중 하나였고, handle이 명령문 핸들이 아니었다.
 - DiagIdentifier가 진단 레코드의 필드를 가리킬 때 RecNumber 인자가 음수 또는 0이다. RecNumber는 헤더 필드에 대해서는 무시된다.
 - 요청된 값이 문자열이고 BufferLength가 0보다 작다.
 - 비동기 알림을 사용할 경우, handle의 비동기 수행이 완료되지 않았다.
- SQL_NO_DATA: RecNumber가 handle에 대해 지정된 진단 레코드의 수보다 크다. Handle에 대한 진단 레코드가 없는 경우, 모든 양수의 RecNumber에 대해 함수가 SQL_NO_DATA를 반환한다.

설명

응용 프로그램은 일반적으로 다음 세 가지 중 하나를 목적으로 SQLGetDiagField를 호출한다.

- 함수가 SQL_ERROR나 SQL_SUCCESS_WITH_INFO를 반환하였을 때 구체적인 에러나 경고 정보를 얻으려는 경우 (SQLBrowseConnect가 SQL_NEED_DATA를 반환한다.)
- SQLExecute, SQLExecDirect, SQLBulkOperations, SQLSetPos를 사용한 INSERT, DELETE 또는 UPDATE 명령을 수행할 때 데이터 소스에서 row의 개수를 얻으려는 경우, 또는 드라이버가 제공할 수 있을 때 현재 열린 커서에 존재하는 row의 개수를 얻으려는 경우
- 어떤 함수가 SQLExecDirect나 SQLExecute를 호출하여 수행하는지에 대한 정보를 얻으려는 경우

이 함수가 호출될 때마다 모든 ODBC 함수가 0 이상의 진단 레코드를 게시할 수 있다. 따라서 응용 프로그램은 모든 함수를 호출한 후에 SQLGetDiagField를 호출할 수 있다. 한 번에 저장할 수 있는 진단 레코드의 개수에는 제한이 없다. SQLGetDiagField는 가장 최근에 handle 인자에 지정된 진단 데이터 구조와 관련된 진단 정보를 검색한다. 응용 프로그램이 SQLGetDiagField나 SQLGetDiagRec 대신 다른 ODBC 함수를 호출하면 같은 핸들이 이전에 호출하여 얻은 진단 정보를 잃을 수 있다.

SQLGetDiagField가 SQL_SUCCESS를 반환하고 있는 한, 응용 프로그램은 RecNumber를 증가시키면서 진단 레코드를 읽을 수 있다. 상태 레코드의 수는 SQL_DIAG_NUMBER 헤더 필드에 표시된다. SQLGetDiagField 호출은 헤더와 레코드 필드에 영향을 미치지 않는다. 진단 함수 이외의 함수를 도중에 호출하지 않는 한, 응용 프로그램은 같은 핸들을 이용하여 SQLGetDiagField를 재호출하여 레코드의 필드를 검색할 수 있다.

응용 프로그램은 언제든지 모든 진단 필드를 반환하기 위해 SQLGetDiagField를 호출할 수 있는데 SQL_DIAG_CURSOR_ROW_COUNT나 SQL_DIAG_ROW_COUNT의 경우, handle이 명령문 핸들이 아닌 경우 SQL_ERROR를 반환하기 때문에 제외된다. 만약 진단 필드가 하나라도 정의되지 않을 경우, SQLGetDiagField 호출은 SQL_SUCCESS와 설정되지 않은 값을 반환한다.

비동기적으로 실행되는 함수 이외의 API 호출은 HY010 (function sequence error)을 만든다. 그러나 에러 레코드는 비동기 수행이 완료되기 전에 검색될 수 없다.

HandleType 인자

각각의 핸들 타입은 관련된 진단 정보를 가지며 HandleType 인자는 handle의 타입을 가리킨다.

일부 헤더와 레코드 필드는 환경, 연결, 명령문, 설명자 핸들에 대해 반환되지 않는다. 아래 헤더 필드와 레코드 필드에서 필드에 적절하지 못한 핸들에 대해 설명한다.

HandleType이 SQL_HANDLE_ENV일 경우, handle은 공유 또는 비공유 환경 핸들일 수 있다.

드라이버의 특정 헤더 진단 필드는 환경 핸들과 관련이 없다.

설명자 핸들에 대해 정의된 진단 헤더는 SQL_DIAG_NUMBER와 SQL_DIAG_RETURNCODE 뿐이다.

DiagIdentifier 인자

이 인자는 진단 데이터 구조에 필요한 필드의 식별자이다. RecNumber가 1과 같거나 크면, 필드에 있는 데이터는 함수에서 반환되는 진단 정보이다. RecNumber가 0이면, 필드는 진단 데이터 구조의 헤더에 있고 구체적인 정보가 아닌 진단 정보를 반환하는 함수 호출과 관련된 데이터를 포함한다.

드라이버는 진단 데이터 구조에서 드라이버별 헤더와 레코드 필드를 정의할 수 있다.

ODBC 2.x 드라이버를 이용하는 ODBC 3.x 응용 프로그램은 DiagIdentifier 인자가 SQL_DIAG_CLASS_ORIGIN, SQL_DIAG_CLASS_SUBCLASS_ORIGIN, SQL_DIAG_CONNECTION_NAME, SQL_DIAG_MESSAGE_TEXT, SQL_DIAG_NATIVE, SQL_DIAG_NUMBER, SQL_DIAG_RETURNCODE, SQL_DIAG_SERVER_NAME, SQL_DIAG_SQLSTATE일 때 SQLGetDiagField를 호출할 수 있고 다른 진단 필드에 대해서는 SQL_ERROR를 반환한다.

헤더 필드

DiagIdentifier	반환 타입	반환
SQL_DIAG_CURSOR_ROW_COUNT	SQLLEN	이 필드는 커서에 있는 row 개수를 포함한다. 이것의 의미는 SQLGetInfo 정보 타입에 좌우된다. 정보 타입에는 SQL_DYNAMIC_CURSOR_ATTRIBUTES2, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2, SQL_KEYSET_CURSOR_ATTRIBUTES2, SQL_STATIC_CURSOR_ATTRIBUTES2가 있고, 각 커서 타입 (SQL_CA2_CRC_EXACT과 SQL_CA2_CRC_APPROXIMATE 비트에 있다)에 사용할 수 있는 row의 개수를 나타낸다. 이 필드의 내용은 명령문 핸들, SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출된 이후에만 정의된다. DiagIdentifier가 SQL_DIAG_CURSOR_ROW_COUNT 이외의 명령문 핸들이면 SQLGetDiagField는 SQL_ERROR를 반환한다.
SQL_DIAG_DYNAMIC_FUNCTION	SQLCHAR *	기본적인 함수를 실행하는 SQL 명령문을 설명하는 문자열이다. 이 필드 내용은 SQLExecute, SQLExecDirect, 또는 SQLMoreResults를 호출한 후에 정의된다. DiagIdentifier가 SQL_DIAG_DYNAMIC_FUNCTION 이외의 명령문 핸들일 때 SQLGetDiagField를 호출하면 SQL_ERROR가 반환된다.
SQL_DIAG_DYNAMIC_FUNCTION_CODE	SQLINTEGER	기본적인 함수를 실행하는 SQL 명령문을 설명하는 숫자 코드이다. 이 필드 내용은 SQLExecute, SQLExecDirect 또는 SQLMoreResults를 호출한 후에 정의된다. DiagIdentifier가 SQL_DIAG_DYNAMIC_FUNCTION_CODE 이외의 명령문 핸들일 때 SQLGetDiagField를 호출하면 SQL_ERROR가 반환된다.
SQL_DIAG_NUM	SQLINTEGER	지정된 핸들의 사용 가능한 상태 레코드의 개수이다.

DiagIdentifier	반환 타입	반환
BER		
SQL_DIAG_RETURNCODE	SQLRETURN	함수에서 반환되는 코드이다. 드라이버는 SQL_DIAG_RETURNCODE를 구현할 필요 없고 드라이버 관리자가 이를 구현한다. 어떠한 함수도 handle을 호출하지 않았을 경우, SQL_DIAG_RETURNCODE에 대해 SQL_SUCCESS가 반환된다.
SQL_DIAG_ROW_COUNT	SQLLEN	SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos에 의해 수행된 INSERT, DELETE, UPDATE가 영향을 미친 row의 개수이다. Cursor specification이 수행된 이후 드라이버에 의해 정의된다. 이 필드의 내용은 오직 명령문 핸들을 위해 정의된다. DiagIdentifier가 SQL_DIAG_ROW_COUNT 이외의 명령문 핸들일 때 SQLGetDiagField를 호출하면 SQL_ERROR가 반환된다. 또한 이 필드의 데이터는 SQLRowCount의 RowCountPtr에도 반환된다. 이 필드의 데이터 중 SQLRowCount에 반환되는 row 개수는 명령문이 다시 준비 또는 할당 상태로 설정될 때까지 동일하게 유지되다가 비진단 함수가 호출된 후에 재설정된다.

레코드 필드

DiagIdentifier	반환 타입	반환
SQL_DIAG_CLASS_ORIGIN	SQLCHAR *	이 레코드에 있는 SQLSTATE 값의 클래스 부분을 정의하는 문서를 나타내는 문자열이다. 이 값은 오픈 그룹과 ISO 호출 레벨 인터페이스가 정의한 모든 SQLSTATE에 대해 ISO 9075이다. 지정된 (SQLSTATE 클래스가 IM을 갖는 모든) ODBC SQLSTATES에 대하여 이 값은 ODBC 3.0이다.
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	SQL_DIAG_ROW_NUMBER가 row 집합 또는 매개 변수 집합에서 유효한 row 번호일 경우, 이 필드는 결과 집합의 column 번호 또는 매개 변수 집합의 매개 변수 번호를 나타내는 값이다. 결과 집합 column 번호는 항상 1부터 시작한다. 이 상태 레코드가 북마크 column과 관련이 있을 경우, 이 필드는 0이 될 수 있다. 매개 변수 번호는 1부터 시작한다. 상태 레코드가 column 번호 및 매개 변수 번호와 관련이 없을 경우, 그 값은 SQL_NO_COLUMN_NUMBER이다. 만약 드라이버가 레코드와 관련된 column 번호나 매개 변수 번호를 결정할 수 없다면 이 필드값은 SQL_COLUMN_NUMBER_UNKNOWN이다. 이 필드의 내용은 명령문 핸들에 대해서만 정의된다.
SQL_DIAG_CONNECTION_NAME	SQLCHAR *	진단 레코드와 관련된 연결의 이름을 나타내는 문자열이다. 이 필드는 드라이버 정의이다. 진단 데이터 구조와 관련된 환경 핸들과 서버 사이에 어떤 관련도 없는 진단을 위해 이 필드는 길이가 0인 문자열이 된다.
SQL_DIAG_MESSAGE_TEXT	SQLCHAR *	에러 또는 경고에 대한 정보 메시지이다.
SQL_DIAG_NATIVE	SQLINTEGER	드라이버/ 데이터 소스의 구체적인 원시 에러 코드이다. 원시 에러 코드가 없으면 드라이버는 0을 반환한다.

DiagIdentifier	반환 타입	반환
SQL_DIAG_ROW_NUMBER	SQLLEN	이 필드에는 상태 레코드와 관련된 row 집합의 row 번호 또는 매개 변수 집합의 매개 변수 번호가 포함된다. Row 번호와 매개 변수 번호는 1부터 시작한다. 상태 레코드가 row 번호 또는 매개 변수 번호와 관련이 없으면 이 필드값은 SQL_NO_ROW_NUMBER이다. 드라이버가 이 레코드와 관련된 row 번호 또는 매개 변수 번호를 결정할 수 없다면 이 필드값은 SQL_ROW_NUMBER_UNKNOWN이다. 이 필드의 내용은 명령문 핸들에 대해서만 정의된다.
SQL_DIAG_SERVER_NAME	SQLCHAR *	진단 레코드와 관련된 서버 이름을 나타내는 문자열이다. 이는 SQL_DATA_SOURCE_NAME 옵션으로 SQLGetInfo를 호출하여 반환된 값과 동일하다. 진단 데이터 구조와 관련된 환경 핸들과 서버 사이에 어떤 관련도 없는 진단을 위해서 이 필드는 길이가 0인 문자열이 된다.
SQL_DIAG_SQLSTATE	SQLCHAR *	다섯 문자의 SQLSTATE 진단 코드이다.
SQL_DIAG_SUBCLASS_ORIGIN	SQLCHAR *	SQLSTATE 코드의 하위 클래스 부분을 정의하는 식별인 SQL_DIAG_CLASS_ORIGIN과 동일한 포맷인 유효한 값의 문자열이다. ODBC 3.0에서 ODBC 특정 SQLSTATE 코드는 다음과 같다. 01S00, 01S01, 01S02, 01S06, 01S07, 07S01, 08S01, 21S01, 21S02, 25S01, 25S02, 25S03, 42S01, 42S02, 42S11, 42S12, 42S21, 42S22, HY095, HY097, HY098, HY099, HY100, HY101, HY105, HY107, HY109, HY110, HY111, HYT00, HYT01, IM001, IM002, IM003, IM004, IM005, IM006, IM007, IM008, IM010, IM011, IM012

동적 함수 필드의 값

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
alter-domain-statement	"ALTER DOMAIN"	SQL_DIAG_ALTER_DOMAIN
alter-table-statement	"ALTER TABLE"	SQL_DIAG_ALTER_TABLE
assertion-definition	"CREATE ASSERTION"	SQL_DIAG_CREATE_ASSERTION
character-set-definition	"CREATE CHARACTER SET"	SQL_DIAG_CREATE_CHARACTER_SET
collation-definition	"CREATE COLLATION"	SQL_DIAG_CREATE_COLLATION
create-index-statement	"CREATE INDEX"	SQL_DIAG_CREATE_INDEX
create-table-statement	"CREATE TABLE"	SQL_DIAG_CREATE_TABLE
create-view-statement	"CREATE VIEW"	SQL_DIAG_CREATE_VIEW
cursor-specification	"SELECT CURSOR"	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	"DYNAMIC DELETE CURSOR"	SQL_DIAG_DYNAMIC_DELETE_CURSOR
delete-statement-searched	"DELETE WHERE"	SQL_DIAG_DELETE_WHERE
domain-definition	"CREATE DOMAIN"	SQL_DIAG_CREATE_DOMAIN
drop-assertion-statement	"DROP ASSERTION"	SQL_DIAG_DROP_ASSERTION
drop-character-set-stmt	"DROP CHARACTER SET"	SQL_DIAG_DROP_CHARACTER_SET

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
drop-collation-statement	"DROP COLLATION"	SQL_DIAG_DROP_COLLATION
drop-domain-statement	"DROP DOMAIN"	SQL_DIAG_DROP_DOMAIN
drop-index-statement	"DROP INDEX"	SQL_DIAG_DROP_INDEX
drop-schema-statement	"DROP SCHEMA"	SQL_DIAG_DROP_SCHEMA
drop-table-statement	"DROP TABLE"	SQL_DIAG_DROP_TABLE
drop-translation-statement	"DROP TRANSLATION"	SQL_DIAG_DROP_TRANSLATION
drop-view-statement	"DROP VIEW"	SQL_DIAG_DROP_VIEW
grant-statement	"GRANT"	SQL_DIAG_GRANT
insert-statement	"INSERT"	SQL_DIAG_INSERT
ODBC-procedure-extension	"CALL"	SQL_DIAG_CALL
revoke-statement	"REVOKE"	SQL_DIAG_REVOKE
schema-definition	"CREATE SCHEMA"	SQL_DIAG_CREATE_SCHEMA
translation-definition	"CREATE TRANSLATION"	SQL_DIAG_CREATE_TRANSLATION
update-statement-positioned	"DYNAMIC UPDATE CURSOR"	SQL_DIAG_DYNAMIC_UPDATE_CURSOR
update-statement-searched	"UPDATE WHERE"	SQL_DIAG_UPDATE_WHERE
Unknown	empty string	SQL_DIAG_UNKNOWN_STATEMENT

상태 레코드의 순차

상태 레코드는 row 번호와 진단 종류에 따라 순차적으로 위치한다. 드라이버 관리자는 생성되는 상태 레코드를 반환하는 마지막 순서를 결정한다.

드라이버 관리자와 드라이버가 진단 레코드를 게시하면 드라이버 관리자는 진단 레코드의 순서를 결정해야 한다.

둘 이상의 상태 레코드가 있을 경우, 레코드의 순서는 row 번호에 의해 가장 먼저 결정된다. Row가 진단 레코드의 순서를 결정할 때 다음과 같은 규칙이 적용된다.

- SQL_NO_ROW_NUMBER가 -1로 정의되어 어떤 row에도 대응되지 않는 레코드는 특정 row에 대응되는 레코드의 앞에 위치한다.
- SQL_ROW_NUMBER_UNKNOWN가 -2로 정의되어 row 번호를 알 수 없는 레코드는 다른 모든 레코드의 앞에 위치한다.
- 특정 row와 관련된 모든 레코드는 SQL_DIAG_ROW_NUMBER 필드의 값으로 분류된다. 첫 번째 row의 모든 에러와 경고가 나열된 후 순차적으로 다음 row의 모든 에러와 경고가 나열된다.

노트

ODBC 2.x 드라이버에서 SQLSTATE 01S01을 반환 또는 ODBC 3.x 드라이버에서 SQLExtendedFetch를 호출하거나 SQLExtendedFetch에 위치한 커서에 SQLSetPos를 호출하였을 때 SQLSTATE 01S01을 반환

할 경우, ODBC 3.x 드라이버 관리자는 진단 queue에 상태 레코드를 요청하지 않는다.

각 row 내에서, 또는 row와 대응하지 않거나 row 번호를 알 수 없는 모든 레코드에 대해서, 또는 SQL_NO_ROW_NUMBER와 같은 row 번호를 갖는 모든 레코드에 대해서, 처음 나열되는 레코드는 정렬 규칙의 집합을 사용하여 결정된다. 첫 레코드 이후, row에 영향을 미치는 다른 레코드의 순서는 정의되지 않는다. 응용 프로그램은 첫 레코드 이후, 에러가 경고 앞에 있다고 가정할 수 없다. 응용 프로그램은 실패한 함수 호출에 대한 최대한의 정보를 얻기 위해 완전한 진단 데이터 구조를 검사해야 한다.

Row 내의 첫 번째 레코드를 결정하는 데는 다음과 같은 규칙이 사용된다. 가장 높은 등급의 레코드가 첫 번째 레코드이다. 레코드의 소스 (드라이버 관리자, 드라이버, 게이트웨이 등)는 레코드 순위 결정에 영향을 미치지 않는다.

- 에러를 기술하는 errors 상태 레코드는 제일 높은 등급이다. 에러를 정렬하는 데는 다음 규칙이 적용된다.
 - 트랜잭션 실패 또는 발생가능한 트랜잭션 실패를 가리키는 레코드는 다른 레코드들보다 등급이 높다.
 - 동일한 에러 상태를 기술하는 둘 이상의 레코드의 경우, 오픈 그룹 CLI 사양 (클래스 03~HZ)에 정의된 SQLSTATE는 ODBC와 드라이버 정의된 SQLSTATE 보다 등급이 높다.
- 데이터 값이 없는 (클래스 02) 드라이버가 정의한 Implementation-defined No Data Values 상태 레코드는 두 번째 등급이다.
- 경고 (클래스 01)를 기술하는 warnings 상태 레코드는 제일 낮은 등급이다. 동일한 에러 상태를 기술하는 둘 이상의 레코드에서 오픈 그룹 CLI 사양에 정의된 SQLSTATE는 ODBC와 드라이버 정의된 SQLSTATE 보다 등급이 높다.

SQLGetDiagRec

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetDiagRec는 에러, 경고, 상태 정보가 포함되는 (지정된 핸들과 관련된) 진단 데이터 소스의 레코드 필드의 현재 값을 반환한다.

구문

```
SQLRETURN SQLGetDiagRec(
    SQLSMALLINT      HandleType,
    SQLHANDLE         Handle,
    SQLSMALLINT      RecNumber,
    SQLCHAR *         SQLState,
    SQLINTEGER *      NativeErrorPtr,
    SQLCHAR *         MessageText,
    SQLSMALLINT      BufferLength,
    SQLSMALLINT *     TextLengthPtr);
```

인자

HandleType

[입력] 진단이 요구되는 핸들의 타입 식별자이다. 식별자는 다음 중 하나이어야 한다.

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

[입력] HandleType이 가리키는 타입의 진단 데이터 구조를 위한 핸들이다. HandleType이 SQL_HANDLE_ENV이면 handle은 공유 또는 비공유 환경 핸들일 수 있다.

RecNumber

[입력] 응용 프로그램이 찾는 정보에서 나온 상태 레코드를 나타낸다. 상태 레코드 번호는 1부터 시작한다.

SQLState

[출력] 진단 레코드 RecNumber에 대해 다섯 문자의 SQLSTATE 코드를 반환하는 버퍼 포인터이다. 처음 두 문자는 클래스, 다음 세 문자는 하위 클래스를 나타낸다. 이 정보는 SQL_DIAG_SQLSTATE 진단 필드에 들어 있다.

NativeErrorPtr

[출력] 데이터 소스에 구체적인 원시 에러 코드를 반환하는 버퍼 포인터이다. 이 정보는 SQL_DIAG_NATIVE 진단 필드에 들어 있다.

MessageText

[출력] 진단 메시지 텍스트 문자열이 반환되는 버퍼의 포인터이다. 이 정보는 SQL_DIAG_MESSAGE_TEXT 진단 필드에 들어 있다.

MessageText가 NULL이면, TextLegnthPtr은 MessageText가 가리키는 버퍼에 반환될 수 있는 문자(null 종료 문자 제외)의 총 개수를 반환한다.

BufferLength

[입력] *MessageText 버퍼 안의 문자 길이이다. 진단 메시지 텍스트의 최대 길이는 없다.

TextLegnthPtr

[출력] *MessageText에 반환할 수 있는 문자 (null 종료 문자 제외)의 총 개수를 반환하는 버퍼의 포인터이다. 반환되는 문자의 수가 BufferLength보다 클 경우, *MessageText의 진단 메시지 텍스트는 BufferLength에서 null 종료 문자를 뺀 만큼 잘린다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLGetDiagRec는 진단 레코드를 게시하지 않는다. 다음과 같은 반환값 결과를 사용한다.

- SQL_SUCCESS: 함수가 성공적으로 진단 정보를 반환했다.
- SQL_SUCCESS_WITH_INFO: *MessageText 버퍼가 요청된 진단 메시지를 담기에 작아서 진단 레코드가 생성되지 않았다. 잘렸는지 여부를 확인하려면 응용 프로그램이 BufferLength를 *StringLengthPtr에 기록된 실제 바이트 길이와 비교해야 한다.
- SQL_INVALID_HANDLE: Handle이 HandleType이 가리키는 타입에 대해 유효하지 않다.
- SQL_ERROR: 다음 중 하나가 발생했다.
 - RecNumber 인자가 음수 또는 0이었다.
 - BufferLength가 0보다 작았다.
 - 비동기 알림을 사용할 경우, 핸들의 비동기 수행이 완료되지 않았다.
- SQL_NO_DATA: RecNumber가 handle을 위해 지정된 진단 레코드의 수보다 크다. Handle을 위한 진단 레코드가 없는 경우, 모든 양수의 RecNumber에 대해 함수가 SQL_NO_DATA를 반환한다.

설명

응용 프로그램은 일반적으로 ODBC 함수가 SQL_ERROR나 SQL_SUCCESS_WITH_INFO를 반환했을 때 SQLGetDiagRec를 호출한다. 그러나 모든 ODBC 함수가 0 이상의 진단 레코드를 게시할 수 있기 때문에 응용 프로그램은 함수를 호출한 이후에 SQLGetDiagRec를 호출할 수 있다. 응용 프로그램은 진단 데이터 구조안의 레코드 일부 또는 모두를 반환하기 위해 SQLGetDiagRec를 여러 번 호출할 수 있다. ODBC는 한 번에 저장할 수 있는 진단 레코드의 수를 제한하지 않는다.

SQLGetDiagRec는 진단 데이터 구조의 헤더 필드를 반환하는데 사용할 수 없다. (RecNumber 인자는 반드시 0보다 커야 한다.) 응용 프로그램은 이에 대해 SQLGetDiagRec를 대신하여 SQLGetDiagField를 호출해야 한다.

SQLGetDiagRec는 handle 인자에 지정된 핸들과 가장 최신의 연관 진단 정보만 검색한다. 응용 프로그램이 다른 ODBC 함수를 호출할 경우, 같은 핸들에서 (SQLGetDiagRec, SQLGetDiagField, SQLError를 제외한) 이전 호출의 진단 정보는 손실된다.

응용 프로그램은 SQLGetDiagRec가 SQL_SUCCESS를 반환하는 동안 RecNumber를 증가시키면서 진단 레코드를 반복하여 검색할 수 있다. SQLGetDiagRec 호출은 헤더와 레코드 필드에 영향을 미치지 않는다. 응용 프로그램은 SQLGetDiagRec, SQLGetDiagField, SQLError 함수를 제외한 다른 함수의 호출에 중간 개입이 없다면, 레코드에서 필드를 검색하기 위해 나중이라도 SQLGetDiagRec를 다시 호출할 수 있다. 응용 프로그램은 SQL_DIAG_NUMBER 필드의 값을 검색하기 위해 SQLGetDiagField를 호출할 수 있고 SQLGetDiagRec를 여러 번 호출하여 사용 가능한 진단 레코드의 전체 개수를 검색할 수도 있다.

HandleType 인자

각각의 핸들 타입은 관련된 진단 정보를 가지며 HandleType 인자는 handle 타입을 가리킨다.

일부 헤더와 레코드 필드는 환경, 연결, 명령문, 설명자 핸들에 대해 반환되지 않는다. SQLGetDiagField의 헤더 필드와 레코드 필드 표에서 필드에 적절하지 못한 핸들에 대해 설명한다.

HandleType이 공유 환경 핸들을 나타내는 SQL_HANDLE_ENV이면 SQLGetDiagRec 호출은 SQL_INVALID_HANDLE를 반환한다. HandleType이 SQL_HANDLE_ENV이면 handle은 공유나 비공유 환경 핸들이 될 수 있다.

SQLGetEnvAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetEnvAttr는 환경 속성의 현재 설정을 반환한다.

구문

```
SQLRETURN SQLGetEnvAttr(
    SQLHENV      EnvironmentHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

인자

EnvironmentHandle

[입력] 환경 핸들이다.

Attribute

[입력] 검색할 속성이다.

ValuePtr

[출력] Attribute에 지정된 속성의 현재값을 반환할 버퍼 포인터이다. ValuePtr이 NULL이면 StringLengthPtr은 ValuePtr이 가리키는 버퍼에 반환될 수 있는 (null 종료 문자를 제외한) 바이트의 합계를 반환한다.

BufferLength

[입력] ValuePtr이 문자열을 가리키면, 인자는 *ValuePtr의 길이여야 한다. ValuePtr이 정수일 경우, BufferLength는 무시된다. *ValuePtr이 (SQLGetEnvAttrW를 호출할 때) 유니코드 문자열이면, BufferLength 인자는 짝수여야 한다. 속성값이 문자열이 아닐 경우, BufferLength는 사용되지 않는다.

StringLengthPtr

[출력] *ValuePtr에 반환되는 (null 종료 문자를 제외한) 바이트의 합계가 반환되는 버퍼의 포인터이다. ValuePtr이 null 포인터이면 길이는 반환되지 않는다. 속성값이 문자열이고, 반환할 수 있는 바이트의 수가 BufferLength와 같거나 크면 *ValuePtr은 BufferLength에서 null 종료 문자만큼 뺀 길이로 잘리고 드라이버에 의해 null로 종료된다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	*ValuePtr로 반환되는 데이터가 *BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘렸다. 남아 있는 문자열 값의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQL_ATTR_ODBC_VERSION이 아직 SQLSetEnvAttr을 통해 설정되지 않았다. SQLAllocHandleStd를 사용한다면 명시적으로 SQL_ATTR_ODBC_VERSION을 설정할 필요가 없다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY092	Invalid attribute/option identifier	Attribute 인자에 지정된 값이 드라이버가 지원하는 ODBC 버전에서 유효하지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional feature not implemented	Attribute 인자에 지정된 값이 드라이버가 지원하는 ODBC 버전의 환경 속성에는 유효하지만, 드라이버가 지원하지 않는다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

드라이버의 고유한 환경 속성은 없다. Attribute가 문자열을 반환하는 속성을 지정할 경우, ValuePtr은 문자열이 반환되는 버퍼의 포인터여야 한다. Null 종료 문자를 포함한 문자열의 최대 길이는 BufferLength 바이트이다.

SQLGetEnvAttr은 환경 핸들이 할당되고 해제되는 도중의 어느 시점에서든 호출될 수 있다. 응용 프로그램이 성공적으로 할당한 모든 환경 속성은 SQL_HANDLE_ENV의 Handletype과 함께 EnvironmentHandle에 SQLFreeHandle이 호출될 때까지 환경을 위해 유지된다. 환경 핸들은 하나만 사용하는 것이 바람직하다.

노트

SQL_ATTR_OUTPUT_NTS 환경 속성은 표준을 준수하는 응용 프로그램에서 지원된다. SQLGetEnvAttr가 호출되면, ODBC 3.x 드라이버 관리자는 이 속성에 대해 항상 SQL_TRUE를 반환한다. SQL_ATTR_OUTPUT_NTS는 오직 SQLSetEnvAttr 호출에 의해서만 SQL_TRUE로 설정할 수 있다.

다음 표는 SQLGetEnvAttr을 통해 조회할 수 있는 속성 목록이다.

Attribute	ValuePtr 내용
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_CP_MATCH (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ODBC_VERSION (ODBC 3.0)	특정 기능이 ODBC 2.x으로 동작할지 또는 ODBC 3.x 으로 동작할지를 나타내는 32 비트 정수이다. 다음 값들이 속성을 설정하는데 사용된다. SQL_OV_ODBC3_80 = 드라이버 관리자와 드라이버가 다음과 같이 ODBC 3.8로 동작한다. • 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 3.x 코드값을 기대하고 반환한다. • 드라이버는 SQLError, SQLGetDiagField 또는 SQLGetDiagRec가 호출될 때 ODBC 3.x SQLSTATE 코드들을 반환한다. • SQLTables의 CatalogName 인자가 패턴검색을 허용한다. SQL_OV_ODBC3 = 드라이버 관리자와 드라이버가 다음과 같이 ODBC 3.x로 동작한다. • 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 3.x 코드값을 기대하고 반환한다. • 드라이버는 SQLError, SQLGetDiagField 또는 SQLGetDiagRec가 호출될 때 ODBC 3.x SQLSTATE 코드들을 반환한다. • SQLTables의 CatalogName 인자가 패턴검색을 허용한다. • 드라이버 관리자가 C 데이터 형식 확장성을 지원하지 않는다. SQL_OV_ODBC2 = 드라이버 관리자와 드라이버는 ODBC 2.x로 동작한다. 이것은 ODBC 2.x 응용 프로그램이 ODBC 3.x 드라이버에서 동작할 때 유용하다. • 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 2.x 코드값을 기대하고 반환한다. • 드라이버는 SQLError, SQLGetDiagField 또는 SQLGetDiagRec가 호출될 때 ODBC 2.x SQLSTATE 코드들을 반환한다. • SQLTables의 CatalogName 인자가 패턴검색을 허용하지 않는다. • 드라이버 관리자가 C 데이터 형식 확장성을 지원하지 않는다. 응용 프로그램은 SQLHENV 인자를 가지거나 SQLSTATE HY010 (function sequence error)를 반환하는 함수를 호출하기 전에 이 환경 속성 값을 설정해야 한다. 이런 환경적 플래그를 위한 추가 동작이 있는지 여부는 드라이버가 명시해야 한다
SQL_ATTR_OUTPUT_NTS	

Attribute	ValuePtr 내용
(ODBC 3.0)	드라이버에서 지원하지 않는다.

SQLGetFunctions

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLGetFunctions는 드라이버가 지정한 ODBC 함수를 지원하는지에 대한 정보를 반환한다. 이 함수는 드라이버 관리자 또는 드라이버에서 구현된다. 드라이버가 SQLGetFunctions를 구현할 경우, 드라이버 관리자는 드라이버 안의 함수를 호출할 것이다.

구문

```
SQLRETURN SQLGetFunctions(
    SQLHDBC          ConnectionHandle,
    SQLUSMALLINT      FunctionId,
    SQLUSMALLINT *    SupportedPtr);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

FunctionId

[입력] SQL_API_ODBC3_ALL_FUNCTIONS 또는 SQL_API_ALL_FUNCTIONS와 관련 있는 ODBC 함수를 식별하는 #define 값이다. ODBC 3.x와 이전 버전 함수에서 지원되는지 결정하기 위해 ODBC 3.x 응용 프로그램에서 SQL_API_ODBC3_ALL_FUNCTIONS를 사용한다. SQL_API_ALL_FUNCTIONS는 ODBC 2.x와 이전 버전 함수에서 지원되는지 여부를 결정하기 위해 ODBC 2.x에서 사용된다.

ODBC 함수를 식별하는 #define 값의 목록은 설명 섹션의 테이블을 참조한다.

SupportedPtr

[출력] FunctionId가 단일 ODBC 함수를 식별하면, SupportedPtr은 단일 SQLUSMALLINT 값을 가리키며 드라이버가 지정된 함수를 지원하면 SQL_TRUE, 지원하지 않으면 SQL_FALSE가 된다.

FunctionId가 SQL_API_ODBC3_ALL_FUNCTIONS이면 SupportedPtr은 SQL_API_ODBC3_ALL_FUNCTIONS_SIZE와 같은 개수의 요소를 갖는 SQLSMALLINT 배열을 가리킨다. 이 배열은 ODBC 3.x 또는 이전 함수에서 지원되는지 여부를 확인하기 위해 사용할 수 있는 4000 bit 비트맵으로써 드라이버 관리자에 의해 처리된다. SQL_FUNC_EXISTS 매크로는 함수가 지원되는지 여부를 확인하기 위해 호출된다. ODBC 3.x 응용 프로그램은 ODBC 3.x 또는 ODBC 2.x에 대비하여 SQL_API_ODBC3_ALL_FUNCTIONS를 사용하여 SQLG

etFunctions를 호출할 수 있다.

FunctionId가 SQL_API_ALL_FUNCTIONS이면 SupportedPtr은 100 개의 요소를 갖는 배열을 가리킨다. 이 배열은 각 ODBC 함수를 식별하는 FunctionId가 사용하는 #define 값으로 인덱싱되고, 일부 요소는 사용되지 않고 예약되어 있다. ODBC 2.x 또는 이전 함수가 드라이버에 의해 지원되는 것이 식별되면 요소는 SQL_TRUE가 된다. 드라이버가 지원하지 않는 ODBC 함수이거나 ODBC 함수가 아닌 경우, 요소는 SQL_FALSE가 된다.

*SupportedPtr에 반환되는 배열은 0 기반 인덱싱을 사용한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	SQLConnect, SQLBrowseConnect, SQLDriverConnect 이전에 SQLGetFunctions가 호출되었다. ConnectionHandle을 위해 SQLBrowseConnect이 호출되었고 SQL_NEED_DATA가 반환되었다. SQLBrowseConnect가 SQL_SUCCESS_WITH_INFO 또는 SQL_SUCCESS를 반환하기 전에 SQLGetfunction이 호출되었다. ConnectionHandle을 위해 SQLExecute, SQLExecDirect, SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 데이터가 모든 이어지는 매개 변수를 검색하기 전에 이 함수가 호출되었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY095	Function type out of range	FunctionId 값이 유효하지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.

SQLSTATE	Error	설명
	allowed.	
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 연결 타임아웃 주기는 SQLSetConnectAttr을 통해 SQL_ATTR_CONNECTION_TIMEOUT으로 설정할 수 있다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

SQLGetFunctions는 지원되는 SQLGetFunctions, SQLDataSources, SQLDrivers들을 반환한다. 이는 이러한 함수가 드라이버 관리자에서 구현되어 있기 때문이다. 드라이버 관리자는 유니코드 함수가 존재할 경우, 유니코드 함수에 대응하는 ANSI 함수를 매핑하고, ANSI 함수가 존재할 경우, ANSI 함수에 대응하는 유니코드 함수를 매핑한다.

다음은 ISO 92 표준 준수 레벨을 따르는 함수에 대해 FunctionId에 유효한 값의 목록이다.

- SQL_API_SQLALLOCHANDLE
- SQL_API_SQLBINDCOL
- SQL_API_SQLCANCEL
- SQL_API_SQLCLOSECURSOR
- SQL_API_SQLCOLATTRIBUTE
- SQL_API_SQLCONNECT
- SQL_API_SQLCOPYDESC
- SQL_API_SQLDATASOURCES
- SQL_API_SQLDESCRIBECOL
- SQL_API_SQLDISCONNECT
- SQL_API_SQLDRIVERS
- SQL_API_SQLENDTRAN
- SQL_API_SQLEXECDIRECT
- SQL_API_SQLEXECUTE
- SQL_API_SQLFETCH
- SQL_API_SQLFETCHSCROLL
- SQL_API_SQLFREEHANDLE
- SQL_API_SQLFREESTMT
- SQL_API_SQLGETCONNECTATTR
- SQL_API_SQLGETCURSORNAME
- SQL_API_SQLGETDATA
- SQL_API_SQLGETDESCFIELD
- SQL_API_SQLGETDESCREC
- SQL_API_SQLGETDIAGFIELD
- SQL_API_SQLGETDIAGREC
- SQL_API_SQLGETENVATTR

- SQL_API_SQLGETFUNCTIONS
- SQL_API_SQLGETINFO
- SQL_API_SQLGETSTMTATTR
- SQL_API_SQLGETTYPEINFO
- SQL_API_SQLNUMRESULTCOLS
- SQL_API_SQLPARAMDATA
- SQL_API_SQLPREPARE
- SQL_API_SQLPUTDATA
- SQL_API_SQLROWCOUNT
- SQL_API_SQLSETCONNECTATTR
- SQL_API_SQLSETCURSORNAME
- SQL_API_SQLSETDESCFIELD
- SQL_API_SQLSETDESCREC
- SQL_API_SQLSETENVATTR
- SQL_API_SQLSETSTMTATTR

다음은 오픈 그룹 표준 준수 레벨을 따르는 함수에 대해 FunctionId에 유효한 값의 목록이다.

- SQL_API_SQLCOLUMNS
- SQL_API_SQLSPECIALCOLUMNS
- SQL_API_SQLSTATISTICS
- SQL_API_SQLTABLES

다음은 ODBC 표준 준수 레벨을 따르는 함수에 대한 FunctionId에 유효한 값의 목록이다.

- SQL_API_SQLBINDPARAMETER
- SQL_API_SQLBROWSECONNECT
- SQL_API_SQLBULKOPERATIONS [\[1\]](#)
- SQL_API_SQLCOLUMNPRIVILEGES
- SQL_API_SQLDESCRIBEPARAM
- SQL_API_SQLDRIVERCONNECT
- SQL_API_SQLFOREIGNKEYS
- SQL_API_SQLMORERESULTS
- SQL_API_SQLNATIVESQL
- SQL_API_SQLNUMPARAMS
- SQL_API_SQLPRIMARYKEYS
- SQL_API_SQLPROCEDURECOLUMNS
- SQL_API_SQLPROCEDURES
- SQL_API_SQLSETPOS
- SQL_API_SQLTABLEPRIVILEGES

[\[1\]](#) ODBC 2.x 드라이버로 작업할 때, SQLBulkOperations는 ODBC 2.x 드라이버가 SQLSetPos를 지원하고, 정보

타입 SQL_POS_OPERATIONS가 설정된 SQL_POS_ADD 비트를 반환할 때만 지원되고 반환된다.

ODBC 3.8 이후에 도입된 함수에 대한 FunctionId에 유효한 값은 SQL_API_SQLCANCELHANDLE [2]이다.

[2] 드라이버가 SQLCancel과 SQLCancelHandle을 모두 지원해야 SQLCancelHandle이 반환된다. SQLCancel은 지원되지만 SQLCancelHandle은 지원되지 않을 경우, 응용 프로그램은 SQLCancel에 매핑되기 때문에 여전히 명령문 핸들에 대해 SQLCancelHandle을 호출할 수 있다.

SQL_FUNC_EXISTS 매크로

FunctionId 인자를 사용하여 SQLGetFunctions가 SQL_API_ODBC3_ALL_FUNCTIONS로 호출된 후에 ODBC 3.x 또는 이전 함수의 지원 여부를 확인하기 위해 SQL_FUNC_EXISTS(SupportedPtr, FunctionID) 매크로가 사용된다. 응용 프로그램은 SQLGetFunctions에서 전달된 SupportedPtr을 SupportedPtr 인자로 설정하고, FunctionID 인자를 함수에 대한 #define 값으로 설정하여 SQL_FUNC_EXISTS를 사용한다. SQL_FUNC_EXISTS는 함수가 지원되면 SQL_TRUE, 그렇지 않다면 SQL_FALSE를 반환한다.

노트

ODBC 2.x 드라이버로 작업할 때, ODBC 3.x 드라이버 관리자는 SQLAllocHandle과 SQLFreeHandle에 대해 SQL_TRUE를 반환한다. 왜냐하면 SQLAllocHandle은 SQLAllocEnv, SQLAllocConnect 또는 SQLAllocStmt에 매핑되고, SQLFreeHandle은 SQLFreeEnv, SQLFreeConnect 또는 SQLFreeStmt에 매핑되기 때문이다. 그러나 SQLFreeHandle에 대해 SQL_TRUE를 반환하더라도 SQL_HANDLE_DESC를 HandleType으로 사용하는 SQLFreeHandle은 지원하지 않는다. 이런 경우의 ODBC 2.x 함수에 매핑되는 함수가 없기 때문이다.

SQLGetGroupCount

적합성

표준 준수: 없음

개요

SQLGetGroupCount는 클러스터 그룹의 개수를 반환한다.

구문

```
SQLRETURN SQLGetGroupCount(  
    SQLHDBC      ConnectionHandle,  
    SQLINTEGER * GroupCountPtr );
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

GroupCountPtr

[출력] 클러스터 그룹의 개수이다.

반환

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08003	Connection not open	ConnectionHandle이 연결된 상태가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
HY010	Function sequence error	이 함수는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서 호출되어야 한다.

설명

SQLGetGroupCount는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서만 호출할 수 있다.

SQLGetGroupIDs

적합성

표준 준수: 없음

개요

SQLGetGroupIDs는 클러스터 그룹들의 ID를 반환한다.

구문

```
SQLRETURN SQLGetGroupIDs(  
    SQLHDBC      ConnectionHandle,  
    SQLINTEGER * GroupIDArray );
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

GroupIDArray

[출력] 클러스터 그룹 ID의 배열이다.

반환

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08003	Connection not open	ConnectionHandle이 연결된 상태가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
HY010	Function sequence error	이 함수는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정이 된 상태에서 호출되어야 한다.

설명

SQLGetGroupIDs는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서만 호출할 수 있다.

GroupIDArray의 element 개수는 SQLGetGroupCount에서 반환되는 클러스터 그룹의 개수와 동일해야 한다.

SQLGetGroupName

적합성

표준 준수: 없음

개요

SQLGetGroupName는 GroupID에 해당하는 클러스터 그룹의 이름을 반환한다.

구문

```
SQLRETURN SQLGetGroupName(  
    SQLHDBC      ConnectionHandle,  
    SQLINTEGER    GroupID,  
    SQLCHAR      * GroupName,  
    SQLSMALLINT   BufferLength,  
    SQLSMALLINT * NameLengthPtr );
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

GroupID

[입력] 클러스터 그룹 ID이다.

GroupName

[출력] null로 종료되는 클러스터 그룹의 이름을 반환하는 버퍼 포인터이다.

BufferLength

[입력] *GroupName의 길이이다.

NameLengthPtr

[출력] *GroupName에 반환할 수 있는 전체 바이트 수 (null 종료 문자 제외)를 반환하는 버퍼의 포인터이다. 반환 가능한 길이가 BufferLength 보다 크거나 같으면 *GroupName이 BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘린다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01004	String data, right truncated	*GroupName의 크기가 클러스터 그룹 이름을 반환하는데 충분하지 않아 클러스터 그룹 이름이 잘렸다. 잘리지 않은 클러스터 이름의 길이는 *NameLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08003	Connection not open	ConnectionHandle이 연결된 상태가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
HY010	Function sequence error	이 함수는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서 호출되어야 한다.

설명

SQLGetGroupIDs는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서만 호출할 수 있다.

SQLGetInfo

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLGetInfo는 드라이버 및 데이터 소스와 관련된 연결에 대한 일반적인 정보를 반환한다.

구문

```
SQLRETURN SQLGetInfo(  
    SQLHDBC          ConnectionHandle,  
    SQLUSMALLINT      InfoType,  
    SQLPOINTER        InfoValuePtr,  
    SQLSMALLINT        BufferLength,  
    SQLSMALLINT *      StringLengthPtr);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

InfoType

[입력] 정보 타입이다.

InfoValuePtr

[출력] 정보가 반환되는 버퍼의 포인터이다. 요청된 InfoType에 따라 null 종료 문자열, SQLUSMALLINT 값, SQLUIINTEGER 비트 마스크, SQLUIINTEGER 플래그, SQLUIINTEGER 바이너리 값, 또는 SQLULEN 값 중 하나가 반환된다.

InfoType 인자가 SQL_DRIVER_HDESC 또는 SQL_DRIVER_HSTMT이면 InfoValuePtr 인자는 입력과 출력이다.

InfoValuePtr이 NULL이면, StringLengthPtr은 InfoValuePtr이 가리키는 버퍼에 반환될 수 있는 (null 종료 문자를 제외한) 총 바이트 수를 반환한다.

BufferLength

[입력] *InfoValuePtr 버퍼의 길이이다. *InfoValuePtr이 문자열이 아니거나 InfoValuePtr이 null 포인터일 경우, BufferLength 인자는 무시된다. 드라이버는 InfoType에 따라 *InfoValuePtr의 크기를 SQLUSMALLINT 또는 SQLUIINTEGER로 간주한다. (SQLGetInfoW를 호출할 경우) InfoValuePtr이 유니코드 문자열이면

BufferLength는 짝수여야 한다: 그렇지 않을 경우, SQLSTATE HY090이 반환된다.

StringLengthPtr

[출력] *InfoValuePtr에 반환될 수 있는 (문자 데이터에 대한 null 종료 문자를 제외한) 바이트의 총 수를 반환할 버퍼의 포인터이다.

문자 데이터의 경우, 반환될 수 있는 바이트의 수가 BufferLength와 같거나 클 경우, *InfoValuePtr 안의 정보는 (null 종료 문자를 제외한) BufferLength 바이트 길이로 잘리고 드라이버에 의해 null로 종료된다.

다른 타입의 데이터인 경우, BufferLength는 무시되고 드라이버는 InfoType에 따라 *InfoValuePtr의 크기를 SQLUSMALLINT 또는 SQLINTEGER로 간주한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	*InfoValuePtr의 버퍼의 길이가 요청된 정보가 모두 반환될 만큼 충분하지 않아 잘렸다. 잘리지 않은 정보의 길이는 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08003	Connection not open	InfoType에 요청된 정보의 타입에 대한 연결이 열려있어야 한다. ODBC에 예약된 정보 타입인 SQL_ODBC_VER은 열려있는 연결없이 반환할 수 있다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	StatementHandle을 위해 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 데이터가 모든 이어지는 매개 변수를 검색하기 전에 이 함수가 호출되었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY024	Invalid attribute value	infoType 인자가 SQL_DRIVER_HSTMT이었고, InfoValuePtr이 가리키는 값이 유효하지 않은 명령문 핸들이었다. InfoType 인자가 SQL_DRIVER_HDESC이었고, InfoValuePtr이 가리키는 값이 유효하지 않은 설명자 핸들이었다.
HY090	Invalid string or buffer length	BufferLength 인자에 대한 값이 0보다 작다. BufferLength 인자에 대한 값이 홀수이고, *InfoValuePtr은 유니코드 데이터 타입이었다.

SQLSTATE	Error	설명
HY096	Information type out of range	InfoType 인자에 지정된 값이 드라이버에 지원되는 ODBC 버전에 유효하지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 <code>SQLEndTran</code> 함수를 참조한다.
HYC00	Optional field not implemented	InfoType 인자에 지정된 값이 드라이버가 지원하지 않는 드라이버 특정 값이었다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 <code>SQLSetStmtAttr</code> 의 <code>SQL_ATTR_CONNECTION_TIMEOUT</code> 을 통해 설정할 수 있다.
IM001	Driver does not support this function	<code>DescriptorHandle</code> 과 관련된 드라이버가 함수를 지원하지 않는다.

설명

현재 정의된 정보 타입은 다음 섹션들에서 설명한다. 정보 타입의 범위는 ODBC가 예약한다. 드라이버 개발자는 고유한 드라이버별 사용에 대한 값을 오픈 그룹에 예약해야 한다. `SQLGetInfo`는 유니코드 변환 또는 드라이버가 정의한 `InfoTypes`에 대한 `thunking`을 수행하지 않는다. `*InfoValuePtr`에 반환되는 정보 형식은 요청된 `InfoTypes`에 따라 결정된다.

`SQLGetInfo`는 다음 다섯 가지 형식 중 하나의 정보로 반환된다.

- null 종료 문자열
- `SQLUSMALLINT` 값
- `SQLUINTEGER` 비트 마스크
- `SQLUINTEGER` 값
- `SQLUINTEGER` 바이너리 값

응용 프로그램은 `*InfoValuePtr`에 반환되는 값에 따라 캐스팅해야 한다.

드라이버는 다음 테이블에 정의된 각 정보 타입에 대한 값을 반환해야 한다. 정보 타입이 드라이버나 데이터 소스에 적용되지 않을 경우, 드라이버는 다음 테이블에 기록된 값 중 하나를 반환해야 한다.

Character string ("Y" or "N")

"N"

Character string (not "Y" or "N")

Empty string

SQLUSMALLINT

0

SQLINTEGER 비트 마스크 또는 SQLINTEGER 바이너리 값

0L

예를 들어 데이터 소스가 절차를 지원하지 않을 경우, SQLGetInfo는 절차와 관련된 InfoType의 값에 대해 다음 테이블에 기록된 값을 반환한다.

SQL_PROCEDURES

"N"

SQL_ACCESSIBLE_PROCEDURES

"N"

SQL_MAX_PROCEDURE_NAME_LEN

0

SQL_PROCEDURE_TERM

Empty string

ODBC에서 사용하기 위해 예약된 정보 타입의 범위에 있지만 드라이버가 지원하는 ODBC의 버전이 정의되지 않은 InfoType의 값에 대해 SQLGetInfo는 SQLSTATE HY096을 반환한다. 드라이버는 ODBC 버전을 확인하기 위해 응용 프로그램의 SQL_DRIVER_ODBC_VER 정보 타입으로 SQLGetInfo를 호출하면서 컴파일한다. 드라이버별 사용을 위해 예약된 정보 타입의 범위에는 있지만 드라이버가 지원하지 않는 InfoType의 값에 대해 SQLGetInfo는 SQLSTATE HYC00을 반환한다.

드라이버 관리자 버전을 반환하는 InfoType이 SQL_ODBC_VER인 것을 제외한 모든 SQLGetInfo 호출의 연결은 열려 있어야 한다.

드라이버 정보

InfoType 인자의 다음 값들은 활성화 상태인 명령문, 데이터 소스 이름, 인터페이스 표준 준수 레벨 등과 같은 ODBC 정보를 반환한다.

- SQL_ACTIVE_ENVIRONMENTS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_DATA_SOURCE_NAME
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDBC
- SQL_DRIVER_HDESC
- SQL_DRIVER_HENV
- SQL_DRIVER_HLIB
- SQL_DRIVER_HSTMT

- SQL_DRIVER_NAME
- SQL_DRIVER_ODBC_VER
- SQL_DRIVER_VER
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_FILE_USAGE
- SQL_GETDATA_EXTENSIONS
- SQL_INFO_SCHEMA_VIEWS
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
- SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_DRIVER_CONNECTIONS
- SQL_ODBC_INTERFACE_CONFORMANCE
- SQL_ODBC_STANDARD_CLI_CONFORMANCE
- SQL_ODBC_VER
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_ROW_UPDATES
- SQL_SEARCH_PATTERN_ESCAPE
- SQL_SERVER_NAME
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

노트

SQLGetInfo를 수행할 때, 드라이버는 정보를 전송하거나 서버로부터 요청되는 횟수를 최소화하여 성능을 향상시킬 수 있다.

DBMS 제품 정보

다음과 같은 InfoType 인자값은 DBMS 이름이나 정보와 같은 DBMS 제품 정보를 반환한다.

- SQL_DATABASE_NAME
- SQL_DBMS_NAME
- SQL_DBMS_VER

데이터 소스 정보

다음과 같은 InfoType 인자값은 커서 특성이나 트랜잭션 기능과 같은 데이터 소스에 대한 정보를 반환한다.

- SQL_ACCESSIBLE_PROCEDURES
- SQL_ACCESSIBLE_TABLES
- SQL_BOOKMARK_PERSISTENCE
- SQL_CATALOG_TERM
- SQL_COLLATION_SEQ
- SQL_CONCAT_NULL_BEHAVIOR
- SQL_CURSOR_COMMIT_BEHAVIOR
- SQL_CURSOR_ROLLBACK_BEHAVIOR
- SQL_CURSOR_SENSITIVITY
- SQL_DATA_SOURCE_READ_ONLY
- SQL_DEFAULT_TXN_ISOLATION
- SQL_DESCRIBE_PARAMETER
- SQL_MULT_RESULT_SETS
- SQL_MULTIPLE_ACTIVE_TXN
- SQL_NEED_LONG_DATA_LEN
- SQL_NULL_COLLATION
- SQL_PROCEDURE_TERM
- SQL_SCHEMA_TERM
- SQL_SCROLL_OPTIONS
- SQL_TABLE_TERM
- SQL_TXN_CAPABLE
- SQL_TXN_ISOLATION_OPTION
- SQL_USER_NAME

지원되는 SQL

다음과 같은 InfoType 인자값은 데이터 소스가 지원하는 SQL 명령문에 대한 정보를 반환한다.

- SQLAggregate_FUNCTIONS
- SQLAlter_Domain
- SQLAlter_Schema
- SQLAlter_Table
- SQLANSI_SQL_Datetime_Literals
- SQL_Catalog_Location
- SQL_Catalog_Name
- SQL_Catalog_Name_Separator
- SQL_Catalog_Usage
- SQL_Column_Alias
- SQL_Correlation_Name

- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA
- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION
- SQL_DDL_INDEX
- SQL_DROP_ASSERTION
- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_EXPRESSIONS_IN_ORDERBY
- SQL_GROUP_BY
- SQL_IDENTIFIER_CASE
- SQL_IDENTIFIER_QUOTE_CHAR
- SQL_INDEX_KEYWORDS
- SQL_INSERT_STATEMENT
- SQL_INTEGRITY
- SQL_KEYWORDS
- SQL_LIKE_ESCAPE_CLAUSE
- SQL_NON_NULLABLE_COLUMNS
- SQL_SQL_CONFORMANCE
- SQL_OJ_CAPABILITIES
- SQL_ORDER_BY_COLUMNS_IN_SELECT
- SQL_OUTER_JOINS
- SQL_PROCEDURES
- SQL_QUOTED_IDENTIFIER_CASE
- SQL_SCHEMA_USAGE
- SQL_SPECIAL_CHARACTERS
- SQL_SUBQUERIES
- SQL_UNION

SQL 제한

다음 InfoType 인자값은 같은 식별자의 최대 길이와 선택 목록의 최대 column 개수와 같은 SQL 명령의 식별자나 절에 적용되는 제한을 반환한다. 드라이버 또는 데이터 소스가 제한을 부여할 수 있다.

- SQL_MAX_BINARY_LITERAL_LEN
- SQL_MAX_CATALOG_NAME_LEN
- SQL_MAX_CHAR_LITERAL_LEN
- SQL_MAX_COLUMN_NAME_LEN
- SQL_MAX_COLUMNS_IN_GROUP_BY
- SQL_MAX_COLUMNS_IN_INDEX
- SQL_MAX_COLUMNS_IN_ORDER_BY
- SQL_MAX_COLUMNS_IN_SELECT
- SQL_MAX_COLUMNS_IN_TABLE
- SQL_MAX_CURSOR_NAME_LEN
- SQL_MAX_IDENTIFIER_LEN
- SQL_MAX_INDEX_SIZE
- SQL_MAX_PROCEDURE_NAME_LEN
- SQL_MAX_ROW_SIZE
- SQL_MAX_ROW_SIZE_INCLUDES_LONG
- SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_STATEMENT_LEN
- SQL_MAX_TABLE_NAME_LEN
- SQL_MAX_TABLES_IN_SELECT
- SQL_MAX_USER_NAME_LEN

스칼라 함수 정보

다음 InfoType 인자는 데이터 소스와 드라이버가 지원하는 스칼라 함수에 대한 정보를 반환한다.

- SQL_CONVERT_FUNCTIONS
- SQL_NUMERIC_FUNCTIONS
- SQL_STRING_FUNCTIONS
- SQL_SYSTEM_FUNCTIONS
- SQL_TIMEDATE_ADD_INTERVALS
- SQL_TIMEDATE_DIFF_INTERVALS
- SQL_TIMEDATE_FUNCTIONS

전환 정보

다음 InfoType 인자값은 CONVERT 스칼라 함수를 사용하여 데이터 소스를 지정된 SQL 데이터 타입으로 변환할 수 있는 SQL 데이터 타입의 목록을 반환한다.

- SQL_CONVERT_BIGINT
- SQL_CONVERT_BINARY
- SQL_CONVERT_BIT
- SQL_CONVERT_CHAR
- SQL_CONVERT_DATE

- SQL_CONVERT_DECIMAL
- SQL_CONVERT_DOUBLE
- SQL_CONVERT_FLOAT
- SQL_CONVERT_INTEGER
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CONVERT_LONGVARBINARY
- SQL_CONVERT_LONGVARCHAR
- SQL_CONVERT_NUMERIC
- SQL_CONVERT_REAL
- SQL_CONVERT_SMALLINT
- SQL_CONVERT_TIME
- SQL_CONVERT_TIMESTAMP
- SQL_CONVERT_TINYINT
- SQL_CONVERT_VARBINARY
- SQL_CONVERT_VARCHAR

ODBC 3.x에 추가된 정보 타입

다음과 같은 InfoType 인자값이 ODBC 3.x에 추가되었다.

- SQL_ACTIVE_ENVIRONMENTS
- SQLAggregate_FUNCTIONS
- SQL_ALTER_DOMAIN
- SQL_ALTER_SCHEMA
- SQL_ANSI_SQL_DATETIME_LITERALS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_CATALOG_NAME
- SQL_COLLATION_SEQ
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA
- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION

- SQL_CURSOR_SENSITIVITY
- SQL_DDL_INDEX
- SQL_DESCRIBE_PARAMETER
- SQL_DM_VER
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDESC
- SQL_DROP_ASSERTION
- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_INFO_SCHEMA_VIEWS
- SQL_INSERT_STATEMENT
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
- SQL_MAX_IDENTIFIER_LEN
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2
- SQL_XOPEN_CLI_YEAR

ODBC 3.x에서 개명된 정보 타입

다음과 같은 InfoType 인자값의 이름이 ODBC 3.x에서 바뀌었다.

- SQL_ACTIVE_CONNECTIONS: SQL_MAX_DRIVER_CONNECTIONS
- SQL_ACTIVE_STATEMENTS: SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_OWNER_NAME_LEN: SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_QUALIFIER_NAME_LEN: SQL_MAX_CATALOG_NAME_LEN
- SQL_ODBC_SQL_OPT_IEF: SQL_INTEGRITY
- SQL_OWNER_TERM: SQL_SCHEMA_TERM
- SQL_OWNER_USAGE: SQL_SCHEMA_USAGE
- SQL_QUALIFIER_LOCATION: SQL_CATALOG_LOCATION

- SQL_QUALIFIER_NAME_SEPARATOR: SQL_CATALOG_NAME_SEPARATOR
- SQL_QUALIFIER_TERM: SQL_CATALOG_TERM
- SQL_QUALIFIER_USAGE: SQL_CATALOG_USAGE

ODBC 3.x에 쓰이지 않는 정보 타입

다음 InfoType 인자값은 ODBC 3.x 드라이버에서는 쓰이지 않지만 ODBC 2.x 응용 프로그램과의 호환을 위해 계속 지원하는 정보 타입이다.

- SQL_FETCH_DIRECTION
- SQL_LOCK_TYPES
- SQL_ODBC_API_CONFORMANCE
- SQL_ODBC_SQL_CONFORMANCE
- SQL_POS_OPERATIONS
- SQL_POSITIONED_STATEMENTS
- SQL_SCROLL_CONCURRENCY
- SQL_STATIC_SENSITIVITY

SQLGetStmtAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLGetStmtAttr은 명령문 속성의 현재 설정을 반환한다.

구문

```
SQLRETURN SQLGetStmtAttr(
    SQLHSTMT      StatementHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

Attribute

[입력] 검색할 속성이다.

ValuePtr

[출력] Attribute에 지정된 속성값이 반환될 버퍼의 포인터이다. ValuePtr이 NULL이면, StringLengthPtr은 ValuePtr이 가리키는 버퍼에 반환될 수 있는 (문자 데이터에 대한 null 종료 문자를 제외한) 총 바이트 수를 반환한다.

BufferLength

[입력] Attribute가 ODBC에 정의된 속성이고 ValuePtr이 문자열이나 바이너리 버퍼를 가리키면, 이 인자는 *ValuePtr의 길이어야 한다. Attribute가 ODBC에 정의된 속성이고 *ValuePtr이 정수일 경우, BufferLength는 무시된다. (SQLGetStmtAttrW를 호출할 때) *ValuePtr이 유니코드 문자열일 경우, BufferLength 인자는 반드시 짝수여야 한다.

Attribute가 드라이버 정의 속성일 경우, 응용 프로그램은 BufferLength 인자를 설정하여 드라이버 관리자에 속성의 특성을 나타낸다. BufferLength는 다음 값을 갖는다.

- *ValuePtr이 문자열 포인터일 경우, BufferLength는 문자열의 길이 또는 SQL_NTS이다.

- *ValuePtr이 바이너리 버퍼의 포인터일 경우, 응용 프로그램은 SQL_LEN_BINARY_ATTR (length) 매크로 결과를 BufferLength에 저장한다. 이것은 BufferLength에 음수를 저장한다.
- *ValuePtr이 문자열이나 바이너리 문자열이 아닌 다른 값의 포인터일 경우, BufferLength는 SQL_IS_POINTER 값을 갖는다.
- *ValuePtr이 고정 길이 데이터 타입일 경우, BufferLength는 SQL_IS_INTEGER나 SQL_IS_INTEGER를 갖는다.

StringLengthPtr

[출력] *ValuePtr에 반환될 수 있는 (null 종료 문자를 제외한) 총 바이트 수를 반환할 버퍼의 포인터이다. ValuePtr이 null 포인터일 경우, 길이는 반환되지 않는다. 속성값이 문자열인 경우, 반환 가능한 바이트의 수가 BufferLength 보다 크거나 같다면 *ValuePtr 안의 데이터는 BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘리고 드라이버에 의해 null 종료된다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	*ValuePtr에 반환되는 데이터가 BufferLength에서 null 종료 문자를 뺀 만큼의 길이로 잘렸다. 잘리지 않은 문자열의 값은 *StringLengthPtr에 반환된다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
24000	Invalid cursor state	Attribute 인자가 SQL_ATTR_ROW_NUMBER이었고 커서가 열리지 않았거나 커서가 결과 집합의 시작 전이나 결과 집합의 끝 이후에 위치하였다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대하여 비동기적으로 수행되는 함수를 호출했는데, SQLGetStmtAttr이 호출 되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출했고 이 함수가 호출 되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.

SQLSTATE	Error	설명
HY090	Invalid string or buffer length	*ValuePtr이 문자열이고 BufferLength가 0보다 작았지만 SQL_NTS와 같지 않다.
HY092	Invalid attribute/option identifier	Attribute 인자에 지정된 값이 드라이버가 지원하는 ODBC 버전에서 유효하지 않다.
HY109	Invalid cursor position	Attribute 인자가 SQL_ATTR_ROW_NUMBER이었고 row가 지워졌거나 검색되지 않았다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional field not implemented	Attribute 인자에 지정된 값이 드라이버가 지원하는 ODBC 버전에서 유효한 ODBC 명령문 속성이지만 드라이버가 지원하지 않는다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

SQLGetStmtAttr은 attribute에 지정된 명령문 속성값을 *ValuePtr에 반환한다. 반환되는 값은 SQLULEN 값 또는 null 종료 문자열이 될 수 있다. 이 값이 SQLULEN이면 일부 드라이버가 버퍼의 하위 32 비트 또는 16 비트를 기록하고 상위 비트는 그대로 유지하기 때문에 응용 프로그램이 SQLULEN의 버퍼를 사용하고 함수가 호출되기 전에 0으로 초기화 시켜야한다. 또한 BufferLength와 StringLengthPtr은 사용되지 않는다. 이 값이 null 종료 문자열일 경우, 응용 프로그램은 BufferLength 인자의 최대 문자열 길이를 지정하고, 드라이버는 문자열의 길이를 *StringLengthPtr 버퍼로 반환한다.

ODBC 2.x 드라이버로 작업하는 응용 프로그램이 SQLGetStmtAttr을 호출하려면 SQLGetStmtAttr을 드라이버 관리자의 SQLGetStmtOption에 매핑해야 한다.

다음 명령문 속성은 읽기 전용이기 때문에 SQLGetStmtAttr로 검색할 수는 있지만, SQLSetStmtAttr로 설정할 수는 없다.

- SQL_ATTR_IMP_PARAM_DESC
- SQL_ATTR_IMP_ROW_DESC
- SQL_ATTR_ROW_NUMBER

명령문 속성

현재 정의된 속성들과 그것들이 도입된 ODBC 버전은 다음 표와 같다.

속성	ValuePtr 내용

속성	ValuePtr 내용
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	명령문 핸들에 대한 SQLExecute와 SQLExecDirect에 차후 호출하기 위한 APD 핸들이다. 이 속성의 초기값은 명령문이 초기 할당됐을 때 내재적으로 할당된 설명자이다. 이 속성을 SQL_NULL_DESC나 핸들이 본래 할당한 설명자로 설정할 경우 이전에 명령문과 관련되어 명시적으로 할당된 APD 핸들은 분리되고 명령문 핸들은 내재적으로 할당된 APD 핸들로 돌아간다. 이 속성은 다른 명령문에 내재적으로 할당된 설명자 또는 같은 명령문에 내재된 다른 설명자 핸들로 설정될 수 없다. 내재적으로 할당된 설명자 핸들들은 한 명령문 혹은 설명자 핸들과 관련지어질 수 없다.
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	명령문 핸들의 다음 fetch들을 위한 ARD 핸들이다. 이 속성의 초기값은 명령문이 초기 할당되었을 때 내재적으로 할당된 설명자이다. 이 속성을 SQL_NULL_DESC나 핸들이 본래 할당한 설명자로 설정할 경우 이전에 명령문과 관련되어 명시적으로 할당된 ARD 핸들은 분리되고 명령문 핸들은 내재적으로 할당된 ARD 핸들로 돌아간다. 이 속성은 다른 명령문에 내재적으로 할당된 설명자 또는 같은 명령문에 내재된 다른 설명자 핸들로 설정될 수 없다. 내재적으로 할당된 설명자 핸들들은 한 명령문 혹은 설명자 핸들과 관련지어질 수 없다.
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ATOMIC_EXECUTION	SQLUSMALLINT: Atomic insert의 동작 여부이다. - SQL_ATOMIC_EXECUTION_OFF - SQL_ATOMIC_EXECUTION_ON
SQL_ATTR_CONCURRENCY (ODBC 2.0)	- SQLULEN: 커서의 동시성을 명시한 값이다. - SQL_CONCUR_READ_ONLY: 커서가 읽기 전용이고, 갱신을 허용하지 않는다. - SQL_CONCUR_LOCK: 커서가 row의 갱신을 완료하는데 필요한 최소한의 level locking을 사용한다. - SQL_CONCUR_ROWVER: 커서가 SQLBase ROWID나 Sybase TIMESTAMP와 같이 row의 버전을 제어, 비교하는 동시성을 사용한다. - SQL_CONCUR_VALUES: 커서가 값을 제어하고 비교하는 동시성을 사용한다. SQL_ATTR_CONCURRENCY의 기본값은 SQL_CONCUR_READ_ONLY이다.

속성	ValuePtr 내용
	SQL_ATTR_CURSOR_TYPE attribute가 SQL_ATTR_CONCURRENCY에서 지원하지 않는 값으로 갱신될 경우, SQL_ATTR_CONCURRENCY의 값은 실행 시점에 갱신되고 SQLExecDirect나 SQLPrepare가 호출될 때 경고가 이슈화된다. 드라이버가 SELECT FOR UPDATE 명령문을 지원하고 이런 명령문이 실행되는 동안 SQL_ATTR_CONCURRENCY가 SQL_CONCUR_READ_ONLY로 변경될 경우, 오류를 반환한다. SQL_ATTR_CONCURRENCY 값이 드라이버가 지원하는 SQL_ATTR_CURSOR_TYPE 값이나 지원하지 않는 값으로 갱신될 경우, SQL_ATTR_CURSOR_TYPE 값은 실행되는 시점에 갱신되고 SQLExecDirect나 SQLPrepare가 실행될 때 SQLSTATE 01S02 (option value changed)가 이슈화된다. 명시된 동시성을 데이터 소스에서 지원하지 않을 경우, 드라이버는 다른 동시성으로 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. 드라이버는 SQL_CONCUR_VALUES를 SQL_CONCUR_ROWVER로 혹은 반대로 대체한다. 또한 SQL_CONCUR_LOCK은 SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES 순으로 대체한다. 대체된 값의 유효성은 실행 시점 전까지는 확인되지 않는다.
SQL_ATTR_CURSOR_HOLDABLE	이 속성은 트랜잭션 (transaction)이 종료될 때 커서 (cursor)를 유지 (hold) 할지 여부를 지정하는 SQLULEN 타입의 값이다. - SQL_NONHOLDABLE: 트랜잭션이 종료되면 커서는 사라지고 더 이상 사용할 수 없다. 즉, 커서의 유지 기간이 현재 트랜잭션에만 한정된다. - SQL_HOLDABLE: 트랜잭션이 종료된 후에도 커서는 계속 유지 (hold) 되며, 이후에도 계속 사용할 수 있다. 커서의 유지 기간이 트랜잭션의 경계를 넘어 확장된다. 기본값 (default)은 SQL_HOLDABLE이다. 이 속성을 설정하면 이후 실행되는 SQLExecDirect 또는 SQLExecute 호출에 영향을 미친다.
SQL_ATTR_CURSOR_SCROLLABLE (ODBC 3.0)	이 속성은 애플리케이션에서 필요한 cursor 수준을 지정하는 SQLULEN 타입의 값이다. 이 속성을 설정하면 이후의 SQLExecDirect와 SQLExecute 호출에 영향을 미친다. - SQL_NONSCROLLABLE: 이 값이 기본값 (default)이다. 이 값은 statement handle에서 스크롤 가능한 커서 (scrollable cursor)가 필요하지 않음을 나타낸다. 이 handle에서 애플리케이션이 SQLFetchScroll을 호출하면, FetchOrientation으로 지정할 수 있는 유효한 값은 오직 SQL_FETCH_NEXT 뿐이다. - SQL_SCROLLABLE: 이 값은 statement handle에서 스크롤 가능한 커서 (scrollable cursor)가 필요함을 나타낸다. SQLFetchScroll을 호출할 때, 애플리케이션은 FetchOrientation의 모든 유효한 값을 지정할 수 있으며, 순차적인 방식 (sequential mode)이 아닌 다른 방식으로 커서 위치를 이동할 수 있다.
	이 속성은 statement handle의 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영할지 (visible) 여부를 지정하는 SQLULEN 타입의 값이다. 이 속성을 설정하면 이후의 SQLExecDirect와 SQLExecute 호출에 영향을 미친다. 애플리케이션은 이 속성의 값을 읽어 초기 상태 또는 최근에 설정된 상태를 확인할 수 있다.

속성	ValuePtr 내용
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	- SQL_UNSPECIFIED: 이 값이 기본값 (default) 이다. 커서 유형이나 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영할지 여부가 명시되지 않은 상태이다. 이 statement handle의 cursor는 다른 cursor의 변경 사항을 전혀 반영하지 않을 수도 있고, 일부 또는 전체를 반영할 수도 있다. - SQL_INSENSITIVE: 이 값은 statement handle의 모든 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영하지 않은 상태의 데이터를 표시함을 나타낸다. Inensitive 커서는 읽기 전용 (read-only)이다. 이는 읽기 전용 동시성 (read-only concurrency)을 가지는 static 커서에 해당한다. - SQL_SENSITIVE: 이 값은 statement handle의 모든 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영한 상태의 데이터를 표시함을 나타낸다.
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	SQL_ATTR_CURSOR_TYPE (ODBC 2.0) 이 속성은 cursor의 유형을 지정하는 SQLULEN 값이다: - SQL_CURSOR_FORWARD_ONLY: 커서는 오직 앞으로만 (forward-only) 이동할 수 있다. - SQL_CURSOR_STATIC: 결과 집합 (result set)의 데이터가 정적 (static) 이다. - SQL_CURSOR_KEYSET_DRIVEN: 드라이버가 SQL_ATTR_KEYSET_SIZE statement 속성으로 지정된 행 수만큼의 키를 저장하고 사용한다. - SQL_CURSOR_DYNAMIC: 드라이버가 현재 rowset에 포함된 행들의 키만 저장하고 사용한다. 기본값 (default)은 SQL_CURSOR_FORWARD_ONLY이다. SQL statement가 이미 prepared 된 이후에는 이 값을 설정할 수 없다. 지정된 cursor type을 데이터 소스가 지원하지 않는 경우, 드라이버는 다른 cursor type으로 대체한 후 SQLSTATE 01S02 (Option value changed)를 반환한다. 만약 mixed 또는 dynamic 커서를 지원하지 않는 경우, 드라이버는 우선 keyset-driven cursor로 대체를 시도하고, 이 또한 지원되지 않으면 static cursor로 대체한다. 또한, keyset-driven cursor가 지원되지 않는 경우에도 static cursor로 대체된다.
SQL_ATTR_ENABLE_AUTO_IPD (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_EXPLAIN_PLAN_OPTION	SQLUSMALLINT: Plan 정보 생성 여부이다. - SQL_EXPLAIN_PLAN_OFF: Plan 정보를 생성하지 않는다. - SQL_EXPLAIN_PLAN_ON: SQL 구문을 수행하고, plan 정보를 생성한다. - SQL_EXPLAIN_PLAN_ONLY: SQL 구문을 수행하지 않고, plan 정보를 생성한다.
SQL_ATTR_EXPLAIN_PLAN_TEXT	생성된 plan 문자열 (읽기 전용)이다.
SQL_ATTR_FETCH_BOOK	

속성	ValuePtr 내용
MARK_PTR (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_FETCH_FAILOVER	SQLUSMALLINT: fetch failover 사용 여부이다. - SQL_FETCH_FAILOVER_OFF: fetch failover를 사용하지 않는다. - SQL_FETCH_FAILOVER_ON: fetch failover를 사용한다.
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	IPD의 핸들이다. 이 속성값은 명령문이 초기에 할당되었을 때 할당된 설명자이다. 응용 프로그램은 이 속성을 설정할 수 없다. 이 속성은 SQLGetStmtAttr을 호출하여 조회할 수 있지만 SQLSetStmtAttr을 통해 설정할 수는 없다.
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	IRD의 핸들이다. 이 속성값은 명령문이 초기에 할당되었을 때 할당된 설명자이다. 응용 프로그램은 이 속성을 설정할 수 없다. 이 속성은 SQLGetStmtAttr을 호출하여 조회할 수 있지만 SQLSetStmtAttr을 통해 설정할 수는 없다.
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_MAX_ROWS (ODBC 1.0)	SQLULEN: SELECT 명령문에서 반환되는 row들의 최대 개수에 대응하는 값이다. *ValuePtr이 0과 같으면 드라이버가 모든 row들을 반환한다. 이 속성은 네트워크 트래픽을 감소시키는 것을 목적으로 한다. 개념적으로 이것은 결과 집합이 만들어질 때 적용되고 첫 번째 ValuePtr row들의 결과 집합을 제한한다. 결과 집합 row들의 개수가 ValuePtr보다 클 경우, 결과 집합은 줄어든다. SQL_ATTR_MAX_ROWS는 명령문의 모든 결과 집합들에 적용되고 카탈로그 함수들에 의해 반환되는 것들을 포함한다. SQL_ATTR_MAX_ROWS는 커서 row 개수 값의 최대치를 설정한다. SQL_ATTR_MAX_ROWS가 제대로 구현될 것을 보장할 수 없는 경우, (결과 집합 크기 제한이 데이터 소스에서 구현될 수 없는 경우) 드라이버는 SQLFetch나 SQLFetchScroll에 대해 SQL_ATTR_MAX_ROWS 동작을 모방해서는 안된다. SELECT 명령문들 (카탈로그 함수들과 같은) 이외의 명령문들에 SQL_ATTR_MAX_ROWS를 적용할지 여부는 드라이버에서 정의한다. 이 속성값은 열려 있는 커서에 설정할 수 있다. 하지만 즉각적인 효과는 나타나지 않는다. 이 경우, 드라이버는 SQLSTATE 01S02 (option value changed)를 반환하고 속성을 원래 값으로 설정한다.
	SQLULEN: 카탈로그 함수들의 문자열 인자들을 어떻게 취급할지에 대한 값이다. SQL_TRUE일 경우, 카탈로그 함수들의 문자열 인자들을 식별자처럼 취급한다. 이 경우 대소문

속성	ValuePtr 내용
SQL_ATTR_METADATA_ID (ODBC 3.0)	자를 구분하지 않는다. 범위가 정해져 있지 않은 문자열들에 대해 드라이버는 후 공백을 제거하고 문자열을 대문자로 변환한다. 범위가 정해진 문자열들에 대해 드라이버는 선 공백과 후 공백을 제거하고 구분자들 사이의 문자 그대로를 취한다. 이 인자들 중 하나가 NULL 포인터로 설정될 경우, 함수는 SQL_ERROR와 SQLSTATE HY009 (invalid use of null pointer)를 반환한다. SQL_FALSE일 경우, 카탈로그 함수들의 문자열 인자들을 식별자처럼 취급하지 않는다. 이 경우 대소문자를 구분한다. 인수에 따라 인자들이 문자열 검색 패턴을 포함할 수도 있고 포함하지 않을 수도 있다. 기본값은 SQL_FALSE이다. SQLTables의 TableType 인자가 취하는 값들의 목록은 이 속성에 영향을 받지 않는다. SQL_ATTR_METADATA_ID는 연결 level에서도 설정할 수 있다. (이것과 SQL_ATTR_ASYNC_ENABLE은 고유하며 명령문 속성들인 동시에 연결 속성이기도 하다.) 자세한 내용은 카탈로그 함수의 인자를 참조한다.
SQL_ATTR_NOSCAN (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	SQLULEN*: 동적 매개 변수들의 바인딩을 바꾸는 포인터를 추가할 offset을 표시하는 값이다. 이 필드가 NULL이 아닐 경우, 드라이버는 포인터를 역참조하고 추가적으로 설명자 레코드 (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) 안의 지연된 필드들의 각 값을 역참조하며 바인딩 될 때 새로운 포인터 값들을 사용한다. 이것은 기본적으로 NULL로 설정된다. 바인드 offset은 항상 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR 필드들에 직접적으로 추가된다. 만약 offset이 다른 값으로 바뀌었다면 새로운 값 역시 설명자 필드값에 직접적으로 추가된다. 새로운 offset은 필드값의 이전 offset에 추가되지 않는다. 이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_BIND_TYPE 필드를 설정한다.
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	SQLULEN: 동적 매개 변수에 사용될 바인딩 방향을 표시한다. 이 필드는 column 방향 바인딩을 선택한 SQL_PARAM_BIND_BY_COLUMN으로 설정되어 있다. (기본값) Row 방향 바인딩을 선택하려면 이 필드를 구조체의 길이 또는 동적 매개 변수 집합에 바인딩 될 버퍼의 인스턴스로 설정해야 한다. 이 길이는 바인딩 매개 변수와 구조체의 패딩 모두를 위한 공간을 포함하거나 바인딩 매개 변수 주소가 지정된 길이로 증가할 때의 결과가 반드시 다음 매개 변수의 시작을 가리키도록 버퍼링 해야한다. ANSI C의 sizeof 연산자를 사용하면 이 행동이 보장된다. 이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_BIND_TYPE 필드를 설정한다.
	SQLUSMALLINT*: SQL 명령문을 실행하는 동안 매개 변수를 무시하는데 사용되는 SQL

속성	ValuePtr 내용
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	USMALLINT 값들의 배열을 가리키는 값이다. 각각의 값은 (매개 변수를 실행시키기 위해) SQL_PARAM_PROCEED 또는 (매개 변수를 무시하기 위해) SQL_PARAM_IGNORE 다. APD 안의 SQL_DESC_ARRAY_STATUS_PTR이 가리키는 배열의 상태값을 설정하여 처리 중에 매개 변수 집합을 무시할 수 있다. 이 상태값이 SQL_PARAM_PROCEED이거나 배열 요소가 설정되지 않았을 경우에 매개 변수 집합이 처리된다. 이 명령문 속성은 NULL 포인터로 설정될 수 있다. 이 경우, 드라이버는 매개 변수 상태값을 반환하지 않는다. 이 속성은 언제든지 설정할 수 있지만 새로운 값은 다음 SQLExecDirect나 SQLExecute가 호출될 때까지 사용되지 않는다. 바인딩 된 매개 변수가 없는 경우, 이 속성은 무시된다. 이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_ARRAY_STATUS_PTR 필드를 설정한다.
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	- SQLUSMALLINT*: SQLExecute나 SQLExecDirect를 호출한 후에 매개 변수 값 각각의 row 들의 상태 정보값들을 포함하는 SQLUSMALLINT 배열 값을 가리키는 값이다. 이 필드는 PARAMSET_SIZE가 1보다 클 때만 요구된다. 상태값은 다른 값들을 포함할 수 있다. - SQL_PARAM_SUCCESS: SQL 명령문이 이 매개 변수 집합에 대해 성공적으로 실행되었다. - SQL_PARAM_SUCCESS_WITH_INFO: SQL 명령문이 이 매개 변수 집합에 대해 성공적으로 실행되었지만 진단 데이터 구조체에 경고 정보가 있다. - SQL_PARAM_ERROR: 매개 변수 집합을 처리하는데 오류가 발생했다. 추가 오류 정보는 진단 데이터 구조체에 있다. - SQL_PARAM_UNUSED: 일부 이전 매개 변수 집합에 대한 처리가 중단되거나, SQL_ATTR_PARAM_OPERATION_PTR이 명시한 배열의 매개 변수 집합이 SQL_PARAM_IGNORE로 설정됨에 따라 매개 변수 집합이 사용되지 않았다. - SQL_PARAM_DIAG_UNAVAILABLE: 오류 정보 레벨을 생성하지 않기 때문에 드라이버가 매개 변수 배열을 하나의 단위로 취급한다. 이 명령문 속성은 NULL 포인터로 설정될 수 있고, 이 경우 드라이버는 매개 변수의 상태값들을 반환하지 않는다. 이 속성은 언제든지 설정할 수 있다. 그러나 새로운 값은 다음 SQLExecDirect나 SQLExecute가 호출될 때까지 사용되지 않는다. 이 속성을 설정하면 드라이버에 매개 변수를 출력하는 행동에 영향을 미칠 수 있음을 알린다. 이 명령문 속성을 설정하여 IPD 헤더의 SQL_DESC_ARRAY_STATUS_PTR 필드를 설정한다.
SQL_ATTR_PARAMS_PROCESSED_PTR	SQLULEN*: 처리된 매개 변수 집합의 개수를 반환하는 버퍼를 가리키는 레코드 필드로써 오류 집합을 포함한다. NULL 포인터일 경우 개수를 반환하지 않는다. 이 명령문 속성을 설정하여 IPD 헤더의 SQL_DESC_ROWS_PROCESSED_PTR 필드를 설정한다.

속성	ValuePtr 내용
(ODBC 3.0)	다. 이 속성에 표시된 버퍼를 채울 SQLExecDirect 또는 SQLExecute가 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않는다.
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	SQLULEN: 각 매개 변수 값의 개수를 명시한 값이다. SQL_ATTR_PARAMSET_SIZE가 1보다 크다면 APD의 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR은 배열을 가리킨다. 각각의 배열 상수는 필드의 값과 같다. 바인딩 된 매개 변수가 없을 경우, 이 속성은 무시된다. 이 명령문 속성을 설정하여 APD 헤더의 SQL_DESC_ARRAY_SIZE 필드를 설정한다.
SQL_ATTR_PREFETCH_ROWS	이 속성은 한 번의 fetch 작업으로 드라이버가 미리 가져오는 (prefetch) 행의 개수를 지정하는 SQLULEN 타입의 값이다. 기본값 (default)은 1000이다. 이 값을 증가시키면 네트워크 왕복 (network round-trip) 횟수가 줄어들어 성능이 향상될 수 있지만, 메모리 사용량은 증가할 수 있다. 반대로 이 값을 줄이면 메모리 사용량은 줄어들지만 네트워크 왕복 횟수가 증가할 수 있다. 이 속성은 SQLExecute 또는 SQLExecDirect 이후에도 설정할 수 있으며, 이후 SQLFetch나 SQLFetchScroll을 호출할 때 설정된 값만큼 행을 미리 가져온다 (prefetch). 이미 fetch가 진행된 statement handle이라도, 다음 fetch 작업부터는 새로운 설정값이 적용된다.
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	SQLULEN: SQL 명령문을 실행하고 응용 프로그램으로 반환되기 전에 기다리는 초단위의 값이다. ValuePtr이 0 (기본값)일 경우, 타임아웃은 없다. 명시된 타임아웃 값이 데이터 소스의 최대값을 초과하거나 최소값보다 작을 경우, SQLSetStmtAttr은 값을 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. SELECT 명령문이 응용 프로그램에서 타임아웃되더라도 명령문을 재사용할 때 SQLCloseCursor를 호출할 필요가 없다. 이 명령문 속성의 쿼리 타임아웃을 설정하는 것은 동기방식과 비동기방식 모두에 유효하다.
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	SQLULEN: SQLFetch나 SQLFetchScroll을 각각 호출하고 반환되는 row들의 개수를 명시한 값이다. 이것은 SQLBulkOperations에서 벌크 북마크 연산에 쓰이는 북마크 배열의 row 개수이기도 하다. 기본값은 1이다. 명시한 row 집합의 크기가 데이터 소스에서 지원하는 row 집합 크기의 최대값을 초과할 경우, 드라이버는 값을 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. 이 명령문 속성을 설정하여 ARD 헤더의 SQL_DESC_ARRAY_SIZE 필드를 설정한다.
	SQLULEN: Column 데이터의 바인딩 변경을 표시하기 위해 추가된 offset을 가리키는 값

속성	ValuePtr 내용
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	이다. 이 필드가 NULL이 아닐 경우, 드라이버는 포인터를 역참조하여 설명자 레코드 (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR)에 거치한 필드들 각각에 역참조 값을 추가하고 바인딩 될 때 새로운 포인터 값을 사용한다. 기본적으로 NULL값으로 설정한다. 이 명령문 속성을 설정하여 ARD 헤더의 SQL_DESC_BIND_OFFSET_PTR 필드를 설정한다.
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	SQLULEN: SQLFetch 또는 SQLFetchScroll을 관련된 명령문에서 호출할 때 바인딩 방향을 설정하는 값이다. SQL_BIND_BY_COLUMN 값을 설정하면 column 방향 바인딩이 선택된다. 구조체나 결과 column들이 바인딩 될 버퍼의 인스턴스에 대한 길이를 설정하면 row 방향 바인딩이 선택된다. 길이가 명시되면 바인딩 된 모든 column들과 바인딩 column의 주소가 특정 길이만큼 증가했을 때 구조체나 버퍼 패딩을 위한 공간을 포함해야 한다. ANSI C의 구조체 또는 공용체에 sizeof 연산자를 사용할 때 이 행동이 보장된다. Column 방향 바인딩은 SQLFetch와 SQLFetchScroll의 기본 바인딩 방향이다. 이 명령문 속성을 설정하여 ARD 헤더의 SQL_DESC_BIND_TYPE 필드를 설정한다.
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	SQLULEN: 전체 결과 집합에서 현재 row의 순서이다. 현재 row의 개수를 결정할 수 없거나 현재 row가 존재하지 않을 경우, 드라이버는 0을 반환한다. 이 속성은 SQLGetStmtAttr을 호출하여 조회할 수 있다. SQLSetStmtAttr을 호출하여 설정할 수는 없다.
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	SQLUSMALLINT*: SQLFetch나 SQLFetchScroll을 호출한 뒤 row 상태값들을 포함하는 SQLUSMALLINT 배열값을 가리키는 값이다. 배열은 row 집합에 포함된 row들의 개수만큼의 요소들을 가지고 있다. 이 명령문 속성은 NULL 포인터로 설정될 수 있고 이러한 경우 드라이버는 row 상태값들을 반환하지 않는다. 이 속성은 언제든지 설정할 수 있지만 다음 SQLBulkOperations, SQLFetch, SQLFetchScroll 또는 SQLSetPos를 호출할 때까지 새로운 값이 사용되지 않는다. 이 명령문 속성을 설정하여 IRD 헤더의 SQL_DESC_ARRAY_STATUS_PTR 필드를 설정한다. 이 속성은 ODBC 2.x 드라이버에서 SQLExtendedFetch의 rgbRowStatus 배열에 매핑된다.
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	SQLULEN*: SQLFetch나 SQLFetchScroll을 호출한 뒤에 회수된 row들의 개수를 반환하는 버퍼를 가리킨다. Operation 인자에 SQL_REFRESH 설정하여 SQLSetPos를 호출하거나 SQLBulkOperations를 통한 벌크 연산들의 실행에 따라 row의 개수가 결정된다. 이 row의 개수는 오류 row들을 포함한다. 이 명령문 속성을 설정하여 IRD 헤더의 SQL_DESC_ROWS_PROCESSED_PTR 필드를 설정한다.

속성	ValuePtr 내용
	다. 이 속성이 가리키는 버퍼를 채우는 SQLFetch나 SQLFetchScroll을 호출했을 때 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용들은 정의되지 않는다.
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ROWSET_SIZE (ODBC 2.0)	이 속성은 SQLExtendedFetch 호출 한 번으로 반환되는 행의 개수, 즉 rowset의 크기를 지정하는 32비트 정수 (integer) 타입의 값이다. 기본값 (default)은 1이다. 이 값을 1보다 크게 설정하면 한 번의 SQLExtendedFetch 호출로 여러 개의 행을 한꺼번에 가져올 수 있어 애플리케이션에서 대량의 데이터를 효율적으로 처리할 수 있다. 이 속성은 커서 (cursor)가 열린 (open) 상태에서도 설정할 수 있다. 이 속성은 SQLExtendedFetch 함수에만 적용되며, SQLFetch나 SQLFetchScroll 함수의 rowset 크기를 지정하려면 SQL_ATTR_ROW_ARRAY_SIZE 속성을 사용해야 한다.

SQLGetStmtOption

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

SQLGetStmtOption 함수는 ODBC 3.x에서 SQLGetStmtAttr 함수로 대체되었다.
자세한 내용은 **SQLGetStmtAttr**을 참조한다.

SQLGetSuitableGroupID

적합성

표준 준수: 없음

개요

SQLGetSuitableGroupID는 준비된 명령문이 클러스터 시스템에서 최적으로 수행할 수 있는 클러스터 그룹의 ID를 반환한다.

구문

```
SQLRETURN SQLGetSuitableGroupID(  
    SQLHSTMT      StatementHandle,  
    SQLINTEGER * GroupIDPtr );
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

GroupIDPtr

[출력] 클러스터 그룹의 ID이다.

반환

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간 연결에 실패했다.
HY010	Function sequence error	이 함수가 SQLPrepare 이전에 호출되었다. 이 함수는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정된 상태에서 호출되어야 한다.

설명

SQLGetSuitableGroupID는 SQL_ATTR_LOCALITY_AWARE_TRANSACTION 연결 속성이 설정되어 있고 명령문이 준비된 상태에서만 호출할 수 있다.

SQLGetSuitableGroupID는 명령문에 매개 변수가 있을 경우, 매개 변수 마커의 현재값을 사용하여 준비된 명령문이 클러스터 시스템에서 최적으로 수행할 수 있는 클러스터 그룹의 ID를 반환한다. 만약 최적으로 수행할 수 있는 클러스터 그룹을 결정할 수 없을 경우 SQL_INVALID_GROUP_ID(-1) 를 반환한다.

SQLGetTypeInfo

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLGetTypeInfo는 데이터 소스가 지원하는 데이터 타입에 대한 정보를 반환한다. 드라이버는 SQL 결과 집합의 형태로 정보를 반환한다. 데이터 타입은 데이터 정의 언어 (DDL) 명령문에 사용하기 위한 것이다.

구문

```
SQLRETURN SQLGetTypeInfo(  
    SQLHSTMT      StatementHandle,  
    SQLSMALLINT   DataType);
```

인자

StatementHandle

[입력] 결과 집합에 대한 명령문 핸들이다.

DataType

[입력] SQL 데이터 타입이다. 데이터 타입이나 지정된 드라이버 SQL 데이터 타입이다. SQL_ALL_TYPES는 모든 데이터 타입에 대한 정보가 반환되도록 지정한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
		구현 작업 조건 때문에 지정된 명령문 속성이 유효하지 않아서 비슷한 값이 일시적으로 대체되었다. 대체값은 커서가 닫힐 때까지 StatementHandle에 유효하다.

SQLSTATE	Error	설명
01S02	Option value changed	변경될 수 있는 명령문 속성은 다음과 같다. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	StatementHandle에 커서가 열려있고, SQLFetch 또는 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 결과 집합이 열려있지만 SQLFetch나SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수가 실행되는 도중에 관련된 연결이 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY004	Invalid SQL data type	DataType 인자에 지정된 값이 드라이버가 지원하는 ODBC SQL 데이터 타입 식별자가 아니고 드라이버별 데이터 타입 식별자도 아니다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행되는 함수를 호출했고 SQLGetStmtAttr이 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.

SQLSTATE	Error	설명
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 <code>SQLEndTran</code> 함수를 참조한다.
HYC00	Optional field not implemented	드라이버나 데이터 소스가 <code>SQL_ATTR_CONCURRENCY</code> 와 <code>SQL_ATTR_CURSOR_TYPE</code> 명령문 속성의 현재 설정에 대한 조합을 지원하지 않는다. <code>SQL_ATTR_USE_BOOKMARKS</code> 명령문 속성이 <code>SQL_UB_VARIABLE</code> 에 설정되지 않았고 <code>SQL_ATTR_CURSOR_TYPE</code> 명령문 속성이 드라이버가 지원하지 않는 북마크를 위한 커서 타입으로 설정되었다.
HYT00	Timeout expired	데이터 소스가 결과 집합을 반환하기 전에 쿼리 타임 아웃 기간이 만료되었다. 이 제한 시간은 <code>SQLSetStmtAttr</code> 의 <code>SQL_ATTR_QUERY_TIMEOUT</code> 을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 <code>SQLSetStmtAttr</code> 의 <code>SQL_ATTR_CONNECTION_TIMEOUT</code> 을 통해 설정할 수 있다.
IM001	Driver does not support this function	<code>DescriptorHandle</code> 과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in a synchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
Im018	<code>SQLCompleteAsync</code> has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 <code>SQL_STILL_EXECUTING</code> 을 반환하고 알림 모드가 활성화되면, 핸들에 대한 후 처리와 작업 완료를 위해 반드시 <code>SQLCompleteAsync</code> 를 호출해야 한다.

설명

`SQLGetTypeInfo`는 표준 결과 집합과 같은 결과를 반환하는데, 이는 `DATA_TYPE`, ODBC SQL 데이터 타입에 대응하는 데이터 타입과 밀접하게 매핑되도록 정렬되어 있다. 데이터 소스가 정의한 데이터 타입은 사용자 정의 데이터 타입보다 우선한다. 따라서 정렬 순서는 반드시 일치하지 않지만, `DATA_TYPE`을 첫 번째로 하고 이어서 `TYPE_NAME` 순서의 오름차순으로 일반화될 수 있다.

예를 들어, 데이터 소스가 `INTEGER`와 `COUNTER` 데이터 타입을 정의했고 `COUNTER`는 자동 증가하며, 사용자 정의 데이터 타입 `WHOLENUM`이 정의되었을 경우, `INTEGER`, `WHOLENUM` 그리고 `COUNTER` 순서로 반환될 것이다. 이 결과는 `WHOLENUM`이 ODBC SQL 데이터 타입 `SQL_INTEGER`와 밀접하게 매핑되기 때문이다. 반면에, 데이터 소스가 지원하더라도 자동 증가 데이터 타입은 ODBC SQL 데이터 타입과 밀접하게 매핑되지 않는다.

`DataType` 인자가 드라이버가 지원하는 ODBC 버전에서 유효하더라도 드라이버가 지원하지 않을 경우, 빈 결과 집합

이 반환된다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column 번호로 바인딩하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
MONEY	FIXED_PREC_SCALE
AUTO_INCREMENT	AUTO_UNIQUE_VALUE

다음 column은 ODBC 3.x에서 SQLGetTypeInfo가 반환하는 결과 집합에 추가되었다.

- SQL_DATA_TYPE
- INTERVAL_PRECISION
- SQL_DATETIME_SUB
- NUM_PREC_RADIX

다음 표에는 결과 집합의 column들이 나열되어 있다. Column 19 (INTERVAL_PRECISION) 이후에 추가된 row는 드라이버가 정의할 수 있다. 응용 프로그램은 명시적으로 서수 위치를 지정하기보다는 결과 집합의 끝에서부터 카운트 다운하여 특정 드라이버 column에 접근해야 한다.

노트

SQLGetTypeInfo는 모든 데이터 타입에 대해 반환되지 않을 수 있다. 예를 들어, 드라이버가 사용자 정의 데이터 타입을 반환하지 않을 수 있다. 응용 프로그램은 SQLGetTypeInfo의 반환 여부에 관계없이 유효한 데이터 타입을 사용할 수 있다. SQLGetTypeInfo가 반환하는 데이터 타입은 데이터 소스가 지원하는 것들이다. 그것들은 데이터 정의 언어 (DDL) 명령문에 사용하기 위한 것이다. 드라이버는 SQLGetTypeInfo에서 반환되는 타입보다 결과 집합 데이터에서 사용하는 데이터 타입을 반환할 수 있다. 카탈로그 함수를 위한 결과 집합을 생성할 때 드라이버는 데이터 소스에서 지원하지 않는 데이터 타입을 사용할 수도 있다.

Column name	Column number	Data type	Comments
TYPE_NAME (ODBC 2.0)	1	Varchar not NULL	데이터 소스 종속 데이터 타입 이름이다. CHAR(), VARCHAR(), MONEY, LONG VARBINARY 또는 CHAR () FOR BIT DATA와 같은 것들이 있다. 응용 프로그램은 CREATE TABLE과 ALTER TABLE 명령문에서 반드시 이 이름을 사용해야 한다.
DATA_TYPE (ODBC 2.0)	2	Smallint not NULL	SQL 데이터 타입이다. 이것은 ODBC SQL 데이터 타입이나 특정 드라이버 SQL 데이터 타입이 될 수 있다. 이 column은 DATETIME 또는 INTERVAL 데이터 타입에 대해 (SQL_TYPE_TIME 또는 SQL_INTERVAL_YEAR_TO_MONTH 같은) 간결한 데이터 타입을 반환한다.
			서버가 지원하는 데이터 유형의 최대 column 크기이다. 숫자형 데이터 타

Column name	Column number	Data type	Comments
COLUMN_SIZE (ODBC 2.0)	3	Integer	입은 최대 precision, 문자열 데이터 타입은 문자들의 길이, DATETIME 데이터 타입은 표현되는 문자 길이, INTERVAL 데이터 타입은 INTERVAL 그 대로의 문자 표현에 대한 문자 길이, column 사이즈를 적용할 수 없는 데이터 타입에 대해 NULL을 반환한다.
LITERAL_PREFIX (ODBC 2.0)	4	Varchar	문자나 문자열이 접두사로 사용된다. 예를 들어 단일 인용 부호 (')는 문자 데이터 타입이나 0x 또는 바이너리 데이터 타입을 위한 것이다. 접두사로 사용할 수 없는 데이터 타입에 대해서는 NULL이 반환된다.
LITERAL_SUFFIX (ODBC 2.0)	5	Varchar	문자나 문자열이 종료 문자로 사용된다. 예를 들어서 단일 인용 부호 (')는 문자 데이터 타입을 위한 것이다. 접미사로 사용할 수 없는 데이터 타입에 대해서는 NULL이 반환된다.
CREATE_PARAMS (ODBC 2.0)	6	Varchar	TYPE_NAME에 반환되는 이름을 사용하는 경우, 응용 프로그램이 괄호 안에 지정하는 (콤마로 구분된) 각 매개 변수에 대응하는 키워드 목록이다. 목록의 키워드는 length, precision, 또는 scale이다. 이것들은 문법이 키워드를 사용하는 순서대로 나타난다. 예를 들어서, NUMBER에 대한 CREATE_PARAMS는 "precision,scale"이고 VARCHAR에 대한 CRATE_PARAMS는 "length"이다. 만약 데이터 타입 정의를 위한 매개 변수가 없을 경우, NULL이 반환된다 (예: INTERGER). 드라이버는 CREATE_PARAMS 텍스트를 사용하는 국가/ 지역의 언어로 제공한다.
NULLABLE (ODBC 2.0)	7	Smallint not NULL	데이터 타입이 NULL 값을 받아들이는지 여부이다. SQL_NO_NULLS일 경우, 데이터 타입이 NULL 값을 받아들이지 못한다. SQL_NULLABLE일 경우, 데이터 타입이 NULL 값을 받아들이는. SQL_NULLABLE_UNKNOWN일 경우, column이 NULL 값을 받아들이는 지 여부를 알 수 없다.
CASE_SENSITIVE (ODBC 2.0)	8	Smallint not NULL	문자 데이터 타입이 정렬과 비교를 수행할 때 대소문자를 구분하는지 여부이다. 데이터 타입이 문자 데이터 타입이고 대소문자를 구분하면 SQL_TRUE이다. 문자 데이터 타입이 아니거나 대소문자를 구분하지 않으면 SQL_FALSE이다.
SEARCHABLE (ODBC 2.0)	9	Smallint not NULL	WHERE 절 안에서 데이터 타입이 어떻게 사용되는지를 나타낸다. WHERE 절 안에서 column이 사용될 수 없으면 SQL_PRED_NONE이다. (ODBC 2.x에서 SQL_UNSEARCHABLE 값과 동일하다.) WHERE 절 안에서 column이 사용될 수 있지만 오직 LIKE 조건과 함께인 경우에만 사용될 경우, SQL_PRED_CHAR이다. (ODBC 2.x에서 SQL_LIKE_ONLY 값과 동일하다.) WHERE 절 안에서 LIKE 조건을 제외한 모든 비교 연산자와 함께 column이 사용될 수 있을 경우, SQL_PRED_BASIC이다. (ODBC 2.x에서 SQL_ALL_EXCEPT_LIKE 값과 동일하다.) WHERE 절 안에서 모든 연산자와 column이 함께 사용될 수 있을 경우, SQL_SEARCHABLE이다.
UNSIGNED_ATTR	10	Smallint	데이터 타입에 부호가 있는지 여부이다.

Column name	Column number	Data type	Comments
TRIBUTE (ODBC 2.0)			데이터 타입에 부호가 없으면 SQL_TRUE이다. 데이터 타입에 부호가 있으면 SQL_FALSE이다. 데이터 타입에 속성을 사용할 수 없거나 데이터 타입이 숫자가 아니면 NULL이 반환된다.
FIXED_PRECISION_SCALE (ODBC 2.0)	11	Smallint not NULL	데이터 타입이 (데이터 소스 별) 고정 precision과 scale을 미리 정의하여 가지는지 여부이다. 미리 정의된 고정 precision과 scale을 가질 경우, SQL_TRUE이다. 미리 정의된 고정 precision과 scale을 가지지 않을 경우, SQL_FALSE이다.
AUTO_UNIQUE_VALUE (ODBC 2.0)	12	Smallint	데이터 타입이 자동으로 증가하는지 여부이다. 데이터 타입이 자동 증가하면 SQL_TRUE이다. 데이터 타입이 자동 증가하지 않으면 SQL_FALSE이다. 속성이 데이터 타입에 사용될 수 없거나 데이터 타입이 숫자가 아니면 NULL이 반환된다. 응용 프로그램은 이 속성을 갖는 column에 값을 입력할 수 있지만, 전형적으로 column의 값을 갱신할 수 없다. 자동 증가 column에 삽입이 발생하면, column에 고유한 값이 입력된다. 증가는 정의되지 않고 데이터 소스에 따라 다르다. 응용 프로그램은 자동 증가 column이 어느 특정 지점에서 시작하거나 특정 값에 의해 증가할 것을 기대하지 말아야 한다.
LOCAL_TYPE_NAME (ODBC 2.0)	13	Varchar	데이터 소스에 종속된 데이터 타입 이름의 지역화된 버전이다. 데이터 소스가 지역화된 이름을 지원하지 않으면 NULL이 반환된다. 이 이름은 대화 상자처럼 표시하는데 쓰인다.
MINIMUM_SCALE (ODBC 2.0)	14	Smallint	데이터 소스에서 데이터 타입의 최소 scale이다. 데이터 타입이 고정 scale이면, MINIMUM_SCALE과 MAXIMUM_SCALE column은 이 값을 갖는다. 예를 들어, SQL_TYPE_TIMESTAMP column은 소수 초에 대한 고정 scale을 갖는다. Scale이 사용될 수 없으면 NULL이 반환된다.
MAXIMUM_SCALE (ODBC 2.0)	15	Smallint	데이터 소스에서 데이터 타입의 최대 scale이다. Scale을 사용할 수 없을 경우, NULL이 반환된다. 최대 scale은 데이터 소스에 따로 정의되지 않았지만 최대 precision과 동일하게 정의된 경우, 이 column은 COLUMN_SIZE column과 동일한 값을 갖는다.
SQL_DATA_TYPE (ODBC 3.0)	16	Smallint NOT NULL	설명자의 SQL_DESC_TYPE 필드에 나타나는 SQL 데이터 타입의 값이다. 이 column은 INTERVAL과 DATETIME 데이터 타입을 제외한 DATA_TYPE column과 동일하다. INTERVAL과 DATETIME 데이터 타입에 대해 결과 집합에서 SQL_DATA_TYPE 필드는 SQL_INTERVAL 또는 SQL_DATETIME을 반환하고, SQL_DATETIME_SUB 필드는 INTERVAL 또는 DATETIME 데이터 타입을 위한 하위 코드를 반환한다.
SQL_DATETIME_SUB (ODBC 3.0)	17	Smallint	한 개의 SQL_DATA_TYPE 값이 SQL_DATETIME이나 SQL_INTERVAL인 경우, 이 column은 DATETIME/ INTERVAL 하위 코드를 갖는다. 다른 데이터 타입에 대하여 이 필드는 NULL이다. INTERVAL 또는 DATETIME 데이터 타입에 대해 결과 집합의 SQL_DATA_TYPE은 SQL_INTERVAL이나 SQL_DATETIME을 반환하고 SQL_DATETIME_SUB 필드는 INTERVAL이나 DATETIME 데이터 타입에 대해 하위 코드를 반환한다.

Column name	Column number	Data type	Comments
NUM_PREC_RADIX (ODBC 3.0)	18	Integer	데이터 타입이 근사 숫자 타입이면, 이 column은 COLUMN_SIZE가 비트의 수를 지정하는 것을 나타내기 위한 값 2를 갖는다. 정확한 숫자 타입에 대해, 이 column은 COLUMN_SIZE가 십진수를 지정하는 것을 나타내기 위한 값 10을 갖는다. 그렇지 않다면 이 column은 NULL이다.
INTERVAL_PRECISION (ODBC 3.0)	19	Smallint	INTERVAL 데이터 타입에 대해 이 column은 INTERVAL leading precision의 값을 갖는다. 그렇지 않다면 NULL이다.

속성 정보는 데이터 타입 또는 결과 집합에서 특정 column에 적용할 수 있다. SQLGetTypeInfo는 데이터 타입과 관련된 속성에 대한 정보를 반환한다. SQLColAttribute는 결과 집합에서 column과 관련된 속성에 대한 정보를 반환한다.

SQLMoreResults

적합성

도입된 버전: ODBC 1.0
표준 준수: ODBC

개요

SQLMoreResults는 SELECT, UPDATE, INSERT 또는 DELETE를 포함한 명령문에서 더 많은 결과를 사용할 수 있는지 확인하고, 가능할 경우 그 결과에 대한 처리를 초기화한다

구문

```
SQLRETURN SQLMoreResults(  
    SQLHSTMT      StatementHandle);
```

인자

StatementHandle
[입력] 명령문 핸들이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	일괄 처리되고 있는 동안 명령문 속성의 값이 갱신되었다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수가 실행되는 동안 관련된 연결이 실패하였고 트랜잭션의 상태를 확인할 수 없다.

SQLSTATE	Error	설명
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램의 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle이 호출되었다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했는데, SQLMoreResults가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수를 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	인자로 사용된 버퍼의 크기 값이 0보다 작거나 메모리에 접근할 수 없다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	DescriptorHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in a synchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SELECT 문은 결과 집합을 반환한다. UPDATE, INSERT, DELETE 문은 영향받는 row의 개수를 반환한다.

이런 명령문 중 하나가 일괄 처리되거나 매개 변수의 배열이 전달되었거나 처리 중인 상태일 경우, 다수의 결과 집합이나 row의 개수가 반환될 수 있다.

일괄 처리가 수행된 후 응용 프로그램은 첫 번째 결과 집합에 위치한다. 응용 프로그램은 처음 또는 후속 결과 집합에 대해 단일 결과 집합이 있었던 것처럼 SQLBindCol, SQLBulkOperations, SQLFetch, SQLGetData, SQLFetchScroll, SQLSetPos와 모든 메타 데이터 함수를 호출할 수 있다. 첫 번째 결과 집합에서 SQLMoreResults가 수행될 때 응용 프로그램은 다음 결과 집합으로 이동하기 위해 SQLMoreResults를 호출한다. 만약 다른 결과 집합이나 개수를 사용할 수 있으면 SQLMoreResults는 SQL_SUCCESS를 반환하고 결과 집합을 초기화하거나 추가 작업을 집계한다. 카운트 row 생성 명령문이 결과 집합 생성 명령문 사이에 나타나면 row 생성 명령문은 SQLMoreResults 호출로 넘어갈 수 있다. 응용 프로그램은 UPDATE, INSERT, 또는 DELETE 명령문에 대해 SQLMoreResult를 호출한 후에 SQLRowCount를 호출할 수 있다.

열람되지 않은 row의 현재 결과 집합이 있을 경우, SQLMoreResults는 그 결과 집합을 버리고 다음 결과 집합을 만들거나 집계한다. 모든 결과 집합이 처리 되었을 경우, SQLMoreResults는 SQL_NO_DATA를 반환한다. 일부 드라이버의 경우, 출력 매개 변수와 반환값은 모든 결과 집합과 row 집계 처리되기 전까지 사용할 수 없다. 이러한 드라이버의 경우, 출력 매개 변수와 반환값은 SQLMoreResults가 SQL_NO_DATA를 반환한 후에 사용할 수 있다.

이전 결과 집합에 대해 만들어진 모든 바인딩은 여전히 유효한 상태로 유지된다. Column 구조가 이 결과 집합과 다를 경우, SQLFetch나 SQLFetchScroll을 호출했을 때 에러나 잘림이 발생할 수 있다. 이것을 방지하려면 응용 프로그램이 SQLBindCol을 호출하여 명시적으로 적절하게 다시 바인드해야 한다. 응용 프로그램은 모든 column 버퍼의 바인딩을 해제하기 위해 SQL_UNBIND 옵션과 함께 SQLFreeStmt을 호출할 수 있다.

커서 타입, 커서 동시성, 키 집합 크기, 최대 길이와 같은 명령문 속성의 값은 응용 프로그램이 SQLMoreResults를 호출하여 일괄 처리하는 동안 갱신될 수 있다. 이 경우, SQLMoreResults는 SQL_SUCCESS_WITH_INFO와 SQLSTATE 01S02 (option value has changed)를 반환한다.

SQL_CLOSE option을 사용하여 SQLCloseCursor나 SQLFreeStmt를 호출하면 모든 결과 집합과 배치 실행 결과와 같은 사용가능한 row 개수를 폐기한다. 명령문 핸들은 할당된 상태 또는 준비 (prepared)된 상태 중 하나를 반환한다. 배치가 실행되고 명령문 핸들이 실행되었을 때 비동기 수행 함수를 취소하기 위해 SQLCancel을 호출하고 성공적으로 끝난 경우, 배치로 인해 생성된 모든 결과 집합의 비동기적 상태 결과들과 row 개수를 폐기한다. 이 명령문은 준비 또는 할당 상태를 반환한다.

명령문의 일괄처리 또는 절차가 SELECT, UPDATE, INSERT, DELETE 문과 다른 SQL 명령문을 섞어서 사용할 경우, 이러한 다른 명령문은 SQLMoreResult에 영향을 미치지 않는다.

명령문의 일괄처리에서 검색되던 갱신, 추가 또는 삭제 명령문이 데이터 소스의 어떤 row에도 영향을 미치지 않을 경우, SQLMoreResults는 SQL_SUCCESS를 반환한다. 이것은 SQLExecDirect, SQLExecute, 또는 SQLParamData가 이런 경우에 SQL_NO_DATA를 반환하는 것과 다르다. SQLMoreResults가 어떤 row에도 영향을 미치지 않는 상태에서 응용 프로그램이 row 개수를 검색하기 위해 SQLRowCount를 호출할 경우, SQLRowCount는 SQL_NO_DATA를 반환한다.

Row 개수의 가용성

일괄 처리에 연속된 여러 개의 row 집계 생성 명령문이 포함될 경우, 이러한 row 집계는 하나의 row 개수로 rollup 처리 된다. 예를 들어, 일괄 처리가 다섯 개의 INSERT 명령문을 가질 경우, 특정 데이터 소스는 다섯 개의 개별 row 개수를 반환할 수 있다. 일부 다른 데이터 소스는 다섯 개의 개별 row 개수의 합계를 표현하는 오직 하나의 row 개수를 반환한다.

일괄 처리에 결과 집합을 생성하고 row 집계를 생성하는 명령문의 조합이 포함될 경우, row 개수는 전혀 사용할 수 없게 될 수 있다.

Row 개수의 가용성에 따른 드라이버 동작은 SQLGetInfo를 호출하여 사용할 SQL_BATCH_ROW_COUNT 정보 타입에 나열된다. 예를 들어, 일괄 처리가 두 개의 INSERT 문과 다른 SELECT 문이 뒤에 따르는 SELECT 문을 포함할 경우 다음과 같이 동작할 수 있다.

- 두 개의 INSERT 문에 대응되는 row 개수는 전혀 사용할 수 없다. SQLMoreResults의 첫 번째 호출은 두 번째 SELECT 문의 결과 집합에 위치한다.
- 두 개의 INSERT 문에 대응하는 row 개수는 개별적으로 사용할 수 있다. (SQLGetInfo는 SQL_BATCH_ROW_COUNT 정보 타입에 대해 SQL_BRC_ROLLED_UP 비트를 반환하지 않는다.) 첫 번째 SQLMoreResults 호출은 첫 번째 INSERT 문의 row 개수에 위치하고, 두 번째 호출은 두 번째 INSERT 문의 row 개수에 위치한다. 세 번째 호출은 두 번째 SELECT 문의 결과 집합에 위치한다.
- 두 INSERT 문에 대응하는 row 개수는 사용 가능한 단일 row 개수로 rollup 된다. (SQLGetInfo 호출은 SQL_BATCH_ROW_COUNT 정보 타입에 대해 SQL_BRC_ROLLED_UP 비트를 반환한다.) 첫 번째 SQLMoreResults는 row 개수에 겹쳐져 위치하고, 두 번째 SQLMoreResults는 두 번째 SELECT 문의 결과 집합에 위치한다.

특정 드라이버는 명시적 일괄처리에만 사용 가능한 row 개수를 생성한다.

SQLNativeSql

지원하지 않는다.

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLNativeSql은 드라이버가 수정한 SQL 문자열을 반환한다. SQLNativeSql은 SQL 명령문을 실행하지 않는다.

구문

```
SQLRETURN SQLNativeSql(  
    SQLHDBC          ConnectionHandle,  
    SQLCHAR *        InStatementText,  
    SQLINTEGER        TextLength1,  
    SQLCHAR *        OutStatementText,  
    SQLINTEGER        BufferLength,  
    SQLINTEGER *      TextLength2Ptr);
```

SQLNumParams

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLNumParams는 SQL 명령문에서 매개 변수의 개수를 반환한다.

구문

```
SQLRETURN SQLNumParams(  
    SQLHSTMT      StatementHandle,  
    SQLSMALLINT * ParameterCountPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ParameterCountPtr

[출력] 명령문의 매개 변수 개수가 반환될 버퍼의 포인터이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
		StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 실행이 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle

SQLSTATE	Error	설명
HY008	Operation canceled	에서 호출되었다. 이 함수가 StatementHandle에서 다시 호출된다. Multi thread 응용 프로그램에서 이 함수가 호출된 후 완료되기 전에 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle이 호출되었다.
HY010	Function sequence error	이 함수는 StatementHandle에서 SQLPrepare나 SQLExecDirect를 호출하기 전에 호출되었다. StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했고 SQLNumParams가 호출 되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에서 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에서 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대해 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 핸들에 대한 후처리와 작업 완료를 위해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLNumParams는 SQLPrepare가 호출된 후에만 호출할 수 있다.

StatementHandle과 관련된 명령문이 매개 변수를 갖고 있지 않을 경우, SQLNumParams는 *ParameterCountPtr을 0으로 설정한다.

SQLNumParams가 반환하는 매개 변수의 개수는 IPD의 SQL_DESC_COUNT 필드와 동일한 값이다.

SQLNumResultCols

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLNumResultCols 결과 집합에서 column의 개수를 반환한다.

구문

```
SQLRETURN SQLNumResultCols(  
    SQLHSTMT      StatementHandle,  
    SQLSMALLINT * ColumnCountPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

ColumnCountPtr

[출력] 결과 집합에서 column의 개수가 반환될 버퍼의 포인터이다. 이 개수는 복마크 column을 포함하지 않는다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.

SQLSTATE	Error	설명
	or	
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 실행이 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에서 호출되었다. 이 함수가 StatementHandle에서 다시 호출된다. Multi thread 응용 프로그램에서 이 함수가 호출된 후 완료되기 전에 다른 thread로부터 StatementHandle에 SQLCancel이나 SQLCancelHandle이 호출되었다.
HY010	Function sequence error	StatementHandle에서 SQLPrepare 또는 SQLExecDirect를 호출하기 전에 이 함수가 호출되었다. StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했고 SQLNumResultsCols가 호출 되었을 때 여전히 이 비동기 수행 함수가 실행 중이었다. StatementHandle에서 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행중이었다. StatementHandle에서 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in a synchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대해 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 핸들에 대한 후처리와 작업 완료를 위해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLNumResultCols는 명령문이 준비, 실행, 위치 상태인 경우에만 성공적으로 호출된다.

StatementHandle과 관련된 명령문이 column을 반환하지 않을 경우, SQLNumResultCols는 *ColumnCountPtr을 0으로 설정한다.

SQLNumResultCols가 반환하는 column의 개수는 IRD의 SQL_DESC_COUNT 필드와 같은 값이다.

SQLParamData

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLParamData는 명령문이 수행될 때 매개 변수를 제공하기 위해 SQLPutData와 함께 사용된다.

구문

```
SQLRETURN SQLParamData(  
    SQLHSTMT      StatementHandle,  
    SQLPOINTER *  ValuePtrPtr);
```

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_NO_DATA, SQL_STILL_EXECUTING,
SQL_ERROR, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	이전 함수가 SQL_NEED_DATA를 반환하는 SQLExecDirect, SQLExecute가 아니다. 이전 함수가 SQLParamData이다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.

매개 변수에 대한 데이터를 전송하기 위해 SQLParamData를 호출할 경우, SQLExecute, SQLExecDirect의 SQLSTATE가 반환될 수 있다.

설명

응용 프로그램이 data-at-execution을 필요로 하는 SQLExecute나 SQLExecDirect를 호출할 때, 드라이버는 SQL_NEED_DATA를 반환한다. 응용 프로그램은 전송되는 데이터를 결정하기 위해 SQLParamData를 호출한다. 드라이버가 매개 변수 데이터를 필요로 하는 경우, 응용 프로그램이 입력한 *ValuePtr 값이 반환된다. 응용 프로그램은 드라이버가 요청하는 매개 변수 데이터를 확인하기 위해 이 값을 사용할 수 있다.

응용 프로그램은 data-at-execution 매개 변수를 보내기 위해, SQLPutData를 필요한만큼 호출한다. 매개 변수의 모든 데이터를 전송한 후, 응용 프로그램이 다시 SQLParamData를 호출한다. SQLParamData가 다시 SQL_NEED_DATA를 반환하는 경우, 응용 프로그램은 다시 SQLPutData를 호출하여 다른 매개 변수 데이터를 전송해야 한다. 만약 모든 매개 변수 데이터가 전송되었을 경우, SQLParamData는 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하며, *ValuePtr의 값은 정의되지 않고 SQL 문은 실행될 수 있다.

SQLParamOptions

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.0의 함수 SQLParamOptions는 ODBC 3.x에서 **SQLSetStmtAttr**로 대체되었다.

SQLPrepare

적합성

도입된 버전: ODBC 1.0
표준 준수: ISO 92

개요

SQLPrepare는 실행할 SQL 문자열을 준비한다.

구문

```
SQLRETURN SQLPrepare(  
    SQLHSTMT      StatementHandle,  
    SQLCHAR *      StatementText,  
    SQLINTEGER     TextLength);
```

인자

- StatementHandle**
[입력] 명령문 핸들이다.
- StatementText**
[입력] SQL 텍스트 문자열이다.
- TextLength**
[입력] 문자 *StatementText의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
		지정된 명령문 속성값이 작업을 실행하기에 적합하지 않아서 일시적으로 비슷한 값으로 대체되었다. (일시적으로 변경된 값이 무엇인지 확인하기 위해 SQLG

SQLSTATE	Error	설명
01S02	Option value changed	etStmtAttr을 호출할 수 있다.) 이 대체값은 커서가 닫힐 때까지 유효하며, 커서가 닫힐 때 이전 값으로 변경된다. 변경될 수 있는 명령문 속성은 다음과 같다. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
21S01	Insert value list does not match column list	*StatementText에 INSERT 문이 존재하며, 삽입되는 값의 개수가 파생된 테이블과 일치하지 않는다.
21S02	Degree of derived table does not match column list	*StatementText에 CREATE VIEW 문이 존재하며, 지정된 이름의 개수가 질의 명세에 의해 정의된 파생된 테이블과 일치하지 않는다.
22018	Invalid character value for cast specification	*StatementText에 문자열이나 매개 변수를 포함하는 SQL 명령문이 존재하며, 값이 관련된 테이블 column의 데이터 타입과 호환되지 않는다.
22019	Invalid escape character	StatementText 인자가 WHERE 절의 ESCAPE와 같은 LIKE 술어를 포함하며, ESCAPE의 제어 문자 길이가 1이 아니다.
22025	Invalid escape sequence	StatementText 인자가 WHERE 절에 LIKEpattern value ESCAPEescape character를 포함하고, 패턴값 제어 문자가 %나 -가 아니다.
24000	Invalid cursor state	StatementHandle에 커서가 열려있고, SQLFetch 또는 SQLFetchScroll 함수가 호출되었다.
34000	Invalid cursor name	*StatementText가 위치지정 DELETE 또는 위치지정 UPDATE를 포함하고, 준비된 명령문이 참조하는 커서가 열려있지 않다.
3D000	Invalid catalog name	StatementText에 지정된 카탈로그 이름이 유효하지 않았다.
3F000	Invalid schema name	StatementText에 지정된 스키마 이름이 유효하지 않았다.
42000	Syntax error or access violation	*StatementText가 구문 오류 또는 준비될 수 없는 SQL 명령문을 포함한다. *StatementText가 요구되는 권한을 갖지 않은 사용자를 명령문에 포함한다.
42S01	Base table or view already exists	*StatementText가 CREATE TABLE 또는 CREATE VIEW 문을 포함하고, 테이블 또는 view의 지정된 이름이 이미 존재한다.
42S02	Base table or view not found	*StatementText가 DROP TABLE 또는 DROP VIEW 문을 포함하고, 지정된 테이블 또는 view의 이름이 존재하지 않는다. *StatementText가 ALTER TABLE 문을 포함하고, 지정된 테이블 이름이 존재하지 않는다. *StatementText가 CREATE VIEW 문을 포함하고, 질의 명세에 정의된 테이블 또는 view의 이름이 존재하지 않는다. *StatementText가 CREATE INDEX 문을 포함하고, 지정된 테이블 이름이 존재하지 않는다. *StatementText가 GRANT 또는 REVOKE 문을 포함하고, 지정된 테이블 또는 view의 이름이 존재하지 않는다.

SQLSTATE	Error	설명
		*StatementText가 SELECT 문을 포함하고, 지정된 테이블 또는 view의 이름이 존재하지 않는다. *StatementText가 DELETE, INSERT 또는 UPDATE 문을 포함하고, 지정된 테이블 이름이 존재하지 않는다. *StatementText가 CREATE TABLE 문을 포함하고, 제약 조건이 지정된 테이블 (다른 테이블 참조)이 존재하지 않는다.
42S11	Index already exists	*StatementText에 CREATE INDEX 문이 존재하고, 지정된 INDEX 이름이 이미 존재한다.
42S12	Index not found	*StatementText에 DROP INDEX 문이 존재하고, 지정된 INDEX 이름이 존재하지 않는다.
42S21	Column already exist	*StatementText에 ALTER TABLE 문이 존재하고, ADD 절에 지정된 column이 유일하지 않거나 기본 테이블에 있는 기존 column을 식별한다.
42S22	Column not found	*StatementText에 CREATE INDEX 문이 존재하고, 하나 이상의 column 이름이 지정된 column 목록에 존재하지 않는다. *StatementText에 GRANT 또는 REVOKE 문이 존재하고, 지정된 column 이름이 존재하지 않는다. *StatementText에 SELECT, DELETE 또는 UPDATE 문이 존재하고, 지정된 column 이름이 존재하지 않는다. *StatementText에 CREATE TABLE 문이 존재하고, 제한 조건에 지정된 column이 존재하지 않는다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출되고 실행이 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에서 호출되었다. 이 함수가 StatementHandle에서 다시 호출된다. Multi thread 응용 프로그램에서 이 함수가 호출된 후 완료되기 전에 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle이 호출되었다.
HY009	Invalid use of null pointer	StatementText가 NULL 포인터였다.
HY010	Function sequence error	이 함수가 StatementHandle에서 SQLPrepare 또는 SQLExecDirect를 호출하기 전에 호출되었다. StatementHandle과 관련된 연결 핸들을 위해 비동기적으로 수행하는 함수를 호출했고 SQLPrepare가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에서 비동기적으로 수행하는 함수를 호출했고 이 함수가 호

SQLSTATE	Error	설명
		출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에서 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY090	Invalid string or buffer length	TextLength가 0 보다 작거나 같거나, 또는 SQL_NTS와 같지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional feature not implemented	정의된 커서의 타입에 동시성 설정이 유효하지 않다. SQL_ATTR_USE_BOOKMARKS 명령문 속성이 SQL_UB_VARIABLE로 설정되었고, SQL_ATTR_CURSOR_TYPE 명령문 속성이 드라이버가 지원하지 않는 북마크에 대한 커서 타입으로 설정되었다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대해 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 핸들에 대한 후 처리와 작업 완료를 위해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

응용 프로그램은 준비작업을 위해 SQLPrepare를 호출하여 SQL 명령문을 데이터 소스에 보낸다. 응용 프로그램은 SQL 명령문에 하나 이상의 매개 변수 표시 문자를 포함할 수 있다. 매개 변수 표시 문자를 포함하기 위해 응용 프로그램은 SQL 문자열의 적절한 위치에 물음표 (?)를 기술한다.

노트

응용 프로그램이 준비작업을 위해 SQLPrepare와 COMMIT 또는 ROLLBACK 문을 전달할 때 SQLExecute를 사용할 경우, DBMS 제품 간에 상호 운용할 수 없다.

드라이버는 데이터 소스가 사용하는 SQL 형식을 사용하기 위해 명령문을 수정할 수 있고, 준비작업을 위해 데이터 소스에 전달할 수도 있다. 특히, 드라이버는 특정 함수를 위한 SQL 구문을 정의하기 위해 사용되는 확장 비트열을 수정한다. 드라이버의 명령문 핸들은 embedded SQL 코드의 명령문 식별자와 비슷하다. 데이터 소스가 명령문 식별자를 지원할 경우, 드라이버는 명령문 식별자와 매개 변수 값을 데이터 소스에 전송할 수 있다.

명령문이 준비된 후에, 응용 프로그램은 이후에 호출되는 함수에서 명령문을 참조하기 위해 명령문 핸들을 사용한다. 응용 프로그램이 SQL_DROP 옵션으로 SQLFreeStmt를 호출하여 명령문을 해제하거나 명령문 핸들이 SQLPrepare, SQLExecDirect 또는 카탈로그 함수 (SQLColumns, SQLTables 등) 중 하나를 사용할 때까지 SQLExecute을 호출하여 명령문 핸들과 관련되어 준비된 명령문을 다시 실행할 수 있다. 일단 응용 프로그램이 명령문을 준비하면 결과 집합의 형식에 대한 정보를 요청할 수 있다. 일부 구현 작업의 경우, SQLPrepare한 후에 SQLDescribeCol이나 SQLDescribeParam을 호출하는 것은 SQLExecute나 SQLExecDirect 이후에 호출하는 것만큼 효율적이지 않을 수 있다.

드라이버는 응용 프로그램이 SQLPrepare를 호출할 때 구문 에러 또는 접근 위반을 반환할 수 없다. 드라이버는 구문 에러와 접근 위반을 모두 처리할 수도 있고, 구문 에러만 처리하거나, 구문 에러와 접근 위반 모두 처리하지 못할 수도 있다. 따라서 응용 프로그램은 후속 관련 함수 (SQLNumResultCols, SQLDescribeCol, SQLColAttribute, 그리고 SQLExecute 같은 후속 함수들)를 호출할 때 이러한 조건을 처리할 수 있어야 한다.

드라이버와 데이터 소스의 기능에 따라 (데이터 타입과 같은) 매개 변수 정보는 명령문이 (모든 매개 변수가 바인딩되었다면) 준비되거나 (모든 매개 변수가 바인딩 되지 않았다면) 실행될 때 검사할 수 있다. 응용 프로그램은 동일 명령문에 대한 새 SQL 문을 준비하기 전에 이전 SQL 문에 적용된 모든 매개 변수의 바인딩을 해제해야 한다. 이것은 이전 매개 변수 정보가 새로운 명령문에 적용되는 오류를 방지하기 위함이다.

주의

명시적으로 SQLEndTran을 호출하거나 자동 commit 모드로 작동하여 트랜잭션을 commit할 경우, 데이터 소스가 연결과 관련된 모든 명령문에 대한 접근 플랜을 삭제할 수 있다. 자세한 내용은 SQLGetInfo의 정보 타입 SQL_CURSOR_COMMIT_BEHAVIOR와 SQL_CURSOR_ROLLBACK_BEHAVIOR를 참조한다.

SQLPrimaryKeys

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLPrimaryKeys는 테이블의 기본 키로 구성된 column의 이름을 반환한다. 드라이버는 결과 집합으로 정보를 반환한다. 이 함수는 단일 호출을 통해 여러 테이블로부터 기본 키를 반환하는 것을 지원하지 않는다.

구문

```
SQLRETURN SQLPrimaryKeys(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] 카탈로그 이름이다. 드라이버가 일부 테이블에 대한 카탈로그만 지원할 경우, 빈 문자열 (")은 카탈로그가 없는 테이블을 나타낸다. CatalogName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 인수 문자열로 취급되며, 대소문자를 구분한다.

NameLength1

[입력] *CatalogName의 문자열 길이이다.

SchemaName

[입력] 스키마 이름이다. 드라이버가 일부 테이블에 대한 스키마만 지원할 경우, 빈 문자열 (")은 스키마가 없는 테이블을 나타낸다. SchemaName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 패턴값 문자열로 취급되며, 대소문자를 구분하지 않는다.

NameLength2

[입력] *SchemaName의 문자들 길이이다.

TableName

[입력] 테이블 이름이다. 이 인자는 null 포인터가 될 수 없다. TableName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정될 경우, TableName은 식별자로 간주되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정될 경우, TableName은 일반 문자열로 취급되며, 대소문자를 구분하지 않는다.

NameLength3

[입력] *TableName의 문자들 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE.

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll을 호출하였고 커서가 열려있다. SQLFetch, SQLFetchScroll을 호출하지 않았는데 커서가 열려있다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수를 실행하는 도중에 관련된 연결에 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel이나 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출되고 완료되기 전에 multi thread 응용 프로그램의 다른 thread로부터 StatementHandle에 SQLCancel이나 SQLCancelHandle이 호출되었다.
		TableName 인자가 null 포인터였다.

SQLSTATE	Error	설명
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, CatalogName 인자가 null 포인터였다. SQL_CATALOG_NAME 정보타입을 포함하는 SQLGetInfo는 카탈로그 이름이 지원된다는 사실을 반환한다. SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, SchemaName 인자가 null 포인터였다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행되는 함수를 호출하였고 SQLPrimaryKeys가 호출 되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 모든 연결된 매개 변수에 대한 데이터를 검사하기 전에 이 함수가 호출되었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column에 대한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY090	Invalid string or buffer length	이름 길이 인자 중 하나의 값이 0보다 작지만 SQL_NTS가 아니고 이름 인자가 null 포인터가 아니었다. 이름 길이 인자 중 하나의 값이 이름과 대응하는 최대 길의 값을 초과하였다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional feature not implemented	카탈로그가 지정되었지만, 드라이버 또는 데이터 소스가 카탈로그를 지원하지 않는다. 스키마가 지정되었지만, 드라이버 또는 데이터 소스는 스키마를 지원하지 않는다. 드라이버나 데이터 소스가 SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE 명령문 속성의 현재 설정 조합을 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS 명령문 속성이 SQL_UB_VARIABLE로 설정되었고, SQL_ATTR_CURSOR_TYPE 명령문 속성이 드라이버가 지원하지 않는다.

SQLSTATE	Error	설명
		북마크에 대한 커서 타입으로 설정되었다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLPrimaryKeys는 결과를 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, 그리고 KEY_SEQ로 정렬된 표준 결과 집합으로 반환한다. 이 정보를 사용하는 방법은 **카탈로그 데이터 사용**을 참조한다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column 번호로 바인딩하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

TABLE_CAT, TABLE_SCHEM, TABLE_NAME과 COLUMN_NAME column의 실제 길이를 확인하기 위해, SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, SQL_MAX_COLUMN_NAME_LEN 옵션과 함께 SQLGetInfo를 호출한다.

노트

ODBC 카탈로그 함수의 일반적인 사용, 인자 그리고 반환되는 데이터에 대한 자세한 내용은 **카탈로그 함수**를 참조한다.

다음 표는 결과 집합의 column 목록이다. Column 6 이후에 추가되는 column은 드라이버가 정의할 수 있다. 응용 프로그램은 명시적으로 위치를 지정하는 대신, 결과 집합의 끝에서부터 카운트다운하여 관련 column에 접근해야 한다. 자세한 내용은 **카탈로그 함수의 데이터 반환**을 참조한다.

Column name	Column number	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	Varchar	기본 키 테이블 카탈로그 이름이다. 데이터 소스에 사용할 수 없을 경우, NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블의 카탈로그만 지원할 경우, 카탈로그를 포함하지 않는 테이블은 빈 문자열 ("")을 반환한다.
TABLE_SCHEMA (ODBC 1.0)	2	Varchar	기본 키 테이블 스키마 이름이다. 데이터 소스에 사용할 수 없을 경우, NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블의 스키마만 지원할 경우, 스키마를 포함하지 않는 테이블은 빈 문자열 ("")을 반환한다.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	기본 키 테이블 이름이다.
COLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	기본 키 column 이름이다. 드라이버는 이름이 없는 column에 대해 빈 문자열을 반환한다.
KEY_SEQ (ODBC 1.0)	5	Smallint not NULL	키의 column 순차 번호이다 (1부터 시작한다.).
PK_NAME (ODBC 2.0)	6	Varchar	기본 키 이름이다. 데이터 소스에 적용할 수 없을 경우, NULL이다.

SQLProcedureColumns

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLProcedureColumns는 입출력 매개 변수의 목록 뿐만 아니라 지정된 절차의 결과 집합을 구성하는 column을 반환한다. 드라이버는 지정된 명령문에 대한 결과 집합으로 정보를 반환한다.

구문

```
SQLRETURN SQLProcedureColumns(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      ProcName,
    SQLSMALLINT    NameLength3,
    SQLCHAR *      ColumnName,
    SQLSMALLINT    NameLength4);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] Procedure 카탈로그의 이름이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 procedure들은 카탈로그를 포함하지 않는다. CatalogName에는 문자열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 문자열 인자로 취급되고 대소문자를 구분한다. 자세한 내용은 카탈로그 함수의 인자를 참조한다.

NameLength1

[입력] *CatalogName 문자열의 길이이다.

SchemaName

[입력] Procedure 스키마 이름. 스키마 이름에 대한 문자열 검색 패턴이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 procedure들은 스키마를 포함하지 않는다. SchemaName에는 문자열 검색 패턴이 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 패턴값 문자열 인자로 취급되며 대소문자를 구분한다.

NameLength2

[입력] *SchemaName 문자열의 길이이다.

ProcName

[입력] Procedure 이름이다. 이 인자는 NULL 포인터를 사용할 수 없다. ProcName은 문자열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, ProcName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, ProcName은 패턴값 문자열 인자로 취급되고 대소문자를 구분한다.

NameLength3

[입력] *ProcName 문자열의 길이이다.

ColumnName

[입력] Column 이름이다. 이 인자는 NULL 포인터를 사용할 수 없다. ColumnName은 문자열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, ColumnName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, ColumnName은 패턴값 문자열 인자로 취급되고 대소문자를 구분한다.

NameLength4

[입력] *ColumnName 문자열의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
		StatementHandle에 커서가 열려 있고, SQLFetch 또는 SQLFetchScroll이 호출되었다.
24000	Invalid cursor state	SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에

SQLSTATE	Error	설명
		러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 결과 집합이 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수를 실행하는 중에 관련된 연결이 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	TableName 인자가 null 포인터였다. SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, CatalogName 인자가 null 포인터였다. SQL_CATALOG_NAME 정보타입을 포함하는 SQLGetInfo는 카탈로그 이름이 지원된다는 사실을 반환한다. The SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, SchemaName, ProcName 또는 ColumnName 인자가 null 포인터였다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했고 SQLProcedureColumns가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 모든 연결된 매개 변수에 대한 데이터를 검사하기 전에 이 함수가 호출되었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column에 대한 데이터가 전송되기 전에 이 함수가 호출되었다. StatementHandle에 대해 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다.
HY090	Invalid string or buffer length	이름 길이 인자들이 0보다 작지만 SQL_NTS와 같지 않다. 이름 길이 인자 중 하나의 값이 이름과 대응하는 최대 길의 값을 초과하였다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
		카탈로그가 지정되었지만, 드라이버 또는 데이터 소스가 카탈로그를 지원하지 않는다.

SQLSTATE	Error	설명
HYC00	Optional feature not implemented	스키마가 지정되었지만, 드라이버 또는 데이터 소스가 스키마를 지원하지 않는다. 드라이버나 데이터 소스가 SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE 명령문 속성의 현재 설정 조합을 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS 명령문 속성이 SQL_UB_VARIABLE로 설정되었고, SQL_ATTR_CURSOR_TYPE 명령문 속성이 드라이버가 지원하지 않는 북마크에 대한 커서 타입으로 설정되었다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLProcedureColumns는 입출력 매개 변수의 목록 뿐만 아니라 지정된 절차의 결과 집합을 구성하는 column을 반환한다. 드라이버는 지정된 명령문에 대한 결과 집합으로 정보를 반환한다.

SQLProcedureColumns는 PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN TYPE의 순서로 정렬된 표준 결과 집합으로 결과를 반환한다. Column 이름은 각 매개 변수 이름 (호출 순서)과 procedure가 반환한 결과 집합의 각 column 이름의 순서로 반환된다.

PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN_NAME column의 실제 길이를 확인하기 위해 응용 프로그램에서 SQLGetInfo 함수를 SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME, SQL_MAX_PROCEDURE_NAME_LEN 옵션과 함께 호출할 수 있다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column 번호로 바인딩하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEM

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

다음 column은 ODBC 3.x에서 SQLProcedureColumns가 반환하는 결과 집합에 추가되었다.

- COLUMN_DEF
- DATETIME_CODE
- CHAR_OCTET_LENGTH
- ORDINAL_POSITION
- IS_NULLABLE

다음 표에는 결과 집합의 column들이 나열되어 있다. Column 19 (IS_NULLABLE) 이후에 추가된 row는 드라이버가 정의할 수 있다. 응용 프로그램은 명시적으로 서수 위치를 지정하기보다는 결과 집합의 끝에서부터 카운트 다운하여 특정 드라이버 column에 접근해야 한다.

Column name	Column number	Data type	설명
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	Procedure 카탈로그 이름이다. 데이터 소스에 사용할 수 없을 경우 NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블의 카탈로그만 지원할 경우, 카탈로그를 포함하지 않는 테이블은 빈 문자열 ("")을 반환한다.
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	Procedure 스키마 이름이다. 데이터 소스에 사용할 수 없을 경우 NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블의 스키마만 지원할 경우, 스키마를 포함하지 않는 테이블은 빈 문자열 ("")을 반환한다.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	Procedure 이름이다. Procedure 이름이 없는 경우 빈 문자열을 반환한다.
COLUMN_NAME (ODBC 2.0)	4	Varchar not NULL	Procedure column 이름이다. 이름이 없는 procedure의 경우 드라이버가 빈 문자열을 반환한다.
COLUMN_TYPE (ODBC 2.0)	5	Smallint not NULL	Procedure 유형을 정의한다. • SQL_PARAM_TYPE_UNKNOWN: Procedure column의 타입을 알 수 없다. (ODBC 1.0) • SQL_PARAM_INPUT: Procedure column이 input 매개 변수이다. (ODBC 1.0) • SQL_PARAM_INPUT_OUTPUT: Procedure column이 input/ output 매개 변수이다. (ODBC 1.0) • SQL_PARAM_OUTPUT: Procedure column이 output 매개 변수이다. (ODBC 2.0)

Column name	Column number	Data type	설명
			- SQL_RETURN_VALUES: Procedure column이 procedure의 반환값이다. (ODBC 2.0) - SQL_RESULT_COL: Procedure column은 결과 집합 column이다. (ODBC 1.0)
DATA_TYPE (ODBC 1.0)	6	Smallint not NULL	SQL 데이터 타입이다. Datetime 및 interval 데이터 타입의 경우, 이 column은 SQL_TYPE_DATE, SQL_INTERVAL_YEAR_TO_MONTH와 같은 concise 데이터 타입을 반환한다.
TYPE_NAME (ODBC 2.0)	7	Varchar not NULL	데이터 소스 종속 데이터 타입 이름이다. CHAR(), VARCHAR(), MONEY, LONG VARBINARY 또는 CHAR () FOR BIT DATA와 같은 것들이 있다. 응용 프로그램은 CREATE TABLE과 ALTER TABLE 명령문에서 반드시 이 이름을 사용해야 한다.
COLUMN_SIZE (ODBC 2.0)	8	Integer	데이터 타입에 대해 서버가 지원하는 column의 최대 크기이다. 숫자형 데이터 타입은 최대 precision, 문자열 데이터 타입은 문자들의 길이, DATETIME 데이터 타입은 표현되는 문자 길이, INTERVAL 데이터 타입은 INTERVAL 그대로의 문자 표현에 대한 문자 길이, column 사이즈를 적용할 수 없는 데이터 타입은 NULL이 반환된다.
BUFFER_LENGTH (ODBC 1.0)	9	Integer	SQL_C_DEFAULT가 명시되었을 때, SQLGetData나 SQLFetch 연산에서 전송되는 바이트의 길이이다. Numeric 데이터의 크기는 데이터 소스에 저장된 데이터의 크기와 다를 수 있다. 문자열이나 바이너리 데이터의 경우, 이 값은 COLUMN_SIZE column과 같다.
DECIMAL_DIGITS (ODBC 1.0)	10	Smallint	데이터 소스에서 column의 소수점 자릿수이다. 데이터 형식의 소수점 자릿수를 적용할 수 없을 경우, NULL을 반환한다.
NUM_PREC_RADIX (ODBC 2.0)	11	Smallint	숫자형 데이터 타입의 경우, 2 또는 10이다. 2인 경우, COLUMN_SIZE와 DECIMAL_DIGITS는 column에 허용된 비트 수이다. 10인 경우, COLUMN_SIZE와 DECIMAL_DIGITS는 column에 허용된 자릿수이다. NUM_PREC_RADIX를 적용할 수 없는 데이터 타입은 NULL을 반환한다.
NULLABLE (ODBC 2.0)	12	Smallint not NULL	데이터 타입이 NULL 값을 허용하는지 여부이다. SQL_NULLABLE일 경우, 데이터 타입이 NULL 값을 허용한다. SQL_NO_NULLS일 경우, 데이터 타입이 NULL 값을 허용하지 않는다. SQL_NULLABLE_UNKNOWN일 경우, column이 NULL 값을 허용하는지 여부를 알 수 없다.
REMARKS (ODBC 2.0)	13	Varchar	Procedure column에 대한 설명이다.
COLUMN_DEFINITION (ODBC 3.0)	14	Varchar	Column의 기본값이다. 이 값이 quote로 묶여 있을 경우, 이 column은 문자열로 해석되어야 한다.
			설명자의 SQL_DESC_TYPE 필드에 나타나는 SQL 데이터 타입의 값이다. 이 column은 INTERVAL과 DATETIME 데이터 타입을 제외한 DATA_TYPE column

Column name	Column number	Data type	설명
SQL_DATA_TYPE (ODBC 3.0)	15	Smallint not NULL	mn과 동일하다. 결과 집합의 SQL_DATE_TYPE 필드는 INTERVAL과 DATETIME 데이터 타입에 대해 SQL_INTERVAL이나 SQL_DATETIME을 반환하고, SQL_DATETIME_SUB 필드는 INTERVAL이나 DATETIME 데이터 타입에 대한 하위 코드를 반환한다.
SQL_DATETIME_SUB (ODBC 3.0)	16	Smallint	Datetime과 interval 데이터 타입의 하위 타입 코드이다. 다른 타입의 경우, NULL을 반환한다.
CHAR_OCTET_LENGTH (ODBC 3.0)	17	Integer	문자 또는 이진 데이터 타입 column의 바이트 단위 최대 길이이다. 다른 타입의 경우, NULL을 반환한다.
ORDINAL_POSITION (ODBC 3.0)	18	Integer not NULL	테이블에서 column의 위치이다.
IS_NULLABLE (ODBC 3.0)	19	Varchar	"YES": Column이 NULL을 포함할 수 있다. "NO": Column이 NULL을 포함할 수 없다. - NULL 허용 여부를 알 수 없는 경우, 길이가 0인 문자열을 반환한다.

SQLProcedures

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLProcedures는 지정된 데이터 소스에 저장된 procedure 이름의 목록을 반환한다. Procedure는 실행 가능한 객체나 입력/ 출력 매개 변수를 이용하여 호출할 수 있는 기명의 개체를 설명하는데 사용되는 일반적인 용어이다.

구문

```
SQLRETURN SQLProcedures(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      ProcName,
    SQLSMALLINT    NameLength3);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] Procedure 카탈로그 이름이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 procedure들은 카탈로그를 가지지 않는다. 카탈로그 이름은 문자열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 문자열 인자로 취급되고 대소문자를 구분한다. 자세한 내용은 [카탈로그 함수의 인자](#)를 참조한다.

NameLength1

[입력] *CatalogName 문자열의 길이이다.

SchemaName

[입력] Procedure 스키마 이름이다. 스키마 이름에 대한 문자열 검색 패턴이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 procedure들은 스키마를 가지지 않는다. SchemaName은 문자

열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 패턴값 문자열 인자로 취급되며 대소문자를 구분한다.

NameLength2

[입력] *SchemaName 문자열의 길이이다.

ProcName

[입력] Procedure 이름이다. 이 인자는 NULL 포인터가 될 수 없다. ProcName은 문자열 검색 패턴을 포함할 수 없다. SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, ProcName은 식별자로 취급되고 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, ProcName은 패턴값 문자열 인자로 취급되며 대소문자를 구분한다.

NameLength3

[입력] *ProcName 문자열의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	StatementHandle에 커서가 열려있고 SQLFetch나 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 결과 집합이 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	관련된 연결이 이 함수를 실행하는 중에 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	TableName 인자가 null 포인터였다. SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, CatalogName 인자가 null 포인터였다. SQL_CATALOG_NAME 정보타입을 포함하

SQLSTATE	Error	설명
		는 SQLGetInfo는 카탈로그 이름이 지원된다는 사실을 반환한다. SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정되었고, SchemaName, ProcName 또는 ColumnName 인자가 null 포인터였다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행되는 함수를 호출하였고 SQLProcedureColumns가 호출 되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 모든 연결된 매개 변수에 대한 데이터를 검사하기 전에 이 함수가 호출되었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출하였고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column에 대한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY090	Invalid string or buffer length	이름 길이 인자들이 0보다 작지만 SQL_NTS와 같지 않다. 이름 길이 인자 중 하나의 값이 이름과 대응하는 최대 길이 값을 초과하였다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional feature not implemented	카탈로그가 지정되었지만, 드라이버 또는 데이터 소스가 카탈로그를 지원하지 않는다. 스키마가 지정되었지만, 드라이버 또는 데이터 소스가 스키마를 지원하지 않는다. 드라이버나 데이터 소스가 SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE 명령문 속성의 현재 설정 조합을 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS 명령문 속성이 SQL_UB_VARIABLE로 설정되었고, SQL_ATTR_CURSOR_TYPE 명령문 속성이 드라이버가 지원하지 않는 북마크에 대한 커서 타입으로 설정되었다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수

SQLSTATE	Error	설명
		있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대해 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLProcedure는 요청된 범위의 모든 procedure를 나열한다. 사용자는 procedure 실행 권한을 가질 수도 있고 가지지 않을 수도 있다. 접근 권한을 확인하려면 SQLGetInfo의 SQL_ACCESSIBLE_PROCEDURES를 참조한다. 사용자가 실행할 수 없는 procedure를 선택한 경우, 응용 프로그램이 이를 제어할 수 있어야 한다. SQLProcedures는 표준 결과 집합을 반환하며, PROCEDURE_CAT, PROCEDURE_SCHEMA, PROCEDURE_NAME 순서로 정렬된다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column 번호로 바인딩하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEM

PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME column의 실제 길이를 확인하기 위해, 응용 프로그램에서 SQLGetInfo를 SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME, SQL_MAX_PROCEDURE_NAME_LEN 옵션과 함께 호출할 수 있다.

다음 표에는 결과 집합의 column들이 나열되어 있다. Column 8 (PROCEDURE_TYPE) 이후에 추가된 row는 드라이버가 정의할 수 있다. 응용 프로그램은 명시적으로 서수 위치를 지정하기보다는 결과 집합의 끝에서부터 카운트 다운하여 특정 드라이버 column에 접근해야 한다.

Column name	Column number	Data type	Comments
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	Procedure 카탈로그 이름이다. 데이터 소스에 사용할 수 없을 경우, NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블의 카탈로그만 지원할 경우, 카탈로그를 포함하지 않는 테이블은 빈 문자열("")을 반환한다.
PROCEDURE_SCHEMA			Procedure 스키마 식별자이다. 데이터 소스에 사용할 수 없을 경우, NULL이다. (드라이버가 다른 DBMS에서 데이터를 검색할 때와 같이) 드라이버가 일부 테이블

Column name	Column number	Data type	Comments
E_SCHEM (ODBC 2.0)	2	Varchar	이블의 스키마만 지원할 경우, 스키마를 포함하지 않는 테이블은 빈 문자열 ("")을 반환한다.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not null	Procedure 식별자이다.
NUM_INPUT_PARAMS (ODBC 2.0)	4	N/A	reserved 응용 프로그램은 이러한 결과 column에 반환된 데이터에 의존하지 말아야 한다.
NUM_OUTPUT_PARAMS (ODBC 2.0)	5	N/A	reserved 응용 프로그램은 이러한 결과 column에 반환된 데이터에 의존하지 말아야 한다.
NUM_RESULT_SETS (ODBC 2.0)	6	N/A	reserved 응용 프로그램은 이러한 결과 column에 반환된 데이터에 의존하지 말아야 한다.
REMARK (ODBC 2.0)	7	Varchar	Procedure에 대한 설명이다.
PROCEDURE_TYPE (ODBC 2.0)	8	Smallint	Procedure 유형을 정의한다. - SQL_PT_UNKNOWN: Procedure가 값을 반환하는지 여부를 알 수 없다. - SQL_PT_PROCEDURE: 반환된 객체가 procedure이다. 즉 반환값이 존재하지 않는다. - SQL_PT_FUNCTION: 반환된 객체가 FUNCTION이다. 즉 반환값이 존재한다.

SQLPutData

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLPutData는 명령문이 실행될 때 응용 프로그램이 드라이버에 매개 변수를 전송하거나 column에 데이터를 전송할 수 있도록 한다. 이 함수는 문자, 바이너리 또는 데이터 소스별 데이터 타입 (예를 들어, SQL_LONGVARIABLE 또는 SQL_LONGVARIABLE 타입의 매개 변수)이 있는 column으로 일부 문자나 바이너리 데이터 값을 전송하기 위해 사용될 수 있다. 기본 드라이버가 유니코드를 지원하지 않더라도 SQLPutData는 유니코드 C 데이터 타입에 바인딩을 지원한다.

구문

```
SQLRETURN SQLPutData(
    SQLHSTMT      StatementHandle,
    SQLPOINTER    DataPtr,
    SQLLEN        StrLen_or_Ind);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

DataPtr

[입력] 매개 변수나 column에 대한 실제 데이터가 들어있는 버퍼의 포인터이다. 데이터는 SQLBindParameter의 ValueType 인자 (매개 변수 데이터) 또는 SQLBindCol의 TargetType 인자에 지정되는 C 데이터 타입이어야 한다 (column 데이터).

StrLen_or_Ind

[입력] *DataPtr의 길이이다. SQLPutData 호출로 전송되는 데이터의 양을 지정한다. 데이터의 양은 주어진 매개 변수나 column에 대한 호출에 따라 다를 수 있다. StrLen_or_Ind가 다음 조건 중 하나를 만족하지 않으면 무시된다.

- StrLen_or_Ind가 SQL_NTS, SQL_NULL_DATA, 또는 SQL_DEFAULT_PARAM이다.
- SQLBindParameter 또는 SQLBindCol에 지정된 C 데이터 타입이 SQL_C_CHAR 또는 SQL_C_BINARY이다.
- C 데이터 타입이 SQL_C_DEFAULT이고, 지정된 SQL 데이터 타입에 대한 기본 C 데이터 타입이 SQL_C_

CHAR 또는 SQL_C_BINARY이다.

C 데이터의 모든 다른 타입에 대해, StrLen_or_Ind가 SQL_NULL_DATA 또는 SQL_DEFAULT_PARAM이 아니라면, 드라이버는 *DataPtr의 크기가 ValueType 또는 TargetType에 지정된 C 데이터 타입의 크기라고 측정하여 전체 데이터 값을 전송한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncated	입력/ 출력 또는 출력 매개 변수에 반환된 문자열이나 이진 데이터가 잘렸다. 문자열이 잘린 경우, 오른쪽이 잘렸다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07006	Restricted data type attribute violation	바인드 된 매개 변수에 대해 SQLBindParameter의 ValueType 인자로 식별된 데이터 값을 SQLBindParameter의 ParameterType 인자가 식별하는 데이터 형식으로 변환할 수 없다.
07S01	Invalid use of default parameter	SQLBindParameter에 설정된 매개 변수 값이 SQL_DEFAULT_PARAM이고 대응되는 매개 변수가 기본값을 갖지 않았다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22001	String data, right truncation	문자열이나 이진 데이터가 잘렸다. SQLGetInfo에서 SQL_NEED_LONG_DATA_LEN 정보 타입이 "Y" 이고, StrLen_or_IndPtr 인자로 SQLBindParameter에 지정된 것보다 더 많은 데이터가 긴 매개 변수에 대해 전송되었다. SQLGetInfo에서 SQL_NEED_LONG_DATA_LEN 정보 타입이 "Y" 이고, SQLBulkOperation으로 추가 또는 갱신되거나 SQLSetPos로 갱신된 데이터 row에 대응하는 버퍼 길이에 지정된 것보다 더 많은 데이터가 긴 column에 대해 전송되었다.
22003	Numeric value out of range	관련된 테이블 column을 할당할 때, 바인딩된 숫자 매개 변수 또는 열에 전송한 데이터가 수치값을 잘리게 했다. 입력/ 출력 매개 변수 또는 출력 매개 변수에 대해 반환되는 숫자값이 잘렸다.
22007	Invalid datetime format	DATE, TIME 또는 TIMESTAMP 구조에 바인딩 된 매개 변수 또는 column에 대해 전송된 데이터가 각 타입에 유효하지 않다. 입력/ 출력 또는 출력 매개 변수가 DATE, TIME 또는 TIMESTAMP C 구조에 바인딩되었고, 매개 변수에 반환되는 값이 각 타입에 대해 유효하지 않다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)

SQLSTATE	Error	설명
22008	Datetime field overflow	바인딩 된 DATE, TIME 또는 TIMESTAMP C 구조에 대한 DATETIME 표현이 유효하지 않다.
22012	Division by zero	입력/ 출력 또는 출력 매개 변수에 대한 계산식을 0으로 나눴다.
22015	Interval field overflow	정확한 숫자, INTERVAL column 또는 매개 변수에 대해 INTERVAL SQL 데이터 타입에 전송된 데이터로 인해 유효 숫자가 손실되었다. 하나 이상의 INTERVAL column 또는 매개 변수에 전송된 데이터가 숫자 데이터 형식으로 변환되었으나 숫자 데이터 형식으로 표현될 수 없다. Column 또는 매개 변수 데이터에 대해 전송된 데이터가 INTERVAL SQL 타입에 할당되었지만 데이터가 INTERVAL SQL 타입에서 C 타입값으로 표현될 수 없다. 정확한 숫자 또는 INTERVAL C column 또는 매개 변수로 전송된 데이터가 유효 숫자를 잃었다. Column 또는 매개 변수 데이터에 대해 전송된 데이터가 INTERVAL C 구조에 할당되었지만 데이터를 INTERVAL 데이터 구조로 표현할 수 없다.
22018	Invalid character value for cast specification	결과 집합의 문자 column에 C 버퍼의 character set으로 표현할 수 없는 문자를 포함한다. C 타입은 정확한 numeric, 근사값 numeric, datetime, interval 데이터 타입이고, SQL 타입이 문자 데이터 형식일 때, C 타입에 바인딩 되는 column의 값이 유효한 문자가 아니다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY009	Invalid use of null pointer	DataPtr 인자가 null 포인터였고, StrLen_or_Ind 인자가 SQL_DEFAULT_PARAM 또는 SQL_NULL_DATA가 아니었다.
HY010	Function sequence error	이전의 함수 호출이 SQLPutData 또는 SQLParamData를 위한 것이 아니었다. StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했고 SQLPrimaryKeys가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다.

SQLSTATE	Error	설명
		StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperation 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column을 위한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다
HY019	Non-character and non-binary data sent in pieces	매개 변수 또는 column에 대해 SQLPutData이 한 번 이상 호출되었고, 문자, 바이너리 또는 데이터 소스별 데이터 타입이 있는 column에 문자 C 데이터를 전송하거나 바이너리 C 데이터를 전송하는데 사용되지 않았다.
HY020	Attempt to concatenate a null value	SQL_NEED_DATA가 반환되고 난 후에 SQLPutData가 한 번 이상 호출되었고 이런 호출 중 한 번은 StrLen_or_Ind 인자에 SQL_NULL_DATA 또는 SQL_DEFAULT_PARAM이 포함되었다.
HY090	Invalid string or buffer length	DataPtr 인자가 null 포인터가 아니고, StrLen_or_Ind가 0보다 작거나 SQL_NTS 또는 SQL_NULL_DATA와 동일하지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync을 호출해야 한다.

설명

SQLPutData는 data-at-execution 데이터를 제공하기 위해 두 가지 용도로 호출될 수 있다. 하나는 SQLExecute나 SQLExecDirect 호출의 매개 변수 데이터로 사용하기 위함이고, 다른 하나는 SQLBulkOperations를 호출하여 row를 갱신하거나 추가할 때 또는 SQLSetPos를 호출하여 row를 갱신할 때 column 데이터로 사용하기 위함이다.

어떤 데이터를 전송할지 결정하기 위해 응용 프로그램이 SQLParamData를 호출하면 드라이버는 응용 프로그램이 어떤 매개 변수 데이터를 전송할지 또는 column 데이터를 어디에서 찾을지 판별하기 위한 표시를 반환한다. 또한, 이 함수는 응용 프로그램이 SQLPutData를 호출해야 한다는 표시인 SQL_NEED_DATA를 반환한다. 응용 프로그램은 SQLPutData의 DataPtr 인자에 매개 변수 또는 column에 대한 실제 데이터가 들어있는 버퍼의 포인터를 전달한다.

드라이버가 SQLPutData에 대해 SQL_SUCCESS를 반환할 때, 응용 프로그램은 SQLParamData을 다시 호출한다.

데이터를 더 전송해야 할 경우, `SQLParamData`는 `SQL_NEED_DATA`를 반환하고 응용 프로그램은 `SQLPutData`를 다시 호출한다. 모든 `data-at-execution` 데이터가 전송되면 `SQL_SUCCESS`를 반환한다. 그러면 응용 프로그램은 `SQLParamData`을 다시 호출한다. 드라이버가 `SQL_NEED_DATA`와 `*ValuePtrPtr`에 다른 표시를 반환하였다면, 이것은 다른 매개 변수 또는 `column`에 대한 데이터를 요청하고 `SQLPutData`를 다시 호출한다. 드라이버가 `SQL_SUCCESS`를 반환하면, 모든 `data-at-execution` 데이터가 전송되고 SQL 명령문을 실행할 수 있거나 `SQLBulkOperations` 또는 `SQLSetPos`를 처리할 수 있다.

노트

응용 프로그램이 문자, 바이너리 또는 데이터 소스별 데이터 타입에 문자 C 데이터나 바이너리 C 데이터를 전송할 때만 `SQLPutData`를 사용할 수 있다. 만약 `SQLPutData`가 다른 조건에서 한 번 이상 호출되면, `SQL_ERROR`와 `SQLSTATE HY019`가 반환된다 (non-character and non-binary data sent in pieces).

SQLRowCount

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLRowCount는 UPDATE, INSERT, 또는 DELETE 명령문에 영향을 받는 row의 개수를 반환한다. (SQLBulkOperations에서 SQL_ADD, SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK 작업 또는 SQLSetPos에서 SQL_UPDATE, SQL_DELETE 작업)

구문

```
SQLRETURN SQLRowCount(
    SQLHSTMT StatementHandle,
    SQLLEN * RowCountPtr);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

RowCountPtr

[출력] Row 개수가 반환될 버퍼의 포인터이다. UPDATE, INSERT, DELETE 명령문, SQLBulkOperations의 SQL_ADD, SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK 작업, SQLSetPos의 SQL_UPDATE 또는 SQL_DELETE 작업에 대해 *RowCountPtr에 반환되는 값은 요청이 영향을 미친 row의 개수이거나 영향받은 row를 사용할 수 없을 경우 -1이다.

SQLExecute, SQLExecDirect, SQLBulkOperations, SQLSetPos 또는 SQLMoreResults가 호출될 때, 진단 데이터 구조의 SQL_DIAG_ROW_COUNT 필드가 row 개수로 설정되고, row 개수는 구현에 의존하는 방식으로 캐싱된다. SQLRowCount는 캐싱되는 row 개수를 반환한다. 캐싱되는 row 개수의 값은 명령문 핸들이 준비 또는 할당 상태로 재설정되거나, 명령문이 재실행되거나 또는 SQLCloseCursor이 호출될 때까지 유효하다. SQL_DIAG_ROW_COUNT 필드가 설정된 후 이 함수가 호출될 경우, 함수 호출에 의해 SQL_DIAG_ROW_COUNT 필드가 0으로 설정되기 때문에 SQLRowCount가 반환하는 값은 SQL_DIAG_ROW_COUNT 필드에 있는 값과 다를 수 있다.

다른 명령문과 함수의 경우, 드라이버가 *RowCountPtr에 반환되는 값을 정의할 수 있다. 예를 들어, 일부 데이터 소스는 row가 fetch되기 전에 SELECT 명령문이 반환하는 row 개수를 반환할 수 있다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기적으로 수행하는 함수를 호출했고 SQLRowCount가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 모든 연결된 매개 변수에 대한 데이터를 검사하기 전에 이 함수가 호출되었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos를 호출하기 전에 이 함수가 먼저 호출되었다. StatementHandle에 대해 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column에 대한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT00	Timeout expired	데이터 소스로부터 결과 집합을 받기 전에 쿼리 제한시간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.

설명

명령문 핸들에 대해 실행되는 마지막 SQL 명령문이 UPDATE, INSERT, DELETE가 아니었거나, 또는 이전에 SQLBulkOperations를 호출할 때 operation 인자가 SQL_ADD, SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK가 아니거나, 이전에 SQLSetPos를 호출할 때 operation 인자가 SQL_UPDATE 또는 SQL_DELETE가 아닐 경우, *RowCountPtr의 값은 드라이버에서 정의한 값이다.

SQLSetConnectAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLSetConnectAttr은 연결을 제어하는 속성을 설정한다.

구문

```
SQLRETURN SQLSetConnectAttr(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength);
```

인자

ConnectionHandle

[입력] 연결 핸들이다.

Attribute

[입력] 설정을 위한 속성이다.

ValuePtr

[입력] Attribute와 관련된 값의 포인터이다. Attribute 값에 따라 ValuePtr은 unsigned 정수값이 되거나 null 종료 문자열을 가리킬 수 있다. Attribute 인자의 정수 타입은 고정 길이가 아닐 수 있다. 자세한 내용은 설명 섹션을 참조한다.

StringLength

[입력] Attribute가 ODBC 정의 속성이고 ValuePtr이 문자열이나 바이너리 버퍼를 가리키면, 이 인자는 *ValuePtr의 길이여야 한다. 문자열 데이터에 대해 이 인자는 문자열의 바이트 수를 포함해야 한다.

Attribute가 ODBC 정의 속성이고 ValuePtr이 정수일 경우, StringLength는 무시된다.

Attribute가 드라이버 정의 속성일 경우, 응용 프로그램은 StringLength 인자를 설정하여 속성의 특성을 드라이버 관리자에 나타낸다. StringLength는 다음과 같은 값을 가질 수 있다:

- ValuePtr이 문자열 포인터일 경우, StringLength는 문자열의 길이 또는 SQL_NTS이다.
- ValuePtr이 바이너리 버퍼 포인터일 경우, 응용 프로그램은 음수인 SQL_LEN_BINARY_ATTR (length)

매크로의 결과를 StringLength에 저장한다.

- ValuePtr이 문자열 포인터나 바이너리 버퍼 포인터가 아닐 경우, StringLength가 SQL_IS_POINTER를 가져야 한다.
- ValuePtr이 고정 길이 값을 포함할 경우, StringLength는 SQL_IS_INTEGER 또는 SQL_IS_INTEGER이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	ValuePtr에 지정된 값을 드라이버가 지원하지 않고 유사값으로 대체했다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08002	Connection name in use	Attribute 인자가 SQL_ATTR_ODBC_CURSORS이고, 드라이버가 데이터 소스에 이미 연결되었다.
08003	Connection not open	Attribute 값이 열린 연결을 요청하도록 지정되었지만, ConnectionHandle이 연결 상태가 아니다.
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	Attribute 인자가 SQL_ATTR_CURRENT_CATALOG였고, 결과 집합이 지연되었다.
25000	Illegal operation while in a local transaction	연결 속성 SQL_ATTR_ENLIST_IN_DTC를 설정하여 분산 트랜잭션 연결을 시도하는 동안 로컬 트랜잭션이 연결되었다. 연결이 이미 분산 트랜잭션에 등록되어 있다. 분산 트랜잭션에 연결이 시도되었고, SQL_ATTR_AUTOCOMMIT을 SQL_AUTOCOMMIT_OFF로 설정하여 로컬 트랜잭션을 시작하였다.
3D000	Invalid catalog name	Attribute 인자가 SQL_ATTR_CURRENT_CATALOG이고 지정된 카탈로그 이름이 유효하지 않다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY008	Operation canceled	ConnectionHandle에 대해 비동기 처리가 활성화 되었다. SQLSetConnectAttr 함수가 호출되고 수행이 종료되기 전에 SQLCancelHandle 함수가 ConnectionHandle에 호출되었고, SQLSetConnectAttr 함수가 ConnectionHandle에 다시 호출되었다. SQLSetConnectAttr 함수가 호출되고 수행이 종료되기 전에, 다중 thread 응용

SQLSTATE	Error	설명
		프로그램의 다른 thread로 부터 ConnectionHandle에 SQLCancelHandle을 호출하였다.
HY009	Invalid use of null pointer	Attribute 인자가 문자열 값을 요구하는 연결 속성을 식별하고 ValuePtr 인자가 null 포인터이다.
HY010	Function sequence error	ConnectionHandle과 관련된 StatementHandle에 대해 비동기적으로 수행하는 함수가 호출되었고 수행하는 중에 SQLSetConnectAttr가 호출되었다. ConnectionHandle에 대해 비동기적으로 수행하는 함수를 호출했고 이 함수가 호출되었을 때 이 비동기 수행 함수가 여전히 실행되고 있었다. ConnectionHandle과 관련된 명령문 핸들 중 하나에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출이 되었고 SQL_PARAM_DATA_AVAILABLE을 반환하였다. 스트림 된 모든 매개 변수들에 대한 데이터가 회수되기 전에 이 함수가 호출되었다. SQLExecute, SQLExecDirect, SQLBulkOperations, 또는 SQLSetPos 함수가 ConnectionHandle과 관련된 StatementHandle에 대해 호출되었고, SQL_NEED_DATA를 반환하였다. 이 함수는 모든 data-at-execution 매개 변수 또는 column을 전송하기 전에 호출되었다. SQLBrowseConnect가 ConnectionHandle에 대해 호출되었고 SQL_NEED_DATA를 반환하였다. SQLBrowseConnect가 SQL_SUCCESS_WITH_INFO 또는 SQL_SUCCESS를 반환하기 전에 이 함수가 호출되었다.
HY011	Attribute cannot be set now	Attribute 인자가 SQL_ATTR_TXN_ISOLATION이고, 트랜잭션이 열려있다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY024	Invalid attribute value	지정된 Attribute 값이 주어지고, 유효하지 않은 값이 ValuePtr에 지정되었다. Attribute 인자가 SQL_ATTR_TRACEFILE 또는 SQL_ATTR_TRANSLATE_LIB이고, ValuePtr이 빈 문자열이다.
HY090	Invalid string or buffer length	ValuePtr이 문자열이고 StringLength 인자가 0보다 작지만 SQL_NTS는 아니다.
HY114	Driver does not support connection-level asynchronous function execution	비동기 연결을 지원하지 않는 드라이버에서 응용 프로그램이 SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE로 비동기 실행 기능을 활성화시키려 했다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
	Cursor Library and	

SQLSTATE	Error	설명
HY121	Driver-Aware Pooling cannot be enabled at the same time	드라이버에서 지원하지 않는다.
HYC00	Optional feature not implemented	Attribute 인자에 대해 지정된 값이 드라이버가 지원하는 버전에서 유효한 ODBC 연결 또는 명령문 속성이지만 드라이버가 이를 지원하지 않는다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	ConnectionHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM009	Unable to load translation DLL	드라이버가 연결을 위해 지정된 트랜잭션 DLL을 로드할 수 없다. 이 에러는 attribute가 SQL_ATTR_TRANSLATE_LIB일 때만 반환될 수 있다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.
S1118	Driver does not support asynchronous notification	SQL_ATTR_ASYNC_DBC_EVENT가 설정되었지만, 드라이버가 비동기 알림을 지원하지 않는다.

설명

응용 프로그램은 연결이 할당되고 해제되는 동안 SQLSetConnectAttr을 언제든지 호출할 수 있다. 응용 프로그램이 연결을 위해 성공적으로 설정한 모든 연결과 명령문 속성은 SQLFreeHandle이 호출될 때까지 유지된다. 만약 응용 프로그램이 데이터 소스를 연결하기 전에 SQLSetConnectAttr을 호출하면, 응용 프로그램이 데이터 소스에 연결될 때, SQLSetConnectAttr가 드라이버에서 실패하더라도 그 속성은 유지된다. 만약 응용 프로그램이 드라이버 별 속성을 설정하면, 응용 프로그램이 다른 드라이버에 연결되더라도 속성은 유지된다.

노트

SQLSetConnectAttr을 호출하여 연결 레벨에서 명령문 속성을 설정하는 기능은 ODBC 3.x에서 사용되지 않는다. ODBC 3.x의 응용 프로그램은 연결 레벨에서 명령문 속성을 설정하면 안된다. ODBC 3.x는 SQL_ATTR_METADATA_ID와 SQL_ATTR_ASYNC_ENABLE 속성을 제외한 명령문 속성을 연결 레벨에서 설정할 수 없다. 이 두 속성은 모두 연결 속성과 명령문 속성이며 연결 레벨 또는 명령문 레벨에서 설정될 수 있다. 만약 연결 레벨에서 ODBC 2.x 명령문 옵션을 설정하는 ODBC 2.x 응용 프로그램으로 작업을 해야 하는 경우,

ODBC 3.x 드라이버가 이 기능을 지원해야 한다.

일부 연결 속성은 연결되기 전에만 설정할 수 있고 일부 속성은 연결된 후에만 설정될 수 있다. 다음 테이블은 이러한 연결 속성을 나타낸다.

Attribute	Set before or after connection?
SQL_ATTR_ACCESS_MODE	Either [1]
SQL_ATTR_ASYNC_DBC_EVENT	Either
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE	Either [4]
SQL_ATTR_ASYNC_DBC_PCALLBACK	Either
SQL_ATTR_ASYNC_DBC_PCONTEXT	Either
SQL_ATTR_ASYNC_ENABLE	Either [2]
SQL_ATTR_AUTO_IPD	Either
SQL_ATTR_AUTOCOMMIT	Either [5]
SQL_ATTR_CONNECTION_DEAD	After
SQL_ATTR_CONNECTION_TIMEOUT	Either
SQL_ATTR_CURRENT_CATALOG	Either [1]
SQL_ATTR_DBC_INFO_TOKEN	After
SQL_ATTR_ENLIST_IN_DTC	After
SQL_ATTR_LOGIN_TIMEOUT	Before
SQL_ATTR_METADATA_ID	Either
SQL_ATTR_OLDPWD	Before
SQL_ATTR_ODBC_CURSORS	Before
SQL_ATTR_PACKET_SIZE	Before
SQL_ATTR_QUIET_MODE	Either
SQL_ATTR_TRACE	Either
SQL_ATTR_TRACEFILE	Either
SQL_ATTR_TRANSLATE_LIB	After
SQL_ATTR_TRANSLATE_OPTION	After
SQL_ATTR_TXN_ISOLATION	Either [3]

[\[1\]](#) SQL_ATTR_ACCESS_MODE와 SQL_ATTR_CURRENT_CATALOG는 드라이버에 따라 연결 전후에 설정될 수 있다. 그러나 여러 드라이버를 사용하는 응용 프로그램의 경우, 일부 드라이버가 연결된 후에 변경하는 것을 지원하지 않으므로 연결하기 전에 설정해야 한다.

[\[2\]](#) SQL_ATTR_ASYNC_ENABLE은 명령문이 활성화되기 전에 설정해야 한다.

[\[3\]](#) SQL_ATTR_TXN_ISOLATION은 연결에 열린 트랜잭션이 없을 때만 설정할 수 있다. 일부 연결 속성은 데이터 소스가 *ValuePtr에 지정된 값을 지원하지 않을 경우 유사한 값으로 대체하여 지원한다. 이 경우, 드라이버는 SQL_SUCCESS_WITH_INFO와 SQLSTATE 01S02 (option value changed)를 반환한다. 예를 들어 attribute가 SQL_ATTR_PACKET_SIZE이고 *ValuePtr이 최대 패킷 크기를 초과하면 드라이버는 최대 크기를 대체한다. 대체한 값을 확인

하기 위해 응용 프로그램은 SQLGetConnectAttr을 호출한다.

[4] 연결이 열리기 전에 SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE이 설정되면, 드라이버 관리자는 SQLBrowseConnect, SQLConnect, 또는 SQLDriverConnect가 호출되는 동안 드라이버가 로드될 때 드라이버의 속성을 설정한다. SQLBrowseConnect, SQLConnect, 또는 SQLDriverConnect가 호출되기 전에 드라이버 관리자는 어떤 드라이버가 연결되는지, 이 드라이버가 비동기 연결 작업을 지원하는지 여부를 알 수 없다. 따라서 드라이버 관리자는 SQL_SUCCESS를 반환한다. 그러나 드라이버가 비동기 연결 작업을 지원하지 않는 경우, SQLBrowseConnect, SQLConnect, 또는 SQLDriverConnect 호출에 실패한다.

[5] SQL_ATTR_AUTOCOMMIT이 FALSE로 설정된 경우, API가 SQL_ERROR를 반환하면 트랜잭션 일관성을 보장하기 위해 응용 프로그램이 SQLEndTran (SQL_ROLLBACK)을 호출해야 한다.

*ValuePtr 버퍼의 정보 형식은 지정된 attribute에 따라 다르다. SQLSetConnectAttr은 속성 정보를 null 종료 문자 또는 정수값의 형태로 받는다. SQLSetConnectAttr의 ValuePtr 인자가 가리키는 문자열은 StringLength 바이트 길이를 갖는다.

길이가 속성에 의해 정의된 경우, ODBC 2.x 또는 그 이전에 도입된 모든 속성의 경우와 같이 StringLength 인자는 무시된다.

Attribute	ValuePtr contents
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	SQLINTEGER 값이다. SQL_MODE_READ_WRITE가 기본값이다. SQL_MODE_READ_ONLY는 업데이트 할 SQL 명령문을 지원하기 위해 연결을 요청하지 않는다는 것을 나타내며 드라이버나 데이터 소스에 사용된다. 이 모드는 드라이버나 데이터 소스에 대한 적절한 잠금 전략, 트랜잭션 관리 또는 다른 영역을 최적화하기 위해 사용될 수 있다. 드라이버는 데이터 소스로 전송되는 이러한 내용을 방지할 필요가 없다. 읽기 전용 연결에 읽기 전용이 아닌 SQL 명령문을 처리하도록 요청되는 경우 드라이버와 데이터 소스는 구현 정의로 작동한다.
SQL_ATTR_ASYNC_DBC_EVENT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_DBC_CALLBACK (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_DBC_CONTEXT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_ENABLE (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_AUTO_IPD (ODBC 3.0)	드라이버에서 지원하지 않는다.
	자동 commit 또는 매뉴얼 commit 모드 사용을 지정하는 SQLINTEGER 값이다. - SQL_AUTOCOMMIT_OFF: 드라이버는 매뉴얼 commit 모드를 사용하고, 응용 프로그램은 SQLEndTran으로 트랜잭션을 명시적으로 commit 하거나 rollback 해야 한다. - SQL_AUTOCOMMIT_ON: 드라이버는 자동 commit 모드를 사용한다. 각 명령문은 실행 즉시 commit된다. 연결에 대해 열린 모든 트랜잭션은 SQL_ATTR_AUTOCOMMIT이

Attribute	ValuePtr contents
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	SQL_AUTOCOMMIT_ON으로 설정될 때 commit된다. 일부 데이터 소스는 연결에 명령문이 commit되는 시점에 접근 플랜을 삭제하고 커서를 닫는다. 자동 commit 모드는 각 non-query 명령문이 실행된 후 또는 커서가 질의에 대해 닫혀있을 때 이런 현상을 발생시킬 수 있다. 자세한 내용은 SQLGetInfo의 SQL_CURSOR_COMMIT_BEHAVIOR와 SQL_CURSOR_ROLLBACK_BEHAVIOR 정보 타입을 참조한다. 일괄 처리가 자동 commit 모드에서 실행될 때, 두 가지가 가능하다. 전체 일괄 처리가 자동 처리 가능 단위로 처리되거나 일괄 처리에서 각 명령문이 자동 처리 가능 단위로 처리된다. 특정 데이터 소스는 이러한 두 행위를 지원하면서 하나를 선택하도록할 것이다. 일괄 자동 commit 될지, 일괄 처리 내의 각 개별 명령문이 자동 처리될지 여부는 드라이버가 정의한다.
SQL_ATTR_CONNECTION_DEAD(ODBC 3.5)	드라이버에서 지원하지 않는다.
SQL_ATTR_CONNECTION_TIMEOUT (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_CURRENT_CATALOG (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_DBC_INFO_TOKEN (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ENLIST_IN_DTC (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	응용 프로그램으로 돌아가기 전에 로그인 요청이 완료될 때까지 기다린 시간 (초)에 대응하는 SQLINTEGER 값이다. 기본값은 드라이버에 따라 다르다. ValuePtr이 0이면, 시간 제한은 비활성화되고 연결 시도는 무한정 대기한다. 지정된 로그인 요청 제한 시간이 최대 로그인 제한 시간을 초과할 경우, 드라이버는 값을 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다.
SQL_ATTR_METADATA_ID (ODBC 3.0)	카탈로그 함수의 문자열 인자를 처리하는 방법을 결정하는 SQLINTEGER 값이다. 기본값은 SQL_FALSE이다. SQL_TRUE면 카탈로그 함수의 문자열 인자는 식별자로 간주되고 대소문자를 구분하지 않는다. 구분되지 않은 문자열의 경우, 드라이버는 모든 후행 공백을 제거하고 문자열을 대문자로 바꾼다. 구분된 문자열의 경우, 드라이버는 선행 또는 후행 공백을 제거하고 분리 문자 사이의 문자 그대로를 갖는다. 이러한 인자들 중 하나가 null 포인터이면, 함수는 SQL_ERROR와 SQLSTATE HY009 (invalid use of null pointer)를 반환한다. SQL_FALSE면 카탈로그 함수의 문자열 인자는 식별자로 간주하지 않고, 대소문자를 구분한다. 인자에 따라 문자열 패턴으로 처리될 수 있고 처리되지 않을 수도 있다. 값목록을 갖는 SQLTables의 TableType 인자는 이 속성에 의해 영향받지 않는다. SQL_ATTR_METADATA_ID는 명령문 단계에서 설정될 수 있다. (이것은 명령문 속성의 유일한 연결 속성이다.)

Attribute	ValuePtr contents
	자세한 내용은 카탈로그 함수의 인자 를 참조한다.
SQL_ATTR_OLDPWD	이전 암호 문자열에 대한 SQLPOINTER 이다. 이 값은 쓰기 전용이고, 서버에 연결하기 전에 설정되어야 한다.
SQL_ATTR_ODBC_CURSORS (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_PACKET_SIZE (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_QUIET_MODE (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_TRACE (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_TRACEFILE (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_TRANSLATE_LIB (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_TRANSLATE_OPTION (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	현재 연결에 대한 트랜잭션 고립 레벨을 설정하는 32-bit 비트마스크이다. 응용 프로그램은 연결에서 열린 모든 트랜잭션을 commit 하거나 rollback 하기 위해 SQLSetConnectAttr을 호출하기 전에 이 옵션을 사용하여 SQLEndTran을 호출해야 한다. ValuePtr에 유효한 값은 InfoType이 SQL_TXN_ISOLATION_OPTIONS인 SQLGetInfo를 호출하여 결정할 수 있다. 트랜잭션 고립 레벨에 대한 자세한 내용은 SQLGetInfo의 SQL_DEFAULT_TXN_ISOLATION 정보 타입을 참조한다.

[1] 이러한 함수는 설명자가 응용 프로그램 설명자가 아닌 구현 설명자인 경우에만 비동기적으로 호출될 수 있다.

SQLSetConnectOption

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.0의 SQLSetConnectOption 함수는 ODBC 3.x에서 SQLSetConnectAttr로 대체되었다.
자세한 내용은 **SQLSetConnectAttr**을 참조한다.

SQLSetCursorName

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLSetCursorName은 활성화된 명령문에 커서 이름을 연결한다. 응용 프로그램이 SQLSetCursorName을 호출하지 않으면 드라이버는 SQL 명령문 처리에 필요한 커서 이름을 생성한다.

구문

```
SQLRETURN SQLSetCursorName(  
    SQLHSTMT      StatementHandle,  
    SQLCHAR *      CursorName,  
    SQLSMALLINT    NameLength);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CursorName

[입력] 커서 이름이다. 효율적 처리를 위해 커서 이름은 선행 공간이나 후행 공간을 포함하지 않아야 하고 만일 커서 이름이 제한된 식별자를 포함할 경우, 구분자는 커서 이름의 첫 번째 문자에 위치해야 한다.

NameLength

[입력] *CursorName 문자열의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)

SQLSTATE	Error	설명
01004	String data, right truncated	커서 이름 길이가 최대값을 초과했고 최대 허용 개수만큼의 문자열만 사용되었다.
24000	Invalid cursor state	StatementHandle에 해당하는 명령문이 이미 실행중이거나 위치 지정 커서로 사용 중이다.
34000	Invalid cursor name	*CursorName에 명시된 커서 이름이 드라이버의 최대치를 초과했거나 SQLCUR 또는 SQL_CUR 로 시작했기 때문에 유효하지 않다.
3C000	Duplicate cursor name	*CursorName에 명시된 이름이 이미 존재한다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.
HY001	Memory allocation error	메모리 할당 에러이다.
HY009	Invalid use of null pointer	CursorName 인자가 NULL 포인터 이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기 수행 함수가 호출되었고 SQLSetCursorName 함수가 호출됐을 때 이 비동기 수행 함수가 여전히 실행 중이었다. SQLSetCursorName 함수가 호출되었고 StatementHandle에 대해 비동기 실행 함수가 호출되었다. StatementHandle에 대해 SQLExecute, SQLExeDirect, SQLBulkOperations 또는 SQLSetPos 함수가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY090	Invalid string or buffer length	NameLength 인자가 0보다 작다. (SQL_NTS는 아니다.)
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 SQLSetConnectAttr의 SQL_ATTR_CONNECTION_TIMEOUT 을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.

설명

커서 이름은 위치 지정 갱신 명령문이나 위치 지정 삭제 명령문들에서만 사용된다. (예: UPDATE table-name ... WHERE CURRENT OF cursor-name). 응용 프로그램이 쿼리 명령문을 실행할 때 SQLSetCursorName을 호출하여 커서 이름을 정의하지 않을 경우, 드라이버는 18 문자의 길이를 초과하지 않고 SQL_CUR로 시작하는 이름을 생성한다.

모든 커서 이름은 연결 안에서 반드시 고유해야 한다. 커서 이름의 최대 길이는 드라이버에서 정의한다. 최대 상호 운용성을 위해 응용 프로그램은 커서 이름을 18문자 미만으로 제한하는 것이 좋다. ODBC 3.x 에서 커서 이름이 double quote (") 식별자로 둘러싸여 있을 경우, 대소문자를 구별하는 것으로 간주하고 이것은 SQL 구문이 허용하지 않는 문자 또는 공백이나 예약어처럼 특별하게 취급되는 문자를 포함할 수 있다. 커서 이름의 대소문자를 반드시 구분해야 할 경우, double quote (") 식별자로 둘러싸야 한다.

커서 이름은 SQLFreeHandle을 사용하여 관련된 명령문을 삭제할 때까지 유지된다. 커서가 할당되거나 준비 (prepared) 상태에 있는 명령문의 커서 이름을 다시 부여하기위해 SQLSetCursorName을 호출할 수 있다.

SQLSetDescField

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLSetDescField는 설명자 레코드의 한 개 필드값을 설정한다.

구문

```
SQLRETURN SQLSetDescField(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT    RecNumber,
    SQLSMALLINT    FieldIdentifier,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength);
```

인자

DescriptorHandle

[입력] 설명자 핸들이다.

RecNumber

[입력] 응용 프로그램에서 설정하고자 하는 필드를 포함하는 설명자 레코드를 가르킨다. 설명자 레코드는 0부터 시작하고 레코드 숫자 0은 북마크 레코드이다. RecNumber 인자는 헤더의 필드에 대해 무시된다.

FieldIdentifier

[입력] 설정될 설명자의 필드를 가르킨다.

ValuePtr

[입력] 설명자 정보 또는 정수값을 포함하는 버퍼를 나타낸다. 데이터 형식은 FieldIdentifier 값에 따라 다르다. ValuePtr이 정수값일 경우, FieldIdentifier 인자값에 따라 8바이트 (SQLLEN), 4바이트 (SQLINTEGER), 2바이트 (SQLSMALLINT)로 간주된다.

BufferLength

[입력] FieldIdentifier가 ODBC 정의 필드이고 ValuePtr이 문자열이나 이진 버퍼를 가르킬 경우, 이 인자는 반드시 *ValuePtr의 길이가 되어야 한다. 문자열 데이터에 대해 이 인자가 문자열의 바이트 수를 포함해야 한다.

FieldIdentifier가 ODBC 정의 필드이고 ValuePtr이 정수일 경우, BufferLength는 무시된다.

FieldIdentifier가 드라이버 정의 필드인 경우 응용 프로그램은 BufferLength 인자를 설정하여 드라이버 관리자에 필드의 특성을 나타낸다. BufferLength는 다음과 같은 값을 가질 수 있다.

- ValuePtr이 문자열 포인터일 경우, BufferLength는 문자열의 길이이거나 SQL_NTS이다.
- ValuePtr이 이진 버퍼 포인터일 경우, 응용 프로그램은 BufferLength에 SQL_LEN_BINARY_ATTR(length) 매크로의 결과를 저장한다. 이 BufferLength는 음수값을 저장한다.
- ValuePtr이 고정길이 값을 포함할 경우, BufferLength는 SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT 또는 SQL_IS_USMALLINT 중 하나이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	드라이버가 (ValuePtr 값이 정수일 때) *ValuePtr에 명시된 값을 지원하지 않거나 ValuePtr의 값 또는 *ValuePtr이 작업 구현 조건상 유효하지 않아 드라이버가 유사한 값으로 대체했다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
07009	Invalid descriptor index	FieldIdentifier 인자가 레코드 필드이고 RecNumber 인자가 0일 때 DescriptorHandle 인자가 IPD 핸들을 참조했다. RecNumber 인자는 0보다 작고, DescriptorHandle 인자가 ARD 또는 APD를 참조했다. RecNumber 인자가 매개 변수의 최대값 또는 데이터 소스가 지원하는 column보다 크고 DescriptorHandle 인자가 APD 또는 ARD를 참조했다. FieldIdentifier 인자가 SQL_DESC_COUNT 이고 *ValuePtr 인자가 0보다 작다. RecNumber 인자가 0 과 같고 DescriptorHandle 인자가 내재적으로 할당된 APD를 참조했다. (이 오류는 명백하게 할당된 응용 프로그램 설명자가 APD 또는 ARD 인지 실행되기 전에는 알 수 없기 때문에 명백하게 할당된 응용 프로그램 설명자에 대해서는 발생하지 않는다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
22001	String data, right truncated	FieldIdentifier 인자가 SQL_DESC_NAME이고 BufferLength 인자는 SQL_MAX_IDENTIFIER_LEN보다 큰 값이다.
HY000	General error	특정 SQLSTATE가 없는 에러이다.

SQLSTATE	Error	설명
HY001	Memory allocation error	메모리 할당 에러이다.
HY010	Function sequence error	StatementHandle과 관련된 DescriptorHandle에 대해 비동기 수행 함수가 호출되었고 SQLSetDescField 함수가 호출되었을 때 이 비동기 수행 함수가 여전히 실행되고 있었다. DescriptorHandle과 관련된 StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. Data-at-execution 매개 변수 또는 column의 모든 데이터를 보내기 전에 이 함수가 호출되었다. DescriptorHandle과 관련된 연결 핸들에 대해 비동기 수행 함수가 호출되었다. 이 비동기 수행 함수는 SQLSetDescField가 호출됐을 때 여전히 실행중이었다. DescriptorHandle과 관련된 명령문 핸들 중 하나에 대해 SQLExecute, SQLExecDirect 또는 SQLMoreResults로 호출되었고 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 스트림 된 모든 매개 변수들에 대한 데이터가 회수되기 전에 이 함수가 호출되었다.
HY013	Memory management error	내부 메모리에 접근할 수 없거나 사용할 수 있는 메모리 용량이 부족하다.
HY016	Cannot modify an implementation row descriptor	DescriptorHandle 인자가 IRD와 관련되어 있고 FieldIdentifier 인자가 SQL_DESC_ARRAY_STATUS_PTR 또는 SQL_DESC_ROWS_PROCESSED_PTR이 아니다.
HY021	Inconsistent descriptor information	SQL_DESC_TYPE과 SQL_DESC_DATETIME_INTERVAL_CODE 필드들은 ODBC SQL, 드라이버별 SQL 형식 또는 ODBC C에 유효한 형식이 아니다. 무결성 검사시에 확인된 설명자 정보에 일관성이 없다.
HY090	Invalid string or buffer length	*ValuePtr이 문자열이고 BufferLength가 0보다 작다. (SQL_NTS가 아니다.) 드라이버를 ODBC 2.x 드라이버, 설명자를 ARD, ColumnNumber 인자를 0으로 설정했을 때 BufferLength에 명시된 값이 4가 아니다.
HY091	Invalid descriptor field identifier	FieldIdentifier 인자에 명시된 값이 ODBC 정의 필드 또는 구현 정의 값이 아니다. FieldIdentifier 인자는 DescriptorHandle 인자에 대해 유효하지 않다. FieldIdentifier 인자는 ODBC 정의 필드이고 읽기 전용이다.
HY092	Invalid attribute/option identifier	*ValuePtr 값이 FieldIdentifier 인자에 대해 유효하지 않다. FieldIdentifier 인자가 SQL_DESC_UNNAMED이었고 ValuePtr이 SQL_NAMED였다.
HY105	Invalid parameter type	SQL_DESC_PARAMETER_TYPE 필드에 명시된 값이 유효하지 않다. (자세한 내용은 SQLBindParameter의 InputOutputType 인자를 참조한다.)

SQLSTATE	Error	설명
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 <code>SQLEndTran</code> 함수를 참조한다.
HYT01	Connection timeout expired	데이터 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 <code>SQLSetConnectAttr</code> 의 <code>SQL_ATTR_CONNECTION_TIMEOUT</code> 을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.

설명

응용 프로그램은 `SQLSetDescField`를 한 번 호출할 때마다 하나의 설명자 필드를 설정할 수 있다. 한 번의 호출로 한 개 설명자의 한 개 필드를 설정할 수 있다. 설정 가능한 필드일 경우, 이 함수를 호출하여 설명자 유형의 필드를 설정할 수 있다.

노트

`SQLSetDescField` 호출에 실패하면 `RecNumber` 인자로 식별되는 설명자 레코드의 내용물은 정의되지 않는다.

한 번의 함수 호출로 여러 설명자 필드들을 설정하기 위해 다른 함수들을 호출할 수 있다. `SQLSetDescRec` 함수는 데이터 형식과 바인딩된 column이나 매개 변수 버퍼에 영향을 주는 다양한 필드들을 설정할 수 있다. (`SQL_DESC_TYPE`, `SQL_DESC_DATETIME_INTERVAL_CODE`, `SQL_DESC_OCTET_LENGTH`, `SQL_DESC_PRECISION`, `SQL_DESC_SCALE`, `SQL_DESC_DATA_PTR`, `SQL_DESC_OCTET_LENGTH_PTR`, `SQL_DESC_INDICATOR_PTR`)

`SQLBindCol` 또는 `SQLBindParameter`는 column과 매개 변수의 완전한 설정을 위해 사용될 수 있다. 이 함수들은 한 번의 함수 호출로 설명자 필드들의 그룹을 설정할 수 있다.

바인딩 버퍼들을 변경하기 위해 바인딩 포인터들에 옵셋을 추가하여 `SQLSetDescField`를 호출할 수 있다. (`SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, `SQL_DESC_OCTET_LENGTH_PTR`) 이것은 `SQL_DESC_DATA_TYPE`이 다른 필드들을 변경하지 않고 `SQL_DESC_DATA_PTR`을 변경하듯이 응용 프로그램이 `SQLBindCol`이나 `SQLBindParameter`를 호출하지 않고 바인딩 버퍼들을 변경하도록 할 수 있다.

응용 프로그램이 `SQLSetDescField`를 호출하여 `SQL_DESC_COUNT`나 `SQL_DESC_DATA_PTR`, `SQL_DESC_OCTET_LENGTH_PTR` 또는 `SQL_DESC_INDICATOR_PTR`과 같은 지연된 필드들이 아닌 다른 필드들을 설정할 경우, 레코드 바인딩은 해제된다.

적절한 `FieldIdentifier`와 함께 `SQLSetDescField`를 호출하여 설명자 헤더 필드를 설정할 수 있다. 많은 헤더 필드들

이 명령문 속성이기도 하므로 SQLSetStmtAttr을 호출하여 설정할 수도 있다. 이것은 응용 프로그램이 설명자 핸들을 얻지 않고도 설명자 필드를 우선적으로 설정할 수 있도록 한다. SQLSetDescField를 호출하여 헤더 필드를 설정할 경우, RecNumber 인자는 무시된다.

0인 RecNumber는 북마크 필드를 설정하는데 사용된다.

노트

명령문 속성 SQL_ATTR_USE_BOOKMARKS는 북마크 필드들을 설정하기 위해 SQLSetDescField를 호출하기 전에 설정해야 한다. 반드시 해야 하는 것은 아니지만 권장 사항이다.

설명자 필드들의 설정 순서

SQLSetDescField를 호출하여 설명자 필드들을 설정할 때 응용 프로그램은 다음 명시된 순서를 따라야 한다.

1. 응용 프로그램은 SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE 또는 SQL_DESC_DATETIME_INTERVAL_CODE 필드를 우선적으로 설정해야 한다.
2. 이 필드들 중 하나가 설정된 후에 응용 프로그램은 데이터 형식의 속성을 설정할 수 있고 드라이버는 데이터 형식 설정 필드들을 데이터 형식에 대해 적절한 기본값으로 설정할 수 있다. 형식 속성 필드의 자동 디폴트화는 응용 프로그램이 데이터형식을 지정한 후 설명자가 언제나 사용할 준비가 되어 있음을 보장한다. 응용 프로그램이 명시적으로 데이터 형식 속성을 설정하면 기본 속성은 덮어 쓰여진다.
3. 1단계의 필드 중 하나와 데이터 형식 속성이 설정된 후에 응용 프로그램은 SQL_DESC_DATA_PTR을 설정할 수 있다. 이는 즉각 설명자 필드들에 대한 일관성을 검사한다. 만약 응용 프로그램이 데이터 형식이나 속성들을 변경한 후, SQL_DESC_DATA_PTR 필드를 설정하면 드라이버는 SQL_DESC_DATA_PTR을 NULL 포인터로 설정하고 레코드의 바인딩을 해제한다. 이것은 설명자 레코드를 사용할 수 있게 되기 전에 응용 프로그램이 순차적으로 적절한 절차를 완료하도록 강제한다.

설명자 필드의 초기화

설명자가 할당되었을 때 설명자 필드들은 기본값으로 초기화, 기본값 없는 초기화 또는 설명자 형식에 정의되지 않은 값으로 초기화될 수 있다. 다음 표는 설명자 형식별로 각 필드의 초기화를 기술한다. D는 기본값을 가지는 초기화 필드를 나타낸다. ND는 기본값 없는 초기화 필드를 나타낸다. 숫자의 경우 필드의 기본값이다. 표에서는 필드의 읽기/쓰기 또는 읽기전용 여부 또한 기술하고 있다.

IRD의 필드들은 명령문이 준비되거나 (prepared) 실행되고 IRD가 생성된 후에 기본값을 가질 수 있고, 명령문 핸들이나 설명자가 할당되었을 때 가질 수 있는 것이 아니다. IRD가 생성될 때까지 IRD 필드에 대한 모든 접근 시도는 오류를 반환한다.

일부 설명자 필드들은 모든 설명자 형식 (ARD, IRD, APD, IPD)에 대해서가 아니라 하나 이상의 설명자 형식에 대해 정의된다. 필드가 설명자의 형식에 대해 정의되어 있지 않을 경우 이는 해당 설명자를 사용하는 함수에서 필요로 하지 않는다.

SQLGetDescField가 접근 가능한 필드들을 반드시 SQLSetDescField로 설정할 수 있는 것은 아니다. SQLSetDescField로 설정할 수 있는 필드들은 다음 표와 같다.

해더 필드들의 초기화는 다음 표에 표기되어 있다.

헤더 필드 이름	형식	R/W	기본값
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	ARD:R APD:R IRD:R IPD: R	ARD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit APD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit IRD: SQL_DESC_ALLOC_AUTO IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLULEN	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: [1] APD: [1] IRD: Unused IPD: Unused
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT*	ARD:R/W APD:R/W IRD:R/W IPD:R/W	ARD: Null ptr APD: Null ptr IRD: Null ptr IPD: Null ptr
SQL_DESC_BIND_OFFSET_PTR	SQLLEN*	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_BIND_TYPE	SQLINTEGER	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: SQL_BIND_BY_COLUMN APD: SQL_BIND_BY_COLUMN IRD: Unused IPD: Unused
SQL_DESC_COUNT	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: 0 APD: 0 IRD: D IPD: 0
SQL_DESC_ROWS_PROCESSED_PTR	SQLULEN*	ARD:Unused APD:Unused IRD:R/W IPD:R/W	ARD: Unused APD: Unused IRD: Null ptr IPD: Null ptr

[1] 필드들은 드라이버가 IPD를 자동으로 생성했을 때만 정의된다. 그렇지 않을 경우 그것들은 정의되지 않는다. 응용 프로그램이 이 필드들을 설정하려 시도하면 SQLSTATE HY091 (invalid descriptor field identifier)가 반환된다.

레코드 필드들의 초기화는 다음 표에 기술되어 있다.

레코드 필드 이름	형식	R/W	기본값
		ARD: Unused APD: Unused	ARD: Unused APD: Unused

레코드 필드 이름	형식	R/W	기본값
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	IRD: R IPD: Unused	IRD: D IPD: Unused
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_CATALOG_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CHAR_LENGTH_UNITS	SQLSMALLINT	ARD: Unused APD: Unused IRD: Unused IPD: W	ARD: Unused APD: Unused IRD: ND IPD: Unused
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_ DEFAULT APD: SQL_C_ DEFAULT IRD: D IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused [2]
SQL_DESC_DATETIME_INTERVAL_CODE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLLEN	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
		ARD: Unused	ARD: Unused

레코드 필드 이름	형식	R/W	기본값
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	APD: Unused IRD: R IPD: R	APD: Unused IRD: D IPD: D [1]
SQL_DESC_INDICATOR_PTR	SQLLEN *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_LABEL	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LENGTH	SQLULEN	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R/W	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_NULLABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH	SQLLEN	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND

레코드 필드 이름	형식	R/W	기본값
SQL_DESC_OCTET_LENGTH_PTR	SQLLEN *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	ARD: Unused APD: Unused IRD: Unused IPD: R/W	ARD: Unused APD: Unused IRD: Unused IPD: D =SQL_PARAM_INPUT
SQL_DESC_PRECISION	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_ROWVER	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: ND IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_SCHEMA_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_SEARCHABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_TABLE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
		ARD: Unused APD: Unused	ARD: ND APD: ND

레코드 필드 이름	형식	R/W	기본값
SQL_DESC_UNNAMED	SQLSMALLINT	IRD: R IPD: R/W	IRD: D IPD: ND
SQL_DESC_UNSIGNED	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_UPDATABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

[1] 필드들은 드라이버가 IPD를 자동으로 생성했을 때만 정의된다. 그렇지 않을 경우 그것들은 정의되지 않는다. 응용 프로그램이 이 필드들을 설정하려 시도하면 SQLSTATE HY091 (invalid descriptor field identifier)가 반환된다.

[2] IPD의 SQL_DESC_DATA_PTR 필드는 강제로 일관성을 체크한다. 추후에 SQLGetDescField 또는 SQLGetDescRec를 호출할 때 드라이버가 SQL_DESC_DATA_PTR로 설정한 값을 반환할 필요가 없다.

FieldIdentifier 인자

FieldIdentifier 인자는 설정될 설명자 필드를 나타낸다. 설명자는 설명자 헤더의 다음 헤더 필드 부분에 기술된 헤더 필드들로 구성되어 있고, 설명자 레코드들은 다음 헤더 필드에 설명된 레코드 필드들로 구성되어 있다.

헤더 필드

각 설명자는 다음 필드들로 구성된다.

SQL_DESC_ALLOC_TYPE[All] (읽기 전용)

읽기 전용 SQLSMALLINT인 이 헤더 필드는 서술자가 드라이버에 의해 자동으로 할당되는지 또는 응용 프로그램에 의해 명시적으로 할당되는지 여부를 명시한다. 응용 프로그램은 이 필드를 얻을 수는 있지만 수정할 수는 없다. 드라이버가 설명자를 자동으로 할당했을 때 필드는 드라이버에 의해 SQL_DESC_ALLOC_AUTO로 설정된다.

SQL_DESC_ARRAY_SIZE[Application descriptors]

ARD에 있고 SQLULEN인 이 헤더 필드는 row 집합의 row 개수를 명시한다. 이것은 SQLFetch, SQLFetchScroll을 호출하여 반환되거나 SQLBulkOperations, SQLSetPos를 호출하여 작동되는 row들의 개수이다.

APD에 있고 SQLULEN인 이 헤더 필드는 각 매개 변수 값의 개수를 명시한다.

이 필드의 기본값은 1이다. SQL_DESC_ARRAY_SIZE가 1보다 클 경우 APD 또는 ARD의 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR과 SQL_DESC_OCTET_LENGTH_PTR은 배열을 가리킨다. 각 배열의 상수는 이 필드의 값과 같다.

ARD에 있는 이 필드는 SQL_ATTR_ROW_ARRAY_SIZE 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다. APD의 필드 역시 SQL_ATTR_PARAMSET_SIZE 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다.

SQL_DESC_ARRAY_STATUS_PTR[AI]

각 설명자 형식에 대해 SQLUSMALLINT*인 헤더 필드는 SQLUSMALLINT 값의 배열을 가리킨다. 이 배열들의 이름은 row 상태 배열 (IRD), 매개 변수 상태 배열 (IPD), row 연산 배열 (ARD), 매개 변수 연산 배열 (APD)이다.

IRD에 있는 이 헤더 필드는 SQLBulkOperations, SQLFetch, SQLFetchScroll 또는 SQLSetPos를 호출한 후의 상태값을 포함하는 row 상태 배열을 가리킨다. 응용 프로그램은 SQLUSMALLINT 배열을 할당하고 이 필드가 배열을 가리키게 해야 한다. 이 필드는 기본적으로 NULL 포인터이다. 드라이버는 SQL_DESC_ARRAY_STATUS_PTR 필드를 NULL 포인터로 설정하지 않는 한 배열을 생성한다.

주의

응용 프로그램이 IRD의 SQL_DESC_ARRAY_STATUS_PTR 필드가 가리키는 row 상태 배열의 요소들을 설정할 경우, 드라이버의 행동은 정의되지 않는다.

배열은 처음에 SQLBulkOperations, SQLFetch, SQLFetchScroll 또는 SQLSetPos를 호출하여 채워진다. 이 호출이 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 이 필드가 가리키는 배열의 내용은 정의되지 않는다. 배열의 요소들은 다음 값들을 포함할 수 있다.

- SQL_ROW_SUCCESS: Row가 성공적으로 회수되었고 마지막으로 회수된 이후로 아무 변화가 없었다.
- SQL_ROW_SUCCESS_WITH_INFO: Row가 성공적으로 회수되었고 마지막 회수된 이후로 아무 변화가 없었다. 하지만 row에 대한 경고가 반환되었다.
- SQL_ROW_ERROR: Row를 회수하는 동안 오류가 발생했다.
- SQL_ROW_UPDATED: Row가 성공적으로 회수되었고 마지막으로 회수된 이후로 갱신되었다. Row가 다시 회수되었을 경우, 상태는 SQL_ROW_SUCCESS이다.
- SQL_ROW_DELETED: 마지막으로 회수된 이후로 row가 삭제되었다.
- SQL_ROW_ADDED: SQLBulkOperations가 row를 삽입했다. 다시 row가 회수되었을 경우, 상태는 SQL_ROW_SUCCESS이다.
- SQL_ROW_NOROW: Row 집합이 결과 집합의 마지막과 겹쳐진다. 그리고 row 상태 배열의 요소에 대응하여 반환된 row가 없다.

IRD의 이 필드는 SQLSetStmtAttr을 SQL_ATTR_ROW_STATUS_PTR 속성으로 호출하여 설정할 수 있다.

IRD의 SQL_DESC_ARRAY_STATUS_PTR 필드는 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO가 반환된 이후에만 유효하다. 반환 코드가 이들 중 하나가 아닐 경우, SQL_DESC_ROWS_PROCESSED_PTR이 가리키는 것은 정의되지 않는다.

IPD에 있는 이 헤더 필드는 SQLExecute나 SQLExecDirect를 호출한 후 매개 변수 각각의 상태 정보를 포함하는 매개 변수 상태 배열을 가리킨다. 만약 SQLExecute나 SQLExecDirect를 호출한 후 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 이 필드가 가리키는 배열의 내용은 정의되지 않는다. 응용 프로그램은 SQLUSMALLINT 배열을 할당하고 이 필드가 배열을 가리키게 해야 한다. 드라이버는 SQL_DESC_ARRAY_STATUS_PTR 필드를 NULL 포인터로 설정하지 않는 한, 배열을 생성한다. 배열의 요소는 다음 값들을 포함한다.

- `SQL_PARAM_SUCCESS`: 이 매개 변수 집합에 대해 SQL 명령문이 성공적으로 실행되었다.
- `SQL_PARAM_SUCCESS_WITH_INFO`: 이 매개 변수 집합에 대해 SQL 명령문이 성공적으로 실행되었지만, 진단 데이터 구조체에 사용가능한 경고 정보가 있다.
- `SQL_PARAM_ERROR`: 이 매개 변수 집합을 처리하는 중에 오류가 발생했다. 추가 오류 정보는 진단 데이터 구조체에 있다.
- `SQL_PARAM_UNUSED`: 이전의 일부 매개 변수가 더 이상의 처리를 중단하는 오류를 발생시켰거나 APD의 `SQL_DESC_ARRAY_STATUS_PTR` 필드가 명시한 배열의 매개 변수 집합에 `SQL_PARAM_IGNORE`가 설정되었기 때문에 매개 변수가 사용되지 않았다.
- `SQL_PARAM_DIAG_UNAVAILABLE`: 진단 정보를 사용할 수 없다. 예를 들어 드라이버가 매개 변수의 배열을 단일한 것으로 취급하여 오류 정보의 level을 생성하지 않았다.

IPD에 있는 이 필드는 `SQLSetStmtAttr`을 `SQL_ATTR_PARAM_STATUS_PTR` 속성으로 호출하여 설정할 수 있다.

ARD에 있는 이 필드는 해당 row를 `SQLSetPos` 연산에서 무시할지 여부를 설정하기 위해 응용 프로그램이 설정한 값들에 대한 row 연산 배열을 가리킨다. 배열의 요소들은 다음 값들을 포함할 수 있다.

- `SQL_ROW_PROCEED`: Row가 `SQLSetPos`를 이용한 벌크 연산에 포함된다. (이 설정은 연산이 해당 row에서 발생한다고 보장하지는 않는다. Row가 IRD row 상태 배열의 `SQL_ROW_ERROR` 상태를 가질 경우, 드라이버는 row에 대한 연산을 수행할 수 없다.)
- `SQL_ROW_IGNORE`: Row는 `SQLSetPos`를 이용한 벌크 연산에서 제외된다.

배열 요소가 설정되지 않은 경우 모든 row가 벌크 연산에 포함된다. ARD의 `SQL_DESC_ARRAY_STATUS_PTR` 필드 안의 값이 NULL 포인터이면 모든 row는 벌크 연산에 포함된다. 포인터가 유효한 배열을 가리키고 있고 배열의 모든 요소가 `SQL_ROW_PROCEED`인 것과 같이 해석된다. 배열의 모든 요소를 `SQL_ROW_IGNORE`로 설정할 경우, 무시된 row에 대한 row 상태 배열 안의 값은 갱신되지 않는다.

ARD 안에 있는 이 필드 역시 `SQLSetStmtAttr`을 `SQL_ATTR_ROW_OPERATION_PTR` 속성으로 호출하여 설정할 수 있다.

APD에 있는 이 헤더 필드는 `SQLExecute`나 `SQLExecDirect`가 호출될 때 이 매개 변수 집합을 무시할지 여부를 나타내기 위해 응용 프로그램이 설정할 수 있는 값의 매개 변수 연산배열을 가리킨다. 배열에 있는 요소들은 다음 값들을 포함할 수 있다.

- `SQL_PARAM_PROCEED`: 매개 변수 집합은 `SQLExecute` 또는 `SQLExecDirect` 호출에 포함된다.
- `SQL_PARAM_IGNORE`: 매개 변수 집합은 `SQLExecute` 또는 `SQLExecDirect` 호출에 포함되지 않는다.

배열 요소가 설정되지 않은 경우, 배열의 모든 매개 변수 집합들은 `SQLExecute` 또는 `SQLExecDirect`를 호출하는데 사용된다. APD의 `SQL_DESC_ARRAY_STATUS_PTR` 필드값이 NULL 포인터일 경우, 모든 매개 변수 집합이 사용된다. 포인터가 유효한 배열이고 모든 요소가 `SQL_PARAM_PROCEED`인 배열을 가리키는 것처럼 해석된다.

APD에 있는 이 필드 역시 `SQL_ATTR_PARAM_OPERATION_PTR` 속성으로 `SQLSetStmtAttr`을 호출하여 설정할 수 있다.

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

SQLLEN*인 이 헤더 필드는 바인딩의 offset을 가리킨다. 이것은 기본적으로 NULL 포인터로 설정된다. 이 필드가 NULL 포인터가 아닐 경우 드라이버는 fetching 할 때 포인터를 역참조하고 역참조된 값을 설명자 레코드 (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR 및 SQL_DESC_OCTET_LENGTH_PTR)의 NULL이 아닌 값을 갖는 지연된 필드에 추가하고 바인딩 할 때 새 포인터 값을 사용한다.

바인딩 offset은 항상 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR과 SQL_DESC_OCTET_LENGTH_PTR 필드들에 직접 추가된다. 만약 offset이 다른 값으로 변경되었을 경우, 새로운 값이 각 설명자 필드 값으로 직접적으로 계속 추가된다. 새로운 offset이 필드의 이전 offset 값에 추가되는 것이 아니다.

이 필드는 지연된 필드이다. 이 필드 설정되는 시점에 사용되지 않고 추후에 데이터 버퍼들의 주소를 확인할 필요가 있을 때 드라이버에 의해 사용된다.

ARD 안의 이 필드 역시 SQL_ATTR_ROW_BIND_OFFSET_PTR 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다. ARD에 있는 이 필드는 SQL_ATTR_PARAM_BIND_OFFSET_PTR 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다.

자세한 내용은 SQLFetchScroll 또는 SQLBindParameter를 참조한다.

SQL_DESC_BIND_TYPE[Application descriptors]

SQLINTEGER인 이 헤더 필드는 바인딩의 방향을 설정하는 데 사용된다.

ARD에 있는 이 필드는 SQLFetchScroll 또는 SQLFetch가 관련된 명령문 핸들에 호출되었을 때의 바인딩 방향을 명시한다.

Column들에 대한 column 방향 바인딩을 선택하려면 이 필드를 SQL_BIND_BY_COLUMN (기본값)으로 설정해야 한다.

ARD에 있는 이 필드 역시 SQL_ATTR_ROW_BIND_TYPE 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다.

이 필드는 동적 매개 변수에 사용될 바인딩 방향을 명시한다.

매개 변수에 대한 column 방향 바인딩을 선택하려면 이 필드를 SQL_BIND_BY_COLUMN (기본값)으로 설정해야 한다.

이 필드 역시 SQL_ATTR_PARAM_BIND_TYPE 속성으로 SQLSetStmtAttr을 호출하여 설정할 수 있다.

SQL_DESC_COUNT[All]

SQLSMALLINT인 이 헤더 필드는 데이터를 포함하는 최상위 레코드의 1-based 인덱스를 명시한다. 드라이버가 설명자에 데이터 구조체를 설정할 때 중요한 레코드의 수가 얼마나 되는지 보여주기 위해서 반드시 SQL_DESC_COUNT를 설정해야 한다. 응용 프로그램이 데이터 구조체의 인스턴스를 할당할 때 레코드를 위해 얼마나 큰 공간을 예약해야할지 명시할 필요는 없다. 응용 프로그램이 레코드들의 내용을 명시하는 것처럼 드라이버는 설명자 핸들이 충분한 크기의 데이터 구조체를 나타내도록 보장하기 위해 필요한 요청 작업을 수행한다.

SQL_DESC_COUNT가 바인딩 된 모든 데이터 column 또는 모든 매개 변수의 개수가 아닌 최상위 레코드의 개수이다. 최상위 column 또는 매개 변수의 바인딩이 해제될 경우, SQL_DESC_COUNT는 다음 최상위 column 또는 매개 변수의 개수로 바뀐다. 최상위 column이나 매개 변수 보다 작은 수의 column이나 매개 변수의 바인딩이 해제될 경우 (TargetValuePtr 인자를 NULL 포인터로 설정하고 SQLBindCol을 호출하거나 ParameterValuePtr 인자를 NULL 포인터로 설정하여 SQLBindParameter를 호출할 경우) SQL_DESC_COUNT

는 변경되지 않는다. 추가되는 column이나 매개 변수가 데이터를 포함하는 최상위 레코드보다 큰 수로 바인딩될 경우, 드라이버는 자동으로 SQL_DESC_COUNT 필드의 값을 증가시킨다. SQL_UNBIND 옵션으로 SQLFreeStmt를 호출하여 모든 column들에 대한 바인딩을 해제할 경우, ARD와 IRD 안에 있는 SQL_DESC_COUNT 필드는 0으로 설정된다. SQL_RESET_PARAMS 옵션으로 SQLFreeStmt를 호출할 경우, APD와 IPD에 있는 SQL_DESC_COUNT 필드들은 0으로 설정된다.

SQL_DESC_COUNT 값은 응용 프로그램에서 SQLSetDescField를 호출하여 명시적으로 설정할 수 있다. SQL_DESC_COUNT 값이 명확하게 감소할 경우, 새 SQL_DESC_COUNT 값보다 큰 값을 가지는 모든 레코드들이 효과적으로 삭제된다. ARD의 SQL_DESC_COUNT 필드값이 명시적으로 0으로 설정될 경우, 바인딩된 북마크 column을 제외한 모든 버퍼가 해제된다. ARD의 이 필드의 레코드 개수는 바인딩된 북마크 column을 포함하지 않는다. 북마크 column의 바인딩을 해제하는 유일한 방법은 SQL_DESC_DATA_PTR 필드를 NULL 포인터로 설정하는 것이다.

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

IRD의 SQLULEN*인 이 헤더 필드는 SQLFetch나 SQLFetchScroll을 호출한 후에 회수된 row들의 개수, 또는 SQLBulkOperations나 SQLSetPos를 호출하여 수행한 벌크 연산의 영향을 받은 row들의 개수 및 오류 row들의 개수를 포함하는 버퍼를 가리킨다.

IPD의 SQLINTEGER*인 이 헤더 필드는 처리된 매개 변수 집합과 오류 개수를 포함하는 버퍼를 가리킨다. NULL 포인터일 경우, 개수를 반환하지 않는다.

SQL_DESC_ROWS_PROCESSED_PTR은 (IRD 필드에 대해) SQLFetch나 SQLFetchScroll을 호출하거나 (IPD 필드에 대해) SQLExecute, SQLExecDirect 또는 SQLParamData를 호출하여 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환받은 후에만 유효하다. 만약 앞의 함수들이 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않으며 SQL_NO_DATA를 반환할 때까지 버퍼의 값은 0으로 설정된다.

ARD에 있는 이 필드 역시 SQLSetStmtAttr을 SQL_ATTR_ROWS_FETCHED_PTR 속성으로 호출하여 설정할 수 있다. APD의 경우에도 SQLSetStmtAttr을 SQL_ATTR_PARAMS_PROCESSED_PTR 속성으로 호출하여 설정할 수 있다.

이 필드가 가리키는 버퍼는 응용 프로그램에 의해 할당된 것이다. 이는 드라이버가 설정한 지연된 출력 버퍼이다. 기본적으로 NULL 포인터가 설정된다.

레코드 필드

각 설명자는 설명자의 형식에 따라 column 데이터 또는 동적 매개 변수 중 하나를 정의한 필드들로 구성된 레코드를 하나 이상 포함한다. 각 레코드는 단일 column 또는 매개 변수의 완전한 명세이다.

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs] (읽기 전용)

읽기 전용 SQLINTEGER인 이 레코드 필드는 column이 자동 증가 column일 경우 SQL_TRUE 값을 가지고 그렇지 않을 경우 SQL_FALSE 값을 가진다. 이 필드는 읽기 전용이지만 자동 증가 column인 경우 반드시 읽기 전용일 필요는 없다.

SQL_DESC_BASE_COLUMN_NAME[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 결과 집합 column의 기본 column 이름을 포함한다. 기본 column 이름이 존재하지 않을 경우, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_TABLE_NAME[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 결과 집합 column의 기본 테이블 이름을 포함한다. 기본 테이블 이름을 정의할 수 없거나 사용할 수 없을 경우, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_CASE_SENSITIVE[Implementation descriptors] (읽기 전용)

읽기 전용 SQLINTEGER인 이 레코드 필드는 column 또는 매개 변수를 정렬하거나 비교할 때 대소문자를 구분할 경우 SQL_TRUE를, 대소문자를 구분하지 않거나 문자가 아닌 column의 경우에는 SQL_FALSE를 포함한다.

SQL_DESC_CATALOG_NAME[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 column을 포함하는 기본 테이블의 카탈로그를 포함한다. Column이 표현식이거나 view의 일부분일 경우 반환값은 드라이버 따라 다르다. 데이터 소스가 카탈로그를 지원하지 않거나 카탈로그를 확인할 수 없을 경우, 이 필드는 빈 문자열을 포함한다.

SQL_DESC_CONCISE_TYPE[AII]

SQLSMALLINT인 이 헤더 필드는 datetime과 interval 데이터 형식을 포함하는 모든 데이터 형식에 대해 간결한 데이터 형식을 지정한다.

SQL_DESC_CONCISE_TYPE, SQL_DESC_TYPE과 SQL_DESC_DATETIME_INTERVAL_CODE 필드의 값은 상호 의존적이다. 필드들 중의 하나에 time이 설정되면 다른 필드들에도 설정해야 한다. SQL_DESC_CONCISE_TYPE은 SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 설정할 수 있다. SQL_DESC_TYPE은 SQLSetDescField 또는 SQLSetDescRec를 호출하여 설정할 수 있다.

SQL_DESC_CONCISE_TYPE을 interval 또는 datetime 데이터 형식 이외의 간결한 데이터 형식으로 설정할 경우, SQL_DESC_TYPE 필드는 같은 값으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 0으로 설정된다.

SQL_DESC_CONCISE_TYPE을 간결한 datetime 또는 interval 데이터 형식으로 설정할 경우, SQL_DESC_TYPE 필드는 상세한 형식 (SQL_DATETIME 또는 SQL_INTERVAL)으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 적절한 서브코드로 설정된다.

SQL_DESC_DATA_PTR [Application descriptors 및 IPD]

이 SQLPOINTER 레코드 필드는 매개변수 값 (APD의 경우) 또는 column 값 (ARD의 경우)을 저장할 변수의 주소를 가리킨다. 이 필드는 deferred field로써, 값을 설정하는 시점이 아니라 나중에 driver가 실제 데이터를 가져올 때 사용된다.

ARD의 SQL_DESC_DATA_PTR 필드에 지정된 column은 SQLBindCol을 호출할 때 TargetValuePtr 인수가 null pointer이거나, SQLSetDescField 또는 SQLSetDescRec를 호출하여 ARD의 SQL_DESC_DATA_PTR 필드를 null pointer로 설정할 경우 unbound 된다. 그러나 다른 필드들은 SQL_DESC_DATA_PTR 필드가 null pointer로 설정되더라도 영향을 받지 않는다.

SQLFetch 또는 SQLFetchScroll을 호출하여 이 필드가 가리키는 버퍼가 채워질 때 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO가 반환되지 않으면 버퍼의 내용은 undefined 상태이다.

APD, ARD 또는 IPD의 SQL_DESC_DATA_PTR 필드가 설정될 때마다 driver는 SQL_DESC_TYPE 필드에 유효한 ODBC C 데이터 타입이나 driver-specific 데이터 타입이 설정되었는지, 그리고 데이터 타입과 관련된 다른 필드들이 일관성을 유지하는지 (consistency) 확인한다. 특히 IPD의 SQL_DESC_DATA_PTR 필드는 오직 consistency 검사를 수행하기 위한 목적으로만 사용된다. 즉, 애플리케이션이 IPD의 SQL_DESC_DATA_

PTR 필드를 설정한 후, 나중에 이 필드에 대해 SQLGetDescField를 호출한다고 하더라도, 반드시 이전에 설정한 값이 반환되는 것은 아니다.

SQL_DESC_DATETIME_INTERVAL_CODE[All]

SQLSMALLINT인 이 레코드 필드는 SQL_DESC_TYPE 필드가 SQL_DATETIME 또는 SQL_INTERVAL일 때 datetime이나 interval 데이터 형식을 명시하기 위한 서브코드를 포함하고 있다. 이것은 SQL과 C 모두 마찬가지이다. 코드는 datetime 형식의 TYPE이나 C_TYPE, interval 형식의 INTERVAL이나 C_INTERVAL을 대체한 CODE를 포함하는 데이터 형식 이름을 포함한다.

응용 프로그램 설명자의 SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE을 SQL_C_DEFAULT로 설정하고 설명자가 명령문 핸들과 관련되어 있지 않을 경우, SQL_DESC_DATETIME_INTERVAL_CODE의 내용은 정의되지 않는다.

이 필드는 다음 표에 나열된 datetime 데이터 형식들을 설정할 수 있다.

Datetime type	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/ SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/ SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIME_WITH_TIMEZONE/ SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_CODE_TIME_WITH_TIMEZONE
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE/ SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_CODE_TIMESTAMP_WITH_TIMEZONE

이 필드는 다음 표에 나열된 interval 데이터 형식들을 설정할 수 있다.

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY/ SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR/ SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE/ SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND/ SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR/ SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE/ SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND/ SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_MINUTE/ SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND/ SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MONUTE_TO_SECOND
SQL_INTERVAL_MONTH/ SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND/ SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR/ SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH/ SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[AI]

SQLINTEGER인 이 레코드 필드는 SQL_DESC_TYPE 필드가 SQL_INTERVAL일 경우, interval leading precision을 포함한다. SQL_DESC_DATETIME_INTERVAL_CODE 필드를 interval 데이터 형식으로 설정할 경우, 이 필드는 기본 interval leading precision을 설정한다.

SQL_DESC_DISPLAY_SIZE[IRDs] (읽기 전용)

읽기 전용 SQLLEN인 이 레코드 필드는 column의 데이터를 보여주기 위해 필요한 최대 문자의 개수를 포함한다.

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors] (읽기 전용)

읽기 전용 SQLSMALLINT인 이 레코드 필드는 column이 exact numeric column이고 0이 아닌 scale과 고정된 precision을 가지고 있을 경우 SQL_TRUE로, 그렇지 않을 경우 SQL_FALSE로 설정된다.

SQL_DESC_INDICATOR_PTR[Application descriptors]

ARD에 있고 SQLLEN*인 이 레코드 필드는 지시자 변수를 나타낸다. 이 변수는 column 값이 NULL일 경우, SQL_NULL_DATE를 포함한다. APD의 경우, 표시자 변수는 동적 인자들을 NULL로 명시하기 위해 SQL_NULL_DATA로 설정된다. 그렇지 않으면 변수는 0이다.

ARD의 SQL_DESC_INDICATOR_PTR 필드가 NULL 포인터일 경우, 드라이버가 column의 NULL 여부를 반환하는 것을 방지한다. Column이 NULL이고 SQL_DESC_INDICATOR_PTR이 NULL 포인터일 경우, 드라이버가 SQLFetch나 SQLFetchScroll을 호출한 후에 버퍼를 생성하려고 하면 SQLSTATE 22002 (indicator variable required but not supplied)가 반환된다. SQLFetch나 SQLFetchScroll 호출이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우 버퍼의 내용은 정의되지 않는다.

SQL_DESC_INDICATOR_PTR 필드는 SQL_DESC_OCTET_LENGTH_PTR이 가리키는 필드를 설정할지 여부를 결정한다. Column의 데이터 값이 NULL일 경우, 드라이버는 지시자 변수를 SQL_NULL_DATA로 설정한다. 필드는 SQL_DESC_OCTET_LENGTH_PTR이 나타내는 필드를 이 시점에 설정하지 않는다. 데이터를 회수하는 동안 NULL 값을 받지 않을 경우, 버퍼는 SQL_DESC_INDICATOR_PTR이 가리키는 버퍼를 0으로 설정하고 SQL_DESC_OCTET_LENGTH_PTR이 가리키는 버퍼는 데이터 길이로 설정된다.

APD의 SQL_DESC_INDICATOR_PTR 필드가 NULL 포인터일 경우, 응용 프로그램은 인자들을 NULL로 명시하기 위해 이 설명자 레코드를 사용할 수 없다.

이 필드는 지연된 필드다. 이 필드는 설정될 때는 사용되지 않지만 (ARD일 경우) 드라이버가 NULL 가능성을 나타내거나 (APD일 경우) NULL 가능성을 결정할 때 사용된다.

SQL_DESC_LABEL[IRDs] (읽기 전용)

읽기전용 SQLCHAR*인 이 레코드 필드는 column의 라벨이나 표지를 포함한다. Column이 라벨을 가지지 않을 경우, 이 변수는 column 이름을 포함한다. Column 이름이 없거나 라벨을 사용할 수 없을 경우, 이 변수는 빈 문자열을 포함한다.

SQL_DESC_LENGTH[All]

SQLULEN인 이 레코드 필드는 문자열의 최대 길이나 실제 길이 또는 바이트 단위 이진 데이터이다. 이것은 고정된 길이 데이터 형식의 최대 길이어거나 가변 길이 데이터 형식의 실제 길이이다. 이 값은 항상 문자열의 끝에 있는 NULL 종료 문자를 제외한다. SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP 또는 SQL interval 데이터 형식인 값의 경우, 이 필드는 datetime 또는 interval 값이 문자열로 재표현 될 때의 문자 길이를 가진다.

이 필드의 값은 ODBC 2.x에 정의된 length의 값과 다를 수 있다.

SQL_DESC_LITERAL_PREFIX[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 문자 또는 드라이버가 접두사로 인식하는 문자들을 포함한다. 이 변수는 문자 접두사를 적용할 수 없는 데이터 형식에 대한 빈 문자열을 포함한다.

SQL_DESC_LITERAL_SUFFIX[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 문자 또는 드라이버가 접미사로 인식하는 문자들을 포함한다. 이 변수는 문자 접미사를 적용할 수 없는 데이터 형식에 대한 빈 문자열을 포함한다.

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 데이터 형식에 대해 지역화된 이름을 포함하는데 이는 데이터 형식의 정규 이름과 다를 수 있다. 지역화된 이름이 없을 경우 빈 문자열이 반환된다. 이 필드는 보여주기만을 목적으로 하는 필드이다.

SQL_DESC_NAME[Implementation descriptor]

SQLCHAR*인 이 레코드 필드는 row 설명자 내에서 필드의 별칭을 포함한다. Column의 별칭이 적용되지 않을 경우, column 이름이 반환된다. 두 경우 모두 SQL_DESC_NAME 필드를 설정할 때 드라이버가 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. Column 이름이나 column의 별칭이 없을 경우, 드라이버는 SQL_DESC_NAME 필드의 빈 문자열을 반환하고 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정한다.

응용 프로그램은 IPD의 SQL_DESC_NAME 필드를 매개 변수 이름 또는 별칭으로 설정하여 저장된 procedure 매개 변수를 이름별로 지정할 수 있다. IRD의 SQL_DESC_NAME 필드는 읽기 전용 필드이다. 응용 프로그램이 이 필드를 설정하려 할 경우, SQLSTATE HY091 (invalid descriptor field identifier)이 반환된다.

IPD의 경우, 드라이버가 기명의 매개 변수를 지원하지 않으면 이 필드는 정의되지 않는다. 드라이버가 기명의 매개 변수를 지원하고 이것이 매개 변수를 설명할 수 있다면 매개 변수 이름은 이 필드에 반환된다.

SQL_DESC_NULLABLE[Implementation descriptors] (읽기 전용)

IRD의 경우, 읽기 전용 SQLSMALLINT인 이 레코드 필드는 column이 NULL 값을 가질 수 있는 경우에 SQL_NULLABLE이고 NULL 값을 가질 수 없을 경우, SQL_NO_NULLS이며 column이 NULL 값을 허용하는 여부를 알 수 없을 경우, SQL_NULLABLE_UNKNOWN이다. 이 필드는 특히 결과 집합의 column에 대해 존재한

다.

IPD의 경우, 이 필드는 동적 매개 변수가 항상 NULL 가능하고 응용 프로그램에 의해 설정될 수 없기 때문에 항상 SQL_NULLABLE로 설정된다.

SQL_DESC_NUM_PREC_RADIX[All]

SQL_DESC_TYPE 필드가 approximate numeric 데이터 형식일 때 SQLINTEGER인 이 필드값은 2이다. 이는 SQL_DESC_PRECISION 필드가 비트 수를 포함하기 때문이다. SQL_DESC_TYPE 필드가 exact numeric 데이터 형식을 가지는 경우 SQL_DESC_PRECISION 필드가 소수점 자리의 수를 포함하므로 이 필드값은 10이다. 이 필드는 숫자가 아닌 데이터 형식에는 0을 설정한다.

SQL_DESC_OCTET_LENGTH[All]

SQLENN인 이 레코드 필드는 문자열이나 이진 데이터 형식의 바이트 단위 길이를 포함한다. 고정 길이의 문자나 이진 형식들의 경우 바이트 단위의 실제 길이이다. 가변 길이 문자나 이진 데이터 형식들의 경우 바이트 단위의 최대 길이이다. 이 값은 구현 설명자들에 대해 NULL 종료 문자를 위한 공백을 포함하지 않고 응용 프로그램 설명자들에 대해서는 NULL 종료 문자를 위한 공백을 포함한다. 응용 프로그램 데이터의 경우 이 필드에 버퍼 크기가 포함된다. APD들의 경우 이 필드는 출력 또는 입력/ 출력 매개 변수들에 대해서만 정의된다.

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

SQLENN*인 이 레코드 필드는 (매개 변수 설명자의 경우) 동적 인자 또는 (row 설명자의 경우) 바인딩된 column 값의 바이트 단위 전체 길이를 포함할 변수를 가리킨다.

APD의 경우 이 값은 문자열과 바이너리를 제외한 모든 인자들에 대해 무시된다. 이 필드가 SQL_NTS일 경우, 동적 인자는 NULL로 종료되어야 한다. Data-at-execution 매개 변수가 될 바인딩 된 매개 변수라는 것을 나타내기 위해 응용 프로그램은 실행할 때 SQL_DATA_AT_EXEC 또는 SQL_LEN_DATA_AT_EXEC 매크로의 결과를 포함할 변수를 APD의 레코드에 있는 이 필드에 설정한다. 이러한 필드가 하나 이상일 경우, SQL_DESC_DATA_PTR를 매개 변수를 식별하는 값으로 설정하여 응용 프로그램이 어떤 매개 변수가 요청되는지 판별하는데 도움을 줄 수 있다.

ARD의 OCTET_LENGTH_PTR 필드가 NULL 포인터일 경우, 드라이버는 column의 길이 정보를 반환하지 않는다. APD의 SQL_DESC_OCTET_LENGTH_PTR이 NULL 포인터일 경우, 드라이버는 문자열과 이진값들이 NULL로 종료되었다고 간주한다. (이진값들은 NULL로 종료되지 않아야 하지만 데이터 끊김을 피하려면 길이가 부여되어야 한다.)

이 필드가 가리키는 버퍼를 채울 SQLFetch나 SQLFetchScroll이 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않는다. 이 필드는 지연된 필드이다. 이 필드는 즉시 사용되지 않고 추후 드라이버가 데이터의 octet 길이를 표시하거나 결정할 때 사용된다.

SQL_DESC_PARAMETER_TYPE[IPDs]

SQLSMALLINT인 이 레코드 필드는 입력 매개 변수에 대해 SQL_PARAM_INPUT으로, 입력/ 출력 매개 변수에 대해 SQL_PARAM_INPUT_OUTPUT으로, 출력 매개 변수에 대해 SQL_PARAM_OUTPUT으로, 스트림된 입력/ 출력 매개 변수에 대해 SQL_PARAM_INPUT_OUTPUT_STREAM으로, 스트림된 출력 매개 변수에 대해 SQL_PARAM_OUTPUT_STREAM으로 설정된다. 이것은 기본적으로 SQL_PARAM_INPUT으로 설정된다.

SQL_DESC_PRECISION[All]

SQLSMALLINT인 이 레코드 필드는 exact numeric 형식에 대해 유효 정수 개수를 포함하고 approximate numeric에 대해 mantissa (이진 정밀도)의 비트 수를 포함한다. 또는 SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP

AMP, SQL_INTERVAL_SECOND 데이터 형식에 대해 fractional 초 부분의 자릿수를 포함한다. 이 필드는 다른 모든 데이터 형식에 대해서는 정의되지 않는다.

이 필드의 값은 ODBC 2.x에 정의된 precision 값과는 다를 수 있다.

SQL_DESC_ROWVER[Implementation descriptors] (읽기 전용)

SQLSMALLINT인 이 레코드 필드는 row가 갱신되었을 때 (예: SQL server의 timestamp) DBMS가 column을 자동으로 갱신하는지 여부를 나타낸다. 이 레코드 필드의 값은 column이 row 버저닝 column일 경우 SQL_TRUE로, 그렇지 않을 경우 SQL_FALSE로 설정된다. 이 column 속성은 column의 자동 갱신 여부를 결정하기 위해 SQL_ROWVER의 IdentifierType으로 SQLSpecialColumn를 호출하는 것과 유사하다.

SQL_DESC_SCALE[All]

SQLSMALLINT인 이 레코드 필드는 decimal과 numeric 데이터 형식에 대해 정의된 소수점 자릿수를 포함한다. 이 필드는 다른 모든 데이터 형식들에 대해 정의되지 않는다.

이 필드의 값은 ODBC 2.x 정의된 scale 값과 다를 수 있다.

SQL_DESC_SCHEMA_NAME[IRDs] (읽기 전용)

읽기전용 SQLCHAR*인 이 레코드 필드는 column을 포함하는 기본 테이블의 스키마 이름을 포함한다. 반환 값은 column이 표현식이거나 view의 일부일 경우 드라이버에 따라 다르다. 데이터 소스가 스키마를 지원하지 않거나 스키마 이름을 확인할 수 없을 경우, 이 변수는 빈 문자열을 포함한다.

SQL_DESC_SEARCHABLE[IRDs] (읽기 전용)

읽기 전용 SQLSMALLINT인 이 레코드 필드는 다음 값들 중 하나를 설정한다.

- Column이 WHERE 절에 사용될 수 없을 경우, SQL_PRED_NONE으로 설정된다. (이것은 ODBC 2.X의 SQL_UNSEARCHABLE과 같다.)
- Column이 WHERE 절에서 오직 LIKE 조건과 함께만 사용될 수 있는 경우, SQL_PRED_CHAR로 설정된다.
- Column이 WHERE 절에서 LIKE를 제외한 모든 비교 연산자들과 함께 사용될 수 있는 경우, SQL_PRED_BASIC으로 설정된다. (이것은 ODBC 2.x의 SQL_EXCEPT_LIKE 값과 같다.)
- Column이 WHERE 절의 모든 비교 연산자들과 함께 사용될 수 있는 경우, SQL_PRED_SEARCHABLE로 설정된다.

SQL_DESC_TABLE_NAME[IRDs] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 이 column을 포함하는 기본 테이블의 이름을 포함한다. Column이 표현식이거나 view의 일부일 경우 반환값은 드라이버에 따라 다르다.

SQL_DESC_TYPE[All]

SQLSMALLINT인 이 레코드 필드는 datetime과 interval 데이터 형식들을 제외한 모든 데이터 형식들에 대해 축약된 SQL 또는 C 데이터 형식을 포함한다. Datetime과 interval 데이터 형식들에 대해 이 필드는 SQL_DATETIME 또는 SQL_INTERVAL을 명시한다.

이 필드가 SQL_DATETIME 또는 SQL_INTERVAL을 포함할 때마다, SQL_DESC_DATETIME_INTERVAL_CODE 필드는 반드시 함축 형식의 적절한 서브 코드를 포함해야 한다. Datetime 데이터 형식들에 대해 SQL_DESC_TYPE는 SQL_DATETIME을 포함하고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 datetime 데이터 형식을 명시하는 서브 코드를 포함한다. Interval 데이터 형식들에 대해 SQL_DESC_TYPE는 SQL_INTERVAL을 포함하고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 interval 데이터 형식을 명시하는 서브 코드를 포함한다.

SQL_DESC_TYPE과 SQL_DESC_CONCISE_TYPE 필드의 값은 상호 의존적이다. 필드 중 하나가 설정될 때 마다 다른 필드도 설정되어야 한다. SQL_DESC_TYPE은 SQLSetDescField 또는 SQLSetDescRec를 호출하여 설정할 수 있다. SQL_DESC_CONCISE_TYPE은 SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 설정할 수 있다.

SQL_DESC_TYPE이 interval 또는 datetime 데이터 형식이 아닌 함축된 데이터 형식으로 설정될 경우 SQL_DESC_CONCISE_TYPE 필드는 같은 값으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드는 0으로 설정된다.

SQL_DESC_TYPE이 datetime이나 interval 데이터 형식의 긴 데이터 형식 (SQL_DATETIME 또는 SQL_INTERVAL)으로 설정되고 SQL_DESC_DATETIME_INTERVAL_CODE 필드가 적절한 서브 코드로 설정될 경우, SQL_DESC_CONCISE_TYPE 필드는 함축된 데이터 형식에 대응되는 값으로 설정된다. SQL_DESC_TYPE을 함축된 datetime 또는 interval 형식들 중 하나로 설정하려 하면 SQLSTATE HY021 (inconsistent descriptor information)가 반환된다.

SQLBindCol, SQLBindParameter 또는 SQLSetDescField를 호출하여 SQL_DESC_TYPE 필드를 설정할 경우 다음 필드들은 아래 표에 기술된 것과 같이 기본값들로 설정된다. 같은 레코드의 남은 필드값들은 정의되지 않는다.

SQL_DESC_TYPE 값	암묵적으로 설정된 다른 필드
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH는 1로 설정된다. SQL_DESC_PRECISION은 0으로 설정된다.
SQL_DATETIME	SQL_DESC_DATETIME_INTERVAL_CODE가 SQL_CODE_DATE 또는 SQL_CODE_TIME으로 설정될 때 SQL_DESC_PRECISION은 0으로 설정된다. SQL_DESC_TIMESTAMP으로 설정될 때는 SQL_DESC_PRECISION이 6으로 설정된다.
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE이 0으로 설정된다. SQL_DESC_PRECISION은 각 데이터 형식에 대해 구현 정의된 정밀도로 설정된다.
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION은 SQL_FLOAT에 대해 구현 정의된 기본 정밀도로 설정된다.
SQL_INTERVAL	SQL_DESC_DATETIME_INTERVAL_CODE가 interval 데이터 형식으로 설정될 때 SQL_DESC_DATETIME_INTERVAL_PRECISION은 2 (기본 interval leading precision)로 설정된다. Interval이 초 부분을 가지고 있을 때 SQL_DESC_PRECISION은 6 (기본 interval seconds precision)으로 설정된다.

응용 프로그램이 SQLSetDescRec가 아닌 SQLSetDescField를 호출하여 설명자의 필드를 설정하면 응용 프로그램은 반드시 데이터 형식을 제일 먼저 정의해야 한다. 이 경우, 위의 표에서 설명한 다른 필드들은 암묵적으로 설정된다. 어떤 값도 암묵적으로 설정하도록 허용되지 않을 경우, 응용 프로그램은 허용되지 않은 값을 명백하게 설정하기 위해 SQLSetDescField나 SQLSetDescRec를 호출할 수 있다.

SQL_DESC_TYPE_NAME[Implementation descriptors] (읽기 전용)

읽기 전용 SQLCHAR*인 이 레코드 필드는 데이터 소스에 의존적인 형식 이름 (CHAR, VARCHAR 등)을 포함한다. 데이터 형식 이름이 unknown일 경우, 변수는 빈 문자열을 포함한다.

SQL_DESC_UNNAMED[Implementation descriptors]

SQL_DESC_NAME 필드를 설정할 경우, row 설명자에 있는 SQLSMALLINT인 이 레코드 필드는 드라이버에 의해 SQL_NAME 또는 SQL_UNNAMED 중 하나로 설정된다. SQL_DESC_NAME 필드가 column의 별칭을 포함하거나 column의 별칭이 적용되지 않을 경우, 드라이버는 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. 응용 프로그램이 IPD의 SQL_DESC_NAME 필드를 매개 변수 이름이나 별칭으로 설정할 경우, 드라이버는 IPD의 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정한다. Column의 이름이나 별칭이 없을 경우, 드라이버는 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정한다.

응용 프로그램은 IPD의 SQL_DESC_UNNAMED 필드를 SQL_UNNAMED로 설정할 수 있다. 드라이버는 응용 프로그램이 IPD의 SQL_DESC_UNNAMED 필드를 SQL_NAMED로 설정하려고 시도하면 SQLSTATE HY091 (invalid descriptor field identifier)를 반환한다. 응용 프로그램이 읽기 전용인 IRD의 SQL_DESC_UNNAMED 필드를 설정하려고 시도하면 SQLSTATE HY091 (Invalid descriptor field identifier)가 반환된다.

SQL_DESC_UNSIGNED[Implementation descriptors] (읽기 전용)

읽기 전용 SQLSMALLINT인 이 레코드 필드는 column의 형식이 unsigned 또는 non-numeric일 경우 SQL_TRUE로, column의 형식이 signed일 경우, SQL_FALSE로 설정된다.

SQL_DESC_UPDATABLE[IRDs] (읽기 전용)

읽기 전용 SQLSMALLINT인 이 레코드 필드는 다음 값들 중 하나로 설정된다.

- 결과 집합 column이 읽기 전용일 경우, SQL_ATTR_READ_ONLY로 설정된다.
- 결과 집합 column이 읽고 쓰기일 경우, SQL_ATTR_WRITE로 설정된다.
- 결과 집합 column의 갱신 여부를 알 수 없을 경우, SQL_ATTR_READWRITE_UNKNOWN으로 설정된다.

SQL_DESC_UPDATABLE은 (기본 테이블에 있는 column이 아닌) 결과 집합 안에 있는 column의 갱신 여부를 설명한다. 이 결과 집합 column들이 기반으로 하고 있는 기본 테이블 안에 있는 column의 갱신 가능 여부는 이 필드 안에 있는 값과 다를 수 있다. Column의 갱신 여부는 데이터 형식, 사용자 권한 그리고 결과 집합 자체의 정의를 기반으로 한다. Column의 갱신 여부가 불확실할 경우, SQL_ATTR_READWRITE_UNKNOWN이 반환되어야 한다.

SQL_DESC_CHAR_LENGTH_UNITS[Implementation descriptors] (읽기 전용)

SQLSMALLINT인 이 레코드 필드는 SQL 형식이 SQL_CHAR, SQL_VARCHAR 또는 SQL_LONGVARCHAR인 column의 길이 단위를 나타낸다.

- SQL_CLU_CHARACTERS: 길이 단위가 CHARACTER이다. 예를 들어 인코딩 방식이 UHC (Unified Hangul Code)인 "문자열"이라는 데이터의 길이는 3이다.
- SQL_CLU_OCTETS: 길이 단위가 OCTETS이다. 예를 들어 인코딩 방식이 UHC인 "문자열"이라는 데이터의 길이는 6이다.
- SQL_CLU_NONE: 길이 단위가 정의되어 있지 않다. 위에 열거된 SQL 형식을 제외한 SQL 형식일 경우 반환되는 값이다.

일관성 검사

동시성 검사는 응용 프로그램이 ARD, APD 또는 IPD의 SQL_DESC_DATA_PTR 필드에 값을 넘길 때마다 드라이버에서 자동으로 수행된다. 일부 필드가 다른 필드들과 일치하지 않을 경우, SQLSetDescField는 SQLSTATE HY021 (inconsistent descriptor information)을 반환한다.

자세한 내용은 `SQLSetDescRec`의 일관성 검사를 참조한다.

SQLSetDescRec

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLSetDescRec 함수는 데이터 형식이나 column 또는 매개 변수 데이터에 바인딩된 버퍼에 영향을 주는 다중 설명자 필드들을 설정한다.

구문

```
SQLRETURN SQLSetDescRec(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT    RecNumber,
    SQLSMALLINT    Type,
    SQLSMALLINT    SubType,
    SQLLEN         Length,
    SQLSMALLINT    Precision,
    SQLSMALLINT    Scale,
    SQLPOINTER     DataPtr,
    SQLLEN *       StringLengthPtr,
    SQLLEN *       IndicatorPtr);
```

인자

DescriptorHandle

[입력] 설명자 핸들이다. IRD 핸들은 설정할 수 없다.

RecNumber

[입력] 설정될 필드들을 포함하는 설명자 레코드를 가리킨다. 설명자 레코드들은 0번부터 시작하며, 0번 레코드는 북마크 레코드가 된다. 이 인자는 반드시 0과 같거나 커야 한다. RecNumber가 SQL_DESC_COUNT보다 클 경우, SQL_DESC_COUNT가 RecNumber 값으로 바뀐다.

Type

[입력] 설명자 레코드의 SQL_DESC_TYPE 필드를 설정하는 값이다.

SubType

[입력] SQL_DATETIME이나 SQL_INTERVAL인 레코드 타입에 대해 SQL_DESC_DATETIME_INTERVAL_C
ODE 필드를 설정할 값이다.

Length

[입력] 설명자 레코드의 SQL_DESC_OCTET_LENGTH 필드를 설정하는 값이다.

Precision

[입력] 설명자 레코드의 SQL_DESC_PRECISION 필드를 설정하는 값이다.

Scale

[입력] 설명자 레코드의 SQL_DESC_SCALE 필드를 설정하는 값이다.

DataPtr

[지연된 입력 또는 출력] 설명자 레코드의 SQL_DESC_DATA_PTR 필드를 설정하는 값이다. DataPtr은 NUL
L 포인터로 설정될 수 있다.

DataPtr 인자는 SQL_DESC_DATA_PTR 필드를 NULL 포인터로 설정하기 위해 NULL 포인터가 될 수 있다.
DescriptorHandle 인자의 핸들이 ARD와 관련되어 있을 경우, 이것은 column 바인드를 해제한다.

StringLengthPtr

[지연된 입력 또는 출력] 설명자 레코드의 SQL_DESC_OCTET_LENGTH_PTR 필드를 설정하는 값이다. SQL
_DESC_OCTET_LENGTH_PTR 필드를 NULL 포인터로 설정하기 위해 StringLengthPtr은 NULL 포인터로
설정될 수 있다.

IndicatorPtr

[지연된 입력 또는 출력] 설명자 레코드의 SQL_DESC_INDICATOR_PTR 필드를 설정하는 값이다. SQL_DES
C_INDICATOR_PTR을 NULL 포인터로 설정하기 위해서 IndicatorPtr은 NULL 포인터로 설정될 수 있다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한 다.)
07009	Invalid descriptor index	RecNumber 인자가 0으로 설정되었고 DescriptorHandle은 IPD 핸들을 가리 킨다. RecNumber 인자가 데이터 소스가 지원하는 column이나 매개 변수의 최대 개 수보다 크고 DescriptorHandle 인자는 APD, IPD 또는 ARD이다. RecNumber 인자가 0이고 DescriptorHandle 인자는 내재적으로 할당된 APD 를 가리킨다. (이 오류는 명백하게 할당된 응용 프로그램 설명자에 대해서는 발

SQLSTATE	Error	설명
		생하지 않는다. 왜냐하면 명백하게 할당된 응용 프로그램 설명자가 APD 또는 ARD인지 여부를 실행시점이 되기 전에는 알 수 없기 때문이다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY010	Function sequence error	DescriptorHandle과 관련된 StatementHandle에 대해 비동기 수행 함수가 실행되었고, SQLSetDescRec가 실행되었을 때 비동기 수행 함수가 여전히 실행되고 있었다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY016	Cannot modify an implementation row descriptor	DescriptorHandle 인자는 IRD와 관련되어 있다.
HY021	Inconsistent descriptor information	Type 필드 또는 설명자의 SQL_DESC_TYPE 필드와 관련된 다른 필드가 유효하지 않거나 일치하지 않는다.
HY090	Invalid string or buffer length	드라이버가 ODBC 2.x 드라이버이고 설명자는 ARD, ColumnNumber 인자는 0으로 설정되었으며 BufferLength 인자에 명시된 값이 4가 아니다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	유예 상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
HYT01	Connection timeout expired	데이터 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 SQLSetConnectAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.

설명

응용 프로그램은 SQLSetDescRec를 호출하여 단일 column 또는 매개 변수에 대한 다음 필드들을 설정할 수 있다.

- SQL_DESC_TYPE
- (SQL_DATETIME 또는 SQL_INTERVAL 타입 레코드일 경우) SQL_DESC_DATETIME_INTERVAL_CODE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

노트

SQLSetDescRec 호출에 실패할 경우, RecNumber 인자가 식별한 설명자 레코드의 내용은 정의되지 않는다.

SQLSetDescRec는 column이나 매개 변수를 바인드 할 때 SQLBindCol나 SQLBindParameter을 호출하지 않고 SQLSetDescField도 여러 번 호출하지 않은 상태로 바인딩에 영향을 주는 여러 필드들을 변경할 수 있도록 한다. SQLSetDescRec는 현재 명령문과 관련없는 설명자 필드들을 설정할 수 있다. SQLBindParameter는 SQLSetDescRec보다 많은 필드들을 설정할 수 있고 한 번의 호출로 APD와 IPD의 필드들을 설정할 수 있으며 설명자 핸들을 요구하지 않는다.

노트

북마크 필드를 설정하려면 0인 RecNumber 인자로 SQLSetDescRec를 호출하기 전에 명령문 속성 SQL_ATTR_USE_BOOKMARKS를 설정해야 한다. 반드시 해야 하는 것은 아니지만 권장 사항이다.

일관성 검사

일관성 검사는 응용 프로그램이 APD, ARD 또는 IPD의 SQL_DESC_DATA_PTR 필드를 설정할 때마다 드라이버에 의해 자동으로 수행된다. 다른 필드들과 일치하지 않는 필드가 있을 경우, SQLSetDescRec는 SQLSTATE HY021 (inconsistent descriptor information)를 반환한다.

응용 프로그램이 APD, ARD 또는 IPD의 SQL_DESC_DATA_PTR 필드를 설정할 때마다 드라이버는 SQL_DESC_TYPE의 필드값을 확인하고 그 값이 SQL_DESC_TYPE 필드에 대해 유효하거나 일치하는지 여부를 확인한다. 이 검사는 SQLBindParameter 또는 SQLBindCol이 호출되거나 SQLSetDescRec가 APD, ARD 또는 IPD에 대해 호출될 때마다 수행된다. 이 일관성 검사는 다음 설명자 필드들을 포함한다.

- SQL_DESC_TYPE 필드는 유효한 ODBC C, SQL type 또는 드라이버별 SQL type 중 하나이어야 한다. SQL_DESC_CONCISE_TYPE 필드는 유효한 ODBC C, SQL Type 또는 드라이버별 SQL type 중 하나이어야 하고 간결한 datetime과 interval 형식들을 포함한다.
- SQL_DESC_TYPE 레코드 필드가 SQL_DATETIME 또는 SQL_INTERVAL일 경우, SQL_DESC_DATETIME_INTERVAL_CODE 필드는 반드시 유효한 datetime 또는 interval 코드들 중 하나이어야 한다. (SQLSetDescField의 SQL_DESC_DATETIME_INTERVAL_CODE 필드의 설명을 참조한다.)
- SQL_DESC_TYPE 필드가 NUMERIC 형식을 가리킬 경우, SQL_DESC_PRECISION과 SQL_DESC_SCALE 필드가 유효한 값이 될 수 있는지 여부를 확인한다.
- SQL_DESC_CONCISE_TYPE 필드가 time 또는 timestamp 데이터 형식이고 second 요소 또는 time 요소를 취하는 interval 데이터 형식일 경우, SQL_DESC_PRECISION 필드가 유효한 second precision가 될 수 있는지 여부를 확인한다.
- SQL_DESC_CONCISE_TYPE이 interval 데이터 형식일 경우, SQL_DESC_DATETIME_INTERVAL_PRECISION 필드가 유효한 interval 선행 precision 값이 될 수 있는지 여부를 확인한다.

일반적으로 IPD의 SQL_DESC_DATA_PTR 필드는 설정하지 않는다. 응용 프로그램은 IPD 필드들에 대해 강제로 일관성 검사를 할 수 있다. IRD에서는 일관성 검사를 수행할 수 없다. IPD의 SQL_DESC_DATA_PTR 필드값은 실제로 저장되지 않고 SQLGetDescField 또는 SQLGetDescRec를 통해 검색할 수 없다. 일관성 검사를 강제로 수행하도록 설정된다.

SQLSetEnvAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLSetEnvAttr은 환경 관리 속성을 설정한다.

구문

```
SQLRETURN SQLSetEnvAttr(
    SQLHENV      EnvironmentHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength);
```

인자

EnvironmentHandle

[입력] 환경 핸들이다.

Attribute

[입력] **설명** 부분에 나열되어 있다.

ValuePtr

[입력] Attribute에 연동될 값에 대한 포인터이다. Attribute 값에 따라 ValuePtr은 32 비트 정수값이 될 수도 있고 NULL 종료 문자를 가리킬 수도 있다.

StringLength

[입력] ValuePtr이 문자열이나 이진 버퍼를 가리킬 경우, 이 인자는 *ValuePtr의 길이어야 한다. 문자열 데이터의 경우, 이 인자는 문자열의 바이트 수를 포함해야 한다.

ValuePtr이 정수일 경우, StringLength는 무시된다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	드라이버가 ValuePtr에 명시된 값을 지원하지 않고 유사한 값으로 대체한다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY009	Invalid use of null pointer	속성 인자가 문자열 값이 요구되는 환경 속성을 식별하고 ValuePtr 인자가 NULL 포인터이다.
HY010	Function sequence error	연결 핸들이 EnvironmentHandle에 할당되었다. SQL_ATTR_ODBC_VERSION이 SQLSetEnvAttr에 설정되지 않고 attribute가 SQL_ATTR_ODBC_VERSION과 다르다. SQLAllocHandleStd를 사용하면 명시적으로 SQL_ATTR_ODBC_VERSION을 기술할 필요가 없다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY024	Invalid attribute value	명시된 attribute 값을 감안할 때 ValuePtr의 값이 유효하지 않다.
HY090	Invalid string or buffer length	StringLength 인자가 0보다 작다. (SQL_NTS는 아니다.)
HY092	Invalid attribute/option identifier	입력된 attribute가 드라이버에서 지원하는 ODBC의 버전에서 유효하지 않다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	유예상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
HYC00	Optional feature not implemented	명시된 attribute 인자값이 드라이버에서 지원하는 ODBC 버전의 ODBC 환경 속성에서 유효하지만 드라이버에서 지원하지 않는다. Attribute 인자가 SQL_ATTR_OUTPUT_NTS이고 ValuePtr은 SQL_FALSE이다.

설명

응용 프로그램은 환경 핸들에 할당된 연결 핸들이 없을 때만 SQLSetEnvAttr을 호출할 수 있다. 응용 프로그램이 설정한 환경 속성들은 환경 핸들에 SQLFreeHandle이 호출될 때까지 유지된다. 환경 핸들은 하나만 할당하여 사용할 것을 권장한다.

ValuePtr을 통해 설정된 정보의 포맷은 명시된 attribute에 따라 다르다. SQLSetEnvAttr은 속성 정보를 서로 다른 두 가지 포맷 (NULL 종료 문자열 또는 32 비트 정수 값) 중 하나의 포맷으로 받아들인다. 각각의 포맷은 속성의 설명

부분에 기술되어 있다.

드라이버별 환경 속성은 존재하지 않는다.

연결 속성은 SQLSetEnvAttr을 호출하여 설정할 수 없다. 설정을 시도할 경우 SQLSTATE HY092 (invalid attribute /option identifier)가 반환된다.

Attribute	ValuPtr 내용
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_CP_MATCH (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ODBC_VERSION (ODBC 3.0)	특정 기능이 ODBC 2.x로 작동하는지 아니면 ODBC 3.x로 작동하는지를 가리키는 32 비트 정수이다. 다음 값들이 속성을 설정하는데 사용된다. - SQL_OV_ODBC3_80: 드라이버 관리자와 드라이버가 다음과 같은 ODBC 3.8 행동을 보인다. - 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 3.x 코드값을 예상하고 반환한다. - 드라이버는 SQLError, SQLGetDiagField나 SQLGetDiagRec가 호출될 때 ODBC 3.x SQLSTATE 코드들을 반환한다. - SQLTables의 CatalogName 인자가 패턴 검색을 허용한다. - SQL_OV_ODBC3: 드라이버 관리자와 드라이버가 다음과 같은 ODBC 3.x 행동을 보인다. - 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 3.x 코드값을 예상하고 반환한다. - 드라이버는 SQLError, SQLGetDiagField 또는 SQLGetDiagRec가 호출될 때 ODBC 3.x SQLSTATE 코드들을 반환한다. - SQLTables의 CatalogName 인자가 패턴 검색을 허용한다. - 드라이버 관리자가 C 데이터 형식 확장성을 지원하지 않는다. - SQL_OV_ODBC2: 드라이버 관리자와 드라이버는 ODBC 2.x 행동을 보인다. 이것은 ODBC 2.x 응용 프로그램이 ODBC 3.x 드라이버에서 작동할 때 매우 유용하다. - 드라이버는 DATE, TIME, TIMESTAMP에 대해 ODBC 2.x 코드값을 예상하고 반환한다. - 드라이버는 SQLError, SQLGetDiagField 또는 SQLGetDiagRec가 호출될 때 ODBC 2.x SQLSTATE 코드들을 반환한다. - SQLTables의 CatalogName 인자가 패턴 검색을 허용하지 않는다. - 드라이버 관리자가 C 데이터 형식 확장성을 지원하지 않는다. 응용 프로그램은 SQLHENV 인자를 가진 함수를 호출하기 전에 이 환경 속성 값을 설정해야 한다. 그렇지 않을 경우 SQLSTATE HY010 (function sequence error)가 반환된다. 이런 환경 플래그에 대한 추가 행동이 있는지 여부는 드라이버별로 다르다.

Attribute	ValuPtr 내용
SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	드라이버에서 지원하지 않는다.

SQLSetParam

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 1.0의 함수 `SQLSetParam`은 ODBC 2.0에서 `SQLBindParameter`로 대체되었다.

SQLSetPos

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLSetPos는 row 집합에서 커서의 위치를 설정하고 응용 프로그램이 row 집합의 데이터를 갱신하거나 결과 집합 안의 데이터를 갱신 또는 삭제하도록 허용한다.

구문

```
SQLRETURN SQLSetPos(
    SQLHSTMT      StatementHandle,
    SQLSETPOSIROW RowNumber,
    SQLUSMALLINT  Operation,
    SQLUSMALLINT  LockType);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

RowNumber

[입력] Operation 인자로 명시된 연산을 수행할 row 집합에서의 row 위치이다. RowNumber가 0일 경우, row 집합 안의 모든 row에 연산이 적용된다.

자세한 내용은 **설명**을 참조한다.

Operation

[입력] 수행할 연산은 SQL_POSITION, SQL_REFRESH, SQL_UPDATE, SQL_DELETE이다.

Operation 인자의 SQL_ADD 값은 ODBC 3.x에서 더 이상 사용되지 않는다. ODBC 2.x 드라이버는 하위 버전과의 호환성을 위해 SQL_ADD를 지원해야 한다. 이 기능은 SQL_ADD 연산을 사용하여 SQLBulkOperations를 호출하는 것으로 바뀌었다. ODBC 2.x 드라이버로 ODBC 3.x 응용 프로그램을 수행할 때, 드라이버 관리자는 SQL_ADD 연산을 사용한 SQLBulkOperations 호출을 SQL_ADD 연산을 사용한 SQLSetPos 호출에 매핑한다.

자세한 내용은 **설명**을 참조한다.

LockType

[입력] Operation 인자에 명시된 연산을 이행한 후에 row를 잠그는 방법을 명시한다.

SQL_LOCK_NO_CHANGE, SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK

자세한 내용은 **설명**을 참조한다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01001	Cursor operation conflict	Operation 인자가 SQL_DELETE 또는 SQL_UPDATE이다. 하나 이상의 row가 삭제 또는 갱신되거나 어떤 row도 삭제되거나 갱신되지 않았다. Operation 인자가 SQL_DELETE 또는 SQL_UPDATE이고 최적 병행때문에 연산이 실패하였다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01004	String data, right truncation	Operation 인자가 SQL_REFRESH이고, 데이터 형식이 SQL_C_CHAR 또는 SQL_C_BINARY 형식인 column에 대해 반환되는 문자열이나 이진 데이터의 공백이 아닌 문자 또는 null이 아닌 이진 데이터가 잘렸다.
01S01	Error in row	RowNumber 인자가 0이고, operation 인자에 명시된 연산을 수행하던 중에 하나 이상의 row에서 오류가 발생하였다. (다중 연산에서 전체가 아닌 하나 이상의 row들에 오류가 발생하면 SQL_SUCCESS_WITH_INFO가 반환되지만 단일 row 연산에서 오류가 발생할 경우 SQL_ERROR가 반환된다.) (드라이버가 ODBC 2.x이고 커서 라이브러리가 사용되지 않으면 SQLExtendedFetch 이후에 SQLSetPos를 호출할 때만 이 SQLSTATE가 반환된다.)
01S07	Fractional truncation	Operation 인자가 SQL_REFRESH이고, 응용 프로그램 버퍼의 데이터 형식이 SQL_C_CHAR나 SQL_C_BINARY가 아니며 하나 이상의 column들에 대해 응용 프로그램 버퍼에 반환된 데이터가 잘렸다. Numeric 데이터 형식들의 경우 숫자의 소수점 부분이 잘렸다. Time, timestamp 그리고 time 부분을 포함하는 interval 데이터 형식들의 경우 time의 fractional 부분이 잘렸다.
07006	Restricted data type attribute violation	결과 집합의 column의 데이터 값이 SQLBindCol을 호출할 때 TargetType에 명시한 데이터 형식으로 변환될 수 없다.
07009	Invalid descriptor index	Operation 인자가 SQL_REFRESH 또는 SQL_UPDATE이고 결과 집합의 column의 개수보다 많은 column의 값이 바인딩 되었다.
	Degree of derived table do	Operation 인자가 SQL_UPDATE이고 모든 column의 바인딩이 해제되거나

SQLSTATE	Error	설명
21S02	es not match column list	읽기 전용이거나 바인딩 된 길이/ 지시자 버퍼 값이 SQL_COLUMN_IGNORE 이기 때문에 갱신 가능한 column이 없었다.
22001	String data, right truncation	Operation 인자는 SQL_UPDATE이고, 문자 또는 바이너리 값을 column에 할당하는 바람에 (문자에 대해) 공백이 아닌 문자, (바이너리에 대해) null이 아닌 문자 또는 바이트가 잘렸다.
22003	Numeric value out of range	Operation 인자가 SQL_UPDATE이고, 결과 집합의 column에 numeric 값을 할당하는 바람에 모든 숫자가 잘렸다. Operation 인자가 SQL_REFRESH이고, 하나 이상의 바인딩 column에 대한 numeric 값을 반환하는 바람에 유효숫자가 유실되었다.
22007	Invalid datetime format	Operation 인자가 SQL_UPDATE이고 결과 집합의 column에 date 또는 time stamp 값을 할당하는 바람에 연, 월 또는 일 필드가 범위를 벗어나게 되었다. Operation 인자가 SQL_REFRESH이고 하나 이상의 바인딩된 column에 대해 date 또는 timestamp 값을 반환하는 바람에 연, 월 또는 일 필드가 범위를 벗어나게 되었다.
22008	Date/time field overflow	Operation 인자가 SQL_UPDATE이고, 결과 집합의 column으로 보내지는 데이터에 대해 datetime 연산을 실행하는 바람에 결과의 datetime 필드 (연, 월, 일, 시, 분, 초 필드)를 허용 범위에서 벗어나게 하거나 그레고리력의 규칙에 따라 무효가 되게 한다. Operation 인자가 SQL_REFRESH이고 결과 집합에서 검색되는 데이터에 대해 datetime 연산을 실행하는 바람에 결과의 datetime 필드 (연, 월, 일, 시, 분, 초 필드)가 허용 범위에서 벗어나거나 그레고리력의 규칙에 따라 무효가 되게 한다.
22015	Interval field overflow	Operation 인자가 SQL_UPDATE이고, exact numeric 또는 interval C 값을 interval SQL 데이터 형식으로 할당하는 바람에 유효 숫자가 유실되었다. Operation 인자가 SQL_UPDATE이고, interval SQL 형식에 할당하는 바람에 the interval SQL type에 C type으로 표현되는 값이 없었다. Operation 인자가 SQL_REFRESH이고, exact numeric이나 interval SQL type을 interval C 형식에 할당하는 바람에 선행 필드의 유효숫자가 유실되었다. Operation 인자가 SQL_REFRESH이고, interval C type에 할당하는 바람에 interval C type에 SQL type으로 표현되는 값이 없었다.
22018	Invalid character value for cast specification	Operation 인자가 SQL_REFRESH이고, C 형식은 exact 또는 approximate numeric, datetime 또는 interval 데이터 형식이다. Column의 SQL 형식이 문자 데이터 형식이다. 그리고 column의 값이 바인딩된 C 형식에 대해 유효한 문자가 아니다. Operation 인자가 SQL_UPDATE이고, SQL 형식은 exact 또는 approximate numeric, datetime 또는 interval 데이터 형식이다. C column의 형식은 SQL_C_CHAR이고, column의 값이 바인딩된 SQL 형식에 대해 유효한 문자가 아니

SQLSTATE	Error	설명
		다.
23000	Integrity constraint violation	Operation 인자가 SQL_DELETE 또는 SQL_UPDATE이고 integrity 제약사항을 위반했다.
24000	Invalid cursor state	StatementHandle이 실행된 상태이지만 결과는 StatementHandle과 관련이 없다. 커서가 StatementHandle에 열려있지만 SQLFetch 또는 SQLFetchScroll이 호출되지 않았다. StatementHandle에 커서가 열려 있고 SQLFetch 또는 SQLFetchScroll이 호출되었다. 하지만 커서가 결과 집합의 시작 지점 이전이나 끝 지점 이후에 위치해 있다. Operation 인자가 SQL_DELETE, SQL_REFRESH 또는 SQL_UPDATE이다.
40001	Serialization failure	다른 트랜잭션과의 리소스 데드락이 발생하여 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	함수 실행 중 연결에 실패하였고 트랜잭션의 상태를 확인할 수 없다.
42000	Syntax error or access violation	드라이버가 operation 인자에서 요청된 연산을 수행하는데 필요한만큼 row를 잠글 수 없었다. 드라이버는 LockType 인자에서 요청한대로 row를 잠글 수 없었다.
44000	WITH CHECK OPTION violation	Operation 인자가 SQL_UPDATE이고, 업데이트에 영향받는 하나 이상의 row들이 view 테이블에 존재하지 않도록 하기 위해 WITH_CHECK OPTION으로 생성한 view 테이블이나 view 테이블에서 파생된 테이블이 갱신되었다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리 할당할 수 없다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하다. 이 함수가 호출되고 실행이 완료되기 전에 SQLCancel이나 SQLCancelHandle이 StatementHandle에 호출되었고 이 함수가 StatementHandle에 다시 호출되었다. 함수가 실행되고 실행이 완료되기 전에 SQLCancel이나 SQLCancelHandle이 multi thread 응용 프로그램의 다른 thread에서 StatementHandle에 호출되었다.
HY010	Function sequence error	SQLSetPos가 실행되었을 때 StatementHandle에 비동기 수행 함수가 호출되었으며, SQLSetPos 함수가 호출되었을 때 해당 비동기 수행 함수가 여전히 실행되고 있었다. StatementHandle에 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 호출되었고, SQL_PARAM_DATA_AVAILABLE이 반환되었으며, 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 해당 함수가 호출되었다. 비동기 수행 함수가 StatementHandle에 대해 호출되었으며 이 함수가 호출되

SQLSTATE	Error	설명
		었을 때 해당 비동기 수행 함수가 여전히 실행되고 있었다. StatementHandle에 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 실행되어 SQL_NEED_DATA가 반환되었고, 모든 데이터가 전송되기 전에 함수가 실행되었다. 드라이버가 ODBC 2.x 이고 SQLFetch가 호출된 후에 StatementHandle에 대해 SQLSetPos가 호출되었다.
HY011	Attribute cannot be set now	드라이버 버전이 ODBC 2.x이고 SQL_ATTR_ROW_STATUS_PTR 명령문 속성이 설정된 상태에서 SQLFetch, SQLFetchScroll 또는 SQLExtendedFetch가 호출되기 전에 SQLSetPos가 호출되었다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY090	Invalid string or buffer length	Operation 인자가 SQL_UPDATE이다.
HY092	Invalid attribute identifier	Operation 인자에 명시된 값이 유효하지 않다. LockType 인자에 명시된 값이 유효하지 않다. Operation 인자가 SQL_UPDATE 또는 SQL_DELETE 이고 SQL_ATTR_CONCURRENCY 명령문 속성은 SQL_ATTR_CONCUR_READ_ONLY 이다.
HY107	Row value out of range	RowNumber 인자에 명시된 값이 row 집합 안의 row의 개수보다 크다.
HY109	Invalid cursor position	StatementHandle과 관련된 커서가 forward-only로 설정되어 row를 row 집합 안에 위치시킬 수 없었다. SQLSetStmtAttr의 SQL_ATTR_CURSOR_TYPE 속성에 대한 설명을 참조한다. Operation 인자가 SQL_UPDATE, SQL_DELETE 또는 SQL_REFRESH이고 RowNumber 인자로 식별된 row가 삭제되거나 회수되었다. RowNumber 인자가 0이고 operation 인자는 SQL_POSITION 이다. SQLSetPos는 SQLBulkOperations가 호출된 후 SQLFetchScroll 또는 SQLFetch가 호출되기 전에 호출되었다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 함수를 참조한다.
HYC00	Optional feature not implemented	드라이버 또는 데이터 소스가 operation 인자 또는 LockType 인자에서 요청한 연산을 지원하지 않는다.
HYT00	Timeout expired	데이터 소스가 결과 집합을 반환하기 전에 쿼리 타임 아웃 기간이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_QUERY_TIMEOUT을 통해 설정할 수 있다.

SQLSTATE	Error	설명
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결이 만료되었다. 이 제한 시간은 SQLSetStmtAttr의 SQL_ATTR_CONNECTION_TIMEOUT을 통해 설정할 수 있다.
IM001	Driver does not support this function	StatementHandle과 관련된 드라이버가 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

RowNumber 인자

RowNumber 인자는 operation 인자에 명시된 연산 이행에 영향을 받는 row 집합 안의 row 개수를 명시한다. RowNumber가 0일 경우, row 집합 안의 모든 row에 적용된다. RowNumber는 반드시 0부터 row 집합 안의 row 개수까지여야 한다.

C 언어의 배열은 0부터 시작하지만 RowNumber 인자는 1부터 시작한다. 예를 들어, row 집합의 다섯 번째 row를 갱신할 때 응용 프로그램은 배열 인덱스 4의 row 집합 버퍼를 수정하지만 RowNumber는 5로 명시한다.

모든 연산은 RowNumber가 명시하는 row에 커서를 위치 시킨다. 다음 연산은 커서 위치를 요구한다.

- Positioned 갱신 또는 삭제 명령문들
- SQLGetData 호출
- SQL_DELETE, SQL_REFRESH, SQL_UPDATE 옵션으로 SQLSetPos 호출

Operation 인자

Operation 인자는 다음 연산들을 지원한다. 데이터 소스에서 지원되는 옵션들을 확인하기 위해 응용 프로그램은 커서 형식에 따라 SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1 또는 SQL_STATIC_CURSOR_ATTRIBUTES1 정보 형식으로 SQLGetInfo를 호출한다.

Operation 인자	연산
SQL_POSITION	드라이버는 RowNumber에 명시된 row에 커서를 위치시킨다. SQL_ATTR_ROW_OPERATION_PTR 명령문 속성이 가리키는 row 상태 배열의 내용은 SQL_POSITION에 의해 무시된다.
SQL_REFRESH	드라이버에서 지원하지 않는다.
SQL_UPDATE	드라이버에서 지원하지 않는다.
SQL_DELETE	드라이버에서 지원하지 않는다.

LockType 인자

LockType 인자는 응용 프로그램이 동시성을 제어할 수 있는 방법을 제공한다. 대부분의 경우 동시성 레벨과 트랜잭션을 제공하는 데이터 소스는 LockType 인자의 SQL_BLOCK_NO_CHANGE 값만 지원한다.

LockType 인자는 SQLSetPos가 실행된 후에 row의 잠금 상태를 명시한다. 드라이버가 요청된 연산을 수행하거나 LockType 인자를 만족시키기 위해 row를 잠그지 못할 경우, SQL_ERROR나 SQLSTATE 42000 (syntax error or access violation)을 반환한다.

LockType 인자가 단일 명령문에 대해서만 명시되더라도 같은 권한으로 연결된 모든 명령문에 적용된다. 연결된 단일 명령문이 획득한 잠금은 같은 연결선상의 다른 명령문에 의해 해제될 수 있다.

SQLSetPos에 의해 잠긴 row는 응용 프로그램이 LockType이 SQL_LOCK_UNLOCK으로 설정된 row에 대해 SQLSetPos를 호출할 때까지 또는 응용 프로그램이 SQL_CLOSE 옵션으로 SQLFreeStmt를 호출하거나 명령문에 대해 SQLFreeHandle을 호출할 때까지 유지된다. 트랜잭션을 지원하는 드라이버의 경우, 응용 프로그램에서 SQLSetPos로 잠긴 row는 (SQLGetInfo에서 반환되는 SQL_CURSOR_COMMIT_BEHAVIOR나 SQL_CURSOR_ROLLBACK_BEHAVIOR 정보 형식이 가리키는 것처럼 트랜잭션이 commit 되거나 rollback 될 때 커서가 닫힐 경우) 응용 프로그램이 SQLEndTran을 호출하여 연결되어 있는 트랜잭션을 commit 하거나 rollback 할 때 해제된다.

LockType 인자는 다음과 같은 잠금 형식들을 지원한다. 응용 프로그램은 SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1 또는 SQL_STATIC_CURSOR_ATTRIBUTES1 정보 형식으로 SQLGetInfo를 호출하여 데이터 소스에서 지원되는 잠금을 확인한다.

LockType 인자	잠금 형식
SQL_LOCK_NO_CHANGE	드라이버나 데이터 소스는 SQLSetPos가 호출되기 전과 같은 row의 잠금 또는 잠금해제를 보장한다. 이 LockType 값은 현재 동시성과 트랜잭션 격리 수준에서 요청되는 어떤 수준의 잠금이든지 사용할 수 있도록 명시적인 row 수준의 잠금을 허용하지 않는다.
SQL_LOCK_EXCLUSIVE	드라이버나 데이터 소스는 배타적으로 row를 잠근다. 다른 연결이나 응용 프로그램의 명령문은 해당 row에 대한 잠금을 획득하기 위해 사용될 수 없다.
SQL_LOCK_UNLOCK	드라이버나 데이터 소스가 잠금을 해제한다.

SQLSetPos에서 갱신 및 삭제 연산을 할 때 응용 프로그램은 다음과 같은 LockType 인자를 사용한다.

- Row가 회수된 후에 갱신되지 않도록 하기 위해 응용 프로그램은 operation 집합을 사용하여 SQLSetPos를 SQL_REFRESH로 호출하고, LockType 집합을 SQL_LOCK_EXCLUSIVE로 호출한다.
- 응용 프로그램이 LockType을 SQL_LOCK_NO_CHANGE로 설정하면 드라이버는 응용 프로그램이 SQL_ATTR_CONCURRENCY 명령문 속성에 대해 SQL_CONCUR_LOCK을 명시했을 때만 갱신 또는 삭제 연산에 성공하도록 보장한다.
- 응용 프로그램이 SQL_ATTR_CONCURRENCY 명령문 속성에 대해 SQL_CONCUR_ROWVER 또는 SQL_CONCUR_VALUES를 명시하면 드라이버는 응용 프로그램이 row를 회수한 후 이 row가 갱신될 경우 row의 버전 또는 값을 비교하여 연산을 거부한다.
- 응용 프로그램이 SQL_ATTR_CONCURRENCY 명령문 속성에 대해 SQL_CONCUR_READ_ONLY를 명시하면 드라이버는 모든 갱신과 삭제 연산을 거부한다.

SQL_ATTR_CONCURRENCY 명령문 속성에 대한 자세한 내용은 **SQLSetStmtAttr**을 참조한다.

상태 및 연산 배열

다음 상태 및 연산 배열은 SQLSetPos가 호출될 때 사용된다.

- Row 상태 배열은 row 집합 안에 있는 각각의 row 데이터의 상태값을 포함한다. 드라이버는 SQLFetch, SQLFetchScroll, SQLBulkOperations 또는 SQLSetPos를 호출한 후에 이 배열 안에 있는 상태값을 설정한다. 이 배열은 SQL_ATTR_ROW_STATUS_PTR 명령문 속성에 의해 지정된다.
- Row 연산 배열은 벌크 연산에 대한 SQLSetPos 호출이 무시되었는지 또는 이행되었는지 여부를 나타내는 row 집합 안에 있는 각 row 값을 포함한다. 배열 속의 각 요소는 SQL_ROW_PROCEED 또는 SQL_ROW_IGNORE 중 하나를 설정한다. 이 배열은 SQL_ATTR_ROW_OPERATION_PTR 명령문 속성에 의해 지정된다.

상태 및 연산 배열에 있는 요소의 개수는 반드시 (SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성에 정의된 것과 같이) row 집합 안의 row 개수와 같아야 한다.

Row 상태 배열에 대한 자세한 내용은 **SQLFetch**를 참조하고 row 연산 배열에 대한 자세한 내용은 **벌크 연산에서의 row 무시**를 참조한다.

SQLSetPos 사용

응용 프로그램은 SQLSetPos를 사용하기 전에 다음과 같은 절차를 수행해야 한다.

1. 응용 프로그램이 operation 집합을 사용하여 SQLSetPos를 SQL_UPDATE로 호출할 경우, column의 데이터 형식을 명시하고 column 데이터와 길이에 대한 버퍼를 바인딩하기 위해 각 열에 대해 SQLBindCol (또는 SQLSetDescRec)를 호출한다.
2. 응용 프로그램이 operation 집합을 사용하여 SQLSetPos를 SQL_DELETE 또는 SQL_UPDATE로 호출할 경우, SQLColAttribute를 호출하여 삭제하거나 갱신할 column이 갱신될 수 있도록 한다.
3. SQLExecDirect, SQLExecute 또는 카탈로그 함수를 호출하여 결과 집합을 만든다.
4. SQLFetch 또는 SQLFetchScroll을 호출하여 데이터를 회수한다.

SQLSetPos를 사용한 데이터 삭제

SQLSetPos를 사용하여 데이터를 삭제하기 위해 응용 프로그램은 RowNumber 집합을 사용하여 삭제할 row 번호로 SQLSetPos를 호출하고 operation 집합을 SQL_DELETE로 호출한다.

데이터가 삭제된 후, 드라이버는 해당 row에 대한 구현 row 상태 배열 값을 SQL_ROW_DELETED (또는 SQL_ROW_ERROR)로 변경한다.

SQLSetPos를 사용한 데이터 갱신

응용 프로그램은 바인딩 된 버퍼 또는 한 번 이상의 SQLPutData 호출을 통해 데이터를 column에 전달할 수 있다. SQLPutData를 통해 데이터를 전달 받은 column이 바로 data-at-execution column이다. 보통은 SQL_LONGVAR_BINARY와 SQL_LONGVARCHAR column에 데이터를 전송하기 위해 사용되고 다른 column들과 함께 섞일 수도 있다.

응용 프로그램에서 SQLSetPos를 사용한 데이터 갱신

- 데이터와 SQLBindCol로 바인딩 된 길이/ 지시자 버퍼에 값을 배치한다.
 - 일반 column들의 경우 응용 프로그램은 새로운 column의 값을 *TargetValuePtr 버퍼에 배치하고 값의 길이는 *StrLen_or_IndPtr 버퍼에 배치한다. Row가 갱신되면 안될 경우, 응용 프로그램은 row 연산 배열에서 해당 row의 요소에 SQL_ROW_IGNORE를 배치한다.
 - Data-at-execution column들의 경우, 응용 프로그램은 응용 프로그램에서 정의한 column 번호 같은 값을 *TargetValuePtr에 배치한다. 이 값은 나중에 column을 식별하는데 사용될 수 있다.
응용 프로그램은 *StrLen_or_IndPtr 버퍼에 SQL_LEN_DATA_AT_EXEC (length) 매크로의 결과를 배치한다. Column의 SQL 데이터 형식이 SQL_LONGVARIABLE, SQL_LONGVAR_CHAR 또는 소스 정의 긴 데이터 형식이고 드라이버가 SQLGetInfo의 SQL_NEED_LONG_DATA 정보 형식에 Y를 반환할 경우, length는 매개 변수로 전송할 데이터의 바이트 수이다. 그렇지 않을 경우, 반드시 음수가 아니어야 하며 무시되어야 한다.
- Row 데이터를 갱신하기 위해 operation 인자 집합으로 SQLSetPos를 SQL_UPDATE로 호출한다.
 - Data-at-execution column들이 없으면 처리는 완료된다.
 - Data-at-execution column들이 있으면 이 함수가 SQL_NEED_DATA를 반환하고 3단계로 넘어간다.
- 처리할 첫 data-at-execution column의 *TargetValuePtr 버퍼의 주소를 검색하기 위해 SQLParamData를 호출한다. SQLParamData는 SQL_NEED_DATA를 반환한다. 응용 프로그램은 *TargetValuePtr 버퍼에서 응용 프로그램에서 정의한 값을 검색하도록 한다.

노트

- Data-at-execution 매개 변수는 data-at-execution column과 비슷하지만 SQLParamData가 반환하는 값이 각각 다르다.
- SQL 명령문에서의 data-at-execution 매개 변수는 명령문이 SQLExecDirect 또는 SQLExecute으로 실행되었을 때 SQLPutData로 전송된다. 그것들은 SQLBindParameter로 바인딩되거나 SQLSetDescRec로 설명자들을 설정하여 바인딩된다. SQLParamData가 반환하는 값은 32 비트 값으로 ParameterValuePtr 인자의 SQLBindParameter로 전송된다.
- Data-at-execution column들은 SQLSetPos로 row를 갱신할 때 SQLPutData로 데이터를 전송할 row 집합의 column이다. 그것들은 SQLBindCol로 바인딩된다. SQLParamData가 반환하는 값은 처리 중인 TargetValuePtr 버퍼에 있는 row의 주소이다.

4. Column에 대한 데이터를 전송하기 위해 SQLPutData를 한 번 이상 호출한다. SQLPutData에 명시된 *Target ValuePtr 버퍼에서 모든 데이터 값들을 반환할 수 없으면 여러 번 호출해야 한다. C 언어의 문자 데이터를 column에 문자, 이진 또는 데이터 소스 데이터 형식으로 전송하거나 C 언어의 이진 데이터를 column에 문자, 이진 또는 데이터 소스 정의 데이터 형식으로 보내는 경우에만 동일한 column에 대해 여러 번 SQLPutData를 호출할 수 있다.
5. SQLParamData를 다시 호출하여 모든 데이터가 column으로 전송되었음을 알린다.
 - Data-at-execution column들이 더 있을 경우, SQLParamData는 SQL_NEED_DATA와 다음에 처리될 data-at-execution column의 TargetValuePtr 버퍼의 주소를 반환한다. 응용 프로그램은 4단계와 5단계를 반복한다.
 - Data-at-execution column들이 없을 경우, 처리는 완료된다. 명령문이 성공적으로 실행되었다면, SQLParamData는 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO를 반환한다. 실행에 실패했을 경우, 이것은 SQL_ERROR를 반환한다. 이 때, SQLParamData는 SQLSetPos가 반환할 수 있는 SQLSTATE를 반환한다.

데이터가 갱신되었을 경우, 드라이버는 해당 row의 구현 row 상태 배열 값을 SQL_ROW_UPDATED로 변경한다.

연산이 취소되거나 SQLParamData 또는 SQLPutData에서 오류가 발생할 경우, SQLSetPos가 SQL_NEED_DATA를 반환한 후 모든 data-at-execution column이 전송되기 전에, 응용 프로그램은 명령문이나 명령문과 관련된 연결에 대해 SQLCancel, SQLGetDiagField, SQLGetDiagRec, SQLGetFunctions, SQLParamData 또는 SQLPutData만 호출할 수 있다. 이 때 다른 함수를 호출할 경우, 함수는 SQL_ERROR와 SQLSTATE HY010 (function sequence error)를 반환한다.

드라이버가 여전히 data-at-execution column들에 대한 데이터를 필요로 하는 동안 응용 프로그램이 SQLCancel을 호출하면 드라이버는 연산을 취소한다. 이 때, 응용 프로그램은 SQLSetPos를 다시 호출할 수 있다. 취소는 커서의 상태나 현재 커서 위치에는 영향을 미치지 않는다.

커서와 관련된 쿼리 명세의 SELECT-list가 동일한 column에 대해 하나 이상의 참조를 포함할 때 오류가 생성되는지, 드라이버가 중복된 참조를 무시하고 요청된 연산을 이행할지는 드라이버에서 정의된다.

벌크 연산 실행

RowNumber 인자가 0이면 드라이버는 operation 인자에 명시된 연산을 SQL_ATTR_ROW_OPERATION_PTR 명령문 속성이 설정한 row 연산 배열의 필드에 SQL_ROW_PROCEED 값을 가지는 row 집합의 모든 row에 대해 수행한다.

이 값은 SQL_DELETE, SQL_REFRESH 또는 SQL_UPDATE의 operation 인자에 대한 RowNumber 인자에 대해 유효하지만 operation 인자가 SQL_POSITION일 때는 유효하지 않다. Operation 인자가 SQL_POSITION 이고 RowNumber 인자가 0인 SQLSetPos는 SQLSTATE HY109 (Invalid cursor position)를 반환한다.

SQLSTATE HYT00 (timeout expired)와 같이 모든 row 집합에 오류가 있을 경우, 드라이버는 SQL_ERROR 와 적절한 SQLSTATE를 반환한다. Row 집합 버퍼의 내용은 정의되지 않고 커서 위치는 변경되지 않는다.

단일 row에 오류가 있을 경우, 드라이버는 다음과 같이 작동한다.

- SQL_ATTR_ROW_STATUS_PTR 명령문 속성이 지정한 row 상태 배열에 있는 row의 요소를 SQL_ROW_ERROR로 설정한다.
- 오류 queue의 오류에 하나 이상의 SQLSTATE를 추가하고 진단 데이터 구조체의 SQL_DIAG_ROW_NUMBER

필드를 설정한다.

오류 또는 경고가 처리된 후에 드라이버가 row 집합에 남아 있는 row 연산을 완료하면 SQL_SUCCESS_WITH_INFO를 반환한다. 따라서 오류를 반환한 row들에 대해 오류 queue는 0 개 이상의 SQLSTATE를 포함한다. 드라이버는 오류 또는 경고가 처리된 후에 연산을 중단하고 SQL_ERROR를 반환 한다.

드라이버가 SQLSTATE 01004 (data truncated)와 같은 경고를 반환하면 특정한 row에 적용되는 오류 정보를 반환 하기 전에 전체 row 집합 또는 row 집합의 알 수 없는 row들에 적용되는 경고를 반환한다. 이런 row들에 대한 오류 정보와 함께 특정 row에 대한 경고를 반환한다.

RowNumber가 0이고 operation이 SQL_UPDATE, SQL_REFRESH 또는 SQL_DELETE일 경우, SQL_ATTR_ROW_S_FETCHED_PTR 명령문 속성에 의해 SQLSetPos가 운영하는 row의 개수가 설정된다.

RowNumber가 0이고 연산이 SQL_DELETE, SQL_REFRESH 또는 SQL_UPDATE일 경우, 연산 후의 현재 row는 연 산 전의 현재 row와 같다.

벌크 연산에서의 row 무시

Row 연산 배열은 SQLSetPos를 사용하여 벌크 연산을 하는 동안 현재 row 집합 안에서 무시해야 할 row를 나타내 기 위해 사용된다. 벌크 연산을 하는 동안 드라이버가 하나 이상의 row들을 무시하도록 하려면 응용 프로그램이 다음 절차를 수행해야 한다.

1. SQLSetStmtAttr를 호출하여 SQL_ATTR_ROW_OPERATION_PTR 명령문 속성이 SQLUSMALLINT 배열을 설정한다. 이 필드는 SQLSetDescField를 호출하여 ARD의 SQL_DESC_ARRAY_STATUS_PTR 헤더 필드를 설정하는 방식으로 설정할 수도 있는데, 이 경우에 응용 프로그램이 설명자 핸들을 획득해야 한다.
2. Row 연산 배열의 각 요소를 다음 두 값 중 하나로 설정한다.
 - SQL_ROW_IGNORE로 설정할 경우, 벌크 연산에서 row가 제외됨을 나타낸다.
 - SQL_ROW_PROCEED로 설정할 경우, 벌크 연산에 row가 포함됨을 나타낸다.
3. SQLSetPos를 호출하여 벌크 연산을 수행한다.

이 때, 다음과 같은 규칙이 row 연산 배열에 적용된다.

- SQL_ROW_IGNORE와 SQL_ROW_PROCEED는 SQL_DELETE나 SQL_UPDATE operation으로 SQLSetPos를 사용하는 벌크 연산에만 영향을 준다. 그것들은 SQL_REFRESH나 SQL_POSITION operation으로 SQLSetPos를 호출하는 데는 영향을 미치지 않는다.
- 포인터는 기본적으로 NULL로 설정한다.
- 포인터가 NULL이면 모든 요소들이 SQL_ROW_PROCEED로 설정된 것처럼 모든 row가 갱신된다.
- 요소를 SQL_ROW_PROCEED로 설정해도 특정한 row에 대해 연산이 발생한다고 보장할 수는 없다. 예를 들어, row 집합의 특정 row가 SQL_ROW_ERROR 상태를 가지고 있을 경우, 드라이버는 응용 프로그램이 SQL_ROW_PROCEED를 명시했는지 여부에 관계없이 row를 갱신할 수 없다. 응용 프로그램은 연산이 성공했는지 확인 하기 위해 항상 row 상태 배열을 확인해야 한다.
- SQL_ROW_PROCEED은 헤더 파일에 0으로 정의되어 있다. 응용 프로그램은 모든 row를 처리하기 위해 row 연산 배열을 0으로 초기화할 수 있다.

- Row 연산 배열의 n 번째 요소를 SQL_ROW_IGNORE로 설정하고 SQLSetPos를 호출하여 벌크 갱신이나 삭제 연산을 수행할 경우, row 집합의 n 번째 row는 SQLSetPos를 호출한 후에 갱신되지 않고 남아 있게 된다.
- 응용 프로그램은 자동으로 읽기 전용 column을 SQL_ROW_IGNORE로 설정해야 한다.

벌크 연산에서 column 무시

하나 이상의 읽기 전용 column들에 갱신을 시도하여 생기는 불필요한 처리 진단을 방지하기 위해 응용 프로그램은 바인딩된 길이/ 지시자 버퍼의 값을 SQL_COLUMN_IGNORE로 설정할 수 있다.

자세한 내용은 `SQLBindCol`을 참조한다.

SQLSetScrollOptions

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.0의 SQLSetScrollOptions 함수는 ODBC 3.x에서 SQLGetInfo와 SQLSetStmtAttr을 호출하는 것으로 대체되었다.

노트

드라이버 관리자가 SQLSetScrollOptions를 지원하지 않는 ODBC 3.x 드라이버로 작동하는 응용 프로그램에 대해 SQLSetScrollOptions를 매핑할 때 드라이버 관리자는 SQL_ROW_SET_SIZE 명령문 옵션을 SQL_ATTR_ROW_ARRAY_SIZE 명령문 속성이 아닌 SQLSetScrollOptions의 RowsetSize 인자로 설정한다. 결과적으로 SQLSetScrollOptions는 SQLFetch나 SQLFetchScroll을 호출하여 여러 개의 row를 fetch 할 때는 응용 프로그램에서 사용할 수 없으며, SQLExtendedFetch를 호출하여 여러 개의 row들을 fetch 할 때만 사용할 수 있다.

SQLSetStmtAttr

적합성

도입된 버전: ODBC 3.0

표준 준수: ISO 92

개요

SQLSetStmtAttr은 명령문과 관련된 속성을 설정한다.

구문

```
SQLRETURN SQLSetStmtAttr(
    SQLHSTMT      StatementHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     StringLength);
```

인자

StatementHandle

[입력] 명령 핸들이다.

Attribute

[입력] 설정할 옵션이다. **설명** 부분을 참조한다.

ValuePtr

[입력] 속성과 관련된 값이다. 속성값에 따라 ValuePtr은 다음 중 하나가 될 수 있다.

- ODBC 설명자 핸들
- SQLINTEGER 값
- SQLULEN 값
- 다음 중 하나의 포인터
 - Null 종료 문자열
 - 바이너리 버퍼
 - SQLLEN, SQLULEN, SQLUSMALLINT 값 혹은 값의 배열
 - 드라이버 정의 값

Attribute 인자가 특정 드라이버 값일 경우, ValuePtr 정수가 입력된다.

StringLength

[입력] Attribute가 ODBC 정의 속성이고 ValuePtr이 문자열이나 바이너리 버퍼를 가리킬 경우, 이 인자는 *ValuePtr의 길이이다. Attribute가 ODBC 정의 속성이고 ValuePtr이 정수일 경우, 이 인자는 무시된다. StringLength는 다음과 같은 값을 가질 수 있다.

Attribute가 드라이버 정의 속성일 경우, 응용 프로그램이 StringLength 인자를 설정하여 드라이버 관리자에 속성의 특성값을 표시한다.

- ValuePtr이 문자열 포인터일 경우, Stringlength는 문자열의 길이이거나 SQL_NTS이다.
- ValuePtr이 이진 버퍼이면, 응용 프로그램은 StringLength에 SQL_LEN_BINARY_ATTR(length) 매크로의 결과를 저장한다. StringLength에 음수값을 저장한다.
- ValuePtr이 문자열이나 이진 버퍼가 아닌 다른 값의 포인터일 경우, StringLength는 SQL_IS_POINTER 값을 가져야 한다.
- ValuePtr이 고정된 길이값을 포함할 경우, StringLength는 SQL_IS_INTEGER 또는 SQL_IS_INTEGER이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
01S02	Option value changed	드라이버가 ValuePtr에 명시된 값을 지원하지 않거나 실행 조건때문에 ValuePtr에 명시된 값이 무효화되어 드라이버가 유사한 값을 대체한 경우이다. (SQLGetStmtAttr을 호출하여 임시로 대체된 값을 알아낼 수 있다.) 대체값은 커서가 종료될 때까지 StatementHandle에 대해 유효하고 그 시점에 명령문 속성은 이전 값으로 되돌아간다. 명령문 속성은 다음 값으로 변경될 수 있다. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_SIMULATE_CURSOR (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	Attribute가 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR 또는 SQL_ATTR_USE_BOOKMARKS이며 커서가 열려 있다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.

SQLSTATE	Error	설명
HY009	Invalid use of null pointer	Attribute 인자가 식별한 명령문 속성은 문자열 속성을 요구하고, ValuePtr 인자는 NULL 포인터이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기 수행 함수가 호출되었으며, SQLSetStmtAttr 함수가 호출되었을 때 해당 비동기 수행 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 실행되어 SQL_PARAM_DATA_AVAILABLE이 반환되었다. 이 함수는 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 호출되었다. 비동기 수행 함수가 StatementHandle에 대해 호출되었으며 이 함수가 호출될 때 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos가 실행되어 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수나 column에 대한 데이터가 전송되기 전에 이 함수가 실행되었다.
HY011	Attribute cannot be set now	Attribute가 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR 이거나 SQL_ATTR_USE_BOOKMARKS이고 명령문이 준비된 (prepared) 상태이다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY017	Invalid use of an automatically allocated descriptor handle	Attribute 인자가 SQL_ATTR_IMP_ROW_DESC이거나 SQL_ATTR_IMP_PARAM_DESC이다. Attribute 인자가 SQL_ATTR_APP_ROW_DESC 또는 SQL_ATTR_APP_PARAM_DESC이고 ValuePtr 값은 원래 ARD 또는 APD에 할당된 핸들이 아닌 암시적으로 할당된 설명자 핸들이다.
HY024	Invalid attribute value	명시된 attribute 값을 고려할 때, ValuePtr에 명시된 값이 유효하지 않다. (드라이버 관리자는 SQL_ATTR_ACCESS_MODE 또는 SQL_ATTR_ASYNC_ENABLE과 같이 별개의 집합을 허용하는 명령문 속성과 연결에만 이 SQLSTATE를 반환한다. 다른 연결과 명령문 속성의 경우, 드라이버는 ValuePtr에 명시된 값을 검증해야 한다.) Attribute 인자가 SQL_ATTR_APP_ROW_DESC 또는 SQL_ATTR_APP_PARAM_DESC이고 ValuePtr은 명시적으로 할당된 설명자 핸들이고 Statement Handle 인자와 같은 연결선상에 있지 않다.
HY090	Invalid string or buffer length	*ValuePtr이 문자열이고 StringLength 인자가 0보다 작다. (SQL_NTS는 아니다.)
HY092	Invalid attribute/option identifier	Attribute 인자에 명시된 값이 드라이버가 지원하는 ODBC 버전에 대해 유효하지 않다. Attribute 인자에 명시된 값이 읽기 전용 속성이다.

SQLSTATE	Error	설명
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	유예 상태에 대한 자세한 내용은 <code>SQLEndTran</code> 을 참조한다.
HYC00	Optional feature not implemented	Attribute 인자에 명시된 값이 드라이버가 지원하는 ODBC 버전에는 유효한 속성이지만 드라이버에서 지원하지 않는다. Attribute 인자가 <code>SQL_ATTR_ASYNC_ENABLE</code>이고 <code>SQL_ASYNC_MODE</code>의 <code>InforType</code>로 <code>SQLGetInfo</code>를 호출했을 때 <code>SQL_AM_CONNECTION</code>가 반환된다. Attribute 인자가 <code>SQL_ATTR_ENABLE_AUTO_IPD</code>이고 연결 속성 <code>SQL_ATTR_AUTO_IPD</code>의 값이 <code>SQL_FALSE</code>이다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 <code>SQL_ATTR_CONNECTION_TIMEOUT</code> 의 <code>SQLSetConnectAttr</code> 을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.
S1118	Driver does not support asynchronous notification	<code>SQLSetStmtAttr</code> 을 호출하여 <code>SQL_ATTR_ASYNC_STMT_EVENT</code> 를 설정할 때, 드라이버에서 비동기 알림을 지원하지 않는다.

설명

명령문 속성은 또 다른 `SQLSetStmtAttr`의 호출에 의해 갱신되거나 `SQLFreeHandle`에 의해 삭제되기 전까지 유효하다. `SQL_CLOSE`, `SQL_UNBIND` 또는 `SQL_RESET_PARAMS` 옵션으로 `SQLFreeStmt`를 호출하는 것은 명령문 속성을 초기화하지 않는다.

일부 명령문 속성의 경우 `ValuePtr`에 명시된 값을 드라이버에서 지원하지 않으면 이를 유사한 값으로 대체한다. 이 경우, 드라이버는 `SQL_SUCCESS_WITH_INFO`와 `SQLSTATE 01S02` (option value changed)를 반환한다. 예를 들어 attribute가 `SQL_ATTR_CONCURRENCY`이고 `ValuePtr`이 `SQL_CONCUR_ROWVER`일때 데이터 소스가 이것을 지원하지 않을 경우, 드라이버는 `SQL_CONCUR_VALUES`로 대체하고 `SQL_SUCCESS_WITH_INFO`를 반환한다. 대체된 값을 알아내기 위해 응용 프로그램이 `SQLGetStmtAttr`을 호출한다.

`ValuePtr`로 설정된 정보 집합 형식은 명시된 attribute에 따라 다르다. `SQLSetStmtAttr`은 문자열 또는 정수값 중 하나의 속성 정보 형식을 허용한다. 각 형식에 대한 자세한 내용은 속성의 설명 부분에 기술되어 있다. 이 포맷은 `SQLGetStmtAttr`의 각 속성에 대해 반환되는 정보에 적용된다. `SQLSetStmtAttr`의 `ValuePtr` 인자가 나타내는 문자열의 길이는 `StringLength`이다.

노트

- SQLSetConnectAttr을 호출하여 연결 레벨에서 명령문 속성을 설정하는 것은 ODBC 3.x에서는 사용하지 않는 기능이다. ODBC 3.x의 응용 프로그램은 절대로 연결 레벨에서 명령문 속성을 설정하면 안된다. ODBC 3.x 명령문 속성은 SQL_ATTR_METADATA_ID를 제외하고 연결 레벨에서 설정할 수 없고 SQL_ATTR_ASYNC_ENABLE 속성은 연결 속성과 명령문 속성 양쪽에 다 해당되며, 연결 레벨 또는 명령문 레벨에서 설정할 수 있는 속성이다.
- ODBC 3.x 드라이버가 ODBC 2.x 응용 프로그램과 함께 동작해야 할 경우, ODBC 3.x 드라이버의 연결 레벨에서 드라이버 버전을 ODBC 2.x로 설정하는 옵션 기능이 필요하다.

설명자 필드를 설정하는 명령문 속성

많은 명령문 속성이 설명자의 헤더 필드와 대응한다. 이런 속성들을 설정하는 것은 설명자의 필드들을 설정하는 것과 결과가 같다. 필드들을 설정할 때 SQLSetDescField 대신 SQLSetStmtAttr을 호출하면 함수 호출에 대한 설명자 핸들을 획득할 필요가 없어 더 효과적이다.

주의

단일 명령문에 대해 SQLSetStmtAttr을 호출해도 다른 명령문에 영향을 미칠 수 있다. 이것은 명령문에 APD나 ARD가 명시적으로 할당되고 이것이 다른 명령문과 관련있을 때 발생한다. SQLSetStmtAttr이 APD나 ARD를 변경하기 때문에 변경 사항은 이 설명자와 관련된 모든 명령문에 적용된다. 만약 이것이 의도한 동작이 아닐 경우, 응용 프로그램이 SQLSetStmtAttr을 다시 호출하기 전에 (SQLSetStmtAttr을 호출하여 SQL_ATTR_APP_ROW_DESC 또는 SQL_ATTR_APP_PARAM_DESC 필드를 다른 설명자 핸들에 설정하여) 다른 명령문들과 이 설명자를 분리시켜야 한다.

설명자 필드가 대응되는 명령문 속성을 설정한 결과처럼 설정될 때, 필드는 StatementHandle 인자로 식별되는 현재 명령문에 적용할 수 있는 설명자들에 한해 적용되며 속성 설정은 향후 관련된 어떤 명령문 설명자에도 영향을 미치지 않는다. 명령문 핸들과 관련 있는 설명자 필드가 SQLSetDescField로 설정될 경우, 대응하는 명령문 속성도 설정된다. 만약 명시적으로 할당된 설명자가 명령문과 분리되면, 헤더 필드에 대응하는 명령문의 속성은 암시적으로 할당된 설명자의 필드값으로 되돌아간다.

명령문이 할당될 때, 네 개의 설명자 핸들이 자동으로 할당되며 명령문과 연관지어진다. 명시적으로 할당된 설명자 핸들은 SQL_HANDLE_DESC의 HandleType으로 SQLAllocHandle을 호출하여 할당하고 SQLSetStmtAttr 호출하여 명령문과 연관지을 수 있다.

설명자 헤더 필드에 대응하는 명령문 속성들은 다음과 같다.

명령문 속성	헤더 필드	설명
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	ARD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	ARD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

명령문 속성

현재 정의된 속성들과 그것들이 도입된 ODBC 버전은 다음 표와 같다.

속성	ValuePtr 내용
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	드라이버에서 지원하지 않는다.
SQL_ATTR_ATOMIC_EXECUTION	SQLUSMALLINT: Atomic insert의 동작 여부이다. - SQL_ATTR_ATOMIC_EXECUTION_OFF - SQL_ATTR_ATOMIC_EXECUTION_ON
	SQL_ATTR_CONCURRENCY의 기본값은 SQL_CONCUR_READ_ONLY이다.

속성	ValuePtr 내용
SQL_ATTR_CONCURRENCY (ODBC 2.0)	- SQLULEN: 커서의 동시성을 명시한 값이다. - SQL_CONCUR_READ_ONLY: 커서가 읽기 전용이고, 갱신을 허용하지 않는다. - SQL_CONCUR_LOCK: 커서가 row의 갱신을 완료할 수 있는 최저 level의 locking을 사용한다. - SQL_CONCUR_ROWVER: 커서가 SQLBase ROWID 또는 Sybase TIMESTAMP와 같이 row의 버전을 제어, 비교하는 동시성을 사용한다. - SQL_CONCUR_VALUES: 커서가 값을 제어하고 비교하는 동시성을 사용한다. SQL_ATTR_CURSOR_TYPE attribute가 SQL_ATTR_CONCURRENCY에서 지원하지 않는 값으로 갱신될 경우, SQL_ATTR_CONCURRENCY의 값은 실행 시점에 갱신되며, SQLExecDirect 또는 SQLPrepare가 호출될 때 경고가 발생한다. 드라이버가 SELECT FOR UPDATE 명령문을 지원하고 이런 명령문이 실행되는 동안 SQL_ATTR_CONCURRENCY가 SQL_CONCUR_READ_ONLY로 변경될 경우, 오류를 반환한다. SQL_ATTR_CONCURRENCY 값이 드라이버가 지원하는 SQL_ATTR_CURSOR_TYPE 값이나 지원하지 않는 값으로 갱신될 경우, SQL_ATTR_CURSOR_TYPE 값은 실행되는 시점에 갱신되고 SQLExecDirect 또는 SQLPrepare가 실행될 때 SQLSTATE 01S02 (option value changed)가 발생한다. 명시된 동시성을 데이터 소스에서 지원하지 않을 경우, 드라이버는 다른 동시성으로 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. 드라이버는 SQL_CONCUR_VALUES를 SQL_CONCUR_ROWVER로 대체하거나 반대로 대체한다. 또한 SQL_CONCUR_LOCK은 SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES 순으로 대체한다. 대체된 값의 유효성은 실행 시점 전에는 확인되지 않는다.
SQL_ATTR_CURSOR_HOLDABLE	이 속성은 트랜잭션 (transaction)이 종료될 때 커서 (cursor)를 유지 (hold) 할지 여부를 지정하는 SQLULEN 타입의 값이다. - SQL_NONHOLDABLE: 트랜잭션이 종료되면 커서는 사라지고 더 이상 사용할 수 없다. 즉, 커서의 유지 기간이 현재 트랜잭션에만 한정된다. - SQL_HOLDABLE: 트랜잭션이 종료된 후에도 커서는 계속 유지 (hold) 되며, 이후에도 계속 사용할 수 있다. 커서의 유지 기간이 트랜잭션의 경계를 넘어 확장된다. 기본값 (default)은 SQL_HOLDABLE이다. 이 속성을 설정하면 이후 실행되는 SQLExecDirect 또는 SQLExecute 호출에 영향을 미친다.
SQL_ATTR_CURSOR_SCROLLABLE (ODBC 3.0)	이 속성은 애플리케이션에서 필요한 cursor 지원 수준을 지정하는 SQLULEN 타입의 값이다. 이 속성을 설정하면 이후의 SQLExecDirect와 SQLExecute 호출에 영향을 미친다. SQL_NONSCROLLABLE: 이 값이 기본값 (default)이다. 이 값은 statement handle에서 스크롤 가능한 커서 (scrollable cursor)가 필요하지 않음을 나타낸다. 이 handle에서 애플리케이션이 SQLFetchScroll을 호출하면, FetchOrientation으로 지정할 수 있는 유효한 값은 오직 SQL_FETCH_NEXT 뿐이다.

속성	ValuePtr 내용
	SQL_SCROLLABLE: 이 값은 statement handle에서 스크롤 가능한 커서 (scrollable cursor)가 필요함을 나타낸다. SQLFetchScroll을 호출할 때, 애플리케이션은 FetchOrientation의 모든 유효한 값을 지정할 수 있으며, 순차적인 방식 (sequential mode)이 아닌 다른 방식으로 커서 위치를 이동할 수 있다.
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	이 속성은 statement handle의 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영할지 (visible) 여부를 지정하는 SQLULEN 타입의 값이다. 이 속성을 설정하면 이후의 SQLExecDirect와 SQLExecute 호출에 영향을 미친다. 애플리케이션은 이 속성의 값을 읽어 초기 상태 또는 최근에 설정된 상태를 확인할 수 있다. - SQL_UNSPECIFIED: 이 값이 기본값 (default) 이다. 커서 유형이나 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영할지 여부가 명시되지 않은 상태이다. 이 statement handle의 cursor는 다른 cursor의 변경 사항을 전혀 반영하지 않을 수도 있고, 일부 또는 전체를 반영할 수도 있다. - SQL_INSENSITIVE: 이 값은 statement handle의 모든 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영하지 않은 상태의 데이터를 표시함을 나타낸다. Insensitive 커서는 읽기 전용 (read-only)이다. 이는 읽기 전용 동시성 (read-only concurrency)을 가지는 static 커서에 해당한다. - SQL_SENSITIVE: 이 값은 statement handle의 모든 커서가 다른 커서에 의해 변경된 결과 집합 (result set)의 변화를 반영한 상태의 데이터를 표시함을 나타낸다.
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	SQL_ATTR_CURSOR_TYPE (ODBC 2.0) 이 속성은 cursor의 유형을 지정하는 SQLULEN 값이다: - SQL_CURSOR_FORWARD_ONLY: 커서는 오직 앞으로만 (forward-only) 이동할 수 있다. - SQL_CURSOR_STATIC: 결과 집합 (result set)의 데이터가 정적 (static) 이다. - SQL_CURSOR_KEYSET_DRIVEN: 드라이버가 SQL_ATTR_KEYSET_SIZE statement 속성으로 지정된 행 수만큼의 키를 저장하고 사용한다. - SQL_CURSOR_DYNAMIC: 드라이버가 현재 rowset에 포함된 행들의 키만 저장하고 사용한다. 기본값 (default)은 SQL_CURSOR_FORWARD_ONLY이다. SQL statement가 이미 prepared 된 이후에는 이 값을 설정할 수 없다. 지정된 cursor type을 데이터 소스가 지원하지 않는 경우, 드라이버는 다른 cursor type으로 대체한 후 SQLSTATE 01S02 (Option value changed)를 반환한다. 만약 mixed 또는 dynamic 커서를 지원하지 않는 경우, 드라이버는 우선 keyset-driven cursor로 대체를 시도하고, 이 또한 지원되지 않으면 static cursor로 대체한다. 또한, keyset-driven cursor가 지원되지 않는 경우에도 static cursor로 대체된다.
SQL_ATTR_ENABLE_AUTO_IPD	드라이버에서 지원하지 않는다.

속성	ValuePtr 내용
(ODBC 3.0)	
SQL_ATTR_EXPLAIN_PLAN_OPTION	SQLUSMALLINT: Plan 정보 생성 여부이다. - SQL_EXPLAIN_PLAN_OFF: Plan 정보를 생성하지 않는다. - SQL_EXPLAIN_PLAN_ON: SQL 구문을 수행하고, plan 정보를 생성한다. - SQL_EXPLAIN_PLAN_ONLY: SQL 구문을 수행하지 않고, plan 정보를 생성한다.
SQL_ATTR_EXPLAIN_PLAN_TEXT	생성된 plan 문자열 (읽기 전용)이다.
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_FETCH_FAILOVER	SQLUSMALLINT: fetch failover 사용 여부이다. - SQL_FETCH_FAILOVER_OFF: fetch failover를 사용하지 않는다. - SQL_FETCH_FAILOVER_ON: fetch failover를 사용한다.
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_MAX_ROWS (ODBC 1.0)	SQLULEN: SELECT 명령문에서 반환되는 row들의 최대 개수에 해당하는 값이다. *ValuePtr이 0과 같을 경우, 드라이버는 모든 row들을 반환한다. 이 속성은 네트워크 트래픽을 감소시키는 것을 목적으로 한다. 개념적으로 이것은 결과 집합이 만들어질 때 적용되고 첫 번째 ValuePtr row들의 결과 집합을 제한한다. 결과 집합의 row들의 개수가 ValuePtr보다 클 경우, 결과 집합이 감소한다. SQL_ATTR_MAX_ROWS는 카탈로그 함수가 반환하는 명령문들을 포함하여 명령문의 모든 결과 집합들에 적용된다. SQL_ATTR_MAX_ROWS는 커서 row 개수 값의 최대치를 설정한다. SQL_ATTR_MAX_ROWS가 제대로 구현될 지 여부를 보장할 수 없는 경우 (결과 집합 크기 제한이 데이터 소스에서 구현될 수 없는 경우) 드라이버는 SQLFetch 또는 SQLFetchScroll에 대해 SQL_ATTR_MAX_ROWS 동작을 모방해서는 안된다. (카탈로그 함수들과 같은) SELECT 명령문들 이외의 명령문들에 SQL_ATTR_MAX_ROWS를 적용할지 여부는 드라이버에서 정의한다. 이 속성의 값은 열려 있는 커서에 설정할 수 있다. 그러나 효과가 바로 나타나지는 않는다. 이 경우, 드라이버는 SQLSTATE 01S02 (option value changed)를 반환하고 속성을 원래 값으로 되돌린다.

속성	ValuePtr 내용
SQL_ATTR_METADATA_ID (ODBC 3.0)	SQLULEN: 카탈로그 함수들의 문자열 인자들을 어떻게 취급할지에 대한 값이다. SQL_TRUE일 경우, 카탈로그 함수들은 문자열 인자들을 식별자처럼 취급한다. 이 경우 대소문자를 구분하지 않는다. 범위가 정해져 있지 않은 문자열들의 경우 드라이버가 후 공백을 제거하고 문자열을 대문자로 변환한다. 범위가 정해진 문자열들의 경우 드라이버는 앞 뒤 공백을 제거하고 구분자들 사이의 문자 그대로를 취한다. 이 인자들 중 하나가 NULL 포인터로 설정될 경우, 함수는 SQL_ERROR와 SQLSTATE HY009 (invalid use of null pointer)를 반환한다. SQL_FALSE일 경우, 카탈로그 함수들의 문자열 인자들을 식별자처럼 취급하지 않는다. 이 경우 대소문자를 구분한다. 인수에 따라 인자들이 문자열 검색 패턴을 포함할 수도 있고 그렇지 않을 수도 있다. 기본값은 SQL_FALSE이다. 값 목록을 취하는 SQLTables의 TableType 인자는 이 속성에 영향을 받지 않는다. SQL_ATTR_METADATA_ID 역시 연결 level에서 설정될 수 있다. (이것과 SQL_ATTR_ASYNC_ENABLE은 고유하며 명령문 속성들이면서 연결 속성들이기도 하다.) 자세한 내용은 카탈로그 함수의 인자를 참조한다.
SQL_ATTR_NOSCAN (ODBC 1.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	SQLULEN*: 동적 매개 변수들의 바인딩을 변경하기 위해 포인터에 추가된 offset을 가리키는 값이다. 이 필드가 NULL이 아닐 경우, 드라이버는 포인터를 역참조하고 설명자 레코드 (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) 안의 지어진 각 필드들에 역참조된 값을 추가하며 새로운 포인터 값들은 바인딩 할 때 사용한다. 이것은 기본적으로 NULL로 설정된다. 바인드 offset은 항상 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR 필드들에 직접 추가된다. 만약 offset이 다른 값으로 바뀔 경우, 새로운 값이 설명자 필드값에 직접 추가된다. 새로운 offset은 필드값과 이전의 offset 값을 더한 값에 추가되지 않는다. 이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_BIND_TYPE 필드를 설정한다.
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	SQLULEN: 동적 매개 변수에 사용될 바인딩 방향을 표시한다. 이 필드는 column 방향 바인딩 용 SQL_PARAM_BIND_BY_COLUMN으로 설정되어 있다. (기본값) Row 방향 바인딩을 선택하려면 이 필드는 구조체의 길이 또는 동적 매개 변수 집합에 바인딩될 버퍼의 인스턴스로 설정해야 한다. 이 길이가 바인딩 매개 변수와 구조체의 패딩 모두를 위한 공간을 포함하거나 바인딩 매개 변수 주소가 지정된 길이로 증가할 때 결과가 반드시 다음 매개 변수의 시작을 가리키도록 버퍼링 해야한다. ANSI C의 sizeof 연산자를 사용하여 이 동작을 보장할 수 있다.

속성	ValuePtr 내용
	이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_BIND_TYPE 필드를 설정한다.
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	SQLUSMALLINT*: SQL 명령문이 실행되는 동안 매개 변수를 무시하는데 사용되는 SQL USMALLINT 값들의 배열을 가리키는 값이다. 각각의 값은 (매개 변수를 실행시키려면) SQL_PARAM_PROCEED 또는 (매개 변수를 무시하려면) SQL_PARAM_IGNORE이다. 매개 변수 집합은 APD 안의 SQL_DESC_ARRAY_STATUS_PTR이 가리키는 배열의 상태값을 설정하면 처리 중에 무시할 수 있다. 매개 변수 집합은 이 상태값이 SQL_PARAM_PROCEED로 설정되거나 배열 요소가 설정되지 않았을 경우에 처리된다. 이 명령문 속성은 NULL 포인터로 설정될 수 있다. 이 경우, 드라이버는 매개 변수 상태값을 반환하지 않는다. 이 속성은 어느 때든 설정될 수 있지만 새로운 값은 다음 SQLExecDirect나 SQLExecute가 호출될 때까지 사용되지 않는다. 이 속성은 바인딩된 매개 변수가 없는 경우 무시된다. 이 명령문 속성을 설정하여 APD 헤더 안의 SQL_DESC_ARRAY_STATUS_PTR 필드를 설정한다.
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	- SQLUSMALLINT*: SQLExecute나 SQLExecDirect를 호출한 후에 매개 변수의 각 row들에 대한 상태 정보값들을 포함하는 SQLUSMALLINT 값의 배열을 가리키는 값이다. 이 필드는 PARAMSET_SIZE가 1보다 클 때만 요구된다. 상태값은 다음 값들을 포함할 수 있다. - SQL_PARAM_SUCCESS: SQL 명령문이 이 매개 변수 집합에 대해 성공적으로 실행됐다. - SQL_PARAM_SUCCESS_WITH_INFO: SQL 명령문이 이 매개 변수 집합에 대해 성공적으로 실행됐지만 진단 데이터 구조체에 경고 정보가 있다. - SQL_PARAM_ERROR: 매개 변수 집합을 처리하는데 오류가 발생했다. 추가 오류 정보는 진단 데이터 구조체에 있다. - SQL_PARAM_UNUSED: 일부 이전 매개 변수 집합에 의해 처리가 중단되는 오류가 발생하였거나 SQL_ATTR_PARAM_OPERATION_PTR이 명시한 배열의 매개 변수 집합이 SQL_PARAM_IGNORE로 설정됨에 따라 매개 변수 집합이 사용되지 않았다. - SQL_PARAM_DIAG_UNAVAILABLE: 오류 정보 레벨을 생성하지 않기 때문에 드라이버가 매개 변수 배열을 확실적인 단위로 취급한다. 이 명령문 속성은 NULL 포인터로 설정될 수 있고, 이 경우, 드라이버는 매개 변수의 상태값들을 반환하지 않는다. 이 속성은 언제든지 설정할 수 있다. 하지만 새로운 값은 다음 SQLExecDirect나 SQLExecute가 호출될 때까지 사용되지 않는다. 이 속성을 설정하면 드라이버가 매개 변수를 출력하는 행동에 영향을 미칠 수 있다. 이 명령문 속성을 설정하여 IPD 헤더의 SQL_DESC_ARRAY_STATUS_PTR 필드를 설정한다.
SQL_ATTR_PARAMS_PROCESSED_PTR	SQLULEN*: 처리된 매개 변수 집합의 개수를 반환하는 버퍼를 가리키는 레코드 필드으로써 오류 집합을 포함한다. NULL 포인터는 반환하지 않는다.

속성	ValuePtr 내용
(ODBC 3.0)	이 명령문 속성을 설정하여 IPD 헤더의 SQL_DESC_ROWS_PROCESSED_PTR 필드를 설정한다. 이 속성에 표시된 버퍼를 채울 SQLExecDirect 또는 SQLExecute가 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용은 정의되지 않는다.
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	SQLULEN: 각 매개 변수 값의 개수를 명시한 값이다. SQL_ATTR_PARAMSET_SIZE가 1보다 클 경우, APD의 SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR은 배열을 가리킨다. 각 배열 상수는 필드의 값과 같다. 바인딩 된 매개 변수가 없을 경우, 이 속성은 무시된다. 이 명령문 속성을 설정하여 APD 헤더의 SQL_DESC_ARRAY_SIZE 필드를 설정한다.
SQL_ATTR_PREFETCH_ROWS	이 속성은 한 번의 fetch 작업으로 드라이버가 미리 가져오는 (prefetch) 행의 개수를 지정하는 SQLULEN 타입의 값이다. 기본값 (default)은 1000이다. 이 값을 증가시키면 네트워크 왕복 (network round-trip) 횟수가 줄어들어 성능이 향상될 수 있지만, 메모리 사용량은 증가할 수 있다. 반대로 이 값을 줄이면 메모리 사용량은 줄어들지만 네트워크 왕복 횟수가 증가할 수 있다. 이 속성은 SQLExecute 또는 SQLExecDirect 이후에도 설정할 수 있으며, 이후 SQLFetch나 SQLFetchScroll을 호출할 때 설정된 값만큼 행을 미리 가져온다 (prefetch). 이미 fetch가 진행된 statement handle이라도, 다음 fetch 작업부터는 새로운 설정값이 적용된다.
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	SQLULEN: SQL 명령문을 실행하고 응용 프로그램으로 반환되기 전에 대기하는 초단위의 값이다. ValuePtr이 0(기본값)일 경우, 타임아웃은 발생하지 않는다. 명시된 타임아웃 값이 데이터 소스의 최대값을 초과하거나 최소값보다 작을 경우, SQLSetStmtAttr이 값을 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. SELECT 명령문이 타임아웃되더라도 응용 프로그램에서 명령문을 재사용하기 위해 SQLCloseCursor를 호출할 필요가 없다. 이 명령문 속성의 쿼리 타임아웃을 설정하는 것은 동기 방식과 비동기 방식 모두에 유효하다.
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	SQLULEN: SQLFetch나 SQLFetchScroll을 각각 호출하여 반환되는 row들의 개수를 명시한 값이다. 이것은 SQLBulkOperations에서 벌크 북마크 연산에 쓰이는 북마크 배열의 row의 개수이기도 하다. 기본값은 1이다. 명시한 row 집합의 크기가 데이터 소스에서 지원하는 row 집합 크기의 최대값을 초과할 경우, 드라이버는 값을 대체하고 SQLSTATE 01S02 (option value changed)를 반환한다. 이 명령문 속성을 설정하여 ARD 헤더의 SQL_DESC_ARRAY_SIZE 필드를 설정한다.

속성	ValuePtr 내용
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	SQLULEN: 열 데이터의 바인딩 변경을 표시하기 위해 추가된 offset을 가리키는 값이다. 이 필드가 NULL이 아닐 경우, 드라이버는 포인터를 역참조하여 설명자 레코드(SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR)에 거치한 필드들 각각에 역참조 값을 추가하고 바인딩될 때 새로운 포인터 값을 사용한다. 기본적으로 NULL 값을 설정한다. 이 명령문 속성을 설정하는 것은 ARD 헤더의 SQL_DESC_BIND_OFFSET_PTR 필드를 설정하는 것이다.
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	SQLULEN: SQLFetch나 SQLFetchScroll을 관련된 명령문에서 호출할 때 바인딩 방향을 설정하는 값이다. SQL_BIND_BY_COLUMN 값을 설정하면 column 방향 바인딩이 선택된다. 결과 column들이 바인딩 될 구조나 버퍼의 인스턴스 길이에 대한 값을 설정하면 row 방향 바인딩이 선택된다. 길이가 명시되면 바인딩 된 column의 주소가 일정 길이만큼 증가했을 때 결과가 동일한 column의 시작을 가리키도록 모든 바인딩 된 column과 구조체 또는 버퍼의 패딩을 위한 공간을 포함하여야 한다. ANSI C의 구조체나 공용체에 sizeof 연산자를 사용하여 이 동작을 보장할 수 있다. Column 방향 바인딩은 SQLFetch와 SQLFetchScroll의 기본 바인딩 방향이다. 이 명령문 속성을 설정하여 ARD 헤더의 SQL_DESC_BIND_TYPE 필드를 설정한다.
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	SQLUSMALLINT*: SQLFetch나 SQLFetchScroll을 호출한 뒤 row 상태값들을 포함하는 SQLUSMALLINT 배열값을 가리키는 값이다. 배열은 row 집합에 포함된 row 개수만큼의 요소들을 가지고 있다. 이 명령문 속성은 NULL 포인터로 설정될 수 있는데 이 경우 드라이버는 row 상태값들을 반환하지 않는다. 이 속성은 언제든지 설정할 수 있지만 다음 SQLBulkOperations, SQLFetch, SQLFetchScroll 또는 SQLSetPos를 호출할 때까지 새로운 값은 사용되지 않는다. 이 명령문 속성을 설정하여 IRD 헤더의 SQL_DESC_ARRAY_STATUS_PTR을 설정할 수 있다. 이 속성은 ODBC 2.x 드라이버에서 SQLExtendedFetch의 rgbRowStatus 배열에 매핑된다.
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	SQLULEN*: SQLFetch나 SQLFetchScroll을 호출한 뒤에 회수된 row들의 개수를 반환하는 버퍼를 가리킨다. SQL_REFRESH의 operation 인수로 SQLSetPos를 호출하여 수행한 벌크 연산의 영향을 받는 row의 개수이거나 SQLBulkOperations로 수행한 벌크 연산의 영향을 받는 row의 개수이다. 이 row의 개수는 오류 row들을 포함한다. 이 명령문 속성을 설정하여 IRD 헤더의 SQL_DESC_ROWS_PROCESSED_PTR 필드를 설정할

속성	ValuePtr 내용
	수 있다. 이 속성이 가리키는 버퍼를 채우는 SQLFetch나 SQLFetchScroll을 호출했을 때 SQL_SUCCESS나 SQL_SUCCESS_WITH_INFO를 반환하지 않을 경우, 버퍼의 내용들은 정의되지 않는다.
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	드라이버에서 지원하지 않는다.
SQL_ROWSET_SIZE (ODBC 2.0)	이 속성은 SQLExtendedFetch 호출 한 번으로 반환되는 행의 개수, 즉 rowset의 크기를 지정하는 32비트 정수 (integer) 타입의 값이다. 기본값 (default)은 1이다. 이 값을 1보다 크게 설정하면 한 번의 SQLExtendedFetch 호출로 여러 개의 행을 한꺼번에 가져올 수 있어 애플리케이션에서 대량의 데이터를 효율적으로 처리할 수 있다. 이 속성은 커서 (cursor)가 열린 (open) 상태에서도 설정할 수 있다. 이 속성은 SQLExtendedFetch 함수에만 적용되며, SQLFetch나 SQLFetchScroll 함수의 rowset 크기를 지정하려면 SQL_ATTR_ROW_ARRAY_SIZE 속성을 사용해야 한다.

SQLSetStmtOption

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.0의 SQLSetStmtOption 함수는 ODBC 3.x에서 SQLSetStmtAttr로 대체되었다. 자세한 내용은 [SQLSetStmtAttr](#)을 참조한다.

SQLSpecialColumns

적합성

도입된 버전: ODBC 1.0

표준 준수: Open group

개요

SQLSpecialColumns 함수는 지정된 테이블 내의 column에 대해 다음과 같은 정보를 검색한다.

- 테이블의 row를 유일하게 식별하는 최적의 column 집합
- Row 값이 갱신될 때 트랜잭션이 자동으로 갱신하는 column들

구문

```
SQLRETURN SQLSpecialColumns(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   IdentifierType,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3,
    SQLSMALLINT   Scope,
    SQLSMALLINT   Nullable);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

IdentifierType

[입력] 반환되는 column의 타입이다. 반드시 다음 값 중의 하나이어야 한다.

- SQL_BEST_ROWID: Column(s)에서 값을 검색하여 지정된 테이블의 모든 행을 고유하게 식별하는 최적의 열 또는 열 집합을 반환한다. Column은 특별한 목적에 맞게 설계된 pseudo column (ORACLE의 ROWID 또는 INGRES의 TID), 일반 column 또는 테이블의 고유 인덱스 column이 될 수 있다.
- SQL_ROWVER: Row 값이 (SQLBase ROWID나 Sybase TIMESTAMP와 같은) 트랜잭션에 의해 갱신될 때 데이터 소스에 의해 자동으로 갱신되는 지정된 테이블의 column을 반환한다.

CatalogName

[입력] 테이블 카탈로그이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 카탈로그를 가지지 않는다. 카탈로그 이름은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 문자열 인자로 취급되며, 대소문자를 구분한다. 자세한 내용은 **카탈로그 함수의 인자**를 참조한다.

NameLength1

[입력] *CatalogName 문자열의 길이이다.

SchemaName

[입력] 테이블 스키마의 이름이다. 스키마 이름에 대한 문자열 검색 패턴이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 스키마를 갖지 않는다. SchemaName은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 일반 문자열 인자로 취급되며, 대소문자를 구분한다.

NameLength2

[입력] *SchemaName 문자열의 길이이다.

TableName

[입력] 테이블 이름이다. 이 인자는 NULL 포인터가 될 수 없다. 테이블 이름은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, TableName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, TableName은 일반 문자열 인자로 취급되며, 대소문자를 구분한다.

NameLength3

[입력] *TableName 문자열의 길이이다.

Scope

[입력] 최소한으로 요구되는 rowid의 범위이다. 반환되는 ROWID는 더 큰 범위일 수 있다. 다음 값 중 하나이어야 한다.

- SQL_SCOPE_CURROW: Rowid는 해당 row에 위치하는 동안만 유효하다. 다른 트랜잭션에 의해 갱신되거나 삭제되는 경우, 사용하던 Rowid는 조회되지 않는다.
- SQL_SCOPE_TRANSACTION: Rowid는 현재 트랜잭션이 유지되는 동안 유효하다.
- SQL_SCOPE_SESSION: Rowid는 (트랜잭션의 경계를 넘어) 세션이 유지되는 동안 유효하다.

Nullable

[입력] 특정 column이 NULL 값을 가질 수 있는지 여부를 확인한다. 다음 중 하나이어야 한다.

- SQL_NO_NULLS: 특정 column이 NULL 값을 가질 수 없다. 일부 드라이버들은 SQL_NO_NULLS를 지원하지 않으며 이 드라이버들은 SQL_NO_NULLS가 명시될 경우에 빈 결과 집합을 반환한다. 응용 프로그램은 이런 경우에 대비해 꼭 필요한 경우에만 SQL_NO_NULLS를 요청해야 한다.

- SQL_NULLABLE: 특정 column이 NULL 값을 가질 수 있다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	StatementHandle에 커서가 열려있고, SQLFetch 또는 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 커서가 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	함수가 실행되는 동안 관련된 연결이 실패하여, 트랜잭션의 상태를 알 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY008	Operation canceled	StatementHandle에 비동기 처리를 사용할 수 있었다. 이 함수가 호출되었고 완료되기 전에 StatementHandle로 SQLCancel이나 SQLCancelHandle이 호출되었다. 그 후 해당 함수가 StatementHandle로 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY009	Invalid use of null pointer	TableName 인자가 NULL 포인터이다. SQL_ATTR_METADATA_ID 구문 속성이 SQL_TRUE로 설정되고, CatalogName 인자가 null 포인터이며, SQL_CATALOG_NAME InfoType은 카탈로그 이름이 지원됨을 반환한다. SQL_ATTR_METADATA_ID가 SQL_TRUE로 설정되었고, SchemaName 인자가 null 포인터이다.
		StatementHandle과 관련된 연결 핸들에 대해 SQL 비동기 수행 함수가 호출되었으며, SQLSpecialColumns가 호출되었을 때 이 함수가 여전히 실행되고 있

SQLSTATE	Error	설명
HY010	Function sequence error	었다. StatementHandle에 대해 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 실행되었고, SQL_PARAM_DATA_AVAILABLE을 반환했다. 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 해당 함수가 호출되었다. StatementHandle에 대해 비동기 수행 함수가 호출되었으며 이 함수가 호출되었을 때 해당 비동기 수행 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 혹은 SQLSetPos가 호출되었고 SQL_NEED_DATA이 반환되었고, 모든 data-at-execution 매개 변수와 column에 대한 데이터가 보내지기 전에 이 함수가 실행되었다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY090	Invalid string or buffer length	길이 인자 중 하나의 값이 0보다 작다. (SQL_NTS를 의미하지는 않는다.) 이름 길이 인자 중 하나의 값이 최대 길이를 초과한다.
HY097	Column type out of range	유효하지 않은 IdentifierType이 명시되었다.
HY098	Scope type out of range	유효하지 않은 범위값이 명시되었다.
HY099	Nullable type out of range	유효하지 않은 nullable 값이 명시되었다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
HYC00	Optional feature not implemented	명시된 카탈로그를 드라이버나 데이터 소스에서 지원하지 않는다. 명시된 스키마를 드라이버나 데이터 소스에서 지원하지 않는다. 명시된 문자열 검색 패턴을 카탈로그 이름, 테이블 스키마, 테이블 이름, 데이터 소스 전체 또는 하나 이상에서 지원하지 않는다. SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE의 현상태 조합을 드라이버나 데이터 소스에서 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS가 SQL_UB_VARIABLE로 설정되어 있고, SQL_ATTR_CURSOR_TYPE이 드라이버에서 북마크를 지원하지 않는 커서 타입으로 설정된다.
HYT00	Timeout expired	데이터 소스에서 요청한 결과 집합 데이터를 반환하기 전에 쿼리 타임아웃 주기가 만료되었다. 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQ

SQLSTATE	Error	설명
		LSetStmtAttr을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetConnectAttr을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync had not been called to complete the previous asynchronous on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 SQLCompleteAsync를 호출해야 한다.

설명

IdentifierType 인자가 SQL_BEST_ROWID인 경우, SQLSpecialColumns는 column 또는 테이블 각각의 row를 고유하게 구분할 수 있는 column을 반환한다. 이 column들은 select-list나 WHERE 절에 사용할 수 있다. SQLColumns는 테이블 column의 다양한 정보를 반환하지만, 트랜잭션이 row 값을 갱신할 때 자동으로 갱신되는 column이나 각 row를 고유하게 식별하는 column을 반환할 필요는 없다. 예를 들어, SQLColumns는 Oracle pseudo column인 ROWID는 반환하지 않는다. 이것이 SQLSpecialColumns가 특별한 column들의 정보 반환에 사용되는 이유이다. 자세한 내용은 **카탈로그 데이터 사용**을 참조한다.

노트

ODBC 카탈로그 함수의 일반적인 사용법, 인자, 반환 데이터에 대해 자세한 내용은 **카탈로그 함수**를 참조한다.

테이블의 각 row를 고유하게 식별하는 column이 없을 경우, SQLSpecialColumns는 어떤 column도 반환하지 않는다. 이후 명령문에 대해 SQLFetch나 SQLFetchScroll을 호출하면 SQL_NO_DATA가 반환된다.

IdentifierType, scope, nullable 인자에 명시한 특성을 데이터 소스에서 지원하지 않을 경우, SQLSpecialColumn은 빈 결과 집합을 반환한다.

SQL_ATTR_METADATA_ID 속성이 SQL_TRUE로 설정될 경우, CatalogName, SchemaName, TableName 인자는 식별자로 취급되어 NULL 포인터를 사용할 수 없게 된다. (자세한 내용은 **카탈로그 함수의 인자**를 참조한다.)

SQLSpecialColumns 함수는 SCOPE로 정렬된 표준 결과 집합 결과를 반환한다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column의 번호로 바인드하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS

COLUMN_NAME column의 실제 길이를 확인하기 위해 응용 프로그램에서 SQLGetInfo 함수를 SQL_MAX_COLUMN_NAME_LEN 옵션과 함께 호출할 수 있다.

다음 표는 결과 집합의 column을 나열한 것이다. Column 8 (PSEUDO_COLUMN) 이후 추가되는 column은 드라이버가 정의할 수 있다. 응용 프로그램은 드라이버에서 정의한 column에 접근하기 위해 명시적인 위치를 지정하는 대신 결과 집합의 뒤에서부터 순차적으로 카운트다운 해야한다. 자세한 내용은 **카탈로그 함수의 데이터 반환**을 참조한다.

Column 이름	Column 번호	데이터 형식	설명
SCOPE (ODBC 1.0)	1	Smallint	ROWID의 실제 범위이다. SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION 또는 SQL_SCOPE_SESSION 값 중 하나를 포함한다. IdentifierType이 SQL_ROWVER이면 NULL을 반환한다. 각 값에 대한 자세한 내용은 구문의 scope에 대한 설명을 참조한다.
COLUMN_NAME (ODBC 1.0)	2	Varchar not NULL	Column 이름이다. Column 이름이 없을 경우, 드라이버는 빈 문자열을 반환한다.
DATA_TYPE (ODBC 1.0)	3	Smallint not NULL	SQL 데이터 형식이다. 이것은 ODBC SQL 데이터 형식이거나 드라이버에서 명시한 SQL 데이터 형식일 수 있다. 모든 ODBC SQL 데이터 형식들이 유효하다. 드라이버에서 명시한 SQL 데이터 형식들에 대한 자세한 내용은 드라이버의 문서를 참조한다.
TYPE_NAME (ODBC 1.0)	4	Varchar not NULL	데이터 소스에 의존적인 데이터 형식의 이름이다. 예를 들어 CHAR, VARCHAR, MONEY, LONG VARBINARY, CHAR() FOR BIT DATA 등과 같은 것들이 있다.
COLUMN_SIZE (ODBC 1.0)	5	Integer	데이터 소스의 column 길이이다.
BUFFER_LENGTH (ODBC 1.0)	6	Integer	SQL_C_DEFAULT가 명시될 경우, SQLGetData나 SQLFetch 연산에서 전송되는 바이트 단위의 데이터 길이이다. Numeric 데이터의 크기는 데이터 소스에 저장된 데이터의 크기와 다를 수 있다. 이 값은 문자열이나 바이너리 데이터의 경우 COLUMN_SIZE column과 같은 값이다.
DECIMAL_DIGITS (ODBC 1.0)	7	Smallint	데이터 소스에서 column의 소수점 자릿수다. 데이터 형식의 소수점 자릿수를 적용할 수 없다면 NULL을 반환한다.
PSEUDO_COLUMN (ODBC 2.0)	8	Smallint	Column이 Oracle ROWID처럼 pseudo-column인지 여부를 표시한다. - SQL_PC_UNKNOWN - SQL_PC_NOT_PSEUDO - SQL_PC_PSEUDO

Column 이름	Column 번호	데이터 형식	설명
			상호 운용성을 극대화하기 위해 pseudo-column은 SQLFetInfo에서 반환되는 식별자 quote를 인용할 수 없다.

응용 프로그램은 SQL_BEST_ROWID 값을 검색한 후에 이 값들을 이용하여 정의된 범위 안에서 row를 다시 선택할 수 있다. SELECT 구문은 no rows와 one row 중 하나의 반환을 보장한다.

응용 프로그램이 ROWID나 column을 기반으로 row를 다시 조회했을 때 row를 찾을 수 없을 경우, 그 row가 삭제되었거나 ROWID column이 갱신되었다고 간주할 수 있다. ROWID가 바뀌지 않았다 하더라도 해당 row의 다른 column이 수정되었을 것이다.

Column 형식 SQL_BEST_ROWID에 대해 반환되는 column들은 응용 프로그램이 row의 집합 중 가장 최근 데이터를 검색한 결과 집합 내에서 앞과 뒤로 움직여야 할 경우 매우 유용하다. ROWID의 column(s)은 해당 row가 위치하는 동안 변경되지 않는다.

ROWID의 column들은 커서가 row에 위치해 있지 않을 때도 유효하게 남아 있다. 응용 프로그램은 결과 집합의 SCOPE column을 점검하여 이를 확인할 수 있다.

SQLStatistics

적합성

도입된 버전: ODBC 1.0

표준 준수: ISO 92

개요

SQLStatistics 함수는 한 개의 테이블과 테이블에 관련된 인덱스에 대한 통계 목록을 검색한다. 드라이버는 결과 집합으로 결과를 반환한다.

구문

```
SQLRETURN SQLStatistics(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3,
    SQLUSMALLINT    Unique,
    SQLUSMALLINT    Reserved);
```

인자

StatementHandle

[입력] 명령문 핸들이다.

CatalogName

[입력] 카탈로그 이름이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열("")을 반환하며 그 테이블들은 카탈로그를 가지지 않는다. 카탈로그 이름은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 문자열 인자로 취급되며, 대소문자를 구분한다. 자세한 내용은 **카탈로그 함수의 인자**를 참조한다.

NameLength1

[입력] *CatalogName 문자열의 길이이다.

SchemaName

[입력] 스키마 이름이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열("") 을 반환하며 그 테이블들은 스키마를 가지지 않는다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. SQL_FALSE로 설정할 경우, SchemaName은 일반 인수로써 문자 그대로 취급되며, 대소문자를 구분한다.

NameLength2

[입력] *SchemaName 문자열의 길이이다.

TableName

[입력] 테이블 이름이다. 이 인자는 NULL 포인터가 될 수 없다. TableName은 문자열 검색 패턴을 포함하지 않는다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 패턴값 문자열 인자로 취급되며, 대소문자를 구분한다.

NameLength3

[입력] *TableName 문자열의 길이이다.

Unique

[입력] 인덱스의 형식이다. SQL_INDEX_UNIQUE 또는 SQL_INDEX_ALL이다.

Reserved

[입력] 결과 집합의 CARDINALITY와 PAGES column의 중요성을 표시한다. 다음 옵션들은 CARDINALITY와 PAGES column의 결과에만 영향을 준다. CARDINALITY와 PAGES가 반환되지 않더라도 인덱스 정보는 반환된다.

- SQL_ENSURE: 드라이버가 무조건 통계를 검색하도록 요청한다. (오픈 그룹 표준을 준수하지만 ODBC 확장을 지원하지 않는 드라이버는 SQL_ENSURE를 지원하지 않는다.)
- SQL_QUICK: 서버에서 쉽게 사용할 수 있는 경우에만 드라이버가 CARDINALITY와 PAGES를 무조건 검색하도록 요청한다. 이 경우, 드라이버는 검색된 값이 반드시 최신값임을 보장하지 않는다. (오픈 그룹에 등록된 응용 프로그램은 항상 ODBC 3.x를 준수하는 드라이버로부터 SQL_QUICK 동작을 얻는다.)

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.

SQLSTATE	Error	설명
24000	Invalid cursor state	StatementHandle에 커서가 열려있고, SQLFetch 또는 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 커서가 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수를 실행하는 중에 관련된 연결이 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY009	Invalid use of null pointer	TableName 인자가 NULL 포인터이다. SQL_ATTR_METADATA_ID 구문 속성이 SQL_TRUE로 설정되고, CatalogName 인자가 null 포인터이며, SQL_CATALOG_NAME InfoType은 카탈로그 이름이 지원됨을 반환한다. SQL_ATTR_METADATA_ID가 SQL_TRUE로 설정되고, SchemaName 인자가 null 포인터이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기 수행 함수가 호출되었으며, SQLStatistics가 호출되었을 때 이 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE가 반환되었다. 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 이 함수가 호출되었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출했고 이 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 혹은 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column 데이터가 전송되기 전에 이 함수가 호출되었다.

SQLSTATE	Error	설명
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY090	Invalid string or buffer length	길이 인자 중 하나의 값이 0보다 작다. (SQL_NTS는 아니다.) 이름 길이 인자 중 하나의 값이 최대 길이를 초과했다.
HY100	Uniqueness option type out of range	유효하지 않은 고유값이 지정되었다.
HY101	Accuracy option type out of range	유효하지 않은 예약어 값이 지정되었다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
HYC00	Optional feature not implemented	명시된 카탈로그를 드라이버나 데이터 소스가 지원하지 않는다. 명시된 스키마를 드라이버나 데이터 소스가 지원하지 않는다. 명시된 문자열 검색 패턴을 카탈로그 이름, 테이블 스키마, 테이블 이름, 데이터 소스 전체 또는 하나 이상에서 지원하지 않는다. SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE의 현재 상태 조합을 드라이버나 데이터 소스에서 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS가 SQL_UB_VARIABLE로 설정되어 있고, SQL_ATTR_CURSOR_TYPE이 드라이버에서 북마크를 지원하지 않는 커서 타입으로 설정된다.
HYT00	Timeout expired	데이터 소스가 요청된 결과 집합 데이터를 반환하기 전에 쿼리 타임아웃 주기가 만료되었다. 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetStmtAttr을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetConnectAttr을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLStatistics는 단일 테이블 정보를 NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION의 순서로 표준 결과 집합으로 반환한다. 결과 집합은 각각의 인덱스에 대한 정보와 테이블 통계정보 (결과 집합의 CARDINALITY와 PAGES row)를 결합한다. 이 정보가 어떻게 쓰이는 지에 대한 자세한 내용은 **카탈로그 데이터 사용**을 참조한다.

응용 프로그램에서는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME row의 실제 길이를 결정하기 위해 SQLGetInfo를 SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, SQL_MAX_COLUMN_NAME_LEN과 함께 호출한다.

노트

ODBC 카탈로그 함수의 일반적인 사용법, 인자, 반환 데이터에 대해 자세한 내용은 **카탈로그 함수**를 참조한다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column의 번호로 바인드하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
SEQ_IN_INDEX	ORDINAL_POSITION
COLLATION	ASC_OR_DESC

다음 표는 결과 집합의 column을 나열한 것이다. Column 13 (FILTER_CONDITION) 이후에 추가되는 column은 드라이버가 정의할 수 있다. 응용 프로그램은 드라이버에서 정의한 column에 접근하기 위해 명시적인 위치를 지정하는 대신 결과 집합의 뒤에서부터 순차적으로 카운트다운 해야한다. 자세한 내용은 **카탈로그 함수의 데이터 반환**을 참조한다.

Row 이름	Row 번호	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	Varchar	카탈로그 이름이다. 데이터 소스를 변환할 수 없으면 NULL이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 카탈로그를 가지지 않는다.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	스키마 이름이다. 데이터 소스에 적용할 수 없다면 NULL이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 스키마를 가지지 않는다.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	통계 또는 인덱스가 적용되는 테이블의 이름이다.

Row 이름	Row 번호	데이터 타입	설명
NON_UNIQUE (ODBC 1.0)	4	Smallint	인덱스가 값의 중복을 허용하지 여부를 가리킨다. - SQL_TRUE: 인덱스 값들이 nonunique일 수 있다. - SQL_FALSE: 인덱스 값들이 반드시 unique여야 한다. TYPE이 SQL_TABLE_STAT일 경우, NULL이 반환된다.
INDEX_QUALIFIER (ODBC 1.0)	5	Varchar	DROP INDEX를 수행하고 인덱스 이름을 규정하는 식별자이다. 인덱스 규정이 지원되지 않는 데이터 소스이거나 TYPE이 SQL_TABLE_STAT인 경우 NULL이 반환된다. 만약 이 row에서 NULL이 아닌 값이 반환될 경우, 이 값은 DROP INDEX에서 식별된 인덱스를 정의하는데 사용된다. 그렇지 않으면 반드시 TABLE_SCHEM을 사용하여 인덱스 이름을 규정하여야 한다.
INDEX_NAME (ODBC 1.0)	6	Varchar	인덱스 이름이다. TYPE이 SQL_TABLE_STAT일 경우, NULL을 반환한다.
TYPE (ODBC 1.0)	7	Smallint not NULL	반환되는 정보의 유형이다. - SQL_TABLE_STAT: (CARDINALITY 또는 PAGES row의) 테이블 통계를 가리킨다. - SQL_INDEX_BTREE: B-tree 인덱스를 가리킨다. - SQL_INDEX_CLUSTERED: 클러스터 인덱스를 가리킨다. - SQL_INDEX_CONTENT: 인덱스의 내용을 가리킨다. - SQL_INDEX_HASHED: 해쉬 인덱스를 가리킨다. - SQL_INDEX_OTHER: 다른 유형의 인덱스를 가리킨다.
ORDINAL_POSITION (ODBC 1.0)	8	Smallint	인덱스의 row 순번이다. (1부터 시작한다.) TYPE이 SQL_TABLE_STAT일 경우, NULL을 반환한다.
COLUMN_NAME (ODBC 1.0)	9	Varchar	Column의 이름이다. Row가 SALARY + BENEFITS와 같은 표현식으로 되어 있다면, 표현식을 반환한다. 만약 표현식을 알 수 없다면, 빈 문자열을 반환한다. TYPE이 SQL_TABLE_STAT일 경우, NULL을 반환한다.
ASC_OR_DESC (ODBC 1.0)	10	Char(1)	Column 정렬 순서이다. A는 오름차순 (ascending)을, D는 내림차순 (descending)을 가리킨다. 데이터 소스에서 row 정렬 순서를 지원하지 않거나 TYPE이 SQL_TABLE_STAT일 경우, NULL을 반환한다.
CARDINALITY (ODBC 1.0)	11	Integer	테이블이나 인덱스의 cardinality 이다. TYPE이 SQL_TABLE_STAT일 경우에는 테이블 row의 개수이다. TYPE이 SQL_TABLE_STAT가 아닐 경우에는 인덱스의 유니크 값의 개수이다. 데이터 소스로부터 값을 사용할 수 없을 경우, NULL을 반환한다.
PAGES (ODBC 1.0)	12	Integer	인덱스나 테이블에 저장한 페이지의 개수이다. TYPE이 SQL_TABLE_STAT일 경우에는 테이블의 페이지 개수이다. TYPE이 SQL_TABLE_STAT이 아닐 경우에는 인덱스의 페이지 개수이다. 데이터 소스 값을 사용할 수 없거나 변환할 수 없을 경우, NULL이 반환된다.
FILTER_CONDITION	13	Varchar	인덱스가 SALARY > 30000과 같이 필터링된 인덱스일 경우에는 필터 조건이다. 만약 필터 상태를 확인할 수 없는 경우에는 빈 문자열이다. 인덱스가 아닐 경우에는 필터링된 인덱스인지 여부를 결정할 수 없거나 TYPE이 SQL_TABLE

Row 이름	Row 번호	데이터 타입	설명
(ODBC 2.0)			_STAT이다.

결과 집합의 row가 테이블에 대응하는 경우, 드라이버는 TYPE을 SQL_TABLE_STAT로, NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION, COLUMN, ASC_OR_DESC를 NULL로 설정한다. 만약 데이터 소스에서 CARDINALITY나 PAGES를 사용할 수 없다면, 드라이버는 그것들을 NULL로 설정한다.

SQLTablePrivileges

적합성

도입된 버전: ODBC 1.0

표준 준수: ODBC

개요

SQLTablePrivileges는 테이블의 목록 및 각 테이블과 관련된 권한들을 반환한다. 드라이버는 명시된 구문에 결과 집합 형식의 정보를 반환한다.

구문

```
SQLRETURN SQLTablePrivileges(
    SQLHSTMT      StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3);
```

인자

StatementHandle

[입력] 검색 결과에 대한 명령어 핸들이다.

CatalogName

[입력] 테이블 카탈로그이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열("")을 반환하며 그 테이블들은 카탈로그를 가지지 않는다. 카탈로그 이름은 문자열 검색 패턴을 포함할 수 없다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, CatalogName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, CatalogName은 통상의 인수 문자열로 취급되며, 대소문자를 구분한다. 자세한 내용은 [카탈로그 함수의 인자](#)를 참조한다.

NameLength1

[입력] *CatalogName의 문자열 길이이다.

SchemaName

[입력] 스키마 이름에 대한 문자열 검색 패턴이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열("")을 반환하며 그 테이블들은 스키마를 가지지 않는다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, SchemaName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정할 경우, SchemaName은 패턴값 문자열 인자로 취급되며, 대소문자를 구분한다.

NameLength2

[입력] *SchemaName의 문자열 길이이다.

TableName

[입력] 테이블 이름 문자열 검색 패턴이다.

SQL_ATTR_METADATA_ID를 SQL_TRUE로 설정할 경우, TableName은 식별자로 취급되고, 대소문자를 구분하지 않는다. 만약 SQL_FALSE로 설정될 경우, TableName은 패턴값 문자열 인자로 취급되며, 대소문자를 구분한다.

NameLength3

[입력] *TableName 문자열의 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	커서가 StatementHandle에 열려 있고 SQLFetch나 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 커서가 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	함수가 실행되는 동안 관련된 연결이 실패하여, 트랜잭션의 상태를 알 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY008	Operation canceled	StatementHandle에 비동기 처리를 사용할 수 있었다. 이 함수가 호출되었고 완료되기 전에 StatementHandle로 SQLCancel이나 SQLCancelHandle이 호출되었다. 그 후 해당 함수가 StatementHandle로 다시 호출되었다.

SQLSTATE	Error	설명
		이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID 구문 속성이 SQL_TRUE로 설정되었고, Catalog Name 인자가 null 포인터이며, SQL_CATALOG_NAME InfoType은 카탈로그 이름의 지원됨을 반환한다. SQL_ATTR_METADATA_ID가 SQL_TRUE로 설정되었고, SchemaName 또는 TableName 인자가 null 포인터이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 SQL 비동기 수행 함수가 호출되었으며, SQLTablePrivileges가 호출되었을 때 이 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect 또는 SQLMoreResults가 호출되었고, SQL_PARAM_DATA_AVAILABLE을 반환했다. 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 해당 함수가 호출되었다. StatementHandle에 대해 비동기 수행 함수가 호출되었으며 이 함수가 호출되었을 때 해당 비동기 수행 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 또는 SQLSetPos가 호출되었고 SQL_NEED_DATA이 반환되었고, 모든 data-at-execution 매개 변수와 column에 대한 데이터가 보내지기 전에 SQLTablePrivileges 함수가 실행되었다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY090	Invalid string or buffer length	길이 인자 중 하나의 값이 0보다 작다. (SQL_NTS는 아니다.) 이름 길이 인자 중 하나의 값이 최대 길이를 초과한다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	유예 상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
HYC00	Optional feature not implemented	명시된 카탈로그를 드라이버나 데이터 소스에서 지원하지 않는다. 명시된 스키마를 드라이버나 데이터 소스에서 지원하지 않는다. 명시된 문자열 검색 패턴을 카탈로그 이름, 테이블 스키마, 테이블 이름, 데이터 소스 전체 또는 하나 이상에서 지원하지 않는다. SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE의 현상태 조합을 드라이버나 데이터 소스에서 지원하지 않는다.

SQLSTATE	Error	설명
		SQL_ATTR_USE_BOOKMARKS가 SQL_UB_VARIABLE로 설정되어 있고, SQL_ATTR_CURSOR_TYPE이 드라이버에서 북마크를 지원하지 않는 커서 타입으로 설정된다.
HYT00	Timeout expired	데이터 소스에서 요청한 결과 집합을 반환하기 전에 쿼리 타임아웃 주기가 만료되었다. 타임 아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetStmtAttr을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetConnectAttr을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 SQLCompleteAsync를 호출해야 한다.

설명

SchemaName과 TableName 인자는 검색 패턴을 허용한다.

유효한 검색 패턴에 대한 자세한 내용은 **패턴값 인자**를 참조한다.

SQLTablePrivileges는 결과를 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, PRIVILEGE, GRANTEE의 순서로 표준 결과 집합으로 반환한다.

응용 프로그램에서 TABLE_CAT, TABLE_SCHEM, TABLE_NAME column의 실제 길이를 확인하기 위해SQLGetInfo 함수를 SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN과 함께 호출한다.

노트

ODBC 카탈로그 함수의 일반적인 사용법, 인자, 반환 데이터에 대해 자세한 내용은 **카탈로그 함수**를 참조한다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column의 번호로 바인드하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

다음 표는 결과 집합의 column을 나열한 것이다. Column 7 (IS_GRANTABLE) 이후에 추가되는 column은 드라이버가 정의할 수 있다. 응용 프로그램은 드라이버에서 정의한 column에 접근하기 위해 명시적인 위치를 지정하는 대신 결과 집합의 뒤에서부터 순차적으로 카운트다운 해야한다. 자세한 내용은 **카탈로그 함수의 데이터 반환**을 참조한다.

Column 이름	Column 번호	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	Varchar	카탈로그 이름이다. 데이터 소스에 적용할 수 없다면 NULL이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 카탈로그를 가지지 않는다.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	스키마 이름이다. 데이터 소스에 적용할 수 없다면 NULL이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 스키마를 가지지 않는다.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	테이블 이름이다.
GRANTOR (ODBC 1.0)	4	Varchar	권한을 부여한 사용자의 이름이다. 데이터 소스에 적용할 수 없다면 NULL이다. GRANTEE column의 값이 개체의 소유자인 모든 row의 경우 GRANTOR column은 _SYSTEM이다.
GRANTEE (ODBC 1.0)	5	Varchar not NULL	권한이 부여된 사용자들의 이름이다.
PRIVILEGE (ODBC 1.0)	6	Varchar not NULL	테이블 권한이다. 다음 중 하나이거나 데이터 소스에서 명시한 권한이다. • SELECT: Grantee가 하나 이상의 테이블 column을 조회할 수 있다. • INSERT: Grantee가 하나 이상의 column에 대한 데이터를 가지고 있는 새 row를 테이블에 추가할 수 있다. • UPDATE: Grantee가 하나 이상의 테이블 column을 갱신할 수 있다. • DELETE: Grantee가 테이블의 데이터 row를 삭제할 수 있다. • REFERENCES: Grantee가 제약 조건 안에서 하나 이상의 테이블 column을 참조할 수 있다. (예: unique, referential, 테이블 제약 조건 확인) 테이블 권한을 부여받은 grantee에게 허용된 활동 범위는 데이터 소스에 따라 다르다. 예를 들어 UPDATE 권한은 grantee가 하나의 데이터 소스 안에서 테이블의 모든 column을 갱신하고 grantor가 다른 데이터 소스에 대한 갱신 권한을 가진 column만 갱신할 수 있도록 한다.
IS_GRANTABLE	7	Varchar	GRANTEE가 다른 사용자에서 권한을 부여할 수 있는지 여부를 나타내며 YES 또는 NO이다. 만약 데이터 소스에 적용할 수 없거나 알수 없는 경우에는 NULL이다.

Column 이름	Column 번호	데이터 타입	설명
(ODBC 1.0)			권한은 부여가능한 것과 부여불가능한 것 중 하나이며 양쪽 다 해당될 수는 없다. SQLColumnPrivileges가 반환하는 결과 집합은 IS_GRANTABLE을 제외한 모든 column에 같은 값을 갖는 두 개의 row를 포함할 수 없다.

SQLTables

적합성

도입된 버전: ODBC 1.0

표준 준수: Open group

개요

SQLTables는 지정된 데이터 소스에 저장된 테이블의 목록, 카탈로그 또는 스키마 이름, 테이블 타입을 반환한다. 드라이버는 결과 집합으로 정보를 반환한다.

구문

```
SQLRETURN SQLTables(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3,
    SQLCHAR *     TableType,
    SQLSMALLINT   NameLength4);
```

인자

StatementHandle

[입력] 검색 결과에 대한 명령문 핸들이다.

CatalogName

[입력] 카탈로그 이름이다. SQL_ODBC_VERSION 환경 속성이 SQL_OV_ODBC3이면, CatalogName 인자는 검색 패턴을 허용한다. 드라이버가 일부 테이블의 카탈로그만 지원하면, 예를 들어서 드라이버가 다른 DBMS로부터 데이터를 검색하는 경우, 빈 문자열 ("")은 카탈로그를 갖지 않는 테이블들을 나타낸다.

SQL_ATTR_METADATA_ID 명령문 속성이 SQL_TRUE로 설정된 경우, CatalogName은 식별자로 취급되고 대소문자를 구분하지 않는다. SQL_FALSE로 설정된 경우, CatalogName은 패턴값 인자이다. 즉 그것은 문자 그대로 처리되고, 대소문자를 구분한다. 자세한 내용은 **카탈로그 함수의 인자**를 참조한다.

NameLength1

[입력] *CatalogName의 문자 길이이다.

SchemaName

[입력] 스키마 이름에 대한 문자열 검색 패턴이다. CatalogName과 같이 빈 문자열 ("")은 스키마를 갖지 않는 테이블을 나타낸다. 또한 CatalogName과 같은 SQL_ATTR_METADATA_ID 명령문 속성에 따라 인자의 처리가 달라진다.

NameLength2

[입력] *SchemaName의 문자 길이이다.

TableName

[입력] 테이블 이름에 대한 문자열 검색 패턴이다. 또한 CatalogName과 같은 SQL_ATTR_METADATA_ID 명령문 속성에 따라 인자의 처리가 달라진다.

NameLength3

[입력] *TableName의 문자 길이이다.

TableType

[입력] 매치되는 테이블 타입의 목록이다.

SQL_ATTR_METADATA_ID 명령문 속성은 TableType 인자에 영향을 미치지 않는다. TableType은 SQL_ATTR_METADATA_ID의 설정과 상관없이 값 리스트 인자이다.

NameLength4

[입력] *TableType의 문자 길이이다.

반환

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

진단

SQLSTATE	Error	설명
01000	General warning	드라이버별 정보 메시지이다. (함수는 SQL_SUCCESS_WITH_INFO를 반환한다.)
08S01	Communication link failure	함수 처리가 완료되기 전에 드라이버와 데이터 소스 간의 연결에 실패했다.
24000	Invalid cursor state	커서가 StatementHandle에 열려있고, SQLFetch 또는 SQLFetchScroll이 호출되었다. SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하면 드라이버가 이 에러를 반환하고 SQLFetch나 SQLFetchScroll이 SQL_NO_DATA를 반환하지 않으면 드라이버 관리자가 이 에러를 반환한다. StatementHandle에 커서가 열려있지만 SQLFetch나 SQLFetchScroll이 호출되지 않았다.

SQLSTATE	Error	설명
40001	Serialization failure	다른 트랜잭션의 자원 데드락 때문에 트랜잭션이 rollback 되었다.
40003	Statement completion unknown	이 함수를 실행하는 중에 관련된 연결이 실패하였고 트랜잭션의 상태를 확인할 수 없다.
HY000	General error	특정 SQLSTATE가 없는 에러이며 *MessageText 버퍼의 SQLGetDiagRec가 반환한 오류 메시지는 오류와 원인을 기술한다.
HY001	Memory allocation error	드라이버가 함수의 실행이나 완료에 필요한 메모리를 할당할 수 없다.
HY008	Operation canceled	StatementHandle에 대한 비동기 처리가 가능하고 이 함수가 호출된 후 완료되기 전에 SQLCancel 또는 SQLCancelHandle 함수가 StatementHandle에 호출되었다. 그리고 이 함수가 StatementHandle에 다시 호출되었다. 이 함수가 호출된 후 완료되기 전에 multi thread 응용 프로그램이 다른 thread로부터 StatementHandle에 SQLCancel 또는 SQLCancelHandle을 호출했다.
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID 구문 속성이 SQL_TRUE로 설정되고, CatalogName 인자가 null 포인터이며, SQL_CATALOG_NAME InfoType은 카탈로그 이름이 지원됨을 반환한다. SQL_ATTR_METADATA_ID가 SQL_TRUE로 설정되고, SchemaName 또는 TableName 인자가 null 포인터이다.
HY010	Function sequence error	StatementHandle과 관련된 연결 핸들에 대해 비동기 수행 함수가 호출되었으며, SQLTables가 호출되었을 때 이 함수가 여전히 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, 또는 SQLMoreResults가 호출되었고 SQL_PARAM_DATA_AVAILABLE가 반환되었다. 스트림된 모든 매개 변수들에 대한 데이터가 회수되기 전에 이 함수가 호출되었다. StatementHandle에 대해 비동기적으로 수행되는 함수를 호출했고 SQLTables 함수가 호출되었을 때 여전히 이 비동기 수행 함수가 실행되고 있었다. StatementHandle에 대해 SQLExecute, SQLExecDirect, SQLBulkOperations 혹은 SQLSetPos가 호출되었고 SQL_NEED_DATA가 반환되었다. 모든 data-at-execution 매개 변수 또는 column에 대한 데이터가 전송되기 전에 이 함수가 호출되었다.
HY013	Memory management error	메모리 부족 상태로 인해 메모리 객체에 접근할 수 없어 함수가 실행되지 못했다.
HY090	Invalid string or buffer length	길이 인자 중 하나의 값이 0보다 작다. (SQL_NTS는 아니다.) 이름 길이 인자 중 하나의 값이 최대 길이를 초과했다.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	유예 상태에 대한 자세한 내용은 SQLEndTran 을 참조한다.
		명시된 카탈로그를 드라이버나 데이터 소스가 지원하지 않는다.

SQLSTATE	Error	설명
HYC00	Optional feature not implemented	명시된 스키마를 드라이버나 데이터 소스가 지원하지 않는다. 명시된 문자열 검색 패턴을 카탈로그 이름, 테이블 스키마, 테이블 이름, 데이터 소스 전체 또는 하나 이상에서 지원하지 않는다. SQL_ATTR_CONCURRENCY와 SQL_ATTR_CURSOR_TYPE의 현재 상태 조합을 드라이버나 데이터 소스에서 지원하지 않는다. SQL_ATTR_USE_BOOKMARKS가 SQL_UB_VARIABLE로 설정되어 있고, SQL_ATTR_CURSOR_TYPE이 드라이버에서 북마크를 지원하지 않는 커서 타입으로 설정된다.
HYT00	Timeout expired	데이터 소스가 요청된 결과 집합 데이터를 반환하기 전에 쿼리 타임아웃 주기가 만료되었다. 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetStmtAttr을 통해 설정할 수 있다.
HYT01	Connection timeout expired	데이터 소스가 요청에 응답하기 전에 연결 타임아웃 주기가 만료되었다. 연결 타임아웃 주기는 SQL_ATTR_CONNECTION_TIMEOUT의 SQLSetConnectAttr을 통해 설정할 수 있다.
IM001	Driver does not support this function	드라이버에서 해당 함수를 지원하지 않는다.
IM017	Polling is disabled in asynchronous notification mode	알림 모델을 사용할 때마다 폴링을 사용할 수 없다.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	핸들에 대한 이전 함수 호출이 SQL_STILL_EXECUTING을 반환하고 알림 모드가 활성화되면, 후 처리와 작업 완료를 위해 핸들에 대해 반드시 SQLCompleteAsync를 호출해야 한다.

설명

SQLTables는 요청된 범위 내의 모든 테이블을 나열한다. 사용자는 이 테이블들에 대한 SELECT 권한을 가지고 있을 수도 있고 가지고 있지 않을 수도 있다. 응용 프로그램은 다음과 같이 접근성을 확인한다.

- SQLGetInfo를 호출하여 SQL_ACCESSIBLE_TABLES 정보 유형을 확인한다.
- SQLTablePrivileges를 호출하여 각 테이블에 대한 권한을 확인한다.

그렇지 않으면, 응용 프로그램은 사용자가 SELECT 권한 없이 select를 수행하는 상황에 대처할 수 있어야 한다.

SchemaName과 TableName 인자는 검색 패턴을 사용할 수 있으며, SQL_ODBC_VERSION이 SQL_OV_ODBC3 일 경우, CatalogName 인자도 검색 패턴을 사용할 수 있다.

유효한 검색 패턴에 대한 자세한 내용은 **패턴값 인자**를 참조한다.

노트

ODBC 카탈로그 함수의 일반적인 사용법, 인자, 반환 데이터에 대해 자세한 내용은 **카탈로그 함수**를 참조한다.

카탈로그, 스키마, 테이블 형식들의 나열을 지원하기 위해, SQLTables의 CatalogName, SchemaName, TableName, TableType 인자들에 대해 다음과 같은 특별한 의미가 정의되어 있다.

- CatalogName이 SQL_ALL_CATALOGS이고 SchemaName과 TableName이 빈 문자열일 경우, 결과 집합은 데이터 소스의 유효한 카탈로그를 포함한다. (TABLE_CAT column을 제외한 모든 column은 NULL을 포함한다.)
- SchemaName이 SQL_ALL_SCHEMAS이고 CatalogName과 TableName이 빈 문자열일 경우, 결과 집합은 데이터 소스의 유효한 스키마를 포함한다. (TABLE_SCHEM을 제외한 모든 column은 NULL을 포함한다.)
- TableType이 SQL_ALL_TABLE_TYPES이고 CatalogName, SchemaName, TableName이 빈 문자열일 경우, 결과 집합은 데이터 소스의 유효한 테이블 유형을 포함한다. (TABLE_TYPE column을 제외한 모든 column은 NULL을 포함한다.)

TableType이 빈 문자열이 아닐 경우, 관련된 유형에 대한 콤마로 구분된 값의 목록을 포함해야 하며, 각각의 값은 'TABLE', 'VIEW'와 같은 형태로 single quote로 둘러 싸거나 TABLE, VIEW처럼 quote가 없어도 된다. 응용 프로그램은 항상 대문자로 테이블 형식을 명시해야 한다. 드라이버는 데이터 소스에서 요청한 형태로 TableType을 변환한다. 데이터 소스가 명시된 테이블 유형을 지원하지 않을 경우, SQLTables는 해당 형식에 대한 데이터 결과를 반환하지 않는다.

SQLTables는 결과를 TABLE_TYPE, TABLE_CAT, TABLE_SCHE, TABLE_NAME의 순서로 표준 결과 집합으로 반환한다. 이 정보가 어떻게 사용되는지에 대한 자세한 내용은 **카탈로그 데이터 사용**을 참조한다.

응용 프로그램에서 TABLE_CAT, TABLE_SCHEM, TABLE_NAME column의 실제 길이를 결정하기 위해 SQLGetInfo를 SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN과 함께 호출한다.

다음 column은 ODBC 3.x에서 이름이 변경되었다. 응용 프로그램은 column 번호로 바인딩하기 때문에 column 이름 변경은 이전 버전과의 호환성에 영향을 미치지 않는다.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

다음 표는 결과 집합의 column을 나열한 것이다. Column 5 (REMARKS) 이후 추가되는 column은 드라이버가 정의할 수 있다. 응용 프로그램은 드라이버에서 정의한 column에 접근하기 위해 명시적인 위치를 지정하는 대신 결과 집합의 뒤에서부터 순차적으로 카운트다운 해야 한다.

자세한 내용은 **카탈로그 함수의 데이터 반환**을 참조한다.

Column 이름	Column 번호	데이터 타입	설명
TABLE_CAT (ODBC 1.0)	1	Varchar	카탈로그 이름이다. 데이터 소스에 적용할 수 없을 경우, NULL이다. 드라이버가 카탈로그를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 카탈로그를 갖지 않는다.
TABLE_SCHEMA (ODBC 1.0)	2	Varchar	스키마 이름이다. 데이터 소스에 적용할 수 없을 경우, NULL이다. 드라이버가 스키마를 지원하지 않을 경우 빈 문자열 ("")을 반환하며 그 테이블들은 스키마를 갖지 않는다.
TABLE_NAME (ODBC 1.0)	3	Varchar	테이블 이름이다.
TABLE_TYPE (ODBC 1.0)	4	Varchar	테이블 형식 이름이다. "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM" 또는 데이터 소스에서 명시한 이름 중 하나이다. "ALIAS"와 "SYNONYM"의 의미는 드라이버별로 다르다.
REMARKS (ODBC 1.0)	5	Varchar	테이블에 대한 설명이다.

SQLTransact

적합성

도입된 버전: ODBC 1.0

표준 준수: 없음

개요

ODBC 2.x의 SQLTransact 함수는 ODBC 3.x에서 SQLEndTran 함수로 대체되었다.
자세한 내용은 **SQLEndTran**을 참조한다.

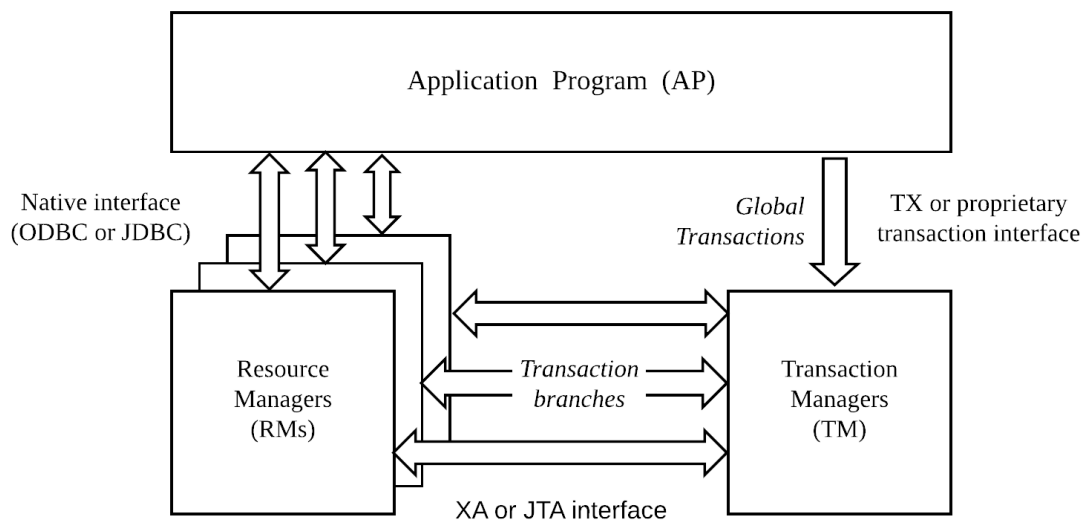
25.6 XA API References

개요

X/Open XA는 X/Open이 제정한 분산 트랜잭션을 처리하기 위한 표준이다. 보통 멀티 트랜잭션 관리자와 로컬 리소스 관리자 사이의 인터페이스를 규정하고 있다. XA에서는 리소스 관리자가 트랜잭션을 처리하기 위해 필요한 사항을 규정하고 있다.

GOLDILOCKS XA는 X/Open CAE 문서 Distributed Transaction Processing: The XA Specification (<http://www.opengroup.org/public/catalog/c193.htm>)을 기반으로 구현되었다.

X/Open Distributed Transaction Processing (DTP) 모델은 다른 컴퓨터의 기종이 다른 데이터베이스 사이의 트랜잭션 관리를 정의하고 있다.



- Application Program (AP): 트랜잭션으로 구성된 작업을 정의한다.
- Resource Managers (RM): 분산 트랜잭션이 접근하는 공유 자원을 관리한다. GOLDILOCKS와 같은 데이터베이스 관리 시스템을 의미한다.
- Transaction Manager (TM): 분산 트랜잭션의 아이디 (XID) 할당하고, 분산 트랜잭션의 진행을 관리하며 분산 트랜잭션의 종료와 회복에 대한 책임을 가진다.

AP는 precompiler나 ODBC를 사용하여 개발한 프로그램일 수 있다. AP는 직접 XA 인터페이스를 사용하지 않고, RMs의 네이티브 인터페이스 (native interface)를 사용하거나 TM에서 제공하고 있는 TX 인터페이스를 사용하여 트랜잭션을 제어한다.

XA 인터페이스

XA 인터페이스는 RM과 TM간의 인터페이스 규정이다. GOLDDILOCKS는 XA 인터페이스를 위한 별도의 라이브러리를 제공하지 않고, GOLDDILOCKS에서 제공하는 ODBC 라이브러리 안에 포함되어 있다.

switch_t 구조체

XA 인터페이스들에 대한 진입점 (entry point)과 RM에 대한 정보를 가지고 있는 구조체이다. GOLDDILOCKS에서 제공하는 xa_switch_t는 goldilocks_xa_switch이다.

변수 이름	설명
char name[RMNAMESZ]	RM의 이름이다.
long flags	RM에서 제공하는 옵션이다. MIGRATE 옵션은 지원하지 않는다. (TMNOMIGRATE로 설정된다.)
long version	RM 버전이다.
int (*xa_open_entry)(char *, int, long);	xa_open 함수 포인터이다.
int (*xa_close_entry)(char *, int, long);	xa_close 함수 포인터이다.
int (*xa_start_entry)(XID *, int, long);	xa_start 함수 포인터이다.
int (*xa_end_entry)(XID *, int, long);	xa_end 함수 포인터이다.
int (*xa_rollback_entry)(XID *, int, long);	xa_rollback 함수 포인터이다.
int (*xa_prepare_entry)(XID *, int, long);	xa_prepare 함수 포인터이다.
int (*xa_commit_entry)(XID *, int, long);	xa_commit 함수 포인터이다.
xint (*xa_recover_entry)(XID *, long, int, long);	xa_recover 함수 포인터이다.
int (*xa_forget_entry)(XID *, int, long);	xa_forget 함수 포인터이다.
int (*xa_complete_entry)(int *, int *, int, long);	xa_complete 함수 포인터이다. GOLDDILOCKS에서는 제공하지 않는다.

XA 관련 ODBC 함수

표준 ODBC 외에 XA 인터페이스를 사용하기 위해서 추가된 함수를 설명한다.

SQLGetXaSwitch

ODBC 함수에서 제공하는 xa_switch_t를 얻는다.

```
xa_switch_t * SQLGetXaSwitch( void );
```

반환

ODBC에서 제공하는 sw_switch_t 구조체의 포인터를 반환한다. NULL을 반환할 수는 없다.

SQLGetXaConnectionHandle

현재 XA 세션과 관련된 연결 핸들 (connection handle)을 얻는다.

```
SQLHANDLE SQLGetXaConnectionHandle( void );
```

반환

해당 thread에 연결된 XA 세션이 있을 경우, 이와 관련된 연결 핸들을 반환하고, 그렇지 않으면 NULL을 반환한다.

XA 함수

xa_switch_t 구조체의 XA 관련 함수들에 대한 자세한 내용을 기술한다.

xa_open

RM에 접속한다.

노트

xa_open을 호출한 thread에서 이미 접속되어 있을 경우, 이를 무시한다.

```
int xa_open(
    char * xa_info,
    int    rmid,
    long   flags );
```

인자

xa_info

[입력] 접속 정보를 포함하는 문자열이며, 최대 길이는 256 바이트이다. 자세한 내용은 의 InConnectionString을 참조한다.

rmid

[입력] 접속할 RM의 고유한 아이디이다. 해당 인자는 무시된다.

flags

[입력] 접속 플래그이다. TMNOFLAGS로 설정되어야 한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	RM을 사용할 수 없는 상태일 때 발생한다.
XAER_RMERR	리소스 부족등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XAER_INVAL	비정상적인 인자를 전달한 경우에 발생한다.

xa_close

RM의 연결을 종료하고, 연결 핸들을 해제한다.

```
int xa_close(  
    char * xa_info,  
    int    rmid,  
    long   flags );
```

인자

xa_info

[입력] 접속 정보를 포함하는 문자열이며 해당 인자는 무시된다.

rmid

[입력] 접속할 RM의 고유한 아이디이다. 해당 인자는 무시된다.

flags

[입력] 종료 플래그이다. 해당 인자는 무시된다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다
XAER_RMFAIL	RM을 사용할 수 없는 상태일 때 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XAER_INVAL	비정상적인 인자를 전달한 경우에 발생한다.

xa_start

새로운 트랜잭션 브랜치를 생성하거나 기존 트랜잭션 브랜치를 시작한다.

```
int xa_start(
    XID * xid,
    int rmid,
    long flags );
```

인자

xid

[입력] 시작할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] 시작 플래그이다.

flags는 다음 값들을 사용할 수 있다.

- TMASYNC: 비동기 모드로 트랜잭션 브랜치를 시작한다. (지원하지 않는다)
- TMNOFLAGS: FLAG를 사용하지 않는다는 것을 의미하며, 어떠한 플래그도 사용하지 않을 경우, 해당 플래그를 지정해야 한다.
- TMNOWAIT: 지정한 트랜잭션 브랜치를 이미 다른 세션에서 사용하고 있는 중이라면 기다리지 않고 XA_RETRY 에러를 반환하도록 한다.
- TMRESUME: 이전에 작업을 중단한 트랜잭션 브랜치를 계속 진행한다. TMJOIN과 같이 사용할 수 없다.
- TMJOIN: 기존 트랜잭션 브랜치에 연결한다. TMRESUME과 함께 사용할 수 없다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XAER_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.
XAER_INVALID	비정상적인 인자를 전달한 경우에 발생한다.
XAER_DUPID	동일한 XID를 갖는 트랜잭션 브랜치가 이미 존재한다.
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다. TMRESUME이나 TMJOIN을 사용한 경우에 발생할 수 있다.
XA_RETRY	지정한 트랜잭션 브랜치를 이미 다른 세션에서 사용 중인 경우에 발생한다. TMNOWAIT를 사용한 경우에 발생할 수 있다.
XAER_OUTSIDE	로컬 트랜잭션이 이미 해당 세션에서 진행되고 있다.

반환값	설명
XA_RBROLLBACK	트랜잭션 브랜치가 rollback-only로 설정되어 있다.

xa_end

트랜잭션 브랜치의 작업을 종료한다.

```
int xa_end(
    XID * xid,
    int rmid,
    long flags );
```

인자

xid

[입력] 작업을 종료할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] 작업 종료 플래그이다.

flags는 다음 값들을 사용할 수 있다.

- TMFAIL: 작업이 실패했음을 가리킨다. TMSUSPEND나 TMSUCCESS와 함께 사용할 수 없으며 트랜잭션 브랜치의 상태를 rollback-only로 변경한다.
- TMMIGRATE: 다른 브랜치와 연계하여 다시 시작한다. (지원하지 않는다.)
- TMSUCCESS: 작업이 성공적으로 종료되었음을 가리킨다. TMSUSPEND나 TMFAIL과 함께 사용할 수 없다.
- TMSUSPEND: 트랜잭션 브랜치를 잠시 멈추고 종료한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XA_NOMIGRATE	MIGRATE 플래그를 지원하지 않는다.
XAER_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.
XAER_INVAL	비정상적인 인자를 전달한 경우에 발생한다.
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다.

xa_prepare

XID에 해당하는 트랜잭션을 commit할 준비를 한다. Two-phase Commit Protocol (2PC)의 첫 번째 단계이다.

```
int xa_prepare(  
    XID * xid,  
    int rmid,  
    long flags );
```

인자

xid

[입력] Prepare할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] Prepare 플래그이다. TMNOFLAGS가 설정되어야 한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XA_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.
XA_RDONLY	트랜잭션 브랜치가 read-only 트랜잭션이다.
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다.
XA_RBROLLBACK	트랜잭션 브랜치가 rollback-only로 설정되어 있다.

xa_commit

XID에 해당하는 트랜잭션을 commit 한다. Two-phase Commit Protocol (2PC)의 두 번째 단계이다.

```
int xa_commit(
    XID * xid,
    int rmid,
    long flags );
```

인자

xid

[입력] Commit할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] Commit 플래그이다.

flags는 다음 값들을 사용할 수 있다.

- TMNOFLAGS: FLAG를 사용하지 않는다는 것을 의미하며, 어떠한 플래그도 사용하지 않을 경우, 해당 플래그를 지정해야 한다.
- TMONEPHASE: One Phase Commit (1PC)를 수행한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XA_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.
XA_RDONLY	트랜잭션 브랜치가 read-only 트랜잭션이다.
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다.
XA_RBROLLBACK	트랜잭션 브랜치가 rollback-only로 설정되어 있다.

xa_rollback

XID에 해당하는 트랜잭션을 rollback 한다.

```
int xa_rollback(  
    XID * xid,  
    int rmid,  
    long flags );
```

인자

xid

[입력] Rollback 할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] Rollback 플래그이다. TMNOFLAGS가 설정되어야 한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XA_HEURRB	트랜잭션 브랜치가 이미 heuristic rollback 되었다
XA_HEURCOM	트랜잭션 브랜치가 이미 heuristic commit 되었다
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다.

xa_recover

Heuristic commit이나 heuristic rollback 한 트랜잭션들의 목록을 얻는다.

```
int xa_recover(
    XID * xids,
    long  count,
    int   rmid,
    long  flags );
```

인자

xids

[출력] Heuristic commit이나 heuristic rollback 한 트랜잭션들의 목록이다.

count

[입력] xids의 배열 크기를 의미한다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] recover 플래그이다.

flags는 다음 값들을 사용할 수 있다.

- TMSTARTSCAN: 스캔을 시작하거나, 처음부터 다시 시작한다.
- TMENDSCAN: 스캔을 종료한다.
- TMNOFLAGS: TMSTARTSCAN 이후에 사용할 경우 다음 목록을 얻는다. 만약 처음부터 TMNOFLAGS를 사용할 경우, XA_PROTO 에러가 발생한다.

진단

반환값	설명
>= 0	반환되는 XID들의 개수이다. (xids에서 유효한 배열의 크기이다.)
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XA_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.

xa_forget

Heuristic commit이나 heuristic rollback 한 트랜잭션들의 정보를 삭제한다.

```
int xa_forget(
    XID * xid,
    int rmid,
    long flags );
```

인자

xid

[입력] 삭제할 트랜잭션 ID이다.

rmid

[입력] 접속할 RM의 고유한 ID이다. 해당 인자는 무시된다.

flags

[입력] 삭제 플래그이다. TMNOFLAGS가 설정되어야 한다.

진단

반환값	설명
XA_OK	연산이 정상적으로 수행되었다.
XAER_RMFAIL	사용 중이던 세션이 비정상적으로 종료된 경우에 발생한다.
XAER_RMERR	리소스 부족 등으로 인해 RM에서 트랜잭션 브랜치에 연산을 수행할 수 없다.
XAER_PROTO	XA 프로토콜에 맞지 않는 순서로 실행된 경우에 발생한다.
XAER_INVALID	비정상적인 인자를 전달한 경우에 발생한다.
XAER_NOTA	XID로 지정된 트랜잭션 브랜치가 존재하지 않는다.

사용 예

다음은 GOLDILOCKS에 접속하여 레코드 추가/ 검색/ 갱신/ 삭제 후에 Two-phase Commit 하는 간단한 예이다. 다음 예와 관련된 완전한 코드는 \$GOLDILOCKS_HOME/sample/ODBC/SAMPLE_XA.c에 있다.

1. SQLGetXaSwitch()

XA 인터베이스를 사용하기 위한 xa_switch_t 구조체를 얻는다.

2. SQLAllocHandle()

ODBC 환경 핸들을 얻는다.

3. xa_open()

주어진 connection string으로 서버에 접속한다.

이미 서버와 연결되어 있다면 기존 연결을 사용하고, 그렇지 않다면 새로운 연결을 시도한다.

4. SQLGetXaConnectionHandle()

해당 thread와 관련된 XA 연결 핸들을 얻는다.

5. xa_start()

트랜잭션 브랜치를 시작한다.

6. 표준 ODBC 함수를 이용한 트랜잭션 작업을 수행한다.

7. xa_end()

트랜잭션 작업을 종료한다.

8. xa_prepare()

2PC의 첫 번째 단계로써 트랜잭션을 commit 하기 위해 준비한다.

9. xa_commit()

2PC의 두 번째 단계로써 트랜잭션을 commit 한다.

10. xa_close()

연결을 종료하고, 연결 핸들을 해제한다.

ODBC 표준 함수인 SQLDisconnect()와 SQLFreeHandle()을 사용해도 무방하다.

11. SQLFreeHandle()

ODBC 환경 핸들을 해제한다.

```
int main( int aArgc, char** aArgv )
{
    SQLHENV      sEnv      = NULL;
    SQLHDBC      sDbc      = NULL;
    SQLINTEGER    sState    = 0;
    xa_switch_t * sXaSwitch;
    XID           sXid;
    sXaSwitch = SQLGetXaSwitch();
```

- GOLDILOCKS ODBC에 포함된 SQLAllocEnv()를 호출할 경우

```
GOLDILOCKS_SQL_TRY( SQLAllocHandle( SQL_HANDLE_ENV,
                                     NULL,
                                     &sEnv ) );

    sState = 1;
```

- SQLSetEnvAttr이 환경을 관리하는 속성을 설정한다.

```

GOLDILOCKS_SQL_TRY( SQLSetEnvAttr( sEnv,
                                   SQL_ATTR_ODBC_VERSION,
                                   (SQLPOINTER)SQL_OV_ODBC3,
                                   0 ) );

if( (sXaSwitch->xa_open_entry)(
    "DSN=GOLDILOCKS;UID=test;PWD=test",
    0,
    TMNOFLAGS ) != XA_OK )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}

sState = 2;
sDbc = SQLGetXaConnectionHandle();
sXid.formatID = 0;
sXid.gtrid_length = 2;
sXid.bqual_length = 1;
memcpy( sXid.data,
        "100",
        sXid.gtrid_length + sXid.bqual_length );

if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}

```

- 함수를 추가하는 SQL_SUCCESS가 성공적으로 수행된 경우

```
GOLDILOCKS_SQL_TRY( testInsert( sDbc ) );
```

- 함수를 선택하는 SQL_SUCCESS가 성공적으로 수행된 경우

```
GOLDILOCKS_SQL_TRY( testSelect( sDbc ) );
```

- 함수를 갱신하는 SQL_SUCCESS가 성공적으로 수행된 경우

```
GOLDILOCKS_SQL_TRY( testUpdate( sDbc ) );
```

- 함수를 삭제하는 SQL_SUCCESS가 성공적으로 수행된 경우

```
GOLDILOCKS_SQL_TRY( testDelete( sDbc ) );
```

```
if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
```

```

{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}

if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}

if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}

sState = 1;
if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}

sDbc = NULL;

```

- SQLFreeHandleEnv이 환경과 관련된 자원을 해제한다.

```

sState = 0;
GOLDDILOCKS_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                     sEnv ) );

sEnv = NULL;
return EXIT_SUCCESS;
GOLDDILOCKS_FINISH;
if( sDbc != NULL )
{
    PrintDiagnosticRecord( SQL_HANDLE_DBC, sDbc );
}
if( sEnv != NULL )
{
    PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
}
switch( sState )
{

```

- 예제 2: SQLDisconnect가 특정 연결 핸들과 관련된 연결을 닫는다.


```
(void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
```

- 예제 1: SQLFreeHandleEnv가 환경과 관련된 자원을 해제한다.

```
(void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );  
    sEnv = NULL;  
    default:  
        break;  
}  
return EXIT_FAILURE;  
}
```


26.

JDBC

26.1 GOLDDILOCKS JDBC 개요

GOLDDILOCKS JDBC 소개

GOLDDILOCKS는 TCP/IP 연결을 기반으로 한 표준 JDBC 4.0을 준수 (일부 기능 제외)하는 GOLDDILOCKS JDBC 드라이버를 제공한다. 사용자는 GOLDDILOCKS JDBC 드라이버를 이용하여 Java 프로그램 내에서 GOLDDILOCKS와 접속하여 각종 트랜잭션 기능 및 데이터 조회 기능을 사용할 수 있다. GOLDDILOCKS JDBC 드라이버는 JDK 1.6 기반으로 작성되고 구축되었기 때문에 JDBC 4.0의 기능을 지원한다. \$GOLDDILOCKS_HOME/lib/golddilocks6.jar 파일을 class path에 추가하여 드라이버를 사용할 수 있다.

뿐만 아니라 하위 버전의 Java (JDK 1.4, JDK 1.5)를 사용하는 사용자를 위해 JDBC 3.0을 구현한 golddilocks4, golddilocks5.jar도 함께 지원한다. golddilocks 다음의 숫자는 JDK 버전을 의미한다. 자세한 내용은 **버전 및 지원 여부**를 참조한다.

GOLDDILOCKS JDBC는 JDBC 표준 스펙을 대부분 준수하고 있으며 일부 고유한 기능을 제공하기 위해 표준 API가 아닌 method와 클래스도 지원한다. 비표준 method에 대해서는 **JDBC API References**의 각 클래스 API나 **추가 타입 사용**을 참조한다.

특징

- Type-4 JDBC 드라이버

GOLDDILOCKS JDBC는 순수 Java로만 구현된 JDBC Type-4 유형이다. 추가적인 라이브러리 없이 jar 파일만으로 JDBC 드라이버를 사용할 수 있다. 뿐만 아니라 JDBC-ODBC bridge 유형보다 빠르고 안정적이며 native API를 사용한 Type-2보다 이식성 (portability)이 좋다.

- JDBC 표준 준수

GOLDDILOCKS JDBC는 JDBC 표준을 준수하므로 여타의 JDBC 프로그램을 거의 변경하지 않고 재활용할 수 있다. Connection을 얻는 방법부터 각종 statement, ResultSet 기능들을 수정없이 사용할 수 있다. 단, connection URL과 프로퍼티 이름, 로딩할 드라이버 클래스 이름은 GOLDDILOCKS에 맞게 변경해야 한다. 그리고 표준에 없는 기능과 타입들은 별도의 클래스와 method API로 사용할 수 있다.

- 다양한 Java 버전 지원

GOLDDILOCKS JDBC 드라이버는 네 개의 파일 (golddilocks7.jar, golddilocks6.jar, golddilocks5.jar, golddilocks4.jar)를 지원하므로 사용자는 Java 실행 환경에 맞는 파일을 선택하여 사용할 수 있다. 각각의 jar 파일은 JDK 1.7, JDK 1.6, JDK 1.5, JDK 1.4를 기반으로 제작, 구축되었기 때문에 JDBC 4.1, JDBC 4.0, JDBC 3.0, JDBC 3.0 스펙을 따른다.

- 서버와의 유연한 버전 호환성

서버와 JDBC 드라이버와의 호환성은 프로토콜 버전을 참조한다. 서버가 JDBC 드라이버보다 프로토콜 버전이 높을 경우에 접속할 수 있다.

- Connection pooling을 위한 API 지원

JDBC connection 객체는 생성 비용이 비싼 자원이다. 따라서 JDBC 표준에는 이를 풀링할 수 있는 체계를 정의했으며 그것을 인터페이스로 만든 것이 ConnectionPoolDataSource와 PooledConnection이다. GOLDILOCKS JDBC는 이 인터페이스를 구현하여 제3자의 미들웨어 제품에서 풀링 기능을 사용할 수 있도록 하는 기반을 제공한다.

- XA API 지원

글로벌 트랜잭션 표준인 XA 인터페이스를 구현하여 사용자가 글로벌 트랜잭션 작업을 할 수 있도록 한다. JDBC 인터페이스인 XAResource를 사용하여 각종 XA 기능을 표준에 맞게 수행할 수 있다.

- GOLDILOCKS 고유 데이터 타입

GOLDILOCKS는 JDBC 표준에서 제공하지 않는 데이터 타입을 사용한다. 예를 들어 interval 관련 타입과 timestamp with time zone과 같이 time zone 정보를 가지는 타입들이 있다. 이들 타입들을 데이터베이스에 삽입하거나 가져올 수 있는 방법들을 드라이버가 제공한다.

- 서버 기반의 강력한 커서 스크롤 기능

타 JDBC 드라이버들은 ResultSet 스크롤을 위해 row set을 드라이버 내에서 캐싱하는 방법을 취하여, client 응용 프로그램이 과도하게 메모리를 많이 사용하는 경우가 있다. 예를 들어 scroll insensitive로 커서를 열어 마지막 row까지 fetch하게 되면 모든 row들을 드라이버 내에 캐싱하게 되는데 이 때 전체 테이블이 client 메모리에 상주하게 되므로 out of memory 에러가 발생할 수도 있고, 과도하게 client 메모리 리소스를 소모하게 된다.

하지만 GOLDILOCKS는 서버 내에 커서 스크롤 기능이 있기 때문에 client가 가볍고, 빠르고 안정적인 성능을 낼 수 있다.

- 효율적인 자원 사용

타 JDBC 드라이버들에 비해 row 정보를 유지하는 메모리를 최소화하고, 되도록 Java 객체를 사용하지 않기 때문에 많은 row들을 탐색할 경우에도 garbage collection 작업을 피할 수 있으며 적은 메모리를 사용하도록 하여 빠르고 안정적으로 테이블을 스캔할 수 있다.

- 정확하고 방대한 메타 데이터

JDBC 표준의 DatabaseMetaData 인터페이스를 충실하고 정확하게 구현했기 때문에 다양한 데이터베이스 툴과 쉽게 연계할 수 있다. 뿐만 아니라 각종 시스템 view를 통해 데이터베이스 메타 정보들을 조회할 수 있어 사용성이 높아졌다.

- 강력한 로깅 기능

문제가 발생했을 때 뿐만 아니라 평상시에도 JDBC API 콜이나 네트워크 사용 상황을 모니터링 할 수 있도록 각종 logging 기능을 제공한다. Connection URL에 로깅 관련 기능을 명시하면 콘솔이나 파일 등으로 해당 내용의 기록을 남길 수 있다. 로깅에는 JDBC method call 기록에 대한 로깅, 프로토콜 송수신에 대한 로깅, 사용한 SQL 문에 대한 로깅, 이 세 가지가 있다.

- Connection failover

GOLDILOCKS 서버와의 연결에 실패하거나 execution 할 때 연결이 끊어지면 미리 등록된 대체 서버에 자동으로 다시 접속하여 기존 connection을 계속해서 사용할 수 있도록 JDBC driver 레벨의 connection failover를 지원한다. 사용자는 connection이 끊겼을 때를 대비한 별도의 예외 처리없이 기존의 JDBC 프로그램으로 connection failover 기능을 사용할 수 있다.

- 서버와 direct attach 연결 가능

TCP/ IP 기반의 연결 이외에 서버와 같은 프로세스로 연동할 수 있는 Direct Attach 방식의 연결도 할 수 있다. ODBC 연결이 Direct Attach, Client/ Server 방식을 모두 지원하듯이 GOLDILOCKS JDBC driver도 두 가지 연결 방법을 모두 제공한다. Direct attach로 연결된 JDBC 프로그램은 TCP/ IP 통신을 하지 않고 직접 서버 프로세스에 연동되어 jvm 안에서 서버 기능을 사용할 수 있다. TCP/ IP로 연결한 JDBC 프로그램보다 두 배 이상의 성능을 낼 수 있다.

버전 및 지원 여부

GOLDILOCKS JDBC 버전 체계

goldilocks6.jar 파일에 대해 다음과 같이 실행하면 GOLDILOCKS JDBC 버전 정보를 볼 수 있다.

```
shell>java -jar goldilocks6.jar
GOLDILOCKS JDBC Driver 1.1 Protocol-1.3.1, JDBC4.0 compiled with JDK1.6
```

위 예제는 현재 GOLDILOCKS JDBC의 드라이버 버전이 1.0이고, 프로토콜 버전은 1.3.1, 그리고 이 드라이버가 준수하는 JDBC 표준 버전은 4.0이며 JDK 1.6에서 구축되었다는 정보를 담고 있다. 드라이버 버전은 GOLDILOCKS 제품 버전과는 별개인 드라이버의 버전을 나타내고 기능이 보강될 때마다 올라간다. JDBC 드라이버 버전에 대한 자세한 내용은 DatabaseMetaData의 **getDriverMajorVersion**, **getDriverMinorVersion**, **getDriverVersion**을 참조한다.

프로토콜 버전은 서버와의 호환성 여부를 결정하며 드라이버는 서버의 프로토콜 버전과 같거나 더 낮을 때 서버에 연동될 수 있다. 즉, 서버는 하위 프로토콜 버전의 모든 client API를 지원한다.

goldilocks6.jar는 JDBC 4.0 표준을 준수하며 JDK 1.6에서 구축되었다. 따라서 사용자 Java 환경이 JDK 1.6이상일 경우에는 goldilocks6.jar를 사용하면 된다. 만일 JDK 1.7 이상의 Java를 사용할 경우, goldilocks6.jar는 사용할 수 있지만 JDBC 5.0의 API나 클래스들은 사용할 수 없다.

JDK 1.5나 1.4 환경의 Java를 사용할 경우, goldilocks5.jar나 goldilocks4.jar를 사용해야 한다. 이 둘은 모두 JDBC 3.0 표준을 준수한다.

사용 예

Class Path 설정

GOLDILOCKS JDBC 드라이버를 사용하려면 CLASSPATH를 설정해야 한다.

```
export CLASSPATH=.:$GOLDILOCKS_HOME/lib/goldilocks6.jar
```

또는 사용자의 Java 실행환경에 맞는 jar 파일을 path에 추가해도 된다.

드라이버 클래스 로딩

드라이버 클래스는 다음과 같이 로딩할 수 있다.

```
Class.forName("sunje.goldilocks.jdbc.GoldilocksDriver");
```

위 예는 전통적인 JDBC 사용 방법으로써 해당 드라이버 클래스를 동적으로 로딩하여 DriverManager에 등록하게 한 다음 연결을 얻는다. 요즘은 DataSource를 통해 연결을 얻는 방법을 더 많이 사용하는데 이에 대해서는 **JDBC API References**의 해당 클래스를 참조한다.

Connection 얻기

Connection을 얻으려면 다음과 같이 코드를 작성한다.

```
Connection con = DriverManager.getConnection(
    "jdbc:goldilocks://127.0.0.1:22581/test", "TEST", "test");
```

GOLDILOCKS JDBC를 사용하기 위한 접속 URL은 "jdbc:goldilocks:"로 시작해야 한다. 그 다음은 서버의 IP와 port 번호이며 URL의 마지막 부분 "/test"는 DB 이름이다. 현재 GOLDILOCKS는 다중 DB를 지원하지 않으므로 DB 이름을 특별히 체크하지는 않는다. 사용자가 접속한 URL은 DatabaseMetaData.getURL()을 통해서 다시 얻을 수 있다.

IP를 0.0.0.0으로, 포트를 0으로 설정하면 Direct Attach (D/A) 모드로 접속한다. 혹은 ip:port 대신 da라는 접속 프로토콜을 사용해도 된다. 즉 다음 두 URL 모두 D/A 모드로 접속하게 한다.

```
"jdbc:goldilocks://0.0.0.0:0/test"
"jdbc:goldilocks:da/test"
```

Username과 password는 각각 데이터베이스의 계정과 암호를 사용해야 한다.

Statement와 ResultSet 사용

Statement와 ResultSet은 다른 JDBC 프로그램과 동일하게 사용하면 된다.

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NAME, ADDRESS FROM EMP");
while (rs.next())
{
    System.out.println("name = " + rs.getString(1));
    System.out.println("address = " + rs.getString(2));
}
rs.close();
stmt.close();
```


26.2 기능 명세

연결

DriverManager를 이용한 연결

Connection을 얻는 전통적인 방법은 DriverManager를 사용하는 것이다.

```
Connection con = DriverManager.getConnection(url_string, user_name, password);
```

url_string에는 다음과 같은 세가지 유형이 있다.

```
jdbc:goldilocks://[ip address]:[port_no]/[db_name]
```

```
jdbc:goldilocks:da/[db_name]
```

```
jdbc:goldilocks:locator//[ip address]:[port_no][, [ip address]:[port_no] ]*/[db_name]
```

IP address에는 IPv4, IPv6 모두 사용 가능하며, port_no에는 설정된 포트 번호를 명시하면 된다. db_name은 현재 접속에 사용되지 않기 때문에 어떤 이름이든 설정할 수 있지만 생략할 수는 없다. 다음은 예제 URL이다.

```
String url_string = "jdbc:goldilocks://127.0.0.1:22581/test";
```

IP "0.0.0.0"과 포트 0은 D/A 접속을 위한 특별한 주소로 사용된다. D/A 모드에 대한 자세한 내용은 **Direct Attach 모드 접속**을 참조한다.

user_name과 password는 GOLDILOCKS 계정을 의미한다.

위의 getConnection() method 외에 각종 property를 설정하기 위해 다음 method를 사용한다.

```
Properties prop = new Properties();
prop.setProperty("user", user_name);
prop.setProperty("password", password);
Connection con = DriverManager.getConnection(url_string, prop);
```

glocator로부터 서버의 접속 정보를 얻기 위해 연결 프로퍼티를 대신하여 url_string에 locator 키워드를 추가하여 사용할 수 있다. 이 때 ip address와 port_no는 glocator의 접속 정보로 간주한다. 다음은 연결 구문에 locator 키워드를 사용하는 예이다.

```
String url_string = "jdbc:goldilocks:locator//127.0.0.1:42581,127.0.0.1:42582/test?locator_
service=S1";
```

처음 접속 정보 127.0.0.1:42581은 glocator의 접속 정보이고 이후의 접속 정보는 127.0.0.1:42582처럼ALTERNATE_LOCATORS 프로퍼티로 처리된다.

다음은 위의 url_string과 같은 의미의 프로퍼티 설정이다.

```
String url_strng = "jdbc:goldilocks://0.0.0.0:0/test";
Properties prop = new Properties();
prop.setProperty("locator_host", "127.0.0.1");
prop.setProperty("locator_port", "42581");
prop.setProperty("alternate_locators", "127.0.0.1:42582");
prop.setProperty("locator_service", "S1");
Connection con = DriverManager.getConnection(url_string, prop);
```

접속에 사용할 수 있는 프로퍼티 목록은 GOLDILOCKSDriver의 getPropertyInfo() method를 통해 확인할 수 있다. 자세한 내용은 표 26-1 연결 프로퍼티를 참조한다.

DriverManager를 사용하면 생성되는 connection 객체의 login timeout과 logger는 DriverManager에 등록된 값을 사용한다. 즉, 이 connection 객체로부터 생성되는 모든 JDBC 인터페이스들이 사용하는 logger는 저 logger이다. 따라서 connection 객체마다 개별 logger를 가질 수 없다.

DataSource를 이용한 연결

JDBC 표준은 과거의 DriverManager보다는 DataSource를 사용할 것을 더 권장한다. DataSource는 어떤 data source에 접근할 수 있도록 하는 단일한 인터페이스이며, 각종 data source에 관계되는 설정 값들을 개별적으로 설정할 수 있으며, DataSource 객체 자체를 원격으로 전송할 수도 있기 때문이다.

다음과 같이 DataSource를 통해 connection 객체를 얻을 수 있다.

```
import sunje.goldilocks.jdbc.GoldilocksDataSource;
GoldilocksDataSource ds = new GoldilocksDataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
Connection con = ds.getConnection();
```

이처럼 DataSource 객체를 생성하기 위해서는 GoldilocksDataSource 클래스를 사용해야 한다.

그 다음 각종 연결 정보는 DataSource 표준 API가 아닌 setter method를 사용해야 하는데, 자세한 내용은 DataSource를 참조한다.

DriverManager를 이용할 때는 login timeout과 logger를 글로벌하게 설정했어야 했지만, DataSource는 login timeout과 logger를 개별적으로 설정할 수 있다.

```
ds.setLoginTimeout(10);
ds.setLogWriter(out);
```

미들웨어 연동

Weblogic, JBoss와 같은 미들웨어와 연동하려면 XADataSource, ConnectionPoolDataSource 인터페이스를 구현한 클래스 이름을 알아야 한다. GOLDILOCKS에서는 sunje.goldilocks.jdbc 패키지에 GoldilocksXADataSource, GoldilocksConnectionPoolDataSource를 제공한다. 이 두 클래스 모두 GoldilocksDataSource와 마찬가지로 각종 setter method를 제공한다.

```
import sunje.goldilocks.jdbc.GoldilocksXADataSource;
GoldilocksXADataSource ds = new GoldilocksXADataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
XAConnection con = ds.getXAConnection();
```

연결 프로퍼티

표 26-1 연결 프로퍼티

이름	필수/선택	유효값	설명
alternate_servers	선택	IP:PORT[,IP:PORT]+	Failover를 위한 대체 서버 목록이다. 쉼표 (,)로 구분한다.
batch_count	선택	Any integer	한 번의 프로토콜 송수신으로 처리할 수 있는 batch job 개수이다. 기본값은 1000이다. 이 값이 너무 작으면 배치를 처리할 때 네트워크 송수신이 잦아져 성능이 저하될 수 있고, 너무 크면 세션이 execution result들을 쌓아놔야 하기 때문에 서버 메모리가 증가될 수 있다.
connection_retry_count	선택	Any integer	연결할 때의 재시도 횟수를 저장한다. 기본값은 0이고 재시도하지 않는다.
connection_retry_delay	선택	Any integer	연결을 재시도할 때 재시도 하기 전의 delay를 초 단위로 지정한다. 기본값은 3이다.
date_format	선택	Any string	드라이버 내부에서 date와 string 사이의 변환 작업에 사용되는 문자 포맷이다.
da_buffer_size	선택	Any integer	Direct attach 모드로 접속했을 경우 fetch, bind 할 때 데이터를 주고 받는 버퍼의 크기를 지정한다. 기본값은 100000이다.
decoding_replacement	선택	Any string	바이트 배열을 string으로 decoding 할 때, decoding할 수 없는 바이트 값을 대체할 문자이다. 기본값은 ?이다.

이름	필수/선택	유효값	설명
failover_granularity	선택	{"0", "1", "2"}	Failover 성공 여부를 판단한다. non-atomic(0), atomic(1), 2는 아직 지원하지 않는다. Failover 도중에 기존 prepared statement들에 대해 prepare 과정에서 에러가 발생할 경우 0은 failover를 계속 수행하고 1은 failover 실패를 결정한다. 기본값은 0이다.
failover_type	선택	{"connection", "session"}	Failover 유형을 결정한다. - Connection: 서버에 접속할 때만 failover를 사용한다. - Session: 서버에 접속할 때 뿐만 아니라 execution 등 서버와 통신하는 모든 경우에 failover를 사용한다. 기본값은 connection이다.
format_grammar	선택	{"db", "java"}	date_format과 같은 속성 문자열이 GOLDILOCKS 문법인지, Java의 SimpleDateFormat에 사용되는 문법인지를 결정한다. 기본값은 db이다.
global_logger	선택	{"console"}	로깅 대상을 지정한다. 현재는 console만 가능하다. 가장 먼저 지정한 것만 유효하다.
home_dir	선택	Any string	Cluster server의 home directory를 지정한다. 기본값은 null이다.
keep_alive	선택	{"on", any}	Connection의 socket 속성을 keep_alive로 설정할지 여부이다. 이 속성을 부여하면 TCP socket 내부에서 주기적으로 ack를 주고받으며 서로 연결된 상태를 유지한다. 즉, 랜선 에러를 감지하여 연결을 강제로 끊기게 할 수 있다.
locator_connection_timeout	선택	Any integer	glocator로부터 패킷을 받을 때까지 대기하는 시간이다.
locator_file	선택	Any string	Location file이다.
locator_host	선택	IP address	glocator의 host address이다.
locator_port	선택	Port no	glocator의 port이다.
locator_service	선택	Any string	서버 접속 정보를 얻기 위한 service 이름이다.
lzeros	선택	Any integer	Numeric을 문자열로 표현할 때 소수점 다음 0의 개수가 이 값을 넘어가면 exponent 표기법으로 표현된다. 기본값은 15이다.
new_password	선택	Any string	old_password 속성과 함께 사용하여 계정 암호를 바꿀 수 있다.
old_password	선택	Any string	new_password 속성과 함께 사용하여 계정 암호를 바꿀 수 있다.
packet_compression_threshold	선택	Any integer	서버로 보낼 통신 데이터의 크기가 packet_compression_threshold 보다 클 경우, 통신 데이터를 압축한다. 속성값의 범위는 32 ~ 2113929216 이다.
password	필수	Any string	사용자 계정 암호이다.
program	선택	Any string	프로그램에 대한 설명이다.
protocol_log	선택	{"on", any}	프로토콜 송수신 로깅을 할 것인지 여부이다. 기본값은 "" (안함)이다.
query_log	선택	{"on", any}	쿼리 로깅을 할 것인지 여부이다. 기본값은 "" (안함)이다.
role	선택	{"", "SYSDBA", "ADMIN"}	계정 롤을 지정한다. 기본값은 ""이다.
		{"1", "2", "3"} 또는	

이름	필수/선택	유효값	설명
session_type	선택	{"dedicate", "shared", "default"}	Dedicated/ shared/ default 중 하나이다. (Default를 선택할 경우, DB에서 선택된다.)
time_format	선택	Any string	드라이버 내부에서 time과 string 사이의 변환 작업에 사용되는 문자 포맷이다.
timestamp_format	선택	Any string	드라이버 내부에서 timestamp와 string 사이의 변환 작업에 사용되는 문자 포맷이다.
timetz_format	선택	Any string	드라이버 내부에서 time with timezone와 string 사이의 변환 작업에 사용되는 문자 포맷이다.
timestamp_tz_format	선택	Any string	드라이버 내부에서 timestamp with timezone와 string 사이의 변환 작업에 사용되는 문자 포맷이다.
trace_log	선택	{"on", any}	트레이스 로깅을 할 것인지 여부이다. 기본값은 "" (안함)이다.
tzeros	선택	Any integer	Numeric을 문자열로 표현할 때 자릿수 0의 개수가 이 값을 넘어가면 exponent 표기법으로 표현된다. 기본값은 15이다.
user	필수	Any string	사용자 계정 이름이다.
use_target_type	선택	{"0", "1", "2"}	통신을 통해 column 타입을 수신할 경우, 같이 수신할 정보이다. 0: none 1: name 2: all

노트

- locator_file은 locator_host와 locator_port 보다 우선적으로 적용된다. locator_file에 대한 자세한 내용은 **Location File**을 참조한다.
- locator_service 속성은 locator_service에 속한 서버에 접속할 수 있게 한다. 자세한 내용은 **glocator**와 **gloctl**를 참조한다.

데이터 조작

Statement를 이용한 데이터 조작

Statement 객체를 사용하여 각종 SQL 문을 실행할 수 있다. 다음과 같이 connection 객체로부터 statement 객체를 얻을 수 있다.

```
Connection con = DriverManager.getConnection(...);
Statement stmt = con.createStatement();
```

이렇게 생성된 statement 객체의 execute method와 executeUpdate method를 사용하여 각종 DML 문이나 DDL 문을 수행할 수 있다.

```
stmt.executeUpdate("create table emp ( id varchar(20), name varchar(30), age integer )");
stmt.executeUpdate("insert into emp values ('1234560000', '김연아', 24)");
int updated = stmt.executeUpdate("delete from emp where age > " + age);
```

execute와 executeUpdate의 차이점은 오직 반환값 뿐이다. execute는 수행한 SQL 문이 ResultSet을 반환하는지 여부를, executeUpdate는 갱신이 반영된 row의 개수를 각각 반환한다. execute는 DML이나 SELECT 구문에 모두 사용할 수 있지만 executeUpdate를 SELECT 문에 사용하면 SQLException이 발생한다.

노트

Statement로 SQL 문을 실행하는 것을 direct-execution이라고 한다. Direct-execution은 SQL 문에 대한 prepare 작업 (파싱, 검증 (validation), 최적화 (optimization) 작업)과 실행 (execution)을 한 번에 수행하는 방법이다. 이에 반해 prepare-execution은 prepare 작업을 한 번 해두고, execution을 반복 수행하는 방법이다. Direct-execution을 한 번 수행할 때마다 prepare, execution 작업이 발생하기 때문에 동일한 SQL 문을 반복적으로 수행해야 할 경우 prepare-execution이 더 효율적이다.

PreparedStatement를 이용한 데이터 조작

JDBC 표준은 데이터 조작을 위해 parameter를 사용할 수 있는 PreparedStatement 인터페이스를 제공하여 사용자가 보다 효율적으로 실행할 수 있도록 한다. 뿐만 아니라 PreparedStatement는 반복적인 실행에 대해 SQL 문 prepare 작업을 한 번만 하기 때문에 빠르게 수행할 수 있다.

```
Connection con = DriverManager.getConnection(...);
PreparedStatement pstmt = con.prepareStatement(
    "insert into emp values (?, ?, ?)");
pstmt.setString(1, "55544123000");
pstmt.setString(2, "김길동");
pstmt.setInt(3, 32);
pstmt.executeUpdate();
pstmt.setString(1, "1357924680");
pstmt.setString(2, "고돌리");
pstmt.setInt(3, 41);
pstmt.executeUpdate();
```

라인 2 ~ 3처럼 SQL 문을 prepare 해놓은 후에 데이터를 바인딩하고 실행한다. 한 번 prepare된 PreparedStatement는 계속해서 바인딩과 실행을 반복 수행할 수 있다.

만약 바인딩을 생략하고 execute()를 수행하면 이전에 바인딩 했던 값들이 사용된다.

```
pstmt.setString(1, "333555777000");
pstmt.setString(2, "강철규");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

이와 같이 라인 6에서 2, 3 번째 parameter에 값을 바인딩하지 않고 바로 실행하면 이전에 바인딩 된 값인 "강철규"와 55가 사용된다. 만약 이전에 바인딩 된 값이 없을 경우 SQLException이 발생한다.

바인딩 된 값들을 모두 제거하려면 clearParameters()를 사용한다.

```
pstmt.setString(1, "333555777000");
pstmt.setString(2, "강철규");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.clearParameters();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

이와 같이 라인 5에서 parameter들을 제거한 후, 첫 번째 parameter에만 값을 바인딩하고 실행하면 (라인 7) SQLException이 발생한다.

PreparedStatement를 사용하는 주된 이유는 다양한 타입의 데이터를 바인딩 할 수 있기 때문이다. Statement로는 값을 문자열로 표현할 수 밖에 없기 때문에 binary나 timestamp 같은 타입의 데이터를 입력하기에는 불편하다. 하지만 PreparedStatement로는 모든 타입으로 바인딩 할 수 있기 때문에 더 편리하다.

```
pstmt.setTimestamp(1,
    new Timestamp(Calendar.getInstance().getTimeInMillis()));
pstmt.setCharacterStream(2, new StringReader(BIG_STRING));
pstmt.setObject(3, someObj, Types.LONGVARCHAR);
pstmt.setObject(4, otherObj);
```

위의 예에서 1 ~ 2번 라인은 java.sql.Timestamp 객체를 바인딩한다. 바인딩 타입 (서버에 데이터가 전해질 때, 그 데이터의 GOLDILOCKS 타입)은 TIMESTAMP이다. 각종 setter method에 의해 결정되는 바인딩 타입에 대한 자세한 내용은 **PreparedStatement**의 해당 API를 참조한다.

라인 3은 reader 객체를 바인딩하는데, 내부적으로 LONG VARCHAR 타입으로 바인딩하게 된다.

라인 4는 Java object 타입의 객체를 바인딩하는데, 타입이 LONG VARCHAR라는 것을 명시적으로 알려준다. Types의 타입에 매핑되는 GOLDILOCKS 타입에 대한 자세한 내용은 표 26-3 SQL 타입 → GOLDILOCKS 타입을 참조한다.

라인 5는 Java object 타입을 그대로 바인딩하는데, 클래스 타입에 따라 해당되는 GOLDDILOCKS 데이터 타입으로 바인딩된다. 클래스 타입과 GOLDDILOCKS 데이터 타입 사이의 매핑에 대한 자세한 내용은 표 26-2 Java 객체 → GOLDDILOCKS 타입을 참조한다.

Statement를 이용한 배치 실행

Statement의 addBatch()와 executeBatch()를 사용하여 서로 다른 SQL 문을 batch job 형태로 실행할 수 있다.

```
stmt.addBatch("insert into emp values ('12345', 'Jake', 22)");
stmt.addBatch("insert into salary values ('12345', 5000)");
stmt.addBatch("update members set total_count=total_count+1 where age=22");
stmt.executeBatch();
```

이처럼 서로 다른 SQL 문을 하나의 executeBatch() method를 통해 수행할 수 있다. 하지만 이 method 호출을 통해 세 개의 구문 모두가 서버에서 수행된 결과를 모아 한 번의 프로토콜로 JDBC 드라이버에 전달하는 것은 아니다. 내부적으로 세 번의 프로토콜 전송과 서버에서의 실행, 결과 전송이라는 작업이 수행되는 것이다. 즉, 성능상 큰 이점을 가질 수 있는 것은 아니라는 점에 주의해야 한다.

executeBatch를 실행한 후 등록된 job들은 모두 지워진다.

실행 도중 에러가 발생해도 끝까지 등록된 job을 모두 수행한다. 에러 발생 유무는 반환된 int[] 형의 값을 참고하면 알 수 있다. 즉, int[]이 EXECUTE_FAILED 값을 가지면 그 job은 실패했다는 의미이다.

PreparedStatement를 이용한 배치 실행

PreparedStatement의 addBatch()와 executeBatch() method를 이용하여 좀 더 강력한 배치 실행을 수행할 수 있다.

```
PreparedStatement pstmt = con.prepareStatement("insert into emp values (?, ?, ?)");
pstmt.setString(1, "12345000");
pstmt.setString(2, "John");
pstmt.setInt(3, 33);
pstmt.addBatch();
pstmt.setString(1, "222333555");
pstmt.setString(2, "Henry");
pstmt.setInt(3, 24);
pstmt.addBatch();
int[] inserted = pstmt.executeBatch();
```

위의 예는 두 개의 row를 추가하는 과정이다. 두 개의 row를 구성할 값들을 addBatch()를 통해 바인드하고, 이후 executeBatch()를 호출하면 바인드 한 값과 execute에 대한 프로토콜이 서버로 전송되어 실행된다. 즉, executeBatch() 할 때만 통신하기 때문에 statement의 배치 실행보다 효율적이고 성능이 빠르다.

executeBatch()를 수행한 후엔 바인딩된 값들이 모두 삭제되기 때문에 executeBatch()을 다시 수행하려면 새로운 값을 다시 바인드 해야 한다. 실행한 후에 등록된 job들을 지우기 위해 clearBatch()를 명시적으로 호출하지 않아도 된다.

실행하는 도중에 예러가 발생해도 등록된 job 모두를 끝까지 수행한다. 예러 발생 유무는 반환된 int[] 형의 값을 참고하여 확인할 수 있다. 즉, int[]이 EXECUTE_FAILED 값을 가지면 그 job은 실패했다는 의미이다.

GOLDILOCKS JDBC는 executeBatch() 외에 executeBatchAtomic()이라는 별도의 method를 제공한다. executeBatch()는 서버에서 등록된 job의 개수만큼 실행하지만, executeBatchAtomic()은 등록된 job (바인딩된 값들)을 한 번에 실행하기 때문에 응답시간이 더 빠르다. 다만 atomic이라는 이름에서 알 수 있듯이 하나라도 실패하게 되면 전부 실패하게 된다. 따라서 반환되는 유형은 int[]형이 아니라 int이다.

```
...
int inserted = ((GoldilocksPreparedStatement)pstmt).executeBatchAtomic();
```

데이터 조회

Column 값 조회

Statement나 PreparedStatement의 executeQuery()를 통해서 얻은 ResultSet 객체를 통해 데이터를 조회할 수 있다.

```
...
ResultSet rs = pstmt.executeQuery();
while(rs.next())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    ...
}
```

테이블의 데이터를 조회한 후 ResultSet의 각종 getter method를 통해 해당 column의 내용을 얻을 수 있다. 테이블의 데이터는 GOLDILOCKS 데이터 타입의 원래 형태로 JDBC 드라이버에 전달되며, 사용자가 호출한 getter method 종류에 따라 적절한 Java 데이터 타입으로 변환되어 사용자에게 전달된다. GOLDILOCKS 데이터 타입과 getter method와의 타입 변환 매핑은 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.

GOLDILOCKS 데이터 타입을 해당 getter의 타입으로 변환할 수 없으면 SQLException이 발생한다.

Closing ResultSet

사용을 마친 ResultSet의 자원은 close()를 통해 해제할 수 있다. 사용자가 명시적으로 close()를 호출하지 않아도 다음과 같은 경우에는 ResultSet의 close()를 호출한 것과 동일한 결과를 가져온다.

- 상위 statement가 close되었을 때
- 상위 connection이 close되었을 때
- Statement의 executeQuery()가 다시 호출되었을 때
- Fetch하는 도중 에러가 발생했을 때

위의 세 번째 경우에 하나의 statement로부터 생성되는 ResultSet의 경우 동시에 두 개가 유지될 수 없다. 마지막 executeQuery()를 통해 생성된 ResultSet 객체만 유효하다.

Fetch Size

JDBC는 statement의 setFetchSize(int rows)를 통해 서버로부터 한 번에 fetch해오는 row 개수를 지정할 수 있다. GOLDBLOCKS의 ResultSet은 이 속성의 기본값이 0이다. 0은 서버가 fetch 할 row의 개수를 자동으로 정하는데, forward only인 경우 한 통신 패킷에 담을 수 있는 최대 row 개수이고, scrollable인 경우 100개로 고정된다. 이 값이 크면 JDBC ResultSet이 차지하는 메모리 양이 많아지고, 너무 작으면 통신이 자주 발생한다.

이 값은 주로 scrollable ResultSet에 많이 사용하는데, 왜냐하면 scroll sensitive라 할지라도 ResultSet의 row 캐시 내에서 이동할 때는 sensitive하지 않기 때문에 이 값을 너무 크게 잡으면 row에 대한 최신 갱신 정보를 가질 수 없기 때문이다.

필드 크기 제한

JDBC는 statement의 setMaxFieldSize(int size)를 통해 한 column의 최대 길이를 제한할 수 있다. 이 설정에 의해 CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, LONG VARBINARY 타입의 최대 길이를 제한할 수 있다. 이 길이를 넘는 데이터는 잘라버린다. 기본값은 0이고 이 경우 최대 길이를 제한하지 않는다.

INTEGER, DATE와 같은 다른 타입에 대해서는 이 속성이 무시된다.

ResultSet 스크롤

Scrollable ResultSet 획득

Scrollable ResultSet은 scroll sensitive와 scroll insensitive 두 가지이다. Scroll sensitive는 ResultSet을 순회하는 동안 자기 트랜잭션에 의해서든, 타 트랜잭션에 의해서든 갱신된 값들을 볼 수 있는 커서 유형이고, scroll insensitive는 그렇지 못한 커서이다. GOLDBLOCKS 서버는 두 가지 스크롤 유형을 모두 지원하며, JDBC를 통해서도 그 기능을 자유롭게 사용할 수 있다.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                       ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select id, name, age from emp");
```

라인 1처럼 statement를 생성할 때 ResultSet 타입을 TYPE_SCROLL_SENSITIVE로 설정하면 이 statement 객체로부터 생성되는 ResultSet은 모두 scroll sensitive 커서가 된다. 단, 현재 GOLDILOCKS는 updatable ResultSet은 지원하지 않는다.

Scroll insensitive인 ResultSet을 얻으려면 타입을 TYPE_SCROLL_INSENSITIVE로 설정해야 한다. 기본값은 TYPE_FORWARD_ONLY이다.

단, scroll sensitive ResultSet이라 하더라도 ResultSet의 row 캐시 내에서 이동할 때는 최신값을 볼 수 없고 갱신 유무도 알 수 없다. Row를 서버로부터 다시 fetch 해와야만 알 수 있다.

스크롤

JDBC API는 커서를 스크롤 하기 위해 다음과 같은 method를 제공한다.

```
public boolean next() throws SQLException;
public boolean previous() throws SQLException;
public boolean first() throws SQLException;
public boolean last() throws SQLException;
public boolean absolute(int rows) throws SQLException;
public boolean relative(int rows) throws SQLException;
public void beforeFirst() throws SQLException;
public void afterLast() throws SQLException;
```

next()는 forward only나 scrollable ResultSet에 모두 사용할 수 있는 method이며, 커서 위치를 다음 row로 이동시킨다. 이동한 위치에서 row를 읽을 수 있으면 true를, 그렇지 않을 경우에는 false를 반환한다. 그 외의 method들은 scrollable ResultSet에만 사용할 수 있다.

previous()는 이전 row로, first()는 첫 번째 row로, last()는 마지막 row로, absolute()는 절대 위치로, relative()는 현재 위치로부터 상대 위치로 이동시킨다. beforeFirst()는 첫 번째 row 이전으로 위치시키며, afterLast()는 마지막 row의 다음에 위치시킨다.

```
rs.afterLast();
while (rs.previous())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    ...
}
```

위 코드는 마지막 row부터 첫 번째 row까지 거꾸로 조회한다.

현재 커서의 위치는 다음 method를 통해 알 수 있다.

```
public int getRow() throws SQLException;
```

스크롤 원리

ResultSet 스크롤은 사용하기 간단한 편이지만, 내부 동작 원리를 알지 못하면 성능이 크게 저하될 수도 있어 주의해서 사용해야 한다. 스크롤은 JDBC 드라이버의 ResultSet 내의 result cache에서 수행되지만, cache를 벗어난 범위에 대해서는 서버에서 스크롤이 수행되며 이에 따라 통신이 발생하면 성능이 저하될 수 있다.

Scrollable ResultSet의 fetch size는 내부 (internal) 기본값이 100이다. (외부 기본값은 0이며, 이 값은 setFetchSize() method를 통해 변경할 수 있다.) 즉, 한 번에 JDBC 드라이버로 가져오는 row 개수가 100개라는 뜻이다. ResultSet의 row 캐시는 100개의 row로 구성되는데, 이동시키려는 row 위치가 캐시 내에 있을 경우, ResultSet의 커서 위치만 바꾸면 된다. 하지만 캐시 내에 없을 경우에는 이동시키려는 위치의 row를 다시 서버로부터 가져와야 한다. 이 때 이동시키려는 row를 포함한 row set을 가져오는 방식을 알아야 한다.

예를 들어, ResultSet이 1번부터 100번의 row를 가지고 있고 현재 커서 위치가 100번에 있을 경우에 next()를 호출하게 되면 다음 row가 캐시 내에 없기 때문에 101번부터 200번까지의 row를 서버로부터 fetch해 와야 한다. 반면에 101번부터 200번까지의 row가 ResultSet 캐시 내에 있고, 현재 위치가 101번 row에 있을 때 previous()를 호출한 경우, 서버로부터 100번부터 199번 row를 fetch 해온다면 이는 적절하지 못하다. 만약 이렇게 구현한다면 previous()를 호출할 때마다 서버에서 매번 fetch 해와야 하기 때문이다. GOLDBLOCKS는 이런 비효율을 방지하기 위해 previous()를 호출하여 해당 row를 서버로부터 가져와야 하는 상황이 되면 해당 row를 마지막으로 하는 row set을 서버로부터 가져온다. 즉, previous()에 의해 100번 row를 fetch 해와야 하는 상황이라면 1번부터 100번까지의 row를 fetch 해온다. 그러면 previous()에 유리하도록 fetch하게 된다.

이 규칙을 일반화하면 다음과 같다.

- 이동시키려는 위치가 현재 위치 이후일 경우, 그 위치로부터 100개 (설정된 fetch size)의 row를 fetch한다.
- 이동시키려는 위치가 현재 위치 이전이면 그 위치 앞쪽의 100개 row를 fetch한다.
- first()로 이동할 때는 첫 100개를 가져온다.
- last()로 이동할 때는 마지막 100개를 가져온다.

즉, 뒤로 이동할 때는 next()에 유리하도록, 앞으로 이동할 때는 previous()에 유리하도록 fetch한다. 이는 absolute(), relative()로 이동할 때도 마찬가지이다.

추가 타입 사용

Interval 타입

GOLDILOCKS는 interval과 관련하여 다음과 같은 13가지 타입을 지원한다.

- interval year
- interval month
- interval year to month
- interval day
- interval hour
- interval minute
- interval second
- interval minute to second
- interval hour to minute
- interval hour to second
- interval day to hour
- interval day to minute
- interval day to second

JDBC를 이용하여 `sunje.goldilocks.jdbc.GoldilocksInterval` 객체를 생성하면 `PreparedStatement`의 `setObject()`를 통해 interval 데이터를 테이블에 추가할 수 있다.

`GoldilocksInterval` 객체는 `GoldilocksInterval`의 static method인 `createIntervalXXX`를 사용하여 생성할 수 있다.

```
import sunje.goldilocks.jdbc.GoldilocksInterval;
...
GoldilocksInterval interval = GoldilocksInterval.createIntervalDayToSecond(2, 6, "2
08:23:54.560843");
PreparedStatement pstmt = con.prepareStatement("insert into interval_table values (?,?)");
pstmt.setString(1, someId);
pstmt.setObject(2, interval);
pstmt.executeUpdate();
```

`createIntervalXXX` method에 대한 자세한 내용은 **GoldilocksInterval**을 참조한다. 이와 같이 `GoldilocksInterval`을 생성하여 `setObject`를 통해 바인딩 할 수 있다. 또는 타입을 명시하기 위해 다음과 같이 실행해도 된다.

```
import sunje.goldilocks.jdbc.GoldilocksTypes;
...
pstmt.setObject(2, interval, GoldilocksTypes.INTERVAL_DAY_TO_SECOND);
```

`GoldilocksInterval` 객체없이 string 문자열로 데이터를 추가할 수 있다.

```
pstmt.setObject(2, "2 08:23:54.560843", GoldilocksTypes.INTERVAL_DAY_TO_SECOND);
```

이 경우, JDBC 드라이버 내에서 GoldilocksInterval 객체를 생성하여 호스트 변수에 바인딩한다. 또는 `setString()` method를 통해 문자열로 바인딩 할 수도 있다.

```
pstmt.setString(2, "2 08:23:54.560843");
```

이렇게 하면 서버에 문자열 그대로 전송되며 서버에서 문자열을 interval day to second 타입으로 변환하여 추가한다.

데이터베이스로부터 interval 타입을 조회하려면 ResultSet의 `getObject()`나 `getString()`을 이용한다.

```
import sunje.goldilocks.jdbc.GoldilocksInterval;
ResultSet rs = stmt.executeQuery("select interval_value from some_table");
while (rs.next())
{
    GoldilocksInterval interval = (GoldilocksInterval)rs.getObject(1);
    System.out.println("type = " + interval.getTypeName() +
        ", value = " + interval.toString());
}
```

ResultSet의 `getObject()`를 통해 GoldilocksInterval 타입의 데이터를 얻을 수 있다. GoldilocksInterval 클래스는 각종 시간 데이터를 얻을 수 있는 `getYear()`, `getHour()` 등의 getter를 제공한다. getter API에 대한 자세한 내용은 GoldilocksInterval을 참조한다.

Time with time zone과 Timestamp with time zone 타입

GOLDILOCKS는 시간과 관련하여 date, time, timestamp와 같은 SQL 표준 타입 뿐만 아니라 time with time zone과 timestamp with time zone 타입도 제공한다.

```
CREATE TABLE SAMPLE_TABLE ( C1 TIME WITH TIME ZONE,
                             C2 TIMESTAMP WITH TIME ZONE );
```

GOLDILOCKS JDBC는 time with time zone과 timestamp with time zone 타입의 데이터를 추가하기 위해 GoldilocksPreparedStatement 클래스에 `setTimeTimeZone (int colIndex, time time, calendar timezone)` method와 `setTimestampTimeZone (int colIndex, timestamp time, calendar timezone)` method를 제공한다. 자세한 내용은 `setTimeTimeZone`과 `setTimestampTimeZone`을 참조한다.

물론 기존의 `setTime()`과 `setTimestamp()` method로도 추가할 수 있지만, 이 column의 time zone 값은 무조건 Java 실행 환경의 local time zone으로 설정된다. 따라서 다양한 time zone 정보를 추가하기 위해서는 `setTimeTimeZone()`이나 `setTimestampTimeZone`을 사용해야 한다.

`setTimeTimeZone`과 `setTimestampTimeZone`으로 주어진 time, timestamp 데이터를 주어진 time zone 값과 함

께 추가할 수 있다.

```
import sunje.goldilocks.jdbc.GoldilocksPreparedStatement;
PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO SAMPLE_TABLE VALUES (?, ?, ?, ?)");
Calendar now = Calendar.getInstance();
Calendar usNow = Calendar.getInstance(TimeZone.getTimeZone("GMT-8"));
Time t = new Time(now.getTimeInMillis());
Timestamp ts = new Timestamp(now.getTimeInMillis());
pstmt.setTime(1, t);
pstmt.setTimestamp(2, ts);
pstmt.executeUpdate();
((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, t, now);
((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(2, ts, now);
pstmt.executeUpdate();
((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, t, usNow);
((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(2, ts, usNow);
pstmt.executeUpdate();
```

11, 12 라인은 해당 column에 time 타입과 timestamp 타입으로 바인딩한다. t와 ts 값이 각각 서버로 전달되어 서버에서 기본 time zone 값이 DB column에 추가된다.

이에 반해 15, 16 라인은 해당 column에 time with time zone 타입과 timestamp with time zone 타입으로 바인딩하여, time zone 정보도 함께 서버로 전달한다. 결과적으로 서버와 client의 time zone 설정이 동일할 경우, 11 ~ 13 라인에 의해 추가되는 row와 15 ~ 17 라인에 의해 추가되는 row는 동일하다.

19, 20 라인은 GMT-8 시간대의 time zone 정보를 추가한다.

Time with time zone과 timestamp with time zone의 데이터를 조회하는 방법은 크게 두 가지이다. 서버로부터 char로 얻는 방법과 JDBC로 해당 타입의 데이터를 가져와서 time이나 timestamp 객체로 얻는 방법이 있다.

```
ResultSet rs = stmt.executeQuery("select c1, cast(c1 as char(33)), c2, cast(c2 as char(33))
from sample_table");
while (rs.next())
{
    System.out.println("c1 = " + rs.getTime(1).toString());
    System.out.println("c1 as char = " + rs.getString(2));
    System.out.println("c2 = " + rs.getTimestamp(3).toString());
    System.out.println("c2 as char = " + rs.getString(4));
}
```

위 코드를 실행하면 다음과 같은 결과를 확인할 수 있다.

```

c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-27 12:35:26.052000 +09:00
c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-27 12:35:26.052000 +09:00
c1 = 12:35:26
c1 as char = 19:35:26.052000 -08:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-26 19:35:26.052000 -08:00

```

로깅

로깅 종류

JDBC를 이용하는 프로젝트를 개발할 때, JDBC 드라이버가 각종 로그를 남길 수 있으면 여러가지로 도움이 된다. GOLDILOCKS는 프로젝트를 개발할 때만이 아니라 운용할 때도 유용한 정보를 로깅할 수 있는 기능을 제공한다.

로깅의 종류는 세 가지인데 trace log, protocol log, query log 이다.

Trace log는 JDBC API가 호출될 때마다 정보를 남긴다. 어떤 JDBC API가 호출되는 지를 알 수 있다. Protocol log는 JDBC 드라이버와 GOLDILOCKS 서버 사이에 통신 패킷을 주고 받는 상황을 보여준다. Query log는 PreparedStatement나 statement가 실행될 때, 실행할 SQL 문을 기록한다.

DriverManager를 이용한 로깅

먼저 DriverManager를 이용하여 로깅을 남길 수 있다. DriverManager에 setLogWriter() method를 이용하여 로깅할 매체를 결정할 수 있고, connection url에 남길 로깅의 종류를 명시한다.

```

DriverManager.setLogWriter(new PrintWriter(System.out));
String url = "jdbc:goldilocks://localhost:22581/test?trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");

```

1번 라인은 로깅 매체를 정의하는 것이다. 모든 로깅을 콘솔로 출력하게 한다. 2번 라인은 connection URL을 정의하는 것인데, ? 문자 뒤에 프로퍼티를 정의할 수 있다. trace_log, query_log를 사용하겠다는 의미이다.

이와 같이 이 프로퍼티들은 URL에 명시할 수도 있고, properties 객체로 정의할 수도 있다.


```

Properties prop = new Properties();
prop.put("trace_log", "on");
prop.put("query_log", "on");
prop.put("protocol_log", "on");
prop.put("user", "TEST");
prop.put("password", test);
Connection con = DriverManager.getConnection(url, prop);

```

GOLDILOCKS에서 사용할 수 있는 connection property들은 표 26-1 연결 프로퍼티에 정의되어 있다.

간혹 DriverManager의 setLogWriter()를 호출하기 어려운 경우가 있다. 예를 들어, 미들웨어를 사용하는 경우 저런 코드를 추가할 수 없다. 그런 경우에 대비해 GOLDILOCKS JDBC는 global_logger라는 전역 프로퍼티를 제공한다. 다른 일반적인 프로퍼티들은 하나의 connection에만 적용되지만, 이 속성은 전역적으로 적용된다. 즉, 이 프로퍼티를 설정하면 DriverManager.setLogWriter()를 호출하지 않아도 된다.

```

String url = "jdbc:goldilocks://localhost:22581/test?" +
            "global_logger=console&trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");

```

global_logger 속성은 처음 한 번만 적용되는데 만약 DriverManager에 이미 log writer가 설정되어 있으면 이 속성은 무시된다. 이 속성의 값으로는 현재 console만 지원된다. 다른 값인 경우에는 아무런 동작을 하지 않는다.

DataSource를 이용한 로깅

DriverManager와는 다르게 DataSource는 객체별로 log writer를 설정할 수 있다. 하나의 DataSource 객체에 log writer를 세팅하면 그 DataSource로부터 생성되는 모든 connection은 그 log writer로 로깅된다. DataSource에 로깅을 설정하는 방법은 다음과 같다.

```

import sunje.goldilocks.jdbc.GoldilocksDataSource;
...
GoldilocksDataSource ds = new sunje.goldilocks.jdbc.GoldilocksDataSource();
ds.setServerName("localhost");
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
ds.setPortNumber(22581);
ds.setLogTarget("console");
ds.setTraceLog("on");
ds.setQueryLog("on");
ds.setProtocolLog("on");
Connection con = ds.getConnection();

```

이와 같이 `setLogTarget()`, `setTraceLog()`, `setQueryLog()`, `setProtocolLog()` method를 사용하여 로깅 기능을 설정할 수 있다. `setLogTarget()` 대신 `DataSource`의 `setLogWriter()`를 호출해도 된다. 다만 미들웨어를 사용하는 경우에는 `setLogWriter()`와 같은 method를 직접 호출할 수 없기 때문에 property로 제어해야 하는데, 이 때 `logTarget`, `traceLog`, `queryLog`, `protocolLog` 속성을 정의하면 미들웨어가 자동으로 이 method들을 호출한다.

Plan Text 조회

사용법

GOLDILOCKS ODBC를 사용하면 실행된 statement의 plan text를 얻어올 수 있는 것처럼 GOLDILOCKS JDBC를 사용해서도 plan text를 얻어올 수 있다. `GoldilocksStatement` 클래스에 비 표준 API method를 사용하여 plan text를 생성하도록 설정할 수 있고, 만들어진 plan text를 얻어올 수도 있다.

```
Connection con = ...
GoldilocksStatement stmt = (GoldilocksStatement)con.createStatement();
stmt.setExplainPlanOption(GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON);
ResultSet rs = stmt.executeQuery("select * from t1");
System.out.println(stmt.getExplainPlan());
```

예를 들어 간단한 테이블을 조회했을 때의 plan text는 다음과 같이 출력된다.

```
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION  |  ROWS  |
=====
|    0  |  SELECT STATEMENT  |         |
|    1  |  TABLE ACCESS ("T1")  |    0  |
=====
1 - READ COLUMNS : A
```

옵션 종류

GOLDILOCKS는 다음과 같이 plan text 생성과 관련된 네가지 속성을 제공한다.

- `EXPLAIN_PLAN_OPTION_OFF`: Statement의 plan text 관련 속성의 기본값으로써 plan text를 생성하지 않는다.
- `EXPLAIN_PLAN_OPTION_ON`: 실행할 때나 fetch 할 때 plan text를 생성한다.
- `EXPLAIN_PLAN_OPTION_ON_VERBOSE`: 실행할 때나 fetch 할 때 실행 시간과 같이 보다 자세한 plan text를 생성한다.
- `EXPLAIN_PLAN_OPTION_ONLY`: 실행할 때나 fetch 할 때 `EXPLAIN_PLAN_OPTION_ON`처럼 plan text를

생성하지만 실제로 실행은 하지 않는다.

이 속성들은 GoldilocksStatement.setExplainPlanOption(int) method를 사용하여 설정할 수 있으며 한 번 설정하면 계속해서 이 속성이 유지된다. 속성의 상수값은 GoldilocksStatement에 정의되어 있다.

노트

SELECT 구문이 아닌 DML이나 기타 SQL 구문은 실행할 때 plan text가 생성되지만 SELECT 구문은 fetch되고 서버 내에서 커서가 끝까지 갔을 때 plan text가 생성된다. 따라서 SELECT 구문일 경우 ResultSet으로 모든 row를 순회한 다음에 plan text를 얻어야 한다. 단, 테이블에 row가 적을 경우에는 끝까지 fetch하지 않아도 서버 커서의 끝까지 순회할 수 있으므로 이 경우 plan text를 얻을 수 있다.

Connection Failover

GOLDILOCKS는 client 레벨 (JDBC, ODBC)의 connection failover를 지원한다. GOLDILOCKS에 JDBC 또는 ODBC를 이용하여 client/ server 모드로 접속한 응용 프로그램은 연결에 실패하거나 연결이 끊길 경우, 자동으로 대체 서버로 연결할 수 있다. 이를 통해 여러 대의 서버를 활용한고가용성 응용 프로그램을 작성할 수 있도록 해준다. 특히 웹 서버와 연동된 환경에서 연결 속성 (connection property)을 통해 주 서버 URL과 대체 서버 URL을 등록하면 connection pool로부터 가져온 connection 객체를 끊길 걱정없이 안정적으로 사용할 수 있다. Failover는 전적으로 드라이버 내부에서 자동 진행되며 사용자는 재접속에 대한 로직을 고민할 필요없다.

Failover 기능을 사용하려면 alternate_servers 속성을 부여해야 한다. 이 속성값으로 대체 서버의 IP와 port를 부여할 수 있다. 콤마 (,)를 사용하여 여러 개의 대체 서버를 명시할 수도 있다.

```
Class.forName("sunje.goldilocks.jdbc.GoldilocksDriver");
String url = "jdbc:goldilocks://192.168.0.101:22581/test";
Property prop = new Properties();
prop.setProperty("user", "TEST");
prop.setProperty("password", "test");
prop.setProperty("alternate_servers", "192.168.0.201:22581");
Connection con = DriverManager.getConnection(url, prop); ❶
Statement stmt = con.createStatement();
stmt.executeUpdate("insert into time_tab values (sysdate)"); ❷
...
```

예를 들어, ❶에서 주 서버 (192.168.0.101)의 연결이 실패하면 자동으로 대체 서버 (192.168.0.201)로 접속한 후 connection 객체를 반환한다. 사용자에게는 별다른 예러가 반환되지 않는다. 만약 ❷에서 executeUpdate() 하는 도중에 연결이 끊기거나 그 전에 끊어져서 서버와 통신을 할 수 없을 경우 드라이버 내부에서 자동으로 대체 서버로 접속하여 기존 connection 객체를 계속해서 사용할 수 있도록 조치한 후 SQLException이 발생한다. 사용자는 S

SQLException을 처리한 후에 기존 connection 객체를 계속해서 사용할 수 있다.

Connection 객체의 getWarnings()를 통해 failover 여부를 알 수 있다.

GOLDILOCKS JDBC의 failover 특징 중에 하나는 connection 객체로부터 생성된 statement 객체나 PreparedStatement 객체를 계속해서 사용할 수 있다는 점이다. (ResultSet은 계속해서 사용할 수 없다)

특히 PreparedStatement들은 failover가 발생하면 내부적으로 대체 서버에서 다시 prepare 작업을 수행하므로 사용자가 계속해서 이 객체를 사용할 수 있다.

```
Connection con = DriverManager.getConnection(url, prop);
PreparedStatement pstmt =
    con.prepareStatement("insert into t1 values (?,?)");
...
try
{
    pstmt.setInt(1, 100);
    pstmt.setString(2, "John");
    pstmt.executeUpdate();
}
catch (SQLException sException)
{
    if (sException.getErrorCode() == 21012)
    {
```

- Communication link failure 에러가 발생하여 failover가 발생했다.
- 계속해서 pstmt 객체를 사용할 수 있다.

```
pstmt.executeUpdate();
    }
    else
    {
        ...
    }
}
```

GOLDILOCKS JDBC 드라이버는 failover와 관련해 몇 가지 속성을 제공한다. 이를 통해 failover 유형, 접속 순서 및 failover시 사용되는 간단한 정책 등을 결정할 수 있다.

failover_type

이 속성으로 failover 유형을 선택할 수 있다. 위 예제 코드 ❶에서 발생하는 failover를 connection failover라 하고 ❷에서 발생하는 failover를 session failover라 부른다. 이 속성값으로 connection을 부여하면 connection failover

만 수행하게 하며, session을 부여하면 connection failover와 session failover를 모두 수행하게 한다. 기본값은 connection이다.

failover_granularity

Failover가 발생하면 현재의 connection 객체로부터 생성된 PreparedStatement 객체들에 대체 서버로 접속한 후 prepare 작업을 다시 수행하는데 이 때 대체 서버에서 prepare가 실패했을 경우 (서버 환경이 달라서 에러가 발생할 수 있음) failover 자체를 실패로 처리할지 또는 prepare 에러를 무시할지 여부를 이 속성이 결정한다. 현재 이 속성 값으로 0과 1을 부여할 수 있는데, 0을 부여하면 prepare 에러를 무시한 채 failover를 계속해서 수행할 수 있게하고, 1을 주면 failover가 실패하도록 한다. 기본값은 0이다. 만약 0을 부여하고 prepare에 실패하면 PreparedStatement 객체를 계속해서 사용할 수 없고 사용자가 다시 직접 PreparedStatement 객체를 생성해야 한다.

Direct Attach 모드 접속

GOLDILOCKS JDBC 1.1부터 기존 TCP/IP 기반의 Client/ Server 모드 (C/S 모드) 외에 direct attach 모드 (D/A 모드) 연결을 제공한다. 이 연결 방식은 ODBC D/A 모드 접속과 마찬가지로 서버 프로세스와 직접 연동하여 하나의 프로세스 안에 (jvm과 같은 프로세스 안에) 서버 모듈이 연동되어 작동한다. 그렇기 때문에 원격 호스트에서는 D/A 모드로 접속할 수 없다.

D/A 모드는 in-memory 데이터베이스인 GOLDILOCKS의 빠른 처리라는 장점을 살리기 위해 고안된 특별한 연결 방식이다. TCP/IP 기반의 접속으로는 GOLDILOCKS의 빠른 트랜잭션 처리 효과가 값비싼 네트워크 비용으로 상쇄되기 때문에 빠른 처리를 요구하는 상황에서는 다른 대안이 필요하다. 비록 GOLDILOCKS 서버와 같은 호스트 내에서 실행해야 한다는 제약이 있지만 빠른 처리를 필요로 하는 경우 D/A 모드 접속은 좋은 대안이 될 수 있다.

D/A 모드로 접속한 JDBC 프로그램은 jvm 내에서 DB connection이 처음 생성되면 GOLDILOCKS jni 라이브러리가 로딩되어 서버의 기능을 직접 호출할 수 있다. 네트워크 비용없이 jvm에서 native interface를 통해 서버 모듈을 직접 호출할 수 있기 때문에 기존의 C/S 모드보다 빠른 처리가 가능하다.

접속 방법

D/A 모드로 접속하기 위해 connection URL에 기존 ip:port 대신 0.0.0.0:0을 사용한다. 기존의 URL 방식을 그대로 사용할 수 있으며 0.0.0.0:0이라는 특별한 ip:port 주소를 사용하여 기존 응용 프로그램의 수정을 최소화할 수 있다. 또는 da라는 특수 프로토콜을 URL에 사용하면 D/A 모드로 접속할 수 있다.

```
Connection con =
    DriverManager.getConnection("jdbc:goldilocks://0.0.0.0:0/test","TEST","test");
```

또는

```
Connection con =  
    DriverManager.getConnection("jdbc:goldilocks:da/test", "TEST", "test");
```

D/A 접속의 특징

D/A 모드로 접속하면 jvm 안에서 GOLDILOCKS server 모듈을 직접 호출할 수 있다. JDBC 프로그램과 GOLDILOCKS server 모듈 사이에는 Java Native Interface (JNI)를 통해 호출한다. JNI 호출 비용이 비싼 것을 고려하여 최소 호출 비용으로 서버 모듈을 사용하기 때문에 기존 TCP/IP 기반의 JDBC보다 두 배 이상 성능이 빨라진다.

기존 TCP/IP 기반의 접속을 사용할 때는 goldilocks6.jar 파일만 사용하지만 D/A 모드로 접속할 때는 \$GOLDILOCKS_HOME/lib에 있는 libgoldilocksjni.so와 libgoldilocksas.so 파일을 동적으로 로딩해서 사용한다. 따라서 두 라이브러리 파일이 없으면 연결할 때 에러가 발생한다. 하지만 Java 프로그램을 구동할 때 라이브러리 파일의 위치를 별도로 명시할 필요는 없다. goldilocks6.jar 파일과 같은 디렉토리에 있는 한 현재 디렉토리부터 이 두 개의 라이브러리 파일을 찾아서 로딩하기 때문이다.

26.3 JDBC API References

Array

클래스가 구현되지 않았다.

free

`void free() throws SQLException`

getArray

`Object getArray() throws SQLException`

`Object getArray(Map<String,Class<?>> map) throws SQLException`

`Object getArray(long index, int count) throws SQLException`

`Object getArray(long index, int count, Map<String,Class<?>> map) throws SQLException`

getBaseType

`int getBaseType() throws SQLException`

getBaseTypeName

`String getBaseTypeName() throws SQLException`

getResultSet

`ResultSet getResultSet() throws SQLException`

`ResultSet getResultSet(Map<String,Class<?>> map) throws SQLException`

| `ResultSet getResultSet(long index, int count)` throws `SQLException`

| `ResultSet getResultSet(long index, int count, Map<String,Class<?>> map)` throws `SQLException`

Blob

클래스가 구현되지 않았다.

free

`void free() throws SQLException`

getBinaryStream

`InputStream getBinaryStream() throws SQLException`

`InputStream getBinaryStream(long pos, long length) throws SQLException`

getBytes

`byte[] getBytes(long pos, int length) throws SQLException`

length

`long length() throws SQLException`

position

`long position(byte[] pattern, long start) throws SQLException`

`long position(Blob pattern, long start) throws SQLException`

setBinaryStream

`OutputStream setBinaryStream(long pos) throws SQLException`

setBytes

```
| int setBytes(long pos, byte[] bytes) throws SQLException
```

```
| int setBytes(long pos, byte[] bytes, int offset, int len) throws SQLException
```

truncate

```
| void truncate(long len) throws SQLException
```

CallableStatement

getArray

Array `getArray(int parameterIndex)` throws `SQLException`

- 동작: Array 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

Array `getArray(String parameterName)` throws `SQLException`

- 동작: Array 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getBigDecimal

BigDecimal `getBigDecimal(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 BigDecimal 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

BigDecimal `getBigDecimal(int parameterIndex, int scale)` throws `SQLException`

- 동작: 구현되어 있지 않다. (Deprecated 된 method이다.)
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

BigDecimal `getBigDecimal(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getBlob

Blob `getBlob(int parameterIndex)` throws `SQLException`

- 동작: Blob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`Blob getBlob(String parameterName)` throws `SQLException`

- 동작: Blob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getBoolean

`boolean getBoolean(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 boolean 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`boolean getBoolean(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getBytes

`byte getByte(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 byte 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조하기 바란다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있다면 `SQLException`이 발생한다.

`byte getByte(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getBytes

`byte[] getBytes(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 `byte[]` 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`byte[] getBytes(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getCharacterStream

`Reader getCharacterStream(int parameterIndex) throws SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 reader 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`Reader getCharacterStream(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getClob

`Clob getClob(int parameterIndex) throws SQLException`

- 동작: Clob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`Clob getClob(String parameterName) throws SQLException`

- 동작: Clob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getDate

`Date getDate(int parameterIndex) throws SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 local timezone과 locale을 사용한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`Date getDate(int parameterIndex, Calendar cal) throws SQLException`

- 동작: parameterIndex 번째 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 cal의 timezone과 locale을 사용한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

`Date getDate(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

`Date getDate(String parameterName, Calendar cal) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getDouble

`double getDouble(int parameterIndex) throws SQLException`

- 동작: parameterIndex 번째 column의 데이터를 double 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

`double getDouble(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getFloat

`float getFloat(int parameterIndex) throws SQLException`

- 동작: parameterIndex 번째 column의 데이터를 float 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

`float getFloat(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getInt

`int getInt(int parameterIndex) throws SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 `int` 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 2 6-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`int getInt(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getLong

`long getLong(int parameterIndex) throws SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 `long` 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`long getLong(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getNCharacterStream

`Reader getNCharacterStream(int parameterIndex) throws SQLException`

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`Reader getNCharacterStream(String parameterName)` throws `SQLException`

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getNClob

`NClob getNClob(int parameterIndex)` throws `SQLException`

- 동작: NClob 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`NClob getNClob(String parameterName)` throws `SQLException`

- 동작: NClob 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getString

`String getString(int parameterIndex)` throws `SQLException`

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`String getString(String parameterName)` throws `SQLException`

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getObject

`Object getObject(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 Java 객체 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`Object getObject(int parameterIndex, Map<String,Class<?>> map)` throws `SQLException`

- 동작: 지원하지 않는다.

- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`Object getObject(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`Object getObject(String parameterName, Map<String,Class<?>> map)` throws `SQLException`

- 동작: 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getRef

`Ref getRef(int parameterIndex)` throws `SQLException`

- 동작: Ref 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`Ref getRef(String parameterName)` throws `SQLException`

- 동작: Ref 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getRowId

`RowId getRowId(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 `rowId` 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.₩
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

`RowId getRowId(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getShort

`short getShort(int parameterIndex) throws SQLException`

- 동작: parameterIndex 번째 column의 데이터를 short 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

`short getShort(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getSQLXML

`SQLXML getSQLXML(int parameterIndex) throws SQLException`

- 동작: SQLXML 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

`SQLXML getSQLXML(String parameterName) throws SQLException`

- 동작: SQLXML 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getString

`String getString(int parameterIndex) throws SQLException`

- 동작: parameterIndex 번째 column의 데이터를 string 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

`String getString(String parameterName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getTime

Time `getTime(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 time 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다. Time 객체를 만들 때, local timezone을 사용한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

Time `getTime(int parameterIndex, Calendar cal)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 time 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다. Time 객체를 만들 때, `cal`의 timezone을 사용한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

Time `getTime(String parameterName)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

Time `getTime(String parameterName, Calendar cal)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getTimestamp

Timestamp `getTimestamp(int parameterIndex)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다. Timestamp 객체를 만들 때, local timezone을 사용한다.
- 예외: `parameterIndex`가 유효하지 않거나, database에 접근할 수 없거나 `CallableStatement`가 이미 닫혀 있을 경우, `SQLException`이 발생한다.

Timestamp `getTimestamp(int parameterIndex, Calendar cal)` throws `SQLException`

- 동작: `parameterIndex` 번째 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다. Timestamp 객체를 만들 때,

cal의 timezone을 사용한다.

- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우, SQLException이 발생한다.

Timestamp getTimestamp(String parameterName) throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

Timestamp getTimestamp(String parameterName, Calendar cal) throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getURL

URL getURL(int parameterIndex) throws SQLException

- 동작: URL 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

URL getURL(String parameterName) throws SQLException

- 동작: URL 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

registerOutParameter

void registerOutParameter(int parameterIndex, int sqlType) throws SQLException

- 동작: parameterIndex 위치에 있는 OUT 매개 변수를 sqlType으로 등록한다. 모든 out 매개 변수는 stored procedure가 실행되기 전에 등록되어야 한다. sqlType으로 지정된 out 매개 변수의 JDBC 타입은 매개 변수 값을 읽기 위해 get method에 사용되는 Java 타입을 결정한다. GOLDILOCKS 타입별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: parameterIndex가 유효하지 않거나, database에 접근할 수 없거나 CallableStatement가 이미 닫혀 있을 경우 SQLException이 발생한다. GOLDILOCKS가 지원하지 않는 타입에 대해서는 SQLFeatureNotSupportedException이 발생한다.

void registerOutParameter(int parameterIndex, int sqlType, int scale) throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

```
void registerOutParameter(int parameterIndex, int sqlType, String typeName) throws
SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

```
void registerOutParameter(String parameterName, int sqlType) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

```
void registerOutParameter(String parameterName, int sqlType, int scale) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

```
void registerOutParameter(String parameterName, int sqlType, String typeName) throws
SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setAsciiStream

```
void setAsciiStream(String parameterName, InputStream x) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

```
void setAsciiStream(String parameterName, InputStream x, int length) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

```
void setAsciiStream(String parameterName, InputStream x, long length) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setBigDecimal

`void setBigDecimal(String parameterName, BigDecimal x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setBinaryStream

`void setBinaryStream(String parameterName, InputStream x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setBinaryStream(String parameterName, InputStream x, int length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setBinaryStream(String parameterName, InputStream x, long length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setBlob

`void setBlob(String parameterName, Blob x) throws SQLException`

- 동작: Blob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`void setBlob(String parameterName, InputStream inputStream) throws SQLException`

- 동작: Blob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`void setBlob(String parameterName, InputStream inputStream, long length) throws SQLException`

- 동작: Blob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

setBoolean

`void setBoolean(String parameterName, boolean x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setByte

`void setByte(String parameterName, byte x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setBytes

`void setBytes(String parameterName, byte[] x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setCharacterStream

`void setCharacterStream(String parameterName, Reader reader) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setCharacterStream(String parameterName, Reader reader, int length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setCharacterStream(String parameterName, Reader reader, long length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setClob

`void setClob(String parameterName, Clob x) throws SQLException`

- 동작: Clob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`void setClob(String parameterName, Reader reader) throws SQLException`

- 동작: Clob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`void setClob(String parameterName, Reader reader, long length) throws SQLException`

- 동작: Clob 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

setDate

`void setDate(String parameterName, Date x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setDate(String parameterName, Date x, Calendar cal) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setDouble

`void setDouble(String parameterName, double x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setFloat

`void setFloat(String parameterName, float x) throws SQLException`

- 동작: 구현되어 있지 않다.

- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setInt

`void setInt(String parameterName, int x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setLong

`void setLong(String parameterName, long x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNCharacterStream

`void setNCharacterStream(String parameterName, Reader value) throws SQLException`

- 동작: `NChar`는 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNCharacterStream(String parameterName, Reader value, long length) throws SQLException`

- 동작: `NChar`는 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNClob

`void setNClob(String parameterName, NClob value) throws SQLException`

- 동작: `NClob`은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNClob(String parameterName, Reader reader) throws SQLException`

- 동작: `NClob`은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNClob(String parameterName, Reader reader, long length) throws SQLException`

- 동작: NClob은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNString

`void setNString(String parameterName, String value) throws SQLException`

- 동작: NChar는 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNull

`void setNull(String parameterName, int sqlType) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNull(String parameterName, int sqlType, String typeName) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setObject

`void setObject(String parameterName, Object x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setObject(String parameterName, Object x, int targetSqlType) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setObject(String parameterName, Object x, int targetSqlType, int scale) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setRowId

`void setRowId(String parameterName, RowId x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setShort

`void setShort(String parameterName, short x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setSQLXML

`void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException`

- 동작: SQLXML 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setString

`void setString(String parameterName, String x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setTime

`void setTime(String parameterName, Time x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setTime(String parameterName, Time x, Calendar cal) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setTimestamp

`void setTimestamp(String parameterName, Timestamp x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setURL

`void setURL(String parameterName, URL val) throws SQLException`

- 동작: URL 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

wasNull

`boolean wasNull()`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface)`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

unwrap

`boolean isWrapperFor(Class<?> iface)`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

Clob

클래스가 구현되지 않았다.

free

`void free() throws SQLException`

getAsciiStream

`InputStream getAsciiStream() throws SQLException`

getCharacterStream

`Reader getCharacterStream() throws SQLException`

`Reader getCharacterStream(long pos, long length) throws SQLException`

getSubString

`String getSubString(long pos, int length) throws SQLException`

length

`long length() throws SQLException`

position

`long position(Clob searchstr, long start) throws SQLException`

`long position(String searchstr, long start) throws SQLException`

setAsciiStream

| OutputStream setAsciiStream(long pos) throws SQLException

setCharacterStream

| Writer setCharacterStream(long pos) throws SQLException

setString

| int setString(long pos, String str) throws SQLException

| int setString(long pos, String str, int offset, int len) throws SQLException

truncate

| void truncate(long len) throws SQLException

CommonDataSource

getLoginTimeout

`int getLoginTimeout() throws SQLException`

- 동작: 설정된 login timeout 값을 반환한다. Login timeout은 서버와 소켓을 연결할 때 timeout 값으로 사용된다. 설정되지 않은 경우, 0을 반환한다. 0은 무한 대기를 의미한다.
- 예외: 발생하지 않는다.

getLogWriter

`PrintWriter getLogWriter() throws SQLException`

- 동작: 이 DataSource에 설정된 log writer를 반환한다. 설정되지 않은 경우 null을 반환한다. Log writer는 각종 trace log를 기록할 PrintWriter를 의미한다. 자세한 내용은 **로깅**을 참조한다.
- 예외: 발생하지 않는다.

setLoginTimeout

`void setLoginTimeout(int seconds) throws SQLException`

- 동작: Login timeout 값을 설정한다. Login timeout은 서버와 소켓을 연결할 때 timeout 값으로 사용된다. 0은 무한 대기를 의미한다.
- 예외: 발생하지 않는다.

setLogWriter

`void setLogWriter(PrintWriter out) throws SQLException`

- 동작: 이 DataSource에 log writer를 설정한다. 설정하지 않으면 기본적으로 null이다. Log writer는 각종 trace log를 기록할 PrintWriter를 의미한다. 이 값을 설정하면 옵션에 따라 trace log, query log, protocol log를 기록하는데, 이 DataSource로부터 생성된 connection 객체와 그로부터 생성되는 모든 statement, ResultSet 등의 객체들이 로깅을 수행한다. Trace log, query log, protocol log 옵션은 connection URL이나 property에 명시할 수 있다. 자세한 내용은 **로깅**을 참조한다.
- 예외: 발생하지 않는다.

setDataSourceName

```
void setDataSourceName(String aDataSourceName)
```

- 동작: Data source 이름을 설정한다. 연결에 필요한 정보는 아니다. 단지 객체를 구별하기 위해 별도로 부여할 수 있는 정보이다.
- 예외: 발생하지 않는다.

setServerName

```
void setServerName(String aServerName)
```

- 동작: 서버 이름, 즉 연결 URL을 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

setDatabaseName

```
void setDatabaseName(String aDBName)
```

- 동작: 데이터베이스 이름을 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

setNetworkProtocol

```
void setNetworkProtocol(String aProtocol)
```

- 동작: 네트워크 프로토콜 정보이다. 연결에 필요한 정보는 아니다.
- 예외: 발생하지 않는다.

setUser

```
void setUser(String aUser)
```

- 동작: 연결 계정 이름을 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

setPassword

```
void setPassword(String aPassword)
```

- 동작: 연결 계정 password를 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

setPortNumber

```
void setPortNumber(int aPort)
```

- 동작: 서버에 접속할 때 사용할 포트 번호를 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

```
void setPortNumber(String aPort)
```

- 동작: 서버에 접속할 때 사용할 포트 번호를 설정한다. 연결에 필요한 필수 정보이다.
- 예외: 발생하지 않는다.

setRoleName

```
void setRoleName(String aRoleName)
```

- 동작: 서버에 접속할 때의 role을 명시한다. 연결에 필요한 필수 정보는 아니지만 연결 관련 정보이다. ""나 "ADMIN", "SYSDBA" 중의 하나를 명시한다
- 예외: 발생하지 않는다.

setDescription

```
void setDescription(String aDescription)
```

- 동작: 이 data source에 대한 설명을 설정한다. 연결에 사용되지 않는다.
- 예외: 발생하지 않는다.

setConnectionProperties

```
void setConnectionProperties(Properties aProps)
```

- 동작: 각종 연결에 사용될 수 있는 속성을 정의한다.
- 예외: 발생하지 않는다.

setURL

`void setURL(String aURL) throws SQLException`

- 동작: URL 형태로 `serverName`, `portNumber`, `databaseName`을 지정해준다. `DriverManager`를 통해서 연결할 때와 똑같은 URL을 취한다.
- 예외: 잘못된 형식이면 `SQLException`이 발생한다.

`void setUrl(String aUrl) throws SQLException`

`setURL (String aURL)`과 동일하다.

setLogTarget

`void setLogTarget(String aTarget)`

- 동작: `setLogWriter`를 호출할 수 없을 경우 이 method를 통해 log writer를 설정할 수 있다. 현재는 `aTarget`이 "console"인 경우에만 지원된다. 다른 값은 무시된다. "console"로 설정하면 이 `DataSource`로부터 생성되는 모든 connection 객체 이하의 로깅들이 콘솔로 출력된다.
- 예외: 발생하지 않는다.

setTraceLog

`void setTraceLog(String aMode)`

- 동작: Trace log를 설정한다. `aMode`가 on 이면 trace logging이 켜진다.
- 예외: 발생하지 않는다.

setQueryLog

`void setQueryLog(String aMode)`

- 동작: Query log를 설정한다. `aMode`가 on 이면 query logging이 켜진다.
- 예외: 발생하지 않는다.

setProtocolLog

`void setProtocolLog(String aMode)`

- 동작: Protocol log를 설정한다. `aMode`가 on 이면 protocol logging이 켜진다.

- 예외: 발생하지 않는다.

Connection

clearWarnings

`void clearWarnings() throws SQLException`

- 동작: 현재 connection 객체가 가지고 있는 warning 객체(들)을 제거한다.
- 예외: 발생하지 않는다.

close

`void close() throws SQLException`

- 동작: 현재 connection을 더 이상 사용하지 않기 위해 GOLDBLOCKS와의 연결을 끊고 이 객체로부터 만든 모든 statement 객체를 close한다. 이미 close되어 있으면 아무런 동작도 하지 않는다.
- 예외: 서버에서 에러가 발생하거나 응답이 없는 경우, 예외가 발생할 수 있다.

commit

`void commit() throws SQLException`

- 동작: Non-auto commit 모드일 경우 현재 connection에 대해 commit을 수행한다.
- 예외: 이미 close되어 있거나 auto commit 모드일 경우 SQLException이 발생한다.

createArrayOf

`Array createArrayOf(String typeName, Object[] elements) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

createBlob

`Blob createBlob() throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

createClob

Clob createClob() throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

createNClob

NClob createNClob() throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

createSQLXML

SQLXML createSQLXML() throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

createStatement

Statement createStatement() throws SQLException

- 동작: Statement 객체를 생성한다. 이 statement로부터 생성되는 ResultSet의 타입은 ResultSet.TYPE_FORWARD_ONLY이며, concurrency는 ResultSet.CONCUR_READ_ONLY, holdability는 ResultSet.HOLD_CURSORS_OVER_COMMIT이다.
- 예외: 이미 close된 경우에는 SQLException이 발생한다.

Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException

- 동작: 사용자가 지정한 resultset type과 resultset concurrency를 가지는 ResultSet을 생성하는 statement 객체를 생성한다. 이 statement 객체로부터 생성되는 ResultSet의 holdability는 connection의 holdability 타입과 동일하다. Connection의 기본 holdability 값은 ResultSet.HOLD_CURSORS_OVER_COMMIT이다.
- 예외: 이미 close된 경우에는 SQLException이 발생한다.

Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException

- 동작: 사용자가 지정한 resultset type, resultset concurrency, holdability를 가지는 ResultSet을 생성하는 sta

tement 객체를 생성한다.

- 예외: 이미 close된 경우에는 SQLException이 발생한다.

createStruct

| Struct createStruct(String typeName, Object[] attributes) throws SQLException

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getAutoCommit

| boolean getAutoCommit() throws SQLException

- 동작: 현재 auto commit 모드를 반환한다. setAutoCommit()을 호출한 적이 없다면 true를 반환한다.
- 예외: 발생하지 않는다.

getCatalog

| String getCatalog() throws SQLException

- 동작: 데이터베이스의 현재 카탈로그 이름을 얻어온다.
- 예외: 서버로부터 에러가 발생하거나 응답이 없는 경우 SQLException이 발생한다.

getClientInfo

| Properties getClientInfo() throws SQLException

- 동작: 사용자가 설정한 client 정보를 반환한다. 설정한 정보가 없으면 null을 반환한다.
- 예외: 발생하지 않는다.

| String getClientInfo(String name) throws SQLException

- 동작: 사용자가 설정한 특정 client 정보를 반환한다. 설정한 정보가 없으면 null을 반환한다.
- 예외: 발생하지 않는다.

getHoldability

`int getHoldability() throws SQLException`

- 동작: 이 객체로부터 만들어지는 statement들의 기본 holdability 값을 반환한다. 기본값은 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 발생하지 않는다.

getMetaData

`DatabaseMetaData getMetaData() throws SQLException`

- 동작: 이 객체로부터 메타 정보를 조회할 수 있는 `DatabaseMetaData` 객체를 얻는다. 항상 동일한 객체가 반환된다.
- 예외: 이미 close되어 있을 경우 `SQLException`이 발생한다.

getTransactionIsolation

`int getTransactionIsolation() throws SQLException`

- 동작: 현재 세션 (connection)에 설정되어 있는 transaction isolation level을 얻는다. 서버에 설정된 기본값은 `Connection.TRANSACTION_READ_COMMITTED`이다.
- 예외: 서버로부터 에러를 응답받는 경우 `SQLException`이 발생한다.

getTypeMap

`Map<String,Class<?>> getTypeMap() throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getWarnings

`SQLWarning getWarnings() throws SQLException`

- 동작: 현재까지 connection 객체가 서버로부터 응답받은 warning을 반환한다. 존재하지 않거나 `clearWarnings()`를 한 후라면 null을 반환한다.
- 예외: 발생하지 않는다.

isClosed

boolean isClosed() throws SQLException

- 동작: close()를 성공적으로 호출한 적이 있는지 여부를 묻는다. close()를 성공적으로 수행했을 경우 true를 반환하고 그렇지 않다면 false를 반환한다.
- 예외: 발생하지 않는다.

노트

이 method는 실제 서버와의 연결이 끊겼는지 여부는 알려주지 않는다. 실제로 연결이 끊겼더라도 사용자가 close()를 호출한 적이 없다면 false를 반환한다.

isReadOnly

boolean isReadOnly() throws SQLException

- 동작: 이 세션 (connection)이 read-only 모드일 경우 true를, 그렇지 않을 경우 false를 반환한다. 기본값은 false이다. 서버와의 통신이 발생한다.
- 예외: 서버로부터 에러를 응답받거나 응답이 없는 경우 SQLException이 발생한다.

isValid

boolean isValid(int timeout) throws SQLException

- 동작: isClosed()가 true를 반환할 상태이거나 heart beat query를 서버로 보내 성공적으로 수행에 대한 응답을 받지 못할 경우 false를, 아닌 경우엔 true를 반환한다.
- 예외: 발생하지 않는다.

nativeSQL

String nativeSQL(String sql) throws SQLException

- 동작: 주어진 사용자 SQL 문에 대해 서버에서 인식하는 native SQL을 반환한다. GOLDILOCKS에서는 사용자의 SQL 문을 서버가 있는 그대로 인식하기 때문에 항상 사용자가 부여한 값 그대로를 반환한다.
- 예외: 발생하지 않는다.

prepareCall

`CallableStatement prepareCall(String sql)` throws `SQLException`

- 동작: Stored procedures 호출을 위한 `CallableStatement` 객체를 사용자에게 반환한다. 이 `CallableStatement`로부터 생성되는 `ResultSet`의 type은 `ResultSet.TYPE_FORWARD_ONLY`이고 concurrency는 `ResultSet.CONCUR_READ_ONLY`, holdability는 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 이미 close 되었거나 SQL 문이 잘못된 경우, `SQLException`이 발생한다.

`CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)` throws `SQLException`

- 동작: 사용자가 지정한 `resultSet type`과 `resultSet concurrency`를 가지는 `ResultSet`을 생성하는 `CallableStatement` 객체를 생성한다. 이 `CallableStatement`로부터 생성되는 `ResultSet`의 holdability는 `connection`의 holdability 타입과 동일하다. `Connection`의 기본 holdability 값은 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 이미 close되었거나 SQL 문이 잘못된 경우, `SQLException`이 발생한다.

`CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)` throws `SQLException`

- 동작: 사용자가 지정한 `resultSet type`과 `resultSet concurrency`, `holdability`를 가지는 `ResultSet`을 생성하는 `CallableStatement` 객체를 생성한다.
- 예외: 이미 close되었거나 SQL 문이 잘못된 경우, `SQLException`이 발생한다.

prepareStatement

`PreparedStatement prepareStatement(String sql)` throws `SQLException`

- 동작: 서버에 SQL 문을 보내어 prepare (parsing, validation, optimization)하고, 그 prepared statement를 제어할 수 있는 `PreparedStatement` 객체를 사용자에게 반환한다. 이 `PreparedStatement`로부터 생성되는 `ResultSet`의 type은 `ResultSet.TYPE_FORWARD_ONLY`이며, concurrency는 `ResultSet.CONCUR_READ_ONLY`, holdability는 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우, `SQLException`이 발생한다.

`PreparedStatement prepareStatement(String sql, int autoGeneratedKeys)` throws `SQLException`

- 동작: `autoGeneratedKeys`가 `Statement.NO_GENERATED_KEYS`일 경우 `prepareStatement(String sql)` method와 동일하게 동작한다. 그 외의 값에 대해서는 예외가 발생한다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우, 그리고 `autoGeneratedKeys` 값이 `Statement.NO_GENERATED_KEYS`도 아니고 `Statement.RETURN_GENERATED_KEYS`도 아닌 경우에는 `SQLException`이 발생한다. `autoGeneratedKeys`가 `Statement.RETURN_GENERATED_KEYS`인 경우에는 `SQLFeatureNotSupportedException`이 발생한다.

`PreparedStatement prepareStatement(String sql, int[] columnIndexes) throws SQLException`

- 동작: columnIndexes가 null일 경우 `prepareStatement(String sql)` method와 동일하게 동작한다. 그 외의 값에 대해서는 예외가 발생한다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우, `SQLException`이 발생한다. columnIndexes가 null이 아닐 경우, `SQLFeatureNotSupportedException`이 발생한다.

`PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency) throws SQLException`

- 동작: 서버에 SQL 문을 보내어 prepare (parsing, validation, optimization)하고, 그 prepared statement를 제어할 수 있는 `PreparedStatement` 객체를 사용자에게 반환한다. 사용자가 지정한 `resultset type`과 `resultset concurrency`를 가지는 `ResultSet`을 생성하는 `PreparedStatement` 객체를 생성한다. 이 `PreparedStatement`로부터 생성되는 `ResultSet`의 `holdability`는 `connection`의 `holdability` 타입과 동일하다. `Connection`의 기본 `holdability` 값은 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우, `SQLException`이 발생한다.

`PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException`

- 동작: 서버에 SQL 문을 보내어 prepare (parsing, validation, optimization)하고, 그 prepared statement를 제어할 수 있는 `PreparedStatement` 객체를 사용자에게 반환한다. 사용자가 지정한 `resultset type`과 `resultset concurrency`, `holdability`를 가지는 `ResultSet`을 생성하는 `PreparedStatement` 객체를 생성한다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우, `SQLException`이 발생한다.

`PreparedStatement prepareStatement(String sql, String[] columnNames) throws SQLException`

- 동작: columnNames가 null일 경우 `prepareStatement(String sql)` method와 동일하게 동작한다. 그 외의 값에 대해서는 예외가 발생한다.
- 예외: 이미 close되었거나 SQL 문이 잘못되었을 경우에는 `SQLException`이 발생한다. columnNames가 null이 아닐 경우, `SQLFeatureNotSupportedException`이 발생한다.

releaseSavepoint

`void releaseSavepoint(Savepoint savepoint) throws SQLException`

- 동작: 해당 `savepoint`를 서버에서 제거한다. 이 `savepoint` 이후의 `savepoint`들도 모두 제거된다.
- 예외: 이미 close되었거나 해당 `savepoint` 객체가 이미 해제되었거나 `GOLDILOCKS` `savepoint` 객체가 아닌 경우 `SQLException`이 발생한다.

rollback

`void rollback()` throws `SQLException`

- 동작: Non auto commit 모드일 경우 현재 트랜잭션을 rollback 한다.
- 예외: 이미 close되었거나 auto commit 모드일 경우, 또는 서버로부터 에러를 응답받은 경우 `SQLException`이 발생한다.

`void rollback(Savepoint savepoint)` throws `SQLException`

- 동작: Non auto commit 모드일 경우 현재 트랜잭션에 대해 해당 savepoint까지 partial rollback 한다.
- 예외: 이미 close되었거나 auto commit mode일 경우, 또는 서버로부터 에러를 응답받은 경우, 그리고 해당 savepoint가 유효하지 않을 경우 `SQLException`이 발생한다.

setAutoCommit

`void setAutoCommit(boolean autoCommit)` throws `SQLException`

- 동작: 현재 connection에 대한 auto commit 모드를 변경한다. 수행된 트랜잭션이 있고 non auto commit 모드에서 auto commit 모드로 변경될 경우 commit이 수행된다.
- 예외: Commit 과정에서 발생할 수 있는 예외와 동일하다.

setCatalog

`void setCatalog(String catalog)` throws `SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setClientInfo

`void setClientInfo(Properties properties)` throws `SQLException`

- 동작: 사용자가 지정한 client 정보를 설정한다. 기존에 있던 client 정보는 없어진다. 서버에 아무런 영향을 주지 않는다.
- 예외: 발생하지 않는다.

`void setClientInfo(String name, String value)` throws `SQLException`

- 동작: 사용자가 지정한 client 정보를 추가한다. 기존에 있던 client 정보는 그대로 유지된다.
- 예외: 발생하지 않는다.

setHoldability

`void setHoldability(int holdability) throws SQLException`

- 동작: 이 객체로부터 생성되는 statement들이 생성하는 ResultSet의 holdability를 설정한다. 이 method를 호출하지 않았을 때의 기본값은 ResultSet.HOLD_CURSORS_OVER_COMMIT이다.
- 예외: 발생하지 않는다.

setReadOnly

`void setReadOnly(boolean readOnly) throws SQLException`

- 동작: 현재 세션 (connection)의 read-only 속성을 설정한다. 기본값은 false이다.
- 예외: 이미 close되었거나 서버로부터 에러를 응답받으면 SQLException이 발생한다.

setSavepoint

`Savepoint setSavepoint() throws SQLException`

- 동작: non auto commit 모드일 경우 현재 트랜잭션에 대해 savepoint를 설정한다. savepoint 이름은 내부적으로 결정된다.
- 예외: 이미 close되었거나 auto commit 모드일 경우, 또는 서버로부터 에러를 응답받은 경우에는 SQLException이 발생한다.

`Savepoint setSavepoint(String name) throws SQLException`

- 동작: Non auto commit 모드일 경우 현재 트랜잭션에 대해 사용자가 지정한 이름의 savepoint를 설정한다.
- 예외: 이미 close되었거나 auto commit 모드일 경우, 또는 서버로부터 에러를 응답받은 경우 SQLException이 발생한다.

setTransactionIsolation

`void setTransactionIsolation(int level) throws SQLException`

- 동작: 현재 세션 (connection)에 대해 트랜잭션 isolation level을 변경한다. 지원되는 값은 Connection.TRANSACTION_READ_COMMITTED, Connection.TRANSACTION_SERIALIZABLE이다. Connection.READ_UNCOMMITTED는 Connection.TRANSACTION_READ_COMMITTED으로, Connection.TRANSACTION_REPEATABLE_READ는 Connection.TRANSACTION_SERIALIZABLE으로 변경되어 설정된다.
- 예외: 이미 close되었거나 level이 올바르지 않을 경우, SQLException이 발생한다.

setTypeMap

`void setTypeMap(Map<String,Class<?>> map) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface) throws SQLException`

- 동작: 이 객체가 `iface` 인터페이스를 구현한 클래스인지 묻는다. 맞으면 `true`를, 아니면 `false`를 반환한다. `GOLD ILLOCKS` connection 객체는 다른 클래스의 wrapper가 아니므로 wrapper 유무는 판단하지 않고 오직 주어진 인자 클래스 타입을 구현했는지 여부만 묻는다.
- 예외: 발생하지 않는다.

unwrap

`<T> T unwrap(Class<T> iface) throws SQLException`

- 동작: `GOLD ILLOCKS` connection은 다른 클래스의 wrapper가 아니므로 unwrap하더라도 결국 자기 자신을 반환한다. 해당 타입으로 캐스팅한 후 반환한다. `iface`가 `isWrapperFor()` method의 인자로 주었을 때 `false`를 반환하는 값일 경우, 이 method는 예외를 발생시킨다.
- 예외: `iface`가 이 객체 타입이 아닌 경우 (이 객체가 implement 하지 않은 타입을 부여한 경우) `SQLException`이 발생한다.

ConnectionPoolDataSource

getPooledConnection

`PooledConnection getPooledConnection()` throws `SQLException`

- 동작: `PooledConnection` 객체를 생성한다. User name, password, connection URL 등 각종 connection 정보는 사전에 setter method로 설정되어 있어야 한다.
- 예외: 연결에 실패하면 `SQLException`이 발생한다.

`PooledConnection getPooledConnection(String user, String password)` throws `SQLException`

- 동작: `PooledConnection` 객체를 생성한다. User name과 password는 argument를 따른다. 그 외 connection 정보들은 사전에 setter method로 설정되어 있어야 한다.
- 예외: 연결에 실패하면 `SQLException`이 발생한다.

DatabaseMetaData

allProceduresAreCallable

`boolean allProceduresAreCallable() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

allTablesAreSelectable

`boolean allTablesAreSelectable() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

autoCommitFailureClosesAllResultSets

`boolean autoCommitFailureClosesAllResultSets() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

dataDefinitionCausesTransactionCommit

`boolean dataDefinitionCausesTransactionCommit() throws SQLException`

- 동작: DDL 구문을 수행할 때 자동으로 commit되지 않기 때문에 항상 false를 반환한다.
- 예외: 발생하지 않는다.

dataDefinitionIgnoredInTransactions

`boolean dataDefinitionIgnoredInTransactions() throws SQLException`

- 동작: DDL 구문도 트랜잭션에 포함되기 때문에 항상 false를 반환한다.
- 예외: 발생하지 않는다.

deletesAreDetected

`boolean deletesAreDetected(int type) throws SQLException`

- 동작: type이 `ResultSet.TYPE_SCROLL_SENSITIVE`이면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: 발생하지 않는다.

doesMaxRowSizeIncludeBlobs

`boolean doesMaxRowSizeIncludeBlobs() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

getAttributes

`ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern) throws SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getBestRowIdentifier

`ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable) throws SQLException`

- 동작: 모든 테이블에 대해서 rowid 타입을 지원하기 때문에 rowid에 대한 정보로 구성된 row 한 개를 가진 `ResultSet`을 반환한다. 해당 테이블이 없을 경우, empty `ResultSet`을 반환한다.
- 예외: 테이블이 null이거나 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getCatalogs

`ResultSet getCatalogs() throws SQLException`

- 동작: 카탈로그 이름을 하나의 column으로 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getCatalogSeparator

`String getCatalogSeparator()` throws `SQLException`

- 동작: Catalog separator 문자를 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getCatalogTerm

`String getCatalogTerm()` throws `SQLException`

- 동작: Catalog term 문자를 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getClientInfoProperties

`ResultSet getClientInfoProperties()` throws `SQLException`

- 동작: Client 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getColumnPrivileges

`ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)` throws `SQLException`

- 동작: 해당 테이블 column의 column privilege 정보를 가지는 `ResultSet`을 반환한다. 해당 테이블이 없거나 column 이름 패턴에 해당하는 column이 없으면 empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getColumns

`ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)` throws `SQLException`

- 동작: 해당 테이블의 모든 column 정보를 가지는 `ResultSet`을 반환한다. 해당 테이블이 없거나 column 이름 패턴에 해당하는 column이 없으면 empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getConnection

`Connection getConnection()` throws `SQLException`

- 동작: 이 `DatabaseMetaData` 객체를 생성한 연결 객체를 반환한다.
- 예외: 발생하지 않는다.

getCrossReference

`ResultSet getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)` throws `SQLException`

- 동작: 주어진 foreign key 테이블에서 주어진 parent table을 참조하는 foreign key들의 정보를 가지는 `ResultSet`을 반환한다. 참조 관계가 없을 경우, empty `ResultSet`을 반환한다.
- 예외: Foreign table이나 parent table이 null이거나 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDatabaseMajorVersion

`int getDatabaseMajorVersion()` throws `SQLException`

- 동작: 제품의 major 버전을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDatabaseMinorVersion

`int getDatabaseMinorVersion()` throws `SQLException`

- 동작: 제품의 minor 버전을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDatabaseProductName

`String getDatabaseProductName()` throws `SQLException`

- 동작: 제품의 이름을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDatabaseProductVersion

`String getDatabaseProductVersion()` throws `SQLException`

- 동작: 제품의 프로덕트 버전을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDefaultTransactionIsolation

`int getDefaultTransactionIsolation()` throws `SQLException`

- 동작: Default transaction isolation을 반환한다. 서버에 설정된 기본값은 `Connection.TRANSACTION_READ_COMMITTED`이다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getDriverMajorVersion

`int getDriverMajorVersion()` throws `SQLException`

- 동작: GOLDILOCKS JDBC driver의 major 버전을 반환한다.
- 예외: 발생하지 않는다.

getDriverMinorVersion

`int getDriverMinorVersion()` throws `SQLException`

- 동작: GOLDILOCKS JDBC driver의 minor 버전을 반환한다.
- 예외: 발생하지 않는다.

getDriverName

`String getDriverName()` throws `SQLException`

- 동작: "GOLDILOCKS JDBC Driver"를 반환한다.
- 예외: 발생하지 않는다.

getDriverVersion

`String getDriverVersion()` throws `SQLException`

- 동작: GOLDILOCKS JDBC driver 버전 문자열을 반환한다. 프로토콜 버전도 포함한다.
- 예외: 발생하지 않는다.

getExportedKeys

`ResultSet getExportedKeys(String catalog, String schema, String table)` throws `SQLException`

- 동작: 주어진 테이블의 column을 참조하는 foreign key들을 정보로 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getExtraNameCharacters

`String getExtraNameCharacters()` throws `SQLException`

- 동작: "-\$"를 반환한다.
- 예외: 발생하지 않는다.

getFunctionColumns

`ResultSet getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)` throws `SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getFunctions

`ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)` throws `SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getIdentifierQuoteString

`String getIdentifierQuoteString()` throws `SQLException`

- 동작: Identifier quote 문자를 반환한다. 서버에 설정된 값은 " 이다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getImportedKeys

`ResultSet getImportedKeys(String catalog, String schema, String table)` throws `SQLException`

- 동작: 주어진 테이블의 foreign key column이 참조하는 parent key 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getIndexInfo

`ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)` throws `SQLException`

- 동작: 주어진 테이블에 존재하는 인덱스 정보를 가지는 `ResultSet`을 반환한다. Unique가 true면 unique index 정보만 보여준다. Approximate 인자는 무시된다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getJDBCMajorVersion

`int getJDBCMajorVersion()` throws `SQLException`

- 동작: GOLDILOCKS JDBC driver의 JDBC major 버전을 반환한다. 사용하는 jar 파일에 따라 다를 수 있다.
- 예외: 발생하지 않는다.

getJDBCMinorVersion

`int getJDBCMinorVersion()` throws `SQLException`

- 동작: GOLDILOCKS JDBC driver의 JDBC minor 버전을 반환한다. 사용하는 jar 파일에 따라 다를 수 있다.
- 예외: 발생하지 않는다.

getMaxBinaryLiteralLength

`int getMaxBinaryLiteralLength() throws SQLException`

- 동작: 바이너리의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxCatalogNameLength

`int getMaxCatalogNameLength() throws SQLException`

- 동작: 카탈로그 이름의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxCharLiteralLength

`int getMaxCharLiteralLength() throws SQLException`

- 동작: 최대 literal 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnNameLength

`int getMaxColumnNameLength() throws SQLException`

- 동작: Column 이름의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnsInGroupBy

`int getMaxColumnsInGroupBy() throws SQLException`

- 동작: Group by 절에 사용할 수 있는 column의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnsInIndex

`int getMaxColumnsInIndex() throws SQLException`

- 동작: 인덱스에 사용할 수 있는 column의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnsInOrderBy

`int getMaxColumnsInOrderBy() throws SQLException`

- 동작: Order by 절에 사용할 수 있는 column의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnsInSelect

`int getMaxColumnsInSelect() throws SQLException`

- 동작: Select target 절에 올 수 있는 column의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxColumnsInTable

`int getMaxColumnsInTable() throws SQLException`

- 동작: 테이블에 올 수 있는 column의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxConnections

`int getMaxConnections() throws SQLException`

- 동작: 서버와의 최대 connection 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxCursorNameLength

`int getMaxCursorNameLength() throws SQLException`

- 동작: 커서 이름의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxIndexLength

`int getMaxIndexLength() throws SQLException`

- 동작: 인덱스 키 한 개가 가질 수 있는 최대 크기를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

주의

JDBC 스펙에는 하나의 인덱스 전체가 가질 수 있는 최대 크기를 바이트로 반환한다고 정의하고 있지만 인덱스 크기에는 제한이 없으므로 그 값은 의미가 없고 ODBC와 동일한 의미가 되도록 하기 위해 이와 같이 작동한다.

getMaxProcedureNameLength

`int getMaxProcedureNameLength() throws SQLException`

- 동작: Procedure 이름의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxRowSize

`int getMaxRowSize() throws SQLException`

- 동작: 한 테이블이 가질 수 있는 최대 row 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxSchemaNameLength

`int getMaxSchemaNameLength() throws SQLException`

- 동작: 스키마 이름의 최대 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxStatementLength

`int getMaxStatementLength() throws SQLException`

- 동작: 하나의 SQL 문이 가질 수 있는 최대 문자열 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxStatements

`int getMaxStatements() throws SQLException`

- 동작: 한 번에 open할 수 있는 최대 statement 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxTableNameLength

`int getMaxTableNameLength() throws SQLException`

- 동작: 테이블 이름이 가질 수 있는 최대 문자열의 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxTablesInSelect

`int getMaxTablesInSelect() throws SQLException`

- 동작: Select 구문에 사용할 수 있는 테이블의 최대 개수를 서버로부터 얻어온다. 0은 최대 개수가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getMaxUserNameLength

`int getMaxUserNameLength() throws SQLException`

- 동작: User name의 최대 문자열 길이를 서버로부터 얻어온다. 0은 최대 길이가 무한대라는 의미이다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getNumericFunctions

`String getNumericFunctions() throws SQLException`

- 동작: SQL 표준에 해당하는 숫자 관련 함수들의 목록을 불러온다. 각 함수 이름은 콤마 (,)로 구분된다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getPrimaryKeys

`ResultSet getPrimaryKeys(String catalog, String schema, String table) throws SQLException`

- 동작: 주어진 테이블의 모든 primary key를 포함하는 ResultSet을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getProcedureColumns

`ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern) throws SQLException`

- 동작: 주어진 procedure들에 대해 주어진 이름 패턴을 가진 column들의 정보를 가지는 ResultSet을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getProcedures

`ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern) throws SQLException`

- 동작: 주어진 이름 패턴을 가지는 procedure들을 정보로 가지는 ResultSet을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

getProcedureTerm

`String getProcedureTerm()` throws `SQLException`

- 동작: Procedure를 지칭하는 키워드를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getResultSetHoldability

`int getResultSetHoldability()` throws `SQLException`

- 동작: `ResultSet`의 기본 holdability 속성을 반환한다. `ResultSet.HOLD_CURSORS_OVER_COMMIT`이다.
- 예외: 발생하지 않는다.

getRowIdLifetime

`RowIdLifetime getRowIdLifetime()` throws `SQLException`

- 동작: 항상 `RowIdLifetime.ROWID_VALID_FOREVER`를 반환한다.
- 예외: 발생하지 않는다.

getSchemas

`ResultSet getSchemas()` throws `SQLException`

- 동작: 서버에 있는 모든 스키마에 대한 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

`ResultSet getSchemas(String catalog, String schemaPattern)` throws `SQLException`

- 동작: 주어진 스키마 이름의 패턴을 만족하는 모든 스키마에 대한 정보를 가지는 `ResultSet`을 반환한다. `Catalog`는 무시한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSchemaTerm

`String getSchemaTerm()` throws `SQLException`

- 동작: 스키마를 지칭하는 키워드를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSearchStringEscape

`String getSearchStringEscape()` throws `SQLException`

- 동작: Like 구문에 사용되는 escape 문자들을 문자열로 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSQLKeywords

`String getSQLKeywords()` throws `SQLException`

- 동작: SQL 문에 사용할 수 없는 키워드들을 콤마 (,)로 구분하여 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSQLStateType

`int getSQLStateType()` throws `SQLException`

- 동작: 항상 `DatabaseMetaData.sqlStateSQL99`를 반환한다.
- 예외: 발생하지 않는다.

getStringFunctions

`String getStringFunctions()` throws `SQLException`

- 동작: SQL 표준에 해당하는 문자열을 다루는 함수들의 목록을 불러온다. 각 함수 이름들은 콤마 (,)로 구분된다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSuperTables

`ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern)` throws `SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSuperTypes

`ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)` throws `SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getSystemFunctions

`String getSystemFunctions()` throws `SQLException`

- 동작: SQL 표준에 해당하는 시스템 함수들의 목록을 불러온다. 각 함수 이름들은 콤마 (,)로 구분된다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getTablePrivileges

`ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)` throws `SQLException`

- 동작: 해당 조건을 만족하는 모든 테이블의 `privilege` 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getTables

`ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)` throws `SQLException`

- 동작: 해당 조건을 만족하는 모든 테이블들의 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getTableTypes

`ResultSet getTableTypes()` throws `SQLException`

- 동작: 모든 테이블 타입을 정보로 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getTimeDateFunctions

`String getTimeDateFunctions()` throws `SQLException`

- 동작: SQL 표준에 해당하는 time, date 관련 함수들의 목록을 불러온다. 각 함수 이름들은 콤마 (,)로 구분된다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getTypeInfo

`ResultSet getTypeInfo()` throws `SQLException`

- 동작: 데이터 타입 정보를 가지는 `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getUDTs

`ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)`
throws `SQLException`

- 동작: Empty `ResultSet`을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

getURL

`String getURL()` throws `SQLException`

- 동작: 서버에 접속할 때 사용한 URL을 반환한다.
- 예외: 발생하지 않는다.

getUserName

`String getUserName()` throws `SQLException`

- 동작: 세션을 유지하고 있는 현재 user name을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

주의

세션 중에 user를 변경할 수 있기 때문에 처음 접속했을 때의 user name과 다를 수도 있다.

getVersionColumns

ResultSet getVersionColumns(String catalog, String schema, String table) throws SQLException

- 동작: Empty ResultSet을 반환한다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

insertsAreDetected

boolean insertsAreDetected(int type) throws SQLException

- 동작: Type과 상관없이 항상 false를 반환한다.
- 예외: 발생하지 않는다.

isCatalogAtStart

boolean isCatalogAtStart() throws SQLException

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

isReadOnly

boolean isReadOnly() throws SQLException

- 동작: 현재 연결이 read-only 모드인지 여부에 대한 정보를 서버로부터 얻어온다.
- 서버로부터 에러를 응답받으면 SQLException이 발생한다.

locatorsUpdateCopy

boolean locatorsUpdateCopy() throws SQLException

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

nullPlusNonNullsNull

`boolean nullPlusNonNullIsNull()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

nullsAreSortedAtEnd

`boolean nullsAreSortedAtEnd()` throws `SQLException`

- 동작: 항상 `false`를 반환한다. `null`만 따로 정렬하지 않는다.
- 예외: 발생하지 않는다.

nullsAreSortedAtStart

`boolean nullsAreSortedAtStart()` throws `SQLException`

- 동작: 항상 `false`를 반환한다. `null`만 따로 정렬하지 않는다.
- 예외: 발생하지 않는다.

nullsAreSortedHigh

`boolean nullsAreSortedHigh()` throws `SQLException`

- 동작: 항상 `true`를 반환한다. `Null`은 default로 last에 위치한다.
- 예외: 발생하지 않는다.

nullsAreSortedLow

`boolean nullsAreSortedLow()` throws `SQLException`

- 동작: 항상 `false`를 반환한다. `Null`은 default로 last에 위치한다.
- 예외: 발생하지 않는다.

othersDeletesAreVisible

`boolean othersDeletesAreVisible(int type) throws SQLException`

- 동작: Type이 `ResultSet.TYPE_SCROLL_SENSITIVE`일 경우 `true`를, 그렇지 않은 경우에는 `false`를 반환한다.
- 예외: 발생하지 않는다.

othersInsertsAreVisible

`boolean othersInsertsAreVisible(int type) throws SQLException`

- 동작: Type과 상관없이 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

othersUpdatesAreVisible

`boolean othersUpdatesAreVisible(int type) throws SQLException`

- 동작: Type이 `ResultSet.TYPE_SCROLL_SENSITIVE`일 경우 `true`를, 그렇지 않은 경우에는 `false`를 반환한다.
- 예외: 발생하지 않는다.

ownDeletesAreVisible

`boolean ownDeletesAreVisible(int type) throws SQLException`

- 동작: Type이 `ResultSet.TYPE_SCROLL_SENSITIVE`일 경우 `true`를, 그렇지 않은 경우에는 `false`를 반환한다.
- 예외: 발생하지 않는다.

ownInsertsAreVisible

`boolean ownInsertsAreVisible(int type) throws SQLException`

- 동작: Type과 상관없이 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

ownUpdatesAreVisible

`boolean ownUpdatesAreVisible(int type) throws SQLException`

- 동작: Type이 `ResultSet.TYPE_SCROLL_SENSITIVE`일 경우 `true`를, 그렇지 않은 경우에는 `false`를 반환한다.
- 예외: 발생하지 않는다.

storesLowerCaseIdentifiers

`boolean storesLowerCaseIdentifiers() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

storesLowerCaseQuotedIdentifiers

`boolean storesLowerCaseQuotedIdentifiers() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

storesMixedCaseIdentifiers

`boolean storesMixedCaseIdentifiers() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

storesMixedCaseQuotedIdentifiers

`boolean storesMixedCaseQuotedIdentifiers() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

storesUpperCaseIdentifiers

`boolean storesUpperCaseIdentifiers() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

storesUpperCaseQuotedIdentifiers

`boolean storesUpperCaseQuotedIdentifiers() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsAlterTableWithAddColumn`

`boolean supportsAlterTableWithAddColumn() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsAlterTableWithDropColumn`

`boolean supportsAlterTableWithDropColumn() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsANSI92EntryLevelSQL`

`boolean supportsANSI92EntryLevelSQL() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsANSI92FullSQL`

`boolean supportsANSI92FullSQL() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsANSI92IntermediateSQL`

`boolean supportsANSI92IntermediateSQL() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsBatchUpdates

`boolean supportsBatchUpdates() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsCatalogsInDataManipulation

`boolean supportsCatalogsInDataManipulation() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

supportsCatalogsInIndexDefinitions

`boolean supportsCatalogsInIndexDefinitions() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

supportsCatalogsInPrivilegeDefinitions

`boolean supportsCatalogsInPrivilegeDefinitions() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

supportsCatalogsInProcedureCalls

`boolean supportsCatalogsInProcedureCalls() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

supportsCatalogsInTableDefinitions

`boolean supportsCatalogsInTableDefinitions() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsColumnAliasing`

`boolean supportsColumnAliasing() throws SQLException`

- 동작: Column aliasing 지원 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

`supportsConvert`

`boolean supportsConvert() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsConvert`

`boolean supportsConvert(int fromType, int toType) throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsCoreSQLGrammar`

`boolean supportsCoreSQLGrammar() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsCorrelatedSubqueries`

`boolean supportsCorrelatedSubqueries() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsDataDefinitionAndDataManipulationTransactions

`boolean supportsDataDefinitionAndDataManipulationTransactions()` throws `SQLException`

- 동작: 항상 `true`를 반환한다. 한 트랜잭션으로 DML과 DDL을 수행할 수 있다.
- 예외: 발생하지 않는다.

supportsDataManipulationTransactionsOnly

`boolean supportsDataManipulationTransactionsOnly()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsDifferentTableCorrelationNames

`boolean supportsDifferentTableCorrelationNames()` throws `SQLException`

- 동작: Table correlation 이름이 테이블 이름과 달라야 하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

supportsExpressionsInOrderBy

`boolean supportsExpressionsInOrderBy()` throws `SQLException`

- 동작: Order by 절에 수식이 사용될 수 있는지의 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

supportsExtendedSQLGrammar

`boolean supportsExtendedSQLGrammar()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsFullOuterJoins

`boolean supportsFullOuterJoins()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsGetGeneratedKeys`

`boolean supportsGetGeneratedKeys()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

`supportsGroupBy`

`boolean supportsGroupBy()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsGroupByBeyondSelect`

`boolean supportsGroupByBeyondSelect()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsGroupByUnrelated`

`boolean supportsGroupByUnrelated()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

`supportsIntegrityEnhancementFacility`

`boolean supportsIntegrityEnhancementFacility()` throws `SQLException`

- 동작: SQL 무결성 보강 기능을 지원하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

supportsLikeEscapeClause

`boolean supportsLikeEscapeClause() throws SQLException`

- 동작: Like 구문에 escape 절을 지원하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

supportsLimitedOuterJoins

`boolean supportsLimitedOuterJoins() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsMinimumSQLGrammar

`boolean supportsMinimumSQLGrammar() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsMixedCaseIdentifiers

`boolean supportsMixedCaseIdentifiers() throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: 발생하지 않는다.

supportsMixedCaseQuotedIdentifiers

`boolean supportsMixedCaseQuotedIdentifiers() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsMultipleOpenResults

`boolean supportsMultipleOpenResults() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsMultipleResultSets

`boolean supportsMultipleResultSets() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsMultipleTransactions

`boolean supportsMultipleTransactions() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsNamedParameters

`boolean supportsNamedParameters() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsNonNullableColumns

`boolean supportsNonNullableColumns() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsOpenCursorsAcrossCommit

`boolean supportsOpenCursorsAcrossCommit() throws SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsOpenCursorsAcrossRollback

`boolean supportsOpenCursorsAcrossRollback() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsOpenStatementsAcrossCommit

`boolean supportsOpenStatementsAcrossCommit() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsOpenStatementsAcrossRollback

`boolean supportsOpenStatementsAcrossRollback() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsOrderByUnrelated

`boolean supportsOrderByUnrelated() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsOuterJoins

`boolean supportsOuterJoins() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsPositionedDelete

`boolean supportsPositionedDelete()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsPositionedUpdate

`boolean supportsPositionedUpdate()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsResultSetConcurrency

`boolean supportsResultSetConcurrency(int type, int concurrency)` throws `SQLException`

- 동작: 모든 `ResultSet` 타입과 모든 `concurrency`에 대해 `true`를, 잘못된 인자에 대해서는 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsResultSetHoldability

`boolean supportsResultSetHoldability(int holdability)` throws `SQLException`

- 동작: `Holdability`가 `ResultSet.CLOSE_CURSORS_AT_COMMIT`이거나 `ResultSet.HOLD_CURSORS_OVER_COMMIT`이면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsResultSetType

`boolean supportsResultSetType(int type)` throws `SQLException`

- 동작: `Type`이 `ResultSet.TYPE_FORWARD_ONLY`이거나 `ResultSet.TYPE_SCROLL_INSENSITIVE` 또는 `ResultSet.TYPE_SCROLL_SENSITIVE`이면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsSavepoints

`boolean supportsSavepoints()` throws `SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSchemasInDataManipulation

`boolean supportsSchemasInDataManipulation() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSchemasInIndexDefinitions

`boolean supportsSchemasInIndexDefinitions() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSchemasInPrivilegeDefinitions

`boolean supportsSchemasInPrivilegeDefinitions() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSchemasInProcedureCalls

`boolean supportsSchemasInProcedureCalls() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSchemasInTableDefinitions

`boolean supportsSchemasInTableDefinitions() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSelectForUpdate

`boolean supportsSelectForUpdate()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsStatementPooling

`boolean supportsStatementPooling()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsStoredFunctionsUsingCallSyntax

`boolean supportsStoredFunctionsUsingCallSyntax()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

supportsStoredProcedures

`boolean supportsStoredProcedures()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsSubqueriesInComparisons

`boolean supportsSubqueriesInComparisons()` throws `SQLException`

- 동작: 항상 `true`를 반환한다.
- 예외: 발생하지 않는다.

supportsSubqueriesInExists

`boolean supportsSubqueriesInExists() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSubqueriesInIns

`boolean supportsSubqueriesInIns() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsSubqueriesInQuantifieds

`boolean supportsSubqueriesInQuantifieds() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsTableCorrelationNames

`boolean supportsTableCorrelationNames() throws SQLException`

- 동작: Table correlation 이름을 지원하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

supportsTransactionIsolationLevel

`boolean supportsTransactionIsolationLevel(int level) throws SQLException`

- 동작: 해당 트랜잭션의 isolation 레벨을 지원하는지 여부를 서버로부터 알아온다.
- 예외: 서버로부터 에러를 응답받으면 SQLException이 발생한다.

supportsTransactions

`boolean supportsTransactions() throws SQLException`

- 동작: 항상 true를 반환한다.
- 예외: 발생하지 않는다.

supportsUnion

`boolean supportsUnion()` throws `SQLException`

- 동작: Union 연산을 지원하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

supportsUnionAll

`boolean supportsUnionAll()` throws `SQLException`

- 동작: Union all 연산을 지원하는지 여부를 서버로부터 얻어온다.
- 예외: 서버로부터 에러를 응답받으면 `SQLException`이 발생한다.

updatesAreDetected

`boolean updatesAreDetected(int type)` throws `SQLException`

- 동작: Type이 `ResultSet.TYPE_SCROLL_SENSITIVE`이면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: 발생하지 않는다.

usesLocalFilePerTable

`boolean usesLocalFilePerTable()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

usesLocalFiles

`boolean usesLocalFiles()` throws `SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

DataSource

getConnection

`Connection getConnection()` throws `SQLException`

- 동작: 새로운 연결 객체를 열어 반환한다. 연결에 필요한 정보들은 별도의 비표준 method들로 사전에 설정되어 있어야 한다.
- 예외: 서버와의 연결에 실패하면 `SQLException`이 발생한다.

`Connection getConnection(String username, String password)` throws `SQLException`

- 동작: Username과 password로 새로운 연결 객체를 열어 반환한다. 연결에 필요한 그 외의 정보들은 별도의 비표준 method들로 사전에 설정되어 있어야 한다.
- 예외: 서버와의 연결에 실패하면 `SQLException`이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface)` throws `SQLException`

- 동작: 이 클래스는 다른 클래스의 wrapper로 구현하지 않았다. 이 객체가 iface의 instance이면 true를, 그렇지 않으면 false를 반환한다.
- 예외: 발생하지 않는다.

unwrap

`<T> T unwrap(Class<T> iface)` throws `SQLException`

- 동작: 이 클래스는 다른 클래스의 wrapper로 구현되지 않았기 때문에 this를 반환한다.
- 예외: 이 객체가 iface의 instance가 아니면 `SQLException`이 발생한다.

Driver

acceptsURL

`boolean acceptsURL(String url) throws SQLException`

- 동작: url이 null이 아니고 "jdbc:goldilocks:"로 시작하면 true를, 그렇지 않으면 false를 반환한다.
- 예외: 발생하지 않는다.

connect

`Connection connect(String url, Properties info) throws SQLException`

- 동작: 새로운 연결 객체를 생성하고 반환한다. url은 서버 주소, DB 이름, 포트를 포함하고 있어야 한다.
- 예외: 잘못된 url이거나 서버로부터의 연결에 성공하지 못하면 SQLException이 발생한다.

getMajorVersion

`int getMajorVersion() throws SQLException`

- 동작: GOLDILOCKS JDBC driver의 major 버전을 반환한다.
- 예외: 발생하지 않는다.

getMinorVersion

`int getMinorVersion() throws SQLException`

- 동작: GOLDILOCKS JDBC driver의 minor 버전을 반환한다.
- 예외: 발생하지 않는다.

getPropertyInfo

`DriverPropertyInfo[] getPropertyInfo(String url, Properties info) throws SQLException`

- 동작: GOLDILOCKS JDBC의 driver가 연결될 때 사용할 수 있는 속성의 목록을 얻어온다.
- 예외: 발생하지 않는다.

jdbcCompliant

`boolean jdbcCompliant() throws SQLException`

- 동작: 항상 `false`를 반환한다.
- 예외: 발생하지 않는다.

NClob

클래스가 구현되지 않았다.

free

`void free() throws SQLException`

getAsciiStream

`InputStream getAsciiStream() throws SQLException`

getCharacterStream

`Reader getCharacterStream() throws SQLException`

`Reader getCharacterStream(long pos, long length) throws SQLException`

getSubString

`String getSubString(long pos, int length) throws SQLException`

length

`long length() throws SQLException`

position

`long position(Clob searchstr, long start) throws SQLException`

`long position(String searchstr, long start) throws SQLException`

setAsciiStream

`OutputStream setAsciiStream(long pos) throws SQLException`

setCharacterStream

`Writer setCharacterStream(long pos) throws SQLException`

setString

`int setString(long pos, String str) throws SQLException`

`int setString(long pos, String str, int offset, int len) throws SQLException`

truncate

`void truncate(long len) throws SQLException`

ParameterMetaData

ParameterMetaData 객체는 PreparedStatement.getParameterMetaData()를 통해서 얻을 수 있는데, GOLDILOCKS JDBC의 ParameterMetaData는 in/ out 여부에 대한 정보 외에 타입 등에 관한 정보는 실제 데이터베이스의 정보를 이용하지 않고 기본 타입인 varchar를 기준으로 한다. Prepare된 후에 parameter에 대해 서버로부터 얻을 수 있는 정보는 in/ out 뿐이기 때문이다. 예를 들어, "select * from t1 where a=?"라는 질의문으로 prepare를 할 경우, 조건절에 사용된 parameter의 타입은 정해지지 않은 상태이다. 서버는 기본적으로 varchar 타입이라고 가정할 뿐이다.

getParameterClassName

String getParameterClassName(int param) throws SQLException

- 동작: java.lang.String을 반환한다. 해당 parameter 타입은 varchar로 간주한다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

getParameterCount

int getParameterCount() throws SQLException

- 동작: Parameter 개수를 반환한다. 질의문에 사용된 ?의 개수이기도 하다.
- 예외: 발생하지 않는다.

getParameterMode

int getParameterMode(int param) throws SQLException

- 동작: 해당 param 번째 parameter의 in/ out 모드를 반환한다. ParameterMetaData.parameterModeIn, ParameterMetaData.parameterModeInOut, ParameterMetaData.parameterModeOut, ParameterMetaData.parameterModeUnknown 중의 하나를 반환한다. 아직까지는 실제로 parameterModeUnknown 값이 반환되는 경우는 없다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

getParameterType

int getParameterType(int param) throws SQLException

- 동작: 모든 parameter에 대해 Types.VARCHAR를 반환한다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

getParameterTypeName

`String getParameterTypeName(int param) throws SQLException`

- 동작: 모든 parameter에 대해 "VARCHAR"를 반환한다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

getPrecision

`int getPrecision(int param) throws SQLException`

- 동작: 모든 parameter에 대해 4000을 반환한다. 기본적으로 parameter는 varchar(4000)으로 간주된다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

getScale

`int getScale(int param) throws SQLException`

- 동작: 모든 parameter에 대해 0을 반환한다. 기본적으로 parameter는 varchar(4000)으로 간주된다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

isNullable

`int isNullable(int param) throws SQLException`

- 동작: 항상 ParameterMetaData.parameterNullableUnknown을 반환한다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

isSigned

`boolean isSigned(int param) throws SQLException`

- 동작: 항상 false를 반환한다.
- 예외: param 값이 parameter 개수 범위를 벗어나면 SQLException이 발생한다.

isWrapperFor

boolean isWrapperFor(Class<?> iface) throws SQLException

- 동작: 이 객체가 iface의 인스턴스인지 묻고 인스턴스일 경우 true를, 그렇지 않으면 false를 반환한다.
- 예외: 발생하지 않는다.

unwrap

<T> T unwrap(Class<T> iface) throws SQLException

- 동작: 이 객체가 iface의 인스턴스일 경우, 이 객체를 iface 타입으로 캐스팅해 반환한다.
- 예외: 이 객체가 iface의 인스턴스가 아닐 경우, SQLException이 발생한다.

PooledConnection

addConnectionEventListener

`void addConnectionEventListener(ConnectionEventListener listener) throws SQLException`

- 동작: ConnectionEventListener 객체를 등록한다. 이후 이 PooledConnection으로부터 얻은 connection 객체 (logical connection)의 close()가 호출되거나 실제 connection이 끊기면 등록된 listener들에게로 ConnectionEvent를 발생시킨다.
- 예외: 발생하지 않는다.

addStatementEventListener

`void addStatementEventListener(StatementEventListener listener) throws SQLException`

- 동작: 아무런 동작을 하지 않는다. GOLDILOCKS JDBC driver는 이 method를 구현하지 않았다. Statement pooling 기능은 외부 미들웨어에 맡겨둔다.
- 예외: 발생하지 않는다.

close

`void close() throws SQLException`

- 동작: 이 객체가 가지고 있는 physical connection의 close를 호출한다.
- 예외: Physical connection의 close에서 발생할 수 있다.

getConnection

`Connection getConnection() throws SQLException`

- 동작: 이 객체가 가지고 있는 logical connection을 반환한다.
- 예외: 발생하지 않는다.

removeConnectionEventListener

`void removeConnectionEventListener(ConnectionEventListener listener) throws SQLException`

- 동작: 등록된 ConnectionEventListener를 제거한다. 이후 이 listener에게는 ConnectionEvent가 전달되지 않는다.

- 예외: 발생하지 않는다.

removeStatementEventListener

`void removeStatementEventListener(StatementEventListener listener) throws SQLException`

- 동작: 아무런 동작을 하지 않는다.
- 예외: 발생하지 않는다.

PreparedStatement

addBatch

`void addBatch() throws SQLException`

- 동작: 현재 바인딩한 데이터들을 batch job으로 등록한다. Batch job이 하나라도 등록되어 있으면 `execute()`, `executeUpdate()`, `executeQuery()`를 실행할 때 에러가 발생한다. Parameter가 하나라도 바인딩되어 있지 않으면 에러가 발생한다. `addBatch`한 후 `setXXX()` 계열의 method로 바인딩하지 않은 상태에서 다시 `addBatch`하면 이전에 바인딩한 값으로 batch job을 등록한다.
- 예외: 한 번도 바인딩되지 않은 parameter가 있으면 예외가 발생한다.

clearParameters

`void clearParameters() throws SQLException`

- 동작: 현재 바인딩한 데이터와 정보를 모두 제거한다. 단 batch job이 하나라도 등록되어 있으면 아무런 작업도 하지 않는다.
- 예외: 발생하지 않는다.

execute

`boolean execute() throws SQLException`

- 동작: 현재 parameter에 바인딩 된 데이터를 기반으로 prepare된 statement를 실행한다. 실행한 statement가 select 구문이면 true를, 그렇지 않으면 false를 반환한다. 즉 true가 반환되면 이 객체로부터 ResultSet을 얻을 수 있다.
- 예외: Batch job이 등록되었거나 바인딩 된 parameter가 부족하거나 execution 할 때 서버에서 에러가 발생하면 예외가 발생할 수 있다.

executeQuery

`ResultSet executeQuery() throws SQLException`

- 동작: 현재 parameter에 바인딩 된 데이터를 기반으로 prepare된 statement를 실행하고 fetch를 수행하여 ResultSet 객체를 생성하고 반환한다.
- 예외: Batch job이 등록되었거나, 바인딩 된 parameter가 부족하거나 execution 할 때 서버에서 에러가 발생하면 예외가 발생할 수 있다.

executeUpdate

`int executeUpdate() throws SQLException`

- 동작: 현재 parameter에 바인딩 된 데이터를 기반으로 prepare된 statement를 실행한다. 갱신된 레코드 수를 반환하는데, DDL 구문 등으로 갱신된 레코드가 없을 경우 0을 반환한다.
- 예외: Batch job이 등록되었거나, statement가 select 구문이거나, 바인딩 된 parameter가 부족하거나 서버에서 에러가 발생하면 예외가 발생할 수 있다.

getMetaData

`ResultSetMetaData getMetaData() throws SQLException`

- 동작: ResultSetMetaData 객체를 얻는다. Prepare된 구문이 select가 아닌 경우, 즉 ResultSet을 반환하는 구문이 아닌 경우 빈 ResultSetMetaData를 반환한다. 이 method는 execute 되기 전에 호출될 수 있다.
- 예외: 서버에서 에러가 발생하면 예외가 발생할 수 있다.

getParameterMetaData

`ParameterMetaData getParameterMetaData() throws SQLException`

- 동작: ParameterMetaData 객체를 얻는다. Execute 하기 전에 호출될 수 있다. 하지만 parameter들의 정확한 타입 관련 정보는 서버에서 알 수 없기 때문에 모든 parameter는 varchar(4000)으로 가정한다.
- 예외: 서버에서 에러가 발생하면 예외가 발생할 수 있다.

setArray

`void setArray(int parameterIndex, Array x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

setAsciiStream

`void setAsciiStream(int parameterIndex, InputStream x) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARCHAR 타입으로 바인딩한다. 바이너리 형태로 데이터를 입력하기 때문에 encoding 작업을 하지 않는다. 따라서 character set을 고려하지 않고 ascii 데이터라고 가정한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

주의

LONG VARCHAR로 바인딩되기 때문에 서버에서 VARCHAR로 변환 비용이 발생한다. 길이를 알 수 있으면 `void setAsciiStream(int parameterIndex, InputStream x, int length)`를 사용하는게 좋다.

`void setAsciiStream(int parameterIndex, InputStream x, int length) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARCHAR 타입으로 바인딩한다. 바이너리 형태로 데이터를 입력하기 때문에 encoding 작업을 하지 않는다. 따라서 character set을 고려하지 않고 ascii 데이터라고 가정한다. 영문 데이터를 입력하거나 서버, client character set 환경이 동일할 경우 varchar에 문자열을 입력하기 위해 이 method를 사용하는 것이 setCharacterStream이나 setString보다 빠르다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setAsciiStream(int parameterIndex, InputStream x, long length) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARCHAR 타입으로 바인딩한다. 바이너리 형태로 데이터를 입력하기 때문에 encoding 작업을 하지 않는다. 따라서 character set을 고려하지 않고 ascii 데이터라고 가정한다. 영문 데이터를 입력하거나 서버, client character set 환경이 동일할 경우 varchar에 문자열을 입력하기 위해 이 method를 사용하는 것이 setCharacterStream이나 setString보다 빠르다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setBigDecimal

`void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException`

- 동작: 해당 parameter 인덱스에 BigDecimal 객체를 NUMBER 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setBinaryStream

`void setBinaryStream(int parameterIndex, InputStream x) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARBINARY 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setBinaryStream(int parameterIndex, InputStream x, int length) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARBINARY 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setBinaryStream(int parameterIndex, InputStream x, long length) throws SQLException`

- 동작: 해당 parameter 인덱스에 InputStream 객체를 LONG VARBINARY 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setBlob

`void setBlob(int parameterIndex, Blob x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

`void setBlob(int parameterIndex, InputStream inputStream) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

`void setBlob(int parameterIndex, InputStream inputStream, long length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

setBoolean

`void setBoolean(int parameterIndex, boolean x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 BOOLEAN 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setByte

`void setByte(int parameterIndex, byte x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 NATIVE_SMALLINT 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setBytes

```
void setBytes(int parameterIndex, byte[] x) throws SQLException
```

- 동작: 해당 parameter 인덱스에 데이터 x를 VARBINARY 또는 LONG VARBINARY 타입으로 바인딩한다. x의 길이가 4000 이하이면 VARBINARY로, 4000보다 크면 LONG VARBINARY로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setCharacterStream

```
void setCharacterStream(int parameterIndex, Reader reader) throws SQLException
```

- 동작: 해당 parameter 인덱스에 reader 객체를 LONG VARCHAR 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

```
void setCharacterStream(int parameterIndex, Reader reader, int length) throws SQLException
```

- 동작: 해당 parameter 인덱스에 reader 객체를 LONG VARCHAR 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

```
void setCharacterStream(int parameterIndex, Reader reader, long length) throws SQLException
```

- 동작: 해당 parameter 인덱스에 reader 객체를 LONG VARCHAR 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setClob

```
void setClob(int parameterIndex, Clob x) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

```
void setClob(int parameterIndex, Reader reader, long length) throws SQLException
```

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

setDate

`void setDate(int parameterIndex, Date x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 DATE 타입으로 바인딩한다. `setDate(parameterIndex, x, Calendar.getInstance())`로 호출하는 것과 동일하다. 즉, local timezone을 적용한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 DATE 타입으로 바인딩한다. Date x는 cal의 timezone의 시간대로 간주한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setDouble

`void setDouble(int parameterIndex, double x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `NATIVE_DOUBLE` 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setFloat

`void setFloat(int parameterIndex, float x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `NATIVE_REAL` 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setInt

`void setInt(int parameterIndex, int x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `NATIVE_INTEGER` 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setLong

`void setLong(int parameterIndex, long x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `NATIVE_BIGINT` 타입으로 바인딩한다.

- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setNCharacterStream

`void setNCharacterStream(int parameterIndex, Reader value) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNCharacterStream(int parameterIndex, Reader value, long length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNClob

`void setNClob(int parameterIndex, NClob value) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNClob(int parameterIndex, Reader reader) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

`void setNClob(int parameterIndex, Reader reader, long length) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNString

`void setNString(int parameterIndex, String value) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setNull

`void setNull(int parameterIndex, int sqlType) throws SQLException`

- 동작: 해당 parameter 인덱스에 null을 sqlType에 해당하는 GOLDILOCKS 타입으로 바인딩한다. sqlType에 매핑되는 GOLDILOCKS 타입은 표 26-3 SQL 타입 → GOLDILOCKS 타입을 참조한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setNull(int parameterIndex, int sqlType, String typeName) throws SQLException`

- 동작: 해당 parameter 인덱스에 null을 sqlType에 해당하는 GOLDILOCKS 타입으로 바인딩한다. sqlType에 매핑되는 GOLDILOCKS 타입은 표 26-3 SQL 타입 → GOLDILOCKS 타입을 참조한다. REF나 사용자 타입은 지원하지 않기 때문에 세 번째 인자인 typeName은 무시된다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setObject

`void setObject(int parameterIndex, Object x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 매핑되는 해당 GOLDILOCKS 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

표 26-2 Java 객체 → GOLDILOCKS 타입

Java 클래스	GOLDILOCKS 타입
null	VARCHAR
Boolean	BOOLEAN
Byte	NATIVE_SMALLINT
Short	NATIVE_SMALLINT
Integer	NATIVE_INTEGER
Long	NATIVE_BIGINT
Float	NATIVE_REAL
Double	NATIVE_DOUBLE
BigInteger	NATIVE_BIGINT
BigDecimal	NUMBER
String	VARCHAR or LONG VARCHAR
byte[]	VARBINARY or LONG VARBINARY
Date	DATE
Time	TIME
Timestamp	TIMESTAMP
Blob	N/A
Clob	N/A

Java 클래스	GOLDILOCKS 타입
InputStream	LONG VARBINARY
Reader	LONG VARCHAR
GoldilocksInterval	해당 INTERVAL 타입
RowId	ROWID
그 외	N/A

`void setObject(int parameterIndex, Object x, int targetSqlType) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 targetSqlType에 해당하는 GOLDILOCKS 타입으로 바인딩한다. targetSqlType에 매핑되는 GOLDILOCKS 타입은 표 26-3 SQL 타입 → GOLDILOCKS 타입을 참조한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 targetSqlType에 해당하는 GOLDILOCKS 타입으로 바인딩한다. targetSqlType에 매핑되는 GOLDILOCKS 타입은 표 26-3 SQL 타입 → GOLDILOCKS 타입을 참조한다. x가 InputStream이나 reader인 경우 scaleOrLength는 데이터 길이를 나타낸다. 다른 타입에 대해서는 이 값이 무시된다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setRef

`void setRef(int parameterIndex, Ref x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

setRowId

`void setRowId(int parameterIndex, RowId x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 ROWID 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setShort

`void setShort(int parameterIndex, short x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `NATIVE_SMALLINT` 타입으로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setSQLXML

`void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setString

`void setString(int parameterIndex, String x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `VARCHAR` 또는 `LONG VARCHAR` 타입으로 바인딩한다. x의 길이가 4000 이하이면 `VARCHAR`로, 그보다 크면 `LONG VARCHAR`로 바인딩한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setTime

`void setTime(int parameterIndex, Time x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `TIME` 타입으로 바인딩한다. `setTime(parameterIndex, x, Calendar.getInstance())`로 호출하는 것과 동일하다. 즉, local timezone을 적용한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

`void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `TIME` 타입으로 바인딩한다. Time x는 cal의 timezone의 시간대로 간주한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

setTimestamp

`void setTimestamp(int parameterIndex, Timestamp x) throws SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `TIME` 타입으로 바인딩한다. `setTimestamp(parameterIndex, x, Calendar.getInstance())`로 호출하는 것과 동일하다. 즉, local timezone을 적용한다.

- 예외: `parameterIndex`가 0보다 작거나 `parameter` 개수보다 클 경우에 예외가 발생한다.

`void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException`

- 동작: 해당 `parameter` 인덱스에 데이터 `x`를 `TIMESTAMP` 타입으로 바인딩한다. `Timestamp x`는 `cal`의 `timeZone`의 시간대로 간주한다.
- 예외: `parameterIndex`가 0보다 작거나 `parameter` 개수보다 클 경우에 예외가 발생한다.

setUnicodeStream

`void setUnicodeStream(int parameterIndex, InputStream x, int length) throws SQLException`

- 동작: 구현되어 있지 않다. (Deprecated된 method이다)
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

setURL

`void setURL(int parameterIndex, URL x) throws SQLException`

- 동작: 구현되어 있지 않다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface) throws SQLException`

- 동작: 이 객체가 `iface` 인터페이스를 구현한 클래스인지 여부를 묻는다. 맞으면 `true`를, 그렇지 않으면 `false`를 반환한다. `GOLDILOCKS PreparedStatement` 객체는 다른 클래스의 wrapper가 아니므로 wrapper의 유무는 판단하지 않고 오직 주어진 인자 클래스 타입을 구현했는지 여부만 묻는다.
- 예외: 발생하지 않는다.

unwrap

`<T> T unwrap(Class<T> iface) throws SQLException`

- 동작: `GOLDILOCKS PreparedStatement`는 다른 클래스의 wrapper가 아니므로 `unwrap` 하더라도 결국 자기 자신을 반환한다. 해당 타입으로 캐스팅한 후에 반환한다. `iface`가 `isWrapperFor()` method의 인자로 주어졌을 때 `false`를 반환하는 값이라면 이 method는 예외를 발생시킨다.
- 예외: `iface`가 이 객체의 타입이 아닌 경우 (이 객체가 implements 하지 않은 타입을 준 경우) `SQLException`이 발생한다.

executeBatchAtomic

`boolean executeBatchAtomic()` throws `SQLException`

- 동작: `executeBatch()`와 같지만 `atomic`하게 수행된다. 즉 batch job이 모두 성공하거나 모두 실패하거나 둘 중 하나이다. `executeBatch()`보다 빠르게 수행된다. 수행한 구문이 `select` 구문이면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: Batch job이 등록되지 않았거나 서버에서 에러가 발생하면 `SQLException`이 발생한다.

노트

GOLDILOCKS JDBC의 고유한 기능으로써 `PreparedStatement` 객체를 `GoldilocksPreparedStatement`로 캐스팅 한 후에 사용할 수 있다.

예: `((GoldilocksPreparedStatement)pstmt).executeBatchAtomic();`

setTimeTimeZone

`void setTimeTimeZone(int parameterIndex, Time x, Calendar cal)` throws `SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `TIME WITH TIME ZONE` 타입으로 바인딩한다. Time x는 cal의 timezone의 시간대로 간주한다. 데이터베이스 column의 timezone 정보는 cal의 timezone을 참조한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

노트

GOLDILOCKS JDBC의 고유한 기능으로써 `PreparedStatement` 객체를 `GoldilocksPreparedStatement`로 캐스팅한 후에 사용할 수 있다.

예: `((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, aTime, aCalendar);`

setTimestampTimeZone

`void setTimestampTimeZone(int parameterIndex, Timestamp x, Calendar cal)` throws `SQLException`

- 동작: 해당 parameter 인덱스에 데이터 x를 `TIMESTAMP WITH TIME ZONE` 타입으로 바인딩한다. Timestamp x는 cal의 timezone의 시간대로 간주한다. 데이터베이스 column의 timezone 정보는 cal의 timezone을 참조한다.
- 예외: parameterIndex가 0보다 작거나 parameter 개수보다 클 경우에 예외가 발생한다.

노트

GOLDILOCKS JDBC의 고유한 기능으로써 PreparedStatement 객체를 GoldilocksPreparedStatement로 캐스팅 한 후에 사용할 수 있다.

예: ((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(1, aTimestamp, aCalendar);

Ref

클래스가 구현되지 않았다.

getBaseTypeName

| String getBaseTypeName() throws SQLException

getObject

| Object getObject() throws SQLException

| Object getObject(Map<String,Class<?>> map) throws SQLException

setObject

| void setObject(Object value) throws SQLException

ResultSet

absolute

`boolean absolute(int row) throws SQLException`

- 동작: Fetch한 row 커서 위치를 row 번째에 둔다. 첫 번째 row는 1이다. 0은 before first를 가리킨다. 음수이면 마지막 row부터 가리킨다. 즉 -1은 마지막 row, -2는 마지막에서 두 번째 row를 가리킨다. Fetch한 row 캐시 내에 커서를 위치시킬 수 있으면 캐시 내의 위치 정보만 바꾸고, 캐시 내에 위치시킬 수 없으면 서버로부터 다시 fetch 해온다. Row가 범위를 벗어나면 before first 또는 after last에 커서를 위치시키고 false를 반환한다. 그 외의 경우에는 해당 row에 위치시키고 true를 반환한다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 예외가 발생하면 예외가 발생한다.

노트

캐시 내에 없어서 다시 fetch 해와야 할 때, row가 현재 위치보다 뒤쪽이면, next()에 유리하도록 row 위치부터 n (fetch 할 때 서버로부터 가져오는 row 개수)개를 서버로부터 fetch해오고, row가 현재 위치보다 앞쪽이면, previous()에 유리하도록 (row-n+1) 위치부터 n개를 fetch해온다.

afterLast

`void afterLast() throws SQLException`

- 동작: Row 커서를 after last에 위치시킨다. Row 캐시가 마지막 row set (전체 result set의 부분을 가리키는 용어)이면 커서 위치만 변경하고 그렇지 않으면 마지막 row set (전체 row 개수-n+1 부터 n개의 row, n은 fetch 할 때 서버로부터 가져오는 row 개수)을 fetch해 온 후에 커서를 after last에 위치시킨다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 예외가 발생하면 예외가 발생한다.

beforeFirst

`void beforeFirst() throws SQLException`

- 동작: Row 커서를 before first에 위치시킨다. Row 캐시가 첫 번째 row set (전체 result set의 부분을 가리키는 용어)이면 커서 위치만 변경하고 그렇지 않으면 첫 번째 row set (1부터 n개의 row, n은 fetch 할 때 서버로부터 가져오는 row 개수)을 fetch해 온 후에 커서를 before first에 위치시킨다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 예외가 발생하면 예외가 발생한다.

cancelRowUpdates

`void cancelRowUpdates() throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우에는 동작을 참조한다.

clearWarnings

`void clearWarnings() throws SQLException`

- 동작: ResultSet 객체가 가지고 있는 SQLWarning 객체들을 모두 제거한다.
- 예외: 발생하지 않는다.

close

`void close() throws SQLException`

- 동작: 서버에 해당 커서가 열려 있으면 이를 닫게 하고 (이미 서버에 커서가 닫혀 있으면 이 작업을 수행하지 않는다. 즉, 프로토콜 전송이 일어나지 않는다.), 현재 ResultSet 객체의 상태를 closed로 변경한다. 이미 닫혀 있는 상태일 경우, 아무런 동작을 하지 않는다.
- 예외: 서버에서 에러가 발생하면 SQLException이 발생한다.

deleteRow

`void deleteRow() throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있을 경우, SQLException이 발생한다. 그 외의 경우에는 동작을 참조한다.

findColumn

`int findColumn(String columnLabel) throws SQLException`

- 동작: 해당 column 이름의 인덱스를 반환한다. 첫 번째 column의 인덱스는 1이다.
- 예외: 이미 close되었거나 해당 이름의 column을 찾지 못하면 SQLException이 발생한다.

first

`boolean first()` throws `SQLException`

- 동작: Row 커서를 first (첫 번째 row)에 위치시킨다. Row 캐시가 첫 번째 row set (전체 result set의 부분을 가리키는 용어)이면 커서 위치만 변경하고 그렇지 않으면 첫 번째 row set (1부터 n개의 row, n은 fetch 할 때 서버로부터 가져오는 row 개수)을 fetch해 온 후 커서를 first에 위치시킨다. 해당 row가 있으면 true를, 없으면 false를 반환한다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

getArray

`Array getArray(int columnIndex)` throws `SQLException`

- 동작: Array 타입은 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`Array getArray(String columnLabel)` throws `SQLException`

- 동작: Array 타입은 지원하지 않는다. GOLDILOCKS 타입 별 지원 여부 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getAsciiStream

`InputStream getAsciiStream(int columnIndex)` throws `SQLException`

- 동작: columnIndex 번째 column의 데이터를 InputStream 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 InputStream으로 변환할 수 없을 경우, `SQLException`이 발생한다.

`InputStream getAsciiStream(String columnLabel)` throws `SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 InputStream 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 InputStream으로 변환할 수 없을 경우, `SQLException`이 발생한다.

getBigDecimal

`BigDecimal getBigDecimal(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 BigDecimal 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 BigDecimal로 변환할 수 없을 경우, SQLException이 발생한다.

`BigDecimal getBigDecimal(int columnIndex, int scale) throws SQLException`

- 동작: Deprecated된 method이다. getBigDecimal(int columnIndex)과 동일하게 동작한다. Scale은 무시된다.
- 예외: **getBigDecimal(int columnIndex)**을 참조한다.

`BigDecimal getBigDecimal(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 BigDecimal 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 BigDecimal로 변환할 수 없을 경우, SQLException이 발생한다.

`BigDecimal getBigDecimal(String columnLabel, int scale) throws SQLException`

- 동작: Deprecated된 method이다. getBigDecimal(String columnLabel)과 동일하게 동작한다. Scale은 무시된다.
- 예외: **getBigDecimal(String columnLabel)**을 참조한다.

getBinaryStream

`InputStream getBinaryStream(int columnIndex) throws SQLException`

- 동작: getAsciiStream(int columnIndex)과 같다.
- 예외: **getAsciiStream(int columnIndex)**을 참조한다.

`InputStream getBinaryStream(String columnLabel) throws SQLException`

- 동작: getAsciiStream(String columnLabel)과 같다.
- 예외 **getAsciiStream(String columnLabel)**을 참조한다.

getBlob

`Blob getBlob(int columnIndex) throws SQLException`

- 동작: Blob 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

`Blob getBlob(String columnLabel) throws SQLException`

- 동작: Blob 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getBoolean

`boolean getBoolean(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 boolean 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 boolean으로 변환할 수 없을 경우, `SQLException`이 발생한다.

`boolean getBoolean(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 boolean 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 boolean으로 변환할 수 없을 경우, `SQLException`이 발생한다.

getBytes

`byte getByte(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 byte 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 byte로 변환할 수 없을 경우, `SQLException`이 발생한다.

`byte getByte(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 byte 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 byte로 변환할 수 없을 경우, `SQLException`이 발생한다.

경우, SQLException이 발생한다.

getBytes

`byte[] getBytes(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 byte[] 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났을 경우 SQLException이 발생한다.

노트

모든 GOLDILOCKS 데이터 타입에 대해 getBytes를 하면 데이터베이스에 저장된 바이너리 형태를 얻는다.

`byte[] getBytes(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 byte[] 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없을 경우 SQLException이 발생한다.

노트

모든 GOLDILOCKS 데이터 타입에 대해 getBytes를 하면 데이터베이스에 저장된 바이너리 형태를 얻는다.

getCharacterStream

`Reader getCharacterStream(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 reader 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 reader로 변환할 수 없을 경우, SQLException이 발생한다.

`Reader getCharacterStream(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 reader 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.

- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 reader로 변환할 수 없을 경우, SQLException이 발생한다.

getClob

`Clob getClob(int columnIndex) throws SQLException`

- 동작: Clob 타입은 현재 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

`Clob getClob(String columnLabel) throws SQLException`

- 동작: Clob 타입은 현재 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

getConcurrency

`int getConcurrency() throws SQLException`

- 동작: 현재 ResultSet 객체의 concurrency를 반환한다. 현재 ResultSet.CONCUR_READ_ONLY만 지원한다.
- 예외: 발생하지 않는다.

getCursorName

`String getCursorName() throws SQLException`

- 동작: 이 ResultSet이 가리키는 서버의 커서 이름을 얻어온다. 서버와 통신이 발생한다.
- 예외: ResultSet이 이미 close되었거나 서버에서 에러가 발생할 경우, SQLException이 발생한다.

getDate

`Date getDate(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 2 6-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 local timezone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 date로 변환할 수 없을 경우, SQLException이 발생한다.

`Date getDate(int columnIndex, Calendar cal)` throws `SQLException`

- 동작: columnIndex 번째 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 cal의 timeZone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 date로 변환할 수 없을 경우, `SQLException`이 발생한다.

`Date getDate(String columnLabel)` throws `SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 local timeZone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 date로 변환할 수 없을 경우, `SQLException`이 발생한다.

`Date getDate(String columnLabel, Calendar cal)` throws `SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 date 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Date 객체를 만들 때 cal의 timeZone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 date로 변환할 수 없을 경우, `SQLException`이 발생한다.

getDouble

`double getDouble(int columnIndex)` throws `SQLException`

- 동작: columnIndex 번째 column의 데이터를 double 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 double로 변환할 수 없을 경우, `SQLException`이 발생한다.

`double getDouble(String columnLabel)` throws `SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 double 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 double로 변환할 수 없을 경우, `SQLException`이 발생한다.

getFetchDirection

`int getFetchDirection() throws SQLException`

- 동작: 항상 `ResultSet.FETCH_FORWARD`만 반환한다. Backward fetch는 지원하지 않는다.
- 예외: `ResultSet`이 이미 close되었으면 `SQLException`이 발생한다.

getFetchSize

`int getFetchSize() throws SQLException`

- 동작: 한 번 fetch 할 때 서버로부터 가져오는 row 개수를 얻어온다. 0일 경우 한 번 전송할 때 통신 패킷에 담을 수 있는 최대 row 개수를 자동으로 계산한다. 기본값은 0이다.
- 예외: `ResultSet`이 이미 close되었으면 `SQLException`이 발생한다.

getFloat

`float getFloat(int columnIndex) throws SQLException`

- 동작: `columnIndex` 번째 column의 데이터를 float 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 2 6-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `ResultSet`이 이미 close되었거나 `columnIndex`가 범위를 벗어났거나 해당 타입을 float으로 변환할 수 없을 경우 `SQLException`이 발생한다.

`float getFloat(String columnLabel) throws SQLException`

- 동작: 이름이 `columnLabel`인 column의 데이터를 float 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: `ResultSet`이 이미 close되었거나 해당 `columnLabel`을 찾을 수 없거나 해당 타입을 float으로 변환할 수 없을 경우 `SQLException`이 발생한다.

getHoldability

`int getHoldability() throws SQLException`

- 동작: 현재 `ResultSet`의 holdability를 반환한다. 이 값은 `ResultSet` 객체가 생성될 때 정해지며 도중에 변경할 수는 없다. 기본값은 `ResultSet.HOLD_CURSOR_OVER_COMMIT`이다.
- 예외: `ResultSet`이 이미 close되었으면 `SQLException`이 발생한다.

getInt

`int getInt(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 int 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 int로 변환할 수 없을 경우 SQLException이 발생한다.

`int getInt(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 int 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 int로 변환할 수 없을 경우 SQLException이 발생한다.

getLong

`long getLong(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 long 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close 되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 long으로 변환할 수 없을 경우 SQLException이 발생한다.

`long getLong(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 long 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 **getter method 지원 여부-1**를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 long으로 변환할 수 없을 경우 SQLException이 발생한다.

getMetaData

`ResultSetMetaData getMetaData() throws SQLException`

- 동작: Column의 상세 정보를 얻을 수 있는 ResultSetMetaData 객체를 생성하고 반환한다.
- 예외: ResultSet이 이미 close되었거나 서버로부터 column 상세 정보를 가져오는 동안 에러가 발생하면 SQLException이 발생한다.

getNCharacterStream

Reader getNCharacterStream(int columnIndex) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

Reader getNCharacterStream(String columnLabel) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getNClob

NClob getNClob(int columnIndex) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

NClob getNClob(String columnLabel) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getString

String getString(int columnIndex) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

String getString(String columnLabel) throws SQLException

- 동작: NCHAR 계열 타입은 현재 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

getObject

Object getObject(int columnIndex) throws SQLException

- 동작: columnIndex 번째 column의 데이터를 가장 적합한 Java 객체 타입으로 얻는다. GOLDILOCKS 타입 별

지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.

- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났을 경우, SQLException이 발생한다.

Object getObject(int columnIndex, Map<String,Class<?>> map) throws SQLException

- 동작: 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

Object getObject(String columnLabel) throws SQLException

- 동작: 이름이 columnLabel인 column의 데이터를 가장 적합한 Java 객체 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없을 경우, SQLException이 발생한다.

Object getObject(String columnLabel, Map<String,Class<?>> map) throws SQLException

- 동작: 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

getRef

Ref getRef(int columnIndex) throws SQLException

- 동작: REF 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

Ref getRef(String columnLabel) throws SQLException

- 동작: REF 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

getRow

int getRow() throws SQLException

- 동작: 현재 ResultSet 객체의 커서 위치를 반환한다. 첫 번째 row는 1이다. Before first인 경우 0을 반환한다.
- 예외: ResultSet이 이미 close되었으면 SQLException이 발생한다.

getRowId

`RowId getRowId(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 RowId 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 RowId로 변환할 수 없을 경우, SQLException이 발생한다.

`RowId getRowId(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 RowId 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 RowId로 변환할 수 없을 경우, SQLException이 발생한다.

getShort

`short getShort(int columnIndex) throws SQLException`

- 동작: columnIndex 번째 column의 데이터를 short 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 short로 변환할 수 없을 경우, SQLException이 발생한다.

`short getShort(String columnLabel) throws SQLException`

- 동작: 이름이 columnLabel인 column의 데이터를 short 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 short로 변환할 수 없을 경우, SQLException이 발생한다.

getSQLXML

`SQLXML getSQLXML(int columnIndex) throws SQLException`

- 동작: SQLXML 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

`SQLXML getSQLXML(String columnLabel) throws SQLException`

- 동작: SQLXML 타입은 지원하지 않는다.
- 예외: 항상 `SQLException`을 반환한다.

getStatement

`Statement getStatement()` throws `SQLException`

- 동작: 이 `ResultSet` 객체를 생성한 `statement` 객체를 반환한다.
- 예외: `ResultSet`이 이미 `close`되었으면 `SQLException`이 발생한다.

getString

`String getString(int columnIndex)` throws `SQLException`

- 동작: `columnIndex` 번째 `column`의 데이터를 `string` 타입으로 얻는다. `GOLDILOCKS` 타입 별 지원 여부는 표 2 6-4 **GOLDILOCKS 타입에 대한 getter method 지원 여부-1**를 참조한다. `BINARY`, `VARBINARY`, `LONG VARBINARY`에 대해 `getString()`을 하면 `hex code`의 문자열을 반환받는다.
- 예외: `ResultSet`이 이미 `close`되었거나 `columnIndex`가 범위를 벗어났을 경우, `SQLException`이 발생한다.

`String getString(String columnLabel)` throws `SQLException`

- 동작: 이름이 `columnLabel`인 `column`의 데이터를 `string` 타입으로 얻는다. `GOLDILOCKS` 타입 별 지원 여부는 표 26-4 **GOLDILOCKS 타입에 대한 getter method 지원 여부-1**를 참조한다. `BINARY`, `VARBINARY`, `LONG VARBINARY`에 대해 `getString()`을 하면 `hex code`의 문자열을 반환받는다.
- 예외: `ResultSet`이 이미 `close`되었거나 해당 `columnLabel`을 찾을 수 없을 경우, `SQLException`이 발생한다.

getTime

`Time getTime(int columnIndex)` throws `SQLException`

- 동작: `columnIndex` 번째 `column`의 데이터를 `time` 타입으로 얻는다. `GOLDILOCKS` 타입 별 지원 여부는 표 2 6-4 **GOLDILOCKS 타입에 대한 getter method 지원 여부-1**를 참조한다. `Time` 객체를 만들 때는 `local timezone`을 사용한다.
- 예외: `ResultSet`이 이미 `close`되었거나 `columnIndex`가 범위를 벗어났거나 해당 타입을 `time`으로 변환할 수 없을 경우, `SQLException`이 발생한다.

`Time getTime(int columnIndex, Calendar cal)` throws `SQLException`

- 동작: `columnIndex` 번째 `column`의 데이터를 `time` 타입으로 얻는다. `GOLDILOCKS` 타입 별 지원 여부는 표 2 6-4 **GOLDILOCKS 타입에 대한 getter method 지원 여부-1**를 참조한다. `Time` 객체를 만들 때 `cal`의 `timezone`을 사용한다.

- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 time으로 변환할 수 없을 경우, SQLException이 발생한다.

Time getTime(String columnLabel) throws SQLException

- 동작: 이름이 columnLabel인 column의 데이터를 time 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Time 객체를 만들 때 local time zone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 time으로 변환할 수 없을 경우 SQLException이 발생한다.

Time getTime(String columnLabel, Calendar cal) throws SQLException

- 동작: 이름이 columnLabel인 column의 데이터를 time 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Time 객체를 만들 때 local time zone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 time으로 변환할 수 없을 경우 SQLException이 발생한다.

getTimestamp

Timestamp getTimestamp(int columnIndex) throws SQLException

- 동작: columnIndex 번째 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Timestamp 객체를 만들 때 local timezone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 timestamp로 변환할 수 없을 경우, SQLException이 발생한다.

Timestamp getTimestamp(int columnIndex, Calendar cal) throws SQLException

- 동작: columnIndex 번째 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Timestamp 객체를 만들 때, cal의 timezone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 columnIndex가 범위를 벗어났거나 해당 타입을 timestamp로 변환할 수 없을 경우, SQLException이 발생한다.

Timestamp getTimestamp(String columnLabel) throws SQLException

- 동작: 이름이 columnLabel인 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Timestamp 객체를 만들 때 local timezone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 timestamp로 변환할

수 없을 경우, SQLException이 발생한다.

Timestamp `getTimestamp(String columnLabel, Calendar cal)` throws SQLException

- 동작: 이름이 columnLabel인 column의 데이터를 timestamp 타입으로 얻는다. GOLDILOCKS 타입 별 지원 여부는 표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1를 참조한다. Timestamp 객체를 만들 때, cal의 timezone을 사용한다.
- 예외: ResultSet이 이미 close되었거나 해당 columnLabel을 찾을 수 없거나 해당 타입을 timestamp로 변환할 수 없을 경우, SQLException이 발생한다.

getType

int `getType()` throws SQLException

- 동작: 현재 ResultSet 타입을 반환한다. ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE 중의 하나를 반환한다.
- 예외: ResultSet이 이미 close되어 있으면 SQLException이 발생한다.

getUnicodeStream

InputStream `getUnicodeStream(int columnIndex)` throws SQLException

- 동작: Deprecated 된 method이다. 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

InputStream `getUnicodeStream(String columnLabel)` throws SQLException

- 동작: Deprecated 된 method이다. 구현되어 있지 않다.
- 예외: 항상 SQLFeatureNotSupportedException이 발생한다.

getURL

URL `getURL(int columnIndex)` throws SQLException

- 동작: URL 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

URL `getURL(String columnLabel)` throws SQLException

- 동작: URL 타입은 지원하지 않는다.
- 예외: 항상 SQLFeatureNotSupportedException을 반환한다.

getWarnings

SQLWarning getWarnings() throws SQLException

- 동작: 현재까지 이 객체에 축적된 SQLWarning 목록을 반환한다. 서버로부터 경고를 전달받으면 SQLWarning을 생성하는데 clearWarning하지 않을 경우 계속 누적된다. 없을 경우, null이 반환된다.
- 예외: 발생하지 않는다.

insertRow

void insertRow() throws SQLException

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우에는 동작을 참조한다.

isAfterLast

boolean isAfterLast() throws SQLException

- 동작: 현재 커서 위치가 after last인지 묻는다. 맞으면 true를, 그렇지 않으면 false를 반환한다.
- 예외: ResultSet이 이미 close되어 있으면 SQLException이 발생한다.

isBeforeFirst

boolean isBeforeFirst() throws SQLException

- 동작: 현재 커서 위치가 before first인지 묻는다. 맞으면 true를, 그렇지 않으면 false를 반환한다.
- 예외: ResultSet이 이미 close되어 있으면 SQLException이 발생한다.

isClosed

boolean isClosed() throws SQLException

- 동작: 현재 ResultSet이 close되었는지 묻는다. Close되었으면 true를, 그렇지 않으면 false를 반환한다. 사용자가 close를 호출하지 않았더라도 서버의 커서가 close되어 ResultSet이 close될 수 있다. 예를 들어 ResultSet을 생성한 statement가 close 되거나 holdability가 ResultSet.CLOSE_CURSOR_AT_COMMIT 모드일 때 트랜잭션이 commit 되거나, fetch 도중에 서버로부터 에러를 반환받을 때 등이다.
- 예외: 발생하지 않는다.

isFirst

`boolean isFirst()` throws `SQLException`

- 동작: 현재 커서 위치가 first (첫 번째 row)인지 여부를 묻는다. 맞으면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: `ResultSet`이 이미 close되어 있으면 `SQLException`이 발생한다.

isLast

`boolean isLast()` throws `SQLException`

- 동작: 현재 커서 위치가 last (마지막 row)인지 여부를 묻는다. 맞으면 `true`를, 그렇지 않으면 `false`를 반환한다.
- 예외: `ResultSet`이 이미 close되어 있으면 `SQLException`이 발생한다.

last

`boolean last()` throws `SQLException`

- 동작: Row 커서를 last (마지막 row)에 위치시킨다. Row 캐시가 마지막 row set (전체 result set의 일부)이면 커서 위치만 변경하고 그렇지 않으면 마지막 row set (last-n+1부터 last의 row, n은 fetch 할 때 서버로부터 가져오는 row 개수)을 fetch해 온 후 커서를 last에 위치시킨다. 해당 row가 있으면 `true`를, 없으면 `false`를 반환한다.
- 예외: `ResultSet`이 close되었거나 `ResultSet` 타입이 `ResultSet.TYPE_FORWARD_ONLY`이거나 fetch 할 때 서버로부터 예외가 발생하면 예외가 발생한다.

moveToCurrentRow

`void moveToCurrentRow()` throws `SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. `ResultSet` concurrency가 `ResultSet.CONCUR_READ_ONLY`일 경우에는 `SQLException`이 발생하고 `ResultSet.CONCUR_UPDATABLE`일 경우에는 `SQLFeatureNotSupportedException`이 발생한다.
- 예외: 이미 close되어 있으면 `SQLException`이 발생한다. 그 외의 경우는 동작을 참조한다.

moveToInsertRow

`void moveToInsertRow()` throws `SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. `ResultSet` concurrency가 `ResultSet.CONCUR_READ_ONLY`일 경우에는 `SQLException`이 발생하고 `ResultSet.CONCUR_UPDATABLE`일 경우에는 `SQLFeatureN`

otSupportedException이 발생한다.

- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

next

`boolean next()` throws SQLException

- 동작: Row 커서를 현재 row 다음에 위치시킨다. 현재 row가 row 캐시의 마지막 row이면 서버로부터 다음 row 캐시를 가져온다. 해당 row가 있으면 true를, 없으면 false를 반환한다.
- 예외: ResultSet이 close되었거나 fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

previous

`boolean previous()` throws SQLException

- 동작: Row 커서를 현재 위치 이전 row에 위치시킨다. 현재 row가 row 캐시의 첫 번째 row이면 서버로부터 이전 row 캐시 (x-n부터 x-1까지 n개의 row, x는 현재 row 인덱스)를 가져온다. 해당 row가 있으면 true를, 없으면 false를 반환한다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

refreshRow

`void refreshRow()` throws SQLException

- 동작: ResultSet 타입이 ResultSet.SCROLL_SENSITIVE일 경우, 서버로부터 현재 row 캐시를 다시 가져온다. 그동안 (동일 트랜잭션에 의해서든, 타 트랜잭션에 의해서든) row에 갱신된 것이 있으면 반영된다. ResultSet 타입이 ResultSet.Scroll_INSENSITIVE일 경우 아무런 작업을 하지 않는다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

relative

`boolean relative(int rows)` throws SQLException

- 동작: Row 커서를 현재 위치로부터 rows 만큼 떨어진 곳으로 이동시킨다. 현재 row 캐시 내에서 이동할 수 있으면 커서 위치만 변경시키고, 그렇지 않으면 서버로부터 해당 row 캐시를 fetch해온 후 커서를 위치시킨다. 이 때 이동할 위치가 현재 위치보다 뒤쪽이면 (next 방향이면) (next에 유리하도록) row 캐시를 rows부터 rows+n-1까지 fetch하고, 현재 위치보다 앞쪽이면 (previous 방향이면) (previous에 유리하도록) row 캐시를 rows-n+1부터 rows까지 fetch한다.
- 예외: ResultSet이 close되었거나 ResultSet 타입이 ResultSet.TYPE_FORWARD_ONLY이거나 fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

버로부터 에러가 발생하면 예외가 발생한다.

rowDeleted

`boolean rowDeleted()` throws `SQLException`

- 동작: 현재 커서 위치의 row가 (동일 트랜잭션에 의해서든, 타 트랜잭션에 의해서든) delete 되었는지 여부를 묻는다. Delete 되었으면 true를, 그렇지 않으면 false를 반환한다.
- 예외: `ResultSet`이 close되었거나 `ResultSet` 타입이 `ResultSet.TYPE_FORWARD_ONLY`이면 예외가 발생한다.

rowInserted

`boolean rowInserted()` throws `SQLException`

- 동작: 현재 커서 위치의 row가 (동일 트랜잭션에 의해서든, 타 트랜잭션에 의해서든) insert 되었는지 여부를 묻는다. Insert되었으면 true를, 그렇지 않으면 false를 반환한다.
- 예외: `ResultSet`이 close되었거나 `ResultSet` 타입이 `ResultSet.TYPE_FORWARD_ONLY`이면 예외가 발생한다.

rowUpdated

`boolean rowUpdated()` throws `SQLException`

- 동작: 현재 커서 위치의 row가 (동일 트랜잭션에 의해서든, 타 트랜잭션에 의해서든) update 되었는지 여부를 묻는다. Update되었으면 true를, 그렇지 않으면 false를 반환한다.
- 예외: `ResultSet`이 close되었거나 `ResultSet` 타입이 `ResultSet.TYPE_FORWARD_ONLY`이면 예외가 발생한다.

setFetchDirection

`void setFetchDirection(int direction)` throws `SQLException`

- 동작: 서버가 backward fetch를 지원하지 않는다. 따라서 `ResultSet.FETCH_FORWARD`만 가능하다. 다른 값이 입력되면 `SQLWarning`을 생성한다.
- 예외: `ResultSet`이 이미 close되었거나 정의된 값이 아닌 다른 값이 인자로 입력되었을 경우 예외가 발생한다.

setFetchSize

`void setFetchSize(int rows)` throws `SQLException`

- 동작: 서버로부터 한 번에 fetch 할 row 개수를 지정한다. 0이면 서버가 자동으로 결정한다. 0일 경우 forward only 커서에 대해서는 한 번의 통신 패킷에 담을 수 있는 row 개수로 정하고, scrollable 커서일 경우 100개로

정한다.

- 예외: ResultSet이 이미 close되었으면 예외가 발생한다.

updateArray

`void updateArray(int columnIndex, Array x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateArray(String columnLabel, Array x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateAsciiStream

`void updateAsciiStream(int columnIndex, InputStream x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우엔 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateAsciiStream(int columnIndex, InputStream x, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateAsciiStream(int columnIndex, InputStream x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateAsciiStream(String columnLabel, InputStream x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateAsciiStream(String columnLabel, InputStream x, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateAsciiStream(String columnLabel, InputStream x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateBigDecimal

`void updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBigDecimal(String columnLabel, BigDecimal x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateBinaryStream

`void updateBinaryStream(int columnIndex, InputStream x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ

_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.

- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBinaryStream(int columnIndex, InputStream x, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBinaryStream(int columnIndex, InputStream x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBinaryStream(String columnLabel, InputStream x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBinaryStream(String columnLabel, InputStream x, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBinaryStream(String columnLabel, InputStream x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateBlob

`void updateBlob(int columnIndex, Blob x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBlob(int columnIndex, InputStream inputStream) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBlob(int columnIndex, InputStream inputStream, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBlob(String columnLabel, Blob x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBlob(String columnLabel, InputStream inputStream) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBlob(String columnLabel, InputStream inputStream, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateBoolean

`void updateBoolean(int columnIndex, boolean x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBoolean(String columnLabel, boolean x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateByte

`void updateByte(int columnIndex, byte x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateByte(String columnLabel, byte x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateBytes

`void updateBytes(int columnIndex, byte[] x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateBytes(String columnLabel, byte[] x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateCharacterStream

`void updateCharacterStream(int columnIndex, Reader x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateCharacterStream(int columnIndex, Reader x, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateCharacterStream(int columnIndex, Reader x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateCharacterStream(String columnLabel, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateCharacterStream(String columnLabel, Reader reader, int length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateCharacterStream(String columnLabel, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateClob

`void updateClob(int columnIndex, Clob x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateClob(int columnIndex, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateClob(int columnIndex, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateClob(String columnLabel, Clob x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateClob(String columnLabel, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateClob(String columnLabel, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateDate

`void updateDate(int columnIndex, Date x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateDate(String columnLabel, Date x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateDouble

`void updateDouble(int columnIndex, double x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateDouble(String columnLabel, double x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateFloat

`void updateFloat(int columnIndex, float x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateFloat(String columnLabel, float x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateInt

`void updateInt(int columnIndex, int x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateInt(String columnLabel, int x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateLong

`void updateLong(int columnIndex, long x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateLong(String columnLabel, long x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateNCharacterStream

`void updateNCharacterStream(int columnIndex, Reader x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNCharacterStream(int columnIndex, Reader x, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNCharacterStream(String columnLabel, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNCharacterStream(String columnLabel, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateNClob

`void updateNClob(int columnIndex, NClob nClob) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNClob(int columnIndex, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNClob(int columnIndex, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNClob(String columnLabel, NClob nClob) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNClob(String columnLabel, Reader reader) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNClob(String columnLabel, Reader reader, long length) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateNString

`void updateNString(int columnIndex, String nString) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNString(String columnLabel, String nString) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateNull

`void updateNull(int columnIndex) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateNull(String columnLabel) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateObject

`void updateObject(int columnIndex, Object x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateObject(int columnIndex, Object x, int scaleOrLength) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateObject(String columnLabel, Object x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateObject(String columnLabel, Object x, int scaleOrLength) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateRef

`void updateRef(int columnIndex, Ref x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateRef(String columnLabel, Ref x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateRow

`void updateRow() throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ

_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.

- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateRowId

`void updateRowId(int columnIndex, RowId x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateRowId(String columnLabel, RowId x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateShort

`void updateShort(int columnIndex, short x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateShort(String columnLabel, short x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateSQLXML

`void updateSQLXML(int columnIndex, SQLXML xmlObject) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureN

otSupportedException이 발생한다.

- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateSQLXML(String columnLabel, SQLXML xmlObject) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateString

`void updateString(int columnIndex, String x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateString(String columnLabel, String x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateTime

`void updateTime(int columnIndex, Time x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateTime(String columnLabel, Time x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

updateTimestamp

`void updateTimestamp(int columnIndex, Timestamp x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

`void updateTimestamp(String columnLabel, Timestamp x) throws SQLException`

- 동작: Cursor update 기능은 아직 구현되어 있지 않다. ResultSet concurrency가 ResultSet.CONCUR_READ_ONLY일 경우에는 SQLException이 발생하고 ResultSet.CONCUR_UPDATABLE일 경우에는 SQLFeatureNotSupportedException이 발생한다.
- 예외: 이미 close되어 있으면 SQLException이 발생한다. 그 외의 경우는 동작을 참조한다.

wasNull

`boolean wasNull() throws SQLException`

- 동작: 마지막으로 읽은 column의 값이 NULL인지 여부를 묻는다. NULL이면 true를, 그렇지 않으면 false를 반환한다.
- 예외: ResultSet이 이미 close되었거나 column 값을 읽은 적이 없으면 SQLException이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface)`

- 동작: 이 객체가 iface 인터페이스를 구현한 클래스인지 여부를 묻는다. 맞으면 true를, 그렇지 않으면 false를 반환한다. GOLDILOCKS ResultSet 객체는 다른 클래스의 wrapper가 아니므로 wrapper의 유무는 판단하지 않고 오직 주어진 인자 클래스 타입을 구현했는지 여부만 묻는다.
- 예외: 발생하지 않는다.

unwrap

`<T> T unwrap(Class<T> iface)`

- 동작: GOLDILOCKS ResultSet은 다른 클래스의 wrapper가 아니므로 unwrap 하더라도 결국 자기 자신을 반환한다. 해당 타입으로 캐스팅한 후에 반환한다. iface가 isWrapperFor() method의 인자로 주어졌을 때 false를 반환하는 값일 경우, 이 method는 예외를 발생시킨다.
- 예외: iface가 이 객체의 타입이 아닌 경우 (이 객체가 implement 하지 않은 타입인 경우) SQLException이 발

생한다.

ResultSetMetaData

getCatalogName

`String getCatalogName(int column) throws SQLException`

- 동작: 해당 column의 카탈로그 이름을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnClassName

`String getColumnClassName(int column) throws SQLException`

- 동작: 해당 column의 타입에 가장 적합한 Java 클래스의 이름을 반환한다. java.math.BigDecimal 등으로 명시되며, Java 내부의 getName()method를 참조한다. Binary 타입의 경우에는 byte[].class.getName()을 참조하기 때문에 '[B'등으로 표시될 수 있다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnCount

`int getColumnCount() throws SQLException`

- 동작: ResultSet이 가지는 column 개수를 반환한다.
- 예외: 발생하지 않는다.

getColumnDisplaySize

`int getColumnDisplaySize(int column) throws SQLException`

- 동작: 해당 column의 값을 출력할 때 최대 폭을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnLabel

`String getColumnLabel(int column) throws SQLException`

- 동작: 해당 column의 label을 얻어온다. 예를 들어 "select C1 + 1 from t1"과 같은 질의문의 경우 label은 "C1 + 1"이고 name은 ""이다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnName

`String getColumnName(int column) throws SQLException`

- 동작: 해당 column의 alias name을 반환한다. JDBC 스펙에는 alias name이 아니라 column의 원래 이름을 얻어오도록 명시했지만, 각종 view의 경우 원래 이름들이 의미없거나 복잡하므로 alias name을 사용하는게 더 좋다. 예를 들어 "select C1 as C2 from t1"과 같은 질의문의 경우 name과 label 모두 "C2"가 된다. name과 label이 다른 경우는 `getColumnLabel`을 참조한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnType

`int getColumnType(int column) throws SQLException`

- 동작: 해당 column의 타입을 반환한다. 반환값은 types에 정의된 값이다. GOLDILOCKS interval 계열 타입에 대해서는 Types.OTHERS가 반환된다. 그 외 다른 타입들에 대해서는 해당 types의 상수값이 반환된다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getColumnTypeName

`String getColumnTypeName(int column) throws SQLException`

- 동작: 해당 column의 GOLDILOCKS column 타입 이름이 반환된다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getPrecision

`int getPrecision(int column) throws SQLException`

- 동작: 해당 column의 precision을 반환한다. Precision이 없는 타입에 대해서는 0을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getScale

`int getScale(int column) throws SQLException`

- 동작: 해당 column의 scale을 반환한다. Scale이 없는 타입에 대해서는 0을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getSchemaName

`String getSchemaName(int column) throws SQLException`

- 동작: 해당 column의 스키마 이름을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

getTableName

`String getTableName(int column) throws SQLException`

- 동작: 해당 column의 테이블 이름을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isAutoIncrement

`boolean isAutoIncrement(int column) throws SQLException`

- 동작: 해당 column이 자동으로 고유값을 부여받는 column인지 여부를 반환한다. 맞으면 true를 그렇지 않으면 false를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isCaseSensitive

`boolean isCaseSensitive(int column) throws SQLException`

- 동작: 해당 column이 대소문자를 구별하는지 여부를 반환한다. 구별하면 true를, 구별하지 않으면 false를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isCurrency

`boolean isCurrency(int column) throws SQLException`

- 동작: 서버에서 column의 currency 판단을 할 수 없으므로 항상 false를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isDefinitelyWritable

`boolean isDefinitelyWritable(int column) throws SQLException`

- 동작: 해당 column의 updatable 여부를 반환한다. GOLDDILOCKS에서 definitely writable은 지원하지 않는다. 항상 isUpdatable()과 같은 값을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isNullable

`int isNullable(int column) throws SQLException`

- 동작: 해당 column이 NULL을 가질 수 있는지 여부를 반환한다. columnNullable이나 columnNoNulls 중 하나를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isReadOnly

`boolean isReadOnly(int column) throws SQLException`

- 동작: 해당 column의 read-only 여부를 반환한다. 항상 isUpdatable()의 반대값을 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isSearchable

`boolean isSearchable(int column) throws SQLException`

- 동작: 해당 column이 조건절에 사용될 수 있는지 여부를 반환한다. GOLDDILOCKS의 모든 target column이 조건절에 사용될 수 있으므로 항상 true를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isSigned

`boolean isSigned(int column) throws SQLException`

- 동작: 해당 column이 부호를 가지는지 여부를 반환한다. 부호를 가지면 true를, 그렇지 않으면 false를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isWritable

`boolean isWritable(int column) throws SQLException`

- 동작: 해당 column이 updatable인지 여부를 반환한다.
- 예외: Column이 잘못된 값이면 SQLException이 발생한다.

isWrapperFor

`boolean isWrapperFor(Class<?> iface)`

- 동작: 이 객체가 iface 인터페이스를 구현한 클래스인지 여부를 묻는다. 맞으면 true를, 그렇지 않으면 false를 반환한다. GOLDILOCKS ResultSetMetaData 객체는 다른 클래스의 wrapper가 아니므로 wrapper의 유무는 판단하지 않고 오직 주어진 인자 클래스 타입을 구현했는지 여부만 묻는다.
- 예외: 발생하지 않는다.

unwrap

`<T> T unwrap(Class<T> iface)`

- 동작: GOLDILOCKS ResultSetMetaData는 다른 클래스의 wrapper가 아니므로 unwrap 하더라도 결국 자기 자신을 반환한다. 해당 타입으로 캐스팅한 후에 반환한다. iface가 isWrapperFor() method의 인자로 주어졌을 때 false를 반환하는 값이라면 이 method는 예외를 발생시킨다.
- 예외: iface가 이 객체의 타입이 아닌 경우 (이 객체가 implement 하지 않은 타입을 부여한 경우) SQLException이 발생한다.

RowId

equals

`boolean equals(Object obj) throws SQLException`

- 동작: 이 객체가 obj와 같은 RowId를 나타내면 true를, 그렇지 않으면 false를 반환한다.
- 예외: 발생하지 않는다.

getBytes

`byte[] getBytes() throws SQLException`

- 동작: RowId의 바이트 배열 값을 반환한다.
- 예외: 발생하지 않는다.

hashCode

`int hashCode() throws SQLException`

- 동작: 해시 코드 값을 반환한다.
- 예외: 발생하지 않는다.

toString

`String toString() throws SQLException`

- 동작: RowId 값의 base-64 문자열을 반환한다.
- 예외: 발생하지 않는다.

RowSet

클래스가 구현되지 않았다.

addRowSetListener

`void addRowSetListener(RowSetListener listener) throws SQLException`

clearParameters

`void clearParameters() throws SQLException`

execute

`void execute() throws SQLException`

getCommand

`String getCommand() throws SQLException`

getDataSourceName

`String getDataSourceName() throws SQLException`

getEscapeProcessing

`boolean getEscapeProcessing() throws SQLException`

getMaxFieldSize

`int getMaxFieldSize() throws SQLException`

getMaxRows

`int getMaxRows() throws SQLException`

getPassword

`String getPassword() throws SQLException`

getQueryTimeout

`int getQueryTimeout() throws SQLException`

getTransactionIsolation

`int getTransactionIsolation() throws SQLException`

getTypeMap

`Map<String,Class<?>> getTypeMap() throws SQLException`

getUrl

`String getUrl() throws SQLException`

getUsername

`String getUsername() throws SQLException`

isReadOnly

`boolean isReadOnly() throws SQLException`

removeRowSetListener

`void removeRowSetListener(RowSetListener listener) throws SQLException`

setArray

`void setArray(int i, Array x) throws SQLException`

setAsciiStream

`void setAsciiStream(int parameterIndex, InputStream x) throws SQLException`

`void setAsciiStream(int parameterIndex, InputStream x, int length) throws SQLException`

`void setAsciiStream(String parameterName, InputStream x) throws SQLException`

`void setAsciiStream(String parameterName, InputStream x, int length) throws SQLException`

setBigDecimal

`void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException`

`void setBigDecimal(String parameterName, BigDecimal x) throws SQLException`

setBinaryStream

`void setBinaryStream(int parameterIndex, InputStream x) throws SQLException`

`void setBinaryStream(int parameterIndex, InputStream x, int length) throws SQLException`

`void setBinaryStream(String parameterName, InputStream x) throws SQLException`

`void setBinaryStream(String parameterName, InputStream x, int length) throws SQLException`

setBlob

`void setBlob(int i, Blob x) throws SQLException`

| `void setBlob(int parameterIndex, InputStream inputStream) throws SQLException`

| `void setBlob(int parameterIndex, InputStream inputStream, long length) throws SQLException`

| `void setBlob(String parameterName, Blob x) throws SQLException`

| `void setBlob(String parameterName, InputStream inputStream) throws SQLException`

| `void setBlob(String parameterName, InputStream inputStream, long length) throws SQLException`

setBoolean

| `void setBoolean(int parameterIndex, boolean x) throws SQLException`

| `void setBoolean(String parameterName, boolean x) throws SQLException`

setByte

| `void setByte(int parameterIndex, byte x) throws SQLException`

| `void setByte(String parameterName, byte x) throws SQLException`

setBytes

| `void setBytes(int parameterIndex, byte[] x) throws SQLException`

| `void setBytes(String parameterName, byte[] x) throws SQLException`

setCharacterStream

| `void setCharacterStream(int parameterIndex, Reader reader) throws SQLException`

| `void setCharacterStream(int parameterIndex, Reader reader, int length) throws SQLException`

| `void setCharacterStream(String parameterName, Reader reader) throws SQLException`

| `void setCharacterStream(String parameterName, Reader reader, int length) throws SQLException`

setClob

- | `void setClob(int i, Clob x) throws SQLException`
- | `void setClob(int parameterIndex, Reader reader) throws SQLException`
- | `void setClob(int parameterIndex, Reader reader, long length) throws SQLException`
- | `void setClob(String parameterName, Clob x) throws SQLException`
- | `void setClob(String parameterName, Reader reader) throws SQLException`
- | `void setClob(String parameterName, Reader reader, long length) throws SQLException`

setCommand

- | `void setCommand(String cmd) throws SQLException`

setConcurrency

- | `void setConcurrency(int concurrency) throws SQLException`

setDataSourceName

- | `void setDataSourceName(String name) throws SQLException`

setDate

- | `void setDate(int parameterIndex, Date x) throws SQLException`
- | `void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException`
- | `void setDate(String parameterName, Date x) throws SQLException`
- | `void setDate(String parameterName, Date x, Calendar cal) throws SQLException`

setDouble

`void setDouble(int parameterIndex, double x) throws SQLException`

`void setDouble(String parameterName, double x) throws SQLException`

setEscapeProcessing

`void setEscapeProcessing(boolean enable) throws SQLException`

setFloat

`void setFloat(int parameterIndex, float x) throws SQLException`

`void setFloat(String parameterName, float x) throws SQLException`

setInt

`void setInt(int parameterIndex, int x) throws SQLException`

`void setInt(String parameterName, int x) throws SQLException`

setLong

`void setLong(int parameterIndex, long x) throws SQLException`

`void setLong(String parameterName, long x) throws SQLException`

setMaxFieldSize

`void setMaxFieldSize(int max) throws SQLException`

setMaxRows

`void setMaxRows(int max) throws SQLException`

setNCharacterStream

- | `void setNCharacterStream(int parameterIndex, Reader value) throws SQLException`
- | `void setNCharacterStream(int parameterIndex, Reader value, long length) throws SQLException`
- | `void setNCharacterStream(String parameterName, Reader value) throws SQLException`
- | `void setNCharacterStream(String parameterName, Reader value, long length) throws SQLException`

setNClob

- | `void setNClob(int parameterIndex, NClob value) throws SQLException`
- | `void setNClob(int parameterIndex, Reader reader) throws SQLException`
- | `void setNClob(int parameterIndex, Reader reader, long length) throws SQLException`
- | `void setNClob(String parameterName, NClob value) throws SQLException`
- | `void setNClob(String parameterName, Reader reader) throws SQLException`
- | `void setNClob(String parameterName, Reader reader, long length) throws SQLException`

setNString

- | `void setNString(int parameterIndex, String value) throws SQLException`
- | `void setNString(String parameterName, String value) throws SQLException`

setNull

- | `void setNull(int parameterIndex, int sqlType) throws SQLException`
- | `void setNull(int paramIndex, int sqlType, String typeName) throws SQLException`
- | `void setNull(String parameterName, int sqlType) throws SQLException`

`void setNull(String parameterName, int sqlType, String typeName) throws SQLException`

setObject

`void setObject(int parameterIndex, Object x) throws SQLException`

`void setObject(int parameterIndex, Object x, int targetSqlType) throws SQLException`

`void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength) throws SQLException`

`void setObject(String parameterName, Object x) throws SQLException`

`void setObject(String parameterName, Object x, int targetSqlType) throws SQLException`

`void setObject(String parameterName, Object x, int targetSqlType, int scale) throws SQLException`

setPassword

`void setPassword(String password) throws SQLException`

setQueryTimeout

`void setQueryTimeout(int seconds) throws SQLException`

setReadOnly

`void setReadOnly(boolean value) throws SQLException`

setRef

`void setRef(int i, Ref x) throws SQLException`

setRowId

`void setRowId(int parameterIndex, RowId x) throws SQLException`

`void setRowId(String parameterName, RowId x) throws SQLException`

setShort

`void setShort(int parameterIndex, short x) throws SQLException`

`void setShort(String parameterName, short x) throws SQLException`

setSQLXML

`void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException`

`void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException`

setString

`void setString(int parameterIndex, String x) throws SQLException`

`void setString(String parameterName, String x) throws SQLException`

setTime

`void setTime(int parameterIndex, Time x) throws SQLException`

`void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException`

`void setTime(String parameterName, Time x) throws SQLException`

`void setTime(String parameterName, Time x, Calendar cal) throws SQLException`

setTimestamp

`void setTimestamp(int parameterIndex, Timestamp x) throws SQLException`

`void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException`

`void setTimestamp(String parameterName, Timestamp x) throws SQLException`

`void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws SQLException`

setTransactionIsolation

`void setTransactionIsolation(int level) throws SQLException`

setType

`void setType(int type) throws SQLException`

setTypeMap

`void setTypeMap(Map<String,Class<?>> map) throws SQLException`

setURL

`void setURL(int parameterIndex, URL x) throws SQLException`

setUrl

`void setUrl(String url) throws SQLException`

setUsername

`void setUsername(String name) throws SQLException`

RowSetMetaData

클래스가 구현되지 않았다.

setAutoIncrement

`void setAutoIncrement(int columnIndex, boolean property) throws SQLException`

setCaseSensitive

`void setCaseSensitive(int columnIndex, boolean property) throws SQLException`

setCatalogName

`void setCatalogName(int columnIndex, String catalogName) throws SQLException`

setColumnCount

`void setColumnCount(int columnCount) throws SQLException`

setColumnDisplaySize

`void setColumnDisplaySize(int columnIndex, int size) throws SQLException`

setColumnLabel

`void setColumnLabel(int columnIndex, String label) throws SQLException`

setColumnName

`void setColumnName(int columnIndex, String columnName) throws SQLException`

setColumnType

`void setColumnType(int columnIndex, int SQLType) throws SQLException`

setColumnName

`void setColumnName(int columnIndex, String typeName) throws SQLException`

setCurrency

`void setCurrency(int columnIndex, boolean property) throws SQLException`

setNullable

`void setNullable(int columnIndex, int property) throws SQLException`

setPrecision

`void setPrecision(int columnIndex, int precision) throws SQLException`

setScale

`void setScale(int columnIndex, int scale) throws SQLException`

setSchemaName

`void setSchemaName(int columnIndex, String schemaName) throws SQLException`

setSearchable

`void setSearchable(int columnIndex, boolean property) throws SQLException`

setSigned

`void setSigned(int columnIndex, boolean property) throws SQLException`

setTableName

`void setTableName(int columnIndex, String tableName) throws SQLException`

Savepoint

getSavepointId

`int getSavepointId() throws SQLException`

- 동작: 자동으로 부여된 ID 값을 반환한다.
- 예외: Savepoint 객체에 이름을 주어 생성한 경우 ID를 가지지 않으므로 SQLException이 발생한다.

getSavepointName

`String getSavepointName() throws SQLException`

- 동작: Savepoint 객체를 생성할 때 지정한 이름을 반환한다.
- 예외: 자동 ID 값으로 savepoint 객체를 생성한 경우, SQLException이 발생한다.

SQLData

클래스가 구현되지 않았다.

getSQLTypeName

| String getSQLTypeName() throws SQLException

readSQL

| void readSQL(SQLInput stream, String typeName) throws SQLException

writeSQL

| void writeSQL(SQLOutput stream) throws SQLException

SQLXML

클래스가 구현되지 않았다.

free

| void free() throws SQLException

getBinaryStream

| InputStream getBinaryStream() throws SQLException

getCharacterStream

| Reader getCharacterStream() throws SQLException

getSource

| <T extends Source> T getSource(Class<T> sourceClass) throws SQLException

getString

| String getString() throws SQLException

setBinaryStream

| OutputStream setBinaryStream() throws SQLException

setCharacterStream

| Writer setCharacterStream() throws SQLException

setResult

`<T extends Result> T setResult(Class<T> resultClass) throws SQLException`

setString

`void setString(String value) throws SQLException`

Statement

addBatch

`void addBatch(String sql) throws SQLException`

- 동작: SQL 문을 batch job에 추가한다.
- 예외: 발생하지 않는다.

cancel

`void cancel() throws SQLException`

- 동작: 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`을 반환한다.

clearBatch

`void clearBatch() throws SQLException`

- 동작: 등록된 batch job들을 모두 제거한다. 등록된 batch job이 없을 경우, 아무런 작업도 하지 않는다.
- 예외: 발생하지 않는다.

clearWarnings

`void clearWarnings() throws SQLException`

- 동작: Statement 객체가 가지고 있는 `SQLWarning` 객체들을 모두 제거한다.
- 예외: 발생하지 않는다.

close

`void close() throws SQLException`

- 동작: 현재 statement 객체를 close하고 서버에 할당된 statement 관련 정보가 있을 경우, 이를 해제한다. 이 객체에서 생성한 `ResultSet`이 있으면 이를 모두 close한다. 이 statement 객체를 생성한 connection 객체로부터 이 객체를 제거한다.
- 예외: 서버로부터 statement 정보를 해제할 때 에러가 발생하면 예외가 발생한다.

execute

`boolean execute(String sql) throws SQLException`

- 동작: SQL 문을 실행한다. 실행한 SQL 문이 ResultSet을 가지면 true를, 그렇지 않으면 false를 반환한다.
- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하면 예외가 발생한다.

`boolean execute(String sql, int autoGeneratedKeys) throws SQLException`

- 동작: SQL 문을 실행한다. 실행한 SQL 문이 ResultSet을 가지면 true를, 그렇지 않으면 false를 반환한다. Auto key generation은 지원하지 않는다.
- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하면 예외가 발생한다. autoGeneratedKeys가 Statement.NO_GENERATED_KEYS가 아닐 경우, SQLFeatureNotSupportedException이 발생한다.

`boolean execute(String sql, int[] columnIndexes) throws SQLException`

- 동작: SQL 문을 실행한다. 실행한 SQL 문이 ResultSet을 가지면 true를, 그렇지 않으면 false를 반환한다. Auto key generation은 지원하지 않는다.
- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하면 예외가 발생한다. acolumnIndexes가 null이 아닐 경우, SQLFeatureNotSupportedException이 발생한다.

`boolean execute(String sql, String[] columnNames) throws SQLException`

- 동작: SQL 문을 실행한다. 실행한 SQL 문이 ResultSet을 가지면 true를, 그렇지 않으면 false를 반환한다. Auto key generation은 지원하지 않는다.
- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하면 예외가 발생한다. columnNames가 null이 아닐 경우, SQLFeatureNotSupportedException이 발생한다.

executeBatch

`int[] executeBatch() throws SQLException`

- 동작: 등록된 batch job을 차례대로 execute한다. 각 batch job마다 서버와 통신이 발생한다. 각 batch job이 실행된 후 갱신된 row 개수들의 배열이 반환된다.
- 예외: Statement가 이미 close되었거나 등록된 batch job이 없거나 실행할 때 서버로부터 에러가 발생하면 예외가 발생한다.

노트

Batch job들이 한 번에 서버로 전송되어 실행되는 구조가 아니기 때문에 일반 `execute()`에 비해 성능상 큰 이점은 없다. 빠른 처리를 위해서는 `PreparedStatement`의 `batch execution`을 사용하는 것이 좋다.

executeQuery

`ResultSet executeQuery(String sql)` throws `SQLException`

- 동작: 주어진 SQL 문을 실행하고 결과의 일부를 받아서 `ResultSet`을 생성한다.
- 예외: `Statement`가 이미 `close`되었거나 `batch job`이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하였거나 SQL 문이 `select` 구문이 아닐 경우 예외가 발생한다.

주의

`execute()`와 동작이 조금 다르다. 동일한 SQL 문에 대해 `execute()` 후 `getResultSet()`을 하면 서버와 두 번 통신하게 되는데, `execute()` 할 때는 실행 명령이, `getResultSet()` 할 때는 `fetch` 관련 명령이 수행된다. 이에 반해 `executeQuery()`는 SQL 문이 `select` 문이라고 가정하고 `fetch` 할 때까지 한 번의 통신으로 모두 수행한다.

executeUpdate

`int executeUpdate(String sql)` throws `SQLException`

- 동작: SQL 문을 실행한다. 실행에 의해 갱신된 `row`의 개수를 반환한다.
- 예외: `Statement`가 이미 `close`되었거나 `batch job`이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하였거나 SQL 문이 `select` 문일 경우, 예외가 발생한다.

`int executeUpdate(String sql, int autoGeneratedKeys)` throws `SQLException`

- 동작: SQL 문을 실행한다. 실행에 의해 갱신된 `row`의 개수를 반환한다. `Auto key generation` 기능은 지원하지 않는다.
- 예외: `Statement`가 이미 `close`되었거나 `batch job`이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하였거나 SQL 문이 `select` 문일 경우 예외가 발생한다. `autoGeneratedKeys`가 `Statement.NO_GENERATED_KEYS`가 아닐 경우, `SQLException`이 발생한다.

`int executeUpdate(String sql, int[] columnIndexes)` throws `SQLException`

- 동작: SQL 문을 실행한다. 실행에 의해 갱신된 `row`의 개수를 반환한다. `Auto key generation` 기능은 지원하지

않는다.

- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하였거나 SQL 문이 select 문일 경우 예외가 발생한다. columnIndexes가 null이 아닐 경우, SQLFeatureNotSupportedException이 발생한다.

`int executeUpdate(String sql, String[] columnNames) throws SQLException`

- 동작: SQL 문을 실행한다. 실행에 의해 갱신된 row의 개수를 반환한다. Auto key generation 기능은 지원하지 않는다.
- 예외: Statement가 이미 close되었거나 batch job이 등록되어 있거나 실행할 때 서버로부터 에러가 발생하였거나 SQL 문이 select 문일 경우 예외가 발생한다. columnNames가 null이 아닐 경우, SQLFeatureNotSupportedException이 발생한다.

getConnection

`Connection getConnection() throws SQLException`

- 동작: 이 객체를 생성한 connection 객체를 반환한다. PooledConnection을 통해 얻은 logical connection으로 statement 객체를 생성하였을 경우, 사용자는 이 method를 통해 physical connection이 아닌 logical connection을 얻는다.
- 예외: Statement가 이미 close 된 경우에는 예외가 발생한다.

getExplainPlan

`String getExplainPlan() throws SQLException`

- 동작: 비표준 method로써 GoldilocksStatement의 고유한 method이다. 생성된 plan text를 얻어온다. 이 method를 사용하려면 setExplainPlanOption() method를 통해 plan text를 생성하도록 설정해야 한다. 자세한 사용법은 **Plan Text 조회**를 참조한다.
- 예외: Statement가 이미 close되었거나 서버에서 에러가 발생할 경우 예외가 발생한다.

getExplainPlanOption

`int getExplainPlanOption() throws SQLException`

- 동작: 비표준 method로써 GoldilocksStatement의 고유한 method이다. 현재 설정된 plan text 생성에 대한 옵션을 얻어온다. 반환되는 값은 다음 값 중의 하나이고, 기본값은 GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF이다.
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE

- GoldilocksStatement.EXPLAIN_PLAN_OPTION_ONLY
- 예외: 발생하지 않는다.

getFetchDirection

`int getFetchDirection() throws SQLException`

- 동작: 항상 `ResultSet.FETCH_FORWARD`를 반환한다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getFetchSize

`int getFetchSize() throws SQLException`

- 동작: 이 statement 객체로부터 얻는 ResultSet의 기본 fetch size를 반환한다. 기본값은 0이고, 0일 경우 fetch 할 row 개수를 서버가 자동으로 결정한다. 자세한 내용 ResultSet의 `getFetchSize`를 참조한다.
- 예외: Statement가 이미 close 되었을 경우, 예외가 발생한다.

getGeneratedKeys

`ResultSet getGeneratedKeys() throws SQLException`

- 동작: 지원하지 않는다.
- 예외: 항상 `SQLFeatureNotSupportedException`이 발생한다.

getMaxFieldSize

`int getMaxFieldSize() throws SQLException`

- 동작: Max field size를 반환한다. 이 값은 column의 최대 길이를 제한한다. Fetch 할 때 column 값이 이 길이 보다 길면 나머지 데이터는 잘린다. 기본값은 0이고, 0일 경우 최대 길이는 무한대이다.
- 예외: Statement가 이미 close 되었을 경우, 예외가 발생한다.

getMaxRows

`int getMaxRows() throws SQLException`

- 동작: Max rows를 반환한다. Max rows는 이 statement로부터 얻은 ResultSet이 가질 수 있는 최대 row 개수를 의미한다. 최대 row 개수 이상의 row들은 무시된다. 기본값은 0이고, 0은 무제한을 의미한다.

- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getMoreResults

`boolean getMoreResults()` throws `SQLException`

- 동작: 현재는 한 번의 execution당 하나의 ResultSet만 가질 수 있기 때문에 항상 false를 반환한다. 현재의 ResultSet은 close된다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

`boolean getMoreResults(int current)` throws `SQLException`

getQueryTimeout

`int getQueryTimeout()` throws `SQLException`

- 동작: Query timeout 값을 얻는다. 이 값은 서버가 execution 할 때 적용하는 timeout 값으로써 실행 시간이 이 시간을 넘기면 execution이 취소되고 사용자는 timeout 관련 에러를 반환받는다. 단위는 초이며 사용자가 특별히 설정하지 않을 경우, 세션의 기본값을 가져온다. 세션의 기본값은 프로퍼티로 지정하지 않을 경우 0이며 0은 무한 대기를 의미한다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getResultSet

`ResultSet getResultSet()` throws `SQLException`

- 동작: 현재 수행된 execution에 대해 fetch를 수행하고 fetch 결과의 일부를 받아 ResultSet을 생성한 후 반환한다. JDBC 스펙에 명시된 것처럼 execution 한 번당 이 method를 한 번만 호출하도록 구현하지 않고 이 method를 여러 번 부르더라도 같은 객체를 반환하도록 했다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다. Fetch 할 때 서버로부터 에러가 발생하면 예외가 발생한다.

getResultSetConcurrency

`int getResultSetConcurrency()` throws `SQLException`

- 동작: ResultSet concurrency를 반환한다. 이 값은 이 객체로부터 생성되는 ResultSet의 concurrency를 결정한다. 기본값은 `ResultSet.CONCUR_READ_ONLY`이다. Updatable cursor는 아직 지원하지 않는다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getResultSetHoldability

`int getResultSetHoldability() throws SQLException`

- 동작: ResultSet holdability를 반환한다. 이 값은 이 객체로부터 생성되는 ResultSet의 holdability를 결정한다. 기본값은 ResultSet.HOLD_CURSORS_OVER_COMMIT이다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getResultSetType

`int getResultSetType() throws SQLException`

- 동작: ResultSet type을 반환한다. 이 값은 이 객체로부터 생성되는 ResultSet의 type을 결정한다. 기본값은 ResultSet.TYPE_FORWARD_ONLY이다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getUpdateCount

`int getUpdateCount() throws SQLException`

- 동작: 마지막으로 수행한 execution에 대한 갱신이 반영된 row 개수를 반환한다. 마지막으로 수행한 SQL 문이 UPDATE, INSERT 문이 아닐 경우에는 -1을 반환한다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

getUpdateRowCount

`long getUpdateRowCount() throws SQLException`

- 동작: getUpdateCount와 같지만 반환되는 타입이 long이다. 비표준 method로써 GoldilocksStatement 타입으로 캐스팅해야 사용할 수 있다.
- 예외: 발생하지 않는다.

getWarnings

`SQLWarning getWarnings() throws SQLException`

- 동작: 이 객체에 축적된 SQLWarning을 반환한다. 없으면 null을 반환한다.
- 예외: 발생하지 않는다.

isClosed

`boolean isClosed() throws SQLException`

- 동작: 이 statement가 close 되었는지 여부를 반환한다. Close되었으면 true를, 그렇지 않으면 false를 반환한다. 사용자가 명시적으로 close()를 호출했을 때 뿐만 아니라 서버에 의해서나 connection 객체에 의해서도 close 될 수 있다.
- 예외: 발생하지 않는다.

isPoolable

`boolean isPoolable() throws SQLException`

- 동작: Statement pooling은 지원하지 않는다. 항상 false를 반환한다.
- 예외: 발생하지 않는다.

setCursorName

`void setCursorName(String name) throws SQLException`

- 동작: 현재 수행된 statement에 의해 생성된 커서의 이름을 설정한다.
- 예외: 커서 이름을 설정할 때 서버로부터 에러가 발생하면 예외가 발생한다.

setEscapeProcessing

`void setEscapeProcessing(boolean enable) throws SQLException`

- 동작: SQL 문의 escape는 서버의 parser에서 처리되기 때문에 JDBC에서 그 기능을 막을 수는 없다. 아무런 동작도 하지 않는다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

setExplainPlanOption

`void setExplainPlanOption(int option) throws SQLException`

- 동작: 비표준 method로써 GoldilocksStatement의 고유 method이다. Plan text 생성 옵션을 지정한다. Option은 다음 값 중의 하나로 설정되어야 하며 그 의미는 다음과 같다.
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF: Plan text를 생성하지 않는다.
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON: 실행할 때 plan text를 생성한다.
 - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE: 실행할 때 더 자세한 plan text를 생성

한다.

- GoldilocksStatement.EXPLAIN_PLAN_OPTION_ONLY: 실행할 때 plan text를 생성하지만 실제 실행은 수행하지 않는다.
- 예외: 위의 네 가지 이외의 다른 값을 설정할 경우 예외가 발생한다.

setFetchDirection

`void setFetchDirection(int direction) throws SQLException`

- 동작: Fetch 방향을 설정한다. GOLDILOCKS는 forward fetch만 지원하기 때문에 direction이 ResultSet.FETCH_FORWARD가 아닐 경우, 예외가 발생한다.
- 예외: Statement가 이미 close되었거나 direction이 FETCH_FORWARD가 아닌 경우 예외가 발생한다.

setFetchSize

`void setFetchSize(int rows) throws SQLException`

- 동작: 이 statement 객체로부터 얻는 ResultSet의 기본 fetch size를 설정한다. 기본값은 0이고, 0일 경우 fetch 할 때 row 개수를 서버가 자동으로 결정한다. 자세한 내용은 ResultSet의 **setFetchSize**를 참조한다.
- 예외: Statement가 이미 close 되었을 경우, 예외가 발생한다.

setMaxFieldSize

`void setMaxFieldSize(int max) throws SQLException`

- 동작: Max field size를 설정한다. 이 값은 column의 최대 길이를 제한한다. Fetch 할 때 column 값이 이 길이 보다 길면 나머지 데이터는 잘린다. 기본값은 0이고, 0은 최대 길이가 무한대라는 의미이다. CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, LONG VARBINARY 타입에 대해서만 유효하다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

setMaxRows

`void setMaxRows(int max) throws SQLException`

- 동작: Max rows를 설정한다. Max rows는 이 statement로부터 얻은 ResultSet이 가질 수 있는 최대 row 개수를 의미한다. 최대 row 개수 이상의 row들은 무시된다. 기본값은 0이고, 0은 무제한을 의미한다.
- 예외: Statement가 이미 close 되었을 경우, 예외가 발생한다.

setPoolable

| void setPoolable(boolean poolable) throws SQLException

- 동작: Statement pooling은 지원하지 않는다. 아무런 동작을 하지 않는다.
- 예외: 발생하지 않는다.

setQueryTimeout

| void setQueryTimeout(int seconds) throws SQLException

- 동작: Query timeout 값을 설정한다. 이 값은 서버가 execution 할 때 적용하는 timeout 값으로써 실행 시간이 이 시간을 넘기면 execution이 취소되고 사용자는 timeout 관련 에러를 반환받는다. 단위는 초이다. 사용자가 특별히 설정하지 않으면 세션의 기본값이 적용된다. 세션의 기본값은 프로퍼티로 지정하지 않을 경우, 0이며 이는 무한 대기를 의미한다.
- 예외: Statement가 이미 close되었을 경우, 예외가 발생한다.

isWrapperFor

| boolean isWrapperFor(Class<?> iface) throws SQLException

- 동작: 이 객체가 iface 인터페이스를 구현한 클래스인지 여부를 묻는다. 맞으면 true를, 그렇지 않으면 false를 반환한다. GOLDBLOCKS statement 객체는 다른 클래스의 wrapper가 아니므로 wrapper의 유무는 판단하지 않고 오직 주어진 인자 클래스 타입을 구현했는지 여부만 묻는다.
- 예외: 발생하지 않는다.

unwrap

| <T> T unwrap(Class<T> iface) throws SQLException

- 동작: GOLDBLOCKS statement는 다른 클래스의 wrapper가 아니므로 unwrap 하더라도 결국 자기 자신을 반환한다. 해당 타입으로 캐스팅한 후에 반환한다. iface가 isWrapperFor() method의 인자로 주어졌을 때 false를 반환하는 값이라면 이 method가 예외를 발생시킨다.
- 예외: iface가 이 객체의 타입이 아닌 경우 (이 객체가 implement 하지 않은 타입을 준 경우) SQLException이 발생한다.

Struct

클래스가 구현되지 않았다.

getAttributes

`Object[] getAttributes() throws SQLException`

getAttributes

`Object[] getAttributes(Map<String,Class<?>> map) throws SQLException`

getSQLTypeName

`String getSQLTypeName() throws SQLException`

XAConnection

getXAResource

XAResource `getXAResource()` throws `SQLException`

- 동작: XA 명령을 수행할 수 있는 XAResource 객체를 반환한다. Method를 여러 번 호출하면 계속 같은 결과를 반환받는다.
- 예외: 발생하지 않는다.

XADataSource

getXAConnection

`XAConnection getXAConnection()` throws `SQLException`

- 동작: XAConnection 객체를 생성하고 반환한다. 연결에 필요한 정보들은 별도의 비표준 method들로 사전에 설정되어 있어야 한다.
- 예외: 서버와의 연결에 실패하면 `SQLException`이 발생한다.

`XAConnection getXAConnection(String user, String password)` throws `SQLException`

- 동작: Username과 password로 새로운 XAConnection 객체를 열어서 반환한다. 그 외에 연결에 필요한 정보들은 별도의 비표준 method들로 사전에 설정되어 있어야 한다.
- 예외: 서버와의 연결에 실패하면 `SQLException`이 발생한다.

XAResource

commit

`void commit(Xid xid, boolean onePhase) throws XAException`

- 동작: 글로벌 트랜잭션 xid에 대해 XA commit 명령을 수행한다. onePhase를 true로 설정하면 one phase commit이 수행된다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

end

`void end(Xid xid, int flags) throws XAException`

- 동작: 글로벌 트랜잭션 xid에 대해 XA end 명령을 수행한다. flags는 TMSUCCESS, TMFAIL 또는 TMSUSPEND 중의 하나를 가진다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

forget

`void forget(Xid xid) throws XAException`

- 동작: 글로벌 트랜잭션 xid에 대해 XA forget 명령을 수행한다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

getTransactionTimeout

`int getTransactionTimeout() throws XAException`

- 동작: GOLDILOCKS는 트랜잭션 타임아웃을 지원하지 않는다. 항상 0을 반환한다.
- 예외: 발생하지 않는다.

isSameRM

`boolean isSameRM(XAResource xares) throws XAException`

- 동작: XAResource 객체가 생성될 때 고유의 rmid를 가지는데 이 rmid로 동일한 XAResource 객체인지 여부를 판단한다.
- 예외: 발생하지 않는다.

prepare

`int prepare(Xid xid) throws XAException`

- 동작: 글로벌 트랜잭션 xid에 대해 XA prepare 명령을 수행한다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

recover

`Xid[] recover(int flag) throws XAException`

- 동작: 주어진 flag로 XA recover 명령을 수행하고 prepare된 트랜잭션 브랜치들의 배열을 반환받는다. flag는 T_MSTARTRSCAN, T_MENDRSCAN, T_MNOFLAGS 중 하나의 값을 가질 수 있다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

rollback

`void rollback(Xid xid) throws XAException`

- 동작: 글로벌 트랜잭션 xid에 대해 XA rollback 명령을 수행한다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

setTransactionTimeout

`boolean setTransactionTimeout(int seconds) throws XAException`

- 동작: GOLDDILOCKS는 트랜잭션 타임아웃을 지원하지 않는다. 아무런 작업을 하지 않는다.
- 예외: 발생하지 않는다.

start

`void start(Xid xid, int flags) throws XAException`

- 동작: 주어진 flag로 글로벌 트랜잭션을 시작한다. flag는 T_MNOFLAGS, T_MJOIN, T_MRESUME 중 하나의 값을 가질 수 있다.
- 예외: 수행 결과 에러가 발생하면 XAException이 발생한다.

GoldilocksInterval

GoldilocksInterval 객체를 사용하여 GOLDILOCKS의 column에 값을 부여하려면 **추가 타입 사용하기**를 참조한다.

createIntervalYear

```
public static GoldilocksInterval createIntervalYear(int yearPrecision, boolean sign, int year)
throws SQLException
```

- 동작: 주어진 year 값을 가지고 GoldilocksInterval 객체를 생성한다. yearPrecision은 year가 가질 수 있는 자릿수를 의미한다. Year는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: 주어진 year 값이 yearPrecision을 초과하면 예러가 발생한다.

```
public static GoldilocksInterval createIntervalYear(int yearPrecision, String year) throws
SQLException
```

- 동작: 주어진 year 값을 가지고 GoldilocksInterval 객체를 생성한다. yearPrecision은 year가 가질 수 있는 자릿수를 의미한다. Year는 0 이상의 정수 값이어야 한다.
- 예외: 주어진 year 값이 yearPrecision을 초과하면 예러가 발생한다.

createIntervalMonth

```
public static GoldilocksInterval createIntervalMonth(int monthPrecision, boolean sign, int
month) throws SQLException
```

- 동작: 주어진 month 값을 가지고 GoldilocksInterval 객체를 생성한다. monthPrecision은 month가 가질 수 있는 자릿수를 의미한다. Month는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: 주어진 month 값이 monthPrecision을 초과하면 예러가 발생한다.

```
public static GoldilocksInterval createIntervalMonth(int monthPrecision, String month) throws
SQLException
```

- 동작: 주어진 month 값을 가지고 GoldilocksInterval 객체를 생성한다. monthPrecision은 month가 가질 수 있는 자릿수를 의미한다. Month는 0 이상의 정수 값이어야 한다.
- 예외: 주어진 month 값이 monthPrecision을 초과하면 예러가 발생한다.

createIntervalYearToMonth

```
public static GoldilocksInterval createIntervalYearToMonth(int yearPrecision, boolean sign,
int year, int month) throws SQLException
```

- 동작: 주어진 year, month 값을 가지고 GoldilocksInterval 객체를 생성한다. yearPrecision은 year가 가질 수 있는 자릿수를 의미한다. Year, month는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: 주어진 year 값이 yearPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalYearToMonth(int yearPrecision, String
yearToMonth) throws SQLException
```

- 동작: 주어진 yearToMonth 값을 가지고 GoldilocksInterval 객체를 생성한다. yearPrecision은 year가 가질 수 있는 자릿수를 의미한다. yearToMonth는 "yy-mm" 패턴을 만족해야 한다.
- 예외: 주어진 year 값이 yearPrecision을 초과하면 예외가 발생한다.

createIntervalDay

```
public static GoldilocksInterval createIntervalDay(int dayPrecision, boolean sign, int day)
throws SQLException
```

- 동작: 주어진 day 값을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. Day는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: 주어진 day 값이 dayPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalDay(int dayPrecision, String day) throws
SQLException
```

- 동작: 주어진 day 값을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. Day는 0 이상의 정수 값이어야 한다.
- 예외: 주어진 day 값이 dayPrecision을 초과하면 예외가 발생한다.

createIntervalHour

```
public static GoldilocksInterval createIntervalHour(int hourPrecision, boolean sign, int hour)
throws SQLException
```

- 동작: 주어진 hour 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. Hour는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고, 음수이면 false이다.
- 예외: 주어진 hour 값이 hourPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalHour(int hourPrecision, String hour) throws
SQLException
```

- 동작: 주어진 hour 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. Hour는 0 이상의 정수 값이어야 한다.
- 예외: 주어진 hour 값이 hourPrecision을 초과하면 예러가 발생한다.

createIntervalMinute

```
public static GoldilocksInterval createIntervalMinute(int minutePrecision, boolean sign, int
minute) throws SQLException
```

- 동작: 주어진 minute 값을 가지고 GoldilocksInterval 객체를 생성한다. minutePrecision은 minute가 가질 수 있는 자릿수를 의미한다. Minute는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false 이다.
- 예외: 주어진 minute 값이 minutePrecision을 초과하면 예러가 발생한다.

```
public static GoldilocksInterval createIntervalMinute(int minutePrecision, String minute)
throws SQLException
```

- 동작: 주어진 minute 값을 가지고 GoldilocksInterval 객체를 생성한다. minutePrecision은 minute가 가질 수 있는 자릿수를 의미한다. Minute는 0 이상의 정수 값이어야 한다.
- 예외: 주어진 minute 값이 minutePrecision을 초과하면 예러가 발생한다.

createIntervalSecond

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, boolean sign, int second, int microsecond) throws SQLException
```

- 동작: 주어진 second 값을 가지고 GoldilocksInterval 객체를 생성한다. secondPrecision은 second가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Second와 microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: 주어진 second 값이 secondPrecision을 초과하거나 microSecond 값이 fractionalPrecision을 초과하면 예러가 발생한다.

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- 동작: 주어진 day, hour, minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. secondPrecision은 day, hour, minute, second를 second로 환산했을 때 second가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Day, hour, minute, second,

microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.

- 예외: 환산된 second 값이 secondPrecision을 초과하거나 microSecond 값이 fractionalPrecision을 초과하면 에러가 발생한다.

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int fractionalPrecision, String second) throws SQLException
```

- 동작: 주어진 second 문자열을 가지고 GoldilocksInterval 객체를 생성한다. secondPrecision은 second가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Second 문자열은 "dd hh:mm:ss.ffffff"나 "dd hh:mm:ss", "dd hh:mm", "dd hh", "hh:mm", "ss.ffffff", "ss" 패턴 중 하나여야 한다.
- 예외: 환산된 second 값이 secondPrecision을 초과하거나 microSecond 값이 fractionalPrecision을 초과하면 에러가 발생한다. 문자열이 정해진 포맷에 적합하지 않을 경우에도 에러가 발생한다.

createIntervalDayToHour

```
public static GoldilocksInterval createIntervalDayToHour(int dayPrecision, boolean sign, int day, int hour) throws SQLException
```

- 동작: 주어진 day, hour 값을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. Day, hour는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Day 값이 dayPrecision을 초과하면 에러가 발생한다.

```
public static GoldilocksInterval createIntervalDayToHour(int dayPrecision, String dayToHour) throws SQLException
```

- 동작: 주어진 dayToHour 문자열을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. dayToHour는 "dd hh" 포맷을 만족해야 한다. 혹은 "dd hh:mm:ss"와 같은 포맷일 경우, dd, hh 외에는 모두 0이어야 한다.
- 예외: Day 값이 dayPrecision을 초과하거나 주어진 문자열이 포맷을 만족하지 않으면 에러가 발생한다.

createIntervalDayToMinute

```
public static GoldilocksInterval createIntervalDayToMinute(int dayPrecision, boolean sign, int day, int hour, int minute) throws SQLException
```

- 동작: 주어진 day, hour, minute 값을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. Day, hour, minute는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Day 값이 dayPrecision을 초과하면 에러가 발생한다.

```
public static GoldilocksInterval createIntervalDayToMinute(int dayPrecision, String
dayToMinute) throws SQLException
```

- 동작: 주어진 dayToMinute 문자열을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. dayToMinute는 "dd hh:mm" 포맷을 만족해야 한다. 혹은 "dd hh:mm:ss"와 같은 포맷일 경우, dd, hh, mm 외에는 모두 0이어야 한다.
- 예외: Day 값이 dayPrecision을 초과하거나 주어진 문자열이 포맷을 만족하지 않으면 예외가 발생한다.

createIntervalDayToSecond

```
public static GoldilocksInterval createIntervalDayToSecond(int dayPrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- 동작: 주어진 day, hour, minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Day, hour, minute, second, microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Day 값이 dayPrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalDayToSecond(int dayPrecision, String
dayToSecond) throws SQLException
```

- 동작: 주어진 dayToSecond 문자열을 가지고 GoldilocksInterval 객체를 생성한다. dayPrecision은 day가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. dayToSecond는 "dd hh:mm:ss.ffffff", "dd hh:mm:ss", "hh:mm:ss" 또는 "hh:mm" 포맷을 만족해야 한다.
- 예외: Day 값 또는 환산된 day 값이 dayPrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하거나 주어진 문자열이 포맷을 만족하지 않을 경우, 예외가 발생한다.

createIntervalHourToMinute

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, boolean sign,
int hour, int minute) throws SQLException
```

- 동작: 주어진 hour, minute 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. Hour, minute는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Hour 값이 hourPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, boolean sign,
int day, int hour, int minute) throws SQLException
```

- 동작: 주어진 day, hour, minute 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. Day, hour, minute는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Hour 값 또는 환산된 hour 값이 이 hourPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, String
hourToMinute) throws SQLException
```

- 동작: 주어진 hourToMinute 문자열을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. hourToMinute는 "hh:mm"를 만족해야 한다. 다른 포맷일 경우 ss나 ffffff 값은 모두 0이어야 한다.
- 예외: Hour 값 또는 환산된 hour 값이 hourPrecision을 초과하면 예외가 발생한다. 주어진 문자열이 포맷을 만족하지 않아도 예외가 발생한다.

createIntervalHourToSecond

```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int
fractionalPrecision, boolean sign, int hour, int minute, int second, int microsecond) throws
SQLException
```

- 동작: 주어진 hour, minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Hour, minute, second, microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Hour 값이 hourPrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- 동작: 주어진 day, hour, minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Day, hour, minute, second, microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Hour 값 또는 환산된 hour 값이 hourPrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하면 예외가 발생한다.


```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int fractionalPrecision, String hourToSecond) throws SQLException
```

- 동작: 주어진 hourToSecond 문자열을 가지고 GoldilocksInterval 객체를 생성한다. hourPrecision은 hour가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. hourToSecond는 "hh:mm:ss.ffffff", "hh:mm:ss" 또는 "hh:mm" 포맷을 만족해야 한다. dd를 포함할 경우, 환산된 hour 값이 hourPrecision을 초과하면 안된다.
- 예외: Hour 값 또는 환산된 hour 값이 hourPrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하거나 주어진 문자열이 포맷을 만족하지 않으면 예외가 발생한다.

createIntervalMinuteToSecond

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int fractionalPrecision, boolean sign, int minute, int second, int microsecond) throws SQLException
```

- 동작: 주어진 minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. minutePrecision은 minute가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Minute, second, microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Minute 값이 minutePrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond) throws SQLException
```

- 동작: 주어진 day, hour, minute, second, microsecond 값을 가지고 GoldilocksInterval 객체를 생성한다. minutePrecision은 minute가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. Day, hour, minute, second, microsecond는 0 이상의 값이어야 한다. Sign은 시간이 양수이면 true이고 음수이면 false이다.
- 예외: Minute 값 또는 환산된 minute 값이 minutePrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하면 예외가 발생한다.

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int fractionalPrecision, String minuteToSecond) throws SQLException
```

- 동작: 주어진 minuteToSecond 문자열을 가지고 GoldilocksInterval 객체를 생성한다. minutePrecision은 minute가 가질 수 있는 자릿수를 의미한다. fractionalPrecision은 microsecond가 가질 수 있는 자릿수를 의미한다. minuteToSecond는 "mm:ss.ffffff"나 "mm:ss" 포맷을 만족해야 한다. dd나 hh를 포함할 경우, 환산된 minute 값이 minutePrecision을 넘으면 안된다.
- 예외: Minute 값 또는 환산된 minute 값이 minutePrecision을 초과하거나 microsecond 값이 fractionalPrecision을 초과하거나 주어진 문자열이 포맷을 만족하지 않을 경우, 예외가 발생한다.

getSign

```
public int getSign()
```

- 동작: 시간이 양수이면 1을, 음수이면 -1을 반환한다.
- 예외: 발생하지 않는다.

getYear

```
public int getYear()
```

- 동작: Year 값을 반환한다. Interval 객체가 음수인지 여부는 getYear()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getMonth

```
public int getMonth()
```

- 동작: Month 값을 반환한다. Interval 객체가 음수인지 여부는 getMonth()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getAccumulatedMonth

```
public int getAccumulatedMonth()
```

- 동작: Year와 month를 month로 환산한 값을 반환한다.
- 예외: 발생하지 않는다.

getDay

```
public int getDay()
```

- 동작: Day 값을 반환한다. Interval 객체가 음수인지 여부는 getDay()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getHour

```
public int getHour()
```

- 동작: Hour 값을 반환한다. Interval 객체가 음수인지 여부는 getHour()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getAccumulatedHour

```
public int getAccumulatedHour()
```

- 동작: Day와 hour를 hour로 환산한 값을 반환한다.
- 예외: 발생하지 않는다.

getMinute

```
public int getMinute()
```

- 동작: Minute 값을 반환한다. Interval 객체가 음수인지 여부는 getMinute()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getAccumulatedMinute

```
public int getAccumulatedMinute()
```

- 동작: Day, hour, minute를 minute로 환산한 값을 반환한다.
- 예외: 발생하지 않는다.

getSecond

```
public int getSecond()
```

- 동작: Second 값을 반환한다. Interval 객체가 음수인지 여부는 getSecond()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getAccumulatedSecond

```
public int getAccumulatedSecond()
```

- 동작: Day, hour, minute, second를 second로 환산한 값을 반환한다.
- 예외: 발생하지 않는다.

getMicroSecond

```
public int getMicroSecond()
```

- 동작: Microsecond 값을 반환한다. Interval 객체의 음수 여부는 getMicroSecond()를 통해 반환되지 않는다.
- 예외: 발생하지 않는다.

getAccumulatedMicroSecond

```
public long getAccumulatedMicroSecond()
```

- 동작: Day, hour, minute, second, microsecond를 microsecond로 환산한 값을 반환한다.
- 예외: 발생하지 않는다.

getTypeName

```
public String getTypeName()
```

- 동작: 타입 이름을 반환한다.
- 예외: 발생하지 않는다.

getSqlType

```
public int getSqlType()
```

- 동작: 이 객체에 해당되는 타입을 GoldilocksTypes에 정의된 타입 상수로 반환한다.
- 예외: 발생하지 않는다.

toString

```
public String toString()
```

- 동작: 이 객체가 나타내는 interval 값을 문자열로 반환한다.
- 예외: 발생하지 않는다.

GOLDILOCKS Type

상수 정의

```
public static final int INTERVAL_YEAR;
public static final int INTERVAL_MONTH;
public static final int INTERVAL_DAY;
public static final int INTERVAL_HOUR;
public static final int INTERVAL_MINUTE;
public static final int INTERVAL_SECOND;
public static final int INTERVAL_YEAR_TO_MONTH;
public static final int INTERVAL_DAY_TO_HOUR;
public static final int INTERVAL_DAY_TO_MINUTE;
public static final int INTERVAL_DAY_TO_SECOND;
public static final int INTERVAL_HOUR_TO_MINUTE;
public static final int INTERVAL_HOUR_TO_SECOND;
public static final int INTERVAL_MINUTE_TO_SECOND;
public static final int TIME_WITH_TIME_ZONE;
public static final int TIMESTAMP_WITH_TIME_ZONE;
```

이 상수들은 `java.sql.Types`의 상수처럼 사용된다. 즉, `PreparedStatement`의 `setObject`나 `ResultSet`의 `getObject`에 타입을 명시할 때 사용된다. 이 타입들은 JDBC 표준에는 정의되어 있지 않으므로 `GoldilocksTypes`를 통해 별도로 제공된다.

타입 변환

다음 표는 타입 변환 방법을 설명한다.

표 26-3 SQL 타입 → GOLDILOCKS 타입

SQL 타입	GOLDILOCKS 타입
<code>Types.BIGINT</code>	<code>NATIVE_BIGINT</code>
<code>Types.BINARY</code>	<code>BINARY(2000)</code>
<code>Types.BIT</code>	<code>BOOLEAN</code>
<code>Types.BOOLEAN</code>	<code>BOOLEAN</code>
<code>Types.BLOB</code>	<code>N/A</code>
<code>Types.CHAR</code>	<code>CHAR(2000)</code>
<code>Types.CLOB</code>	<code>N/A</code>
<code>Types.DATE</code>	<code>DATE</code>

SQL 타입	GOLDILOCKS 타입
Types.DECIMAL	DECIMAL
Types.DOUBLE	NATIVE_DOUBLE
Types.FLOAT	FLOAT
Types.NUMERIC	NUMBER
Types.INTEGER	NATIVE_INTEGER
Types.LONGVARBINARY	LONG VARBINARY
Types.LONGVARCHAR	LONG VARCHAR
Types.REAL	NATIVE_REAL
Types.ROWID	ROWID
Types.SMALLINT	NATIVE_SMALLINT
Types.TIME	TIME
Types.TIMESTAMP	TIMESTAMP
Types.VARBINARY	VARBINARY(4000)
Types.VARCHAR	VARCHAR(4000)
GoldilocksTypes.INTERVAL_YEAR	INTERVAL YEAR
GoldilocksTypes.INTERVAL_MONTH	INTERVAL MONTH
GoldilocksTypes.INTERVAL_YEAR_TO_MONTH	INTERVAL YEAR TO MONTH
GoldilocksTypes.INTERVAL_DAY	INTERVAL DAY
GoldilocksTypes.INTERVAL_HOUR	INTERVAL HOUR
GoldilocksTypes.INTERVAL_MINUTE	INTERVAL MINUTE
GoldilocksTypes.INTERVAL_SECOND	INTERVAL SECOND
GoldilocksTypes.INTERVAL_DAY_TO_HOUR	INTERVAL DAY TO HOUR
GoldilocksTypes.INTERVAL_DAY_TO_MINUTE	INTERVAL DAY TO MINUTE
GoldilocksTypes.INTERVAL_DAY_TO_SECOND	INTERVAL DAY TO SECOND
GoldilocksTypes.INTERVAL_HOUR_TO_MINUTE	INTERVAL HOUR TO MINUTE
GoldilocksTypes.INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR TO SECOND
GoldilocksTypes.INTERVAL_MINUTE_TO_SECOND	INTERVAL MINUTE TO SECOND
Types.OTHER	N/A
Types.ARRAY	N/A
Types.DATALINK	N/A
Types.DISTINCT	N/A
Types.NCHAR	N/A
Types.NCLOB	N/A
Types.NVARCHAR	N/A
Types.JAVA_OBJECT	N/A
Types.REF	N/A
Types.SQLXML	N/A
Types.STRUCT	N/A

표 26-4 GOLDILOCKS 타입에 대한 getter method 지원 여부-1

	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBLE
getByte	O	O	O	O	O
getShort	O	O	O	O	O
getInt	O	O	O	O	O
getLong	O	O	O	O	O
getFloat	O	O	O	O	O
getDouble	O	O	O	O	O
getBigDecimal	O	O	O	O	O
getBoolean	0,1일 경우만 가능	0,1일 경우만 가능	0,1일 경우만 가능	0,1일 경우만 가능	0,1일 경우만 가능
getString	O	O	O	O	O
getBytes	raw 데이터	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getDate	X	X	X	X	X
getTime	X	X	X	X	X
getTimestamp	X	X	X	X	X
getAsciiStream	raw 데이터	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getBinaryStream	raw 데이터	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getCharacterStream	X	X	X	X	X
getClob	X	X	X	X	X
getBlob	X	X	X	X	X
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Short	Integer	Long	Float	Double
getRowId	X	X	X	X	X

표 26-5 GOLDILOCKS 타입에 대한 getter method 지원 여부-2

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/ LONG VARBINARY	ROWID
getByte	0 or 1	O	숫자일 경우만 가능	X	X
getShort	0 or 1	O	숫자일 경우만 가능	X	X
getInt	0 or 1	O	숫자일 경우만 가능	X	X
getLong	0 or 1	O	숫자일 경우만 가능	X	X
getFloat	0 or 1	O	숫자일 경우만	X	X

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
			가능		
getDouble	0 or 1	O	숫자일 경우만 가능	X	X
getBigDecimal	0 or 1	O	숫자일 경우만 가능	X	X
getBoolean	O	0,1일 경우만 가능	"t", "f", "true", "false", "y", "n", "yes", "no", "on", "off", "1", "0" 일 경우만 가능(대소 문자구별안함)	X	X
getString	"TRUE" or "FALSE" "	O	O	O	O
getBytes	raw 데이터	raw 데이터	raw 데이터	O	raw 데이터
getDate	X	X	Date format일 경 우만 가능	X	X
getTime	X	X	Time format일 경 우만 가능	X	X
getTimestamp	X	X	Timestamp form at일 경우만 가능	X	X
getAsciiStream	raw 데이터	raw 데이터	raw 데이터	O	raw 데이터
getBinaryStream	raw 데이터	raw 데이터	raw 데이터	O	raw 데이터
getCharacterStream	X	X	O	X	X
getClob	X	X	X	X	X
getBlob	X	X	X	X	X
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Boolean	BigDecimal	String	byte[]	RowId
getRowId	X	X	X	X	O

표 26-6 GOLDILOCKS 타입에 대한 getter method 지원 여부-3

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL
getByte	X	X	X	단일 항목 타입만 가능
getShort	X	X	X	단일 항목 타입만 가능
getInt	X	X	X	단일 항목 타입만 가능

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL
getLong	X	X	X	단일 항목 타입만 가능
getFloat	X	X	X	단일 항목 타입만 가능
getDouble	X	X	X	단일 항목 타입만 가능
getBigDecimal	X	X	X	단일 항목 타입만 가능
getBoolean	X	X	X	X
getString	O	O	O	O
getBytes	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getDate	O	O	O	X
getTime	O	O	O	X
getTimestamp	O	O	O	X
getAsciiStream	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getBinaryStream	raw 데이터	raw 데이터	raw 데이터	raw 데이터
getCharacterStream	X	X	X	X
getClob	X	X	X	X
getBlob	X	X	X	X
getArray	X	X	X	X
getRef	X	X	X	X
getURL	X	X	X	X
getObject	Date	Time	Timestamp	GoldilocksInterval
getRowId	X	X	X	X

27.

Embedded SQL

27.1 Precompiler

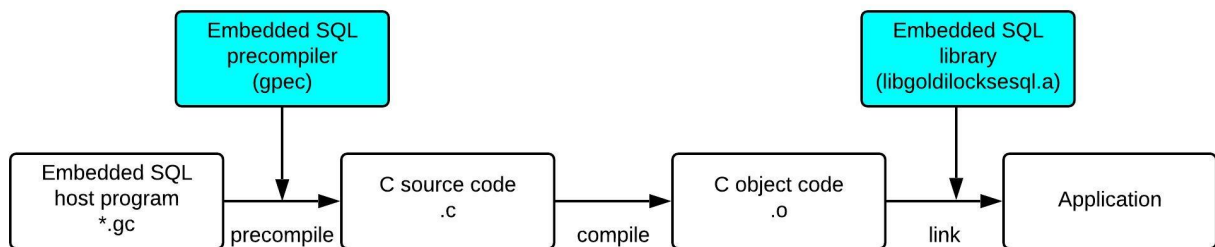
개요

GOLDILOCKS의 precompiler는 high-level 프로그래밍 언어에서 embedded SQL을 사용할 수 있게 해 주는 프로그램 개발 도구이다. 현재 GOLDILOCKS에서는 C/ C++ 언어에 대한 precompiler만 지원하는데 이 도구의 이름은 gpec이다.

Embedded SQL 응용 프로그램 개발

그림 1 Embedded SQL 응용 프로그램 개발에 설명된 것과 같이, 사용자가 embedded SQL을 포함하는 C 소스 프로그램을 작성하고, 이를 gpec precompiler를 통하여 변환하면 소스 코드 상에 있던 embedded SQL이 GOLDILOCKS의 library를 호출하는 내용으로 변환된 순수 C 코드가 만들어진다. 이 C 코드는 시스템의 C compiler를 이용하여 object code로 compile한 다음, GOLDILOCKS에서 제공하는 embedded SQL library인 libgoldilocksesql.a와 함께 링크하여 최종 목적인 응용 프로그램을 만든다.

그림 1 Embedded SQL 응용 프로그램 개발



Embedded SQL 응용 프로그램 개발 도구 구성

GOLDILOCKS의 embedded SQL 응용 프로그램은 다음과 같은 요소들로 구성되어 있다.

Directory or File	설명
\bin\gpec	GOLDILOCKS precompiler embedded SQL for C
\include\goldilocksesql.h	Embedded SQL library header file. Precompiler가 자동으로 삽입하기 때문에 사용자가 별도로 조작할 것은 없다.
\include\sqlca.h	SQLCA 자료 구조 관련 header file 이다.
\lib\libgoldilocksesql.a, \lib\libgoldilocksesqls.so	Embedded SQL run-time library 이다.
\lib\libgoldilocks.a, \lib\libgoldilockss.so	GOLDILOCKS DA/ CS 혼용 mode library 이다.

Directory or File	설명
\lib\libgoldilocks.a, \lib\libgoldilocks.as.so	GOLDILOCKS DA mode library 이다.
\lib\libgoldilocksc.a, \lib\libgoldilocks.cs.so	GOLDILOCKS CS mode library 이다.
\sample\EmbeddedSQL	Sample program 이다.

Building Application

본 절에서는 GOLDILOCKS의 embedded SQL 소스 프로그램을 build하여 실행 형태의 응용 프로그램을 만드는 과정을 설명한다.

Precompile

설명

사용자가 embedded SQL을 사용하여 작성한 C/ C++ 소스 코드를 precompile하여, 순수한 C/ C++ 소스 코드를 생성한다. 이 과정의 핵심은 사용자가 작성한 embedded SQL을 GOLDILOCKS에서 제공하는 library call로 변환하는 것이며, embedded SQL을 제외한 C/ C++ 소스 코드는 변환하지 않는다.

사용 방법

GOLDILOCKS의 precompiler 이름은 gpec이고 \$GOLDILOCKS_HOME/bin/에 위치한다. gpec은 다음과 같은 방법으로 사용한다.

```
$ gpec [OPTION]... <input file>
```

gpec은 <input file>을 입력 받아서 precompile 과정을 거친 다음 C/ C++ 소스 코드를 생성해낸다. <input file>은 기본적으로 *.gc 확장자를 가지고 있는데 이 확장자는 생략할 수 있다. 만약 <input file>이 *.gc 확장자를 가지고 있지 않을 경우, 반드시 file의 이름에 확장자까지 써 주어야 한다.

gpec에 주어지는 옵션에 대한 자세한 내용은 **Precompiler Options**를 참조한다.

Example

```
$ gpec sample1
FileName: sample1
Pre-compile sample1.gc -> sample1.c
```

Compile

Precompile 과정을 거쳐서 생성된 코드는 C/ C++ 소스 코드이다. 이 소스 코드는 platform에서 제공하는 C/ C++ compiler를 사용하여 object code를 생성한다. 이 과정에 대한 자세한 내용은 사용자 각자의 platform에서 제공하는 C/ C++ compiler 사용설명서를 참조한다.

Link

위의 절차들을 통해 생성된 object code들을 link하여 응용 프로그램을 생성하는데, GOLDILOCKS에서는 embedded SQL에 대해 libgoldilocksesql.a를 제공한다. 이 library는 precompiler가 embedded SQL을 변환한 GOLDILOCKS API들을 포함하므로 embedded SQL 응용 프로그램을 만들 때 반드시 필요하다.

추가적으로 GOLDILOCKS의 다양한 동작 모드에 따라서 필요한 library가 달라지게 되는데, 현재 응용 프로그램의 동작 모드에 따라 다음과 같이 library를 선택하여 링크한다.

동작 mode	Static library	Shared object
DA 전용	libgoldilocksa.a	libgoldilocksas.so
CS 전용	libgoldilocksc.a	libgoldilockscs.so
DA/ CS 혼용	libgoldilocks.a	libgoldilockss.so

그 밖의 link 과정 역시 일반적인 C/ C++ 응용 프로그램 생성 과정과 다르지 않으므로, platform에서 제공하는 linker 매뉴얼을 참조한다.

Example

위의 precompile, compile, link 과정을 편하게 수행하기 위하여 make를 많이 이용한다. 다음은 간단한 sample 프로그램을 만들기 위한 makefile의 예이다. 아래 예제를 참조하여 각자의 환경에 맞는 makefile을 만들어 사용해야 한다.

```
CC = gcc
CFLAGS = -g -Wall

INC = -I$(GOLDILOCKS_HOME)/include
LFLAGS = -L$(GOLDILOCKS_HOME)/lib

LIB = -lgoldilocksesql -lpthread -lm -lrt
ifeq ($(CSMODE), 1)
    LIB += -lgoldilocksc
else
    ifeq ($(MIXMODE), 1)
        LIB += -lgoldilocks
    else
```

```

        LIB += -lgoldilocksa
    endif
endif

GPEC = gpec
GPECFLAGS =
#GPECFLAGS = --unsafe-null --no-prompt
BINS = overview sample1 sample2 sample3 sample4 sample5 dyn1 dyn2 number date_time thread1
fetch_struct_array

ifneq ($(MAKECMDGOALS), clean)
ifneq ($(MAKECMDGOALS), all)
TARGET = $(MAKECMDGOALS)
OBJECT = $(TARGET).o
C_SRC = $(TARGET).c
endif
endif

```

- 암묵적 규칙

```

.SUFFIXES: .gc .c .o
.gc.c:
    $(GPEC) $(GPECFLAGS) $^          ❶ Precompile
.c.o:
    $(CC) $(CFLAGS) -c $(INC) $^     ❷ Compile

```

- Build 규칙

```

NoTarget :
    @echo "Syntax : make {all | sample_name | clean}"
    @echo "sample_name is one of '$(BINS)'"

all :
    for target in $(BINS); do \
        $(MAKE) $$target;      \
    done

$(OBJECT) : $(C_SRC)
$(TARGET) : $(OBJECT)
    $(CC) -o $@ $^ $(LFLAGS) $(LIB)  ❸ Link

clean :

```

```
rm -rf $(BINS) *.o *.c *~ core
```

Sample

GOLDILOCKS에서는 embedded SQL 응용 프로그램 작성에 대한 이해를 돕기 위해 간단한 embedded SQL sample 코드를 제공한다. Sample 코드는 \$GOLDILOCKS_HOME/sample/EmbeddedSQL 디렉토리에 있으며, 이 sample들을 수행하기 위해서는 해당 디렉토리에 함께 존재하는 sample.sql을 먼저 수행해야 한다.

```
$ cd $GOLDILOCKS/sample/EmbeddedSQL
$ gsql test test -i sample.sql
$ make
Syntax : make {all | sample_name | clean}
sample_name is one of 'overview sample1 sample2 sample3 sample4 sample5 dyn1 dyn2 number date_
time thread1 fetch_struct_array xa long_binary psm whenever preprocess'
```

make all은 모든 sample을 build하는데 특정 sample만 별도로 만들려면 make <sample_name>을 하면된다. <sample_name>은 위 메시지를 참조한다. sample2를 build하여 수행하면 다음과 같은 결과가 나온다.

```
$ make sample2
gpec sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
gcc -g -Wall -c -I/home/mycomman/work/product/Gliese/home/include sample2.c
gcc -o sample2 sample2.o -L/home/mycomman/work/product/Gliese/home/lib -lgoldilocksesql -
lpthread -lm -lrt -lgoldilocksa
$ ./sample2
Connect GOLDILOCKS ...
```

EMPNO	ENAME	JOB	SALARY
2854	Park	RND	800
2098	Kim	SALESMAN	1600
2175	Choi	SALESMAN	1250
2306	Lee	SUPPORT	2975
2122	Lyu	SALESMAN	1250
2999	Ohn	SUPPORT	2850
2012	Cheon	SUPPORT	2450
2168	Sohn	RND	3000
2836	Seo	CEO	5000
2022	Song	SALESMAN	1500
2232	Jeong	RND	1100
2676	Kang	RND	950


```

2714                Cho        RND        3000
2441                Yoon        RND        1300
=====
Record Count = 14
=====
SUCCESS
#####

```

Precompiler Options

본 절에서는 gpec의 option들을 설명한다.

--no-prompt, -n

설명

Version 정보를 출력하지 않는다.

사용 예

```

$ gpec --no-prompt sample2
FileName: sample2
Pre-compile sample2.gc -> sample2.c
$

```

--version, -v

설명

Version 정보만 출력하고 종료한다.

사용 예

```

$ gpec --version
$

```

--help, -h

설명

Help message를 출력한다. gpec에 아무런 option이나 <input file>을 주지 않을 때도 동일하게 동작한다.

사용 예

```
$ gpec --help
gpec is the GOLDILOCKS embedded SQL precompiler for C programs.

Usage:
  gpec [OPTION]... <input file>

Options:
  --no-prompt      No Print version information
  --version        Print version information and exit
  --help           Print help message
  --output         Describe output filename
  --unsafe-null    Allow a NULL fetch without indicator variable
  --include-path   Describe header file path
  --no-lineinfo    Exclude line information
  --char_map       Mapping of character arrays ( CHARZ | STRING )

$
```

--output, -o

설명

Precompile 결과 file의 이름을 지정한다. 이 옵션이 주어지지 않으면, <input file>과 같은 file 이름을 가지고 확장자가 .c 인 file이 만들어진다.

사용 예

- Output을 사용하지 않은 경우

```
$ gpec sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
$ ls
sample2.c  sample2.gc
```

- Output을 사용한 경우

```
$ gpec --output outfile.cpp sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> outfile.cpp
$ ls
outfile.cpp sample2.gc
```

--unsafe-null

설명

Host indicator variable을 사용하지 않은 경우에도 NULL fetch가 발생했을 때 성공하도록 해준다. 이것은 단지 연산이 성공했다는 의미일 뿐이지 NULL 값을 얻어올 수 있다는 의미는 아니다.

사용 예

```
$ gpec --unsafe-null sample2
FileName: sample2
Option : --unsafe-null
Pre-compile sample2.gc -> sample2.c
$
```

--include-path, -I

설명

Precompile을 할 때 참조해야 할 header file의 경로를 기술한다. Precompile을 수행하면서 EXEC SQL INCLUDE 구문을 통해 다른 header file을 찾게 되는데, 우선 현재 file이 위치한 디렉토리부터 검색하고, 없으면 이 옵션이 기술된 디렉토리들에서 차례대로 찾는다.

사용 예

Header file이 include 디렉토리에 존재하는 경우의 예이다.

- 에러가 발생한 경우

```
$ gpec sample.gc
FileName: sample.gc
Pre-compile sample.gc -> sample.c
ERR-42000(41000): syntax error
Error at line 12, in file sample.gc
```

```
ERR-42000(41004): "decl.h": file not exist
ERR-42000(41000): syntax error
rsEmpRecord gRecord[] = {
^
Error at line 15, in file sample.gc
```

- -I 옵션을 사용한 경우

```
$ gpec -Iinclude sample.gc
FileName: sample.gc
Pre-compile sample.gc -> sample.c
$
```

--no-lineinfo

설명

GPEC은 기본적으로 gc file을 c file로 변환할 때, gc file로 디버깅이 가능하도록 #line 정보를 추가한다. 그러나 이 옵션을 사용하면 c file을 생성할 때 #line preprocessor를 통한 line 정보를 추가하지 않는다.

사용 예

```
$ gpec --no-lineinfo sample2
FileName: sample2
Pre-compile sample2.gc -> sample2.c
$
```

--char_map, -c

설명

DECLARE SECTION 내에 선언된 char 형식의 data를 어떤 type으로 mapping을 할지 설정한다. Default 값은 'STRING'인데 NULL로 종료되는 data형식이다. 'CHARZ'는 space padding되고 NULL로 종료되는 data형식이다.

사용 예

```
$ gpec --char_map=STRING overview
FileName: overview
Pre-compile overview.gc -> overview.c
$ gpec --char_map=CHARZ overview
FileName: overview
```

Pre-compile overview.gc -> overview.c

--define, -D

설명

gpec에서 사용되는 define 이름으로써 1로 설정된다.

사용 예

```
$ gpec --define=AAA preprocess
FileName: preprocess
Pre-compile preprocess.gc -> preprocess.c
$ gpec -D BBB preprocess
FileName: preprocess
Pre-compile preprocess.gc -> preprocess.c
ERR-42000(41028): 'BBB' macro is already defined at line 45, in file preprocess.gc
```

27.2 Embedded SQL

Preprocessing

개요

gpec에서 precompiling하기 전에 전처리 (preprocessing)를 수행한다.

gpec에서 지원하는 preprocess 지시문은 #if, #ifdef, #if defined, #ifndef, #else, #elif, #endif, #define, #undef 등이다.

gpec의 --define 옵션으로 predefine을 사용할 수 있다. gpec의 옵션으로 predefine된 경우, 1로 정의(definition)된다.

예: gpec --define=_DEV_ Test.gc는 Test.gc 파일 내의 #define _DEV_ (1)과 동일하다.

적용 범위

gpec에 대한 SQL precompiler 기능은 declare section 내에서만 host 변수를 적용하고 EXEC SQL 문에 사용된다. (Declare section 외부에 있는 host 변수는 EXEC SQL 문에 사용할 수 없다.)

gpec preprocessor는 source 전체에 적용된다.

Include 파일에는 EXEC SQL INCLUDE로 선언한 header 파일만 적용된다.

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20]; ❶
#elif defined _DEV2_
VARCHAR conn_str[30]; ❷
#endif
EXEC SQL END DECLARE SECTION;
```

노트

❶ _DEV1_가 DECLARE SECTION의 위치와 상관없이 정의되어 있기 때문에 gpec에서 host 변수로 처리된다.

❷ _DEV1_가 정의되어 있기 때문에 해당 위치는 false가 되어 gpec에서 공백으로 처리된다.

gpec을 이용하여 위 파일을 c 파일로 생성하면 다음과 같이 conn_str[20]은 변환되고, #ifdef, #elif, #endif preprocessor 문장과 conn_str[30]은 공백으로 바뀐 것을 확인할 수 있다.

```
/* EXEC SQL BEGIN DECLARE SECTION; */
#define _DEV2_
char username[10];
char password[10];
/* VARCHAR conn_str[20]; */
struct { int len; char arr[20]; } conn_str; ❶

❷

/* EXEC SQL END DECLARE SECTION; */
```

노트

- ❶ _DEV1_가 DECLARE SECTION 위치와 상관없이 정의되어 있기때문에 gpec에서 host 변수로 처리된다.
- ❷ _DEV1_가 정의되어 있기 때문에 해당 위치는 false가 되어 gpec에서 공백으로 처리된다.

유형

#if

- 문법

```
#if constant
```

또는

```
#if defined identifier
```

또는

```
#if !defined identifier
```

- 예제

```
#if 0
int sVar1;
#endif
```

```

#if 3-2 ❶ 연산 가능
int sVar2;
#endif
#if defined _DEV_
int sVar3;
#endif
#if !defined (_DEV_)
int sVar4;
#endif

```

#ifdef, #ifndef

- 문법

```
#ifdef identifier
```

또는

```
#ifndef identifier
```

- 예제

```

#ifdef _DEV_
int sVar1;
#endif
#ifndef _DEV_
int sVar2;
#else
int sVar3;
#endif

```

#else, #elif, #endif

- 문법

```
#else
```

또는

```
#endif
```

또는

#elif constant

또는

#elif defined identifier

- 예제

```
#if 1
int sVar1;
#else
int sVar2;
#endif
#if 0
int sVar3;
#elif 1
int sVar4;
#else
int sVar5;
#endif
#ifdef _DEV1_
int sVar6;
#elif defined _DEV2_
int sVar7;
#elif !defined _DEV3_
int sVar8;
#endif
```

#define, #undef

- 문법

#define identifier

또는

#define identifier constant

또는

#undef identifier

- 예제

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20]; ❶
#elif defined _DEV2_
VARCHAR conn_str[30]; ❷
#endif
#undef _DEV2_
#ifdef _DEV2_
VARCHAR sDept[10]; ❸
#endif
EXEC SQL END DECLARE SECTION;
```

노트

- ❶ _DEV1_가 정의되어 있기 때문에 gpec에서 host 변수로 처리된다.
- ❷ _DEV2_는 #ifdef _DEV1_이 true이므로 gpec에서 공백으로 처리된다.
- ❸ _DEV2_가 undef 되었으므로 gpec에서 공백으로 처리된다.

주의

#define에는 주석을 사용할 수 있다. 그러나 #define이 여러 라인으로 나뉘어 작성되면 주석을 정확하게 처리할 수 없다.

```
#define _DEF1_ 1 \ ❶
    + 1
#define _DEF2_ 1 \ /* this ❷
is comment */ + 1
#define _DEF3_ 1 /* this is comment */ + 1 ❸
```

노트

- ❶ _DEF1_은 1 + 1로 처리된다.
- ❷ _DEF2_는 1로 처리된다.
- ❸ _DEF3_은 1 + 1로 처리된다.

제약사항

Declare section의 내부에 정의되었다고 하더라도 EXEC SQL 문까지 확장되지는 않는다.

```
EXEC SQL BEGIN DECLARE SECTION;
#define C_EMP_NO 14
char username[10];
EXEC SQL END DECLARE SECTION;
EXEC SQL
    SELECT USERNAME INTO :username
    FROM EMP
    WHERE EMPNO = C_EMP_NO; ❶ 잘못된 MACRO 사용
```

확장

MACRO는 c 선언문의 중간이나 EXEC SQL 문의 중간에 사용할 수 있다.

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char
#ifdef _DEV_
sTrue[10];
#else
sFalse[10];
#endif
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT
#ifdef _DEV_
    "true" INTO :sTrue
#else
    "false" INTO :sFalse
#endif
FROM DUAL;
```

gpec이 preprocessor를 선행 처리하여 다음과 같이 c code를 생성할 수 있다. 다음은 SQL statement가 c code로 변환되는 것이 생략된 예이다.

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char

sTrue[10];

EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT

    "true" INTO :sTrue

FROM DUAL;
```

연결

Database 연결

Embedded SQL 프로그램에서 database에 접속하여 작업을 수행하기 위해서는, database server에 연결하는 과정이 필요하다.

GOLDILOCKS에서 database에 연결하는 구문은 다음과 같다.

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password> [ AT <db_name> ] [
    USING <conn_string> ]
<db_name> := dbname | :hostvar
<user_name> := username | :hostvar
<password> := password | :hostvar
<conn_string> := connection_string | :hostvar
```

Database에 접속하는 가장 기본적인 방법은 다음과 같다.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
```

```
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
...
EXEC SQL CONNECT :username IDENTIFIED BY :password;
```

GOLDILOCKS는 shared memory에 직접 attach하여 구동하는 D/A 모드와, TCP 통신을 사용하여 database에 접속하는 C/S 모드 둘 다 지원한다. D/A 모드로 동작할 때는 database의 동일한 호스트에서 직접 접근하게 되므로 위와 같이 별도의 서버 정보를 포함하지 않아도 사용할 수 있지만 C/S 모드로 동작하기를 원할 때는 Data Source Name (DSN)을 지정하여 접근하도록 해야 한다. DSN에 대한 자세한 내용은 **데이터 원본 구성**을 참조한다.

DSN을 사용할 때는 connection_string 정보가 주어져야 하는데, 이 때 connection_string 정보를 사용하기 위하여 USING 절을 이용한다. 다음은 이름이 "GOLDILOCKS"인 DSN과 USING 절을 사용하는 연결 구문의 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
strcpy( conn_str, "DSN=GOLDILOCKS" );
...
EXEC SQL CONNECT :username IDENTIFIED BY :password USING :conn_str;
```

응용 프로그램을 개발하면서 각 연결을 독자적으로 식별해야 할 때가 있다. D/A 모드의 multi-thread 프로그램에서 각각 connection을 맺거나, C/S 모드에서 connection을 여러 개 맺는 경우가 이에 해당하는데, 이 때 각 connection에 이름을 부여하려면 AT 절을 사용한다.

AT 절은 CONNECT 구문의 가장 앞에 올 수도 있고, USING 절 앞에 위치하는 것도 가능하다. 다음은 AT 절을 사용한 CONNECT 구문의 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
char conn_name[10];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
strcpy( conn_str, "DSN=GOLDILOCKS" );
strcpy( conn_name, "DBCONN1" );
...
```

```
EXEC SQL CONNECT :username IDENTIFIED BY :password AT :conn_name USING :conn_str;
```

Database 연결 해제

응용 프로그램에서 database에 대한 연결을 해제한다. Connect 구문과 마찬가지로, 연결 해제 역시 기본연결을 해제하는 방법과 connection 이름을 주어서 해제하는 방법이 있다. 또한, 현재 응용 프로그램에서 연결한 모든 연결을 해제하는 구문도 제공한다.

단일 연결 해제

단일 연결 해제에는 명시적인 방법과 암묵적인 방법이 있다.

명시적인 방법은 DISCONNECT 구문을 사용하는 것으로써 그 사용 방법은 다음과 같다.

```
EXEC SQL [ AT <db_name> ] DISCONNECT;
```

Transaction은 commit이나 rollback을 통해 그 주기를 종료한다. 이 때 transaction 종료 구문 뒤에 RELEASE 옵션을 추가로 기술하여 암묵적으로 연결을 해제할 수 있다.

```
EXEC SQL [ AT <db_name> ] { COMMIT/ROLLBACK } [ WORK ] RELEASE;
```

전체 연결 해제

현재 응용 프로그램의 모든 연결을 한꺼번에 해제하기를 원할 경우, 다음과 같은 구문을 사용한다.

```
EXEC SQL DISCONNECT ALL;
```

Transaction

Database 응용 프로그램은 트랜잭션 단위로 구성된다. 따라서 embedded SQL 프로그램 역시 트랜잭션을 조작할 수 있어야 하며, 본 절에서는 그 방법에 대해 설명한다.

Transaction의 시작과 종료

Transaction이 connection 된 후에 가장 처음으로 수행되는 SQL에서 시작된다. 이렇게 시작된 transaction은 명시적으로 종료 명령이 발생할 때까지 유지되며, 종료 명령은 완료 (COMMIT) 명령과 취소 (ROLLBACK) 명령 두 가지가 있다.

COMMIT

Transaction의 완료 명령이 주어지면 다음과 같은 작업들이 발생한다.

- 현재 transaction이 시작되고 나서 발생한 모든 data 갱신이 database에 영구적으로 반영된다.
- 반영된 갱신 사항이 이후에 시작된 모든 transaction이나 sensitive한 cursor에 visible하게 적용된다.
- 현재 transaction에서 생성된 모든 savepoint가 삭제된다.
- 현재 transaction에서 획득한 모든 lock이 해제된다.
- 현재 transaction에서 open 된 cursor들이 close된다. (단, holdable cursor는 예외이다.)
- Transaction을 종료한다

완료 명령은 다음과 같이 사용한다.

```
EXEC SQL COMMIT [ WORK ];
```

ROLLBACK

취소 (rollback) 명령의 종류는 다음과 같다.

- Transaction 전체 rollback
- Transaction 부분 rollback
- Statement-level rollback

Transaction rollback 명령이 주어진다면 다음과 같은 작업들이 발생한다.

- 현재 transaction이 시작되고 나서 발생한 모든 data 갱신 사항이 취소되어 transaction 발생 이전의 상태로 되 돌아간다.
- 현재 transaction에서 생성된 모든 savepoint가 삭제된다.
- 현재 transaction에서 획득한 모든 lock이 해제된다.
- 현재 transaction에서 open 된 cursor들이 close된다. (단, holdable cursor는 예외이다.)
- Transaction을 종료한다

취소 명령은 다음과 같이 사용한다.

```
EXEC SQL ROLLBACK [ WORK ];
```

Transaction을 부분 rollback하려면 savepoint를 이용한다. 응용 프로그램 개발자는 명시적으로 savepoint를 지정하고, 이 savepoint까지 작업을 되돌려 취소하도록 함으로써 transaction을 부분적으로 rollback 할 수 있다. Transaction을 부분 rollback하면 다음과 같은 작업들이 발생한다.

- 취소된 savepoint 이후에 발생한 모든 갱신 사항이 취소된다.
- 취소된 savepoint 이후에 생성된 savepoint들이 삭제된다.
- 취소된 savepoint 이후에 획득한 lock들이 해제된다.

Transaction을 부분적으로 취소하는 명령은 다음과 같이 사용한다.

```
EXEC SQL ROLLBACK TO SAVEPOINT <savepoint_name>;
```

Statement-level rollback은 현재 수행중이었던 statement만 단독으로 취소하는 기능이다. 예를 들어, table에 row를 삽입하다가 unique violation이 발생한 경우처럼, 현재 statement를 더 이상 수행할 수 없는 경우에는 현재 statement만 취소해야 다음 작업을 계속 진행할 수 있다.

이 경우, GOLDDILOCKS 내부에서 자체적으로 statement를 취소하기 때문에 별도로 구문상으로 지정할 명령은 없다.

Auto Commit

일반적으로 GOLDDILOCKS embedded SQL이 connection 될 때 transaction은 non auto-commit mode로 작동한다.

그러나 응용 프로그램의 개발 편의성이나, 특정 응용 프로그램의 논리 환경에서 auto-commit mode를 수정해야 할 때도 있다. 이러한 경우에 대비하여 GOLDDILOCKS의 embedded SQL precompiler에서는 다음과 같은 구문을 사용하여 auto-commit mode를 켜거나 끌 수 있다.

```
EXEC SQL [ AT <db_name> ] AUTOCOMMIT { ON | OFF };
```

RELEASE Option

Transaction을 종료 (commit/ rollback) 할 때 RELEASE option을 사용하여 현재 사용 중인 connection을 해제할 수 있다.

이 옵션은 transaction 전체 종료에만 적용할 수 있고 transaction을 부분적으로 취소 (ROLLBACK TO SAVEPOINT) 할 때는 사용할 수 없다.

Host Variables and Datatypes

Embedded SQL 응용 프로그램은 database server와 연동하여 data를 조작 (manipulation)하고 질의(query)하여 원하는 결과를 획득하는 것을 목적으로 한다.

이러한 작업을 위해서는 응용 프로그램의 data를 database server로 전달하고, database server로부터 data를 얻어올 수 있는 수단이 필요한데 이 역할을 수행하는 매개체를 host variable이라고 정의한다.

Host variable은 C 언어의 변수로 선언되기 때문에, 응용 프로그램은 C 변수를 사용하는 것과 동일한 방법으로 host variable을 사용할 수 있고 이 변수가 SQL 문의 일부처럼 처리되어 database server와 응용 프로그램 사이의 value 입출력을 담당한다.

Host Variable 선언

Host variable은 다음과 같이 embedded SQL directive 내에 선언되어야 한다.

```
EXEC SQL BEGIN DECLARE SECTION;
```

- Host variable 선언

```
EXEC SQL END DECLARE SECTION;
```

위와 같은 영역을 declare section이라고 하며 declare section 내에는 host variable을 선언할 수 있는데, 그 선언 방법은 C variable을 선언하는 방법과 동일하다. 다음은 일부 host variable을 선언하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    int      empno;
    char      ename[20];
    double    salary;
EXEC SQL END DECLARE SECTION;
```

Host Variable을 위한 C Data Type

Host variable로 사용되는 C data type은 C 언어에서 제공하는 native type과 GOLDILOCKS에서 부가적으로 제공하는 data type이 있다. 다음 표는 GOLDILOCKS의 embedded SQL에서 제공하는 data type을 설명한다.

C Native Datatype

C native datatype은 C 언어에서 제공하는 기본적인 type으로써 그 범위나 크기가 응용 프로그램을 개발할 때 사용된 platform에 전적으로 종속된다.

표 27-1 C native datatype

C datatype	설명
char	single character
char[n]	최대 길이가 n인 문자열
short	small integer (2 bytes)
int	integer (4 bytes)
long	large integer (4/8 bytes)
long long	very large integer (8 bytes)
float	single precision floating-point number
double	double precision floating-point number

- char

char type은 single character를 나타낸다.

- char[n]

char[n] type은 최대 길이 n을 갖는 문자열(string) data를 나타낸다.
다음 예제를 참조한다.

```
EXEC SQL BEGIN DECLARE SECTION;
char strName[20];
EXEC SQL BEGIN DECLARE SECTION;
```

위와 같이 선언할 경우, strName은 최대 길이 20을 갖는 문자열 data를 의미한다.
strName이 ESQL의 output으로 사용되면 space padding이 된다.

노트

char[]의 최대 길이는 2001을 초과할 수 없다.

- short

2 byte 정수형 datatype을 나타낸다.

- int

4 byte 정수형 datatype을 나타낸다.

- long

Long type은 large integer라는 의미이지만 그 값의 실제 범위는 platform에 따라 결정된다. 64 bit Unix/ Linux platform에서는 long이 8 byte형 정수이지만 32 bit Unix/ Linux platform에서는 4 byte 정수이다.

- long long

Very large integer라는 의미이고 8 byte 정수형을 표현한다.

- float

C 언어의 float type으로써 4 byte single-precision floating-point number를 나타낸다.

- double

C 언어의 double type으로써 double-precision floating-point number를 나타낸다.

Pseudo Datatype

Pseudo type은 다양한 형태의 GOLDBLOCKS type을 지원하고 개발 편의성을 위해 GOLDBLOCKS embedded SQL precompiler에서 제공하는 type으로써 그 내용의 대부분은 C의 구조체를 사용하여 구현된다.

표 27-2 GOLDBLOCKS embedded SQL pseudo type

Pseudo type	설명
VARCHAR[n]	최대 길이가 n인 가변 길이 문자열
LONG VARCHAR	최대 길이가 100M (104857600)인 가변 길이 문자열
BINARY[n]	최대 길이가 n인 binary data
VARBINARY[n]	최대 길이가 n인 가변 길이 binary data
LONG VARBINARY	최대 길이가 100M (104857600)인 가변 길이 binary data
NUMBER	유효 자릿수가 38 자리인 정수
NUMBER(p)	유효 자릿수가 p 자리인 정수
NUMBER(p, s)	유효 자릿수가 p, scale이 s인 실수
BOOLEAN	Boolean type
DATE	날짜형 data
TIME	시간형 data
TIME WITH TIMEZONE	Timezone을 갖는 시간형 data
TIMESTAMP	날짜 시간형 data
TIMESTAMP WITH TIMEZONE	Timezone을 갖는 날짜 시간형 data
INTERVAL YEAR	Interval data type
INTERVAL MONTH	
INTERVAL DAY	
INTERVAL HOUR	
INTERVAL MINUTE	
INTERVAL SECOND	
INTERVAL YEAR TO MONTH	
INTERVAL DAY TO HOUR	
INTERVAL DAY TO MINUTE	
INTERVAL DAY TO SECOND	
INTERVAL HOUR TO MINUTE	
INTERVAL HOUR TO SECOND	
INTERVAL MINUTE TO SECOND	

VARCHAR

VARCHAR type은 가변 길이 문자열을 저장할 수 있는 datatype으로써 다음과 같은 구조체로 구성되어 있다.

```
struct {
    int len;
    char arr[n];
}
```

```
}
```

여기서 n은 VARCHAR type이 가질 수 있는 최대 길이를 의미한다.

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varstr[100];
EXEC SQL END DECLARE SECTION;
```

예를 들어 위와 같이 선언할 경우 precompile 과정에서 다음과 같이 변환된다.

```
struct VARCHAR_varstr {
    int len;
    char arr[100];
} varstr;
```

응용 프로그램에서는 다음과 같은 방법으로 VARCHAR type을 사용할 수 있다.

```
strcpy( varstr.arr, "abcde" );
varstr.len = strlen( varstr.arr );

EXEC SQL INSERT INTO TEST_T1 VALUES ( :varstr );
```

노트

VARCHAR의 길이는 4000을 초과할 수 없다

LONG VARCHAR

LONG VARCHAR type은 길이가 긴 가변 길이 문자열을 저장할 수 있는 datatype으로써 다음과 같은 구조체로 구성되어 있다.

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT len;
    SQLCHAR * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

LONG VARCHAR type은 다음과 같이 선언할 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARCHAR long_text;
EXEC SQL END DECLARE SECTION;
```

LONG VARCHAR와 VARCHAR의 차이점은 LONG VARCHAR 길이는 최대 100M (104857600)까지 가능하므로 선언할 때 문자열을 저장할 공간을 미리 할당해 놓지 않는다는 것이다. 즉, LONG VARCHAR를 선언한 후에는 실제로 사용하기 전에 long_text.arr에 실제 메모리 공간을 할당해 주어야 하며, 사용 후에 이 메모리 공간을 해제하는 것 역시 응용 프로그램에서 담당해야 한다. 다음은 LONG VARCHAR를 사용하는 예이다.

```
long_text.arr = malloc( 1048576 );

gets( long_text.arr );
long_text.len = strlen( long_text.arr );

EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_text );
...
free( long_text.arr );
```

노트

LONG VARCHAR type의 길이는 현재 최대 100M (104857600)까지 지정하여 선언할 수 있다.

BINARY

BINARY type은 정형화되지 않은 raw data를 그대로 다루기 위한 datatype으로써 다음과 같이 구성되어 있다. 다음은 BINARY type을 선언하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    BINARY binary[100];
EXEC SQL END DECLARE SECTION;
```

위와 같이 선언하면 precompile 과정에서 다음과 같이 변환된다.

```
char binary[100];
```

문자열 저장과 형태는 같지만 char[n]로 정의하는 것이 database의 문자열을 저장하기 위한 것이라면 BINARY type의 내용은 binary data라는 것만 다르다.

따라서 응용 프로그램에서는 문자열 데이터를 다루는 방법과 동일한 방법으로 BINARY type을 사용할 수 있다.

노트

BINARY의 최대 길이는 2000을 초과할 수 없다

VARBINARY

VARBINARY type은 가변 길이 binary data를 저장할 수 있는 datatype으로써 다음과 같은 구조체로 구성되어 있다.

```
struct {
    int len;
    char arr[n];
}
```

여기에서 n은 VARCHAR type이 가질 수 있는 최대 길이를 의미한다.

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varbin[100];
EXEC SQL END DECLARE SECTION;
```

예를 들어 위와 같이 선언할 경우 precompile 과정에서 다음과 같이 변환된다.

```
struct VARBINARY_varbin {
    int len;
    char arr[100];
} varbin;
```

VARBINARY type은 VARCHAR type과 그 형태가 동일하다. VARCHAR는 문자열을 저장하기 위해 사용되는 반면에 VARBINARY는 binary data를 다룬다는 것 외에는 사용방법을 포함한 모든 면에서 동일하다. 따라서 응용 프로그램에서는 VARCHAR를 사용할 때처럼 다음과 같은 방법으로 VARBINARY type을 사용할 수 있다.

```
memcpy( varbin.arr, binary_data, 50 );
varbin.len = 50;

EXEC SQL INSERT INTO TEST_T1 VALUES ( :varbin );
```

노트

VARBINARY의 길이는 4000을 초과할 수 없다

LONG VARBINARY

LONG VARBINARY type은 길이가 긴 가변 길이 binary data를 저장할 수 있는 datatype으로써 다음과 같은 구조체로 구성되어 있다.

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT    len;
    SQLCHAR      * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

LONG VARBINARY type은 다음과 같이 선언할 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARBINARY long_bin;
EXEC SQL END DECLARE SECTION;
```

LONG VARBINARY와 VARBINARY의 차이점은 LONG VARBINARY type을 선언할 때 binary data를 저장할 공간을 미리 할당해 놓지 않는다는 것이다. (LONG VARCHAR와 VARCHAR의 차이점과 같다.) 즉, LONG VARBINARY를 선언한 후에는 실제로 사용하기 전에 long_bin.arr에 실제 메모리 공간을 할당해 주어야 하며, 사용 후에 이 메모리 공간을 해제하는 것 역시 응용 프로그램에서 담당해야 한다. 다음은 LONG VARBINARY를 사용하는 예이다.

```
long_bin.arr = malloc( 1048576 );

memcpy( long_bin.arr, long_binary_data, 1048576);
long_bin.len = 1048576;

EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_bin );
...
free( long_bin.arr );
```

노트

LONG VARBINARY type의 길이는 현재 최대 100M (104857600)까지 지정하여 선언할 수 있다.

NUMBER

NUMBER type은 ODBC에서 정의된 SQL_NUMERIC_STRUCT를 응용 프로그램에서 사용할 수 있도록 제공하는 datatype이다.

```
#define SQL_MAX_NUMERIC_LEN 16
typedef struct tagSQL_NUMERIC_STRUCT
{
    SQLCHAR precision;
    SQLSCHAR scale;
    SQLCHAR sign; /* 1=pos 0=neg */
    SQLCHAR val[SQL_MAX_NUMERIC_LEN];
} SQL_NUMERIC_STRUCT;
```

NUMBER type은 precision과 scale을 가지는 실수형 data를 표현할 수 있도록 구성되어 있다. NUMBER type을 선언할 때는 precision과 scale을 결정하여 선언할 수 있고 경우에 따라 scale과 precision을 생략할 수도 있는데 그 의미는 다음과 같다.

표 27-3 NUMBER type의 precision과 scale

NUMBER type 선언	설명
NUMBER	NUMBER(38, 0)와 동일하다.
NUMBER(p)	NUMBER(p, 0)와 동일하다.
NUMBER(p, s)	Precision이 p이고 scale이 s인 실수형 data이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    NUMBER      number_default;
    NUMBER(20)   number_20;
    NUMBER(30,10) number_30_10;
EXEC SQL END DECLARE SECTION;
```

예를 들어 위와 같이 변수를 선언하였을 경우, number_default 변수는 NUMBER(38, 0)으로 선언된 것과 동일하고, number_20 변수는 NUMBER(20, 0)으로 선언된 것과 동일하다. NUMBER type은 ODBC type인 SQL_NUMERIC_STRUCT를 사용한다. 다음은 NUMBER type을 사용하는 예이다.

```
/*
 * number.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
```



```

        printf("\n");
        printf(aMsg);
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
            sqlca.sqlcode,
            SQLSTATE,
            sqlca.sqlerrm.sqlerrmc );
    }

```

```
int Connect(char *aHostInfo, char *aUserID, char *sPassword);
```

```
int CreateTable();
```

```
int DropTable();
```

```
unsigned long long ConvertMantisaToDecimal(SQLCHAR *aNumStrValue)
```

```

{
    unsigned long long sResult = 0;
    unsigned long long sLast=1;
    unsigned int sCurrent;
    unsigned int sLSD = 0;
    unsigned int sMSD = 0;
    int i = 1;
    for(i = 0; i < SQL_MAX_NUMERIC_LEN; i ++)
    {
        sCurrent = (unsigned char) aNumStrValue[i];
        sLSD = sCurrent % 16; //Obtain LSD
        sMSD = sCurrent / 16; //Obtain MSD
        sResult += sLast * sLSD;
        sLast = sLast * 16;
        sResult += sLast * sMSD;
        sLast = sLast * 16;
    }
    return sResult;
}

```

```
void PrintNumber(SQL_NUMERIC_STRUCT *aNumber)
```

```

{
    unsigned long long sDigit;
    unsigned long long sFraction;
    unsigned long long sMantisa;
    unsigned long long sFactor;
    int i;
    sMantisa = ConvertMantisaToDecimal( aNumber->val );

```

```

    sFactor = 1;
    for( i = 0; i < aNumber->scale; i ++ )
    {
        sFactor *= 10;
    }
    sDigit = sMantisa / sFactor;
    sFraction = sMantisa % sFactor;
    if( sFraction != 0 )
    {
        printf("%llu.%-3llu", sDigit, sFraction);
    }
    else
    {
        printf("%llu", sDigit);
    }
}

int main(int    argc,
          char  **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    NUMBER      sNumber;
    NUMBER(10,5) sResultNumber1;
    NUMBER(10,5) sResultNumber2;
    char        sCharNumber[20];
    int         sNo;
    int         sResultNo;
    EXEC SQL END DECLARE SECTION;
    int i;
    int sState = 0;
    printf("#### Number Datatype Test ####\n");
    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    printf("Create table ...\n");
    if(CreateTable() != SUCCESS)
    {
        goto fail_exit;
    }
}

```

```

sState = 1;
printf("Insert record ...\n");
for(i = 0; i < 20; i++)
{
    sNo = i + 1;
    memset( &sNumber, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
    sNumber.precision = 38;
    sNumber.scale = 3;
    sNumber.sign = 1;
    /*
     * 0x627d = 25213
     */
    sNumber.val[0] = 0x7d + i;
    sNumber.val[1] = 0x62;
    snprintf( sCharNumber, 20, "25.2%02d", 13 + i );
    EXEC SQL
        INSERT INTO TEST_T1(C1, C2, C3)
        VALUES(:sNo, :sNumber, :sCharNumber);
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}
EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
printf("Retrive record\n");
EXEC SQL
    DECLARE CUR1 CURSOR FOR
    SELECT C1, C2, C3
    FROM   TEST_T1;
EXEC SQL OPEN CUR1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
printf(" NO   Number1  Number2\n");
printf("==== =====\n");
memset( &sResultNumber1, 0x00, sizeof(SQL_NUMERIC_STRUCT) );

```

```

memset( &sResultNumber2, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
while( 1 )
{
    EXEC SQL
        FETCH FROM CUR1
        INTO :sResultNo, :sResultNumber1, :sResultNumber2;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        /*
         * No more data
         */
        break;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    printf("%3d  ", sResultNo);
    PrintNumber( &sResultNumber1 );
    printf("  ");
    PrintNumber( &sResultNumber2 );
    printf("\n");
}
printf("==== ===== \n");
EXEC SQL CLOSE CUR1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
sState = 0;
printf("Drop table ... \n");
if(DropTable() != SUCCESS)
{
    goto fail_exit;
}
printf("Disconnect GOLDILOCKS ... \n");

```

```

EXEC SQL COMMIT WORK RELEASE;
printf("SUCCESS\n");
printf("#####\n");
return 0;
fail_exit:
PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
printf("FAILURE\n");
printf("#####\n\n");
switch(sState)
{
    case 1:
        printf("Drop table ...\n");
        (void)DropTable();
    default:
        break;
}
EXEC SQL ROLLBACK WORK RELEASE;
return 0;
}

int CreateTable()
{
    EXEC SQL DROP TABLE IF EXISTS TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

```

- Create table

```

EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  NUMERIC(38,4),
                                C3  VARCHAR(20) );

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

```

```

    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;
    return FAILURE;
}

```

- Drop table

```

int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;
    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR sUId[80];
    VARCHAR sPwd[20];
    VARCHAR sConnStr[1024];
    EXEC SQL END DECLARE SECTION;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUId.arr, aUserID);
sUId.len = (short)strlen((char *)sUId.arr);
strcpy((char *)sPwd.arr, sPassword);

```

```
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB 연결

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");
    return FAILURE;
}
```

BOOLEAN

BOOLEAN type은 TRUE나 FALSE 값을 갖는다. 실제로는 이 변수가 C 언어의 변수로 사용되기 때문에 변수값이 1이면 TRUE를, 0이면 FALSE의 의미를 갖는다.

다음은 BOOLEAN type 변수를 사용하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    BOOLEAN boolean;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT IsEnable INTO :boolean FROM STATUS WHERE ID = 100;

if( boolean != 0 )
{
    print( "Enable Status : TRUE\n" );
}
else
{
    print( "Enable Status : FALSE\n" );
}
```

DATE

Date type은 날짜와 시간을 다룰 수 있다. Date type은 ODBC의 SQL_TIMESTAMP_STRUCT를 구조로 갖는다.

```
typedef struct tagTIMESTAMP_STRUCT
{
    SQLSMALLINT    year;
    SQLUSMALLINT    month;
    SQLUSMALLINT    day;
    SQLUSMALLINT    hour;
    SQLUSMALLINT    minute;
    SQLUSMALLINT    second;
    SQLINTEGER      fraction;
} TIMESTAMP_STRUCT;
typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

DATE 타입의 변수를 선언하면, precompiler가 위와 같은 구조체로 변환시키는데 각 field의 의미는 ODBC의 SQL_TIMESTAMP_STRUCT와 같다.

TIME

TIME type은 시간을 다루는 datatype으로써 ODBC의 SQL_TIME_STRUCT를 구조로 갖는다.

```
typedef struct tagTIME_STRUCT
{
    SQLUSMALLINT    hour;
    SQLUSMALLINT    minute;
    SQLUSMALLINT    second;
} TIME_STRUCT;
typedef TIME_STRUCT SQL_TIME_STRUCT;
```

TIME 타입의 변수를 선언하면 precompiler가 위와 같은 구조체로 변환시키는데 각 field의 의미는 ODBC의 SQL_TIME_STRUCT와 같다.

TIME WITH TIMEZONE

TIME WITH TIMEZONE type은 timezone이 있는 시간을 다루는 datatype으로써 다음과 같은 구조를 갖는다.

```
typedef struct tagTIME_WITH_TIMEZONE_STRUCT
{
    SQLUSMALLINT    hour;
    SQLUSMALLINT    minute;
    SQLUSMALLINT    second;
    SQLINTEGER      fraction;
    SQLSMALLINT     timezone_hour;
    SQLSMALLINT     timezone_minute;
} TIME_WITH_TIMEZONE_STRUCT;
typedef TIME_WITH_TIMEZONE_STRUCT SQL_TIME_WITH_TIMEZONE_STRUCT;
```


TIME WITH TIMEZONE 타입의 변수를 선언하면 precompiler가 위와 같은 구조체로 변환시키는데 각 field의 의미는 다음과 같다.

표 27-4 TIME WITH TIMEZONE의 field

파일 이름	설명
hour	시간
minute	분
second	초
fraction	소수점 이하 초
timezone_hour	Timezone의 시간
timezone_minute	Timezone의 분

TIMESTAMP

TIMESTAMP type은 날짜~시간을 다루는 datatype으로써 ODBC의 SQL_TIMESTAMP_STRUCT를 구조로 갖는다.

```
typedef struct tagTIMESTAMP_STRUCT
{
    SQLSMALLINT    year;
    SQLUSMALLINT   month;
    SQLUSMALLINT   day;
    SQLUSMALLINT   hour;
    SQLUSMALLINT   minute;
    SQLUSMALLINT   second;
    SQLINTEGER      fraction;
} TIMESTAMP_STRUCT;

typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

TIMESTAMP 타입의 변수를 선언하면 precompiler가 위와 같은 구조체로 변환시키는데 각 field의 의미는 ODBC의 SQL_TIMESTAMP_STRUCT와 같다.

TIMESTAMP WITH TIMEZONE

TIMESTAMP WITH TIMEZONE type은 timezone이 있는 timestamp를 다루는 datatype으로써 다음과 같은 구조를 갖는다.

```
typedef struct tagTIMESTAMP_WITH_TIMEZONE_STRUCT
{
    SQLSMALLINT    year;
    SQLUSMALLINT   month;
    SQLUSMALLINT   day;
    SQLUSMALLINT   hour;
    SQLUSMALLINT   minute;
```

```

    SQLSMALLINT second;
    SQLINTEGER fraction;
    SQLSMALLINT timezone_hour;
    SQLSMALLINT timezone_minute;
} TIMESTAMP_WITH_TIMEZONE_STRUCT;
typedef TIMESTAMP_WITH_TIMEZONE_STRUCT SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT;

```

TIMESTAMP WITH TIMEZONE 타입의 변수를 선언하면 precompiler가 위와 같은 구조체로 변환시키는데 각 field의 의미는 다음과 같다.

표 27-5 TIMESTAMP WITH TIMEZONE의 field

Field 이름	설명
year	연
month	월
day	일
hour	시간
minute	분
second	초
fraction	소수점 이하 초
timezone_hour	Timezone의 시간
timezone_minute	Timezone의 분

다음은 DATE, TIME, TIMESTAMP, TIME WITH TIMEZONE, TIMESTAMP WITH TIMEZONE type을 다루는 간단한 예이다.

```

/*
 * date_time.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1

#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
}

```

```

        printf(aMsg);
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
            sqlca.sqlcode,
            SQLSTATE,
            sqlca.sqlerrm.sqlerrmc );
    }

```

```
int Connect(char *aHostInfo, char *aUserID, char *sPassword);
```

```
int CreateTable();
```

```
int DropTable();
```

```
int main(int    argc,
        char **argv)
{

```

```
    EXEC SQL BEGIN DECLARE SECTION;
```

```
    int                sNo;
```

```
    int                sResultNo;
```

```
    DATE               sDate, sResultDate;
```

```
    TIME               sTime, sResultTime;
```

```
    TIME WITH TIMEZONE sTimeTz, sResultTimeTz;
```

```
    TIMESTAMP          sTimestamp, sResultTimestamp;
```

```
    TIMESTAMP WITH TIMEZONE sTimestampTz, sResultTimestampTz;
```

```
    EXEC SQL END DECLARE SECTION;
```

```
    int  sState = 0;
```

```
    printf("#### Datatype Insert Test ####\n");
```

```
    printf("Connect GOLDILOCKS ...\n");
```

```
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
```

```
    {
```

```
        goto fail_exit;
```

```
    }
```

```
    sState = 1;
```

```
    printf("Create table ...\n");
```

```
    if(CreateTable() != SUCCESS)
```

```
    {
```

```
        goto fail_exit;
```

```
    }
```

```
    sState = 2;
```

```
    printf("Insert record ...\n");
```

```
    sNo = 1;
```

```
/**
 * Date : 2014-7-14 21:29:30
 */
sDate.year   = 2014;
sDate.month  = 7;
sDate.day    = 14;
sDate.hour   = 21;
sDate.minute = 29;
sDate.second = 30;
sDate.fraction = 0;

/**
 * Time : 17:46:35
 */
sTime.hour    = 17;
sTime.minute  = 46;
sTime.second  = 35;

/**
 * Time With Timezone : 17:46:35.6789(+9:00)
 */
sTimeTz.hour      = 17;
sTimeTz.minute    = 46;
sTimeTz.second    = 35;
sTimeTz.fraction   = 678900000;
sTimeTz.timezone_hour = 9;
sTimeTz.timezone_minute = 0;

/**
 * Timestamp : 2014-02-13 17:46:28.123
 */
sTimestamp.year    = 2014;
sTimestamp.month   = 2;
sTimestamp.day     = 13;
sTimestamp.hour    = 17;
sTimestamp.minute  = 46;
sTimestamp.second  = 28;
sTimestamp.fraction = 123000000;

/**
 * Time With Timezone : 2014-05-18 17:46:35.001(+9:00)
```

```

    */
    sTimestampTz.year      = 2014;
    sTimestampTz.month    = 5;
    sTimestampTz.day       = 18;
    sTimestampTz.hour      = 17;
    sTimestampTz.minute    = 46;
    sTimestampTz.second     = 35;
    sTimestampTz.fraction = 1000000;
    sTimestampTz.timezone_hour = 9;
    sTimestampTz.timezone_minute = 0;

/**
 * Insert record
 */
EXEC SQL
    INSERT INTO TEST_T1(C1, C2, C3, C4, C5, C6)
    VALUES(:sNo, :sDate, :sTime, :sTimeTz, :sTimestamp, :sTimestampTz);
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

printf("Retrive record\n");
EXEC SQL
    SELECT C1, C2, C3, C4, C5, C6
    INTO   :sResultNo, :sResultDate, :sResultTime, :sResultTimeTz, :sResultTimestamp, :
sResultTimestampTz
    FROM   TEST_T1
    WHERE  C1 = 1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

printf("=====\n");

```

```

printf( "DATE                      : %04d-%02d-%02d %02d:%02d:%02d\n",
        sResultDate.year,
        sResultDate.month,
        sResultDate.day,
        sResultDate.hour,
        sResultDate.minute,
        sResultDate.second );

printf( "TIME                      : %02d:%02d:%02d\n",
        sResultTime.hour,
        sResultTime.minute,
        sResultTime.second );

printf( "TIME WITH TIMEZONE      : %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
        sResultTimeTz.hour,
        sResultTimeTz.minute,
        sResultTimeTz.second,
        sResultTimeTz.fraction,
        sResultTimeTz.timezone_hour,
        sResultTimeTz.timezone_minute );

printf( "TIMESTAMP                : %04d-%02d-%02d %02d:%02d:%02d.%09u\n",
        sResultTimestamp.year,
        sResultTimestamp.month,
        sResultTimestamp.day,
        sResultTimestamp.hour,
        sResultTimestamp.minute,
        sResultTimestamp.second,
        sResultTimestamp.fraction );

printf( "TIMESTAMP WITH TIMEZONE: %04d-%02d-%02d %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
        sResultTimestampTz.year,
        sResultTimestampTz.month,
        sResultTimestampTz.day,
        sResultTimestampTz.hour,
        sResultTimestampTz.minute,
        sResultTimestampTz.second,
        sResultTimestampTz.fraction,
        sResultTimestampTz.timezone_hour,
        sResultTimestampTz.timezone_minute );

```

```

printf("=====\\n");

sState = 0;
printf("Drop table ...\\n");
if(DropTable() != SUCCESS)
{
    goto fail_exit;
}

sState = 0;
printf("Disconnect GOLDSLOCKS ...\\n");
EXEC SQL COMMIT WORK RELEASE;

printf("SUCCESS\\n");
printf("#####\\n");

return 0;

fail_exit:
printf("\\n");
PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

printf("FAILURE\\n");
printf("#####\\n\\n");

switch(sState)
{
    case 1:
        printf("Drop table ...\\n");
        (void)DropTable();
    default:
        break;
}

EXEC SQL ROLLBACK WORK RELEASE;

return 0;
}

int CreateTable()
{

```

```
EXEC SQL DROP TABLE IF EXISTS TEST_T1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
```

- Create table

```
EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  DATE,
                                C3  TIME,
                                C4  TIME WITH TIME ZONE,
                                C5  TIMESTAMP,
                                C6  TIMESTAMP WITH TIME ZONE );

if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

EXEC SQL ROLLBACK WORK;

return FAILURE;
}
```

- Drop table

```
int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
```



```

        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR   sUid[80];
    VARCHAR   sPwd[20];
    VARCHAR   sConnStr[1024];
    EXEC SQL END DECLARE SECTION;

```

- Log on DB

```

strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

```

```

    }

    return SUCCESS;

fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

```

INTERVAL Types

INTERVAL type들은 두 개 시간 사이의 간격을 나타내는 datatype으로써 크게 year ~ month 계열과 day ~ second 계열로 구분되며 각 계열별로 다시 세분화된 type으로 구분된다.

INTERVAL type의 상세한 구분은 다음 표와 같다

표 27-6 INTERVAL type 구분

계열	세부 type	설명
YEAR TO MONTH	INTERVAL YEAR	연도 차이
	INTERVAL MONTH	개월 차이
	INTERVAL YEAR TO MONTH	연도부터 개월 차이
DAY TO SECOND	INTERVAL DAY	날짜 차이
	INTERVAL HOUR	시간 차이
	INTERVAL MINUTE	분 차이
	INTERVAL SECOND	초 차이
	INTERVAL DAY TO HOUR	날짜부터 시간 차이
	INTERVAL DAY TO MINUTE	날짜부터 분 차이
	INTERVAL DAY TO SECOND	날짜부터 초 차이
	INTERVAL HOUR TO MINUTE	시간부터 분 차이
	INTERVAL HOUR TO SECOND	시간부터 초 차이
	INTERVAL MINUTE TO SECOND	분부터 초 차이

INTERVAL type은 ODBC의 SQL_INTERVAL_STRUCT를 구조로 가지며 그 내용은 다음과 같다.

```

typedef enum
{
    SQL_IS_YEAR      = 1,
    SQL_IS_MONTH     = 2,
    SQL_IS_DAY       = 3,
    SQL_IS_HOUR      = 4,
    SQL_IS_MINUTE    = 5,
    SQL_IS_SECOND    = 6,

```

```

    SQL_IS_YEAR_TO_MONTH    = 7,
    SQL_IS_DAY_TO_HOUR      = 8,
    SQL_IS_DAY_TO_MINUTE    = 9,
    SQL_IS_DAY_TO_SECOND    = 10,
    SQL_IS_HOUR_TO_MINUTE   = 11,
    SQL_IS_HOUR_TO_SECOND   = 12,
    SQL_IS_MINUTE_TO_SECOND = 13
} SQLINTERVAL;
typedef struct tagSQL_YEAR_MONTH
{
    SQLUINTEGER year;
    SQLUINTEGER month;
} SQL_YEAR_MONTH_STRUCT;
typedef struct tagSQL_DAY_SECOND
{
    SQLUINTEGER day;
    SQLUINTEGER hour;
    SQLUINTEGER minute;
    SQLUINTEGER second;
    SQLUINTEGER fraction;
} SQL_DAY_SECOND_STRUCT;
typedef struct tagSQL_INTERVAL_STRUCT
{
    SQLINTERVAL interval_type;
    SQLSMALLINT interval_sign;
    union {
        SQL_YEAR_MONTH_STRUCT year_month;
        SQL_DAY_SECOND_STRUCT day_second;
    } intval;
} SQL_INTERVAL_STRUCT;

```

위의 모든 INTERVAL type은 동일한 구조체를 사용한다. 따라서 각 INTERVAL type에 대해 구조체 내에서 유효한 field를 별도로 구분하는데 구조체 내의 interval_type 필드를 가지고 INTERVAL type을 구분하고, 구조체 내의 intval 공용체로 실제 INTERVAL value를 표현한다. 각 type 별로 intval이 공용체에서 사용되는 field가 다른데 type별로 유효한 field는 다음 표와 같다.

표 27-7 INTERVAL type별 유효 field

Type	? .interval_type	? .intval 유효 field
INTERVAL YEAR	SQL_IS_YEAR	*.year_month.year
INTERVAL MONTH	SQL_IS_MONTH	*.year_month.month
INTERVAL YEAR TO MONTH	SQL_IS_YEAR_TO_MONTH	*.year_month.year

Type	?interval_type	?intval 유효 field
		*.year_month.month
INTERVAL DAY	SQL_IS_DAY	*.day_second.day
INTERVAL HOUR	SQL_IS_HOUR	*.day_second.hour
INTERVAL MINUTE	SQL_IS_MINUTE	*.day_second.minute
INTERVAL SECOND	SQL_IS_SECOND	*.day_second.second *.day_second.fraction
INTERVAL DAY TO HOUR	SQL_IS_DAY_TO_HOUR	*.day_second.day *.day_second.hour
INTERVAL DAY TO MINUTE	SQL_IS_DAY_TO_MINUTE	*.day_second.day *.day_second.hour *.day_second.minute
INTERVAL DAY TO SECOND	SQL_IS_DAY_TO_SECOND	*.day_second.day *.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction
INTERVAL HOUR TO MINUTE	SQL_IS_HOUR_TO_MINUTE	*.day_second.hour *.day_second.minute
INTERVAL HOUR TO SECOND	SQL_IS_HOUR_TO_SECOND	*.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction
INTERVAL MINUTE TO SECOND	SQL_IS_MINUTE_TO_SECOND	*.day_second.minute *.day_second.second *.day_second.fraction

Year, month, day, hour, minute, second는 각각 연, 월, 일, 시, 분, 초를 나타내며, fraction은 소수점 이하의 초를 표현한다. (Fraction field는 SECOND를 포함하는 INTERVAL type에서만 유효하다.)

Special Type

Special type은 자체적으로 data를 다루기보다는 응용 프로그램 작성에 부가적인 기능과 편의성을 제공하는 특별한 형태의 data type이다. Special type에는 다음과 같은 type들이 있다.

표 27-8 Special type

Type	설명
SQL_CONTEXT	Multi-connection 구조에서 run-time context를 관리한다.
Struct	Column의 집합을 구조체로 구성하여 row를 다룰 때 사용한다.
Typedef	기존에 정의된 type을 다른 이름으로 다시 정의한다.

SQL_CONTEXT

SQL_CONTEXT는 run-time context를 관리하기 위한 특별한 유형의 data type이다. Run-time context는 응용 프로그램이 실행되는 중에 run-time으로 connection과 connection에 관련된 개별 data를 관리하기 위해 사용된다.

SQL_CONTEXT 변수는 다음과 같이 선언한다.

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CONTEXT my_context;
EXEC SQL BEGIN DECLARE SECTION;
```

SQL_CONTEXT 변수를 선언한 뒤에는 다음과 같이 할당해 주어야 한다.

```
EXEC SQL CONTEXT ALLOCATE :my_context;
```

SQL_CONTEXT 변수를 사용하려면 USE 구문을 사용한다.

```
EXEC SQL CONTEXT USE :my_context;
```

USE 구문은 사용할 context를 지정하는 구문으로써, 응용 프로그램에서 선언한 context가 아닌 기본 context로 되돌리고자 할 때는 다음과 같이 사용한다.

```
EXEC SQL CONTEXT USE DEFAULT;
```

더 이상 사용하지 않는 SQL_CONTEXT 변수는 다음과 같이 해제할 수 있다.

```
EXEC SQL CONTEXT FREE :my_context;
```

다음은 SQL_CONTEXT를 사용하는 예이다.

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1
```

```

#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
void *clientThread(void *args);

typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;

#define THREAD_COUNT    2
char gJobName[THREAD_COUNT][20]= {
    "RND",
    "SUPPORT"
};

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sENAME[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    pthread_t     thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int          i;

```

```

printf("Connect GOLDILOCKS ...\n");
if(Connect(NULL, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
{
    goto fail_exit;
}

if(CreateEmpTempTable() != SUCCESS)
{
    goto fail_exit;
}

```

- Create client thread

```

for( i = 0; i < THREAD_COUNT; i ++ )
{
    param[i].mNo = i;
    param[i].mJobName = gJobName[i];
    if( pthread_create(&thread_id[i],
                      NULL,
                      clientThread,
                      &param[i]) != 0 )
    {
        printf( "Can't create thread %d!\n", i );
    }
    else
    {
        printf( "Create thread %d!\n", i );
    }
}

for( i = 0; i < THREAD_COUNT; i ++ )
{
    if( pthread_join(thread_id[i],
                     NULL) != 0 )
    {
        printf( "Error when waiting for thread %d to terminate!\n", i );
    }
    else
    {
        printf( "Stopped thread %d!\n", i );
    }
}

```

```
}
```

- Retrieve employee

```
EXEC SQL
```

```
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM    EMP_TEMP
    ORDER BY empno;
```

```
EXEC SQL OPEN EMP_CUR;
```

```
if(sqlca.sqlcode != 0)
```

```
{
```

```
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
```

```
    goto fail_exit;
```

```
}
```

```
printf(" EMPNO      ENAME                JOB      SALARY\n");
```

```
printf("===== \n");
```

```
while( 1 )
```

```
{
```

```
    EXEC SQL
```

```
        FETCH EMP_CUR
```

```
        INTO  :sEmpNo, :sENAME, :sJob, :sSalary;
```

```
if(sqlca.sqlcode == SQL_NO_DATA)
```

```
{
```

```
    break;
```

```
}
```

```
else if(sqlca.sqlcode != 0)
```

```
{
```

```
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
```

```
    goto fail_exit;
```

```
}
```

```
sRecordCount ++;
```

```
printf("%6d %20s %10s %8ld\n",
```

```
    sEmpNo, sENAME.arr, sJob, sSalary);
```

```
}
```

```
printf("===== \n");
```



```
printf("Record Count = %d\n", sRecordCount);
printf("===== \n");
```

```
EXEC SQL CLOSE EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}
```

```
if(DropEmpTempTable() != SUCCESS)
{
    goto fail_exit;
}
```

```
EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}
```

```
printf("SUCCESS\n");
printf("#####\n");
```

```
return 0;
```

```
fail_exit:
```

```
printf("FAILURE\n");
printf("#####\n\n");
EXEC SQL ROLLBACK WORK RELEASE;
```

```
return 0;
```

```
}
```

```
int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword)
```

```
{
```

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR sUid[80];
VARCHAR sPwd[20];
```

```

    VARCHAR sConnStr[1024];
EXEC SQL END DECLARE SECTION;
struct sqlca sqlca;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

if( aCtx != NULL )
{
    EXEC SQL CONTEXT USE :aCtx;
    EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
}
else
{
    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
}
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:

PRINT_SQL_ERROR("[ERROR] Connection Failure!");

return FAILURE;
}

int Disconnect(sql_context aCtx)
{
    struct sqlca sqlca;

```

- DB 연결 해제

```

if( aCtx != NULL )
{
    EXEC SQL CONTEXT USE :aCtx;
    EXEC SQL DISCONNECT;
}
else
{
    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL DISCONNECT;
}
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

return FAILURE;
}

```

- Create table

```

int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),

```

```

        SAL NUMBER(7,2),
        DEPTNO NUMBER(2) );
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
return SUCCESS;

fail_exit:

PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
EXEC SQL ROLLBACK WORK;

return FAILURE;
}

```

- Drop table

```

int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

```

```

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT    my_context;
    char           job_name[20 + 1];
    EXEC SQL END DECLARE SECTION;
    int            state = 0;
    thread_param *param = (thread_param *)args;

    EXEC SQL CONTEXT ALLOCATE :my_context;
    state = 1;

    EXEC SQL CONTEXT USE :my_context;
    if(Connect(my_context, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    state = 2;

    strcpy( job_name, param->mJobName );

    EXEC SQL
        INSERT INTO EMP_TEMP
        SELECT *
        FROM    EMP
        WHERE   JOB = :job_name;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {

```

```

        goto fail_exit;
    }
    state = 1;
    if(Disconnect(my_context) != SUCCESS)
    {
        goto fail_exit;
    }
    state = 0;
    EXEC SQL CONTEXT FREE :my_context;
    pthread_exit(0);
    return NULL;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    switch(state)
    {
        case 2:
            (void)Disconnect(my_context);
        case 1:
            EXEC SQL CONTEXT FREE :my_context;
            break;
        default:
            break;
    }

    pthread_exit(0);
    return NULL;
}

```

Host Structure

GOLDILOCKS의 embedded SQL에서는 C의 구조체를 host 변수로 사용할 수 있다. 일반적인 scalar 변수는 단일 column을 나타내는 반면에 구조체를 사용하면 여러 개의 column 집합을 간단하게 표현할 수 있다.

Host structure는 그 member 변수들을 차례대로 나열한 것과 결과가 같으므로 SELECT INTO, FETCH INTO의 INTO 절과 INSERT 구문의 VALUES 절에만 사용할 수 있고, WHERE 절이나 UPDATE SET 절 등에서는 사용할 수 없다.

Host structure를 사용하려면 declare section 내에 구조체를 정의하고 이 구조체 변수를 선언한 뒤에 host variable 처럼 사용하면 된다. 구조체 정의 방법은 C struct 정의 방법과 같고 typedef를 통해 type을 정의한 후에 사용할 수도 있다.

다음은 host structure를 사용하는 예이다.

```

/*
 * sample4.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int      mEmpNo;
    varchar  mENAME[20 + 1];
    char     mJob[20 + 1];
    long     mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsRecord    sRecord;
    EXEC SQL END DECLARE SECTION;
    int  sRecordCount = 0;

    printf("Connect GOLDILOCKS ...\n");

```

```

if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
{
    goto fail_exit;
}

```

- Retrieve employee

EXEC SQL

```

    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM EMP
    ORDER BY EMPNO;

EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO      ENAME                JOB      SALARY\n");
printf("===== =====\n");
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO :sRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    printf("%6d %20s %10s %8ld\n",
        sRecord.mEmpNo, sRecord.mENAME.arr, sRecord.mJob, sRecord.mSalary);
}

```



```

}

printf("===== \n");
printf("Record Count = %d\n", sRecordCount);
printf("===== \n");

EXEC SQL CLOSE EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("\n\nSUCCESS\n");
printf("#####\n");

return 0;

fail_exit:

printf("\n\nFAILURE\n");
printf("#####\n\n");

EXEC SQL ROLLBACK WORK RELEASE;

return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR sUId[80];
    VARCHAR sPwd[20];
    VARCHAR sConnStr[1024];

```

```
EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUId.arr, aUserID);
sUId.len = (short)strlen((char *)sUId.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB 연결

```
EXEC SQL CONNECT :sUId IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

Host 변수를 사용하기 위해 구조체를 선언할 때는 제약사항이 있다. 구조체 선언에서 중첩된 구조체는 사용할 수 없다. 다음과 같이 구조체 선언 내부에 또 다른 구조체가 존재할 경우에는 host 변수로 사용할 수 없다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int          mEmpNo;
        varchar      mENAME[20 + 1];
    } person;
    char            mJob[20 + 1];
    long            mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

Indicator Variable

Scalar Indicator

Host 변수들에는 그와 결합된 indicator 변수들이 함께 올 수 있는데 indicator 변수는 현재 host 변수의 값이 NULL 인지 여부를 판별한다. Indicator 변수는 host 변수와 마찬가지로 선언하여 사용할 수 있는데 C의 정수형 type (short, int, long, long long)으로만 선언할 수 있다. Indicator는 다음과 같이 사용한다.

```
:hostvar INDICATOR :hostind
:hostvar :hostind (INDICATOR keyword는 생략할 수 있다.)
```

Indicator 변수값은 다음과 같은 의미를 갖는다.

표 27-9 Input indicator 값

Value	의미
-1	NULL
>= 0	Host 변수값을 input한다.

표 27-10 Output indicator 값

Value	의미
-1	NULL
0	Value가 host 변수에 모두 저장되었다.
> 0	Host 변수의 buffer size가 부족하여 value를 host 변수에 모두 저장하지 못한 경우의 D B data 길이이다.

다음은 indicator를 사용하는 예이다.

```
/*
 * sample5.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
}
```

```

        printf(aMsg);
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
            sqlca.sqlcode,
            SQLSTATE,
            sqlca.sqlerrm.sqlerrmc );
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sENAME[20 + 1];
    char         sJob[20];
    long         sSalary;
    int          sDeptNo;
    int          sDeptNoInd;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    int          state = 0;

    printf("Connect GOLDSILLOCKS ...\n");
    if(Connect("DSN=GOLDSILLOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    state = 1;

```

- Update Dept NULL where deptno = 10

```

EXEC SQL
    UPDATE EMP_TEMP
    SET     DEPTNO = NULL

```

```

        WHERE DEPTNO = 10;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

```

- Retrieve employee

EXEC SQL

```

    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal, deptno
    FROM EMP_TEMP
    ORDER BY empno;

EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO          ENAME          JOB          SALARY  DEPTNO\n");
printf("===== \n");
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO :sEmpNo, :sENAME, :sJob, :sSalary, :sDeptNo :sDeptNoInd;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;
}

```

```

        if(sDeptNoInd == -1)
        {
            printf("%6d %20s %10s %8ld (null)\n",
                sEmpNo, sENAME.arr, sJob, sSalary);
        }
        else
        {
            printf("%6d %20s %10s %8ld %4d\n",
                sEmpNo, sENAME.arr, sJob, sSalary, sDeptNo);
        }
    }

    printf("=====\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("=====\n");
    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    state = 0;
    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("\n\nSUCCESS\n");
    printf("#####\n");

    return 0;

fail_exit:

```

```

printf("\n\nFAILURE\n");
printf("#####\n\n");

switch( state )
{
    case 1:
        (void)DropEmpTempTable();
        break;
    default:
        break;
}

EXEC SQL ROLLBACK WORK RELEASE;

return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR sUId[80];
    VARCHAR sPwd[20];
    VARCHAR sConnStr[1024];
    EXEC SQL END DECLARE SECTION;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUId.arr, aUserID);
sUId.len = (short)strlen((char *)sUId.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

EXEC SQL CONNECT :sUId IDENTIFIED BY :sPwd USING :sConnStr;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

```

```

        return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

```

- Create table

```

int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        INSERT INTO EMP_TEMP
        SELECT * FROM EMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
}

```



```

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

```

- Drop table

```

int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

```

Structure Indicator

Host 변수가 scalar 변수일 경우, indicator 변수를 선언하고 결합하여 사용한다. 하지만 host 변수가 structure일 경우에는 구조체의 구성 변수에 각각 indicator를 붙여쓸 수가 없다. 이렇게 host 변수가 구조체일 경우에는 indicator 역시 구조체로 만들어서 사용한다.

Indicator 구조체를 선언할 때는 다음과 같은 사항을 준수해야 한다.

- Indicator의 멤버 수와 결합할 host 변수 구조체의 멤버 수가 동일해야 한다.
- Indicator 구조체의 멤버는 모두 정수형 데이터 type을 가져야 한다.

예를 들어 다음과 같은 구조체를 선언했을 경우, 구조체 변수가 네 개이므로 indicator 구조체의 멤버도 네 개가 되어야 한다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mENAME[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

즉, 다음과 같이 선언되어야 한다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecordInd
{
    int          mEmpNoInd;
    int          mENAMEInd;
    int          mJobInd;
    int          mSalaryInd;
} rsRecordInd;
EXEC SQL END DECLARE SECTION;
```

Indicator 구조체는 그것과 결합하는 host 변수 구조체와 순서대로 1 : 1 대응한다. 즉, 위와 같이 구조체 선언이 이루어졌을 경우, 다음과 같은 결합 관계를 갖는다.

```
:rsRecordVar INDICATOR :rsRecordIndVar
```

Host variable	결합되는 indicator
rsRecordVar.mEmpNo	rsRecordIndVar.mEmpNoInd

Host variable	결합되는 indicator
rsRecordVar.mENAME	rsRecordIndVar.mENAMEInd
rsRecordVar.mJob	rsRecordIndVar.mJobInd
rsRecordVar.mSalary	rsRecordIndVar.mSalaryInd

Embedded SQL

Host Variable

Host variable은 응용 프로그램과 GOLDILOCKS 사이의 data 입출력 매개체로 사용된다. 응용 프로그램에서 GOLDILOCKS로 data를 전달할 때는 input host variable이라고 하고 GOLDILOCKS에서 응용 프로그램으로 data를 받아올 때는 output host variable이라고 한다. 그러나 이 두 가지 사이에 선언상의 차이점은 없으며 사용되는 SQL 문장에서 그 역할이 결정된다.

SELECT 또는 FETCH 문의 INTO 절에 위치한 host variable은 GOLDILOCKS에서 data를 받아오는 output host variable이며 그 외에는 모두 input host variable이 된다. Input host variable 값은 해당 SQL 문을 수행하기 전에 설정되어야 한다.

Host Indicator

Host variable은 C 언어의 변수를 그대로 사용하기 때문에, NULL을 표시할 수 있는 별도의 방법이 없다. 이 경우에 indicator 변수를 사용할 수 있으며 한 개의 indicator 변수는 한 개의 host variable과 결합한다.

Indicator 변수는 다음과 같은 의미를 갖는다.

표 27-11 Input indicator 값

Value	의미
-1	NULL
>= 0	Host 변수값을 input 한다.

표 27-12 Output indicator 값

Value	의미
-1	NULL
0	Value가 host 변수에 모두 저장되었다.
> 0	Host 변수의 buffer size가 부족하여 value를 host 변수에 모두 저장하지 못한 경우의 DB data 길이이다.

Insert NULL

NULL 값은 다음과 같이 column에 insert 된다.

```
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES ( :empno, NULL );
```

그러나 위와 같이 hard coding하여 응용 프로그램을 만들면 유연성이 매우 떨어진다. 다음은 indicator 변수를 사용하는 예이다.

```
deptno_ind = -1;
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES ( :empno, :deptno :deptno_ind );
```

Indicator 변수인 deptno_ind에 -1 값이 주어진다면 host variable인 deptno 값에 관계없이 NULL 값으로 인식된다.

Fetch NULL

GOLDILOCKS에서 data를 받을 때 indicator 변수를 사용하여 NULL 여부를 판단할 수 있다. 다음 예제를 참조한다.

```
EXEC SQL DECLARE CUR_1 CURSOR FOR
    SELECT  EMPNO, DEPTNO
    FROM    EMP
    WHERE   EMPNO = :emp_number;
EXEC SQL OPEN CUR_1;

while( 1 )
{
    EXEC SQL FETCH CUR_1 INTO :empno, :deptno :deptno_ind;
    if( sqlca.sqlcode == SQL_NO_DATA )
    {
        break;
    }

    if( deptno_ind == -1 )
    {
        printf( "empno : %d, deptno : (null)\n", empno );
    }
    else
    {
        printf( "empno : %d, deptno : %d\n", empno, deptno );
    }
}
```

```
EXEC SQL CLOSE CUR_1;
```

FETCH 후에 deptno_ind가 -1이면 deptno가 NULL인 것으로 판단한다.

Basic SQL statement

Embedded SQL에서는 GOLDDILOCKS에서 제공하는 모든 SQL 문을 사용할 수 있는데 SQL 문에 대한 자세한 내용은 파트 III. **SQL Manual**을 참조한다. Embedded SQL에서 SQL 문을 사용할 때는 EXEC SQL 키워드 뒤에 SQL 문을 기술하는데 본 장에서는 Data Definition Language (DDL)이나 Data Manipulation Language (DML)에 대한 내용을 기술하고 query와 같이 반복적으로 data를 가져오는 구문에 대해서는 다음 장에서 설명한다.

SQL 문을 실행한 이후에는 SQLCA를 검사하여 실행한 SQL 문의 성공 여부를 판단할 수 있으며, 자세한 내용은 **Handling Run-time Error** 부분을 참조한다.

DDL Statement

DDL statement는 GOLDDILOCKS의 table, view, index 등과 같은 object들을 생성하거나, 제거, 변경 등의 작업을 수행하는 SQL 문이다. Embedded SQL 응용 프로그램에서 DDL문을 수행할 때는 EXEC SQL keyword 뒤에 해당 SQL 문을 기술한다.

```
EXEC SQL DROP TABLE IF EXISTS EMP;
EXEC SQL
    CREATE TABLE EMP (
        EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        SAL NUMBER(7,2),
        DEPTNO NUMBER(2) );
```

DDL 문에서는 host variable을 사용할 수 없다. 따라서 다음은 잘못된 사용 방법이다.

```
strcpy( table_name, "T1" );
EXEC SQL CREATE TABLE :table_name ( C1 INTEGER );
```

만약, 응용 프로그램을 작성하는 도중에 DDL 구문을 확정하지 못하여 위와 같이 가변적으로 DDL 구문을 사용하고자 할 때는 다음과 같이 dynamic SQL 문을 사용할 수 있다.

```
strcpy( table_name, "T1" );
sprintf( sql_stmt, "CREATE TABLE %s ( C1 INTEGER )", table_name );
EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
```

이에 대한 자세한 내용은 **Embedded Dynamic SQL**을 참조한다.

Select Into Statement

GOLDILOCKS에서 data를 가져 오기 위해서 query를 사용한다. 일반적인 경우에는 해당 결과의 개수를 알지 못하는 경우가 많기 때문에, query를 사용하기 위해서는 cursor라는 객체를 선언하여 open, fetch, close라는 복잡한 과정을 거치게 된다. 이러한 과정의 실행 방식은 다음 **Cursor** 부분에서 기술할 것이다.

그런데 특별한 경우에는 이미 결과의 개수를 알고 있을 수도 있다. 예를 들어 primary key를 알고 있고 해당 primary key와 같은 key를 가진 record를 검색할 때는 결과가 없거나 최대 한 개의 결과만 가진다는 것을 예측할 수 있다. 이렇게 결과가 한 개 이하일 때 사용할 수 있는 구문이 select into 구문이며 다음과 같이 간단하게 실행할 수 있다.

```
EXEC SQL
    SELECT ename, job, sal
    INTO :emp_name, :job_title, :salary
    FROM emp
    WHERE empno = :emp_number;
```

노트

Host array를 사용하면 select into 구문을 사용하여 여러 개의 결과를 가져올 수 있다. Host array 사용법은 **Host Arrays** 를 참조한다.

Insert Statement

Insert 구문은 table에 row를 삽입하는 용도로 사용된다. Host variable을 사용하여 column값을 결정할 수 있고 indicator를 사용하여 NULL 값을 삽입할 수도 있다.

다음은 insert 구문의 예이다.

```
EXEC SQL
    INSERT INTO EMP ( empno, ename, job, sal )
    VALUES ( :emp_number, :emp_name, :job_name :job_ind, :saraly );
EXEC SQL
    INSERT INTO DEPT ( deptno, dname, loc )
    VALUES ( 1, :dept_name, NULL );
```

노트

Host structure를 사용하면 host variable을 개별적으로 사용하지 않고 structure 단위로 삽입할 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mENAME[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;

rsRecord  sInsertRec;
EXEC SQL END DECLARE SECTION;

sInsertRec.mEmpNo = 3000;
strcpy( sInsertRec.mENAME.arr, "John" );
sInsertRec.mENAME.len = strlen( sInsertRec.mENAME.arr );
strcpy( sInsertRec.mJob , "RND" );
sInsertRec.mSalary = 3500;

EXEC SQL INSERT INTO EMP ( empno, ename, job, sal ) VALUES ( :sInsertRec );
```

노트

Host array를 사용하면 한 번에 여러 개의 row를 삽입할 수 있다. 자세한 내용은 **Host Array**를 참조한다.

Update Statement

Update 구문은 table에서 특정 row의 column 값들을 갱신하는 용도로 사용된다. Host variable을 사용하여 column 값을 결정할 수 있고, indicator를 사용하여 NULL 값을 삽입할 수도 있다.

다음은 update 구문의 예이다.

```
EXEC SQL
    UPDATE emp
    SET sal = :salary, deptno = :dept_number :deptno_ind
    WHERE empno = :emp_number;
```

노트

Host array를 사용하면 한 번에 여러 개의 row를 갱신할 수 있다. 자세한 내용은 **Host Array**를 참조한다.

Delete Statement

Delete 구문은 특정 row를 table에서 삭제한다.

다음은 delete 구문의 예이다.

```
EXEC SQL
    DELETE FROM emp
    WHERE empno = :emp_number;
```

노트

Host array를 사용하면 한 번에 여러 개의 row를 삭제할 수 있다. 자세한 내용은 **Host Array**를 참조한다.

PSM Statement

PSM 구문은 서버 내부에 procedure나 function을 만들어서 사용한다.

PSM에 대한 자세한 내용은 파트 IV. **PSM Manual**을 참조한다.

기본적으로 EXEC SQL EXECUTE로 시작해서 END-EXEC;로 끝난다. 단, procedure나 function을 생성할 때는 EXEC SQL EXECUTE 대신 EXEC SQL를 사용한다.

다음은 procedure를 생성하고 호출하는 구문의 예이다.

```
EXEC SQL
    CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
    IS
        V1 INTEGER;
    BEGIN
        SELECT COUNT(*)
        INTO V1
        FROM T1
        WHERE T1.I1 >= A1 AND T1.I1 <= A2;
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END;
END-EXEC;
EXEC SQL CALL PROC1( 2, 4 );
```

다음은 function을 생성하고 호출하는 구문의 예이다.


```

EXEC SQL BEGIN DECLARE SECTION;
    int          sV1 = 0;
EXEC SQL END DECLARE SECTION;
EXEC SQL
    CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
    RETURN INTEGER
    IS
        V1 INTEGER;
    BEGIN
        SELECT COUNT(*)
            INTO V1
            FROM T1
            WHERE T1.I1 >= A1 AND T1.I1 <= A2;
        RETURN V1;
    END;
END-EXEC;
EXEC SQL CALL FUNC1( 2, 4 ) INTO :sV1;

```

다음은 anonymous block 구문의 예이다.

```

EXEC SQL EXECUTE
    DECLARE
        V1 INTEGER := 0;
        FUNCTION FUNC1( A1 INTEGER )
        RETURN INTEGER
        IS
            BEGIN
                RETURN A1 * 10;
            END;
    BEGIN
        V1 := FUNC1( 10 );
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END;
END-EXEC;

```

```

EXEC SQL EXECUTE
    DECLARE
        PROCEDURE PROC1( A1 INTEGER )
        IS
            BEGIN
                DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
            END;

```

```

        END;
    BEGIN
        PROC1( 100 );
    END;
END-EXEC;

```

Cursor

응용 프로그램은 원하는 data를 GOLDILOCKS로 얻어오기 위해 질의를 수행한다. 일반적으로 질의를 수행할 때는 그 질의 결과의 양이 어느정도인지 정확히 알 수 없는 경우가 많기 때문에 cursor를 사용한다. Cursor는 질의 결과 집합에서 현재 row의 위치를 명시하는 식별자이며 다음과 같은 연산을 통해 cursor를 조작할 수 있다.

Declare Cursor

Cursor를 선언한다. Cursor를 선언할 때는 cursor 이름과 그에 부합하는 질의를 기술해야 한다. 여기서 선언된 cursor 이름은 이후에 다른 cursor 조작 명령에 사용된다. 다음은 cursor 선언문의 예이다.

```

EXEC SQL
    DECLARE RECORD_CUR1 CURSOR FOR
    SELECT  empno, ename, dept
    FROM    SEMP
    ORDER BY empno;

```

Cursor 이름은 precompiler에 의해 인식되고 사용되는 식별자로서 실제 C 프로그램의 변수와는 무관하다. 그러므로 cursor 선언문의 위치는 프로그램 소스 코드 상에서 해당 cursor에 접근하는 구문 앞에 위치해야만 하며 다른 파일에 정의된 cursor 선언문은 사용할 수 없다. 즉, cursor를 open/ fetch/ close 하는 구문은 모두 하나의 소스 파일 내에 위치해야 하며 cursor 선언문은 모든 구문들보다 앞서 위치해야 한다. 또한, 하나의 소스 파일 내에서 서로 다른 질의에 대해 동일한 cursor 이름을 중복하여 사용할 수 없다.

```
EXEC SQL OPEN RECORD_CUR1;
```

- 잘못된 cursor 선언문 위치

```

EXEC SQL
    DECLARE RECORD_CUR1 CURSOR FOR
    SELECT  empno, ename, dept
    FROM    SEMP
    ORDER BY empno;

```

위와 같이 cursor 선언문이 뒤에 위치하면 precompile 과정에서 cursor가 선언되지 않았다는 에러가 발생한다.

```

FileName: overview.gc
Pre-compile overview.gc -> overview.c
ERR-42000(41000): syntax error
Error at line 310, in file overview.gc
ERR-42000(41003): Cursor "RECORD_CUR1" not declared
OPEN RECORD_CUR1;
.....^
Error at line 1:
ERR-42000(41006): Fatal error while doing embedded SQL precompiling

```

Open Cursor

Cursor를 open한다.

```

EXEC SQL OPEN <cursor_name>;
Example)
EXEC SQL OPEN EMP_CURSOR;

```

커서를 open하면 선언된 커서의 질의를 수행한 결과 집합을 가져올 준비가 된다. 여기서는 실제로 결과를 가져오는 것은 아니기 때문에 실제 결과를 얻어오려면 **Fetch Cursor**를 수행해야 한다.

질의를 수행할 때 사용한 host variable은 현재 커서가 close 될 때까지 결과 집합에 영향을 미치지 않는다.

```

EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
    SELECT    empno, ename, dept
    FROM      EMP
    WHERE     empno < :sNo
    ORDER BY  empno;

sNo = 100;
EXEC SQL OPEN EMP_CURSOR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

while( 1 )
{
    sNo = 10;
    EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
}

```

```

    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}

```

위의 예에서 커서가 sNo = 100으로 open 되었으면 중간에 sNo를 다른 값으로 바꾸더라도 이 커서가 close 될 때까지는 결과 집합이 변하지 않는다.

Fetch Cursor

커서의 위치에 있는 row를 가져온다.

```

EXEC SQL FETCH <cursor_name> INTO <host_variable_list>;
Example)
EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;

```

FETCH 연산을 수행하려면 해당 커서가 미리 선언되어야 하고 OPEN 되어 있어야 한다. 최초로 FETCH를 수행하면 커서가 결과 집합의 첫 번째 row로 이동하여 current row로 지정한 다음 current row를 INTO 절의 host variable에 넣어 반환한다. 이후에 FETCH가 반복적으로 수행되면 커서가 다음 row로 current row를 갱신한 후 current row를 INTO 절의 host variable로 반환하는 작업을 반복한다. FETCH를 수행했는데 더 이상 결과가 존재하지 않을 경우에는 sqlca.sqlcode에 SQL_NO_DATA 코드를 반환하므로 응용 프로그램에서는 이 코드값을 보고 검색이 종료되었는지 여부를 판단할 수 있다.

Close Cursor

커서를 close 한다.

```

EXEC SQL CLOSE <cursor_name>;
Example)
EXEC SQL CLOSE EMP_CURSOR;

```

커서를 close 하기 위해서는 해당 커서가 미리 open 되어 있어야 하고 커서를 close 한 뒤에는 더 이상 이 커서에서 FETCH를 수행할 수 없다. 만약 커서를 close 한 이후에 다시 사용하기 위해서 open할 경우, 이는 새로 open 된 커서이므로 이전에 close 했던 커서와 결과 집합이 달라질 수 있다.

다음은 DDL, DML, cursor를 사용하는 예이다.

```

/*
 * overview.gc
 *
 * Connect / Disconnect

```

```

* DDL(Create/Drop table)
* Basic DML(Insert, Delete, Update)
* Standing Cursor
*/
EXEC SQL INCLUDE SQLCA;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmpRecord
{
    int      mEmpNo;
    char     mENAME[20];
    char     mDept[10];
} rsEmpRecord;

rsEmpRecord gRecord[10] = {
    { 1, "Park", "RND" },
    { 2, "Kim", "CEO" },
    { 3, "Choi", "SALES" },
    { 4, "Lee", "CTO" },
    { 5, "Lyu", "RND" },
    { 6, "Ohn", "SUPPORT" },
    { 7, "Cheon", "RND" },
    { 8, "Sohn", "SALES" },
    { 9, "Smith", "WAIT" },
    { 10, "mycomman", "WAIT" }
}

```

```

};
EXEC SQL END DECLARE SECTION;

int CreateEmpTable();
int DropEmpTable();
int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int Disconnect();
int PrintRecord();

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    char          sDept[10 + 1];
    EXEC SQL END DECLARE SECTION;
    int  sState = 0;

    printf("Connect GOLDDILOCKS ...\n");
    if(Connect("DSN=GOLDDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 1;

    printf("Create SEMP table ...\n");
    if(CreateEmpTable() != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 2;

    printf("Insert record ...\n");
    EXEC SQL
        INSERT INTO SEMP(empno, ename, dept)
        VALUES(:gRecord);
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
    printf("===== \n");
}

```

```

printf("%d Record Inserted\n", sqlca.sqlerrd[2]);
printf("===== \n");

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("Current record\n");
PrintRecord();

sEmpNo = 9;
printf("Delete record WHERE empno == %d\n", sEmpNo);
EXEC SQL
    DELETE FROM SEMP
    WHERE empno = :sEmpNo;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("After Delete record\n");
PrintRecord();

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

strcpy( sDept, "RND" );
printf("Update record WHERE dept == 'WAIT'\n");
EXEC SQL
    UPDATE SEMP
    SET dept = :sDept
    WHERE dept = 'WAIT';
if(sqlca.sqlcode != 0)

```

```

    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("After Update record\n");
    PrintRecord();

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sState = 1;
    printf("Drop SEMP table ...\n");
    if(DropEmpTable() != SUCCESS)
    {
        goto fail_exit;
    }

    sState = 0;
    printf("Disconnect GOLDBLOCKS ...\n");
    if(Disconnect() != SUCCESS)
    {
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("#####\n");

    return 0;

fail_exit:

    printf("FAILURE\n");
    printf("#####\n\n");

    EXEC SQL ROLLBACK WORK;
    switch(sState)

```



```

{
    case 2:
        printf("Drop SEMP table ...\n");
        (void)DropEmpTable();
    case 1:
        printf("Disconnect GOLDILOCKS ...\n");
        (void)Disconnect();
        break;
    default:
        break;
}

return 0;
}

```

- Create table

```

int CreateEmpTable()
{
    EXEC SQL DROP TABLE IF EXISTS SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL CREATE TABLE SEMP ( empno      INTEGER,
                                ename       VARCHAR(20),
                                dept        VARCHAR(10),
                                PRIMARY KEY (empno) );

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;
}

```

```

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

```

- Drop table

```

int DropEmpTable()
{
    EXEC SQL DROP TABLE SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR sUId[80];

```

```

VARCHAR sPwd[20];
VARCHAR sConnStr[1024];
EXEC SQL END DECLARE SECTION;

```

- Log on GOLDDILOCKS

```

strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

int Disconnect()
{
    EXEC SQL DISCONNECT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

```

```

    PRINT_SQL_ERROR("[ERROR] [ERROR] Disconnect Failure!");

    return FAILURE;
}

int PrintRecord()
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsEmpRecord sResultRecord;
    EXEC SQL END DECLARE SECTION;
    int    sRecordCount = 0;

    EXEC SQL
        DECLARE RECORD_CUR1 CURSOR FOR
        SELECT    empno, ename, dept
        FROM      SEMP
        ORDER BY empno;

    EXEC SQL OPEN RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    printf(" EMPNO      ENAME                      DEPT\n");
    printf("===== \n");
    while( 1 )
    {
        EXEC SQL FETCH RECORD_CUR1 INTO :sResultRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        sRecordCount ++;

        printf("%6d %20s %10s\n",

```

```

        sResultRecord.mEmpNo,
        sResultRecord.mENAME,
        sResultRecord.mDept);
    }

    printf("===== \n");
    printf("Record Count = %d\n", sRecordCount);
    printf("===== \n");

    EXEC SQL CLOSE RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    return FAILURE;
}

```

Cursor Property

앞 장에서는 가장 기본적인 커서에 대해서만 설명하였다. 그러나 커서는 다양한 속성들을 가질 수 있으며 커서가 갖는 속성에 따라 다양한 기능을 수행할 수 있다. GOLDILOCKS의 embedded SQL은 ISO/ IEC-9075-2 SQL foundation에서 제안하는 cursor 속성과 ODBC에서 제안하는 cursor 속성을 제공한다.

커서 정의 구문과 속성에 대한 자세한 내용은 **DECLARE cursor_name**을 참조한다.

Scrollable Cursor

SCROLL은 ISO type의 cursor 속성으로써 커서의 scroll 가능 여부를 결정한다. Scroll 가능한 커서는 FETCH 구문에 서 위치 옵션을 받을 수 있고 이 옵션에 따른 위치의 row를 FETCH한다. Scroll 불가능한 커서는 결과 집합에 대해 순차적으로만 FETCH 할 수 있다.

Scroll cursor를 선언할 때는 SCROLL 옵션을 사용하는데 다음은 그 예이다.

```
EXEC SQL DECLARE cur_scroll SCROLL CURSOR FOR SELECT c1, c2 FROM t1;
```

NO SCROLL 옵션을 사용할 때는 scroll 불가능한 커서가 선언되고 scroll 옵션을 주지 않았을 경우에는 기본적으로 NO SCROLL이 된다.

Scrollable cursor의 경우에는 FETCH 할 때 위치 정보를 줄 수 있다. 이 위치 정보를 fetch orientation이라고 하며 다음 표와 같다.

표 27-13 Fetch orientation

Fetch orientation	설명
NEXT	현재 위치의 다음 row를 fetch 한다.
PRIOR	현재 위치의 이전 row를 fetch 한다.
FIRST	결과 집합의 첫 번째 row를 fetch 한다.
LAST	결과 집합의 마지막 row를 fetch 한다.
CURRENT	현재 위치의 row를 fetch 한다.
ABSOLUTE <position>	- 결과 집합에서 position의 위치에 해당하는 row를 fetch 한다. - Position 값이 음수일 경우, AFTER THE LAST ROW로부터 이전 위치에 해당하는 row를 fetch한다.
RELATIVE <position>	현재 위치에서 position 만큼 떨어진 위치에 해당하는 row를 fetch 한다.

Fetch orientation이 ABSOLUTE, RELATIVE인 경우에는 <position> 값을 추가적으로 요구하는데, 이 <position> 값으로 정수형 상수나 정수형 type의 host variable이 사용될 수 있다. 다음은 fetch orientation을 사용하는 예이다.

```
EXEC SQL FETCH NEXT;
EXEC SQL FETCH PRIOR;
EXEC SQL FETCH FIRST;
EXEC SQL FETCH LAST;
EXEC SQL FETCH CURRENT;
EXEC SQL FETCH ABSOLUTE 100;
EXEC SQL FETCH RELATIVE :position;
```

Sensitive Cursor

Sensitivity는 ISO type의 cursor 속성으로써 커서를 운용하는 도중에 결과 집합이 변경될 경우 그 변경 내용을 볼 수 있는지 여부를 결정한다. Sensitivity에는 다음과 같은 세 가지 옵션이 있다.

표 27-14 Sensitivity

Option	설명
SENSITIVE	다른 트랜잭션에서 변경하거나 삭제한 내용을 볼 수 있다.
INSENSITIVE	커서를 open한 이후에 변경하거나 삭제한 내용을 볼 수 없다.
ASECSITIVE	<query>의 내용에 따라 SENSITIVE/ INSENSITIVE가 결정된다.

Sensitive option을 명시하지 않았을 경우 기본값은 INSENSITIVE이며 sensitive 옵션을 사용한 커서는 다음과 같은 방법으로 선언한다.

```
EXEC SQL DECLARE <cur_name> SENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> INSENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> ASENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
```

Holdable Cursor

Holdability는 ISO type의 cursor 속성으로써 현재 커서를 open한 트랜잭션이 commit된 이후에도 커서가 계속 유지되고 있는지 여부를 결정한다. Holdability에는 다음과 같은 두 가지 옵션이 있다.

표 27-15 Holdability

Option	설명
WITH HOLD	- 트랜잭션의 COMMIT 이후에도 커서가 유지된다. - FOR UPDATE 구문과 함께 사용할 수 없다 - INSERT INTO ... RETURNING 구문과 함께 사용할 수 없다. - UPDATE ... RETURNING 구문과 함께 사용할 수 없다. - DELETE FROM ... RETURNING 구문과 함께 사용할 수 없다.
WITHOUT HOLD	트랜잭션을 COMMIT/ ROLLBACK 하면 커서를 닫는다.
Rollback과 Cursor	- 트랜잭션을 rollback하면 트랜잭션에 포함된 커서를 닫는다. - Savepoint까지 rollback하면 savepoint 이후에 생성된 커서를 닫는다.

Holdable option을 명시하지 않았을 경우 기본값은 <cursor updatability>에 따라 결정된다.

- FOR READ ONLY 또는 <cursor updatability>가 명시되지 않은 경우, WITH HOLD이다.
- FOR UPDATE 구문과 함께 사용될 경우, WITHOUT HOLD이다.

Holdable option을 사용한 커서는 다음과 같은 방법으로 선언한다.

```
EXEC SQL DECLARE <cur_name> CURSOR WITH HOLD FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> CURSOR WITHOUT HOLD FOR SELECT c1, c2 FROM t1 FOR UPDATE;
```

Static Cursor

Static 커서는 ODBC type의 cursor 속성으로써 ISO type의 INSENSITIVE SCROLL 커서와 동일하다.

```
EXEC SQL DECLARE cur_static STATIC CURSOR FOR SELECT c1, c2 FROM t1;
```

Static cursor의 fetch 방법은 **Scrollable Cursor**를 참조한다

Keyset Driven Cursor

Keyset driven 커서는 ODBC type의 cursor 속성으로써 ISO type의 ASENSITIVE SCROLL 커서와 동일하다.

```
EXEC SQL DECLARE cur_static KEYSET CURSOR FOR SELECT c1, c2 FROM t1;
```

Keyset driven 커서 역시 scroll 기능을 가지고 있으므로 fetch 방법은 **Scrollable Cursor**를 참조한다.

Positioned DML

CURRENT OF <cursor_name> 구문을 사용하여 마지막으로 fetch된 row에 대해 delete나 update 구문을 사용한 연산을 수행할 수 있는데, 이를 positioned DML이라고 한다. Positioned DML을 수행하려면 cursor가 open되어 있어야 하고, fetch가 최소 한 번 이상 수행되어 커서가 row의 위치를 가리키고 있어야만 한다.

다음은 positioned DML을 수행하는 예이다.

```
EXEC SQL DECLARE emp_cursor CURSOR FOR
    SELECT ename, sal FROM emp WHERE job = 'SALES'
    FOR UPDATE;

...
EXEC SQL OPEN emp_cursor;
EXEC SQL WHENEVER NOT FOUND GOTO ...
while( 1 )
{
    EXEC SQL FETCH emp_cursor INTO :emp_name, :salary;
    ...
    EXEC SQL UPDATE emp SET sal = :new_salary
        WHERE CURRENT OF emp_cursor;
}
```

Options

본 장에서는 embedded SQL 소스 코드를 precompile 할 때 적용할 수 있는 option에 대해 설명한다.

Precompiled Header File

Embedded SQL 프로그램을 작성할 때, 여러 개의 소스 코드에서 공통적으로 참조하는 내용들은 별도의 header file로 작성하여 소스 코드가 해당 header File을 포함하도록 하는 것이 효율적이다. 이미 C 언어에서는 #include 문장을 사용하여 이런 기능을 지원하고 있다. 그런데 C 언어의 #include를 통해 header file을 포함할 때는 대상 파일을 pre compile을 하지 않고 C 언어 컴파일러에 그 해석을 맡기기 때문에 그 파일 내의 declare section 등은 precompiler에 의해 변환되지 않는다.

이렇게 header file을 만들 때, 이 header file을 precompiler로 변환하여 소스 코드에 삽입하려면 EXEC SQL INCLUDE 구문을 사용해야 하며, 그 구문은 다음과 같다.


```
EXEC SQL INCLUDE <filename>;
```

이 구문은 주어진 파일을 precompile하여 소스 코드에 삽입해 준다.

Header File 경로 지정

기본적으로 header file을 찾을 때는 현재 소스 코드가 위치한 디렉토리를 우선 검색한다. 그러나 대체로 header file 들은 별도로 모아 놓는 경우가 많고 모아놓지 않더라도 여러 가지 이유로 인해 다른 경로에 존재하는 경우가 많다.

이 때, header file을 검색할 디렉토리를 별도 옵션으로 줄 수 있으며, 그 구문은 다음과 같은 형식이다.

```
EXEC SQL OPTION( INCLUDE = <directory path> );
```

이 옵션 여러 개를 나열하여 부여할 수 있고 EXEC SQL INCLUDE를 통해 header file을 찾을 때, 이 옵션이 주어진 순서대로 디렉토리를 검색한다.

노트

이 옵션은 precompiler의 command-line option을 통해서도 부여할 수 있다. 자세한 내용은 **Precompiler Options**의 `--include-path`, -I를 참조한다.

Host Array

지금까지는 host variable로 단일 값을 갖는 scalar 변수에 대해 설명하였다. 본 장에서는 array를 host variable로 사용하는 방법에 대해 설명한다.

Host array를 사용하면 프로그램 소스 코드가 간결해지고 performance가 향상되지만 사용에 제한적인 면이 있으므로 주의해서 사용해야 한다.

Host Array 선언

Host array 선언은 scalar variable을 선언하는 경우와 다르지 않다. 변수 선언 자체를 배열로 선언하기만 하면 된다. 다음은 host array를 배열 크기 10으로 선언하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    int    empno[10];
    char   ename[10][20];
    double salary[10];
EXEC SQL END DECLARE SECTION;
```

Host array 선언에는 다음과 같은 제약 사항이 있다.

- 2차원 이상의 배열은 허용되지 않는다.
- 예외적으로 문자열 계열 (char, varchar)과 binary 계열 (binary, varbinary)은 array size를 나타내는 배열 첨자와 데이터 size를 나타내는 배열 첨자로 구성되어 2차원 배열이 쓰일 수 있다.
- Pointer의 배열은 허용되지 않는다.

Host Array 사용

Host Array 접근

Host array를 SQL 구문에서 사용하는 방법은 scalar host variable을 사용하는 것과 동일하다. 다음은 간단한 host array의 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number 값을 INSERT

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
VALUES (:emp_number, :emp_name, :dept_number);
```

위의 예는 다음 코드와 동일한 작업을 수행한다.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```

for( i = 0; i < 20; i ++ )
{
    EXEC SQL INSERT INTO emp (empno, ename, deptno)
        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);
}

```

여러 개의 host variable array가 사용되었을 때는, 각 host array 중에 array size가 가장 작은 것에 대해서만 연산이 이루어진다. 위의 예에서는 모든 host variable의 크기가 20 이므로 20 개의 row가 삽입되지만 다음과 같이 dept_number의 배열 크기를 10으로 지정한 경우, 결과적으로 삽입되는 row의 수는 10 개가 된다.

```

EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[10];
EXEC SQL END DECLARE SECTION;

```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```

EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);

```

Host Indicator Array 사용

Host 변수가 배열일 경우, 여기에 결합되는 indicator 변수도 배열이어야 하며 indicator 변수의 배열 크기와 host 변수의 배열 크기는 같아야 한다. 위의 예에 indicator 변수가 추가될 경우, 다음과 같은 형태가 될 것이다.

```

EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
int    emp_number_ind[20];
char   emp_name[20][10];
int    emp_name_ind[20];
int    dept_number[20];
int    dept_number_ind[20];
EXEC SQL END DECLARE SECTION;

```

- emp_number, emp_name, dept_number 값을 설정
- emp_number_ind, emp_name_ind, dept_number_ind의 각 indicator 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
VALUES (:emp_number :emp_number_ind,
        :emp_name :emp_name_ind,
        :dept_number :dept_number_ind);
```

제약 사항

- Host 변수가 여러 개 사용될 때는 scalar 변수와 array 변수를 혼합하여 사용할 수 없다.
- Select 구문의 WHERE 절에는 host array를 사용할 수 없다.
- Update, delete 구문의 CURRENT OF 절에는 host array를 사용할 수 없다.

INTO 절에서의 Array

GOLDILOCKS에서 row를 가져 오기 위해서는 SELECT INTO 구문이나 cursor를 사용할 수 있다. Embedded SQL에서는 이 두 가지 방법에 공통적으로 INTO 절을 사용하는데 이 INTO 절에 host array를 사용하면 여러 row를 가지고 올 수 있다.

SELECT INTO에서의 Array

가져와야 할 row의 개수를 정확히 알고 있을 경우에는 **Select Into Statement**를 사용하여 간단하게 구현할 수 있다. **Select Into Statement**에서는 결과가 없거나 한 개일 경우에 대해서만 언급하였지만 여기에 host array를 사용하면 다수의 row를 가져올 수 있다.

사용 방법은 scalar 변수를 쓰는 것과 동일하지만 host variable을 array로 선언하는 것만으로도 array를 사용한 select into 구문을 작성할 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
char   emp_name[50][20];
int    emp_number[50];
float  salary[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ENAME, EMPNO, SAL
        INTO :emp_name, :emp_number, :salary
        FROM EMP
        WHERE SAL > 1000;
```

위의 예를 보면 단지 host 변수들을 배열로 선언하는 것만으로 SELECT INTO 구문으로 50 개의 row를 가져오는 문장이 작성된다. 이 문장은 독립적인 실행 단위를 갖기 때문에 질의 조건에 부합하는 처음의 50개 row만 가져온다.

주의

만약 질의 조건에 부합하는 row의 개수가 50 개 이상이라고 해도 SELECT INTO 구문을 사용할 때는 뒷 부분의 51 번째 row부터 가져올 수 있는 방법은 없다. 연속적으로 row를 fetch하려면 cursor를 사용해야만 한다.

Cursor 사용할 때의 Array

현재 질의에 대한 결과 집합 개수를 알 수 없는 경우에는 커서를 사용해야만 한다. 커서를 사용하는 방법은 **Cursor** 부분을 참조한다. 커서를 선언한 이후에 FETCH 구문에서 INTO 절의 host 변수를 array로 사용하는 것만으로도 다수의 row를 한 번에 가져올 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[50];
    char   emp_name[50][20];
    char   dept_name[50][20];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
    SELECT    empno, ename, dept
    FROM      EMP
    WHERE     empno < :sNo
    ORDER BY  empno;

sNo = 100;
EXEC SQL OPEN EMP_CURSOR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}
```

```

    }
    ...
}

```

FETCH 구문에서 host array를 사용하는 것만으로도 한 번에 50 개의 row를 fetch하는 기능을 구현할 수 있다.

sqlca.sqlerrd[2]

Array를 사용하면 선언된 배열 크기만큼의 row를 fetch하는데 더 이상 row가 없어서 배열 크기만큼 가져오지 못하는 경우가 있다. 예를 들어, 배열 크기는 50으로 선언되었지만 결과 집합의 row가 30 개일 경우 30 개의 row밖에 가지고 오지 못한다. 이런 경우에 대비하여 GOLDILOCKS의 embedded SQL은 현재 수행한 구문이 몇 개의 row를 처리하였는지에 대한 정보를 sqlca.sqlerrd[2]에 제공한다.

sqlca.sqlerrd[2]에는 INSERT, UPDATE, DELETE, SELECT INTO, FETCH 구문에서 처리된 row의 개수를 반환한다.

```

EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[50];
    char   emp_name[50][20];
    char   dept_name[50][20];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
    SELECT    empno, ename, dept
    FROM      EMP
    WHERE     empno < :sNo
    ORDER BY  empno;

sNo = 100;
EXEC SQL OPEN EMP_CURSOR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {

```

```

        goto fail_exit;
    }

    for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
    {
        printf( "%d %s %s\n", emp_number[i], emp_name[i], dept_name[i] );
    }
    ...
}

```

sqlca에 대한 자세한 내용은 **Handling Run-time Error**를 참조한다.

Insert 구문에서의 Array

INSERT 구문에서 host 변수를 array로 선언하면 array insert가 구현된다.

```

EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char    emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;

```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```

EXEC SQL INSERT INTO emp (empno, ename, deptno)
VALUES (:emp_number, :emp_name, :dept_number);

```

위의 예는 다음 코드와 동일한 작업을 수행한다.

```

EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char    emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;

```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```
for( i = 0; i < 20; i ++ )
{
    EXEC SQL INSERT INTO emp (empno, ename, deptno)
        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);
}
```

노트

- Array insert를 사용할 때는 host 변수가 모두 array 변수이거나 모두 scalar 변수이어야 한다.
- sqlca.sqlerrd[2]를 확인하여 몇 건의 row가 insert 되었는지 확인할 수 있다.

Atomic Insert

Atomic insert는 array insert의 특별한 형태이지만 다음과 같은 두 가지 특징이 있다.

- Insert 하는 모든 row 중에 한 개라도 실패하면 전체 insert 작업이 실패한다.
- 응용 프로그램에서 GOLDDLOCKS로 한 번만 명령어가 전달되기 때문에 performance가 뛰어나다.

Atomic insert를 위해 다음과 같은 ATOMIC 키워드를 사용한다.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```


Update 구문에서의 Array

다음은 update 구문에서 array를 사용하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;

...

EXEC SQL UPDATE emp SET comm = :commission
      WHERE job = :job_title;
```

위의 예는 다음과 동일한 기능을 수행한다.

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;

...
for( i = 0; i < 10; i ++ )
{
    EXEC SQL UPDATE emp SET comm = :commission[i]
          WHERE job = :job_title[i];
}
```

노트

- Array update를 사용할 때는 host 변수가 모두 array 변수이거나 모두 scalar 변수이어야 한다.
- `sqlca.sqlerrd[2]`를 확인하여 몇 건의 row가 update 되었는지 확인할 수 있다.
- UPDATE 구문의 CURRENT OF 절에는 array를 사용할 수 없다.

Delete 구문에서의 Array

Array는 delete 구문에서 다음과 같이 사용할 수 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL DELETE FROM emp
    WHERE job = :job_title;
```

위의 예는 다음과 동일한 기능을 수행한다.

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;
...
for( i = 0; i < 10; i ++ )
{
    EXEC SQL DELETE FROM emp
        WHERE job = :job_title[i];
}
```

노트

- Array delete를 사용할 때는 host 변수가 모두 array 변수이거나 모두 scalar 변수이어야 한다.
- `sqlca.sqlerrd[2]`를 확인하여 몇 건의 row가 delete 되었는지 확인할 수 있다.
- DELETE 구문의 CURRENT OF 절에는 array를 사용할 수 없다.

FOR 절 사용

SQL 문을 실행하면서 처리하고자 하는 배열 크기를 명시하고자 할 때 FOR 구문을 사용한다. FOR 구문은 다음과 같은 문장에서 사용될 수 있다.

- SELECT INTO
- FETCH
- INSERT
- UPDATE
- DELETE

For 절은 다음과 같이 사용된다.

```
EXEC SQL FOR :host_variable <sql_stmt>
EXEC SQL FOR <integer_constant> <sql_stmt>
```

For 문을 사용하는 예는 다음과 같다.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
int     record_cnt;
EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```
record_cnt = 10;
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)
VALUES (:emp_number, :emp_name, :dept_number);
```

위의 예에서 host array의 크기는 20으로 주어졌지만 FOR 구문을 사용하여 record_cnt 만큼만 수행하도록 지정하였기 때문에 실제로는 10 건만 insert 된다.

노트

FOR 절은 UPDATE/ DELETE CURRENT OF 구문에서는 사용할 수 없다.

구조체 Array

일반적인 scalar 변수들을 배열로 사용하면 한 번에 다수의 row를 처리할 수 있어 편리하지만 한 개의 변수당 한 개의 column만 표시할 수 있다는 한계가 있다. 한 개의 host 변수에서 다수의 column에 접근하려면 **Host Structure**를 사용하면 되는데 이렇게 host 변수를 구조체로 선언하여 그 구조체를 array로 사용하면 다수의 column을 갖는 다수의 row를 한 번에 처리할 수 있다.

구조체 배열은 다음과 같은 경우에 사용될 수 있다.

- Output host variable array: SELECT INTO, FETCH INTO 구문
- Input host variable array: INSERT 구문의 VALUES 항목

제약 사항

다음과 같은 경우에는 구조체 배열을 사용할 수 없다.

- WHERE 절이나 FROM 절
- UPDATE 구문의 SET 절

구조체 배열 선언

구조체는 일반적인 C 언어의 구조체 선언 방식으로 선언한다. 직접 구조체 변수를 선언할 수도 있고 typedef를 통해서 type을 선언한 후에 host 변수를 선언할 수도 있다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
rsRecord      sRecord[10];

struct {
    int          mEmpNo;
    varchar      mName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} sResultRecord[10];
EXEC SQL END DECLARE SECTION;
```

Host 변수를 사용하기 위한 구조체를 선언할 때 중첩된 구조체는 사용할 수 없다. 다음과 같이 구조체 선언 내부에 또 다른 구조체가 존재할 경우에는 host 변수로 사용할 수 없다.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int          mEmpNo;
        varchar      mName[20 + 1];
    } person;
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
```

```
EXEC SQL END DECLARE SECTION;
```

구조체 배열 Indicator

Structure Indicator에 기술된 것과 같이 host 변수가 구조체이면 indicator 변수도 구조체이어야 한다. 따라서 host 변수가 구조체 배열일 경우 indicator 변수도 대응하는 구조체 배열이어야 한다.

Indicator 구조체 배열 선언은 다음 사항을 준수한다.

- Indicator의 멤버 수는 결합할 host 변수 구조체의 멤버 수와 같아야 한다.
- Indicator 구조체의 멤버는 모두 정수형 데이터 type을 가져야 한다.
- Indicator 구조체 배열의 크기는 host 변수 구조체 배열의 크기와 같아야 한다. 만약 indicator 구조체 배열의 크기가 작으면 SQL 문을 실행할 때 FOR 절을 이용하여 배열 실행 횟수를 제한해야 한다.

Structure와 Scalar 변수 혼용

Host structure가 GOLDILOCKS로 전달될 때 그 structure member들은 순차적으로 나열된다. 다음은 host structure와 scalar 변수를 섞어서 사용하는 예이다.

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int          mEmpNo;
        varchar      mName[20 + 1];
    } rsEmp;
    rsEmp    sEmp[5];
    char     sJob[5][20 + 1];
    long     sSalary[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP
    ORDER BY EMPNO;
EXEC SQL OPEN EMP_CUR;

EXEC SQL
    FETCH EMP_CUR
    INTO   :sEmp, :sJob, :sSalary;
```

FETCH 구문에는 rsEmp라는 structure와 sJob, sSalary라는 scalar 변수가 함께 사용되었다. 이 문장은 내부적으로 sEmp.mEmpNo, sEmp.mENAME, sJob, sSalary 라는 네 개의 변수로 해석된다. 이런 식으로 host structure와 ho

st scalar variable을 섞어서 사용할 수 있다.

다음은 host structure array, structure array indicator, structure와 scalar 변수를 섞어서 사용하는 예이다.

```

/*
 * fetch_struct_array.gc
 * : structure array fetch
 * : structure indicators
 * : mix structure, scalar variable
 * : sqlca.sqlerrd[2]
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1

#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmp
{
    int          mEmpNo;
    varchar      mENAME[20 + 1];
} rsEmp;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);

int main(int argc, char **argv)
{

```

```

EXEC SQL BEGIN DECLARE SECTION;
rsEmp      sEmp[5];
char       sJob[5][20 + 1];
long       sSalary[5];
EXEC SQL END DECLARE SECTION;
int  sRecordCount = 0;
int  i;

printf("Connect GOLDDILOCKS ...\n");
if(Connect("DSN=GOLDDILOCKS", "test", "test") != SUCCESS)
{
    goto fail_exit;
}

```

- Retrieve employee

```

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP
    ORDER BY EMPNO;

EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO      ENAME                JOB      SALARY\n");
printf("===== \n");

while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmp, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
}

```

```

        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        sRecordCount += sqlca.sqlerrd[2];

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            printf("%6d %20s %10s %8ld\n",
                sEmp[i].mEmpNo, sEmp[i].mENAME.arr, sJob[i], sSalary[i]);
        }
    }
    printf("===== \n");
    printf("Record Count = %d\n", sRecordCount);
    printf("===== \n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("\n\nSUCCESS\n");
    printf("#####\n");

    return 0;

fail_exit:

    printf("\n\nFAILURE\n");
    printf("#####\n\n");

```



```

EXEC SQL ROLLBACK WORK RELEASE;

return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR sUid[80];
    VARCHAR sPwd[20];
    VARCHAR sConnStr[1024];
    EXEC SQL END DECLARE SECTION;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

```

Handling Run-time Error

개요

응용 프로그램이 실행되는 중에 기대했던 수행 결과를 내지 못하는 경우에 대비하여 embedded SQL 응용 프로그램을 작성할 때 이런 예외 상황에 대처할 필요가 있다. 본 절에서는 SQL 실행 후에 반환되는 실행 결과를 감지하는 방법에 대해서 설명한다.

Run-time Error 감지

SQLCA

Embedded SQL에서 발생하는 모든 종류의 에러는 SQL Communication Area (SQLCA)라고 하는 영역에 보고된다. 따라서 응용 프로그램에서 SQLCA의 내용을 확인하면 현재 연산의 성공 여부와 함께 에러가 발생했을 경우 그 에러의 종류까지 파악할 수 있다.

SQLCA는 error, warning, SQL 문의 수행 상태 등을 저장하는 자료 구조이다. SQLCA는 이 자료 구조가 결합된 마지막 SQL의 실행 결과를 가지고 있을 뿐, SQL 문 실행 이력에 대한 내용은 저장되지 않는다. Embedded SQL 문이 실행되면 기존의 SQLCA 내용은 사라지기 때문에 예외를 처리하기 위해서는 SQL을 실행한 후에 SQLCA를 바로 확인해서 필요한 조치를 취해야 한다.

SQLCA 사용

SQLCA를 사용하기 위해서는 다음과 같은 구문을 사용한다.

```
EXEC SQL INCLUDE SQLCA;
```

위 문장은 precompile 과정에서 다음 문장으로 치환된다.

```
#include "sqlca.h"
```

이 문장은 첫 번째 embedded SQL 문장이 사용되기 전에 미리 쓰여야 하며 통상적으로 소스 코드의 상단에 배치하도록 권장한다.

GOLDILOCKS의 embedded SQL 응용 프로그램은 기본적으로 sqlca를 전역으로 가지고 있다. 따라서 single thread program에서는 별다른 선언 없이 sqlca를 사용할 수 있다. 그러나 multi thread program에서는 동시에 sqlca에 접근할 경우 동시성 문제가 발생하기 때문에 별도의 선언 방법을 가져야 한다. 이에 대한 내용은 **Multi-threaded Application**에서 설명한다.

SQLCA 구조

다음은 sqlca의 구조체이다.

```
struct sqlca
{
    char    sqlcaid[8]; ❶ 문자열 SQLCA로 초기화한다.
    int     sqlabc;      ❷ sqlca 구조체의 크기이다.
```

- 가장 최근에 실행한 문장에서 발생한 error code가 0이면 성공, 양수이면 warning, 음수이면 error이다.

```
int     sqlcode;
```

- sqlcode에 해당하는 에러 메시지를 저장한다.
- .sqlerrml은 sqlerrmc의 길이이다.
- .sqlerrmc는 에러 메시지를 string 형태로 저장한다.

```
struct
{
    unsigned short sqlerrml;
    char           sqlerrmc[SQLERRMC_LEN];
} sqlerrm;

char    sqlerrp[8]; ❸ unused
int     sqlerrd[6];
```

- 0: empty
- 1: empty
- 2: INSERT, UPDATE, DELETE 후에 처리된 row의 개수이다.
- 3: empty
- 4: empty
- 5: empty

```
char    sqlwarn[8];
```

- 0: 임의의 warning이 한 개라도 발생할 경우 'W'이다.
- 1: SELECT, FETCH에서 결과 string이 truncate 된 경우 'W'이다.
- 2: unused
- 3: unused
- 4: unused
- 5: unused
- 6: unused
- 7: unused

```

char    sqltext[8];    ❷ unused
char    sqlstate[8];   ❸ SQLSTATE
unsigned short *rowstatus; ❹ fetched row status array
};

```

다음 절에서 각 구성 요소들을 설명한다.

SQLCODE

SQLCODE는 다음과 같이 정의되어 있다.

```
#define SQLCODE          (sqlca.sqlcode)
```

SQLCODE는 embedded SQL 문을 실행한 후에 그 실행 결과 코드를 반환한다. SQLCODE는 ISO/ IEC-9075에서 초기에 제안되었으나 SQL-92에서 deprecate 되었다. 그러나 많은 응용 프로그램에서 사용되고 있기 때문에 하위 호환성을 위해서 제공되고 있다. 실행 결과 코드는 다음과 같다.

SQLCODE	결과
0	성공
> 0	Warning 발생
SQL_NO_DATA	결과 없음
< 0	Error 발생

sqlcode는 다음과 같은 방법으로 확인할 수 있다. sqlca.sqlcode나 SQLCODE를 사용할 수 있다.

```

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP;

EXEC SQL OPEN EMP_CUR;
if(SQLCODE != 0)
{
    goto fail_exit;
}
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sENAME, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
}

```

```

    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    ...
}

```

SQLSTATE

SQLSTATE는 다음과 같이 정의되어 있다.

```
#define SQLSTATE      (sqlca.sqlstate)
```

ISO/ IEC-9075에서 SQLCODE가 deprecate된 후 이를 대신해 제안되었다. SQLSTATE는 다섯 개의 문자 (숫자, 영문 대문자 알파벳)로 구성되어 있는데 앞의 두 자리는 class이고, 뒤의 세 자리는 subclass 이다. SQLSTATE의 실행 결과는 다음과 같다.

표 27-16 SQLSTATE 실행 결과

SQLSTATE	결과
00000	성공
01xxx	Warning 발생
02000	결과 없음
그 밖의 모든 state	Error 발생

위 예제의 SQLCODE 대신 SQLSTATE를 확인하여 코드를 수정하면 다음과 같다. sqlca.sqlstate나 SQLSTATE를 사용할 수 있다.

```

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP;

EXEC SQL OPEN EMP_CUR;
if(strcmp(SQLSTATE, "00000") != 0)
{
    goto fail_exit;
}
while( 1 )
{
    EXEC SQL

```

```

        FETCH EMP_CUR
        INTO :sEmpNo, :sENAME, :sJob, :sSalary;
        if(strcmp(sqlca.sqlstate, "02000") == 0)
        {
            break;
        }
        else if(strcmp(sqlca.sqlstate, "00000") != 0)
        {
            goto fail_exit;
        }

        ...
    }

```

처리된 Row의 개수

Update, delete 구문이나 array를 사용한 insert, select into, fetch 구문을 수행했을 때 몇 개의 row가 처리되었는지를 알려준다. 해당 정보는 sqlca.sqlerrd[2]에 저장되며 응용 프로그램에서는 해당 필드의 값을 참조하여 처리된 row의 개수를 파악할 수 있다. 다음은 sqlca.sqlerrd[2]를 사용하는 예이다.

```

EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int            mEmpNo;
        varchar        mName[20 + 1];
    } rsEmp;
    rsEmp    sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP;

EXEC SQL OPEN EMP_CUR;
if(SQLCODE != 0)
{
    goto fail_exit;
}
while( 1 )
{

```

```

EXEC SQL
    FETCH EMP_CUR
    INTO :sEmpNo, :sENAME, :sJob, :sSalary;
if(sqlca.sqlcode == SQL_NO_DATA)
{
    break;
}
else if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sRecordCount += sqlca.sqlerrd[2];
...
}

```

처리된 Row의 상태

sqlca.rowstatus는 현재 처리된 row의 상태를 나타낸다. Scroll sensitive cursor를 사용하거나 where CURRENT OF를 사용하여 row를 갱신/ 삭제하였을 경우, row의 상태가 갱신될 수 있으며 array를 사용한 SQL 구문을 사용했을 경우, array 크기만큼의 row status를 갖는다.

표 27-17 Row 상태

Row 개수	Row 상태
1개	*sqlca.rowstatus
Array size n	sqlca.rowstats[0] sqlca.rowstats[1] sqlca.rowstats[2] ... sqlca.rowstats[n-1]

표 27-18 Row 상태 값

Row 상태	설명
SQL_ROW_SUCCESS	Row의 상태가 정상이다.
SQL_ROW_DELETED	Row가 삭제되었다.
SQL_ROW_UPDATED	Row가 갱신되었다.
SQL_ROW_NOROW	Row가 존재하지 않는다.
SQL_ROW_ADDED	Row가 추가되었다.
SQL_ROW_ERROR	Row의 상태가 비정상이다.

다음은 row status를 참조하는 예이다.

```

EXEC SQL BEGIN DECLARE SECTION;

typedef struct rsEmp
{
    int          mEmpNo;
    varchar      mENAME[20 + 1];
} rsEmp;

rsEmp    sEmp[5];
char      sJob[5][20 + 1];
long      sSalary[5];

EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP;

EXEC SQL OPEN EMP_CUR;
if(SQLCODE != 0)
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sENAME, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sRecordCount += sqlca.sqlerrd[2];

    for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
    {
        if( sqlca.rowstatus[i] == SQL_ROW_SUCCESS )
        {
            printf("%6d %20s %10s %8ld\n",

```



```

        sEmp[i].mEmpNo, sEmp[i].mENAME.arr, sJob[i], sSalary[i]);
    }
}

...
}

```

Error Message Text

Embedded SQL 문을 실행한 결과 error나 warning이 발생하면 해당 메시지를 text 형태로 전송할 수 있다. 에러 메시지는 `sqlca.sqlerrm`에 저장되는데 `sqlca.sqlerrm.sqlerrml`은 text의 길이이고 실제 메시지는 `sqlca.sqlerrm.sqlerrmc`에 저장되어 있다. 에러 메시지 text는 응용 프로그램에서 이상 현상이 발생하였을 경우 이를 출력해서 사용자에게 정보를 전달할 때 유용하다. 다음은 error message text를 사용하는 예이다.

```

EXEC SQL INSERT INTO EMP VALUES ( :sEmp );
if( SQLCODE != 0 )
{
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
        SQLCODE,
        SQLSTATE,
        sqlca.sqlerrm.sqlerrmc );
}

```

Warning Flags

`sqlca.sqlwarn`은 embedded SQL을 수행한 후에 warning이 발생했을 때 warning을 mark해 주는 flag로 사용된다. 여덟 개의 char array로 구성되어 있으며 warning이 발생하였을 경우 'W' 마크를 남긴다.

표 27-19 Warning flag

Warning flag	설명
<code>sqlca.sqlwarn[0]</code>	임의의 warning이 한 개라도 발생할 경우 'W'가 된다.
<code>sqlca.sqlwarn[1]</code>	Select into, fetch에서 결과 string이 truncate된 경우 'W'가 된다.
<code>sqlca.sqlwarn[2]</code>	reserved
<code>sqlca.sqlwarn[3]</code>	reserved
<code>sqlca.sqlwarn[4]</code>	reserved
<code>sqlca.sqlwarn[5]</code>	reserved
<code>sqlca.sqlwarn[6]</code>	reserved
<code>sqlca.sqlwarn[7]</code>	reserved

Handling Implicit Error

Embedded SQL 문을 실행한 후에는 반드시 SQLCA를 확인하여 수행 결과에 대한 조치를 취해야만 한다. 그러나 Embedded SQL 문을 실행한 뒤에 동일한 예외 처리를 하는 경우에는 WHENEVER 지시자를 사용하여 이를 자동화할 수 있다.

WHENEVER 구문 사용

WHENEVER 구문의 문법은 다음과 같다.

```
EXEC SQL WHENEVER conditions actions;
```

WHENEVER Condition

WHENEVER 구문의 condition은 다음과 같다.

```
<conditions> ::=
    SQLERROR
  | SQLWARNING
  | NOT FOUND
  | SQLSTATE <sqlstate class value>[<sqlstate subclass value>]
  ;

<sqlstate_char> ::= [0-9A-Z];
<sqlstate class value> ::= <sqlstate_char><sqlstate_char>;
<sqlstate subclass value> ::= <sqlstate_char><sqlstate_char><sqlstate_char>;
```

표 27-20 Conditions of WHENEVER statement

Condition	설명
SQLERROR	Embedded SQL을 수행하는 중에 error가 발생했다.
SQLWARNING	Embedded SQL을 수행하는 중에 warning이 발생했다.
NOT FOUND	결과 row가 없다.
SQLSTATE <sqlstate>	Embedded SQL을 수행하는 중에 SQLSTATE <sqlstate>가 발생하였다.

WHENEVER Action

Action은 앞서 기술한 condition을 만족할 때 실제로 수행되는 동작을 기술한 것으로써 그 문법은 다음과 같다.

```
<actions> ::=
    CONTINUE
  | GOTO <label>
  | STOP
  | DO <c statements>
```

;

표 27-21 Actions of WHENEVER statement

Action	설명
CONTINUE	아무런 action을 취하지 않는다. 즉, 주어진 condition을 무시한다.
GOTO <label>	<label>로 프로그램 흐름을 분기한다.
STOP	프로그램 실행을 종료한다.
DO <c statements>	<c statements>를 실행한다.

WHENEVER 구문 적용 범위

WHENEVER 구문은 conditions에 대한 actions를 기술하는데 하나의 condition에 대해 하나의 action만 기술할 수 있다. 따라서 WHENEVER 구문을 사용하면 현재 정의된 condition 이후에 다른 condition 정의가 나올 때까지 모든 embedded SQL 구문에 동일한 action이 사용된다.

WHENEVER 구문은 각 condition 별로 따로 관리되기 때문에 특정 시점에 적용할 수 있는 WHENEVER 구문의 최대 개수는 네 개이다. 같은 condition에 대해서 새로운 WHENEVER 구문이 적용될 경우, 기존 action은 취소되고 이후부터 해당 action이 적용된다. 다음은 그 예이다.

```
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary ); ❶
...
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL UPDATE emp SET sal = sal * 1.1 WHERE sal < :sal_bound; ❷
...
EXEC SQL WHENEVER SQLERROR GOTO exit_label;
EXEC SQL WHENEVER NOT FOUND DO break;
EXEC SQL DECLARE EMP_CURSOR FOR
    SELECT empno, ename, sal
    FROM emp;

EXEC SQL OPEN EMP_CURSOR; ❸

EXEC SQL WHENEVER SQLERROR GOTO close_label;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
        INTO :emp_number, :emp_name, :salary; ❹

    printf( "emp number : %d, emp name : %s, salary : %lf\n",
        emp_number, emp_name, salary );
}
```

```

close_label:
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CLOSE EMP_CURSOR; ❸
...
exit_label:
...

```

Line 1에서 SQLERROR의 경우에 STOP하도록 지시하였으므로 문장 ❶을 실행하다가 에러가 발생하면 응용 프로그램이 종료된다. Line 4에서 SQLERROR일 경우 CONTINUE하도록 지시하므로 SQLERROR의 action은 아무것도 하지 않고 진행하는 것으로 변경이 되고, 그에 따라 문장 ❷을 실행하다가 에러가 발생하더라도 아무 일도 없이 다음으로 넘어가게 된다. Line 7에서 SQLERROR의 action을 exit_label로 분기하는 것으로 설정하므로 문장 ❸을 실행하다가 에러가 발생하면 exit_label로 분기한다. Line 8에서 NOT FOUND에 대해 break를 수행하도록 지정하였고 line 15에서 SQLERROR의 action은 close_label로 다시 지정하였기 때문에 문장 ❹에는 두 가지의 condition이 적용되어 FETCH를 실행하는 도중에 error가 발생할 경우에는 close_label로 분기하고 FETCH의 결과가 없으면 break 문이 실행된다. Line 26에서 SQLERROR의 action을 sql_error()를 실행하는 것으로 지정하였으므로 문장 ❺를 실행하는 도중에 error가 발생하면 sql_error() 함수가 호출된다.

WHENEVER 구문 사용 시 주의 사항

WHENEVER 구문을 사용할 때는 그 동작 원리를 정확히 파악하여 주의 깊게 사용해야 한다. WHENEVER 구문을 사용할 때 주의할 점은 다음과 같다.

- 소스 코드 상에서 WHENEVER 구문의 위치: WHENEVER 구문은 precompiler에게 exception handling 코드를 자동으로 삽입하도록 알려주는 지시자일 뿐, 논리적으로 실행되는 코드가 아니다. 따라서 WHENEVER 구문을 사용할 때는 소스 코드 상에서 적용할 embedded SQL 문의 앞 부분에 써 주어야 하고 이후에 등장하는 다른 embedded SQL에 영향을 주지 않으려면 반드시 CONTINUE action을 사용하여 초기화해 주어야 한다.
- Break, continue 키워드 사용: Action 구문에서 DO break, DO continue를 사용할 때는 반드시 loop의 영역을 확인해야 한다. 다음 예제를 참조한다.

```

EXEC SQL WHENEVER SQLERROR GOTO fail_exit;
EXEC SQL WHENEVER NOT FOUND DO break; ❶
EXEC SQL DECLARE EMP_CURSOR FOR
        SELECT empno, ename, sal
        FROM   emp;

EXEC SQL OPEN EMP_CURSOR;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
            INTO   :emp_number, :emp_name, :salary; ❷
    printf( "emp number : %d, emp name : %s, salary : %lf\n",
            emp_number, emp_name, salary );
}

```

```
EXEC SQL CLOSE EMP_CURSOR;
```

```
EXEC SQL
    SELECT MAX(sal)
    INTO    :max_salary
    FROM    emp; ❸
...

```

❶에서 NOT FOUND에 대한 action으로 DO break;를 정의하였으므로 이후 SQL 문을 실행할 때 NOT FOUND가 발생하면 break;를 실행하려 할 것이다. ❷에서 FETCH를 수행하다가 NOT FOUND가 발생하면 break가 수행되어 while loop에서 빠져 나온다. 그러나 ❸의 경우에는 NOT FOUND가 발생할 때 break를 수행하려고 하여도 loop가 아니기 때문에 break를 실행할 수 없다. 이 경우에는 소스 프로그램을 build하여 응용 프로그램을 만드는 단계에서 이미 compile 에러가 발생한다.

- 무한 loop 회피

Action으로 분기를 사용할 때 잘못하면 무한 loop에 빠질 수 있다. 다음 예제를 참조한다.

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL ROLLBACK WORK RELEASE;
```

SQL을 실행하는 중에 error가 발생하면 sql_error 레이블로 분기하게 하였다. 이 때 embedded SQL 문을 실행하다가 error가 발생하면 sql_error로 분기한다. 그런데 이 에러처리 과정에서 또 다른 embedded SQL을 실행하도록 하고 여기서 반복적으로 error가 발생할 경우, 응용 프로그램이 무한 loop에 빠지게 된다. 이 경우에는 다음과 같이 error 처리 부분을 안전하게 초기화해 주는 것이 좋다.

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
```

- 분기 label의 scope

Action에서 분기할 때는 반드시 접근 가능한 곳에 해당 label이 존재해야 한다. 다음 예제를 참조한다.

```

func1()
{

    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;

    ...
labelA:
    ...
}
func2()
{

    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);

    ...

}

```

func1()에서 SQLERROR에 대하여 labelA 분기를 지정하였다. func1()에는 labelA가 있기 때문에 문제가 없지만 func2()에는 labelA가 존재하지 않기 때문에 compile 오류가 발생한다. 이 경우에는 func2()에도 동일한 labelA를 만들어 주거나 다음과 같이 action을 초기화 해주어야 한다.

```

func1()
{

    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;

    ...
labelA:
    ...
}
func2()
{

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);

    ...

}

```

27.3 Advanced Topic

Embedded Dynamic SQL

개요

대부분의 embedded SQL 응용 프로그램은 GOLDILOCKS에 대한 구체적인 동작을 가지고 있다. 특정 테이블에 row를 삽입, 갱신, 삭제, 조회하는 것을 목표로 하고 있으며 그에 따른 동작 방법을 SQL이라는 database 언어를 사용하여 기술한다. 이를 위해 embedded SQL 소스 코드에 SQL을 직접 작성하면 precompiler는 이 SQL을 해석하여 주어진 SQL 내용과 input/output host variable을 모두 알고 있는 상태에서 GOLDILOCKS에 동작 가능한 API 호출로 변환한다.

그러나 어떤 응용 프로그램의 경우에는 프로그램을 작성할 때 SQL을 미리 알 수 없는 경우도 있다. 예를 들어 GUI to 이과 같은 응용 프로그램에서 사용자가 연산 종류, 테이블 이름, 조건 등을 선택하여 조회하기 위해 static SQL을 사용할 경우, 사용자가 선택할 수 있는 모든 조합에 대해 미리 SQL을 생성해야 하는데 이는 현실적으로 불가능할 뿐만 아니라 만약에 가능하다고 하더라도 매우 비효율적인 작업이 된다. 이 때 사용자가 선택하는 옵션을 가지고 SQL을 생성하여 실행할 수 있다면 프로그램 작성이 매우 유연해진다. 이렇게 소스 코드 상에서 SQL이 정의되지 않고 실행 시점에 가변적으로 변경되는 SQL을 dynamic SQL 이라고 하며 GOLDILOCKS에서는 embedded dynamic SQL 기능을 제공하고 있다.

Embedded dynamic SQL 응용 프로그램은 static SQL에 비해 좀 더 유연한 사용방법을 제공한다는 장점이 있지만 소스 코드 작성이 좀 더 까다로워진다는 특징이 있고 동일한 query를 하더라도 static SQL에 비해 dynamic SQL의 수행 성능이 더 떨어질 수 있다는 단점이 있다.

따라서 응용 프로그램을 작성할 때는 이러한 특징을 고려하여 static SQL과 dynamic SQL 중에 주의깊게 선택해야 한다.

본 장에서는 embedded dynamic SQL 프로그램을 작성하는 방법에 대해 설명한다.

Dynamic SQL의 종류

Dynamic SQL은 그 사용 방법에 따라 다음과 같이 분류된다.

표 27-22 Dynamic SQL의 종류

Method	설명	지원 여부
Method 1	Non-query, host variable이 없다.	O
Method 2	Non-query, 그 개수와 type을 알고 있는 host variable이 있다.	O
Method 3	Query, 그 개수와 type을 알고 있는 host variable이 있다.	O
Method 4	Query, host variable의 존재 여부, 개수와 type등을 알지 못한다.	X

현재 GOLDILOCKS에서는 method 3까지만 지원하고 있다.

Method 1

가장 단순한 형태의 dynamic SQL로써, non-query이면서 host variable이 없는 경우에 사용할 수 있다. 일반적으로 DDL이나 host variable 없는 DML에 사용한다.

- EXECUTE IMMEDIATE

Method 1은 그 특성상 non-query이면서 host variable도 없기 때문에 SQL 문을 즉시 실행할 수 있다. SQL의 즉시 실행 문법은 다음과 같다.

```
EXEC SQL EXECUTE IMMEDIATE { :host_variable | <string_literal> };
<string_literal> ::=
    ' <sql_statement> '
  | " <sql_statement> "
  | <sql_statement>
  ;
```

Method 1은 다음과 같이 사용할 수 있다.

```
sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"
        "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"
        "ENAME VARCHAR2(10),\n"
        "JOB VARCHAR2(9),\n"
        "SAL NUMBER(7,2),\n"
        "DEPTNO NUMBER(2) )\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
        "SELECT *\n"
        "FROM EMP\n"
        "WHERE JOB = 'RND'\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
```


Method 2

Method 2는 non-query이면서 input host variable이 존재하는 경우에 사용한다. 이 때 input host variable의 개수와 data type을 모두 알고 있어야 하며 준비 (prepare) 단계와 실행 (execute) 단계를 통해 구현된다.

- Prepare

Prepare 단계에서는 SQL statement를 분석하고 이 statement에 이름을 부여한다. Prepare의 문법은 다음과 같다.

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable | <string_literal> };
<string_literal> ::=
    ' <sql_statement> '
  | " <sql_statement> "
  | <sql_statement>
  ;
```

<statement_name>은 precompiler에게 알려주는 식별자로서 host variable이 아니기 때문에 별도의 type을 갖거나 변수를 선언할 필요가 없다.

- Execute

Execute 단계에서는 분석된 statement를 실행한다. Execute 구문의 문법은 다음과 같다.

```
EXEC SQL EXECUTE <statement_name> [ USING <host_variable_list> ];
<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];
<host_variable_entry> ::= :host_variable [ [ INDICATOR ] :host_indicator ] ;
```

Input host variable이 있을 경우에는 USING 절을 사용한다. Host variable이 없을 경우, USING 절은 생략한다. USING 절에 등장하는 host variable의 순서대로 prepare된 SQL 문의 host variable에 binding된다.

Method 2에서 동일한 SQL 문을 반복 실행할 때, prepare는 한 번만 수행한 뒤에 execute를 반복 실행할 수 있다. 이 prepare된 statement는 현재 소스 코드 내에서 같은 statement name으로 다른 SQL 문을 prepare 하거나 disconnect할 때까지 유효하다.

Method 3

Method 3는 query를 지원할 수 있도록 method 2에서 확장된 형태이다. 기본적으로 query prepare 과정을 통해 statement를 분석하고 query하기 위해 cursor를 다룰 수 있어야 하기 때문에 cursor에 대해 declare, open, fetch, close하는 구문이 추가된다.

- Prepare

Prepare 단계는 method 2와 마찬가지로 SQL statement를 분석하고 이 statement에 이름을 부여한다. Prepare의 문법은 다음과 같다.

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable | <string_literal> };
<string_literal> ::=
    ' <sql_statement> '
  | " <sql_statement> "
  | <sql_statement>
  ;
```

<statement_name>은 precompiler에게 알려주는 식별자로서 host variable이 아니기 때문에 별도의 type을 갖거나 변수를 선언할 필요가 없다.

- Declare cursor

Declare 구문은 prepare된 statement에 대해 cursor를 선언한다. Cursor를 선언할 때 standing cursor와 동일한 cursor property를 사용할 수 있으며 그 구문은 다음과 같다.

```
EXEC SQL <dynamic declare cursor>;

<dynamic declare cursor> ::=
    DECLARE <cursor_name> <cursor properties> { FOR | IS } <statement_name>
    ;

<cursor properties> ::=
    [ <cursor sensitivity> ] [ <cursor scrollability> ] CURSOR [ <cursor holdability> ]
    | [ <odbc cursor type> ] CURSOR [ <cursor holdability> ]
    ;

<cursor sensitivity> ::=
    INSENSITIVE
  | SENSITIVE
  | ASENSITIVE
  ;

<cursor scrollability> ::=
    NO SCROLL
  | SCROLL
  ;

<cursor holdability> ::=
    WITH HOLD
  | WITHOUT HOLD
  ;

<odbc cursor type> ::=
    STATIC
  | KEYSET
  ;
```

<cursor properties>는 standing cursor와 의미 및 사용 이름이 동일하므로 이와 관련된 내용은 **Cursor Property**를 참조한다.

<statement_name>은 PREPARE 구문에서 지정된 이름이며, <cursor name>과 <statement name>은 precompiler에게 알려주는 식별자로서 host variable이 아니기 때문에 별도의 type을 갖거나 변수를 선언할 필요가 없다.

- Open cursor

Cursor를 open한다. Standing cursor의 **Open**과 비교하여 기본적인 면에서는 동일하지만 dynamic cursor의 open만 갖는 중요한 차이점이 있다. Dynamic cursor는 run-time에 SQL 문을 자유롭게 변경하여 사용할 수 있기 때문에 declare된 statement의 SQL에 따라 host variable이 결정된다. 따라서 dynamic cursor를 open할 때는 open 시점에 USING 절을 사용하여 host variable을 전달한다.

Dynamic cursor의 open 구문은 다음과 같다.

```
EXEC SQL <dynamic cursor open>;
```

```
<dynamic cursor open> ::=
    OPEN <cursor_name> [ USING <host_variable_list> ]
    ;
```

```
<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];
<host_variable_entry> ::= :host_variable [ [ INDICATOR ] :host_indicator ];
```

- Fetch cursor

Cursor로부터 fetch한다. Dynamic cursor의 fetch는 **Standing cursor의 Fetch**와 동일하다.

- Close cursor

Cursor를 close한다. Dynamic cursor의 close는 **Standing cursor의 Close**와 동일하다.

Example Program

다음은 dynamic method 1, 2, 3을 사용한 sample program이다.

```
/*
 * dyn2.gc
 * : dynamic method 1
 * : dynamic method 2
 * : dynamic method 3
 *
 */
#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

EXEC SQL BEGIN DECLARE SECTION;
typedef struct Record
{
    int          mEmpNo;
    varchar      mENAME[20 + 1];
    char         mJob[20];
    char         mSalary[10];
} Record;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
int UpdateSalary(char *aJob, int aBound, double aRatio);

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    EXEC SQL END DECLARE SECTION;
    printf("Connect GOLDSILOCKS ...\n");
    if(Connect("DSN=GOLDSILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
}

```

```

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

```

- Print RND employee increase 20% salary where salary < 2000

```

printf("print RND employee increase 20%% salary where salary < 2000\n");
UpdateSalary( "RND", 2000, 1.2 );
printf("\n\n");

```

- Print SUPPORT employee increase 10% salary where salary < 3000

```

printf("print SUPPORT employee increase 10%% salary where salary < 3000\n");
UpdateSalary( "SUPPORT", 3000, 1.1 );
printf("\n\n");

```

```

if(DropEmpTempTable() != SUCCESS)
{
    goto fail_exit;
}

```

```

printf("Disconnect GOLDILOCKS ...\n");
EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

```

```

printf("SUCCESS\n");
printf("#####\n");

```

```

return 0;

```

```

fail_exit:

```

```

    printf("\n\nFAILURE\n");
    printf("#####\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

```

```

    return 0;

```

```

}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR   sUid[80];
    VARCHAR   sPwd[20];
    VARCHAR   sConnStr[1024];
    EXEC SQL END DECLARE SECTION;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

```

- Create table

```

int CreateEmpTempTable()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char   sSqlStmt[8192];

```

```

EXEC SQL END DECLARE SECTION;
sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_RND" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"
        "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"
        "ENAME VARCHAR2(10),\n"
        "JOB VARCHAR2(9),\n"
        "SAL NUMBER(7,2),\n"
        "DEPTNO NUMBER(2) )\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
        "SELECT *\n"
        "FROM   EMP\n"
        "WHERE  JOB = 'RND'\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_SUPPORT" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "CREATE TABLE EMP_SUPPORT (\n"
        "EMPNO NUMBER(4) CONSTRAINT PK_EMP_SUPPORT PRIMARY KEY,\n"
        "ENAME VARCHAR2(10),\n"
        "JOB VARCHAR2(9),\n"

```

```

        "SAL NUMBER(7,2),\n"
        "DEPTNO NUMBER(2) )\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "INSERT INTO EMP_SUPPORT\n"
        "SELECT *\n"
        "FROM    EMP\n"
        "WHERE   JOB = 'SUPPORT'\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:

PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
EXEC SQL ROLLBACK WORK;

return FAILURE;
}

```

- Drop table

```

int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_RND;
    if(sqlca.sqlcode != 0)
    {

```



```

        goto fail_exit;
    }

    EXEC SQL DROP TABLE EMP_SUPPORT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int UpdateSalary(char *aJob, int aBound, double aRatio)
{
    EXEC SQL BEGIN DECLARE SECTION;
    Record      sRecord;
    char         sSelectSql[128];
    char         sUpdateSql[128];
    int          sBound = aBound;
    double       sRatio = aRatio;
    EXEC SQL END DECLARE SECTION;

    int sRecordCount = 0;
    int i;
    int sIsOpenCur = 0;

    sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s WHERE sal < :v1 FOR UPDATE
", aJob );

```

```

sprintf( sUpdateSql, "UPDATE EMP_%s SET sal = sal * :v1 WHERE CURRENT OF DYN_CUR", aJob);

EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL DECLARE DYN_CUR KEYSET CURSOR FOR SELECT_STMT;
EXEC SQL OPEN DYN_CUR USING :sBound;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
sIsOpenCur = 1;

EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL
        FETCH NEXT DYN_CUR
        INTO :sRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

```

```

}

sIsOpenCur = 0;
EXEC SQL CLOSE DYN_CUR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s ORDER BY SAL DESC", aJob
);
EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL OPEN DYN_CUR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
sIsOpenCur = 1;

printf("%s salary list\n", aJob);
sRecordCount = 0;
printf(" EMPNO      ENAME                JOB          SALARY\n");
printf("===== \n");
while( 1 )
{
    EXEC SQL
        FETCH DYN_CUR
        INTO :sRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
}

```

```

    }

    for(i = 0; i < sqlca.sqlerrd[2]; i++)
    {
        sRecordCount++;
        printf("%6d %20s %10s %10s\n",
            sRecord.mEmpNo,
            sRecord.mENAME.arr,
            sRecord.mJob,
            sRecord.mSalary);
    }
}

printf("=====\n");
printf("Record Count = %d\n", sRecordCount);
printf("=====\n");

sIsOpenCur = 0;
EXEC SQL CLOSE DYN_CUR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:

PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
if(sIsOpenCur == 1)
{
    EXEC SQL CLOSE DYN_CUR;
}
return FAILURE;
}

```

Multi-threaded Application

Multi-threaded application이란 하나의 process 안에서 다수의 실행 단위를 갖도록 작성된 응용 프로그램이다. Multi-threaded application은 여러 개의 응용 프로그램을 동시에 실행하는 것처럼 여러 개의 작업을 병렬로 처리할 수 있으며 하나의 process이기 때문에 동일한 주소 영역을 공유할 수 있다는 장점이 있다.

동일한 주소 영역을 공유한다는 것은 global 변수와 static 변수를 공유한다는 의미이므로 이러한 변수들에 접근할 때는 각 thread에서 이들에 대한 동시성 제어를 감안해서 접근할 수 있도록 응용 프로그램을 작성할 때 주의를 기울여야 한다.

GOLDILOCKS에서는 이를 위해 run-time context를 제공하고 있으며 run-time context는 Direct Attach (D/A) 모드와 Client/Server (C/S) 모드에서 약간의 차이점이 있다. 다음 장에서는 run-time context와 multi-threaded application을 작성하는 guideline에 대해 설명한다.

Run-time Context

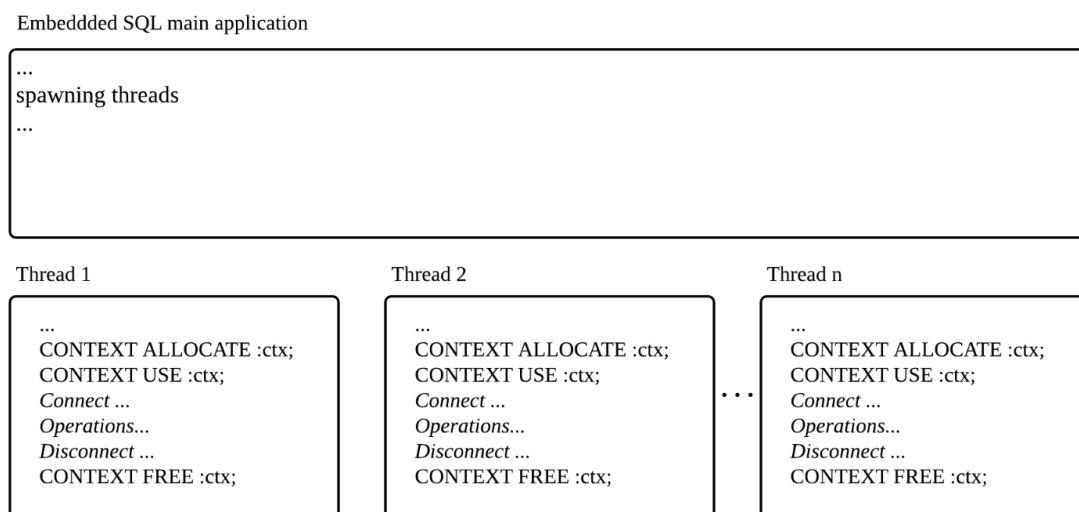
GOLDILOCKS의 embedded SQL에서 run-time context는 응용 프로그램에서 GOLDILOCKS로의 connection을 관리하기 위해 사용된다. Run-time context와 connection은 1 : 1 대응 관계를 가지며 run-time context를 connection 자체로 보아도 무방하다.

GOLDILOCKS는 구조적으로 D/A 모드에서 하나의 thread에 하나의 connection만 허용하므로 다수의 connection을 사용하려면 응용 프로그램을 multi-thread로 작성해야만 한다. 물론 C/S 모드는 network를 통한 server와의 connection이므로 하나의 thread에서 다수의 connection을 가질 수도 있다.

Direct Attach (D/A) 모드

D/A 모드로 동작할 때는 한 개의 thread가 한 개의 connection만 가질 수 있다. 따라서 n개의 connection을 소유한 응용 프로그램을 작성할 때는 n개의 thread로 작동시켜야 한다.

그림 2 D/A 모드에서 각 thread가 각자의 connection을 가지는 경우



Client/Server (C/S) 모드

C/S 모드로 동작할 때는 connection에 대한 별도의 제약이 없다. 한 개의 thread가 다수의 connection을 가질 수도 있고, 다수의 thread가 한 개의 connection을 공유할 수도 있다. 물론, n개의 thread가 m개의 connection을 공유할 수도 있다.

그림 3 다수의 thread가 한 개의 connection을 공유하는 경우

Embeddded SQL main application

```
...
CONTEXT ALLOCATE :ctx;
CONTEXT USE :ctx;
Connect ...
spawning threads
wait for threads
Disconnect ...
CONTEXT FREE :ctx;
...
```

Thread 1

```
...
CONTEXT USE :ctx;
Operations ...
...
```

Thread 2

```
...
CONTEXT USE :ctx;
Operations ...
...
```

Thread n

```
...
CONTEXT USE :ctx;
Operations ...
...
```

그림 4 한 개의 thread가 다수의 connection을 갖는 경우

Embeddded SQL main application

```
...
CONTEXT ALLOCATE :ctx1;
CONTEXT ALLOCATE :ctx2;
CONTEXT ALLOCATE :ctx3;

CONTEXT USE :ctx1;
Connect ...
CONTEXT USE :ctx2;
Connect ...
CONTEXT USE :ctx3;
Connect ...

CONTEXT USE :ctx1;
SQL operations ...
CONTEXT USE :ctx2;
SQL operations ...
CONTEXT USE :ctx3;
SQL operations ...

CONTEXT USE :ctx1;
Disconnect ...
CONTEXT USE :ctx2;
Disconnect ...
CONTEXT USE :ctx3;
Disconnect ...

CONTEXT FREE :ctx1;
CONTEXT FREE :ctx2;
CONTEXT FREE :ctx3;
...
```

Guideline

Multi-threaded application을 작성할 때는 다음 사항들을 고려해야 한다.

- SQLCA 변수를 thread-safe하게 선언한다. 각 thread 내에서 stack 변수로 선언하는 방법이 권장되며 그 예는 다음 단락의 example program을 참조한다.
- Multi-thread는 하나의 process 내에서 동일한 주소 영역을 갖기 때문에 static 변수나 global 변수를 공유한다. 따라서 이러한 변수들을 사용할 때 응용 프로그램에서 동시성 제어를 고려해야 한다.
- 하나의 run-time context를 여러 thread에서 동시에 사용하지 못하도록 해야 한다. 즉, run-time context를 사용할 때의 동시성 제어도 고려해야 한다.

Example Program

다음은 multi-threaded application의 sample 프로그램이다.

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define SUCCESS 0
#define FAILURE -1

#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
```

```

void *clientThread(void *args);

typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;

#define  THREAD_COUNT    2
char gJobName[THREAD_COUNT][20]= {
    "RND",
    "SUPPORT"
};

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sEName[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    pthread_t     thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int          i;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect(NULL, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
}

```

- Create client thread


```

for( i = 0; i < THREAD_COUNT; i ++ )
{
    param[i].mNo = i;
    param[i].mJobName = gJobName[i];
    if( pthread_create(&thread_id[i],
                      NULL,
                      clientThread,
                      &param[i]) != 0 )
    {
        printf( "Can't create thread %d!\n", i );
    }
    else
    {
        printf( "Create thread %d!\n", i );
    }
}

for( i = 0; i < THREAD_COUNT; i ++ )
{
    if( pthread_join(thread_id[i],
                     NULL) != 0 )
    {
        printf( "Error when waiting for thread %d to terminate!\n", i );
    }
    else
    {
        printf( "Stopped thread %d!\n", i );
    }
}

```

- Retrieve employee

EXEC SQL

```

DECLARE EMP_CUR CURSOR FOR
SELECT empno, ename, job, sal
FROM   EMP_TEMP
ORDER BY empno;

```

```

EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{

```

```

        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf(" EMPNO      ENAME                JOB      SALARY\n");
    printf("===== \n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO :sEmpNo, :sENAME, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        sRecordCount ++;

        printf("%6d %20s %10s %8ld\n",
            sEmpNo, sENAME.arr, sJob, sSalary);
    }

    printf("===== \n");
    printf("Record Count = %d\n", sRecordCount);
    printf("===== \n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

```

```

    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("#####\n");

    return 0;

fail_exit:

    printf("FAILURE\n");
    printf("#####\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUId[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
    struct sqlca sqlca;

```

- Log on GOLDILOCKS

```

strcpy((char *)sUId.arr, aUserID);
sUId.len = (short)strlen((char *)sUId.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);

```

- DB 연결

```

if( aCtx != NULL )
{
    EXEC SQL CONTEXT USE :aCtx;
    EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
}
else
{
    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
}
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

return FAILURE;
}

```

- DB 연결 해제

```

int Disconnect(sql_context aCtx)
{
    struct sqlca sqlca;
    if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL DISCONNECT;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL DISCONNECT;
    }
}

```

```

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

```

- Create table

```

int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

```

```

        return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

```

- Drop table

```

int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT    my_context;
    char           job_name[20 + 1];
}

```

```

EXEC SQL END DECLARE SECTION;
int          state = 0;
thread_param *param = (thread_param *)args;

EXEC SQL CONTEXT ALLOCATE :my_context;
state = 1;

EXEC SQL CONTEXT USE :my_context;
if(Connect(my_context, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
{
    goto fail_exit;
}
state = 2;

strcpy( job_name, param->mJobName );

EXEC SQL
    INSERT INTO EMP_TEMP
    SELECT *
    FROM   EMP
    WHERE  JOB = :job_name;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
state = 1;
if(Disconnect(my_context) != SUCCESS)
{
    goto fail_exit;
}
state = 0;
EXEC SQL CONTEXT FREE :my_context;
pthread_exit(0);
return NULL;

```

```

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    switch(state)
    {
        case 2:
            (void)Disconnect(my_context);
        case 1:
            EXEC SQL CONTEXT FREE :my_context;
            break;
        default:
            break;
    }

    pthread_exit(0);
    return NULL;
}

```

C++ Application

Output Filename의 확장자

GOLDILOCKS의 precompiler (gpec)은 embedded SQL 소스 코드를 precompile하여 C 코드를 생성한다. 따라서 출력 파일의 확장자는 기본적으로 .c가 된다. 일반적인 C++ 소스 코드는 compiler의 종류에 따라 다양한 형태의 파일 확장자를 갖는다.

이렇게 파일 확장자를 원하는 것으로 출력하고 싶은 경우 gpec의 출력 파일 지정 옵션인 -o를 사용한다. 예를 들어, testfile.gc를 testfile.cpp로 변환하려면 다음과 같이 실행한다.

```
gpec $(GPEC_OPT) testfile.gc -o testfile.cpp
```

SQLCA_STORAGE_CLASS

C++ application에서 동일한 symbol을 global 변수로 선언한 경우, symbol에 대한 충돌이 발생한다. GOLDILOCKS의 embedded SQL에서는 default로 sqlca 변수를 가지고 있는데 다수의 C++ file을 link할 때 이 변수의 충돌이 문제가 된다.

이를 피하기 위해 하나의 파일을 제외한 나머지 모든 파일들에 SQLCA_STORAGE_CLASS 매크로를 정의해야 한다. 가령 세 개의 파일들을 링크하여 응용 프로그램을 만들면 세 개 중에 두 개의 파일에 다음과 같이 SQLCA_STORAGE_CLASS 매크로를 정의한다.


```
#define SQLCA_STORAGE_CLASS extern
EXEC SQL INCLUDE SQLCA;
```

SQLCA_STORAGE_CLASS 매크로 정의는 반드시 다음 문장의 앞에 위치해야 한다.

```
EXEC SQL INCLUDE SQLCA;
```

XA

xa_open string 정의

xa_open string은 Resource Manager (RM)에 접속하기 위한 정보를 포함하고 있는데 자세한 내용은 **SQLDriverConnect 인자설명을** 참조한다.

다음은 xa_open string의 예이다.

```
DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN
```

Precompiler에서 XA 사용

Precompiler에서 XA를 사용할 때 다음 중 하나를 선택할 수 있다.

- Default connection을 사용
- Named connection을 사용

Default Connection 사용

xa_open string에 접속 정보만을 가지고 맺어진 connection을 사용한다.

다음은 default connection을 위한 xa_open string의 예이다.

```
DSN=GOLDILOCKS;UID=test;PWD=test
```

Default connection을 사용한 embedded SQL을 사용할 때는 다음과 같은 형태로 기술한다.

```
EXEC SQL
    UPDATE Deposit
    Set InterestRates = :value :value_ind
    WHERE AccountNumber = :account_number;
```

Named Connection 사용

xa_open string에 접속 정보와 함께 connection 이름을 사용한다.

Named connection을 위한 xa_open string의 예는 다음과 같다.

```
DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN
```

Named connection을 사용한 embedded SQL에서는 다음과 같이 connection 이름을 기술해 주어야 한다.

```
EXEC SQL AT XA_CONN
    UPDATE Deposit
    Set InterestRates = :value :value_ind
    WHERE AccountNumber = :account_number;
```

Example Program

- GOLDILOCKS Sample - XA

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

- Include GOLDILOCKS ODBC header

```
#include <goldilocks.h>
EXEC SQL INCLUDE SQLCA;
#define SUCCESS 0
#define FAILURE -1
#define PRINT_SQL_ERROR(aMsg) \
{ \
    printf("\n"); \
    printf(aMsg); \
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n", \
        sqlca.sqlcode, \
        SQLSTATE, \
        sqlca.sqlerrm.sqlerrmc ); \
}
```

- User-specific definitions

```

#define BUF_LEN 101
#define GOLDDILOCKS_SQL_THROW( aLabel )          \
    goto aLabel;
#define GOLDDILOCKS_SQL_TRY( aExpression )      \
    do                                           \
    {                                           \
        if( !(SQL_SUCCEEDED( aExpression ) ) ) \
        {                                           \
            goto GOLDDILOCKS_FINISH_LABEL;      \
        }                                           \
    } while( 0 )
#define GOLDDILOCKS_FINISH                     \
    goto GOLDDILOCKS_FINISH_LABEL;             \
GOLDDILOCKS_FINISH_LABEL:

```

- Print diagnostic record to console

```

void PrintDiagnosticRecord( SQLSMALLINT aHandleType, SQLHANDLE aHandle )
{
    SQLCHAR      sSQLState[6];
    SQLINTEGER    sNaiveError;
    SQLSMALLINT   sTextLength;
    SQLCHAR      sMessageText[SQL_MAX_MESSAGE_LENGTH];
    SQLSMALLINT   sRecNumber = 1;
    SQLRETURN     sReturn;

```

- SQLGetDiagRec returns the current values which includes an error, warning.

```

while( 1 )
{
    sReturn = SQLGetDiagRec( aHandleType,
                            aHandle,
                            sRecNumber,
                            sSQLState,
                            &sNaiveError,
                            sMessageText,
                            100,
                            &sTextLength );

    if( sReturn == SQL_NO_DATA )
    {
        break;
    }
}

```

```

    }
    GOLDILOCKS_SQL_TRY( sReturn );
    printf("\n=====\\n" );
    printf("SQL_DIAG_SQLSTATE      : %s\\n", sSQLState );
    printf("SQL_DIAG_NATIVE          : %d\\n", sNaiveError );
    printf("SQL_DIAG_MESSAGE_TEXT : %s\\n", sMessageText );
    printf("=====\\n" );
    sRecNumber++;
}
return;
GOLDILOCKS_FINISH;
printf("SQLGetDiagRec failure.\\n" );
return;
}

```

- Create table

```

int testCreateTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE IF EXISTS DEPOSIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN
        CREATE TABLE DEPOSIT (
            NAME          VARCHAR(30),
            BALANCE        INTEGER,
            ACCOUNTNUMBER VARCHAR(100),
            ACCOUNTDAY     DATE,
            INTERESTRATES  NUMBER(10, 5),
            PHONENUMBER    VARCHAR(30) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

```

```

    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
}

```

- Drop table

```

int testDropTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE DEPOSIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
}

```

- Insert function

```

int testInsert( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char          sName[BUF_LEN];
    int           sNameInd          = 0;
    int           sBalance          = 0;
    int           sBalanceInd       = 0;
    char          sAccountNumber[BUF_LEN];
    int           sAccountNumberInd = 0;
    DATE          sAccountDay;

```

```

int          sAccountDayInd          = 0;
double       sInterestRates         = 0;
int          sInterestRatesInd       = 0;
char         sPhoneNumber[BUF_LEN];
int          sPhoneNumberInd          = 0;
EXEC SQL END DECLARE SECTION;
sNameInd     = snprintf( (char*)sName, BUF_LEN, "sunje" );
sBalance     = 300000000;
sAccountNumberInd = snprintf( (char*)sAccountNumber, BUF_LEN, "9999-99-9999" );
sAccountDay.year      = 2009;
sAccountDay.month     = 1;
sAccountDay.day       = 1;
sAccountDay.hour      = 0;
sAccountDay.minute    = 0;
sAccountDay.second    = 0;
sAccountDay.fraction  = 0;
sInterestRates        = (double)5.0;
sPhoneNumberInd       = snprintf( (char*)sPhoneNumber, BUF_LEN, "010-9999-9999" );
EXEC SQL AT XA_CONN
    INSERT INTO DEPOSIT
    VALUES ( :sName :sNameInd,
             :sBalance :sBalanceInd,
             :sAccountNumber :sAccountNumberInd,
             :sAccountDay :sAccountDayInd,
             :sInterestRates :sInterestRatesInd,
             :sPhoneNumber :sPhoneNumberInd );
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

```

- The number of rows affected by INSERT statement

```

printf("\n%d row created.\n\n", sqlca.sqlerrd[2] );
return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}

```

- Update function

```

int testUpdate( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char      sCondition[BUF_LEN];
    int       sConditionInd      = 0;
    double    sValue             = 0;
    int       sValueInd          = 0;
    EXEC SQL END DECLARE SECTION;
    sValue     = (SQLREAL)6.0;
    sConditionInd = snprintf( (char*)sCondition,
                             BUF_LEN,
                             "9999-99-9999" );

    EXEC SQL AT XA_CONN
        UPDATE Deposit
        Set InterestRates = :sValue :sValueInd
        WHERE AccountNumber = :sCondition :sConditionInd;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

```

- The number of rows affected by UPDATE statement

```

printf("\n%d row updated.\n\n", sqlca.sqlerrd[2] );
return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}

```

- Select function

```

int testSelect( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char      sName[BUF_LEN];
    int       sNameInd          = 0;
    int       sBalance          = 0;
    int       sBalanceInd       = 0;
    char      sAccountNumber[BUF_LEN];
    int       sAccountNumberInd = 0;
}

```

```

DATE            sAccountDay;
int              sAccountDayInd          = 0;
double          sInterestRates          = 0;
int             sInterestRatesInd        = 0;
char            sPhoneNumber[BUF_LEN];
int             sPhoneNumberInd          = 0;
EXEC SQL END DECLARE SECTION;
int             sCount                   = 0;
int             sIsOpen                  = 0;
EXEC SQL DECLARE CUR1 CURSOR FOR
    SELECT NAME, BALANCE, ACCOUNTNUMBER, ACCOUNTDAY, INTERESTRATES, PHONENUMBER
    FROM DEPOSIT;
EXEC SQL AT XA_CONN
    OPEN CUR1;
if( sqlca.sqlcode != 0 )
{
    goto fail_exit;
}
sIsOpen = 1;
printf( "=====\n" );
while( 1 )
{
    EXEC SQL AT XA_CONN
        FETCH CUR1 INTO
            :sName :sNameInd,
            :sBalance :sBalanceInd,
            :sAccountNumber :sAccountNumberInd,
            :sAccountDay :sAccountDayInd,
            :sInterestRates :sInterestRatesInd,
            :sPhoneNumber :sPhoneNumberInd;
    if( sqlca.sqlcode == SQL_NO_DATA )
    {
        break;
    }
    else if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
    printf( "NAME          : " );
    if( sNameInd == -1 )
    {

```



```

        printf( "(null)" );
    }
    else
    {
        printf( "%s", sName );
    }
    printf( "\n" );
    printf( "BALANCE      : " );
    if( sBalanceInd == -1 )
    {
        printf( "(null)" );
    }
    else
    {
        printf( "%d", sBalance );
    }
    printf( "\n" );
    printf( "ACCOUNTNUMBER : " );
    if( sAccountNumberInd == -1 )
    {
        printf( "(null)" );
    }
    else
    {
        printf( "%s", sAccountNumber );
    }
    printf( "\n" );
    printf( "ACCOUNTDAY      : " );
    if( sAccountDayInd == -1 )
    {
        printf( "(null)" );
    }
    else
    {
        printf( "%4d-%02d-%02d", sAccountDay.year, sAccountDay.month, sAccountDay.day);
    }
    printf( "\n" );
    printf( "INTERESTRATES : " );
    if( sInterestRatesInd == -1 )
    {
        printf( "(null)" );
    }

```

```

    }
    else
    {
        printf( "%lf", sInterestRates );
    }
    printf( "\n" );
    printf( "PHONENUMBER   : " );
    if( sPhoneNumberInd == -1 )
    {
        printf( "(null)" );
    }
    else
    {
        printf( "%s", sPhoneNumber );
    }
    printf( "\n" );
    printf( "-----\n" );
    sCount ++;
}
printf( "=====\n" );
printf( "\n%d rows selected.\n\n", sCount );
sIsOpen = 0;
EXEC SQL AT XA_CONN
    CLOSE CUR1;
if( sqlca.sqlcode != 0 )
{
    goto fail_exit;
}
return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    if( sIsOpen == 1 )
    {
        EXEC SQL AT XA_CONN
            CLOSE CUR1;
    }
    return FAILURE;
}

```

- Delete function

```

int testDelete( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char      sCondition[BUF_LEN];
    int       sConditionInd          = 0;
    EXEC SQL END DECLARE SECTION;
    sConditionInd = snprintf( (char*)sCondition,
                             BUF_LEN,
                             "9999-99-9999" );

    EXEC SQL AT XA_CONN
        DELETE FROM DEPOSIT WHERE AccountNumber = :sCondition :sConditionInd;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
}

```

- The number of rows affected by DELETE statement

```
printf("\n%d row deleted.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

- Start function

```
int main( int aArgc, char** aArgv )
{
    SQLHENV          sEnv      = NULL;
    SQLINTEGER        sState    = 0;
    xa_switch_t *     sXaSwitch;
    XID                sXid;
    sXaSwitch = SQLGetXaSwitch();
}
```

- If a user calls `SQLAllocEnv()` which is included in `GOLDILOCKS` ODBC

[illegible]

- SQLSetEnvAttr sets attributes which controls aspects of environments.

```
GOLDILOCKS_SQL_TRY( SQLSetEnvAttr( sEnv,
                                   SQL_ATTR_ODBC_VERSION,
                                   (SQLPOINTER)SQL_OV_ODBC3,
                                   0 ) );

    if( (sXaSwitch->xa_open_entry)( "DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN", 0,
TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 2;
    sXid.formatID = 0;
    sXid.gtrid_length = 2;
    sXid.bqual_length = 1;
    memcpy( sXid.data, "100", sXid.gtrid_length + sXid.bqual_length );
```

- Create table

```
if( testCreateTable() != SUCCESS )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}
sState = 3;
if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}
sState = 4;
```

- Insert row

```
if( testInsert() != SUCCESS )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}
```

- Update row

```
if( testUpdate() != SUCCESS )
{
    GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
}
```

```
}
```

- Select row

```
if( testSelect() != SUCCESS )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
```

- Delete row

```
if( testDelete() != SUCCESS )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
sState = 3;
if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
sState = 2;
```

- Drop table

```
if( testDropTable() != SUCCESS )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
sState = 1;
if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
{
    GOLDDILOCKS_SQL_THROW( GOLDDILOCKS_FINISH_LABEL );
}
```

- SQLFreeHandleEnv releases resources related to the environment.

```

sState = 0;
GOLDILOCKS_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                   sEnv ) );

sEnv = NULL;
return EXIT_SUCCESS;
GOLDILOCKS_FINISH;
if( sEnv != NULL)
{
    PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
}
switch( sState )
{
    case 4:
        (void)(sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS );
        (void)(sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS );
        (void)(sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS );
    case 3:
        (void)testDropTable();
    case 2:
        (void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
    case 1:
        (void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );
        sEnv = NULL;
    default:
        break;
}
return EXIT_FAILURE;
}

```

27.4 Embedded SQL Reference

본 장에서는 GOLDBLOCKS의 embedded SQL 응용 프로그램에서만 사용할 수 있는 SQL 문에 대해 설명한다. 소스 코드 상에서 embedded SQL 문을 다룰 때는 반드시 다음 문법을 따른다.

```

<statement> ::= EXEC SQL <exec sql statement>;
<exec sql statement> ::=
    <embedded SQL statement>
  | <embedded get group_id statement>
  | <embedded specific statement>
  ;

<embedded SQL statement> ::=
    [ AT <db_name> ] [ ATOMIC ] [ FOR <iteration_count> ] <sql statement>
  ;

<embedded get group_id statement> ::=
    [ AT <db_name> ] <get group_id statement> <sql statement>
<embedded specific statement> ::=
    <autocommit statement>
  | <declare section statement>
  | <include statement>
  | <exception statement>
  | <context statement>
  | <option statement>
  ;

<autocommit statement> ::= [ AT <db_name> ] AUTOCOMMIT { ON | OFF };
<declare section statement> ::= { BEGIN | END } DECLARE SECTION;
<include statement> ::= INCLUDE { SQLCA | <identifier> };
<exception statement> ::= WHENEVER <exception_condition> <exception_action>;
<context statement> ::= CONTEXT <context action>;
<context action> ::=
    ALLOCATE :context_name
  | FREE :context_name
  | USE :context_name
  | USE DEFAULT
  ;

<option statement> ::= OPTION ( <option> );
<get group_id statement> :: GET GROUPID INTO :group_id;

```

EXEC SQL AT

기능

Embedded SQL 문에 적용할 connection 이름을 지정한다.

구문

```
EXEC SQL [ AT <db_name> ] ...  
<db_name> ::=  
    <identifier>  
    | :hostvar  
    ;
```

설명

Embedded SQL 응용 프로그램에서 connect 할 때, connection 이름을 지정할 수 있다. 이 이름은 특정 connection을 사용하여 embedded SQL 문을 수행하려 할 때 사용된다.

사용 예

```
{  
    ...  
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;  
  
    EXEC SQL AT :conn_name  
        UPDATE EMP  
        SET     sal = sal * 1.1  
        WHERE  JOB = 'SALES';  
    ...  
}
```

참조

관련 내용은 [연결](#)을 참조한다.

EXEC SQL ATOMIC INSERT

기능

Atomic array insert를 수행한다.

구문

```
EXEC SQL ATOMIC <insert_statement>;
```

설명

Embedded SQL 응용 프로그램에서 atomic array insert를 수행한다. Atomic array insert는 한 번의 명령으로 다수의 row를 삽입하기 위한 구문으로써 삽입되는 모든 row가 성공을 해야만 성공을 반환하고, row 중에 한 개라도 실패할 경우 전체 row 삽입이 실패한다.

한 번의 명령으로 삽입되기 때문에 개별적으로 row를 삽입하는 것보다 성능이 좋다.

사용 예

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값 설정

```
...
```

- emp_number, emp_name, dept_number를 INSERT

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
                                VALUES (:emp_number, :emp_name, :dept_number);
}
```

참조

관련 내용은 **Atomic Insert**를 참조한다.

EXEC SQL AUTOCOMMIT

기능

Autocommit 설정을 변경한다.

구문

```
EXEC SQL AUTOCOMMIT { ON | OFF };
```

설명

Autocommit은 다음과 같이 설정할 수 있다.

Flag	설명
ON	Statement를 수행한 후에 자동으로 commit 한다.
OFF	명시적인 commit 문장이 올 때까지 commit 하지 않는다.

사용 예

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL AUTOCOMMIT ON;
    EXEC SQL AT :conn_name
        UPDATE EMP
        SET     sal = sal * 1.1
        WHERE  JOB = 'SALES';

    ...
}
```

참조

관련 내용은 **Auto Commit**을 참조한다.

EXEC SQL BEGIN DECLARE SECTION

기능

Precompiler의 지시자로써 host variable 선언 영역을 지정한다.

구문

```
EXEC SQL BEGIN DECLARE SECTION;
```

설명

Host variable 선언 영역을 지정하는 precompiler 지시자로써 항상 EXEC SQL END DECLARE SECTION과 함께 사용된다. Precompiler가 이 문장을 만나면 declare section의 시작으로 판단하고 이후에 나오는 변수 선언을 host variable로 인식하여 처리한다.

사용 예

```
{  
    EXEC SQL BEGIN DECLARE SECTION;  
    int    emp_number[20];  
    char    emp_name[20][10];  
    int    dept_number[20];  
    EXEC SQL END DECLARE SECTION;  
  
    ...  
}
```

참조

관련 내용은 다음을 참조한다.

- Host Variable 선언
- EXEC SQL END DECLARE SECTION

EXEC SQL COMMIT RELEASE

기능

트랜잭션을 완료한 후에 connection을 종료한다.

구문

```
EXEC SQL [ AT <db_name> ] COMMIT [ WORK ] RELEASE;
```

설명

트랜잭션을 완료한 후에 현재 connection을 종료한다.

```
EXEC SQL AT :conn_name COMMIT RELEASE;
```

위의 문장은 다음과 동일하다.

```
EXEC SQL AT :conn_name COMMIT;
```

```
EXEC SQL AT :conn_name DISCONNECT;
```

사용 예

```
{  
    ...  
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;  
  
    EXEC SQL AT :conn_name  
        UPDATE EMP  
        SET     sal = sal * 1.1  
        WHERE  JOB = 'SALES';  
    EXEC SQL AT :conn_name COMMIT RELEASE;  
    ...  
}
```

참조

관련 내용은 **RELEASE Option**을 참조한다.

EXEC SQL CONNECT

기능

GOLDILOCKS와 connection을 맺는다.

구문

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password> [ AT <db_name> ] [
  USING <conn_string> ]
<db_name> ::=
    <identifier>
  | :hostvar
  ;
<user_name> ::=
    <identifier>
  | :hostvar
  ;
<password> ::=
    <identifier>
  | :hostvar
  ;
<conn_string> ::= :hostvar;
```

설명

GOLDILOCKS에 연결을 설정한다.

사용 예

```
{
  ...
  EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;
  ...
}
```

참조

관련 내용은 **Database 연결**을 참조한다.

EXEC SQL CONTEXT ALLOCATE

기능

Run-time context 메모리를 할당한다.

구문

```
EXEC SQL CONTEXT ALLOCATE :context;
```

설명

Run-time context 메모리를 할당한다. Run-time context를 할당하려면 declare section에 SQL_CONTEXT type의 변수를 선언한 후, 이 변수에 대해서 할당해야 한다. 이 구문은 단지 메모리를 할당하는 역할만 수행하므로 이를 사용하려면 USE를 지정한 뒤에 connect를 수행해야 한다.

사용 예

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET     sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL DISCONNECT;
    EXEC SQL CONTEXT FREE :ctxt;

    ...
}
```


참조

관련 내용은 `SQL_CONTEXT`를 참조한다.

EXEC SQL CONTEXT FREE

기능

Run-time context 메모리를 해제한다.

구문

```
EXEC SQL CONTEXT FREE :context;
```

설명

Run-time context 메모리를 해제한다. Run-time context를 해제하기 전에는 반드시 disconnect를 하여 더 이상 connection을 사용하지 않도록 해야한다. 그렇지 않으면 예기치 못한 오류가 발생할 수 있다.

사용 예

```
{  
    ...  
    EXEC SQL BEGIN DECLARE SECTION;  
    SQL_CONTEXT ctxt;  
    EXEC SQL END DECLARE SECTION;  
    EXEC SQL CONTEXT ALLOCATE :ctxt;  
  
    EXEC SQL CONTEXT USE :ctxt;  
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;  
  
    EXEC SQL  
        UPDATE EMP  
        SET     sal = sal * 1.1  
        WHERE  JOB = 'SALES';  
    EXEC SQL DISCONNECT;  
    EXEC SQL CONTEXT FREE :ctxt;  
  
    ...  
}
```

참조

관련 내용은 `SQL_CONTEXT`를 참조한다.

EXEC SQL CONTEXT USE

기능

Run-time context 사용을 알린다.

구문

```
EXEC SQL CONTEXT USE { :context | DEFAULT };
```

설명

Run-time context의 사용을 precompiler에 알리는 지시자로서 이제부터 사용할 run-time context를 지정한다. USE 구문에 SQL_CONTEXT 변수를 사용하면 사용자가 선언하여 할당한 run-time context를 사용하게 할 수 있으며, USE DEFAULT를 사용하면 응용 프로그램이 기본적으로 가지고 있는 default context를 사용하게 한다.

사용 예

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    double      max_sal;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET     sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL COMMIT RELEASE;
    EXEC SQL CONTEXT FREE :ctxt;

    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;
```

```
EXEC SQL
    SELECT MAX(sal)
    INTO    max_sal
    FROM    EMP
    WHERE   JOB = 'SALES';
EXEC SQL DISCONNECT;

...

}
```

참조

관련 내용은 `SQL_CONTEXT`를 참조한다.

EXEC SQL DISCONNECT

기능

GOLDILOCKS와의 connection을 종료한다.

구문

```
EXEC SQL [ AT <db_name> ] DISCONNECT [ ALL ]
```

설명

GOLDILOCKS와의 연결을 종료한다. AT 절을 이용하여 특정 connection을 해제할 수도 있고, AT 절을 사용하지 않으면 현재 자신이 사용하던 connection이 해제된다. DISCONNECT ALL을 사용하면 현재 응용 프로그램에서 사용하던 모든 connection이 해제된다.

사용 예

```
{  
    ...  
    EXEC SQL AT :conn_name DISCONNECT;  
    ...  
}
```

참조

관련 내용은 **Database 연결 해제**를 참조한다.

EXEC SQL END DECLARE SECTION

기능

Precompiler 지시자로써 host variable 선언 영역을 지정한다.

구문

```
EXEC SQL END DECLARE SECTION;
```

설명

Host variable 선언 영역을 지정하는 precompiler 지시자로써 항상 EXEC SQL BEGIN DECLARE SECTION과 함께 사용된다. Precompiler가 declare section의 분석을 진행하는 도중에 이 문장을 만나면 declare section이 종료되었다고 판단한다.

사용 예

```
{  
    EXEC SQL BEGIN DECLARE SECTION;  
    int    emp_number[20];  
    char   emp_name[20][10];  
    int    dept_number[20];  
    EXEC SQL END DECLARE SECTION;  
  
    ...  
}
```

참조

관련 내용은 다음을 참조한다.

- Host Variable 선언
- EXEC SQL BEGIN DECLARE SECTION

EXEC SQL FOR

기능

Array operation에서 array 개수를 지정한다.

구문

```
EXEC SQL FOR { :array_count | integer_constant } <sql statement>;
```

설명

SQL 문의 host 변수가 array일 경우 array count를 지정하는 precompiler 지시자이다. FOR 절이 주어진다면 실제 host array의 array count는 무시하고 FOR 절에서 지정된 개수만큼의 array만 수행한다.

Array count를 의미하는 상수나 변수는 반드시 정수형이어야만 한다.

사용 예

```
EXEC SQL BEGIN DECLARE SECTION;  
int    emp_number[20];  
char   emp_name[20][10];  
int    dept_number[20];  
int    record_cnt;  
EXEC SQL END DECLARE SECTION;
```

- emp_number, emp_name, dept_number 값을 설정

...

- emp_number, emp_name, dept_number를 INSERT

```
record_cnt = 10;  
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)  
VALUES (:emp_number, :emp_name, :dept_number);
```

참조

관련 내용은 **FOR 절 사용**을 참조한다.

EXEC SQL GET GROUPID INTO

기능

SQL statement의 group ID를 얻는다.

구문

```
EXEC SQL [ AT <db_name> ] GET GROUPID INTO :group_id { delete_stmt | insert_stmt | select_stmt
| update_stmt };
```

설명

Global connection을 사용하는 cluster 환경에서 SQL statement의 group ID를 얻는다. 호스트 변수 :group_id에는 signed numeric type만 사용할 수 있다. Group ID를 얻을 수 있는 SQL statement는 delete, insert, select, update로 한정되며 table에 shard key가 설정되어 있어야 한다.

group ID를 얻은 SQL statement는 내부적으로 SQLExecute를 실행하지 않은 SQLPrepare 상태로 cache 된다.

주의

유효하지 않은 group ID 값인 -1이 반환될 수 있다.

사용 예

```
{
EXEC SQL BEGIN DECLARE SECTION;
int  group_id[10];
int  emp_no[10];
char emp_name[10][20];
int  dept_no[10];
EXEC SQL END DECLARE SECTION;
```

- emp_no, emp_name, dept_no 값을 설정한다.

...

- group id를 얻는다.

```
EXEC SQL GET GROUPID INTO :group_id  
        INSERT INTO emp (empno, ename, deptno) VALUES (:emp_no, :emp_name, :dept_no);
```

- emp_no, emp_name, dept_no을 INSERT 한다.

```
EXEC SQL INSERT INTO emp (empno, ename, deptno) VALUES (:emp_no, :emp_name, :dept_no);  
}
```

EXEC SQL INCLUDE

기능

Embedded SQL header file을 포함시킨다.

구문

```
EXEC SQL INCLUDE <Header file name>;
```

설명

Embedded SQL header file을 포함시킨다. C 언어의 `#include` 문으로 header file을 포함시킬 경우, 이는 precompiler에서 해석하지 않기 때문에 header file 내에 declare section 등 precompiler가 알아야 할 내용이 있어도 이를 인식할 수가 없다. 따라서 precompiler가 알아야 하는 embedded SQL 구문이 사용될 경우, EXEC SQL INCLUDE 구문을 사용해야 한다.

사용 예

```
EXEC SQL INCLUDE decl.h;
```

참조

관련 내용은 [Precompiled Header File](#)을 참조한다.

EXEC SQL INCLUDE SQLCA

기능

sqlca.h header file을 포함시킨다.

구문

```
EXEC SQL INCLUDE SQLCA;
```

설명

EXEC SQL INCLUDE 구문의 특별한 형태로써 GOLDDILOCKS에서 제공하는 *sqlca.h* header file을 포함시킨다. 이 header file은 embedded SQL 응용 프로그램에서 run-time exception handling을 하기 위해 반드시 필요하다.

사용 예

```
EXEC SQL INCLUDE SQLCA;
```

참조

관련 내용은 [Run-time Error 감지](#)를 참조한다.

EXEC SQL OPTION

기능

Embedded SQL 소스 코드를 precompile 하는 과정에서 option을 적용한다.

구문

```
EXEC SQL OPTION ( <option_desc> );  
<option_desc> ::=  
    INCLUDE = <directory path>  
    ;
```

설명

Embedded SQL 소스 코드를 precompiler 하는 과정에 적용할 option을 기술한다. 현재 version에서는 INCLUDE 경로 지정만 지원하고 있으며 이 option은 EXEC SQL INCLUDE에서 precompile 할 header file이 위치한 디렉토리를 기술한다.

사용 예

```
EXEC SQL OPTION ( INCLUDE = include );
```

참조

관련 내용은 **Header File 경로 지정**을 참조한다.

EXEC SQL ROLLBACK RELEASE

기능

트랜잭션을 rollback 한 후에 connection을 종료한다.

구문

```
EXEC SQL [ AT <db_name> ] ROLLBACK [ WORK ] RELEASE;
```

설명

트랜잭션을 rollback 한 후에 현재 connection을 종료한다.

```
EXEC SQL AT :conn_name ROLLBACK RELEASE;
```

위의 문장은 다음과 동일하다.

```
EXEC SQL AT :conn_name ROLLBACK;  
EXEC SQL AT :conn_name DISCONNECT;
```

사용 예

```
{  
    ...  
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;  
  
    EXEC SQL AT :conn_name  
        UPDATE EMP  
        SET     sal = sal * 1.1  
        WHERE  JOB = 'SALES';  
    EXEC SQL AT :conn_name ROLLBACK RELEASE;  
    ...  
}
```

참조

관련 내용은 **RELEASE Option**을 참조한다.

EXEC SQL WHENEVER

기능

Embedded SQL 응용 프로그램에서 run-time exception handling을 실행한다.

구문

```
EXEC SQL WHENEVER <conditions> <actions>;
<conditions> ::=
    SQLERROR
  | SQLWARNING
  | NOT FOUND
  | SQLSTATE <sqlstate class value>[<sqlstate subclass value>]
  ;
<sqlstate_char> ::= [0-9A-Z];
<sqlstate class value> ::= <sqlstate_char> <sqlstate_char>;
<sqlstate subclass value> ::= <sqlstate_char> <sqlstate_char> <sqlstate_char>;
<actions> ::=
    CONTINUE
  | GOTO <label>
  | STOP
  | DO <c statements>
  ;
```

설명

Embedded SQL 응용 프로그램에서 run-time exception handling을 자동화하여 처리한다. 네 가지 condition이 있고 condition마다 한 개의 action을 지정할 수 있는데 이 action은 필요에 따라 다시 지정할 수도 있다. 자세한 내용은 **Handling Implicit Error**를 참조한다.

사용 예

```
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLERROR GOTO exit_label;
EXEC SQL WHENEVER NOT FOUND DO break;
EXEC SQL WHENEVER SQLERROR GOTO close_label;
EXEC SQL WHENEVER SQLERROR DO sql_error();
```

```
EXEC SQL WHENEVER SQLWARNING CONTINUE;  
EXEC SQL WHENEVER SQLSTATE HY000 DO sql_error();
```

참조

관련 내용은 **Handling Implicit Error**를 참조한다.

28.

PDO

28.1 PDO 개요

GOLDILOCKS PDO (PDO_GOLDILOCKS) 드라이버는 PDO에서 GOLDILOCKS 데이터베이스에 접근할 수 있도록 PDO 인터페이스를 제공하며, PECL 패키지로 제공된다.

PDO_GOLDILOCKS는 GOLDILOCKS CLI 드라이버를 기반으로 작성되었다.

28.2 설치/구성

요구 사항

PHP와 동일한 시스템에 GOLDILOCKS의 클라이언트 패키지를 설치해야 하고 \$GOLDILOCKS_HOME 환경 변수도 설정해야 한다.

설치

PECL을 이용한 설치

PDO_GOLDILOCKS 패키지 파일이 없을 경우 패키지 파일을 생성한다.

```
% pecl pickle
Attempting to process the second package file
Package PDO_GOLDILOCKS-3.2.0.tgz done
```

PDO_GOLDILOCKS 패키지 파일이 있을 경우 PECL을 이용해 패키지를 설치한다.

```
% sudo -E pecl install PDO_GOLDILOCKS-3.2.0.tgz
```

소스 빌드를 통한 설치

1. PDO_GOLDILOCKS 압축 해제

```
% tar xvzf PDO_GOLDILOCKS-3.2.0.tgz
% cd PDO_GOLDILOCKS-3.2.0
```

2. 빌드 환경 생성

phpize 명령어를 사용하여 PDO_GOLDILOCKS 확장 모듈을 위한 빌드 환경을 만들어 준다.
만약 phpize 명령어를 찾지 못할 경우, 적절한 버전의 PHP devel 패키지를 설치해야 한다.

```
% whereis phpize
phpize: /usr/bin/phpize /usr/share/man/man1/phpize.1.gz
```

```
% /usr/bin/phpize
Configuring for:
PHP Api Version:      20121113
Zend Module Api No:   20121212
Zend Extension Api No: 220121212
```

3. 빌드

--with-pdo-goldilocks 옵션에 \$GOLDILOCKS_HOME 경로를 추가하고 --with-php-config 옵션에 php-config 경로를 추가하여 configure를 실행한 후에 빌드한다.

```
% whereis php-config
php-config: /usr/bin/php-config /usr/share/man/man1/php-config.1.gz
```

```
% ./configure --with-php-config=/usr/bin/php-config --with-pdo-goldilocks=/home/goldilocks/
goldilocks_home/
```

```
% make
```

4. 설치

```
% sudo -E make install
```

PHP 설정 파일 수정

CentOS 6.0 이상 버전이나 Fedora 15 이상 버전을 사용할 경우 /etc/php.d 디렉토리에 pdo_goldilocks.ini 파일을 생성하고 다음과 같이 extension 속성을 추가해야 한다.

```
extension=pdo_goldilocks.so
```

다른 운영체제를 사용할 경우 php.ini 파일에 extension_dir을 적합하게 설정한 후에 다음과 같이 extension 속성을 추가한다.

```
extension=pdo_goldilocks.so
```

웹 서버 재시작

웹 서버를 다시 시작한다.

PDO_GOLDILOCKS 설치 확인

phpinfo()를 사용하여 PDO_GOLDILOCKS 활성화 여부를 확인한다.

```
% php -i | grep PDO
PDO
PDO support => enabled
PDO drivers => goldilocks, sqlite
PDO Driver for GOLDILOCKS => enabled
PDO Driver for SQLite 3.x => enabled
```

28.3 사용

데이터 원본 이름 (DSN)

PDO_GOLDILOCKS의 데이터 원본 이름 (DSN)은 다음과 같이 구성된다.

속성	설명
DSN prefix	goldilocks
DSN	ODBC 데이터 원본 이름 (DSN)
HOST	서버의 IP 주소
PORT	서버의 PORT 번호
UID	사용자 ID
PWD	사용자 password

28.4 사용 예

- Connecting to GOLDDLOCKS

```
print "\n[Connecting to GOLDDLOCKS]\n";
$db = new PDO('goldilocks:HOST=192.168.0.16;PORT=22581', 'test', 'test');
//$db = new PDO('goldilocks:DSN=GOLDDLOCKS;UID=test;PWD=test');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

- Set test data

```
$result = $db->exec('DROP TABLE IF EXISTS t1');
$result = $db->exec('CREATE TABLE t1 ( id INTEGER GENERATED BY DEFAULT AS IDENTITY, name
VARCHAR(32) )');
$result = $db->exec("INSERT INTO t1(name) VALUES ('Carol')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Ted')");
$result = $db->exec("INSERT INTO t1(name) VALUES (null)");
$result = $db->exec("INSERT INTO t1(name) VALUES ('William')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Chelsea')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Colin')");
$result = $db->exec('DROP TABLE IF EXISTS t2');
$result = $db->exec('CREATE TABLE t2 ( id INTEGER, name VARCHAR(32) )');
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");
$result = $db->exec('DROP TABLE IF EXISTS images');
$result = $db->exec('CREATE TABLE images ( id INTEGER, contenttype VARCHAR(256), imagedata
LONG VARBINARY )');
$result = $db->exec('DROP PROCEDURE IF EXISTS sp_out_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_out_string( v1 OUT VARCHAR(32) ) ".
    "IS ".
    "BEGIN ".
    "    v1 := 'WORLD'; ".
    "END; ");
$result = $db->exec('DROP PROCEDURE IF EXISTS sp_inout_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_inout_string( v1 INOUT VARCHAR(32) ) ".
    "IS ".
    "BEGIN ";
```

```

        v1 := V1 || ' WORLD'; ".
    "END; ");

```

- Quoting a normal string

```

print "\n[Quoting a normal string]\n";
$string = 'Nice';
print "Unquoted string: $string\n";
print "Quoted string : " . $db->quote($string) . "\n";

```

- Quoting a dangerous string

```

print "\n[Quoting a dangerous string]\n";
$string = 'Naughty \' string';
print "Unquoted string: $string\n";
print "Quoted string : " . $db->quote($string) . "\n";

```

- Quoting a complex string

```

print "\n[Quoting a complex string]\n";
$string = "Co'mpl'lex \"st\"ring";
print "Unquoted string: $string\n";
print "Quoted string : " . $db->quote($string) . "\n";

```

- Handling an error

```

print "\n[Error Handling]\n";
try
{
    //connect as appropriate as above
    $db->query('invalid query'); //invalid query!
}
catch(PDOException $ex)
{
    print "ERROR:". $ex->getMessage(). "\n";
}

```

- Retrieving column metadata

```

print "\n[Retrieving column metadata]\n";
$stmt = $db->query('SELECT * FROM t1');
foreach(range(0, $stmt->columnCount() - 1) as $column_index)

```

```
{
    $meta = $stmt->getColumnMeta($column_index);
    var_dump($meta);
}
```

- Running simple select statements

```
print "\n[Running Simple Select Statements]\n";
print "\n#1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row['ID'].' '.$row['NAME']."\n";
}
print "\n#2\n";
$stmt = $db->query('SELECT * FROM t1');

while($row = $stmt->fetch(PDO::FETCH_ASSOC))
{
    print $row['ID'].' '.$row['NAME']."\n";
}
```

- Fetching rows using different fetch styles

```
print "\n[Fetching rows using different fetch styles]\n";
$stmt = $db->prepare("SELECT id, name FROM t1");
$stmt->execute();
print("PDO::FETCH_ASSOC: ");
print("Return next row as an array indexed by column name\n");
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
print("\n");
print("PDO::FETCH_BOTH: ");
print("Return next row as an array indexed by both column name and number\n");
$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
print("\n");
print("PDO::FETCH_LAZY: ");
print("Return next row as an anonymous object with column names as properties\n");
$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
print("\n");
print("PDO::FETCH_OBJ: ");
```

```

print("Return next row as an anonymous object with column names as properties\n");
$result = $stmt->fetch(PDO::FETCH_OBJ);
print $result->NAME;
print("\n");

```

- Fetching rows with a scrollable cursor

```

print "\n[Fetching rows using different fetch styles]\n";
print "\n[Fetching rows with a scrollable cursor]\n";
$sql = 'SELECT id, name FROM t1 ORDER BY id';
print "Reading forwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT) )
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    }
    $stmt = null;
}
catch (PDOException $e)
{
    print $e->getMessage();
}
print "Reading backwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
    do
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    } while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR) );
    $stmt = null;
}
catch (PDOException $e)
{

```



```
    print $e->getMessage();
}
```

- Constructing an order

```
print "\n[Construction order]\n";
class Person
{
    private $NAME;
    public function __construct()
    {
        $this->tell();
    }
    public function tell()
    {
        if (isset($this->NAME))
        {
            print "I am {$this->NAME}.\n";
        }
        else
        {
            print "I don't have a name yet.\n";
        }
    }
}

$stmt = $db->query("SELECT name FROM t1");
$stmt->setFetchMode(PDO::FETCH_CLASS, 'Person');
$person = $stmt->fetch();
$person->tell();
$stmt->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE, 'Person');
$person = $stmt->fetch();
$person->tell();
```

- Running simple INSERT, UPDATE, or DELETE statements

```
print "\n[Running Simple INSERT, UPDATE, or DELETE statements]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('John'), ('Marry')");
print $affected_rows." were affected\n";
```

- Getting the last insert id

```

print "\n[Getting the Last Insert Id]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('Smith')");
$insertId = $db->lastInsertId();
print "last insert id : ".$insertId."\n";

```

- Running statements with parameters

```

print "\n[Running Statements With Parameters]\n";
print "\n#1 array\n";
$id = 1;
$name = 'John';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->execute(array($id, $name));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->bindValue(1, $id, PDO::PARAM_INT);
$stmt->bindValue(2, $name, PDO::PARAM_STR);
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);

```

- Named placeholders

```

print "\n[Named Placeholders]\n";
print "\n#1 array\n";
$id = 1;
$name = 'John';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->execute(array(':name' => $name, ':id' => $id));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$stmt->bindValue(':name', $name, PDO::PARAM_STR);

```

```
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
```

- Executing prepared statements in a loop

```
print "\n[Executing prepared statements in a loop]\n";
$values = array('bob', 'alice', 'lisa', 'john');
$name = '';
$stmt = $db->prepare("INSERT INTO t1(name) VALUES(:name)");
$stmt->bindParam(':name', $name, PDO::PARAM_STR);
foreach($values as $name)
{
    $stmt->execute();
}
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
```

- Calling a stored procedure with an output parameter

```
print "\n[Calling a stored procedure with an output parameter]\n";
$stmt = $db->prepare("CALL sp_out_string(?)");
$stmt->bindParam(1, $value, PDO::PARAM_STR, 32);
$stmt->execute();
print "procedure returned $value\n";
```

- Calling a stored procedure with an input/output parameter

```
print "\n[Calling a stored procedure with an input/output parameter]\n";
$stmt = $db->prepare("CALL sp_inout_string(?)");
$value = 'HELLO';
$stmt->bindParam(1, $value, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 32);
$stmt->execute();
print "procedure returned $value\n";
```

- Transactions

```
print "\n[Transactions]\n";
print "\n#1 commit\n";
try
```

```

{
    $db->beginTransaction();

    $db->exec("INSERT INTO t1(name) VALUES('kim')");
    $name = 'lee';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));

    $id = 3;
    $name = 'park';
    $stmt = $db->prepare("INSERT INTO t2 VALUES(?, ?)");
    $stmt->execute(array($id, $name));

    $db->commit();
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}
print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\n#2 rollback\n";
try
{
    $db->beginTransaction();

    $db->exec("INSERT INTO t1(name) VALUES('KIM')");
    $name = 'LEE';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));

```

```

$id = 3;
$name = 'PARK';
$stmt = $db->prepare("INSERT INTO invalid VALUES(?, ?)");
$stmt->execute(array($id, $name));

$db->commit();
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}
print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}

```

- Inserting an image into a database

```

print "\n[Inserting an image into a database]\n";
try
{
    $stmt = $db->prepare("INSERT INTO images (id, contenttype, imagedata) VALUES (?, ?, ?)");
    $id = 1;
    $type = "image/png";
    $fp = fopen("logo.png", "rb");
    $stmt->bindParam(1, $id, PDO::PARAM_INT);
    $stmt->bindParam(2, $type, PDO::PARAM_STR, 256);
    $stmt->bindParam(3, $fp, PDO::PARAM_LOB);
    $db->beginTransaction();
    $stmt->execute();
    $db->commit();
}
catch(PDOException $ex)

```

```

{
    $db->rollBack();
    print $ex->getMessage()."\n";
}
foreach($db->query('SELECT id, contenttype, lengthb(imagedata) FROM images') as $row)
{
    print $row[0].' '.$row[1].' '.$row[2]."\n";
}

```

- Displaying an image from a database

```

print "\n[Displaying an image from a database]\n";
try
{
    $stmt = $db->prepare("SELECT contenttype, imagedata FROM images WHERE id = ?");
    $id = 1;
    $stmt->execute(array($id));
    $stmt->bindColumn(1, $type, PDO::PARAM_STR, 256);
    $stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
    $stmt->fetch(PDO::FETCH_BOUND);
    print 'Content-Type: '.$type."\n";
    print 'size: '.file_put_contents($id.".png", $lob)."\n";
}
catch(PDOException $ex)
{
    print $ex->getMessage()."\n";
}

```

29.

PyDBC

29.1 GOLDDILOCKS PyDBC

개요

PyDBC는 **Python Database API Specification v2.0(PEP 249)**를 준수하는 API를 사용하여 GOLDDILOCKS 데이터베이스에 접속하는 Python을 프로그래밍한다.

PyDBC에는 Python 표준 라이브러리가 필요하고, GOLDDILOCKS 데이터베이스를 연결하고 조작하는 내부 연산은 ODBC API를 호출하기 때문에 ODBC 라이브러리가 필요하다. PyDBC는 ODBC 라이브러리 \$GOLDDILOCKS_HOME/lib/libgoldilocksas.so를 기본으로 사용하는데 이는 사용자가 갱신할 수도 있다.

PyDBC의 내부 연산은 ODBC 드라이버를 사용하기 때문에 **ODBC 구성요소 개요**와 동일하다. 응용 프로그램이 드라이버 관리자로 링크하는 아키텍처와 응용 프로그램이 GOLDDILOCKS ODBC driver 라이브러리로 링크하는 아키텍처가 있다.

버전 체계

다음과 같이 pygoldilocks.so 파일을 실행하여 GOLDDILOCKS PyDBC 버전 정보를 확인할 수 있다.

```
shell>python
>>> import pygoldilocks
>>> print pygoldilocks.version
3.2.0
```

현재 GOLDDILOCKS PyDBC 드라이버 버전은 GOLDDILOCKS 버전에 맞춘 3.2.0이며 이 드라이버는 표준 Python database API 2.0을 준수한다. PyDBC 드라이버는 Python 2.7, 3.4, 3.5, 3.6 버전을 지원하는데 PyDBC 드라이버 라이브러리는 각 Python 버전에 맞게 설치해야 한다.

설치

PyDBC를 설치하려면 소스를 구축해야 한다. PyDBC는 GOLDDILOCKS_HOME/lib에 위치한 gdlds 라이브러리를 링크하며 GOLDDILOCKS_HOME/include에 위치한 goldilocks.h 헤더 파일을 포함하므로 환경 변수 GOLDDILOCKS_HOME이 적절한 위치에 설정되어야 한다.

Linux에서 설치하기

Linux에는 gcc 컴파일러가 필요하며 다음과 같이 구축한다.

```
shell> sudo python setup.py install
```

노트

HP-UX와 AIX 플랫폼은 지원하지 않는다.

Windows에서 설치하기

Window에서는 다음과 같이 구축한다.

```
shell> python setup.py install
```

PyDBC를 컴파일 하려면 python 버전에 맞는 적절한 Microsoft Visual C++ 컴파일러를 사용해야 한다.

자세한 내용은 <https://wiki.python.org/moin/WindowsCompilers>를 참조한다.

- Python 2.4 또는 2.5 버전을 구축하려면 Visual Studio 2003.NET 컴파일러가 필요하다. 이 컴파일러는 유료 버전만 있다.
- Python 2.6, 2.7, 3.0, 3.2 버전을 구축하려면 Visual C++ 2008 컴파일러가 필요하다. 이 컴파일러의 무료 버전은 Visual C++ 2008 Express이다.
- Python 3.3, 3.4 버전을 구축하려면 Visual C++ 2010 컴파일러가 필요하다. 이 컴파일러의 무료 버전은 Visual C++ 2010 Express이다.
- Python 3.5, 3.6 버전을 구축하려면 Visual C++ 2014 또는 VC 2017 컴파일러가 필요하다.
- Python 3.7 버전을 구축하려면 Visual C++ 2017 컴파일러가 필요하다.

사용 예

- Connection을 얻기 위해 다음과 같이 작성한다.

```
import pygoldilocks
cnxn = pygoldilocks.connect( 'DSN=GOLDILOCKS;UID=test;PWD=test' )
```

PyDBC의 모듈인 pygoldilocks의 내장함수 connect를 호출하여 커넥션을 얻을 수 있다.

노트

DSN을 사용하기 위해서는 **odbc.ini 파일**이 사전에 작성되어 있어야 한다.

- 다음과 같이 cursor와 row 클래스를 사용할 수 있다.

```
cursor = cnxn.cursor()
cursor.execute( "SELECT NAME, ADDRESS FROM EMP" )
rows = cursor.fetchall()
for row in rows:
    print row.A, row.B
cursor.close()
cnxn.close()
```

29.2 API Reference

pygoldilocks 모듈

pygoldilocks object는 Python Database API Specification v2.0을 준수한다.
자세한 내용은 Python DB API 모듈을 참조한다.

속성

- version
 - pygoldilocks 모듈의 버전은 GOLDDLOCKS 데이터베이스의 버전을 따른다. 버전은 major.minor.patch 형식의 문자열이다.
- apilevel
 - DB API level 2.0을 가리키며 값은 "2.0" 문자열 상수이다.
- lowercase
 - 결과값으로 나온 row 객체에서 column 이름을 소문자로 할지 여부를 제어한다. 기본값은 false이다. 데이터베이스 column의 대소문자가 일치하지 않을 때 유용하다.
- threadsafety
 - 상수 1이며, thread가 모듈은 공유해도 연결은 공유하지 않는다.
- paramstyle
 - 매개 변수를 나타내며 그 값은 물음표를 의미하는 문자열 상수 "qmark"이다

connect

데이터베이스와 새로운 연결을 만든다.

`connect( *connectionstring, **kwargs )`

ODBC 연결 문자열과 키워드를 입력한다. 키워드는 다음과 같다.

키워드	설명	기본값
attrs_before	Connection되기 전에 설정해야 하는 속성을 지정한다. 값을 dictionary 타입으로 받는다.	-
autocommit	auto commit 여부를 설정한다. False인 경우, connection.commit을 호출해야 데이터베이스에 반영된다.	False
readonly	True인 경우, connection은 readonly로 설정된다.	False

키워드	설명	기본값
timeout	Connection을 위한 timeout을 설정한다. SQL_ATTR_LOGIN_TIMEOUT이 설정된다.	-

- attrs_before
 - 연결 전에 설정되는 옵션을 지정한다. 이 옵션들은 `SQLSetConnectAttr`을 사용하여 설정한다. 속성과 값은 딕셔너리 타입으로 받는다. 설정에 대한 자세한 내용은 `ODBC 속성`을 참조한다.

```
cnxn = pygoldilocks.connect( "DSN=GOLDDILOCKS", attr_before={ pygoldilocks.SQL_ATTR_MAX_ROWS :
1000 })
```

Date

```
>>> print pygoldilocks.Date(1984,11,23), type(pygoldilocks.Date(1984,11,23))
1984-11-23 <type 'datetime.date'>
```

주어진 값에 해당하는 date 객체를 생성한다.

Time

```
>>> print pygoldilocks.Time(11,23,23), type(pygoldilocks.Time(11,23,23))
11:23:23 <type 'datetime.time'>
```

주어진 값에 해당하는 time 객체를 생성한다.

Timestamp

```
>>> print pygoldilocks.Timestamp(1984,11,23,11,23,23), type(pygoldilocks.Timestamp
(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

주어진 값에 해당하는 datetime 객체를 생성한다.

DATETIME

```
>>> print pygoldilocks.DATETIME(1984,11,23,11,23,23), type(pygoldilocks.DATETIME
(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

주어진 값에 해당하는 datetime 객체를 생성한다. **Timestamp**와 동일하다.

Binary

```
>>> print pygoldilocks.Binary('binary'), type(pygoldilocks.Binary('binary'))
binary <type 'bytearray'>
```

주어진 값에 해당하는 bytearray 객체를 생성한다. **BINARY**와 동일하다.

BINARY

```
>>> print pygoldilocks.BINARY('binary'), type(pygoldilocks.BINARY('binary'))
binary <type 'bytearray'>
```

주어진 값에 해당하는 bytearray 객체를 생성한다.

STRING

```
>>> print pygoldilocks.STRING('str'), type(pygoldilocks.STRING('str'))
str <type 'str'>
```

주어진 값에 해당하는 str 객체를 생성한다.

NUMBER

```
>>> print pygoldilocks.NUMBER(100.001), type(pygoldilocks.NUMBER(100.001))
100.001 <type 'float'>
```

주어진 값에 해당하는 float 객체를 생성한다.

ROWID

```
>>> print pygoldilocks.ROWID('AA'), type(pygoldilocks.ROWID('AA'))
AA <type 'str'>
```

데이터베이스의 row ID column을 기술하기 위해 사용되며 str 객체를 반환한다.

TimeFromTicks

```
>>> print pygoldilocks.TimeFromTicks( 10 )  
09:00:10
```

인자값으로 설정된 `datetime.time` 객체를 반환한다.

DateFromTicks

```
>>> print pygoldilocks.DateFromTicks( 360000 )  
1970-01-05
```

인자값으로 설정된 `datetime.date` 객체를 반환한다.

TimestampFromTicks

```
>>> print pygoldilocks.DateFromTicks( 360000 )  
1970-01-05
```

인자값으로 설정된 `datetime.timestamp` 객체를 반환한다.

setDecimalSeparator

데이터베이스로부터 얻은 NUMERIC 타입의 소수점 구분 문자를 설정한다. 기본값은 `.`를 사용한다.

getDecimalSeparator

설정된 NUMERIC 타입의 소수점 구분 문자를 얻는다.

Connection

데이터베이스와 연결을 관리하는 객체이며 `pygoldilocks` 모듈의 `connect()` 함수로 생성된다.

속성

- `autocommit`
 - `Connection`의 `autocommit` 모드를 설정할 수 있다.

- `searchescape`
 - ODBC의 Escape 문자를 얻는다. `pygoldilocks`는 '/'를 사용한다.
- `timeout`
 - `SQL_ATTR_QUERY_TIMEOUT`을 `SQLSetConnectAttr` 함수를 이용하여 설정한다.

함수

- `cursor()`
 - 새로운 `cursor` 객체를 반환한다.
- `commit()`
 - 실행했던 SQL 구문을 `commit` 한다.
- `rollback()`
 - 실행했던 SQL 구문을 `rollback` 한다.
- `close()`
 - 연결을 닫는다. `autocommit`이 `false`인 경우, `commit` 되지 않았던 SQL 구문이 `rollback` 된다.
- `getinfo( info )`
 - ODBC의 `SQLGetInfo` 함수를 이용하여 연결 관련 속성을 얻을 수 있다. 자세한 내용은 `SQLGetInfo` 를 참조한다.

```
dns_name = cnxn.getinfo( pygoldilocks.SQL_DATA_SOURCE_NAME )
```

- `execute( sql, [*params] )`
 - 새 `cursor` 객체를 생성하고 이 객체의 `execute` 함수를 실행한 후 `cursor` 객체를 반환한다.

```
cursor = cnxn.execute( "SELECT COUNT(*) FROM EMP" )
```

자세한 내용은 `Cursor.execute()` 함수를 참조한다. 이 함수는 Python API에는 없지만 편의를 위해 제공된다. 이 함수가 호출될 때마다 `cursor` 객체가 할당되기 때문에 하나 이상의 SQL 구문을 실행해야 하는 경우에는 사용을 권장하지 않는다.

- `set_attr( attr_id, value )`
 - `SQLSetConnectAttr` 함수를 실행하여 연결 속성을 설정할 수 있다.
 - 다음은 `set_attr` 함수를 사용하여 데이터베이스의 트랜잭션 격리 단계를 조절하는 예이다.

```
connection.set_attr( pygoldilocks.SQL_ATTR_TXN_ISOLATION, pygoldilocks.SQL_TXN_SERIALIZABLE )
```

Cursor

일반적으로 cursor 객체는 fetch 작업을 관리하는데 사용되는 데이터베이스 cursor를 의미한다. 데이터베이스 cursor는 ODBC statement handle (HSTMT)에 매핑된다. 동일한 connection이 생성한 cursor 객체는 서로 분리되지 않는다. 즉, 하나의 cursor가 데이터베이스에 수행한 모든 갱신 사항이 다른 cursor에도 적용된다.

노트

Cursor는 데이터베이스 트랜잭션을 관리하지 않고 connection이 트랜잭션을 commit 하거나 rollback 한다.

속성

Description

읽기 전용 속성이며 튜플 타입으로 마지막에 수행된 SELECT 구문이 반환한 각 column에 대한 내용이 들어있다. 각 튜플은 다음을 포함한다.

1. Column name (또는 alias)
2. Type code
3. Display size
4. Internal size
5. Precision
6. Scale
7. Nullable

SELECT 구문이 호출되지 않은 경우, descriptin은 none이다.

rowcount

마지막에 수행한 SQL 구문이 갱신한 row의 개수이다.

arraysize

`fetchmany( [size = cursor.arraysize] )` 함수를 사용하여 한 번에 가져올 수 있는 row의 개수이다. 기본값은 1이다.

connection

읽기 전용 속성이므로 해당 cursor 객체를 생성한 connection 객체를 가리킨다.

fast_executemany

True로 설정되면 `executemany( sql, [*params] )` 함수를 실행할 때 매개 변수를 배열로 구성하여 한 번의 `execute`로 처리한다. False로 설정되면 매개 변수마다 개별적으로 `execute`를 실행한다.

함수

execute(sql, [*params])

SQLPrepare와 SQLExecute 함수를 통해 SQL 구문을 수행하고 이 함수를 호출한 `cursor`를 반환한다. 옵션인 매개 변수는 다음과 같이 사용할 수 있다.

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", x, y )
```

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", (x, y) )
```

executemany(sql, [*params])

각 매개 변수에 대한 SQL 구문을 실행하고 `none`을 반환한다. 매개 변수 `params`는 반드시 `sequence`의 `sequence` 타입이거나 `sequence generator`이어야 한다.

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

위 예에서 SQL 구문은 두 번 수행된다. 즉, (1, 'A')와 (2, 'B')에 대해 각각 실행된다. `Cursor` 객체의 `fast_executemany`가 `true`로 설정되었는지 또는 `false`로 설정되었는지에 따라 `executemany` 동작이 달라진다.

위의 예는 다음과 동일하다.

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
for p in params:
    cursor.execute( "INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", p )
```

`fast_executemany`를 `true`로 설정하면 `executemany`는 한 번의 `execute`만으로 작업을 처리하려 한다. 이를 위해서는 매개 변수 `params`의 아이템에서 같은 인덱스 위치에 있는 데이터의 데이터 타입이 서로 동일해야 한다.

```
params = [ ( 1, 'A' ), ( '2', 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

위의 예에서 매개 변수 `params`의 두 아이템 중 첫 번째 아이템의 데이터 타입이 다르다. 이처럼 아이템들 간에 동일한 인덱스 위치의 데이터 타입이 다르면 `executemany`가 SQL 구문을 한 번에 처리하지 않고 개별적으로 처리한다.

`Connection` 객체의 `autocommit`이 `true`일 경우, SQL 구문이 분할 처리되어 각각의 SQL 구문이 개별적으로 `commit`된다. 레코드를 순차적으로 처리하면서 오류가 발생하면 일부 레코드만 데이터베이스에 `commit`되고 일부는 `com`

mit 되지 못한 상태로 작업이 완료된다. 따라서 executemany()를 이용할 경우, 모든 레코드가 데이터베이스에 commit 되었는지 확인하기 위해 autocommit을 false로 먼저 설정하여 실행하는 것이 좋다.

fetchone()

질의의 다음 row를 반환한다. 다음 데이터가 없을 경우에는 none이다.

fetchall()

질의에 남아 있는 모든 row를 반환한다. 모든 row를 메모리로 읽어들이기 때문에 사용 시 주의해야 한다.

fetchmany([size = cursor.arraysize])

size나 cursor.arraysize 만큼 남아있는 row를 반환한다. 다음 데이터는 빈 sequence 데이터를 반환한다. cursor.arraysize의 기본값은 1이다.

commit()

SQL 구문을 commit 한다. Cursor 객체를 생성한 connection 객체가 실행하는 함수로써 동일한 connection 객체에 생성된 모든 cursor에 적용된다. Connection 객체의 commit과 동일하다.

rollback()

SQL 구문을 rollback 한다. Cursor 객체를 생성한 connection 객체가 실행하는 함수로써 동일한 connection 객체에 생성된 모든 cursor에 적용된다. Connection 객체의 rollback과 동일하다.

skip(count)

SQLFetchScroll과 SQL_FETCH_NEXT를 통해 count에 설정된 횟수만큼 레코드를 통과한다.

nextset()

GOLDILOCKS ODBC에서 SQLMoreResults를 지원하지 않으므로 false를 반환한다.

close()

Cursor 객체를 닫는다.

setinputsizes(size_list)

선택적 함수로써 sequence 타입을 매개 변수로 받는다. SQLBindParameter의 INPUT 매개 변수 크기를 설정한다.

setoutputsize(size)

선택적 함수로써 DB API와 다른 용도로 사용되어 OUTPUT 매개 변수에 대한 버퍼 크기를 할당한다.

callproc(procname [, params])

procname에 해당하는 저장 프로시저를 호출한다. 매개 변수는 sequence 타입이어야 하며 출력 매개 변수를 포함한다. 단, 입력할 때 출력 매개 변수에 위치된 데이터는 무의미하다. callproc 함수는 입력 매개 변수 데이터의 INOUT, OUT에 해당하는 데이터를 갱신하여 sequence 타입으로 반환한다.

```
create_proc = """CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 OUT CHAR(10) )
IS
    V1 CHAR(10);
BEGIN
    SELECT T1.I1
        INTO V1
        FROM T1
        WHERE T1.I1 >= A1 AND T1.I1 <= A1;
    A2 := V1;
END;\
"""

cursor.execute( create_proc )
result = cursor.callproc( 'PROC1', ( 1, 0 ) )
```

callfunc(funcname [, params])

funcname에 해당하는 함수를 호출한다. callfunc()은 함수의 데이터를 반환한다.

```
create_func = """
CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
RETURN INTEGER
IS
    V1 INTEGER;
BEGIN
    SELECT COUNT(*)
        INTO V1
        FROM T1
        WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    RETURN V1;
END;\
"""

cursor.execute( create_func )
cursor.commit()
result = cussr.callfunc( 'FUNC1', ( 1, 4 ) )
```

tables(table=None, catalog=None, schema=None, tableType=None)

지정된 조건을 만족하는 데이터베이스의 테이블 정보를 반환한다. 문자 '_'와 '%'는 와일드 카드로 해석된다. 각 row는 다음의 column 정보를 가지는데 자세한 내용은 `SQLTables`를 참조한다.

1. table_cat: 카탈로그 이름이다.
2. table_schem: 스키마 이름이다.
3. table_name: 테이블 이름이다.
4. table_type: 'TABLE', 'VIEW', 'SYSTEM TABLE', 'GLOBAL TEMPORARY', 'LOCAL TEMPORARY', 'ALIAS', 'SYNONYM' 또는 특정 타입의 이름이 올 수 있다.
5. remarks: 테이블 명세이다.

```
print cursor.tables( table= 'TEST' ).fetchone()
#print table name
for row in cursor.tables():
    print row.table_name
```

노트

매개 변수가 비어있을 경우, 사용자에게 권한이 있는 모든 테이블의 정보를 반환한다.

columns(table=None, catalog=None, schema=None, tableType=None)

`SQLColumns` 함수를 통해 지정된 테이블의 column 정보를 얻는다. 각 row는 다음과 같은 column 정보를 포함한다.

1. table_cat
2. table_schem
3. table_name
4. column_name
5. data_type
6. type_name
7. column_size
8. buffer_length
9. decimal_digits
10. num_prec_radix
11. nullable
12. remarks
13. column_def
14. sql_data_type

15. sql_datetime_sub
16. char_octet_length
17. ordinal_position
18. is_nullable: SQL_NULLABLE, SQL_NO_NULLS 또는 SQL_NULLS_UNKNOWN.

```
#print column name of table TEST
for r in cursor.columns( table = 'TEST' ):
    print r.column_name
```

statistics(table, catalog=None, schema=None, unique=False, quick=True)

SQLStatistics 함수를 통해 지정된 테이블과 관련된 정보를 얻는다.

unique가 true일 경우, unique 인덱스만 반환하고, false일 경우, 모든 인덱스를 반환한다.

quick이 true일 경우, CARDINALITY와 PAGES는 즉시 사용할 수 있는 경우에만 반환되고 그렇지 않으면 해당 열에는 NULL이 반환된다.

1. table_cat
2. table_schem
3. table_name
4. non_unique
5. index_qualifier
6. index_name
7. type
8. ordinal_position
9. column_name
10. asc_or_desc
11. cardinality
12. pages
13. filter_condition

노트

와일드카드 문자는 허용되지 않는다.

rowIdColumns(table, catalog=None, schema=None, nullable=True)

SQL_BEST_ROWID로 **SQLSpecialColumns**를 실행하여 row를 고유하게 식별하는 column의 결과 집합을 반환한다. 각 row는 다음과 같은 column 정보를 갖는다.

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, 또는 SQL_SCOPE_SESSION
2. column_name

3. data_type: ODBC의 SQL 타입 상수
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO 또는 SQL_PC_PSEUDO

rowVerColumns(table, catalog=None, schema=None, nullable=True)

SQL_ROWVER으로 **SQLSpecialColumns**를 실행하여 row가 업데이트 될 때 자동으로 업데이트 되는 column의 결과 집합을 반환한다. 각 row는 다음과 같은 column 정보를 갖는다.

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, 또는 SQL_SCOPE_SESSION
2. column_name
3. data_type: ODBC의 SQL 타입 상수
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO 또는, SQL_PC_PSEUDO

primaryKeys(table, catalog=None, schema=None)

SQLPrimaryKeys 함수를 실행하여 테이블의 주요 키를 구성하는 column의 결과 집합을 반환한다. 각 row는 다음과 같은 column 정보를 갖는다.

1. table_cat
2. table_schem
3. table_name
4. column_name
5. key_seq
6. pk_name

foreignKeys(table=None, catalog=None, schema=None, foreignTable=None, foreignCatalog=None, foreignSchema=None)

SQLForeignKeys 함수를 실행하여 지정된 테이블 또는 지정된 테이블의 기본 키를 참조하는 다른 테이블의 외래 키인 column 이름의 결과 집합을 만든다. 각 row는 다음과 같은 column 정보를 갖는다.

1. pktable_cat
2. pktable_schem
3. pktable_name
4. pkcolumn_name

5. fktable_cat
6. fktable_schem
7. fktable_name
8. fkcolumn_name
9. key_seq
10. update_rule
11. delete_rule
12. fk_name
13. pk_name
14. deferrability

procedures(procedure=None, catalog=None, schema=None)

SQLProcedures를 실행하여 프로시저에 대한 정보의 결과 집합을 만든다. 각 row는 다음과 같은 column 정보를 갖는다.

1. procedure_cat
2. procedure_schem
3. procedure_name
4. num_input_params
5. num_output_params
6. num_result_sets
7. remarks
8. procedure_type

getTypeInfo(sqlType=None)

SQLGetTypeInfo 함수를 실행하여 지정된 데이터 타입 또는 GOLDILOCKS ODBC가 지원하는 모든 데이터 타입에 대한 정보의 결과 집합을 만든다. 각 row는 다음과 같은 column을 갖는다.

1. type_name
2. data_type
3. column_size
4. literal_prefix
5. literal_suffix
6. create_params
7. nullable
8. case_sensitive
9. searchable
10. unsigned_attribute
11. fixed_prec_scale
12. auto_unique_value
13. local_type_name

14. minimum_scale
15. maximum_scale
16. sql_data_type
17. sql_datetime_sub
18. num_prec_radix
19. interval_precision

Row

Row 객체는 cursor 객체의 fetch 함수로 반환된다. DB API에 명시된 것처럼 튜플 타입처럼 처리된다.

```
row = cursor.fetchone()
for column in row:
    print column
```

다음과 같은 기능이 pygoldilocks에 추가되었다.

- Column 이름을 사용하여 데이터에 접근할 수 있다.
- Cursor 객체가 닫힌 후에도 row를 통해 cursor.description 값에 접근할 수 있다.
- Row의 값을 갱신할 수 있다.

Column 이름을 사용하여 row에 접근하면 편리할 뿐만 아니라 가독성도 높아진다. 그러나 column 이름에 Python 예약어나 공백이 포함되면 row.__getattr__()를 통해 접근해야만 한다.

```
cursor.execute( "select c1 from test")
print cursor.description
row = cursor.fetchone()
print row.C1
```

```
(( 'C1', <type 'str'>, 10, 10, 10, 0, True),)
test
```

노트

GOLDILOCKS 데이터베이스의 식별자는 기본적으로 대문자이다. 하지만 소문자로 지정하는 경우도 있기 때문에 column 이름을 이용하여 row에 접근할 때는 대소문자 사용에 유의해야 한다.

속성

- `cursor_description`

해당 row를 생성한 cursor 객체의 속성 `description`의 복사본이다. 자세한 내용은 **`Cursor.description`**을 참조한다.

29.3 Exception

Python 예외는 GOLDILOCKS ODBC에서 오류를 감지했을 때 pygoldilocks에 의해 발생한다. 예외 클래스는 다음과 같이 **Python DB API**와 동일하다.

- Error
 - DatabaseError
 - DataError
 - OperationalError
 - IntegrityError
 - InternalError
 - ProgrammingError
 - NotSupportedError

오류가 발생할 경우, 일반적으로 예외 유형은 데이터베이스에서 제공하는 SQLSTATE 값을 기반으로 처리된다.

SQLSTATE	Exception
0A000	NotSupportedError
01002	OperationalError
08001	OperationalError
08003	OperationalError
08004	OperationalError
08007	OperationalError
08S01	OperationalError
28000	InterfaceError
40002	IntegrityError
22***	DataError
23***	IntegrityError
24***	ProgrammingError
25***	ProgrammingError
42***	ProgrammingError

29.4 Data Type

Python 매개 변수를 GOLDBLOCKS로 전달

Python 매개 변수를 GOLDBLOCKS ODBC에 전달할 때는 다음과 같이 데이터가 변환된다.

표 29-1 Python 3

Python datatype	설명	ODBC datatype
None	-	SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytes, bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

표 29-2 Python 2

Python datatype	설명	ODBC datatype
None	-	SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
buffer	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	32 bit: SQL_INTEGER, 64 bit: SQL_BIGINT
long	bigint	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

GOLDILOCKS로부터 전달받는 SQL 값

GOLDILOCKS 데이터베이스의 데이터를 Python으로 전달할 때는 다음과 같이 데이터가 변환된다.

표 29-3 Python 3

ODBC datatype	설명	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

표 29-4 Python 2

ODBC datatype	설명	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

Python 데이터 타입의 text는 Python 3에서는 unicode로 변환된다. Python 2에서는 데이터베이스의 character set에 따라 unicode나 string으로 변환된다.

表 29-5 Python 2 text

DB character set	Python type
UTF-8	str
SQL_ASCII	str
UHC	unicode
GB18030	unicode

30.

Hibernate

30.1 개요

Hibernate는 Java Persistent API (JPA) 모델을 데이터베이스에 적용한 Object-Relational Mapping (ORM) 프레임워크이다. DB 테이블과 Java 객체와의 관계를 매핑시켜 지속성 있는 로직 (persistent logic) 처리를 도와준다. 즉, Hibernate를 이용하면 SQL을 이용하지 않고도 DB에 쉽게 접근하고 제어할 수 있다.

30.2 연동

다운로드

Hibernate는 <https://sourceforge.net/projects/hibernate/> 에서 다운로드 받을 수 있다. 다운로드 받은 파일의 압축을 풀면 lib 디렉토리에 다수의 jar 파일들이 있다. 이 파일들을 이용하여 Hibernate와 연동할 수 있다.

노트

GOLDILOCKS의 Hibernate는 Hibernate 5.2 버전을 기준으로 작성되었다.

GoldilocksDialect 클래스 연동

Hibernate를 연동하려면 다운로드 받은 Hibernate jar 파일에 이식해야 한다. 다음과 같이 GoldilocksDialect.class 와 GoldilocksDialect\$1.class 파일을 Hibernate jar 파일에 포함시킬 수 있다.

1. Hibernate jar 파일의 압축을 푼다.
2. 압축이 풀린 org/hibernate/dialect 디렉토리로 GoldilocksDialect.class와 GoldilocksDialect\$1.class 파일을 이동시킨다.
3. Hibernate jar 파일로 다시 압축한다.
4. 새로 생성한 jar 파일을 classpath에 추가한다.

```
$ jar -xvf hibernate-core-5.2.11.Final.jar
$ pwd
hibernate-release-5.2.11.Final/lib/required/
$ mv GoldilocksDialect*.class org/hibernate/dialect/
```



```
$ jar -cvf hibernate.5.2.11.jar META-INF/MANIFEST.MF .
```

30.3 사용 예

설정

Hibernate를 DB와 연동하려면 hibernate-configuration XML 파일과 hibernate-mapping XML 파일이 필요하다.

Hibernate Configuration 파일

Configuration XML 파일은 dialect 클래스와 DB에 접속하기 위한 연결 정보 및 Hibernate-mapping 파일들을 설정하는 파일이다.

자세한 내용은 **Configuration**을 참조한다.

다음은 hibernate.conf.xml 파일의 예이다.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">sunje.goldilocks.jdbc.
GoldilocksDriver</property>
    <property name="hibernate.connection.url">jdbc:goldilocks://192.168.0.21:22581/test</
property>
    <property name="hibernate.connection.username">test</property>
    <property name="hibernate.connection.password">test</property>
    <property name="hibernate.dialect">org.hibernate.dialect.GoldilocksDialect</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="hbm2ddl.auto"> create </property>
    <mapping resource="SampleTable.mapping.xml" />
  </session-factory>
</hibernate-configuration>
```

<session-factory> 태그에 DB 연결 정보를 설정하고 hibernate.dialect 속성에 연결하려는 DB의 dialect 클래스 이름을 명시한다. 또한 <mapping-resource> 태그를 이용하여 hibernate-mapping 파일들을 명시한다.

Hibernate Mapping 파일

Mapping XML은 DB 테이블과 Java object의 매핑 정보를 담고 있는 설정 파일이다.

자세한 내용은 **Mapping**을 참조한다.

다음은 위 예에서 사용한 SampleTable.mapping.xml 파일의 예이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="SampleTable" table="SAMPLETABLE">
    <id column="ID" name="id" type="long">
      <generator class="increment" />
    </id>
    <property column="STUDENT_NAME" name="name" type="string" />
    <property column="BIGINT_VALUE" name="bigintValue" type="long" />
    <property column="INT_VALUE" name="intValue" type="int" />
    <property column="SMALL_VALUE" name="smallValue" type="short" />
    <property column="FLOAT_VALUE" name="floatValue" type="float" />
    <property column="DOUBLE_VALUE" name="doubleValue" type="double" />
    <property column="NUMERIC_VALUE" name="numericValue" type="big_integer" precision="10"
scale="5" />
    <property column="DATE_VALUE" name="dateValue" type="date" />
    <property column="TIME_VALUE" name="timeValue" type="time" />
    <property column="TIMESTAMP_VALUE" name="timestampValue" type="timestamp" />
    <property column="VARBINARY_VALUE" name="binaryValue" type="byte[]" />
  </class>
</hibernate-mapping>
```

<class> 태그에는 Java 클래스와 DB 테이블의 매핑 정보를 기술한다. <property> 태그에는 테이블의 column과 Java에서 사용하는 데이터 타입을 지정한다.

Application 예제

DB 테이블에 매핑되는 Java 클래스를 기술해야 한다. 다음은 위에서 설명한 SampleTable.java의 예이다.

```
import java.math.BigInteger;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
public class SampleTable {
    private long id;
    private String    name;
    private long      bigintValue;
    private int       intValue;
    private short     smallValue;
    private float     floatValue;
    private double    doubleValue;
    private BigInteger numericValue;
    private Date      dateValue;
    private Time      timeValue;
    private Timestamp timestampValue;
    private byte[]    binaryValue;
    private String    longvarcharValue;
    private byte[]    longvarbinaryValue;

    public SampleTable() {
        this.name = null;
    }

    public SampleTable( String name ) {
        this.name = name;
    }

    public long getId() {
        return id;
    }

    public void setId( long id ) {
        this.id = id;
    }

    public String getName() {
```

```

        return name;
    }

    public void setName( String name ) {
        this.name = name;
    }
    ....

```

SampleTable 테이블의 column에 대응하는 변수의 get/set 함수를 작성해야 한다.

SamleTable 테이블에서는 다음과 같이 데이터를 처리한다.

1. configuraion 파일을 통해 org.hibernate.SessionFactory 객체를 얻어온다.
2. SessionFactory 객체를 통해 org.hibernate.Session 객체를 얻어온다.
3. Session 객체의 method를 호출하여 원하는 작업을 한다.

다음은 DB의 SampleTable 테이블에 데이터를 추가, 갱신, 조회, 삭제하는 응용 프로그램의 예이다. org.hibernate.Configuration 클래스를 통해서 SessionFactory 객체를 얻는다.

```

public class HibernateSample
{
    private static SessionFactory mFactory;
    public static void main( String[] args ) {
        try {
            mFactory = new Configuration().configure( "hibernate.conf.xml" ).
buildSessionFactory();
        } catch( Throwable ex ) {
            System.err.println( "Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }
        HibernateSample sHibernateSample = new HibernateSample();

```

- Insert 3 records

```

Long sRecord1 = sHibernateSample.addSample( "RECORD_1" );
Long sRecord2 = sHibernateSample.addSample( "RECORD_2" );
Long sRecord3 = sHibernateSample.addSample( "RECORD_3" );

```

- Update record 1

```

sHibernateSample.updateSample( sRecord1, ....

```

- Fetch table

```
sHibernateSample.fetchSample( 0, 3 );
```

- Delete record 2

```
sHibernateSample.deleteSample( sRecord2 );
sHibernateSample.closeFactory();
    }
public void closeFactory() {
    mFactory.close();
}
....
```

다음은 HibernateSample 클래스에 레코드를 삽입하는 method의 예이다. 먼저 SessionFactory 클래스를 통해 session 객체를 얻은 후 org.hibernate.Transaction 객체를 얻는다. Transaction 클래스를 이용하여 transaction을 시작하고 레코드를 삽입한 후에 commit 하거나 rollback 할 수 있다.

```
public Long addSample( String name ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    Long id = null;

    try{
```

- Begin transaction

```
tx = (Transaction) session.beginTransaction();
SampleTable sample = new SampleTable( name );
```

- Insert

```
id = (Long) session.save(sample);
```

- Commit transaction

```
tx.commit();

    System.out.println("add success");
} catch( HibernateException e) {
    if( tx != null ) {
        tx.rollback();
    }
    System.out.println("add fail");
```

```

        e.printStackTrace();
    } finally {
        session.close();
    }
    return id;
}

```

다음은 SampleTable 테이블을 조회하는 method의 예이다. Session 클래스를 통해 org.hibernate.Query 객체를 얻는다. fetchSample method는 SampleTable을 offset부터 count 만큼 조회한다.

```

public void fetchSample( int offset, int count ) {
    Session session = mFactory.openSession();
    Query query = null;

    try{
        query = session.createQuery( "FROM SampleTable" );
        query.setFirstResult( offset );
        query.setMaxResults( count );
        query.setFetchSize( 20 );

        @SuppressWarnings( "unchecked" )
        List<SampleTable> samples = (List<SampleTable>) query.list();
        for( Iterator<SampleTable> iter = samples.iterator(); iter.hasNext(); ) {
            SampleTable sample = (SampleTable) iter.next();
            System.out.print( "id: " + sample.getId() );
            ... 중략 ...
        }
    } catch ( HibernateException e ) {
        e.printStackTrace();
    } finally {
        session.close();
    }
}

```

다음은 SampleTable 테이블을 갱신하는 method의 예이다. SampleTable 클래스에 작성된 set method로 변경한 후에 session 클래스의 update method를 호출하여 갱신한다.

```

public void updateSample( long id, ... ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    try {
        tx = session.beginTransaction();

```

```

        SampleTable sample = (SampleTable)session.get(SampleTable.class, id);
        sample.setName( name );
        ...
        session.update( sample );
        tx.commit();
        System.out.println( "update success" );
    } catch ( HibernateException e ) {
        if( tx != null ){
            tx.rollback();
        }
        System.out.println("update fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
}

```

다음은 레코드를 삭제하는 예이다. Session 클래스의 delete method를 이용해서 삭제할 수 있다.

```

public void deleteSample( long id ) {
    Session session = mFactory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        SampleTable sample = (SampleTable) session.get( SampleTable.class, id );
        session.delete( sample );
        tx.commit();

        System.out.println( "delete success" );
    } catch ( HibernateException e ) {
        if( tx != null ) {
            tx.rollback();
        }
        System.out.println("delete fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
}

```


파트 VI.

Utility Manual

31. gcreatedb	3,199
31.1 gcreatedb 소개	3,200
정의	3,200
기능	3,200
사용법	3,200
Option	3,200
사용 예	3,201
31.2 Command Option	3,202
--cluster	3,202
--db_name	3,202
--db_comment	3,202
--timezone	3,202
--character_set	3,202
--char_length_units	3,203
--home	3,203
--member	3,204
--host	3,204
--port	3,204
--silent	3,204
--help	3,204
32. glsnr	3,207
32.1 glsnr 소개	3,208
32.2 Command Option	3,209
--silent	3,209
--start	3,209
--stop	3,209
--status	3,210
--home	3,210
--help	3,211
32.3 Listener Configuration	3,212
Configuration File 및 환경 변수	3,212
LISTEN_PORT	3,212
TCP_HOST	3,213
BACKLOG	3,213
DEFAULT_CS_MODE	3,213
TCP_VALIDNODE_CHECKING	3,214
TCP_INVITED_FILE	3,214
TCP_EXCLUDED_FILE	3,215
TIMEOUT	3,215

LISTENER_LOG_DIR	3,215
UDS_DIR	3,216
33. gsql/gsqlnet (Interactive SQL Tool)	3,217
33.1 gsql 개요	3,218
정의	3,218
사용 예	3,218
33.2 gsql 실행	3,222
gsql 정보	3,222
서버 접속	3,223
대화형 모드 제어	3,224
gsql 설정 저장	3,225
33.3 Interactive Command 사용	3,228
gsql 대화형 모드 명령어	3,228
서버 구동 및 종료	3,230
SQL 구문 수행	3,234
출력 결과 제어	3,243
출력 결과 저장	3,249
SQL 객체 정보 조회	3,250
SQL 객체의 DDL 구문 출력	3,252
History 제어	3,254
SQL 편집	3,255
연결 제어	3,257
호스트 변수 사용	3,257
SQL 구문 처리 방식 제어	3,258
SQL 실행 계획 정보	3,263
33.4 Command Option References	3,265
Username Password	3,265
--as {SYSDBA ADMIN}	3,265
--conn-string	3,266
--dsn	3,266
--enable-color	3,267
--help	3,268
--import	3,269
--no-prompt	3,270
--prompt	3,270
--silent	3,271
--version	3,271
33.5 Interactive Command References	3,273
\\	3,273

\connect	3,274
\cstartup	3,275
\cshutdown	3,276
\ddl_cluster	3,277
\ddl_db	3,278
\ddl_tablespace	3,282
\ddl_profile	3,283
\ddl_audit_policy	3,284
\ddl_auth	3,285
\ddl_schema	3,288
\ddl_publicsynonym	3,291
\ddl_table	3,291
\ddl_constraint	3,293
\ddl_index	3,295
\ddl_view	3,296
\ddl_sequence	3,297
\ddl_synonym	3,298
\ddl_procedure	3,299
\desc	3,299
\dynamic sql :var	3,302
\edit	3,303
\exec	3,307
\exec :var := value	3,308
\exec sql	3,312
\explain plan	3,313
\help	3,314
\host	3,316
\history	3,316
\import	3,317
\idesc	3,319
\{n}	3,320
\prepare sql	3,321
\print	3,322
\quit	3,324
\set autocommit	3,324
\set autotrace	3,326
\set color	3,327
\set colsize	3,328
\set ddlsize	3,330

\set error	3,331
\set heading	3,332
\set history	3,333
\set linesize	3,334
\set numsize	3,336
\set pagesize	3,336
\set serveroutput	3,338
\set time	3,340
\set timing	3,340
\set vertical	3,341
\shutdown	3,343
\spool	3,344
\startup	3,346
\var	3,347
34. gloder/gloadernet (Upload/download Tool)	3,351
34.1 gloder 개요	3,352
Environment	3,352
사용 예	3,353
34.2 gloder 사용	3,356
Datafile Type	3,356
데이터 다운로드	3,356
데이터 업로드	3,364
Upload 문제 해결	3,373
34.3 Control File 구문	3,379
CHARACTERSET	3,379
TABLE	3,380
FIELDS TERMINATED BY	3,381
OPTIONALLY ENCLOSED BY	3,381
LINES TERMINATED BY	3,382
LTRIM	3,383
RTRIM	3,384
WHERE	3,385
34.4 gloder Argument References	3,386
사용법	3,386
필수 인자	3,387
옵션 인자	3,388
35. gdump	3,399
35.1 gdump 개요	3,400
정의	3,400

인자	3,400
35.2 사용 예	3,403
Control File	3,403
Datafile	3,405
Log File	3,407
Incremental Backup File	3,409
Property File	3,412
Commit Log	3,413
Log Buffer File	3,413
Pending Log Buffer File	3,414
36. tablediff	3,417
36.1 tablediff 개요	3,418
배경	3,418
기능	3,418
특징	3,419
파일 구성	3,419
36.2 사용법	3,420
커맨드 사용법	3,420
Property Option	3,421
37. gsyncher	3,427
37.1 gsyncher 소개	3,428
정의	3,428
특징	3,428
사용법	3,428
Option	3,428
37.2 사용 예	3,429
38. gmon	3,433
38.1 gmon 소개	3,434
정의	3,434
기능	3,434
사용법	3,434
Option	3,434
38.2 사용 예	3,435
39. gtrclogger	3,437
39.1 gtrclogger 소개	3,438
정의	3,438
특징	3,438
사용법	3,438

Options	3,439
39.2 사용 예	3,440
40. glocator	3,441
40.1 glocator 소개	3,442
정의	3,442
사용법	3,442
Option	3,442
40.2 glocator 사용	3,447
데이터 파일	3,447
CSTARTUP과 CSHUTDOWN	3,447
다중화	3,448
40.3 기능	3,450
Connection Service	3,450
Cluster Failover	3,451
40.4 glocator Configuration	3,452
Configuration File 및 환경 변수	3,452
Configuration Property	3,452
41. gagent	3,459
41.1 gagent 소개	3,460
정의	3,460
사용법	3,460
Option	3,460
41.2 gagent Configuration	3,464
Configuration File 및 환경 변수	3,464
Configuration Property	3,464
42. gloctl	3,469
42.1 gloctl 소개	3,470
정의	3,470
사용법	3,470
Option	3,470
42.2 Interactive Command Reference	3,474
ADD MEMBER	3,474
ADD SERVICE	3,475
DROP MEMBER	3,476
DROP SERVICE	3,476
EXPORT	3,477
HELP	3,478
IMPORT	3,478

QUIT	3,479
SET TIMEOUT	3,479
42.3 Location File	3,481
설명	3,481
42.4 Configuration	3,483
PORT	3,483
LOCATOR_HOST	3,483
LOCATOR_PORT	3,483
MESSAGE_TIMEOUT	3,484

31.

gcreatedb

31.1 gcreatedb 소개

정의

gcreatedb는 GOLDDILOCKS에서 제공하는 데이터베이스 생성 유틸리티이다.

기능

사용자가 지정한 DB-name, comment, time zone, character set으로 새로운 데이터베이스를 생성한다.

사용법

```
gcreatedb [options]
```

Option

--cluster	cluster system (if not specified, stand-alone system)
--db_name	database name
--db_comment	database comment
--timezone	timezone ({+/-}{TZh:TzM})
--character_set	character set
	SQL_ASCII
	UTF8
	UHC
	GB18030
--char_length_units	char length units
	OCTETS
	CHARACTERS
--home	home directory
--member	local member name
--host	host address
--port	host port
--silent	supprsses the display of the result message

`--help` print help message

사용 예

```
$ gcreatedb --db_name="goldilocks" --db-comment="goldilocks database" --timezone="+09:00" --  
character_set="UTF8" --silent
```

31.2 Command Option

--cluster

Cluster 환경의 database를 생성한다.

이를 사용하지 않을 경우 standalone database를 생성한다.

--db_name

사용자가 생성하려는 database name을 지정한다.

Default value는 "goldilocks"이다.

Database name의 최대 길이는 128 이다.

--db_comment

사용자가 생성하려는 database comment를 지정한다.

Default value는 "goldilocks database" 이다.

Database comment의 최대 길이는 1024 이다.

--timezone

Database의 time zone 값이다.

Default value는 property의 **TIMEZONE** 값이다.

Database를 생성할 때 적용되는 속성으로써 '-14:00' ~ '+14:00' 범위의 값을 사용할 수 있다.

--character_set

Database의 character set이다.

Default value는 property의 **CHARACTER_SET** 값이다.

Database를 생성할 때 적용되는 속성으로써 다음 중 하나의 값으로 설정할 수 있다.

- GB18030
- SQL_ASCII

- UHC
- UTF8

--char_length_units

CHAR, VARCHAR와 같은 문자열 column을 정의하면서 다음과 같이 char length unit을 생략할 경우에 사용되는 char length units 값이다.

```
CREATE TABLE t1
(
  id CHAR( 10 OCTETS ),           ❶ 10 bytes를 의미한다.
  name VARCHAR( 128 CHARACTERS ), ❷ 128 글자를 의미한다.
  addr VARCHAR( 1024 )           ❸ char length unit이 생략되었다.
);
```

Default value는 property의 **CHAR_LENGTH_UNITS** 값이다.

Database를 생성할 때 적용되는 속성으로써 다음 중 하나의 값으로 설정할 수 있다.

- OCTETS: Byte 수를 의미한다.
- CHARACTERS: 문자 개수를 의미한다.

SQL 표준은 기본값을 CHARACTERS로 정의하고 있으며 다른 DBMS들의 char length unit 기본값은 다음과 같다.

노트

- Oracle과 DB2에서는 OCTETS를 사용한다.
- MS-SQL, MySQL, PostgreSQL에서는 CHARACTERS를 사용한다.

--home

Database의 home directory를 설정한다.

Default value는 \$GOLDILOCKS_HOME 환경 변수를 사용한다.

--member

Cluster 환경에서 local database의 이름을 설정한다.

--host

Cluster 환경에서 local database의 host address를 설정한다.

--port

Cluster 환경에서 local database의 port 번호를 설정한다.

--silent

결과 메시지를 출력하지 않는다.

--help

Help 메시지를 출력한다.

```
$ gcreatedb --help
Usage
  gcreatedb [options]
Options:
  --db_name           database name
  --db_comment        database comment
  --timezone          timezone ( {+/-}{TZH:TZM} )
  --character_set     character set
                     SQL_ASCII
                     UTF8
                     UHC
                     GB18030
  --char_length_units char length units
                     OCTETS
```

CHARACTERS

<code>--silent</code>	suppresses the display of the result message
<code>--help</code>	print help message

examples:

```
gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00" --  
character_set="UTF8" --char_length_units="OCTETS" --silent
```


32.

glsnr

32.1 glsnr 소개

glsnr은 GOLDDILOCKS가 client/ server 환경에서 network를 통해 원격으로 접속할 수 있게 해주는 listener이다. Network를 통해 GOLDDILOCKS에 접속하려면 반드시 서버 쪽에 glsnr이 구동되어 있어야 한다.

glsnr은 다음과 같이 사용한다.

```
$ glsnr [options]
```

32.2 Command Option

glsnr을 사용하기 위한 셀 프롬프트 옵션은 다음과 같다.

--silent

설명

실행에 대한 message를 출력하지 않는다.

사용 예

```
$ glsnr --start --silent
```

--start

설명

glsnr을 구동한다. 만약 이미 구동중이었으면 에러가 발생한다.

사용 예

```
$ glsnr --start  
Listener is started successfully.
```

--stop

설명

현재 구동 중인 glsnr을 멈춘다.

사용 예

```
$ glsnr --stop  
Listener is stopped.
```

--status

설명

glsnr의 상태 메시지를 출력한다.

사용 예

```
$ glsnr --status  
Listener is not running.  
$ glsnr --start  
Listener is started successfully.  
$ glsnr --status  
Listener process ID : 27880  
Listener configuration file : /home/goldilocks/goldilocks_home/conf/goldilocks.listener.conf  
Unix Domain Path : /tmp/unix-glsnr.22581  
TCP Listen Host : 0.0.0.0, Port : 22581  
default C/S mode : Dedicated  
Connection Timeout(second) : 100  
Listener is running.
```

--home

설명

db home을 설정한다.

사용 예

```
$ glsnr --start --home Gliese/home/g1n1_home  
Listener is started successfully.  
$ glsnr --status
```

```
Listener process ID : 20777
Listener configuration file : /home/goldilocks/Gliese/home/g1n1_home/conf/goldilocks.listener.
conf
Unix Domain Path : /tmp/unix-glsnr.22581
TCP Listen Host : 0.0.0.0, Port : 22581
default C/S mode : Dedicated
Connection Timeout(second) : 100
Listener is running.
```

--help

설명

Help message를 출력한다.

사용 예

```
$ glsnr --help
```

Usage:

```
glsnr [options]
```

Options:

--silent	don't print message
--start	start listener
--stop	stop listener
--status	show listener status
--help	show listner help messages

32.3 Listener Configuration

Configuration File 및 환경 변수

glsnr은 configuration을 설정하기 위해 configuration file이나 환경 변수를 사용할 수 있다.

환경 변수를 설정하려면 configuration의 property name에 prefix로 'GOLDILOCKS_'를 추가한 name을 사용한다. 예를 들어 configuration file에 LISTEN_PORT를 설정하면 \$GOLDILOCKS_LISTEN_PORT를 지정하는 것과 같은 효과가 있다.

Configuration file의 내용은 환경 변수 설정값보다 우선한다. (즉, 환경 변수는 configuration file이 설정되지 않은 경우에만 적용된다.)

glsnr의 환경 파일은 \$GOLDILOCKS_DATA/conf/goldilocks.listener.conf 파일이다. glsnr의 구동 환경을 변경하려면 해당 파일의 내용을 변경하거나 환경 변수를 설정한 후에 glsnr을 구동한다.

glsnr이 실행되고 있는 상태에서 glsnr의 환경을 바꾸어 적용하려면 glsnr을 stop 시킨 후에 configuration 파일의 내용을 수정하거나 환경 변수를 설정한 후에 다시 start 한다.

glsnr을 stop하면 이미 연결이 완료된 client에는 문제가 없고 새로 연결을 시도하는 client만 실패한다.

LISTEN_PORT

glsnr이 연결을 기다리는 port이다.

항목	설명
이름	LISTEN_PORT
설명	glsnr이 연결을 기다리는 port이다.
Data type	INT
기본값/ 범위	22581 / 1024 ~ 49151

설명

TCP 연결을 원하는 client들은 여기에 지정된 port로 접속을 시도해야 한다.

Port는 1024부터 49151까지 사용할 수 있다.

TCP_HOST

glsnr이 연결을 기다리는 NIC의 IP address이다.

항목	설명
이름	TCP_HOST
설명	glsnr이 bind 하는 IP address이다.
Data type	ip address (ip v4)
기본값	0.0.0.0

설명

TCP 연결을 원하는 client들은 여기에 지정된 IP address로 접속을 시도해야 한다. IP address는 ip v4 형식을 사용한다.

BACKLOG

Client가 동시에 접속할 경우 glsnr이 동시에 처리할 수 있는 client의 개수이다.

항목	설명
이름	BACKLOG
설명	glsnr이 연결을 기다릴 수 있는 client의 개수이다.
Data type	INT
기본값 및 범위	1024 / 1 ~ 32768

설명

이 설정값이 client의 동시 접속자 수를 보장해주지는 않는다.

DEFAULT_CS_MODE

Client에서 접속 모드를 dedicated나 shared로 선택하지 않았을 경우, 접속 모드를 설정한다.

항목	설명
이름	DEFAULT_CS_MODE
설명	Default 접속 모드를 설정한다.
Data type	String (dedicated shared)

항목	설명
기본값	dedicated

설명

- glsnr을 통해 접속하는 Client/ Server (C/S) 모델은 dedicated와 shared라는 두 가지 모드를 지원한다.
- 기본적으로 client 단계에서 dedicated나 shared를 (odbc의 경우는 .odbcini) 선택하여 접속하지만 client에서 설정이 안된 경우에는 DEFAULT_CS_MODE 설정에 따라 접속모드가 결정된다.

TCP_VALIDNODE_CHECKING

접속을 시도한 client의 유효성 검증 여부를 설정한다.

항목	설명
이름	TCP_VALIDNODE_CHECKING
설명	Client 유효성 검증 여부를 설정한다.
Data type	String (NO INVITED EXCLUDED)
기본값	NO

설명

- 이 값이 NO로 설정되면 client를 검증하지 않는다.
- 이 값이 INVITED로 설정되고 TCP_INVITED_FILE에 설정된 파일이 존재하면 TCP_INVITED_FILE에 설정된 파일의 ip address를 가진 client만 유효한 사용자로 설정된다.
- 이 값이 "EXCLUDED"로 설정되고 TCP_EXCLUDED_FILE에 설정된 파일이 존재하면 TCP_EXCLUDED_FILE에 설정된 파일의 ip address를 가진 client를 제외한 나머지 client들만 유효한 사용자로 설정된다.

TCP_INVITED_FILE

TCP_VALIDNODE_CHECKING 값이 INVITED인 경우에만 사용된다.

항목	설명
이름	TCP_INVITED_FILE
설명	유효한 사용자 (IP address) list가 있는 파일이다.
Data type	String
기본값	'goldilocks.invited.conf'

설명

여기에 설정된 파일이 존재할 경우, 파일에 포함된 사용자 (IP address)만 접속할 수 있다.

TCP_EXCLUDED_FILE

TCP_VALIDNODE_CHECKING 값이 EXCLUDED인 경우에만 사용된다.

항목	설명
이름	TCP_EXCLUDED_FILE
설명	유효하지 않은 사용자 (IP address) list가 있는 파일이다.
Data type	String
기본값	'goldilocks.excluded.conf'

설명

여기에 설정된 파일이 존재할 경우, 파일에 포함된 사용자 (IP address)를 제외한 모든 사용자가 접속할 수 있다.

TIMEOUT

glsnr의 timeout 값이며 단위는 초 (second)이다

항목	설명
이름	TIMEOUT
설명	glsnr의 timeout 이다.
Data type	INT
기본값/ 범위	100 / (0 ~ 2147483647)

설명

glsnr에서 client와 통신할 때 client로부터 응답이 없거나 반응이 느린 경우 timeout에 의해 접속이 해제된다.

LISTENER_LOG_DIR

glsnr에서 출력되는 log가 저장되는 디렉토리를 설정한다.

항목	설명
이름	LISTENER_LOG_DIR
설명	glsnr의 log가 저장되는 디렉토리를 설정한다.
Data type	String
기본값	'<GOLDILOCKS_DATA>/trc'

설명

설정값의 <GOLDILOCKS_DATA>는 환경 변수 \$GOLDILOCKS_DATA 값으로 대체된다.

UDS_DIR

glsnr에서 사용되는 Unix Domain Socket 파일이 저장되는 디렉토리를 설정한다

항목	설명
이름	UDS_DIR
설명	glsnr에서 사용되는 Unix Domain Socket 파일이 저장되는 디렉토리를 설정한다.
Data type	String
기본값/ 범위	'/tmp' / 최대 60 byte

설명

디렉토리의 최대 길이는 60 byte 이내로 설정해야 한다. Unix Domain Socket 파일의 절대 path (디렉토리 + 파일 이름)의 최대 크기는 OS마다 다르지만 보통 100 byte 내외이다.

33.

gsq/gsqInet (Interactive SQL Tool)

33.1 gsql 개요

다음은 실행파일이다.

표 33-1 실행파일 구분

실행파일 이름	구분
gsql	Direct attach (D/A) 환경에서 사용된다.
gsqlnet	Client/ server (C/S) 환경에서 사용된다.

노트

gsql에서 제공되는 모든 명령어는 gsqlnet에서도 동일하게 사용된다.

정의

gsql은 SQL 구문을 처리하기 위해 GOLDILOCKS에서 제공하는 대화형 유틸리티이다.

별도로 응용 프로그램을 작성하지 않아도 gsql 프로그램을 통해 SQL 구문을 수행할 뿐 아니라 SELECT 구문의 결과를 조회할 수도 있다.

gsql 프로그램은 SQL 구문을 수행할 뿐만 아니라 \ 로 시작하는 gsql 고유의 명령어를 통해 테이블과 인덱스 등의 간략한 객체 정보를 조회할 수 있으며 서버 구동과 종료, 출력 결과의 제어, SQL 구문의 실행 계획 정보 조회 등의 다양한 기능들을 제공한다.

사용 예

본 장에서는 gsql을 사용하여 테이블을 생성/ 제거하고 데이터를 조작하며 질의를 수행하는 예를 설명한다.

다음과 같이 gsql을 대화형 모드로 구동한다. 정상적으로 접속할 경우 *gSQL>* 프롬프트와 함께 SQL 구문이나 gsql 명령이 입력되기를 기다린다.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

다음과 같이 **CREATE TABLE** 구문을 수행하여 테이블을 생성하고 트랜잭션을 COMMIT 한다.

```
gsq!> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gsq!> COMMIT;
Commit complete.
```

생성한 테이블에 **INSERT INTO** 구문을 수행하여 데이터를 추가하고 트랜잭션을 COMMIT 한다. 두 번째 INSERT 구문은 두 개의 row를 추가하는 예이다.

```
gsq!> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- 두 개의 row가 추가된다.

```
gsq!> INSERT INTO t1 VALUES ( 2, 'mkkim' ), ( 3, 'xcom73' );
2 rows created.
gsq!> COMMIT;
Commit complete.
```

다음은 테이블에 addr column을 추가하고 **UPDATE** 구문을 수행하여 데이터를 갱신하는 예이다.

```
gsq!> ALTER TABLE t1 ADD COLUMN ( addr VARCHAR(1024) );
Table altered.
gsq!> UPDATE t1 SET addr = 'Seoul, Korea' WHERE id = 1;
1 row updated.
gsq!> UPDATE t1 SET addr = 'Inchon, Korea' WHERE id = 3;
1 row updated.
gsq!> COMMIT;
Commit complete.
```

다음과 같이 입력된 데이터를 **SELECT** 질의를 이용하여 조회한다. 질의 결과는 column 이름을 표현하는 header와 각 row를 하나의 line에 출력한 정보로 구성되어 있다. NULL 값은 소문자 null로 표현된다.

```
gsq!> SELECT * FROM t1 ORDER BY 1;
ID NAME  ADDR
-- -----
1 leekmo Seoul, Korea
2 mkkim  null
3 xcom73 Inchon, Korea
3 rows selected.
```

다음과 같이 호스트 변수를 선언하고 호스트 변수에 값을 대입하고 호스트 변수를 이용하여 질의를 수행할 수 있다.

```

gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID NAME   ADDR
--  -----
1 leekmo Seoul, Korea
1 row selected.

```

\prepare sql 명령을 이용해 SQL 구문을 준비하고 호스트 변수의 값을 바꿔가며 **\exec** 명령을 이용해 위의 SELECT 구문을 반복 수행한다. 다음과 같이 gsql 대화형 모드에서 ODBC와 JDBC의 prepare/ execute 동작을 simulation 할 수 있다.

```

gSQL> \var v_id INTEGER
gSQL> \prepare sql SELECT * FROM t1 WHERE id = :v_id;
SQL prepared.
gSQL> \exec :v_id := 2
gSQL> \exec
ID NAME   ADDR
--  -----
2 mkkim null
1 row selected.
gSQL> \exec :v_id := 3
gSQL> \exec
ID NAME   ADDR
--  -----
3 xcom73 Inchon, Korea
1 row selected.

```

다음은 embedded SQL, ODBC, JDBC에서 scrollable 커서를 선언하고 gsql을 통해 커서를 이용한 데이터 검색을 simulation 하는 예이다. **DECLARE cursor_name** 구문을 이용해 선언한 커서 cur1은 스크롤이 가능한 KEYSET 커서로써 **FETCH cursor_name** 구문을 이용해 다양한 fetch orientation을 사용할 수 있다.

```

gSQL> \var v_id INTEGER
gSQL> \var v_name VARCHAR(128)
gSQL> DECLARE cur1 KEYSET CURSOR FOR SELECT id, name FROM t1 ORDER BY id;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
----  -----

```

```

    1 leekmo
1 row fetched.
gsq!> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
----
    2 mkkim
1 row fetched.
gsq!> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
----
    3 xcom73
1 row fetched.
gsq!> FETCH cur1 INTO :v_id, :v_name;
no rows fetched.
gsq!> FETCH ABSOLUTE 2 cur1 INTO :v_id, :v_name;
V_ID V_NAME
----
    2 mkkim
1 row fetched.
gsq!> FETCH PRIOR cur1 INTO :v_id, :v_name;
V_ID V_NAME
----
    1 leekmo
1 row fetched.
gsq!> CLOSE cur1;
Cursor closed.

```

다음은 생성한 테이블 t1을 제거하고 gsql 대화형 모드를 종료하는 예이다.

```

gsq!> DROP TABLE t1;
Table dropped.
gsq!> COMMIT;
Commit complete.
gsq!> \q
%

```

33.2 gsql 실행

gsql 정보

다음은 참조 옵션이다.

표 33-2 참조 옵션

옵션	설명
--help	옵션 목록
--version	gsql 버전 정보

Shell 프롬프트 상에서 수행할 수 있는 gsql의 command option은 다음과 같이 **--help** 옵션을 통해 확인할 수 있다.

```
% gsql --help
Usage
  gsql [user_name [password]] [options]
Arguments:
  user_name      user name
  password       password
Options:
  --version              print version information and exit
  --import FILE          import sql FILE
... 중략 ...
%
```

gsql 프로그램은 GOLDILOCKS 서버와 달리 별도의 version을 가지는데 다음과 같이 **--version** 옵션을 사용하여 확인할 수 있다. GOLDILOCKS 서버의 version과 동일한 gsql 프로그램을 사용할 것을 권장한다.

```
% gsql --version
%
```

참고로 GOLDILOCKS 서버의 version은 다음과 같이 **VERSION** 함수를 통해 확인할 수 있다.

```
gSQL> SELECT version() FROM dual;
VERSION()
-----
Release Name.X.X.X revision(XXXXX)
1 row selected.
```


서버 접속

다음은 참조 옵션이다.

표 33-3 참조 옵션

옵션	설명
Username Password	접속 사용자와 패스워드이다.
--as {SYSDBA ADMIN}	접속 role을 지정한다.
--conn-string	Connection string을 지정한다.
--dsn	Data Source Name (DSN)을 지정한다.

사용자 접속

gsqI 프로그램을 이용하여 다양한 방법으로 GOLDILOCKS 서버에 접속할 수 있다.

다음과 같이 사용자 이름과 패스워드를 이용해 접속하는 것이 가장 단순한 사용 예이다. 다음 예는 test 사용자가 test 패스워드를 이용하여 접속하는 예이다. 접속에 성공할 경우, 대화형 모드로 동작하여 *gsQL>* 프롬프트가 사용자의 명령을 기다린다.

```
% gsql test test
Connected to GOLDILOCKS Database.
gsQL>
```

다음은 잘못된 사용자 이름이나 잘못된 패스워드를 사용한 경우에 발생하는 에러이다. 로그인에 실패할 경우, 에러 메시지를 출력한 후에 gsql 프로그램을 종료한다.

```
% gsql test invalid_password
ERR-28000(16004): invalid username/password; logon denied
%
```

다음은 GOLDILOCKS 서버가 구동되지 않았을 경우에 에러가 발생하는 예이다. 다음과 같은 에러 메시지와 함께 gsql 프로그램이 종료된다.

```
% gsql test test
ERR-HY000(11031): Unable to attach the shared memory segment
%
```

이 외에도 다음과 같은 옵션을 통해 접속할 수 있는데 자세한 내용은 다음 옵션들을 참조한다.

- --conn-string
- --dsn

SYSDBA 접속

서버를 구동하거나 database에 대한 모든 권한을 가진 SYSDBA role 등에 접속하려면 `--as {SYSDBA|ADMIN}` 옵션을 사용해야 한다.

다음은 GOLDILOCKS 서버를 구동하지 않은 상태에서 `--as sysdba`로 접속하는 예이다. *Connected to an idle instance* 메시지와 함께 `gSQL>` 프롬프트가 GOLDILOCKS를 구동할 수 있는 상태로 대기하고 있다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

다음은 GOLDILOCKS 서버가 이미 구동되어 있는 경우 `--as sysdba`로 접속하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

서버 구동 및 종료에 대한 자세한 내용은 [서버 구동 및 종료](#)를 참조한다.

gsql 종료

`gsql`은 다음과 같이 `\quit` 또는 `\q` 명령을 이용하여 종료한다.

자세한 내용은 [\quit](#)을 참조한다.

```
gSQL> \quit
%
```

대화형 모드 제어

다음은 참조 옵션이다.

표 33-4 참조 옵션

옵션	설명
<code>--prompt</code>	프롬프트를 지정한다.
<code>--no-prompt</code>	프롬프트를 제거한다.
<code>--import</code>	입력으로 SQL 파일을 실행한다.
<code>--silent</code>	SQL 파일을 실행할 때 명령어와 실행 결과를 출력하지 않는다.
<code>--enable-color</code>	질의 결과를 row 별로 구분하여 출력한다.

프롬프트 제어

대화형 모드를 시작할 때 다음 예와 같이 `--prompt` 옵션을 이용하여 프롬프트를 변경할 수 있다. 다음은 `gsqI` 프롬프트가 아닌 `GOLDILOCKS` 프롬프트로 변경하는 예이다.

```
% gsql test test --prompt GOLDILOCKS
Connected to GOLDILOCKS Database.
GOLDILOCKS>
```

프롬프트 제어에 대한 자세한 내용은 다음 옵션들을 참조한다.

- `--prompt`
- `--no-prompt`

파일로부터 수행

`gsqI` 프로그램을 이용하여 배치 작업 등을 수행할 때 `--import` 옵션을 이용하여 파일에 포함된 SQL 구문을 수행할 수 있다. 다음은 `$GOLDILOCKS_HOME/admin` 디렉토리에서 `PerformanceViewSchema.sql` 파일을 수행하는 예이다. 해당 예는 `--silent` 옵션을 이용하여 수행 결과를 출력하지 않도록 하였다.

```
% cd $GOLDILOCKS_HOME/admin
% gsql sys gliese --as sysdba --import 'PerformanceViewSchema.sql' --silent
%
```

파일로부터 `gsqI`을 수행하는 방법에 대한 자세한 내용은 다음 옵션들을 참조한다.

- `--import`
- `--silent`
- `--enable-color`

gsqI 설정 저장

gsqI 환경 파일 (gsqI.ini)

`gsqI` 환경 파일 (`.gsqI.ini`)은 `gsqI`을 구동할 때 자동으로 적용되는 옵션들로 구성된 파일이며 `$HOME` 디렉토리에 위치해야 한다.

환경 파일에 다수의 `dsn`을 설정할 수 있고 각 `dsn`마다 다른 옵션들을 지정할 수 있다. `gsqI`을 구동할 때 `dsn`을 입력하지 않을 경우, `GOLDILOCKS`의 옵션들을 적용한다.

사용 예

```
[GOLDILOCKS]
AUTOCOMMIT = OFF
AUTOTRACE = OFF
LINESIZE = 80
PAGESIZE = 20
VERTICAL = OFF
TIME = OFF
TIMING = OFF
ERROR = ON
COLSIZE = 8192
NUMSIZE = 20
DDLSize = 10000
HISTORY = 128
```

표 33-5 옵션

옵션	최소값	최대값	기본값
AUTOCOMMIT	OFF	ON	OFF
AUTOTRACE	OFF	ON/TRACEONLY	OFF
LINESIZE	1	10000	80
PAGESIZE	1	10000	20
VERTICAL	OFF	ON	OFF
TIME	OFF	ON	OFF
TIMING	OFF	ON	OFF
ERROR	ON	OFF	ON
COLSIZE	1	10485760	8192
NUMSIZE	1	50	20
DDLSize	1	10485760	10000
HISTORY	0	100000	128

노트

.gsql.ini가 없거나 옵션값이 유효하지 않은 경우에는 이를 무시한다.

glogin.sql

glogin.sql은 전역 설정 파일로써 \$GOLDILOCKS_DATA/conf/glogin.sql에 위치한다. gsql을 구동할 때 glogin.sql 파일이 있으면 해당 파일을 읽고 포함된 명령문을 실행한다. 이렇게 하면 모든 gsql 세션을 설정 (예: 라인 크기) 할

수 있다.

login.sql

login.sql은 사용자 설정 파일이다. gsql을 구동할 때 현재 디렉토리에 login.sql 파일이 있으면 해당 파일을 읽고 포함된 명령문을 실행한다.

login.sql의 설정은 glogin.sql의 설정보다 우선한다.

33.3 Interactive Command 사용

gsql 대화형 모드 명령어

다음은 참조 명령어이다.

표 33-6 참조 명령어

gsql 명령	설명
<code>\help</code>	gsql 명령어 목록

gsql 프로그램은 서버 접속을 통해 다음과 같은 대화형 모드로 동작한다.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

사용자는 프롬프트 상에서 SQL 구문을 입력하여 수행하거나 gsql 대화형 모드에서 수행할 수 있는 고유 명령어를 입력하여 수행할 수 있다. gsql 고유 명령어는 SQL 구문과 구별하기 위해 back-slash(\)로 시작한다. 즉, SQL 구문은 응용 프로그램에서 직접 사용할 수 있는 반면에 back-slash(\)로 시작하는 gsql 명령어는 응용 프로그램에서 사용할 수 없다.

노트

gsql 대화형 모드 명령어는 \ 로 시작한다.

다음은 SQL 구문을 대화형 모드에서 수행하는 예이다.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

다음은 gsql 명령어 `\exec sql`을 사용하여 동일한 SELECT 구문을 사용하는 예이다. `\exec sql` 명령은 Embedded SQL의 SQL 구문 시작을 의미하는 EXEC SQL과 구문이 동일할 뿐 gsql 고유의 명령어이다.

```
gSQL> \exec sql SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

\set time을 통해 현재 시간을 프롬프트에 출력하도록 설정할 수 있다.

```
gSQL> \set time on
12:45:34 gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
12:45:37 gSQL>
```

gsq! 대화형 모드의 고유 명령어들은 다음과 같이 **\help** 명령을 이용해 조회할 수 있다.

```
gSQL> \help
\help
\q[uit]
\i[mport] {'FILE'} Import SQL
\ed[it] [{'FILE'}|[HISTORY] num]] Edit SQL statement
\\ Executes the most recent history entry
\{n} Executes n'th history entry
\hi[story] Show history entries
\desc {[schema.]table_name} Show table description
\idesc {[schema.]index_name} Show index description
\spo[ol] ['filename' | OFF] Stores query results in a file
\ho[st] [command] Executes an operating system command
\set vertical {ON|OFF}
\set time {ON|OFF}
\set timing {ON|OFF}
\set color {ON|OFF}
\set error {ON|OFF}
\set autocommit {ON|OFF}
\set autotrace {ON|TRACEONLY|OFF}
\set serveroutput {ON|OFF}
\set heading {ON|OFF}
\set linesize {n} 0 < n <= 100000
\set pagesize {n} 0 < n <= 100000
```

```

\set colsize      {n}      0 < n <= 104857600
\set numsize      {n}      0 < n <= 50
\set ddlsize      {n}      0 < n <= 100000
\set history      {n}      n <= 100000 ( if n < 0, clear history buffer )
\var              {host_var_name} {INTEGER|BIGINT|VARCHAR(n)}
\exec             [{:host_var_name} := {constant}]
\exec sql         {sql string}
\prepare sql      {sql string}
\dynamic sql      {host_var_name}
\explain plan     [{ON|ONLY}] {sql string}
\print            [{host_var_name}]
\ddl_cluster
\ddl_db
\ddl_tablespace   {name}
\ddl_profile      {name}
\ddl_auth         {name}
\ddl_schema       {name}
\ddl_table        [{schema.}name]
\ddl_constraint   [{schema.}name]
\ddl_index        [{schema.}name]
\ddl_view         [{schema.}name]
\ddl_sequence     [{schema.}name]
\ddl_synonym      [{schema.}name]
\ddl_publicsynonym {name}
\ddl_procedure    [{schema.}name]
\startup          [{nomount|mount|open}]
\shutdown         [{abort|immediate|transactional|normal}]
\cstartup         [{nomount|mount|open}]
\cshutdown        [{abort|normal}]
\connect          userid password [as {sysdba|admin}]

```

서버 구동 및 종료

다음은 참조 명령어이다.

표 33-7 참조 명령어

gsql 명령	설명
\startup	서버를 구동한다.
\shutdown	서버를 종료한다.

gsql 명령	설명
<code>\cstartup</code>	Cluster 환경을 위한 서버를 구동한다.
<code>\cshutdown</code>	Cluster 환경을 위한 서버를 종료한다.

Standalone

서버를 구동하거나 종료하려면 다음과 같이 SYSDBA 또는 ADMIN role로 접속해야 한다.

- SYSDBA role
 - 서버 구동 및 종료 등 DBA 관련 모든 권한을 가진다.
- ADMIN role
 - SYSDBA와 동일한 권한을 가지지만 한 session에만 접속할 수 있다.
 - 유효 session이 없거나 비정상적인 상황에서의 비상 접속을 위한 role이다.

다음은 GOLDILOCKS 서버가 구동되지 않은 상태에서 SYSDBA role로 접속하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

gsql은 GOLDILOCKS 서버가 없는 상태에서 idle instance에 접속하고 `\startup`명령을 통해 GOLDILOCKS 서버를 특정 단계로 구동하여 서버에 접속한다. 다음은 NOMOUNT 단계로 GOLDILOCKS 서버를 구동하는 예이다.

```
gSQL> \startup NOMOUNT
Startup success
gSQL>
```

GOLDILOCKS를 최초로 구동할 때만 `\startup`명령을 사용하며 GOLDILOCKS 서버를 구동하고 서버에 접속한 후에는 `ALTER SYSTEM {MOUNT | OPEN} DATABASE` 구문을 이용해 GOLDILOCKS 서버의 각 단계로 전환한다.

```
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
gSQL>
```

GOLDILOCKS 서버를 별도의 단계 전환 없이 서비스 가능한 OPEN 단계로 구동할 경우, 다음과 같이 별도의 옵션없이 `\startup` 명령을 수행한다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
```

```
Startup success
gSQL>
```

이미 서버가 특정 단계로 구동된 경우 `\startup` 명령은 다음과 같이 에러를 출력한다. 서버가 OPEN 단계까지 구동된 경우가 아니라면 다음과 같이 ALTER SYSTEM 구문을 이용하여 서버를 OPEN 단계로 구동할 수 있다.

```
% gsql sys gliese--as sysdba
Connected to GOLDILOCKS Database.
gSQL> \startup
ERR-HY000(11029): shared memory segment exists
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
gSQL>
```

서버 구동과 관련한 자세한 내용은 다음 링크를 참조한다.

- 서버 구동 단계: **다단계 시작**
- 최초 서버 구동 명령: **\startup**
- 서버 단계 변경: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**

서버를 종료하려면 다음과 같이 `\shutdown` 명령을 사용한다. 서버 종료에 대한 자세한 내용은 **\shutdown** 명령을 참조한다.

```
gSQL> \shutdown
Shutdown success
gSQL>
```

Cluster

`\cstartup` 명령어와 `\cshutdown` 명령어를 사용하여 서버를 구동하거나 종료하기 위해서는 `odbc.ini` 파일에 `LOCATOR_DSN` 속성이 정의되어야 하고, `gsqlnet`을 이용하여 `SYSDBA` 또는 `ADMIN` role 로 접속해야 한다.

다음은 GOLDILOCKS 서버가 구동되지 않은 상태에서 `gsqlnet`을 사용하여 `SYSDBA` role로 접속하는 예이다.

```
% gsqlnet sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

다음은 `\cstartup` 명령을 이용하여 GOLDILOCKS 서버를 LOCAL OPEN 단계로 구동하고 서버를 OPEN 단계로 전환하는 예이다.

```
gSQL> \cstartup LOCAL OPEN
Startup success
gSQL> ALTER SYSTEM OPEN GLOBAL DATABASE;
System altered.
gSQL>
```

\cstartup 명령어와 \cshutdown 명령어는 CS 환경에서 사용하기 때문에 서버를 구동하거나 종료하기 위해서 모든 서버에 glsnr가 서비스 중이어야 한다.

다음은 glsnr이 서비스하고 있지 않은 GOLDILOCKS 서버를 구동하려고 시도하는 예이다.

```
gSQL> \cstartup
ERR-HY000(58000): MEMBER(G1N1): the sender failed to connect to the member(1)
ERR-HY000(11067): MEMBER(G1N1): fail to connect to an host with a socket : connect() :
stnConnect() returned errno(111)
gSQL>
```

다음은 **odbc.ini 파일**에 LOCATOR_DSN 속성과 LOCATOR DSN이 정의된 예이다.

```
% cat .odbc.ini
# Edit the SYSTEM or USER DSN ini file (/etc/odbc.ini or ~/.odbc.ini) and add a data source
using the syntax:
[GOLDILOCKS]
HOST=127.0.0.1
PORT=20101
UID=test
PWD=test
LOCATOR_DSN=LOCATOR
[LOCATOR]
FILE=/home/test/.locator.ini
```

다음은 odbc.ini 파일에 LOCATOR_DSN 속성이 정의되지 않은 상태에서 서버를 구동하려는 예이다.

```
gSQL> \cstartup
ERR-HY000(40057): not specified valid location information
gSQL>
```

\CSTARTUP {LOCAL | GLOBAL} OPEN 이외의 명령어는 해당 GOLDILOCKS 서버에만 적용된다. 따라서 \CSTARTUP NOMOUNT, MOUNT 명령어 이후 **ALTER SYSTEM {MOUNT | OPEN} DATABASE** 을 이용하여 단계별로 구동하기 위해서는 다른 서버들도 LOCAL OPEN 상태까지 직접 구동해야 한다.

다음은 \CSTARTUP MOUNT 이후 단계별 구동을 시도하는 예이다. 해당 서버는 MOUNT 단계로 구동 되지만 다른 서버는 LOCAL OPEN 단계까지 구동되지 않아 에러가 발생한다.

```

gSQL> \CSTARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN LOCAL DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN GLOBAL DATABASE;
ERR-HY000(58000): MEMBER(G1N1): the sender failed to connect to the member(1)
ERR-HY000(11067): MEMBER(G1N1): fail to connect to an host with a socket : connect() :
stnConnect() returned errno(111)
gSQL>

```

서버 구동과 관련한 자세한 내용은 아래의 링크를 참조한다.

- 서버 구동 단계: **다단계 시작**
- 최초 서버 구동 명령: **\cstartup**
- 서버 단계 변경: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**

전체 서버를 종료하려면 다음 예와 같이 \cshutdown 명령을 사용한다. 전체 서버 종료에 대한 자세한 내용은 **\cshutdown** 명령을 참조한다.

```

gSQL> \cshutdown
Shutdown success
gSQL>

```

SQL 구문 수행

다음은 참조 명령어이다.

표 33-8 참조 명령어

gsql 명령	설명
\import	SQL 파일을 실행한다.
\set autocommit	SQL 구문을 실행할 때마다 자동으로 COMMIT 한다.
16. SQL References	SQL 구문의 종류이다.

SQL 구문 입력

대화형 모드에서 gsql은 대화형 모드 프롬프트인 gSQL>을 출력하고 사용자의 입력을 기다린다. 사용자가 SQL 구문 입력을 완료하면 gsql은 SQL 구문을 서버로 전달하여 수행한다. 대화형 모드에서 SQL 구문을 종료하려면 semi-colon (;) 다음에 **enter**를 입력한다. SQL 구문을 처리한 후에는 수행 결과를 출력하고 프롬프트 gSQL>이 다시 출력된 후 사용자의 입력을 기다린다.

다음은 한 개 라인에 SQL 구문을 입력하고 실행하는 예이다.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
gSQL>
```

다음은 여러 라인에 걸쳐 SQL 구문을 입력하고 수행하는 예이다. SQL 구문이 완료되지 않은 상태에서 **enter**를 입력할 경우 다음 라인으로 이동하여 라인번호를 출력하고 사용자의 입력을 기다린다.

```
gSQL> SELECT id, name
2
```

계속해서 다음과 같이 SQL 구문을 모두 입력하고 semi-colon (;)과 **enter**를 입력할 경우, SQL 구문을 실행하고 *gSQL* L> 프롬프트가 다음 명령을 기다린다.

```
gSQL> SELECT id, name
2 FROM t1
3 WHERE id > 0;
ID NAME
-- -----
1 leekmo
2 mkkim
3 xcom73
3 rows selected.
gSQL>
```

다음과 같이 single quote (') 안에 포함된 semi-colon (;)은 문자열로 인식되어 SQL 구문 입력이 완료된 것으로 인식하지 않는다.

```
gSQL> SELECT id, ';'
2 FROM t1
3 WHERE id = 1;
ID ';'
-- ---
1 ;
1 row selected.
gSQL>
```

파일로부터 SQL 구문 입력

대화형 모드에서 SQL 구문을 직접 입력하지 않고 파일로부터 수행할 경우, `\import` 명령을 사용한다. 다음은 sample.sql 파일을 읽어 다수의 SQL 구문들을 수행하는 예이다. 자세한 내용은 `\import` 명령을 참조한다.

```
gSQL> \import 'sample.sql'
```

- Drop table

```
DROP TABLE IF EXISTS t1;
Table dropped.
```

- Create table

```
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name  VARCHAR(128),
    addr  VARCHAR(128)
);
Table created.
```

- Create index

```
CREATE INDEX t1_idx_name ON t1(name);
Index created.
```

- Insert three rows

```
INSERT INTO t1
VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
       ( 2, 'mkkim' , 'Seoul, Korea' ),
       ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
```

- Commit transaction

```
COMMIT;
Commit complete.
gSQL>
```

gsql 주석

대화형 모드에서 사용되는 주석은 SQL 구문의 주석과 동일하다. **--import** 옵션을 통해 파일로부터 입력되는 주석도 SQL 구문의 주석과 동일하다. SQL 구문의 주석에 대한 자세한 내용은 **Comments**를 참조한다.

다음은 대화형 모드에서 line comment를 사용하는 예이다.

```
gsQL> -- outside line-comment
gsQL> SELECT id, name
2 FROM t1 -- inside line-comment
3 WHERE id = 1;
ID NAME
-- -----
1 leekmo
1 row selected.
gsQL>
```

다음은 \import 명령을 이용해 파일의 내용을 수행하는 예이다. 대화형 모드에서의 주석과 동일하게 처리되며 파일의 내용이 모두 실행된 후에 *gsQL>* 프롬프트가 다음 SQL 구문을 기다린다.

```
gsQL> \import 'sample.sql'
```

- Drop table

```
DROP TABLE IF EXISTS t1;
Table dropped.
```

- Create table

```
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name  VARCHAR(128),
    addr  VARCHAR(128)
);
Table created.
```

- Create index

```
CREATE INDEX t1_idx_name ON t1(name);
Index created.
```

- Insert three rows

```
INSERT INTO t1
VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
      ( 2, 'mkkim' , 'Seoul, Korea' ),
      ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
```

- Commit transaction

```
COMMIT;
Commit complete.
gSQL>
```

다음은 대화형 모드에서 multi-line comment를 사용하는 예이다.

```
gSQL> SELECT id, name
2      /*
3      multi-line
4      comment
5      */
6      FROM t1
7      WHERE id = 1;
ID NAME
-- -----
1 leekmo
1 row selected.
gSQL>
```

gsql 대화형 모드에서 사용할 수 있는 주석은 SQL 구문의 주석과 동일하며 주석에 대한 자세한 내용은 SQL 구문의 **Comments**를 참조한다.

SQL 수행 결과

대화형 모드에서 SELECT 구문 등의 질의를 수행하면 gsql은 질의 결과를 출력한다. 출력 결과는 column 이름으로 구성된 상단부, 질의 결과에 해당하는 데이터들, 질의 결과의 요약인 하단부로 구성된다.

다음은 SELECT 구문을 수행하는 예이다.

```
gSQL> SELECT * FROM t1;
ID NAME      ADDR
-- -----
1 leekmo Seoul, Korea
2 mkkim  Seoul, Korea
```



```
3 xcom73 Incheon, Korea
3 rows selected.
gSQL>
```

위의 예에서 출력 결과의 상단부는 column 이름과 함께 데이터와 구분하기 위한 hyphen (-)으로 구성된다. 출력 결과의 중앙에는 질의 결과가 출력되며 하나의 row는 한 line을 기준으로 출력되며 각 데이터 값은 공백으로 구분된다. 출력 결과의 하단부는 질의 결과의 요약으로써 세 개의 row가 검색되었음을 출력한다.

SQL 질의 이외의 **Data Definition Language**, **Data Manipulation Language**, **Control Language** 등의 구문은 각 SQL 구문의 특성에 맞는 수행 결과를 요약한 정보를 출력한다.

다음은 DDL, DML, control language를 각각 수행하고 그 결과를 출력하는 예이다.

- DDL 구문

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
```

- DML 구문

```
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- Transaction control 구문

```
gSQL> COMMIT;
Commit complete.
gSQL>
```

SQL 수행 에러

SQL 구문을 수행할 때 발생하는 에러는 다음과 같은 정보로 구성된다.

- SQLSTATE 정보: SQL 표준의 SQLSTATE 값
- Error code: GOLDILOCKS 고유의 error code 값
- Error message: 에러 발생 원인을 기술하는 에러 메시지

다음은 SELECT 구문을 수행할 때 에러가 발생하는 예이다. ERR-42000 값이 SQLSTATE에 해당하며, (16040) 값이 GOLDILOCKS 고유의 error code이고 두 번째 라인에서 invalid_table에 해당하는 테이블이나 view가 존재하지 않는다는 에러 메시지로 구성되어 있다.

```
gSQL> SELECT id, name
2      FROM invalid_table
3      WHERE id > 0;
```

```
ERR-42000(16040): table or view does not exist :
      FROM invalid_table
      *
ERROR at line 2:
gSQL>
```

대화형 명령어의 수행 결과

SQL 구문과 함께 사용하는 대화형 명령어들은 SQL 구문을 수행한 것과 동일한 결과와 에러를 출력한다.

다음은 `\prepare sql` 명령을 이용해 SQL 구문을 준비하고 `\exec` 명령을 이용해 준비된 SQL 구문을 수행하는 예이다.

```
gSQL> \prepare sql SELECT * FROM t1;
SQL prepared.
gSQL> \exec
ID NAME      ADDR
--  -----
1 leekmo Seoul, Korea
2 mkkim  Seoul, Korea
3 xcom73 Inchon, Korea
3 rows selected.
gSQL>
```

다음은 `\prepare sql` 명령을 사용할 때 SQL 구문에 에러가 존재하는 경우로서 SQL 구문을 수행한 것과 동일한 에러를 출력한다.

- `\prepare sql` 명령의 에러

```
gSQL> \prepare sql SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
      *
ERROR at line 1:
```

- 동일한 SQL 구문의 에러

```
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
      *
ERROR at line 1:
gSQL>
```

대화형 모드의 gsql 명령어들 중에 SQL 구문을 포함하지 않거나 출력할 내용이 없는 명령어의 경우, 해당 명령이 성공할 때 별도의 메시지 없이 *gSQL>* 프롬프트만 출력한다.

다음은 별도의 성공 메시지 없이 대화형 명령어를 수행하는 예이다. 에러가 발생할 경우, 에러 메시지를 출력한다.

- 대화형 gsql 명령어가 정상적으로 수행되는 경우

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
```

- 대화형 gsql 명령어가 에러를 발생시키는 경우

```
gSQL> \var v2 INVALID TYPE
ERR-42000(40000): syntax error
\var v2 INVALID TYPE
.....^      ^
Error at line 1
gSQL>
```

대화형 명령어의 수행 결과는 **Interactive Command References**의 각 명령어 설명과 사용 예를 참조한다.

SQL 구문의 자동 COMMIT

gsql의 대화형 모드에서 수행하는 SQL 구문은 **COMMIT** 또는 **ROLLBACK** 구문을 통해 트랜잭션을 완료하거나 철회할 수 있다.

다음은 다수의 INSERT 구문을 수행하고 COMMIT 또는 ROLLBACK을 수행하는 예이다. 다음 예에서 첫 번째 INSERT 구문은 COMMIT에 의해 정상적으로 완료된 반면에 두 번째와 세 번째 INSERT 구문은 ROLLBACK에 의해 수행이 철회되었다.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom73' );
1 row created.
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
```

```

1 leekmo
1 row selected.
gSQL>

```

SQL 구문을 수행할 때마다 자동으로 트랜잭션을 COMMIT 하려면 다음과 같이 **\set autocommit** 명령을 이용하여 제어할 수 있다. 위의 예와 동일한 SQL 구문을 수행한 경우, 각 SQL 구문이 자동으로 COMMIT 되어 두 번째, 세 번째 INSERT 구문의 트랜잭션도 ROLLBACK과 관계없이 완료되었음을 알 수 있다.

```

gSQL> \set autocommit on
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom73' );
1 row created.
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
-- -----
1 leekmo
2 mkkim
3 xcom73
3 rows selected.
gSQL>

```

자동 COMMIT 제어에 대한 자세한 내용은 **\set autocommit** 명령어를 참조한다.

수행 중인 SQL 구문의 강제종료

대화형 모드에서 실행 중인 SQL 구문을 강제로 종료하려면 **Ctrl+C**를 입력한다.

다음은 오래 수행되는 SELECT 구문을 강제 종료하는 예이다. SQL 구문을 수행하는 중에 키보드를 이용해 **Ctrl+C**를 입력할 경우 *operation canceled* 에러 메시지를 출력하고 수행 중이던 SQL 구문이 강제 종료된다.

```

gSQL> SELECT COUNT(*)
FROM
    t1 AS v01,
    t1 AS v02,
    t1 AS v03,

```

```

t1 AS v04,
t1 AS v05,
t1 AS v06,
t1 AS v07,
t1 AS v08,
t1 AS v09,
t1 AS v10,
t1 AS v11,
t1 AS v12,
t1 AS v13,
t1 AS v14,
t1 AS v15,
t1 AS v16,
t1 AS v17,
t1 AS v18,
t1 AS v19,
t1 AS v20;

```

```

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

```

```

^C

```

```

ERR-HY008(13043): operation canceled

```

```

gSQL>

```

출력 결과 제어

다음은 참조 명령어이다.

표 33-9 참조 명령어

gsql 명령	설명
\set color	각 row를 다른 색깔로 구별한다.
\set colsize	Column 결과의 최대 크기를 제어한다.
\set error	에러 메시지 출력 여부를 제어한다.
\set linesize	Row의 최대 출력 길이를 제어한다.
\set numsize	숫자값의 최대 digit 개수를 제어한다.
\set pagesize	Page에 포함할 row의 개수를 제어한다.
\set timing	SQL 구문 수행 시간을 출력한다.
\set vertical	각 column을 한 line에 출력한다.

Page 구성 제어

앞서 SQL 수행 결과에 기술한대로 질의 결과의 상단부에는 column 이름을 출력하고 데이터를 출력한 후, 하단부에 요약 결과를 출력한다.

질의 결과가 많을 경우 pagesize (기본값 20) 단위로 나누어 각 page마다 column 이름과 데이터를 출력하고 모든 page의 출력이 완료된 후 하단부에 요약 결과를 출력한다.

다음은 pagesize를 5로 변경하여 질의 결과를 출력한 예이다. Pagesize 제어에 대한 자세한 내용은 `\set pagesize` 명령을 참조한다.

```
gSQL> \set pagesize 5
gSQL> SELECT table_schema, table_name FROM dictionary WHERE table_name like 'ALL_%' FETCH 20;
TABLE_SCHEMA      TABLE_NAME
-----
DICTIONARY_SCHEMA ALL_ALL_TABLES
DICTIONARY_SCHEMA ALL_COL_COMMENTS
DICTIONARY_SCHEMA ALL_COL_PRIVS
DICTIONARY_SCHEMA ALL_COL_PRIVS_MADE
DICTIONARY_SCHEMA ALL_COL_PRIVS_RECD
TABLE_SCHEMA      TABLE_NAME
-----
DICTIONARY_SCHEMA ALL_CONSTRAINTS
DICTIONARY_SCHEMA ALL_CONS_COLUMNS
DICTIONARY_SCHEMA ALL_DB_PRIVS
DICTIONARY_SCHEMA ALL_DB_PRIVS_MADE
DICTIONARY_SCHEMA ALL_DB_PRIVS_RECD
TABLE_SCHEMA      TABLE_NAME
-----
DICTIONARY_SCHEMA ALL_INDEXES
DICTIONARY_SCHEMA ALL_IND_COLUMNS
DICTIONARY_SCHEMA ALL_SCHEMAS
DICTIONARY_SCHEMA ALL_SCHEMA_PATH
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS
TABLE_SCHEMA      TABLE_NAME
-----
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS_MADE
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS_RECD
DICTIONARY_SCHEMA ALL_SEQUENCES
DICTIONARY_SCHEMA ALL_SEQ_PRIVS
DICTIONARY_SCHEMA ALL_SEQ_PRIVS_MADE
20 rows selected.
gSQL>
```

질의 결과의 각 row는 linesize (기본값 80) 단위로 출력되는데 한 row의 길이가 linesize를 넘어갈 경우 다음 line에 결과를 출력한다.

다음은 linesize를 200으로 조정하기 전과 후의 질의 결과이다. Linesize 제어에 대한 자세한 내용은 **\set linesize** 명령을 참조한다.

```
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME      COLUMN_NAME
-----
COMMENTS
-----
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA
Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME
Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME
Name of the tablespace containing the table
3 rows selected.
```

- Linesize를 200으로 설정한다.

```
gSQL> \set linesize 200
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME      COLUMN_NAME      COMMENTS
-----
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA      Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME          Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME      Name of the tablespace containing the table
3 rows selected.
gSQL>
```

터미널 상에서 각 row를 구별하여 가독성을 높이고 싶은 경우 **\set color** 명령을 통해 다음과 같이 row마다 색깔을 달리하여 출력할 수 있다. 자세한 내용은 **\set color** 명령을 참조한다.

```
gSQL> \set color on
gSQL> SELECT table_schema, table_name FROM dictionary WHERE table_name like 'ALL_%' FETCH 10;
TABLE_SCHEMA      TABLE_NAME
-----
DICTIONARY_SCHEMA ALL_ALL_TABLES
DICTIONARY_SCHEMA ALL_COL_COMMENTS
DICTIONARY_SCHEMA ALL_COL_PRIVS
DICTIONARY_SCHEMA ALL_COL_PRIVS_MADE
```

```

DICTIONARY_SCHEMA ALL_COL_PRIVS_RECD
DICTIONARY_SCHEMA ALL_CONSTRAINTS
DICTIONARY_SCHEMA ALL_CONS_COLUMNS
DICTIONARY_SCHEMA ALL_DB_PRIVS
DICTIONARY_SCHEMA ALL_DB_PRIVS_MADE
DICTIONARY_SCHEMA ALL_DB_PRIVS_RECD
10 rows selected.
gSQL>

```

질의 결과를 출력할 때 row를 line 단위가 아닌 column 단위로 하나의 line에 출력하려면 `\set vertical on` 명령을 사용한다. 다음 예와 같이 column 단위로 출력 결과가 생성되는데 자세한 내용은 `\set vertical` 명령을 참조한다.

```

gSQL> SELECT * FROM v$system_stat FETCH 5;
STAT_NAME          STAT_VALUE COMMENTS
-----
SYSTEM_SAR          3 system available resource( 0:none, 1:session 2:database )
MAX_ENVIRONMENT_COUNT 128 maximum environment count
FREE_ENVIRONMENT_ID  3 available environment identifier
MAX_SESSION_COUNT   128 maximum session count
FREE_SESSION_ID      4 available session identifier
5 rows selected.
gSQL> \set vertical on
gSQL> SELECT * FROM v$system_stat FETCH 5;
      STAT_NAME # SYSTEM_SAR
      STAT_VALUE # 3
      COMMENTS # system available resource( 0:none, 1:session 2:database )
      STAT_NAME # MAX_ENVIRONMENT_COUNT
      STAT_VALUE # 128
      COMMENTS # maximum environment count
      STAT_NAME # FREE_ENVIRONMENT_ID
      STAT_VALUE # 3
      COMMENTS # available environment identifier
      STAT_NAME # MAX_SESSION_COUNT
      STAT_VALUE # 128
      COMMENTS # maximum session count
      STAT_NAME # FREE_SESSION_ID
      STAT_VALUE # 4
      COMMENTS # available session identifier
5 rows selected.
gSQL>

```


데이터 값의 출력 제어

gsql은 질의 결과에 포함되는 모든 데이터를 문자열로 변환하여 출력한다.

숫자형은 numsize (기본값 20)를 digit 단위로 출력하며 numsize 보다 큰 숫자값은 exponent 형태로 표현한다. 다음은 numsize를 이용해 출력값을 제어하는 예이며 자세한 내용은 `\set numsize` 명령을 참조한다.

```
gsQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM              RESULT
-----
1234567890123 1.52415787532276E+24
1 row selected.
gsQL> \set numsize 50
gsQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM              RESULT
-----
1234567890123 1524157875322755800955129
1 row selected.
gsQL>
```

숫자값 출력에 대한 다양한 제어가 필요한 경우, 다음과 같이 `TO_CHAR( number )` 함수를 이용해 출력을 제어할 수 있다.

```
gsQL> SELECT num, TO_CHAR( num * num, '$999,999,999,999,999,999,999,999' ) AS result FROM
t1;
          NUM RESULT
-----
1234567890123 $1,524,157,875,322,755,800,955,129
1 row selected.
gsQL>
```

DATE/ TIME/ TIMESTAMP와 같은 날짜/ 시간 타입의 경우, gsql은 다음과 같은 프로퍼티 정보를 이용해 출력 형태를 결정한다.

- NLS_DATE_FORMAT
- NLS_TIMESTAMP_FORMAT
- NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT
- NLS_TIME_FORMAT
- NLS_TIME_WITH_TIME_ZONE_FORMAT

해당 프로퍼티는 `TO_CHAR()`, `TO_DATE()` 등의 함수에서 format 정보를 입력하지 않았을 때 사용되는 프로퍼티이다. ODBC나 JDBC를 이용한 응용 프로그램이 SQL 구문에서 `TO_CHAR()`, `TO_DATE()` 등의 함수를 사용할 때만 위의 프로퍼티를 사용하는 반면에, gsql 은 날짜/ 시간 데이터 값을 출력하기 위해 최초로 접속할 때 해당 프로퍼티 정보

를 획득하여 이를 이용한다. 대화형 모드에서 다음과 같은 SQL 구문으로 프로퍼티를 변경할 경우, gsql은 갱신된 프로퍼티 정보를 사용한다.

- ALTER SESSION SET property_name
- ALTER SYSTEM SET property_name

다음은 DATE column에 대한 출력을 제어하는 예이다.

```
gSQL> SELECT enter_date FROM t1;
ENTER_DATE
-----
2014-08-26
1 row selected.
gSQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
Session altered.
gSQL> SELECT enter_date FROM t1;
ENTER_DATE
-----
26-AUG-2014
1 row selected.
gSQL>
```

프로퍼티를 변경하지 않고 날짜/ 시간 출력값을 제어하려면 다음과 같이 **TO_CHAR(datetime)** 함수를 사용한다.

```
gSQL> SELECT TO_CHAR( enter_date, 'YYYY"년" MM"월" DD"일"' ) as result FROM t1;
RESULT
-----
2014년 08월 26일
1 row selected.
gSQL>
```

BINARY나 VARBINARY와 같은 이진 문자열의 경우, 다음과 같이 hex 값으로 결과를 출력한다.

```
gSQL> select * from t1;
BINARY_VALUE
-----
A010002F37000000000000
1 row selected.
gSQL>
```

LONG VARCHAR, LONG VARBINARY 타입의 데이터는 아주 긴 문자열을 포함할 수 있어 colsize (기본값 8192) 만큼만 데이터를 출력한다. 다음은 colsize를 줄여 LONG VARCHAR 타입인 TEXT column의 일부 데이터를 출력한

예인데 colsize 값을 늘려 데이터를 모두 출력할 수 있다. 자세한 내용은 `\set colsize` 명령을 참조한다.

```
gSQL> \set colsize 100
gSQL> SELECT view_name, text FROM all_views WHERE view_name = 'ALL_ALL_TABLES';
VIEW_NAME      TEXT
-----
ALL_ALL_TABLES SELECT
                        auth.AUTHORIZATION_NAME      ❶ OWNER
                        , sch.SCHEMA_NAME
```

1 row selected.

gSQL>

SQL 구문 수행시간

SQL 구문을 수행할 때 수행 시간을 출력하려면 다음과 같이 `\set timing on` 명령을 사용한다. 자세한 내용은 `\set timing` 명령을 참조한다.

```
gSQL> \set timing on
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
Elapsed time: 0.08500 ms
```

출력 결과 저장

Terminal 출력과 동시에 gsql에서 수행되는 모든 결과를 file에 기록한다.

표 33-10 참조 명령어

gsql 명령	설명
<code>\spool</code>	출력 결과를 file에 저장한다.

출력 결과 저장과 관련된 기능들은 `\spool`의 명령어들을 참조한다.

출력 결과는 다음과 같이 저장할 수 있다.

- result.txt 파일에 실행 결과를 저장하기 시작

```
gSQL> \SP00L 'result.txt'
```

- Spool 상태 확인

```
gSQL> \SP00L
```

```
currently spooling to result.txt
```

```
gSQL> SELECT * FROM T1 WHERE C1 < 10;
```

- Spool 기능 종료

```
gSQL> \SP00L OFF
```

SQL 객체 정보 조회

SQL 객체에 대한 정보는 **DICTIONARY_SCHEMA**, **INFORMATION_SCHEMA**의 각종 view들로 조회할 수 있다.

표 33-11 참조 명령어

gsql 명령	설명
\desc	테이블 정보를 조회한다.
\idesc	인덱스 정보를 조회한다.

예를 들어 다음과 같이 생성된 테이블과 인덱스가 있을 경우,

```
CREATE TABLE t1
(
  id    INTEGER PRIMARY KEY,
  name  VARCHAR(128),
  addr  VARCHAR(128)
);
CREATE INDEX t1_idx_name ON t1(name);
```

사용자는 다음과 같이 일련의 SQL 구문들을 수행하여 테이블과 관련된 정보를 조회할 수 있다.

```
gSQL> SELECT table_schema, table_name FROM user_tables WHERE table_name = 'T1';
```

```
TABLE_SCHEMA TABLE_NAME
```

```
-----
```

```
PUBLIC        T1
```

```
1 row selected.
```

```
gSQL> SELECT column_name, data_type, nullable FROM user_tab_columns WHERE table_schema = '
```

```

PUBLIC' AND table_name = 'T1';
COLUMN_NAME DATA_TYPE          NULLABLE
-----
ID            NUMBER             N
NAME          VARCHAR            Y
ADDR          VARCHAR            Y
3 rows selected.
gSQL> SELECT index_name FROM user_indexes WHERE table_schema = 'PUBLIC' AND table_name = 'T1';
INDEX_NAME
-----
T1_PRIMARY_KEY_INDEX
T1_IDX_NAME
2 rows selected.
gSQL>

```

gsql은 다음과 같이 \desc 명령어를 이용하여 테이블과 관련된 정보를 쉽게 조회할 수 있다. 출력 결과 및 자세한 내용은 **\desc** 명령을 참조한다.

```

gSQL> \desc t1
COLUMN_NAME TYPE                IS_NULLABLE
-----
ID            NUMBER(10,0)             FALSE
NAME          VARCHAR(128)          TRUE
ADDR          VARCHAR(128)          TRUE
INDEX_NAME    TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-----
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS    BTREE      TRUE     ID
T1_IDX_NAME      MEM_TEMP_TBS    BTREE      FALSE    NAME
CONSTRAINT_NAME CONSTRAINT_TYPE ASSOCIATED_INDEX COLUMNS
-----
T1_PRIMARY_KEY   PRIMARY KEY     T1_PRIMARY_KEY_INDEX ID
gSQL>

```

명령어 \idesc는 다음 예와 같이 인덱스와 관련된 정보를 보여준다. 자세한 내용은 **\idesc** 명령을 참조한다.

```

gSQL> \idesc t1_idx_name
COLUMN_NAME ORDINAL_POSITION IS_ASCENDING_ORDER IS_NULLS_FIRST
-----
NAME                1 TRUE                FALSE
gSQL>

```

SQL 객체의 DDL 구문 출력

다음과 같은 gsql 명령어들은 SQL 객체의 현재 상태에 해당하는 DDL 구문을 출력한다. SQL 객체를 생성하는 CREATE 구문 뿐만 아니라 다양한 옵션을 통해 관련 객체의 DDL 구문을 출력할 수 있다.

표 33-12 참조 명령어

gsql 명령	설명
\ddl_cluster	클러스터와 관련된 DDL을 출력한다.
\ddl_db	데이터베이스와 관련된 DDL을 출력한다.
\ddl_tablespace	테이블스페이스와 관련된 DDL을 출력한다.
\ddl_profile	Profile과 관련된 DDL을 출력한다.
\ddl_audit_policy	Audit policy와 관련된 DDL을 출력한다.
\ddl_auth	계정과 관련된 DDL을 출력한다.
\ddl_schema	스키마와 관련된 DDL을 출력한다.
\ddl_publicsynonym	Public synonym과 관련된 DDL을 출력한다.
\ddl_table	테이블과 관련된 DDL을 출력한다.
\ddl_constraint	제약 조건과 관련된 DDL을 출력한다.
\ddl_index	인덱스와 관련된 DDL을 출력한다.
\ddl_view	View와 관련된 DDL을 출력한다.
\ddl_sequence	시퀀스와 관련된 DDL을 출력한다.
\ddl_synonym	Synonym과 관련된 DDL을 출력한다.
\ddl_procedure	Procedure와 관련된 DDL을 출력한다.
\set ddlsize	DDL 출력 버퍼의 크기를 제어한다.

다음은 orders 테이블을 생성하는 예이다.

```
gSQL>
CREATE TABLE ORDERS
(
    O_ID          INTEGER,
    O_D_ID        INTEGER,
    O_W_ID        INTEGER,
    O_C_ID        INTEGER,
    O_ENTRY_D     TIMESTAMP,
    O_CARRIER_ID INTEGER,
    O_OL_CNT      NUMERIC(8),
    O_ALL_LOCAL   NUMERIC(1),
    PRIMARY KEY(O_W_ID, O_D_ID, O_ID) INDEX ORDERS_PK_IDX
);
Table created.
```

위에서 orders 테이블을 생성한 CREATE TABLE 구문은 다음과 같이 \ddl_table 명령을 통해 출력한다.

```
gSQL> \ddl_table orders CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE TABLE "PUBLIC"."ORDERS"
(
    "O_ID" NUMBER( 10, 0 )
    , "O_D_ID" NUMBER( 10, 0 )
    , "O_W_ID" NUMBER( 10, 0 )
    , "O_C_ID" NUMBER( 10, 0 )
    , "O_ENTRY_D" TIMESTAMP( 6 ) WITHOUT TIME ZONE
    , "O_CARRIER_ID" NUMBER( 10, 0 )
    , "O_OL_CNT" NUMBER( 8, 0 )
    , "O_ALL_LOCAL" NUMBER( 1, 0 )
)
PCTFREE 10
PCTUSED 60
INITRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
)
TABLESPACE "MEM_DATA_TBS"
;
```

위의 결과에서 SET AUTHORIZATION 구문은 CREATE TABLE 구문을 수행한 소유자를 의미한다. 출력한 CREATE TABLE 구문에는 사용자가 입력하지 않은 정보, 즉 테이블이 속한 스키마 이름, 테이블이 저장된 테이블스페이스 이름 및 테이블의 물리적 정보도 함께 출력된다.

테이블에 생성된 제약 조건에 해당하는 DDL은 다음과 같이 \ddl_table의 CONSTRAINT 옵션을 사용하여 출력한다.

```
gSQL> \ddl_table orders CONSTRAINT
SET SESSION AUTHORIZATION "TEST";
ALTER TABLE "PUBLIC"."ORDERS"
ADD CONSTRAINT "PUBLIC"."ORDERS_PRIMARY_KEY"
PRIMARY KEY
(
    "O_W_ID" ASC NULLS LAST
```

```

        , "O_D_ID" ASC NULLS LAST
        , "O_ID" ASC NULLS LAST
    )
    INDEX "ORDERS_PK_IDX"
    PCTFREE 10
    INITTRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT 262144
        MINSIZE 524288
        MAXSIZE 562949953159168
    )
    NOLOGGING
    TABLESPACE "MEM_TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;

```

자세한 내용은 각 SQL 객체에 대응하는 gsql 명령어를 참조한다.

주의

\ddl_table 등의 명령어를 사용하는 동시에 관련 SQL 객체에 DDL을 수행하면 서로 다른 결과를 출력할 수도 있으므로 동시에 DDL을 수행하지 않도록 한다.

History 제어

gsql은 대화형 모드에서 수행된 SQL 구문에 대한 이력을 관리한다.

표 33-13 참조 명령어

gsql 명령	설명
\\	이전 SQL을 수행한다.
\{n}	SQL 이력 번호에 해당하는 구문을 실행한다.
\history	SQL 수행 이력을 조회한다.
\set history	History 버퍼 개수를 제어한다.

다음은 이미 실행했던 SQL 구문을 다시 실행하는 예이다.

- 가장 최근에 성공적으로 실행한 SQL 구문을 실행한다.

```
gSQL> \\  
DUMMY  
-----  
X  
1 row selected.  
gSQL> \history  
ID SQL  
-----  
1 DROP TABLE IF EXISTS t1  
2 CREATE TABLE t1  
  (  
    id    INTEGER PRIMARY KEY,  
    name  VARCHAR(128),  
    addr  VARCHAR(128)  
  )  
3 CREATE INDEX t1_idx_name ON t1(name)  
4 INSERT INTO t1  
    VALUES ( 1, 'leekmo', 'Seoul, Korea' ),  
            ( 2, 'mkkim' , 'Seoul, Korea' ),  
            ( 3, 'xcom73', 'Inchon, Korea' )  
5 COMMIT  
6 select * from t1
```

- 이력에 존재하는 1번 SQL을 실행한다.

```
gSQL> \1  
Table dropped.
```

gsq의 history 관련 기능들은 **참조 명령어**의 명령어들을 참조한다.

SQL 편집

Text editor를 사용하여 SQL 구문을 편집한다. 이 때 사용할 text editor는 환경 변수 EDITOR에 지정할 수 있으며 환경 변수 EDITOR가 존재하지 않을 경우에는 기본적으로 vi를 사용한다.

표 33-14 참조 명령어

gsql 명령	설명
\edit	Text editor를 사용하여 SQL 구문을 편집한다.

편집 기능을 사용하면 gsql은 편집기를 실행하여 제어권을 넘긴다. 사용자는 편집기에서 SQL 구문 내용을 자유롭게 편집할 수 있으며 편집기를 종료하면 gsql에서 제어권을 돌려받은 후 해당 내용을 마지막 이력으로 추가한다.

이렇게 편집기를 통해서 편집된 SQL 구문은 가장 마지막 이력 실행 명령인 \를 통해서 실행할 수 있다.

주의

SQL 편집 기능은 단일 SQL 문장만 편집할 수 있다. 다중 SQL 구문을 실행할 경우, 오류가 발생한다.

\edit 명령을 사용하여 편집할 수 있는 내용은 다음과 같다.

- 가장 최근에 수행한 SQL 구문
- File에 저장된 SQL 구문
- gsql 이력에 저장된 SQL 구문

gsql 편집 관련 기능에 대한 자세한 내용은 **\edit**의 명령어들을 참조한다.

다음은 SQL 구문을 편집하는 예이다.

- 가장 최근에 실행한 SQL 구문을 편집한다.

```
gSQL> SELECT * FROM T1;
C1
--
1
11
2 rows selected.
gSQL> \EDIT
SELECT * FROM T1 WHERE C1 < 10;
```

- 편집된 SQL 구문을 실행한다.

```
gSQL> \
C1
--
1
1 row selected.
```

연결 제어

다음은 참조 명령어이다.

표 33-15 참조 명령어

gsql 명령	설명
<code>\connect</code>	새로운 사용자로 접속한다.
<code>\quit</code>	접속을 종료한다.

대화형 모드에서 `\connect` 명령을 이용해 다음과 같이 새로운 사용자로 접속할 수 있다. `\connect` 명령은 기존의 session을 종료하고 새로운 session을 생성한다. 자세한 내용은 `\connect` 명령을 참조한다.

```
gSQL> \connect test test
gSQL>
```

다음과 같이 `SET SESSION AUTHORIZATION user_identifier` 구문을 이용해 사용자를 변경할 수 있다. `\connect` 명령이 수행 중인 트랜잭션을 모두 COMMIT 하고 새로운 session을 생성하는 반면에 `SET SESSION AUTHORIZATION` 구문은 기존의 session을 그대로 유지한 채 사용자를 변경한다.

```
gSQL> SET SESSION AUTHORIZATION test;
Session set.
gSQL>
```

대화형 모드를 종료하려면 다음과 같이 `\quit` 명령을 사용한다. 자세한 내용은 `\quit` 명령을 참조한다.

```
gSQL> \quit
%
```

호스트 변수 사용

gsql은 대화형 모드에서 호스트 변수를 선언하고 호스트 변수에 값을 대입하며 SQL 구문에 호스트 변수를 함께 사용할 수 있다.

표 33-16 참조 명령어

gsql 명령	설명
<code>\var</code>	호스트 변수를 선언한다.
<code>\exec :var := value</code>	호스트 변수에 값을 할당한다.
<code>\print</code>	호스트 변수의 값을 출력한다.
<code>\dynamic sql :var</code>	호스트 변수에 저장된 SQL 문장을 수행한다.

다음은 `\var` 명령을 이용하여 호스트 변수를 선언하고 `\exec :var := value` 명령을 이용해 호스트 변수에 값을 대입하고 `\print` 명령을 이용해 호스트 변수의 값을 조회하는 예이다. 자세한 내용은 각 명령을 참조한다.

```
gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> \print v_id
V_ID
----
    1
gSQL>
```

대화형 모드에서 선언한 변수는 다음과 같이 SQL 구문 내에서 입력 인자로 사용할 수 있다.

```
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID NAME    ADDR
--
1 leekmo Seoul, Korea
1 row selected.
gSQL>
```

대화형 모드에서 선언한 변수는 다음과 같이 SQL 구문 내에서 출력 인자로 사용할 수 있다.

```
gSQL> SELECT id INTO :v_id FROM t1 WHERE name = 'mkkim';
V_ID
----
    2
1 row selected.
gSQL>
```

SQL 구문 처리 방식 제어

응용 프로그램을 작성하기 전에 gsql 대화형 모드의 명령어를 이용하여 SQL을 처리할 경우, 응용 프로그램에서 자주 사용하는 prepare/ execute 수행, cursor를 이용한 검색 등의 SQL 구문 처리 방식을 simulation 할 수 있다.

표 33-17 참조 명령어

gsql 명령	설명
<code>\exec sql</code>	SQL 문장을 바로 실행한다.
<code>\prepare sql</code>	SQL 문장을 준비한다.
<code>\exec</code>	준비된 SQL 문장을 실행한다.
<code>\dynamic sql :var</code>	호스트 변수에 저장된 SQL 문장을 실행한다.

다음 Java 응용 프로그램은 \$GOLDILOCKS_HOME/sample/JDBC/JdbcSample.java의 일부 코드 내용이다.

```
public static void main(String[] args) throws SQLException
{
    Connection con = createConnectionByDriverManager("TEST", "test");
    Statement stmt = con.createStatement();
    stmt.execute("CREATE TABLE SAMPLE_TABLE ( ID INTEGER, NAME CHAR(20) )");
    PreparedStatement pstmt = con.prepareStatement("INSERT INTO SAMPLE_TABLE VALUES (?,
?));");
    pstmt.setInt(1, 100);
    pstmt.setString(2, "Tom");
    pstmt.executeUpdate();
    pstmt.setInt(1, 200);
    pstmt.setString(2, "Jerry");
    pstmt.executeUpdate();
    ResultSet rs = stmt.executeQuery("SELECT * FROM SAMPLE_TABLE");
    while (rs.next())
    {
        System.out.println("ID = " + rs.getInt(1) + ": " + rs.getString(2));
    }
    rs.close();
    stmt.close();
    pstmt.close();
    con.close();
    Connection con2 = createConnectionByDataSource("TEST", "test");
    Statement stmt2 = con2.createStatement();
    stmt2.execute("DROP TABLE SAMPLE_TABLE");
    stmt2.close();
    con2.close();
}
```

다음과 같이 **\prepare sql** 명령과 **\exec** 명령을 이용하여 위에서 Java 코드의 PreparedStatement class를 이용하여 구현한 것을 simulation 할 수 있다.

```
gSQL> \connect test test
gSQL> \var v_int INTEGER
gSQL> \var v_string VARCHAR(128)
gSQL> CREATE TABLE SAMPLE_TABLE ( ID INTEGER, NAME CHAR(20) );
Table created.
gSQL> \prepare sql INSERT INTO SAMPLE_TABLE VALUES (:v_int, :v_string);
SQL prepared.
```

```

gSQL> \exec :v_int := 100
gSQL> \exec :v_string := 'Tom'
gSQL> \exec
1 row created.
gSQL> \exec :v_int := 200
gSQL> \exec :v_string := 'Jerry'
gSQL> \exec
1 row created.
gSQL> SELECT * FROM SAMPLE_TABLE;
 ID NAME
-----
100 Tom
200 Jerry
2 rows selected.
gSQL> \connect test test
gSQL> DROP TABLE SAMPLE_TABLE;
Table dropped.
gSQL> \quit
%
```

다음 embedded SQL 프로그램은 \$GOLDILOCKS_HOME/sample/EmbeddedSQL/sample2.gc의 일부 코드 내용이다.

```

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sENAME[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    ... 중략 ...
}
```

- Retrieve employee

```

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM EMP;
EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{

```

```

        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
    printf(" EMPNO      ENAME                      JOB      SALARY\n");
    printf("===== \n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO :sEmpNo, :sENAME, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }
        sRecordCount ++;
        printf("%6d %20s %10s %8ld\n",
            sEmpNo, sENAME.arr, sJob, sSalary);
    }
    printf("===== \n");
    printf("Record Count = %d\n", sRecordCount);
    printf("===== \n");
    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    ... 중략 ...
}

```

위의 embedded SQL 프로그램의 커서를 이용한 질의 처리 과정은 다음과 같이 gsql의 대화형 모드에서 simulation 할 수 있다.

```

gSQL> \var sEmpNo INTEGER
gSQL> \var sENAME VARCHAR(20)

```

```

gSQL> \var sJob CHAR(20)
gSQL> \var sSalary BIGINT
gSQL> DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;
Cursor declared.
gSQL> OPEN EMP_CUR;
Cursor is open.
gSQL> FETCH EMP_CUR
        INTO :sEmpNo, :sENAME, :sJob, :sSalary;
SEMPNO SENAME SJOB          SSALARY
-----
2854 Park  RND              800
1 row fetched.
gSQL> FETCH EMP_CUR
        INTO :sEmpNo, :sENAME, :sJob, :sSalary;
SEMPNO SENAME SJOB          SSALARY
-----
2098 Kim   SALESMAN         1600
1 row fetched.
gSQL> FETCH EMP_CUR
        INTO :sEmpNo, :sENAME, :sJob, :sSalary;
SEMPNO SENAME SJOB          SSALARY
-----
2175 Choi  SALESMAN         1250
1 row fetched.
gSQL> CLOSE EMP_CUR;
Cursor closed.
gSQL>

```

노트

다른 DBMS에서는 다음과 같은 SQL 구문을 embedded SQL에서만 제한적으로 사용할 수 있는 반면에 GO LDILOCKS에서는 embedded SQL 뿐만 아니라 ODBC나 JDBC를 이용한 응용 프로그램을 작성할 때 SQL 처리 함수들의 인자로 사용할 수 있다.

- SELECT .. INTO
- DECLARE cursor_name
- OPEN cursor_name

- FETCH cursor_name
- CLOSE cursor_name

SQL 실행 계획 정보

다음은 참조 명령어이다.

표 33-18 참조 명령어

gsql 명령	설명
<code>\explain plan</code>	SQL 문장의 실행 계획을 출력한다.
<code>\set autotrace</code>	실행 계획 출력 여부를 설정한다.

대화형 모드에서 다음과 같이 `\explain plan` 명령이나 `\set autotrace` 명령을 이용하여 SQL 구문의 실행 계획을 조회할 수 있다. 명령에 대한 자세한 내용은 `\explain plan`, `\set autotrace` 를 참조하고 실행 계획에 대한 해석은 [SQL 실행 계획](#)을 참조한다.

다음은 `\explain plan` 명령을 이용하여 TPC-H 벤치마크의 4번 질의를 수행하는 예이다.

```
\explain plan
select
  o_orderpriority,
  count(*) as order_count
from
  orders
where
  o_orderdate >= date '1993-07-01'
 and o_orderdate < date '1993-07-01' + interval '3' month
 and exists (
    select
      *
    from
      lineitem
    where
      l_orderkey = o_orderkey
      and l_commitdate < l_receiptdate
  )
group by
  o_orderpriority
order by
```

```
o_orderpriority;
```

```
O_ORDERPRIORITY ORDER_COUNT
```

```
-----
1-URGENT          10594
2-HIGH            10476
3-MEDIUM         10410
4-NOT SPECIFIED   10556
5-LOW             10487
```

```
5 rows selected.
```

```
>>> start print plan
```

```
< Execution Plan >
```

```
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
-----
|   0   |  SELECT STATEMENT                                |        |
|   1   |    SORT INSTANT ACCESS                          |    5   |
|   2   |      GROUP HASH INSTANT ACCESS                  |    5   |
|   3   |        NESTED LOOP JOIN (LEFT SEMI)             | 52523  |
|   4   |          TABLE ACCESS ("ORDERS")               | 57218  |
|   5   |            INDEX ACCESS ("LINEITEM, LINEITEM_PK_INDEX") | 52523  |
=====
```

```
1 - SORT KEY : "ORDERS.O_ORDERPRIORITY ASC NULLS LAST"
    RECORD COLUMNS : COUNT(*)
    READ COLUMNS : O_ORDERPRIORITY, COUNT(*)

2 - AGGREGATIONS : COUNT(*)
    GROUPING COLUMNS : O_ORDERPRIORITY
    RECORD COLUMNS : COUNT(*)
    READ COLUMNS : O_ORDERPRIORITY, COUNT(*)

3 - JOINED COLUMNS : ORDERS.O_ORDERPRIORITY

4 - READ COLUMNS : O_ORDERKEY, O_ORDERDATE, O_ORDERPRIORITY
    PHYSICAL FILTER : O_ORDERDATE >= CAST( '1993-07-01' AS DATE ) AND O_ORDERDATE < (
CAST( '1993-07-01' AS DATE ) + CAST( '3' AS INTERVAL(MONTH) ) )

5 - READ INDEX COLUMNS : L_ORDERKEY
    READ TABLE COLUMNS : L_COMMITDATE, L_RECEIPTDATE
    MIN RANGE : L_ORDERKEY = {O_ORDERKEY}
    MAX RANGE : L_ORDERKEY = {O_ORDERKEY}
    PHYSICAL TABLE FILTER : L_COMMITDATE < L_RECEIPTDATE
```

```
<<< end print plan
```

33.4 Command Option References

본 장에서는 셸 프롬프트 상에서 gsql 명령을 수행할 때 사용하는 option에 대해 설명한다.

Username Password

설명

사용자 이름과 password를 이용하여 GOLDILOCKS에 접속한다.

사용 예

다음은 사용자 test로 접속하는 예이다.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

다음은 사용자 정보나 password가 잘못되어 GOLDILOCKS 접속에 실패하는 예이다.

```
% gsql invalid_user invalid_password
ERR-28000(16004): invalid username/password; logon denied
%
```

--as {SYSDBA|ADMIN}

설명

SYSDBA role이나 ADMIN role로 GOLDILOCKS에 접속한다.
해당 role에 대한 자세한 내용은 **서버 구동 및 종료**를 참조한다.

```
% gsql sys gliese --as sysdba
```

사용 예

다음은 SYSDBA role로 접속하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

--conn-string

설명

Connection string을 통해 GOLDILOCKS에 접속한다.

기술할 connection string의 내용은 ODBC 함수 **SQLDriverConnect**의 input string과 동일한 형태로 사용해야 한다. Connection string에 대한 자세한 내용은 **SQLDriverConnect** 함수를 참조한다.

사용 예

다음은 --conn-string을 이용하여 접속하는 예이다.

```
% gsql --conn-string 'DSN=GOLDILOCKS;UID=test;PWD=test'
Connected to GOLDILOCKS Database.
gSQL>
```

--dsn

설명

odbc.ini 파일에 정의된 Data Source Name (DSN)을 이용하여 접속한다.

DSN 이름은 odbc.ini 파일에 정의된 이름이어야 한다. odbc.ini 파일에 대한 자세한 내용은 **UNIX에서 DSN 구성**을 참조한다.

DSN이 생략된 경우 기본값은 GOLDILOCKS이다.

사용 예

다음은 DSN을 이용해 접속하는 예이다.

```
% gsql test test --dsn GOLDILOCKS
Connected to GOLDILOCKS Database.
gSQL>
```

위의 예에서 사용한 odbc.ini 파일의 내용은 다음과 같다.

[GOLDILOCKS]

DATE_FORMAT = SYYYY-MM-DD

TIME_FORMAT = HH24:MI:SS.FF6

노트

gsq!은 odbc.ini 속성 중에 HOST, PORT를 인식하지 않는다.

--enable-color

설명

터미널 상에서 질의 결과의 각 row를 다른 색깔로 출력하여 쉽게 구분할 수 있도록 한다.

대화형 모드로 질의를 수행할 경우 질의 결과를 row마다 다른 색깔로 출력하며 이는 --import 옵션을 이용해 파일에 포함된 SELECT 구문의 결과에도 적용한다.

사용 예

다음은 --enable-color 옵션을 이용하여 gsql을 수행하고 대화형 모드에서 질의 결과를 출력하는 예이다.

```
% gsql test test --enable-color
Connected to GOLDILOCKS Database.
gSQL> SELECT id, addr FROM t1;
ID ADDR
--
1 Seoul, Korea
2 Seoul, Korea
3 Incheon, Korea
3 rows selected.
gSQL>
```

다음은 --import 옵션을 이용하여 파일 내에 포함된 SELECT 질의를 수행하는 예이다.

```
% gsql test test --import 'sample_select.sql' --enable-color
SELECT id, addr FROM t1;
ID ADDR
--
1 Seoul, Korea
```

2 Seoul, Korea

3 Incheon, Korea

3 rows selected.

%

--help

설명

gsql 프로그램의 옵션 목록을 간략히 보여준다.

사용 예

다음은 --help 옵션을 사용하는 예이다.

```
% gsql --help
Usage
    gsql [user_name [password]] [options]
Arguments:
    user_name      user name
    password       password
Options:
    --version              print version information and exit
    --import FILE          import sql FILE
    --no-prompt            suppresses the display of the banner and prompts
    --dsn DSN              dsn string (default is GOLDILOCKS)
    --conn-string 'CONN-STRING' connection string
    --prompt STRING        change prompt string
    --enable-color         enable colored text
    --as {SYSDBA|ADMIN}    privilege
    --silent               suppresses the display of the result message and echoing of
commands
    --help                 print help message
%
```

--import

설명

대화형 모드가 아닌 파일 내의 SQL 구문을 배치로 수행한다.

기술할 파일 이름은 single-quote (')로 묶어서 기술하며 절대 경로 또는 상대 경로를 사용할 수 있다.

사용 예

다음은 파일의 절대 경로를 이용하여 SQL 구문을 수행하는 예이다.

```
% gsql test test --import '/home/GOLDILOCKS/sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name  VARCHAR(128),
    addr  VARCHAR(128)
);
Table created.
CREATE INDEX t1_idx_name ON t1(name);
Index created.
INSERT INTO t1
VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
      ( 2, 'mkkim' , 'Seoul, Korea' ),
      ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
%
```

다음은 파일의 상대 경로를 이용하여 SQL 구문을 수행하는 예이다.

```
% gsql test test --import 'sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id    INTEGER PRIMARY KEY,
    name  VARCHAR(128),
```

```

        addr VARCHAR(128)
    );
Table created.
CREATE INDEX t1_idx_name ON t1(name);
Index created.
INSERT INTO t1
    VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
            ( 2, 'mkkim' , 'Seoul, Korea' ),
            ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
%
```

--no-prompt

설명

대화형 모드를 수행할 때 프롬프트를 출력하지 않는다.

사용 예

다음은 --no-prompt를 사용하는 예이다.

```

% gsql test test --no-prompt
SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

--prompt

설명

대화형 모드를 수행할 때 gsql의 프롬프트를 설정한다.

기본값은 gSQL이다.

특수문자나 공백을 포함할 경우, 다음과 같이 double-quote (")로 묶어서 기술한다.


```
% gsql test test --prompt "GOLDILOCKS Venus.3.2"
```

사용 예

다음은 --prompt 옵션을 이용하여 대화형 모드의 프롬프트를 GOLDILOCKS로 변경하는 예이다.

```
% gsql test test --prompt GOLDILOCKS
Connected to GOLDILOCKS Database.
GOLDILOCKS> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
GOLDILOCKS>
```

--silent

설명

SQL 구문의 수행 결과를 출력하지 않는다.

--import 명령을 이용해 파일을 읽어 대량으로 SQL 구문을 수행할 경우에 유용하다.

--silent 옵션은 대화형 모드로 동작할 때도 적용된다.

사용 예

다음은 \$GOLDILOCKS_HOME/admin 디렉토리에서 --silent 옵션을 이용하여 DictionarySchema.sql 파일을 수행하는 예이다.

```
% gsql sys gliese --as sysdba --import 'DictionarySchema.sql' --silent
%
```

--version

설명

gsql 프로그램의 version 정보를 출력한다.

GOLDILOCKS와 동일한 version의 gsql 프로그램을 사용할 것을 권장한다. GOLDILOCKS의 version은 다음과 같이 SQL 함수 **VERSION**을 통해 확인할 수 있다.

```
gSQL> SELECT version() FROM dual;
```

```
VERSION()
```

```
-----  
Release Name.X.X.X revision(XXXXX)
```

```
1 row selected.
```

사용 예

다음은 --version 옵션을 사용하는 예이다.

```
% gsql --version
```

```
%
```

33.5 Interactive Command References

대화형 모드에서 사용하는 gsql 고유 명령어는 모두 backslash (\)로 시작한다.

\\

구문

| \\

설명

가장 최근에 성공한 SQL 구문을 실행한다.

사용 예

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

- 가장 최근에 성공한 SQL 구문을 실행한다.

```
gSQL> \\
DUMMY
-----
X
1 row selected.
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
          *
ERROR at line 1:
```

- 실패한 SQL 구문이 아닌 가장 최근에 성공한 SQL 구문을 실행한다.

```
gSQL> \\  
DUMMY  
-----  
X  
1 row selected.
```

\connect

구문

```
\connect username password [as sysdba]
```

설명

입력된 username과 password로 새로 접속한다.
종료되지 않은 트랜잭션이 존재하는 경우 COMMIT을 수행한다.

사용 예

다음은 test 계정으로 접속하는 예이다.

```
gSQL> \connect test test  
gSQL>
```

다음과 같이 password가 잘못된 경우, 에러가 발생한다.

```
gSQL> \connect test invalid_password  
ERR-28000(16004): invalid username/password; logon denied  
gSQL> SELECT * FROM dual;  
ERR-08003(40044): connection does not exist
```

다음은 SYSDBA role로 접속하는 예이다.

```
gSQL> \connect sys gliese as sysdba  
gSQL>
```

\cstartup

구문

```
\cstartup
\cstartup nomount
\cstartup mount
\cstartup open
\cstartup local open
\cstartup global open
```

설명

Cluster 환경에서 GOLDDLOCKS 서버를 구동한다.

\cstartup 명령을 수행하려면 SYSDBA 또는 ADMIN role로 접속해야 한다. SYSDBA로 접속하는 방법은 **서버 구동 및 종료**를 참조한다.

- \cstartup nomount
 - 해당 서버를 NOMOUNT 단계로 시작한다.
- \cstartup mount
 - 해당 서버를 MOUNT 단계로 시작한다.
- \cstartup local open
 - 해당 서버 및 다른 서버를 LOCAL OPEN 단계로 시작한다.
- \cstartup global open
 - 모든 서버를 GLOBAL OPEN 단계로 시작한다.
- \cstartup open
 - \cstartup global open과 동일하다.
- \cstartup
 - \cstartup global open과 동일하다.

\cstartup local open 명령어와 \cstartup global open 명령어는 해당 서버와 다른 서버들도 해당 단계까지 구동시킨다. 이 외의 명령어는 해당 서버에만 적용되므로 **ALTER SYSTEM {MOUNT | OPEN} DATABASE**를 이용하여 단계별로 구동하려면 직접 서버를 LOCAL OPEN 단계까지 구동해야 한다.

서버의 구동 단계는 NOMOUNT, MOUNT, LOCAL OPEN, GLOBAL OPEN 단계로 구분되며 자세한 내용은 **다단계 시작**을 참조한다.

\cstartup 명령을 수행한 후에 다음 단계로 넘어가기 위해서는 **ALTER SYSTEM {MOUNT | OPEN} DATABASE** 구문을 수행해야 한다.

glocator를 이용한 \cstartup은 CSTARTUP과 CSHUTDOWN을 통해 확인할 수 있다.

노트

- CS 환경에서만 사용하는 명령어로서 gsqlnet에서만 사용할 수 있다.
- \cstartup nomount/mount 이외의 명령은 해당 서버와 함께 다른 서버에까지 영향을 미치는 명령어이다. 따라서 **odbc.ini 파일**에 LOCATOR_DSN 속성이 기술되지 않았을 경우, 에러가 반환된다.

사용 예

다음은 GOLDILOCKS를 구동하는 예이다.

```
% gsqlnet sys gliese --as sysdba
Connected to an idle instance.
gSQL> \cstartup
Startup success
```

\cshutdown

구문

```
\cshutdown
\cshutdown abort
\cshutdown normal
```

설명

Cluster 환경에서 GOLDILOCKS 서버를 종료한다.

\cshutdown 명령을 수행하려면 SYSDBA 또는 ADMIN role로 접속해야 한다. SYSDBA로 접속하는 방법은 **서버 구동 및 종료**를 참조한다.

- \cshutdown normal
 - 새로운 session의 접속을 차단하고 현재 접속된 모든 session의 종료를 기다린 후, checkpoint를 수행하고 서버를 종료한다.
- \cshutdown abort
 - 접속 중인 session들의 상태와 무관하게 바로 서버를 강제 종료한다.
- \cshutdown
 - \cshutdown normal과 동일하다.

glocator를 이용한 \cshutdown은 CSTARTUP과 CSHUTDOWN을 통해 확인할 수 있다.

노트

- CS 환경에서만 사용하는 명령어로서 gsqlnet에서만 사용할 수 있다.
- \cshutdown 명령은 해당 서버와 함께 다른 서버에까지 영향을 미치는 명령어이다. 따라서 **odbc.ini 파일**에 LOCATOR_DSN 속성이 기술되지 않았을 경우, 에러가 반환된다.

사용 예

다음은 GOLDILOCKS를 종료하는 예이다.

```
% gsqlnet sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \cshutdown
Shutdown success
gSQL>
```

\ddl_cluster

구문

```
\ddl_cluster
```

설명

Cluster system의 현재 상태에 해당하는 cluster DDL 구문을 출력한다.

Cluster system에서 유효한 구문이다.

사용 예

다음은 3 x 2로 구성된 cluster system에 대해 \ddl_cluster 명령을 수행하는 예이다.

```
gSQL> \ddl_cluster
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G1"
    CLUSTER MEMBER "G1N1" HOST '127.0.0.1' PORT '10110'
```

```

;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G1" ADD
    CLUSTER MEMBER "G1N2" HOST '127.0.0.1' PORT '10120'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G2"
    CLUSTER MEMBER "G2N1" HOST '127.0.0.1' PORT '10210'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G2" ADD
    CLUSTER MEMBER "G2N2" HOST '127.0.0.1' PORT '10220'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G3"
    CLUSTER MEMBER "G3N1" HOST '127.0.0.1' PORT '10310'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G3" ADD
    CLUSTER MEMBER "G3N2" HOST '127.0.0.1' PORT '10320'
;

```

\ddl_db

구문

```

\ddl_db
\ddl_db GRANT
\ddl_db COMMENT

```

설명

데이터베이스 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-19 \ddl_db 명령

명령어	설명
\ddl_db	데이터베이스에 생성된 모든 객체의 DDL 구문을 출력한다.
\ddl_db GRANT	데이터베이스 객체의 권한 정보에 해당하는 GRANT .. ON DATABASE 구문을 출력한다.
\ddl_db COMMENT	데이터베이스 객체의 주석 정보에 해당하는 COMMENT ON DATABASE 구문을 출력한다.

옵션이 없는 \ddl_db 명령을 수행하면 다음과 같은 순서로 DDL 구문을 출력한다.

- Database DDL
- Tablespace DDL
- Profile DDL
- Authorization DDL
- Schema DDL
- Authorization schema path DDL
- Database privilege DDL
- Tablespace privilege DDL
- Schema privilege DDL
- Public synonym DDL
- Table DDL
- Table privilege DDL
- Table option DDL
- Constraint DDL
- Index DDL
- View DDL
- View privilege DDL
- Sequence DDL
- Sequence privilege DDL
- Synonym DDL
- Stored procedure/ function DDL
- Audit policy DDL

단, dictionary 정보를 포함하고 있는 다음 스키마와 관련된 정보는 출력하지 않는다.

- DICTIONARY_SCHEMA
- INFORMATION_SCHEMA
- PERFORMANCE_VIEW_SCHEMA
- DEFINITION_SCHEMA
- FIXED_TABLE_SCHEMA

위의 dictionary 정보를 포함하는 schema에 대한 DDL 구문은 **\ddl_schema** 명령을 이용해 출력할 수 있다.

사용 예

다음은 \ddl_db 명령을 수행하는 예이다.

```
gSQL> \ddl_db
```

- Database DDL

```
SET SESSION AUTHORIZATION "SYS";
COMMENT
    ON DATABASE
    IS 'goldilocks database'
;
```

- Tablespace DDL

```
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test1.dbf'
    SIZE 10477568 REUSE
    ONLINE
    LOGGING
    EXTSIZE 262144
;
SET SESSION AUTHORIZATION "SYS";
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test2.dbf'
    SIZE 10477568 REUSE
;
SET SESSION AUTHORIZATION "SYS";
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test3.dbf'
    SIZE 10477568 REUSE
;
SET SESSION AUTHORIZATION "SYS";
COMMENT
    ON TABLESPACE "TEST_TBS"
    IS 'comment tablespace TPC data'
;
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY TEMPORARY TABLESPACE "TEMP_TBS"
    MEMORY 'test_mem'
```

```

        SIZE 10477568
    EXTSIZE 262144
;
... 중략 ...

```

- Sequence privilege DDL

```

SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "PUBLIC"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "TEST"
    WITH GRANT OPTION
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "C_USER"
;
SET SESSION AUTHORIZATION "C_USER";
GRANT
    USAGE ON SEQUENCE "C_USER"."C_SEQ"
    TO "TEST"
;
SET SESSION AUTHORIZATION "C_USER";
GRANT
    USAGE ON SEQUENCE "C_USER"."C_SEQ"
    TO "H_USER"
;

```

다음은 \ddl_db GRANT 명령을 수행하는 예이다.

```

gSQL> \ddl_db GRANT
SET SESSION AUTHORIZATION "SYS";
GRANT
    ALTER DATABASE ON DATABASE
    TO "TEST"
    WITH GRANT OPTION

```

```

;
SET SESSION AUTHORIZATION "SYS";
GRANT
    ALTER SYSTEM ON DATABASE
    TO "TEST"
    WITH GRANT OPTION
;
... 중략 ...

```

\ddl_tablespace

구문

```

\ddl_tablespace name
\ddl_tablespace name CREATE
\ddl_tablespace name ALTER
\ddl_tablespace name TABLE
\ddl_tablespace name CONSTRAINT
\ddl_tablespace name INDEX
\ddl_tablespace name GRANT
\ddl_tablespace name COMMENT

```

설명

테이블스페이스 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-20 \ddl_tablespace 명령

명령어	설명
\ddl_tablespace name	아래의 모든 옵션을 수행한다.
\ddl_tablespace name CREATE	테이블스페이스의 CREATE TABLESPACE 구문을 출력한다.
\ddl_tablespace name ALTER	테이블스페이스에 추가된 datafile 또는 memory에 대한 ALTER TABLESPACE 구문을 출력한다.
\ddl_tablespace name TABLE	테이블스페이스에 저장된 table들의 CREATE TABLE 구문을 출력한다.
\ddl_tablespace name CONSTRAINT	테이블스페이스에 저장된 constraint들의 ALTER TABLE 구문을 출력한다.
\ddl_tablespace name INDEX	테이블스페이스에 저장된 index들의 CREATE INDEX 구문을 출력한다.
\ddl_tablespace name GRANT	테이블스페이스의 권한 정보에 대한 GRANT .. ON TABLESPACE 구문을 출력한다.
\ddl_tablespace name COMMENT	테이블스페이스의 주석에 해당하는 COMMENT ON TABLESPACE 구문을 출력한다.

사용 예

다음은 \ddl_tablespace CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_tablespace test_tbs CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test1.dbf'
    SIZE 10477568 REUSE
    ONLINE
    EXTSIZE 262144
;
```

다음은 \ddl_tablespace ALTER 명령을 수행하는 예이다.

```
gSQL> \ddl_tablespace test_tbs ALTER
SET SESSION AUTHORIZATION "SYS";
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE
        '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test2.dbf'
        SIZE 10477568 REUSE
    ,
        '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test3.dbf'
        SIZE 10477568 REUSE
;
```

\ddl_profile

구문

```
\ddl_profile name
\ddl_profile name CREATE
\ddl_profile name COMMENT
```

설명

Profile 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-21 \ddl_profile 명령

명령어	설명
\ddl_profile name	아래의 모든 옵션을 수행한다.
\ddl_profile name CREATE	Profile 객체의 CREATE PROFILE 구문을 출력한다.
\ddl_profile name COMMENT	Profile의 주석에 해당하는 COMMENT ON PROFILE 구문을 출력한다.

사용 예

다음은 \ddl_profile CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_profile prof1 CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE PROFILE "PROF1" LIMIT
    FAILED_LOGIN_ATTEMPTS    DEFAULT
    PASSWORD_LOCK_TIME       1/86400
    PASSWORD_LIFE_TIME        UNLIMITED
    PASSWORD_GRACE_TIME       100
    PASSWORD_REUSE_MAX        DEFAULT
    PASSWORD_REUSE_TIME       DEFAULT
    PASSWORD_VERIFY_FUNCTION   KISA_VERIFY_FUNCTION
;
COMMIT;
```

\ddl_audit_policy

구문

```
\ddl_audit_policy name
\ddl_audit_policy name CREATE
\ddl_audit_policy name AUDIT
\ddl_audit_policy name COMMENT
```

설명

Audit policy 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-22 \ddl_audit_policy 명령

명령어	설명
\ddl_audit_policy name	아래의 모든 옵션을 수행한다.
\ddl_audit_policy name CREATE	Audit policy 객체의 CREATE AUDIT POLICY 구문을 출력한다.
\ddl_audit_policy name AUDIT	Audit policy 객체의 AUDIT POLICY 구문을 출력한다.
\ddl_audit_policy name COMMENT	Audit policy의 주석에 해당하는 COMMENT ON AUDIT POLICY 구문을 출력한다.

사용 예

다음은 \ddl_audit_policy CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_audit_policy p1 CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE AUDIT POLICY "P1"
    ACTIONS SELECT ON "PUBLIC"."T1"
        , INSERT ON "PUBLIC"."T1"
        , UPDATE ON "PUBLIC"."T2"
        , ALL ON "PUBLIC"."SEQ1"
        , SELECT ON "PUBLIC"."SEQ2"
        , EXECUTE ON "PUBLIC"."FUNC2"
;
COMMIT;
```

\ddl_auth

구문

```
\ddl_auth name
\ddl_auth name CREATE
\ddl_auth name SCHEMA PATH
\ddl_auth name SCHEMA
\ddl_auth name TABLE
\ddl_auth name CONSTRAINT
\ddl_auth name INDEX
\ddl_auth name VIEW
\ddl_auth name SEQUENCE
\ddl_auth name SYNONYM
\ddl_auth name PROCEDURE
\ddl_auth name COMMENT
```

설명

계정 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-23 \ddl_auth 명령

명령어	설명
\ddl_auth name	아래의 모든 옵션을 수행한다.
\ddl_auth name CREATE	사용자 객체의 CREATE USER 구문을 출력한다.
\ddl_auth name SCHEMA	사용자가 소유한 schema들의 CREATE SCHEMA 구문을 출력한다.
\ddl_auth name SCHEMA PATH	계정의 schema path에 해당하는 ALTER USER 구문을 출력한다.
\ddl_auth name TABLE	사용자가 소유한 table들의 CREATE TABLE 구문을 출력한다.
\ddl_auth name CONSTRAINT	사용자가 소유한 constraint들의 ALTER TABLE 구문을 출력한다.
\ddl_auth name INDEX	사용자가 소유한 index들의 CREATE INDEX 구문을 출력한다.
\ddl_auth name VIEW	사용자가 소유한 view들의 CREATE VIEW 구문을 출력한다.
\ddl_auth name SEQUENCE	사용자가 소유한 sequence들의 CREATE SEQUENCE 구문을 출력한다.
\ddl_auth name SYNONYM	사용자가 소유한 synonym들의 CREATE SYNONYM 구문을 출력한다.
\ddl_auth name PROCEDURE	사용자가 소유한 procedure와 function들의 CREATE PROCEDURE/FUNCTION 구문을 출력한다.
\ddl_auth name COMMENT	계정의 주석에 해당하는 COMMENT ON AUTHORIZATION 구문을 출력한다.

사용 예

다음은 \ddl_auth CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_auth h_user CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE USER "H_USER"
    IDENTIFIED BY H_USER
    DEFAULT TABLESPACE "TEST_TBS"
    TEMPORARY TABLESPACE "TEMP_TBS"
    INDEX TABLESPACE NULL
    WITHOUT SCHEMA
;
```

다음은 \ddl_auth SCHEMA 명령을 수행하는 예이다.

```
gSQL> \ddl_auth h_user SCHEMA
SET SESSION AUTHORIZATION "SYS";
CREATE SCHEMA "H_USER"
    AUTHORIZATION "H_USER"
;
```


다음은 \ddl_auth TABLE 명령을 수행하는 예이다.

```
gSQL> \ddl_auth h_user TABLE
SET SESSION AUTHORIZATION "H_USER";
CREATE TABLE "H_USER"."REGION"
(
    "R_REGIONKEY" NUMBER( 10, 0 )
    , "R_NAME" CHAR( 25 OCTETS )
    , "R_COMMENT" VARCHAR( 152 OCTETS )
)
PCTFREE 10
PCTUSED 60
INITTRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
)
TABLESPACE "TEST_TBS"
;
SET SESSION AUTHORIZATION "H_USER";
CREATE TABLE "H_USER"."NATION"
(
    "N_NATIONKEY" NUMBER( 10, 0 )
    , "N_NAME" CHAR( 25 OCTETS )
    , "N_REGIONKEY" NUMBER( 10, 0 )
    , "N_COMMENT" VARCHAR( 152 OCTETS )
)
PCTFREE 10
PCTUSED 60
INITTRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
```

```
)
TABLESPACE "TEST_TBS"
;
```

\ddl_schema

구문

```
\ddl_schema name
\ddl_schema name CREATE
\ddl_schema name TABLE
\ddl_schema name CONSTRAINT
\ddl_schema name INDEX
\ddl_schema name VIEW
\ddl_schema name SEQUENCE
\ddl_schema name SYNONYM
\ddl_schema name PROCEDURE
\ddl_schema name GRANT
\ddl_schema name COMMENT
```

설명

스키마 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-24 \ddl_schema 명령

명령어	설명
\ddl_schema name	아래의 모든 옵션을 수행한다.
\ddl_schema name CREATE	스키마 객체의 CREATE SCHEMA 구문을 출력한다.
\ddl_schema name TABLE	스키마에 속한 table들의 CREATE TABLE 구문을 출력한다.
\ddl_schema name CONSTRAINT	스키마에 속한 constraint들의 ALTER TABLE 구문을 출력한다.
\ddl_schema name INDEX	스키마에 속한 index들의 CREATE INDEX 구문을 출력한다.
\ddl_schema name VIEW	스키마에 속한 view들의 CREATE VIEW 구문을 출력한다.
\ddl_schema name SEQUENCE	스키마에 속한 sequence들의 CREATE SEQUENCE 구문을 출력한다.
\ddl_schema name SYNONYM	스키마에 속한 synonym들의 CREATE SYNONYM 구문을 출력한다.
\ddl_schema name PROCEDURE	스키마에 속한 procedure와 function들의 CREATE PROCEDURE/ FUNCTION 구문을 출력한다.
\ddl_schema name GRANT	스키마의 권한 정보에 대한 GRANT .. ON SCHEMA 구문을 출력한다.
\ddl_schema name COMMENT	스키마의 주석에 해당하는 COMMENT ON SCHEMA 구문을 출력한다.

사용 예

다음은 \ddl_schema CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_schema h_user CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE SCHEMA "H_USER"
    AUTHORIZATION "H_USER"
;
```

다음은 \ddl_schema CONSTRAINT 명령을 수행하는 예이다.

```
gSQL> \ddl_schema h_user CONSTRAINT
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."REGION"
    ADD CONSTRAINT "H_USER"."REGION_PK"
    PRIMARY KEY
    (
        "R_REGIONKEY" ASC NULLS LAST
    )
    INDEX "REGION_PK_INDEX"
    PCTFREE 10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT 262144
        MINSIZE 524288
        MAXSIZE 562949953159168
    )
    NOLOGGING
    TABLESPACE "TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."NATION"
    ADD CONSTRAINT "H_USER"."NATION_PK"
    PRIMARY KEY
    (
        "N_NATIONKEY" ASC NULLS LAST
```

```

)
INDEX "NATION_PK_INDEX"
PCTFREE 10
INITTRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
)
NOLOGGING
TABLESPACE "TEMP_TBS"
NOT DEFERRABLE
INITIALLY IMMEDIATE
;
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."SUPPLIER"
    ADD CONSTRAINT "H_USER"."SUPPLIER_PK"
    PRIMARY KEY
    (
        "S_SUPPKEY" ASC NULLS LAST
    )
INDEX "SUPPLIER_PK_INDEX"
PCTFREE 10
INITTRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
)
NOLOGGING
TABLESPACE "TEMP_TBS"
NOT DEFERRABLE
INITIALLY IMMEDIATE
;

```

\ddl_publicsynonym

구문

```
\ddl_public_synonym name
\ddl_public_synonym name CREATE
```

설명

Public synonym 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-25 \ddl_public_synonym 명령

명령어	설명
\ddl_public_synonym	아래의 모든 옵션을 수행한다.
\ddl_public_synonym name CREATE	Public synonym 객체의 CREATE PUBLIC SYNONYM 구문을 출력한다.

사용 예

다음은 \ddl_public_synonym CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_public_synonym pubsyn CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE PUBLIC SYNONYM "PUBSYN" FOR "PUBLIC"."T1"
;
COMMIT;
```

\ddl_table

구문

```
\ddl_table name
\ddl_table name CREATE
\ddl_table name CONSTRAINT
\ddl_table name INDEX
\ddl_table name IDENTITY
\ddl_table name SUPPLEMENTAL
\ddl_table name GRANT
\ddl_table name COMMENT
```

설명

테이블 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-26 \ddl_table 명령

명령어	설명
\ddl_table name	아래의 모든 옵션을 수행한다.
\ddl_table name CREATE	테이블 객체의 CREATE TABLE 구문을 출력한다.
\ddl_table name CONSTRAINT	테이블에 생성된 constraint들의 ALTER TABLE 구문을 출력한다.
\ddl_table name INDEX	테이블에 생성된 index들의 CREATE INDEX 구문을 출력한다.
\ddl_table name IDENTITY	테이블이 identity column을 가질 경우 restart 값에 대한 ALTER TABLE 구문을 출력한다.
\ddl_table name SUPPLEMENTAL	테이블에 supplemental log 옵션이 설정된 경우 ALTER TABLE 구문을 출력한다.
\ddl_table name GRANT	테이블의 권한 정보에 대한 GRANT .. ON TABLE 구문을 출력한다.
\ddl_table name COMMENT	테이블의 주석에 해당하는 COMMENT ON TABLE 구문을 출력한다.

사용 예

다음은 \ddl_table CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_table h_user.orders CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE TABLE "H_USER"."ORDERS"
(
    "O_ORDERKEY" NUMBER( 10, 0 )
    , "O_CUSTKEY" NUMBER( 10, 0 )
    , "O_ORDERSTATUS" CHAR( 1 OCTETS )
    , "O_TOTALPRICE" NUMBER( 12, 2 )
    , "O_ORDERDATE" DATE
    , "O_ORDERPRIORITY" CHAR( 15 OCTETS )
    , "O_CLERK" CHAR( 15 OCTETS )
    , "O_SHIPPRIORITY" NUMBER( 10, 0 )
    , "O_COMMENT" VARCHAR( 79 OCTETS )
)
PCTFREE 10
PCTUSED 60
INITRANS 4
MAXTRANS 8
STORAGE
(
    INITIAL 524288
```

```

        NEXT      262144
        MINSIZE 524288
        MAXSIZE 562949953159168
    )
    TABLESPACE "TEST_TBS"
;

```

다음은 \ddl_table GRANT 명령을 수행하는 예이다.

```

gSQL> \ddl_table h_user.nation GRANT
SET SESSION AUTHORIZATION "H_USER";
GRANT
    DELETE ON TABLE "H_USER"."NATION"
    TO "TEST"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    SELECT ( "N_NATIONKEY" ) ON TABLE "H_USER"."NATION"
    TO "C_USER"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    SELECT ( "N_NAME" ) ON TABLE "H_USER"."NATION"
    TO "C_USER"
;

```

\ddl_constraint

구문

```

\ddl_constraint name
\ddl_constraint name ALTER
\ddl_constraint name COMMENT

```

설명

제약 조건 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-27 \ddl_constraint 명령

명령어	설명
\ddl_constraint name	아래의 모든 옵션들을 수행한다.
\ddl_constraint name ALTER	제약 조건 객체의 ALTER TABLE 구문을 출력한다.
\ddl_constraint name COMMENT	제약 조건의 주석에 해당하는 COMMENT ON CONSTRAINT 구문을 출력한다.

사용 예

다음은 \ddl_constraint ALTER 명령을 수행하는 예이다.

```
gSQL> \ddl_constraint h_user.lineitem_pk ALTER
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."LINEITEM"
  ADD CONSTRAINT "H_USER"."LINEITEM_PK"
  PRIMARY KEY
  (
    "L_ORDERKEY" ASC NULLS LAST
    , "L_LINENUMBER" ASC NULLS LAST
  )
  INDEX "LINEITEM_PK_INDEX"
  PCTFREE 10
  INITRANS 4
  MAXTRANS 8
  STORAGE
  (
    INITIAL 524288
    NEXT 262144
    MINSIZE 524288
    MAXSIZE 562949953159168
  )
  NOLOGGING
  TABLESPACE "TEMP_TBS"
  NOT DEFERRABLE
  INITIALLY IMMEDIATE
;
```


\ddl_index

구문

```
\ddl_index name
\ddl_index name CREATE
\ddl_index name COMMENT
```

설명

인덱스 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-28 \ddl_index 명령

명령어	설명
\ddl_index name	아래의 모든 옵션을 수행한다.
\ddl_index name CREATE	인덱스 객체의 CREATE INDEX 구문을 출력한다.
\ddl_index name COMMENT	인덱스의 주석에 해당하는 COMMENT ON INDEX 구문을 출력한다.

사용 예

다음은 \ddl_index CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_index public.idx2 CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE INDEX "PUBLIC"."IDX2"
  ON "PUBLIC"."T1"
  (
    "C3" ASC NULLS LAST
    , "C1" DESC NULLS FIRST
  )
PCTFREE 10
INITRANS 4
MAXTRANS 8
STORAGE
(
  INITIAL 524288
  NEXT 262144
  MINSIZE 524288
  MAXSIZE 562949953159168
```

```

    )
    NOLOGGING
    TABLESPACE "MEM_TEMP_TBS"
;

```

\ddl_view

구문

```

\ddl_view name
\ddl_view name CREATE
\ddl_view name GRANT
\ddl_view name COMMENT

```

설명

View 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-29 \ddl_view 명령

명령어	설명
\ddl_view name	아래의 모든 옵션을 수행한다.
\ddl_view name CREATE	View 객체의 CREATE VIEW 구문을 출력한다.
\ddl_view name GRANT	View의 권한 정보에 대한 GRANT .. ON TABLE 구문을 출력한다.
\ddl_view name COMMENT	View의 주석에 해당하는 COMMENT ON TABLE 구문을 출력한다.

사용 예

다음은 \ddl_view CREATE 명령을 수행하는 예이다.

```

gSQL> \ddl_view h_user.revenue CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE OR REPLACE FORCE VIEW "H_USER"."REVENUE"
    (supplier_no, total_revenue)
AS SELECT
    l_suppkey,
    ROUND( sum(l_extendedprice * (1 - l_discount)), 2)
FROM
    lineitem
WHERE

```

```

        l_shipdate >= date '1996-01-01'
    AND l_shipdate < date '1996-01-01' + interval '3' month
GROUP BY
        l_suppkey
;

```

\ddl_sequence

구문

```

\dddl_sequence name
\dddl_sequence name CREATE
\dddl_sequence name RESTART
\dddl_sequence name GRANT
\dddl_sequence name COMMENT

```

설명

시퀀스 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-30 \ddl_sequence 명령

명령어	설명
\ddl_sequence name	아래의 모든 옵션을 수행한다.
\ddl_sequence name CREATE	시퀀스 객체의 CREATE SEQUENCE 구문을 출력한다.
\ddl_sequence name RESTART	시퀀스 객체의 restart 값에 해당하는 ALTER SEQUENCE 구문을 출력한다.
\ddl_sequence name GRANT	시퀀스의 권한 정보에 대한 GRANT .. ON SEQUENCE 구문을 출력한다.
\ddl_sequence name COMMENT	시퀀스의 주석에 해당하는 COMMENT ON SEQUENCE 구문을 출력한다.

사용 예

다음은 \ddl_sequence CREATE 명령을 수행하는 예이다.

```

gSQL> \ddl_sequence h_user.h_seq CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE SEQUENCE "H_USER"."H_SEQ"
    START WITH 1
    INCREMENT BY 1
    MAXVALUE 9223372036854775807
    MINVALUE 1

```

```
NO CYCLE
CACHE 20
;
```

다음은 \ddl_sequence RESTART 명령을 수행하는 예이다.

```
gSQL> \ddl_sequence h_user.h_seq RESTART
SET SESSION AUTHORIZATION "H_USER";
ALTER SEQUENCE "H_USER"."H_SEQ"
    RESTART WITH 21
;
```

\ddl_synonym

구문

```
\ddl_synonym name
\ddl_synonym name CREATE
```

설명

Synonym 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-31 \ddl_synonym 명령

명령어	설명
\ddl_synonym	아래의 모든 옵션을 수행한다.
\ddl_synonym name CREATE	Synonym 객체의 CREATE SYNONYM 구문을 출력한다.

사용 예

다음은 \ddl_synonym CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_synonym syn CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE SYNONYM "PUBLIC"."SYN" FOR "PUBLIC"."T1"
;
COMMIT;
```

\ddl_procedure

구문

```
\ddl_procedure name
\ddl_procedure name CREATE
```

설명

Stored procedure나 function 객체의 현재 상태에 해당하는 DDL 구문을 출력한다.

표 33-32 \ddl_procedure 명령

명령어	설명
\ddl_procedure	아래의 모든 옵션을 수행한다.
\ddl_procedure name CREATE	Stored procedure/ function 객체의 CREATE PROCEDURE/ FUNCTION 구문을 출력한다.

사용 예

다음은 \ddl_procedure CREATE 명령을 수행하는 예이다.

```
gSQL> \ddl_procedure proc1 CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE OR REPLACE PROCEDURE "PUBLIC"."PROC1"
is
begin
  null;
end;
/
COMMIT;
```

\desc

구문

```
\desc table_name
\desc schema_name.table_name
```

설명

테이블의 구조 정보를 조회한다.

테이블은 다음과 같이 테이블 이름만 기술하거나 테이블 이름을 스키마 이름과 함께 기술할 수 있다. 스키마 이름을 명시하지 않은 경우 테이블의 스키마 이름은 사용자의 **Schema Path**에 의해 결정된다.

- \desc t1
- \desc public.t1

테이블을 생성할 때 다음과 같이 **delimited identifier**를 이용해 테이블 이름을 생성한 경우, double quote (")를 사용해 기술한다.

- 테이블 생성
 - CREATE TABLE "TaBle#@^*" (id INTEGER);
- \desc "TaBle#@^"
- \desc "PUBLIC"."TaBle#@^"

수행 결과는 해당 테이블의 다음과 같은 정보를 포함한다.

- Column 정보
 - Column 이름
 - 데이터 타입
 - NULL 가능 여부
- 인덱스 정보
 - 인덱스 이름
 - 인덱스 저장 공간
 - 인덱스 타입
 - UNIQUE 여부
 - Key column 이름
- 제약 조건 정보
 - 제약 조건 이름
 - 제약 조건 유형
 - 관련 인덱스
 - 제약 조건 column

사용 예

다음은 t1 테이블의 정보를 조회하는 예이다.

```
gSQL> \desc t1
COLUMN_NAME TYPE                IS_NULLABLE
```

```

ID          NUMBER(10,0)          FALSE
NAME        VARCHAR(128)          TRUE
ADDR        VARCHAR(1024)         TRUE
INDEX_NAME   TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-----
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS   BTREE     TRUE     ID
T1_IDX_NAME      MEM_TEMP_TBS   BTREE     FALSE    NAME
CONSTRAINT_NAME  CONSTRAINT_TYPE ASSOCIATED_INDEX  COLUMNS
-----
T1_PRIMARY_KEY   PRIMARY KEY      T1_PRIMARY_KEY_INDEX ID

```

다음은 스키마 이름 public과 함께 기술하여 t1 테이블의 정보를 조회하는 예이다.

```

gSQL> \desc public.t1
COLUMN_NAME TYPE                IS_NULLABLE
-----
ID          NUMBER(10,0)          FALSE
NAME        VARCHAR(128)          TRUE
ADDR        VARCHAR(1024)         TRUE
INDEX_NAME   TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-----
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS   BTREE     TRUE     ID
T1_IDX_NAME      MEM_TEMP_TBS   BTREE     FALSE    NAME
CONSTRAINT_NAME  CONSTRAINT_TYPE ASSOCIATED_INDEX  COLUMNS
-----
T1_PRIMARY_KEY   PRIMARY KEY      T1_PRIMARY_KEY_INDEX ID

```

다음은 delimited identifier를 사용하여 생성한 테이블의 정보를 조회하는 예이다.

```

gSQL> CREATE TABLE "TaBle#@^*" ( id INTEGER );
Table created.
gSQL> \desc "TaBle#@^*"
COLUMN_NAME TYPE                IS_NULLABLE
-----
ID          NUMBER(10,0) TRUE

```

다음과 같이 delimited identifier를 사용하지 않은 경우, 에러가 발생한다.

```

gSQL> \desc TaBle#@^*
ERR-42000(40000): syntax error
\desc TaBle#@^*
.....^ ^

```

Error at line 1

\dynamic sql :var

구문

\dynamic sql :var

설명

호스트 변수 var에 저장된 SQL 문장을 수행한다.

Embedded SQL의 정의되지 않은 SQL 문장을 수행하는 **Embedded Dynamic SQL**과 유사한 개념으로써 정의되지 않은 SQL 문장을 수행할 때 사용한다.

다음과 같은 절차로 수행한다.

1. 호스트 변수를 선언한다. (**\var**를 참조한다.)
`\var var_stmt VARCHAR(1024)`
2. 호스트 변수의 값으로 SQL 문장을 대입한다. (**\exec :var := value**를 참조한다.)
`\exec :var_stmt := 'SELECT # FROM t1'`
3. Dynamic SQL을 수행한다.
`\dynamic sql :var_stmt`

사용 예

다음은 var_stmt 호스트 변수를 선언하고 SQL 문장을 대입하여 dynamic SQL을 실행하는 예이다.

- 호스트 변수에 SELECT 문장을 대입한다.

```
gSQL> \exec :var_stmt := 'SELECT * FROM t1'
```

- Dynamic SQL을 수행한다.

```
gSQL> \dynamic sql :var_stmt
ID NAME      ADDR
--
1 leekmo Seoul, Korea
1 row selected.
```

다음 INSERT 문장과 같이 SQL 문장에 문자열이 존재할 경우 명령어 **\exec :var := value**를 사용할 때와 같이 sin

gle quote (')를 두 번 기술한다.

- 호스트 변수를 선언한다.

```
gSQL> \var var_stmt VARCHAR(1024)
```

- 호스트 변수에 INSERT 문장을 대입한다.

```
gSQL> \exec :var_stmt := 'INSERT INTO t1(id, name, addr) VALUES ( 1, ''leekmo'', ''Seoul, Korea'' )'
```

- Dynamic SQL을 수행한다.

```
gSQL> \dynamic sql :var_stmt
1 row created.
```

\edit

설명

Text editor를 사용하여 SQL 구문을 편집한다. 이 때 사용할 text editor는 환경 변수 EDITOR에 지정할 수 있으며 환경 변수 EDITOR가 존재하지 않을 경우에는 기본적으로 vi를 사용한다.

편집 기능을 사용하게 되면 gsql은 편집기를 실행하여 제어권을 넘긴다. 사용자는 편집기에서 SQL 구문 내용을 자유롭게 편집할 수 있고 편집기를 종료하게 되면 gsql에서 제어권을 돌려 받은 후에 해당 내용을 마지막 이력으로 추가한다.

이렇게 편집기를 통해서 편집된 SQL 구문은 가장 마지막 이력 실행 명령인 \를 통해서 실행할 수 있다.

주의

단일 SQL 구문만 SQL 편집 기능을 사용하여 편집할 수 있다. 다중 SQL 구문을 실행할 경우 오류가 발생한다.

\edit 명령을 사용하여 편집할 수 있는 내용은 다음과 같다.

- 가장 최근에 수행한 SQL 구문
- File에 저장된 SQL 구문
- gsql 이력에 저장된 SQL 구문

다음 절에서 각각에 대한 사용 방법을 설명한다.

\edit

구문

```
\edit
\ed
```

설명

Text editor를 사용하여 가장 마지막에 수행한 SQL 구문을 편집한다. 만약 아무런 SQL도 수행한 적이 없다면 아무 내용도 없는 상태로 text editor가 실행된다.

사용 예

다음은 가장 최근에 수행한 SQL 구문을 편집하는 예이다.

- 가장 최근에 수행한 SQL 구문을 편집한다.

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \EDIT
SELECT * FROM T1 WHERE C1 < 10;
```

- 편집된 SQL 구문을 실행한다.

```
gSQL> \
C1
--
 1
1 row selected.
```

\edit 'file_name'

구문

```
\edit 'file_name'
\ed  'file_name'
```

설명

Text editor를 사용하여 주어진 file_name을 편집한다.

file_name은 single quote (') 내에 기술한다.

file_name은 다음과 같이 절대 경로 또는 상대 경로를 사용할 수 있으며 상대 경로를 사용할 경우 gsql을 수행한 경로를 기준으로 file_name을 찾는다.

- 절대 경로 사용

```
gSQL> \edit '/home/GOLDILOCKS/sample.sql'
```

- 상대 경로 사용

```
gSQL> \edit 'sample.sql'
```

사용 예

다음은 file에 저장된 SQL 구문을 편집하는 예이다.

- 가장 최근에 수행한 SQL 구문을 편집한다.

```
gSQL> SELECT * FROM T1;
C1
--
1
11
2 rows selected.
gSQL> \EDIT 'sample.sql'
SELECT * FROM T1 WHERE C1 < 10;
```

- 편집된 SQL 구문을 실행한다.

```
gSQL> \\\
C1
--
1
1 row selected.
```

\edit [history] {n}

구문

```
\edit [history] {n}
\ed  [history] {n}
```

설명

\history 명령으로 조회할 수 있는 SQL 수행 이력 중에서 number에 해당하는 구문을 편집한다.
number 값은 **\history** 수행 결과의 ID 값이어야 한다.

사용 예

gsql의 이력에 저장된 SQL 구문을 편집하는 예이다.

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \history
ID SQL
-- -----
 1 SELECT * FROM T1
```

- 저장된 이력 중 ID가 1인 내용을 편집한다.

```
gSQL> \edit 1
SELECT * FROM T1 WHERE C1 < 5;
```

- 편집된 SQL 구문을 실행한다.

```
gSQL> \
C1
--
 1
1 row selected.
```

\exec

구문

\exec

설명

\prepare sql 명령에 의해 준비된 SQL 구문을 수행한다.

\exec 명령은 ODBC의 **SQLExecute** 함수와 JDBC의 PreparedStatement:: **execute** 함수와 유사하게 동작한다. 준비된 SQL 구문은 \exec 명령을 통해 반복적으로 수행할 수 있다.

사용 예

다음은 INSERT 구문을 준비하고 반복적으로 수행하는 예이다.

- INSERT 구문을 prepare 한다.

```
gSQL> \prepare sql INSERT INTO t1(id, addr) VALUES ( seq.NEXTVAL, 'N/A' );
SQL prepared.
```

- Prepare 된 구문을 수행한다.

```
gSQL> \exec
1 row created.
```

- Prepare 된 구문을 수행한다.

```
gSQL> \exec
1 row created.
```

- 테이블 조회 결과는 다음과 같다.

```
gSQL> SELECT * FROM t1;
ID ADDR
-- ----
1 N/A
2 N/A
2 rows selected.
```

다음은 호스트 변수와 함께 SELECT 구문을 prepare 하고 호스트 변수의 값을 바꿔가며 반복 수행하는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v_id INTEGER
```

- 호스트 변수를 사용한 SELECT 구문을 prepare 한다.

```
gSQL> \prepare sql SELECT * FROM t1 WHERE ID = :v_id;
SQL prepared.
```

- 호스트 변수에 1 값을 대입한다.

```
gSQL> \exec :v_id := 1
```

- 준비된 SELECT 구문을 수행한다.

```
gSQL> \exec
ID ADDR
-- ----
1 N/A
1 row selected.
```

- 호스트 변수에 2 값을 대입한다.

```
gSQL> \exec :v_id := 2
```

- 준비된 SELECT 구문을 수행한다.

```
gSQL> \exec
ID ADDR
-- ----
2 N/A
1 row selected.
```

\exec :var := value

구문

```
\exec :variable := <value expression>
```

설명

호스트 변수에 값을 대입한다.

해당 명령은 다음과 같은 **SELECT .. INTO** 구문을 수행한 것과 동일한 방식으로 호스트 변수에 값을 대입한다.

```
gSQL> \exec :v_value := 1234
gSQL> SELECT 1234 INTO :v_value FROM DUAL;
```

호스트 변수는 **\var** 명령을 통해 선언되어 있어야 한다.

호스트 변수는 colon (:)과 함께 사용하여야 하며, 대입 연산자 (:=)에서 (:)을 생략하면 안된다. 호스트 변수에 입력되는 <value expression>에는 단순한 값이나 연산을 사용할 수 있다. 자세한 내용은 **Expressions**를 참조한다.

호스트 변수에 입력되는 값은 호스트 변수의 데이터 타입에 입력 가능한 값이어야 한다. 자세한 내용은 **타입간 변환**을 참조한다.

입력된 호스트 변수의 값은 **\print** 명령을 이용해 조회할 수 있다.

```
\exec :v_value := 1234
\print v_value
```

호스트 변수에 문자열을 대입할 경우 다음과 같이 문자열을 single quote (')로 묶어서 입력한다.

- 단순 문자열
 - 값: abcd
 - \exec :v_value := 'abcd'
- Single quote가 포함된 문자열
 - 값: Tom's House
 - 문자열 안의 single quote (')는 다음과 같이 single quote (')를 두 번 사용한다.
 - \exec :v_value := 'Tom''s House'
- Single quote가 포함된 SQL 문장
 - 값: INSERT INTO t1 VALUES (1, 'Tom''s House')
 - 문자열 안의 single quote (')는 다음과 같이 single quote (')를 두 번 사용한다.
 - \exec :v_value := 'INSERT INTO t1 VALUES (1, "Tom''s House")'

사용 예

다음은 다양한 형태의 문자열을 호스트 변수에 대입하는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v_value VARCHAR(1024)
```

- 단순 문자열을 대입한다.

```
gSQL> \exec :v_value := 'abcd'
gSQL> \print v_value
V_VALUE
-----
abcd
```

- Single quote (')가 포함된 문자열을 대입한다.

```
gSQL> \exec :v_value := 'Tom's House'
gSQL> \print v_value
V_VALUE
-----
Tom's House
```

- Single quote (')가 포함된 SQL 문장을 대입한다.

```
gSQL> \exec :v_value := 'INSERT INTO t1 VALUES ( 1, 'Tom's House' )'
gSQL> \print v_value
V_VALUE
-----
INSERT INTO t1 VALUES ( 1, 'Tom's House' )
```

다음은 호스트 변수에 값을 할당할 때 연산결과를 사용하는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v1 INTEGER
gSQL> \var v2 INTEGER
```

- v1에 값을 할당한다.

```
gSQL> \exec :v1 := 100
```

- v1과의 연산을 수행한 값을 v2에 할당한다.

```
gSQL> \exec :v2 := :v1 + 1000
```

- 호스트 변수 v1과 v2의 값을 조회한다.

```
gSQL> \print v1
V1
---
100
```



```
gSQL> \print v2
V2
----
1100
```

다음은 호스트 변수에 값을 할당하고 호스트 변수를 SELECT 구문에 사용하는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v_id INTEGER
```

- 호스트 변수에 값을 할당한다.

```
gSQL> \exec :v_id := 1
```

- 호스트 변수를 이용하여 SELECT 구문을 수행한다.

```
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID ADDR
-- ----
1 N/A
1 row selected.
```

다음은 SELECT .. INTO 구문을 통해 호스트 변수에 값을 얻어오는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v_id INTEGER
```

- 어떠한 값도 설정되어 있지 않다.

```
gSQL> \print v_id
V_ID
----
null
```

- SELECT INTO 구문 수행을 통해 v_id에 값을 할당한다.

```
gSQL> SELECT MAX(id) INTO :v_id FROM t1;
V_ID
----
2
1 row selected.
```

- 호스트 변수의 값을 확인한다.

```
gSQL> \print v_id
V_ID
----
  2
```

\exec sql

구문

```
\exec sql <sql_statement>
```

설명

<sql_statement>에 기술된 SQL 구문을 실행한다.

다음과 같이 프롬프트 상에서 SQL 구문을 수행한 것과 동일하게 동작한다.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
gSQL> \exec sql SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

SQL 구문을 수행할 때 **\prepare sql** 명령과 **\exec** 명령을 이용하여 PREPARE/EXECUTE 방식으로 실행하는 것과 달리, **\exec sql** 명령은 ODBC의 **SQLExecDirect** 함수와 JDBC 의 Statement:: **execute** 함수에 대응하는 DIRECT EXECUTE 방식으로 SQL 구문을 수행한다.

사용 예

다음은 SELECT 구문을 수행하는 간단한 예이다.

```
gSQL> \exec sql SELECT * FROM dual;
DUMMY
```

```
-----
X
1 row selected.
```

\explain plan

구문

```
\explain plan <sql_statement>
\explain plan on <sql_statement>
\explain plan only <sql_statement>
```

설명

<sql_statement>에 기술된 SELECT 구문 등의 실행 계획을 출력한다.

- \explain plan on
 - SQL 구문을 수행하고 질의 결과와 함께 실행 계획을 출력한다.
- \explain plan only
 - SQL 구문을 수행하지 않고 질의 결과 없이 실행 계획을 출력한다.
- \explain plan
 - on이나 only를 명시하지 않은 경우, 기본값은 on이다.

실행 계획에 대한 자세한 내용은 **SQL 실행 계획**을 참조한다.

사용 예

다음과 같이 ON과 함께 사용할 경우 SQL 구문을 수행하고 질의 결과와 함께 실행 계획을 출력한다.

```
gSQL> \explain plan on SELECT * FROM t1 WHERE id = 1;
ID NAME      ADDR
--  -----
1 leekmo Seoul, Korea
1 row selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                               |      ROWS  |
=====
```

```

| 0 | SELECT STATEMENT | |
| 1 | TABLE ACCESS ("T1") | 1 |
=====
1 - READ COLUMNS : ID, NAME, ADDR
    PHYSICAL FILTER : ID = 1
<<< end print plan

```

다음과 같이 ONLY를 사용할 경우 SQL 구문을 수행하지 않고 질의 결과 없이 실행 계획만 출력한다.

```

gSQL> \explain plan only SELECT * FROM t1 WHERE id = 1;
>>> start print plan
< Execution Plan >
=====
| IDX | NODE DESCRIPTION | ROWS |
=====
| 0 | SELECT STATEMENT | |
| 1 | TABLE ACCESS ("T1") | 0 |
=====
1 - READ COLUMNS : ID, NAME, ADDR
    PHYSICAL FILTER : ID = 1
<<< end print plan

```

\help

구문

```
\help
```

설명

gsql 대화형 모드에서 (\)로 시작하는 명령어 목록을 간략히 보여준다.

사용 예

다음은 \help 명령을 사용하는 예이다.

```

gSQL> \help
\help
\q[uit]

```

```

\i[mport] {'FILE'}          Import SQL
\ed[it] [{'FILE'}[HISTORY] num}] Edit SQL statement
\\                          Executes the most recent history entry
\{n}                        Executes n'th history entry
\hi[story]                  Show history entries
\desc    {[schema.]table_name} Show table description
\idesc   {[schema.]index_name} Show index description
\spo[ol] ['filename' | OFF]  Stores query results in a file
\ho[st]  [command]          Executes an operating system command

\set vertical    {ON|OFF}
\set time        {ON|OFF}
\set timing      {ON|OFF}
\set color       {ON|OFF}
\set error       {ON|OFF}
\set autocommit  {ON|OFF}
\set autotrace   {ON|TRACEONLY|OFF}
\set serveroutput {ON|OFF}
\set heading     {ON|OFF}
\set linesize    {n}        0 < n <= 100000
\set pagesize    {n}        0 < n <= 100000
\set colsize     {n}        0 < n <= 104857600
\set numsize     {n}        0 < n <= 50
\set ddlsize     {n}        0 < n <= 100000
\set history     {n}        n <= 100000 ( if n < 0, clear history buffer )
\var             {host_var_name} {INTEGER|BIGINT|VARCHAR(n)}
\exec            [{:host_var_name} := {constant}]
\exec sql        {sql string}
\prepare sql     {sql string}
\dynamic sql     {host_var_name}
\explain plan    [{ON|ONLY}] {sql string}
\print           [{host_var_name}]

\ddl_db
\ddl_tablespace {name}
\ddl_auth       {name}
\ddl_schema     {name}
\ddl_table      {[schema.]name}
\ddl_constraint {[schema.]name}
\ddl_index      {[schema.]name}
\ddl_view       {[schema.]name}
\ddl_sequence   {[schema.]name}
\ddl_synonym     {[schema.]name}

```

```

\ddl_publicsynonym {name}
\startup          {[nomount|mount|open]}
\shutdown         {[abort|immediate|transactional|normal]}
\cstartup         {[nomount|mount|open]}
\cshutdown        {[abort|normal]}
\connect          [userid password] [as {sysdba|admin}]

```

\host

구문

```

\host [command]
\ho [command]

```

설명

gsql 프로그램을 종료하지 않은 채 운영 체제 명령을 실행한다.
command 없이 HOST를 입력하면 운영 체제 프롬프트가 표시되며 운영 체제 명령어를 계속 입력할 수 있다.
HOST 대신 Windows에서는 "\$"를, UNIX에서는 "!" 문자를 사용할 수 있다.

사용 예

다음은 UNIX 운영 체제 명령인 ls *.sql 구문을 수행하는 예이다.

```

gSQL> \host ls *.sql
DictionarySchema.sql  InformationSchema.sql  PerformanceViewSchema.sql

```

\history

구문

```

\history
\hi

```

설명

gsql 프로그램을 수행한 이후의 SQL 구문 수행 목록을 보여준다.

수행에 성공한 SQL 구문만 관리하며, gsql (\)로 시작하는 대화형 명령어로 수행한 명령어나 실패한 SQL 구문은 포함하지 않는다.

해당 목록을 참조하여 \ 명령이나 \{n} 명령을 통해 이전에 수행한 SQL 구문을 다시 수행할 수 있다.

관리할 수 있는 SQL 구문의 개수는 \set history 명령을 통해 제어한다.

사용 예

다음은 SQL 구문 수행 이력을 조회하고 8번 SQL 구문을 다시 수행하는 예이다.

```
gSQL> \history
ID SQL
-----
1 drop table t1
2 create table t1 ( id integer, name varchar(128), addr varchar(1024) )
3 create index t1_idx on t1(id)
4 insert into t1 values ( 1, 'leekmo', 'Seoul, Korea' )
5 insert into t1 values ( 2, 'mkkim', 'Seoul, Korea' )
6 insert into t1 values ( 3, 'xcom73', 'Inchon, Korea' )
7 commit
8 select * from dual
gSQL> \8
DUMMY
-----
X
1 row selected.
```

\import

구문

```
\import 'file_name'
\i 'file_name'
```

설명

file_name에 포함된 SQL 구문들을 수행한다.

file_name은 single quote (') 안에 기술한다.

file_name은 다음과 같이 절대 경로 또는 상대 경로를 사용할 수 있으며 상대 경로를 사용할 경우 gsql을 수행한 경로를 기준으로 file_name을 찾는다.

- 절대 경로 사용

```
gSQL> \import '/home/GOLDILOCKS/sample.sql'
```

- 상대 경로 사용

```
gSQL> \import 'sample.sql'
```

사용 예

다음은 절대 경로를 사용하여 file 내의 SQL 구문을 사용하는 예이다.

```
gSQL> \import '/home/GOLDILOCKS/sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(128),
    addr  VARCHAR(128)
);
Table created.
INSERT INTO t1
VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
      ( 2, 'mkkim' , 'Seoul, Korea' ),
      ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
```

다음은 상대 경로를 사용하여 file 내의 SQL 구문을 사용하는 예이다.

```
gSQL> \import 'sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
```



```

CREATE TABLE t1
(
    id    INTEGER,
    name  VARCHAR(128),
    addr  VARCHAR(128)
);
Table created.
INSERT INTO t1
VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
      ( 2, 'mkkim' , 'Seoul, Korea' ),
      ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.

```

\idesc

구문

```

\idesc index_name
\idesc schema_name.index_name

```

설명

인덱스의 구조 정보를 조회한다.

인덱스 이름은 다음과 같이 인덱스 이름만 기술하거나 스키마 이름과 함께 기술할 수 있다. 스키마 이름을 명시하지 않은 경우, 인덱스의 스키마 이름은 사용자의 **Schema Path**에 의해 결정된다.

- \idesc t1_idx_name
- \idesc public.t1_idx_name

수행 결과는 다음과 같이 인덱스의 key column 정보를 포함한다.

- Key column의 이름
- Key column의 위치
- Key column의 정렬 순서 (오름차순/ 내림차순)
- Key column의 NULL 값 위치 (FIRST/ LAST)

사용 예

다음은 t1_idx_name 인덱스의 정보를 조회하는 예이다.

```
gSQL> \idesc t1_idx_name
COLUMN_NAME  ORDINAL_POSITION  IS_ASCENDING_ORDER  IS_NULLS_FIRST
-----
NAME          1 TRUE              FALSE
```

\{n}

구문

\number

설명

\history 명령으로 조회할 수 있는 SQL 수행 이력 중에 number에 해당하는 구문을 수행한다.
number 값은 \history 수행 결과의 ID 값이어야 한다.

사용 예

다음은 \history 명령으로 수행한 SQL 구문을 수행하고 ID 값 1에 해당하는 DROP TABLE 구문을 수행하는 예이다.

```
gSQL> \history
ID SQL
-----
1 DROP TABLE IF EXISTS t1
2 CREATE TABLE t1
  (
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
  )
3 CREATE INDEX t1_idx_name ON t1(name)
4 INSERT INTO t1
  VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
          ( 2, 'mkkim' , 'Seoul, Korea' ),
          ( 3, 'xcom73', 'Inchon, Korea' )
```

```

5 COMMIT
6 select * from t1
gSQL> \1
Table dropped.

```

\prepare sql

구문

```
\prepare sql <sql_statement>
```

설명

<sql_statement>에 기술된 SQL 구문을 준비한다. 준비된 SQL 구문은 **\exec** 명령을 통해 반복적으로 수행할 수 있다.

\prepare sql 명령은 ODBC의 **SQLPrepare** 함수와 JDBC의 **Connection::prepareStatement** 함수와 유사하게 동작한다.

사용 예

다음은 SELECT 구문을 준비하고 반복적으로 수행하는 예이다.

- SELECT 구문을 준비한다.

```

gSQL> \prepare sql SELECT * FROM t1 WHERE id > 2;
SQL prepared.

```

- 준비된 SQL을 수행한다.

```

gSQL> \exec
ID NAME      ADDR
--  -----
3 xcom73 Inchon, Korea
1 row selected.
gSQL> INSERT INTO t1 VALUES ( 4, 'GOLDILOCKS', 'Better Place' );
1 row created.

```

- 준비된 SQL을 다시 수행한다.

```
gSQL> \exec
ID NAME      ADDR
--  -
3 xcom73 Inchon, Korea
4 goldilocks Better Place
2 rows selected.
```

\print

구문

```
\print
\print variable
```

설명

다음과 같이 **\var** 명령으로 선언한 호스트 변수 값을 조회한다. 호스트 변수 값이 설정되지 않은 경우 NULL 값을 가진다.

```
gSQL> \var v1 INTEGER
gSQL> \print v1
V1
----
null
```

다음과 같이 호스트 변수 이름을 기술하지 않을 경우, 선언한 모든 호스트 변수의 값을 조회한다. 다음 예에서 VAR_ELAPSED_TIME_ 호스트 변수는 구문 수행시간을 관리하는 built-in 호스트 변수이다.

```
gSQL> \print
NAME                VALUE
-----
VAR_ELAPSED_TIME__  null
V1                  21
V2                  10
```

사용 예

다음은 호스트 변수를 선언하고 호스트 변수에 값을 할당하고 이를 조회하는 예이다.

- 호스트 변수를 선언한다.

```
gSQL> \var v1 INTEGER
gSQL> \var v2 INTEGER
gSQL> \var v3 INTEGER
```

- 모든 호스트 변수를 조회한다.

```
gSQL> \print
NAME                VALUE
-----
VAR_ELAPSED_TIME__  null
V1                  null
V2                  null
V3                  null
```

- 호스트 변수에 값을 할당한다.

```
gSQL> \exec :v1 := 10
gSQL> \exec :v2 := 20
gSQL> \exec :v3 := :v1 + :v2
```

- v3 호스트 변수를 조회한다.

```
gSQL> \print v3
V3
--
30
```

- 모든 호스트 변수를 조회한다.

```
gSQL> \print
NAME                VALUE
-----
VAR_ELAPSED_TIME__  null
V1                  10
V2                  20
V3                  30
```

\quit

구문

```
\quit  
\q
```

설명

gsql을 종료한다.

종료할 때 COMMIT 되지 않은 트랜잭션을 모두 COMMIT 한다.

사용 예

```
gSQL> \quit  
%
```

\set autocommit

구문

```
\set autocommit on  
\set autocommit off
```

설명

SQL 구문을 수행한 후에 자동으로 COMMIT 할지 여부를 설정한다.

- \set autocommit on
 - SQL 구문을 수행한 후에 자동으로 COMMIT 한다.
- \set autocommit off
 - SQL 구문을 수행한 후에 자동으로 COMMIT 하지 않는다.
- autocommit의 기본값은 OFF 이다.

사용 예

다음은 AUTOCOMMIT 값을 ON으로 설정했을 때의 사용 예이다.

- autocommit을 on으로 설정한다.

```
gSQL> \set autocommit on
```

- INSERT 구문이 자동으로 COMMIT 된다.

```
gSQL> INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
1 row created.
```

- Rollback의 영향을 받지 않는다.

```
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME      ADDR
-- -----
1 leekmo Seoul, Korea
1 row selected.
```

다음은 AUTOCOMMIT 값을 OFF로 설정했을 때의 사용 예이다.

- autocommit을 off로 설정한다.

```
gSQL> \set autocommit off
```

- INSERT 구문을 수행한 후에 트랜잭션이 종료되지 않는다.

```
gSQL> INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
1 row created.
```

- 트랜잭션을 철회한다.

```
gSQL> ROLLBACK;
Rollback complete.
```

- INSERT 구문이 철회되어 결과가 없다.

```
gSQL> SELECT * FROM t1;
no rows selected.
```

\set autotrace

구문

```
\set autotrace on
\set autotrace traceonly
\set autotrace off
```

설명

실행 계획 출력 여부를 설정한다.

- \set autotrace on
 - SQL 구문을 수행하고 질의 결과와 함께 실행 계획을 출력한다.
- \set autotrace traceonly
 - SQL 구문을 수행하지 않고 질의 결과 없이 실행 계획을 출력한다.
- \set autotrace off
 - 실행 계획을 출력하지 않는다.
 - 기본값은 off 이다.

실행 계획에 대한 자세한 내용은 [SQL 실행 계획](#)을 참조한다.

사용 예

다음과 같이 ON과 함께 사용할 경우 SQL 구문을 수행하고 질의 결과와 함께 실행 계획을 출력한다.

```
gSQL> \set autotrace on
gSQL> SELECT * FROM t1 WHERE id = 1;
ID NAME      ADDR
--  -----
1 leekmo Seoul, Korea
1 row selected.
>>> start print plan
< Execution Plan >

=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |    TABLE ACCESS ("T1")                          |         1 |
=====
```



```

1 - READ COLUMNS : ID, NAME, ADDR
    PHYSICAL FILTER : ID = 1
<<< end print plan

```

다음과 같이 TRACEONLY를 사용할 경우, SQL 구문을 수행하지 않고 질의 결과 없이 실행 계획만 출력한다.

```

gSQL> \set autotrace traceonly
gSQL> SELECT * FROM t1 WHERE id = 1;
>>> start print plan
< Execution Plan >
=====
|  IDX  |  NODE DESCRIPTION                                |  ROWS  |
=====
|    0  |  SELECT STATEMENT                                |         |
|    1  |  TABLE ACCESS ("T1")                            |         0 |
=====
1 - READ COLUMNS : ID, NAME, ADDR
    PHYSICAL FILTER : ID = 1
<<< end print plan

```

\set color

구문

```

\set color on
\set color off

```

설명

터미널 상에서 질의 결과의 각 row를 구별할 수 있도록 출력 결과의 색깔을 다르게 할지 여부를 설정한다.

- \set color on
 - Row 별로 출력 색깔을 다르게 한다.
- \set color off
 - Row 별 출력 색깔이 동일하다.
- color의 기본값은 OFF 이다.

각 row의 길이가 너무 길어서 터미널의 창을 넘어갈 경우 여러 줄에 걸쳐 출력된다. 이 경우, row와 row 를 구별하기 어려워서 가독성이 떨어지기 때문에 터미널에서의 가독성을 높이기 위해 이 구문을 사용한다.

사용 예

다음은 color 값을 ON으로 설정한 예이다.

```
gSQL> \set color on
gSQL> SELECT id, addr FROM t1;
ID ADDR
-- -----
1 Seoul, Korea
2 Seoul, Korea
3 Incheon, Korea
3 rows selected.
```

\set colsize

구문

```
\set colsize number
```

설명

LONG VARCHAR, LONG VARBINARY 타입 데이터의 최대 출력 크기를 설정한다.

- colsize의 값은 1 ~ 104857600 (100M, LONG VARCHAR의 최대 크기) 범위의 양의 정수값이다.
- colsize의 기본값은 8192이다.

LONG VARCHAR column은 문자열 길이가 길기 때문에 다음과 같은 질의 수행 결과의 가독성이 떨어지게 된다. 이 때, colsize를 줄여 LONG VARCHAR column의 가독성을 높이거나, colsize를 크게 하여 원하는 길이만큼 출력할 수 있다.

```
gSQL> SELECT view_name, text FROM all_views WHERE view_name LIKE 'ALL_%' FETCH 3;
VIEW_NAME
-----
TEXT
-----
ALL_ALL_TABLES
SELECT
    auth.AUTHORIZATION_NAME      ❶ OWNER
    , sch.SCHEMA_NAME            ❷ TABLE_SCHEMA
```

... 중략 ...

```
WHERE      , pvc01.GRANTOR_ID
      , pvc01.GRANTEE_ID
      , pvc01.PRIVILEGE_TYPE_ID
```

3 rows selected.

사용 예

다음은 colsize를 줄여 LONG VARCHAR column의 출력 가독성을 높이는 예이다.

```
gSQL> \set colsize 200
gSQL> SELECT view_name, text FROM all_views WHERE view_name LIKE 'ALL_%' FETCH 3;
VIEW_NAME      TEXT
-----
ALL_ALL_TABLES  SELECT
                auth.AUTHORIZATION_NAME      ❶ OWNER
                , sch.SCHEMA_NAME            ❷ TABLE_SCHEMA
                , tab.TABLE_NAME              ❸ TABLE_NAME
                , spc.TAB
ALL_COL_COMMENTS SELECT
                auth.AUTHORIZATION_NAME
                , sch.SCHEMA_NAME
                , tab.TABLE_NAME
                , col.COLUMN_NAME
                , col.COMMENTS
                FROM
                DICTIONARY_SCHEMA.WHOLE_COLUMNS AS col
                , DICTIONARY_S
ALL_COL_PRIVS   SELECT
                grantor.AUTHORIZATION_NAME
                , grantee.AUTHORIZATION_NAME
                , owner.AUTHORIZATION_NAME
                , sch.SCHEMA_NAME
                , tab.TABLE_NAME
                , col.COLUMN_NAME
                , pvc01.PRIVILEGE_TY
3 rows selected.
```

\set ddlsize

구문

```
\set ddlsize number
```

설명

다음 명령을 통해 DDL을 출력할 때 구문을 출력할 버퍼의 크기를 설정한다.

- \ddl_db
 - \ddl_tablespace
 - \ddl_auth
 - \ddl_schema
 - \ddl_table
 - \ddl_constraint
 - \ddl_index
 - \ddl_view
 - \ddl_sequence
 - \ddl_synonym
 - \ddl_public_synonym
- ddlsize 값은 1 ~ 10485760 (10M) 범위의 양의 정수값이다.
 - ddlsize의 기본값은 10000 이다.

다음과 같이 버퍼 공간 부족으로 인한 에러가 발생할 경우, ddlsize 값을 증가시켜 DDL 구문을 출력할 수 있다.

```
gSQL> \set ddlsize 1000
gSQL> \ddl_view dictionary_schema.all_tables
ERR-HY000(40052): not enough DDLSIZE.
use command: \set ddlsize {n}
gSQL> \set ddlsize 100000
gSQL> \ddl_view dictionary_schema.all_tables
SET SESSION AUTHORIZATION "SYS";
CREATE OR REPLACE FORCE VIEW "DICTIONARY_SCHEMA"."ALL_TABLES"
... 중략 ...
```

사용 예

다음은 ddlsize를 변경하는 예이다.

```
gSQL> \set ddlsize 100000
gSQL>
```

\set error

구문

```
\set error on
\set error off
```

설명

에러 메시지 출력 여부를 설정한다.

- \set error on
 - 에러 메시지를 출력한다.
- \set error off
 - 에러 메시지를 출력하지 않는다.
- error의 기본값은 ON 이다.

SQL 구문을 수행할 때 에러가 발생하면 gsql은 다음과 같은 정보를 출력한다.

- SQLSTATE: SQL 표준 상태 코드
- 에러 코드: GOLDBLOCKS 에러 코드
- 에러 메시지

다음 예에서 SQL 구문에 에러가 발생하였고 ERR-42000(16040)에서 42000이 SQLSTATE 값이며 에러 코드는 () 안의 16040 이다.

```
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
          *
ERROR at line 1:
```

사용 예

다음은 에러 메시지를 출력하지 않도록 설정하는 예이다.

- 에러 메시지를 출력하지 않도록 한다.

```
gSQL> \set error off
```

- SQLSTATE 값과 에러 코드만 출력한다.

```
gSQL> SELECT * FROM invalid_table;  
ERR-42000(16040)
```

\set heading

구문

```
\set heading {ON|OFF}
```

설명

질의 결과에 헤더를 출력할지 여부를 설정한다.

사용 예

- 헤더 메시지를 출력하지 않도록 한다.

```
gSQL> \set heading off
```

- 다음은 질의 결과에 헤더 메시지를 출력하지 않는 예이다.

```
gSQL> select * from dual;  
X  
1 row selected.
```

\set history

구문

```
\set history number
```

설명

이력 정보를 관리할 SQL 구문의 개수를 설정한다.

- history 값은 1 ~ 100000 범위의 양의 정수값이다.
- history 값을 음수로 설정하면 모든 SQL 이력을 제거한다.
- history의 기본값은 128이다.

history 값이 줄어들면 오래된 SQL 구문을 제거한다.

history 정보는 \ 명령이나 \{n} 명령을 통해 이전에 수행한 SQL 구문을 다시 수행하려 할 때 사용한다.

사용 예

다음은 history 개수를 100으로 설정하는 예이다.

```
gSQL> \set history 100
gSQL>
```

다음은 저장되어 있는 SQL 구문의 개수보다 history 개수를 작게 설정하는 예이다.

- 다섯 개의 SQL 구문이 저장되어 있다.

```
gSQL> \history
ID SQL
-----
1 DROP TABLE IF EXISTS t1
2 CREATE TABLE t1
  (
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
  )
3 CREATE INDEX t1_idx_name ON t1(name)
4 INSERT INTO t1
    VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
```

```

        ( 2, 'mkkim' , 'Seoul, Korea' ),
        ( 3, 'xcom73', 'Inchon, Korea' )

5 COMMIT

```

- history 값을 3으로 줄인다.

```
gSQL> \set history 3
```

- 세 개의 SQL 구문이 저장되어 있다.

```

gSQL> \history
ID SQL
-----
3 CREATE INDEX t1_idx_name ON t1(name)
4 INSERT INTO t1
      VALUES ( 1, 'leekmo', 'Seoul, Korea' ),
              ( 2, 'mkkim' , 'Seoul, Korea' ),
              ( 3, 'xcom73', 'Inchon, Korea' )

5 COMMIT

```

다음은 history에 저장된 모든 SQL 이력을 제거하는 예이다.

```

gSQL> \history
ID SQL
-----
1 SELECT * FROM dual
2 SELECT * FROM user_tables

```

- 음수값을 이용해 모든 SQL 이력을 제거한다.

```

gSQL> \set history -1
gSQL> \history
gSQL>

```

\set linesize

구문

```
\set linesize number
```


설명

질의 결과를 출력할 때 한 line의 최대 크기를 설정한다.

- linesize 값은 1 ~ 10000 범위의 양의 정수값이다.
- linesize의 기본값은 80 이다.

질의 결과를 출력할 때 한 row는 하나의 line을 기준으로 출력된다. Row의 column 개수가 많거나 출력 길이가 linesize보다 클 경우 다음 예와 같이 여러 line에 걸쳐 출력되기 때문에 가독성이 떨어진다.

```
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME
-----
COMMENTS
-----
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA
Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME
Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME
Name of the tablespace containing the table
3 rows selected.
```

이 경우, linesize 값을 변경하여 하나의 row가 하나의 line에 출력되도록 제어할 수 있다.

사용 예

다음은 linesize를 크게 설정하여 가독성을 높이는 예이다.

```
gSQL> \set linesize 400
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME  COMMENTS
-----
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA  Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME      Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME  Name of the tablespace containing the table
3 rows selected.
```

\set numsize

구문

```
\set numsize number
```

설명

숫자값 출력을 위한 최대 digit 개수를 설정한다.

- numsize 값은 1 ~ 50 범위의 양의 정수값이다.
- numsize의 기본값은 20 이다.

다음과 같이 숫자값의 digit 개수가 numsize의 범위를 벗어나는 경우 exponent 형태로 출력된다.

```
gSQL> SELECT num, ( num * num ) AS result FROM t1;
```

NUM	RESULT
1234567890123	1.52415787532276E+24

1 row selected.

이런 숫자값을 exponent 형태가 아닌 숫자로 모두 표현하고자 할 때 numsize 값을 변경하여 출력할 수 있다.

사용 예

다음은 numsize를 크게 설정하여 숫자의 모든 digit를 출력하는 예이다.

```
gSQL> \set numsize 50
gSQL> SELECT num, ( num * num ) AS result FROM t1;
```

NUM	RESULT
1234567890123	1524157875322755800955129

1 row selected.

\set pagesize

구문

```
\set pagesize number
```

설명

한 page 단위로 구성할 row의 개수를 설정한다.

- pagesize 값은 1 ~ 10000 범위의 양의 정수값이다.
- pagesize의 기본값은 20 이다.

질의 결과로 출력되는 row의 개수가 많거나 특정 개수 단위로 page를 구성하고자 할 때 설정한다.

사용 예

다음은 10 개의 row 단위로 하나의 page를 구성하는 예이다.

```
gSQL> \set linesize 120
gSQL> \set pagesize 10
gSQL> SELECT column_name, comments FROM dict_columns WHERE table_name = 'SEQUENCES';
```

COLUMN_NAME	COMMENTS
OWNER_ID	authorization identifier who owns the table of the sequence generator
SCHEMA_ID	schema identifier of the sequence generator
SEQUENCE_ID	sequence generator identifier
SEQUENCE_TABLE_ID	table id of sequence for naming resolution
TABSPACE_ID	tablespace identifier of the sequence generator
PHYSICAL_ID	physical identifier of the sequence generator
SEQUENCE_NAME	sequence generator name
DTD_IDENTIFIER	unsupported feature
START_VALUE	the start value of the sequence generator
MINIMUM_VALUE	the minimum value of the sequence generator
COLUMN_NAME	COMMENTS
MAXIMUM_VALUE	the maximum value of the sequence generator
INCREMENT	the increment of the sequence generator
CYCLE_OPTION	The values of CYCLE_OPTION have the following meanings: - TRUE : The cycle option of the sequence generator is CYCLE. - FALSE : The cycle option of the sequence generator is NO CYCLE.
CACHE_SIZE	number of sequence numbers to cache

```

CREATED_TIME      created time of the sequence generator
MODIFIED_TIME     last modified time of the sequence generator
COMMENTS          comments of the sequence generator
SEQUENCE_CATALOG  catalog name of the sequence
SEQUENCE_OWNER    owner name of the sequence
SEQUENCE_SCHEMA   schema name of the sequence
COLUMN_NAME       COMMENTS
-----
SEQUENCE_NAME     sequence name
DATA_TYPE         the standard name of the data type
NUMERIC_PRECISION the numeric precision of the numerical data type
NUMERIC_PRECISION_RADIX the radix ( 2 or 10 ) of the precision of the numerical data type
NUMERIC_SCALE     the numeric scale of the exact numerical data type
START_VALUE       the start value of the sequence generator
MINIMUM_VALUE     the minimum value of the sequence generator
MAXIMUM_VALUE     the maximum value of the sequence generator
INCREMENT         the increment of the sequence generator
CYCLE_OPTION      cycle option
COLUMN_NAME       COMMENTS
-----
CACHE_SIZE        number of sequence numbers to cache
DECLARED_DATA_TYPE the data type name that a user declared
DECLARED_NUMERIC_PRECISION the precision value that a user declared
DECLARED_NUMERIC_SCALE the scale value that a user declared
CREATED_TIME      created time of the sequence generator
MODIFIED_TIME     last modified time of the sequence generator
COMMENTS          comments of the sequence generator
37 rows selected.

```

\set serveroutput

구문

```

\set serveroutput on
\set serveroutput off

```

설명

gsq!이나 gsqlnet에서 DBMS_OUTPUT package가 제공하는 함수들로 작성한 메시지들을 자동으로 출력하는 기능을 제어한다.

- \set serveroutput on
 - SQL을 수행한 후에 DBMS_OUTPUT.PUT_LINE이 서버에 축적한 메시지들을 자동으로 출력한다.
- \set serveroutput off
 - DBMS_OUTPUT package를 사용하지 않는다.
- serveroutput의 기본값은 OFF 이다.

기본적으로 서버에 축적할 수 있는 메시지의 최대 크기는 20000 byte 이다.

사용 예

다음은 SQL 구문에서 사용된 user-defined function에서 축적한 메시지를 출력하는 예이다.

```
gSQL> create or replace function my_msg( msg varchar(100) )
return integer
is
begin
  dbms_output.put_line( 'my message is : ' || msg );
  return length( msg );
end;
/
2 3 4 5 6 7 8
Function created.
gSQL> commit;
Commit complete.
gSQL> \set serveroutput on
gSQL> select my_msg( 'Hello World!' ) from dual;
MY_MSG( 'Hello World!' )
-----
                        12
my message is : Hello World!
1 row selected.
```

\set time

구문

```
\set time on  
\set time off
```

설명

현재 시간의 출력 여부를 설정한다.

- \set time on
 - 현재 시간을 출력한다.
- \set time off
 - 현재 시간을 출력하지 않는다.
- time의 기본값은 OFF 이다.

사용 예

다음은 현재 시간을 출력하는 예이다.

```
gSQL> \set time on  
12:45:34 gSQL> SELECT * FROM dual;  
DUMMY  
-----  
X  
1 row selected.  
Elapsed time: 0.07600 ms
```

\set timing

구문

```
\set timing on  
\set timing off
```

설명

SQL 구문 수행시간의 출력 여부를 설정한다.

- `\set timing on`
 - 수행시간을 출력한다.
- `\set timing off`
 - 수행시간을 출력하지 않는다.
- `timing`의 기본값은 OFF 이다.

수행시간의 단위는 ms (millisecond) 이다.

사용 예

다음은 SQL 구문의 수행 시간을 출력하는 예이다.

```
gSQL> \set timing on
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
Elapsed time: 0.07600 ms
```

`\set vertical`

구문

```
\set vertical on
\set vertical off
```

설명

Column 값을 line 단위로 출력할지 여부를 설정한다.

- `\set vertical on`
 - Column을 line 단위로 출력한다.
- `\set vertical off`
 - Row를 line 단위로 출력한다.
- `vertical`의 기본값은 OFF 이다.

각 row가 개별적인 정보를 가지고 있는 경우, 질의 출력 결과를 column 단위로 출력하여 가독성을 높일 수 있다.

\set vertical on으로 설정할 경우, 각 row는 빈 줄로 구분되며 하나의 line은 하나의 값을 표현하고 다음과 같은 형태로 구성된다.

```
column 이름 # 데이터 값
```

사용 예

다음은 질의 결과를 column 단위로 출력하는 예이다.

```
gSQL> \set vertical on
gSQL> SELECT * FROM v$system_stat FETCH 10;
      STAT_NAME # SYSTEM_SAR
      STAT_VALUE # 3
      COMMENTS # system available resource( 0:none, 1:session 2:database )
      STAT_NAME # MAX_ENVIRONMENT_COUNT
      STAT_VALUE # 128
      COMMENTS # maximum environment count
      STAT_NAME # FREE_ENVIRONMENT_ID
      STAT_VALUE # 81
      COMMENTS # available environment identifier
      STAT_NAME # MAX_SESSION_COUNT
      STAT_VALUE # 128
      COMMENTS # maximum session count
      STAT_NAME # FREE_SESSION_ID
      STAT_VALUE # 82
      COMMENTS # available session identifier
      STAT_NAME # MAX_PROCESS_COUNT
      STAT_VALUE # 128
      COMMENTS # maximum process count
      STAT_NAME # FREE_PROCESS_ID
      STAT_VALUE # 1
      COMMENTS # available process identifier
      STAT_NAME # CACHE_ALIGNED_SIZE
      STAT_VALUE # 64
      COMMENTS # cache aligned size
      STAT_NAME # CPU_COUNT
      STAT_VALUE # 8
      COMMENTS # count of CPUs
      STAT_NAME # SYSTEM_TIME
      STAT_VALUE # 1408676900172925
```



```
COMMENTS # system time
```

```
10 rows selected.
```

\shutdown

구문

```
\shutdown
\shutdown abort
\shutdown immediate
\shutdown transactional
\shutdown normal
```

설명

GOLDILOCKS 서버를 종료한다.

\shutdown 명령을 수행하기 위해서는 SYSDBA 또는 ADMIN role로 접속해야 한다. SYSDBA로 접속하는 방법은 **서버 구동 및 종료**를 참조한다.

- \shutdown normal
 - 새로운 session의 접속을 차단하고 현재 접속된 모든 session이 종료될 때까지 기다린 후, checkpoint를 수행하고 서버를 종료한다.
- \shutdown transactional
 - 새로운 transaction의 시작을 차단하고 현재 수행 중인 transaction들이 종료될 때까지 기다린 후, checkpoint를 수행하고 서버를 종료한다.
- \shutdown immediate
 - 새로운 단위 연산 (예: FETCH, EXECUTE 등)의 수행을 차단하고 현재 수행 중인 모든 단위 연산이 종료될 때까지 기다린 후, 모든 transaction들을 rollback 하고, checkpoint를 수행하고 서버를 종료한다.
- \shutdown abort
 - 접속 중인 session들의 상태와 관계없이 바로 서버를 강제 종료한다.
- \shutdown
 - \shutdown normal과 동일하다.

사용 예

다음은 GOLDILOCKS를 종료하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown
```

Shutdown success

gSQL>

\spool

설명

gsql을 사용하여 출력되는 모든 결과를 terminal과 file에 기록한다.

\spool 'filename'

구문

```
\spool 'filename' [CREATE | REPLACE | APPEND]
\spo  'filename' [CREATE | REPLACE | APPEND]
```

설명

Spool 기능을 시작한다. 이 구문이 실행된 이후부터 gsql에서 수행된 모든 출력 결과가 주어진 *filename*에 기록된다.

다음 option들은 file을 open하는 방법을 기술한다.

- CREATE: File을 새로 생성한다. 기존 file이 있을 경우, 에러가 발생한다.
- REPLACE: 기존 file의 내용을 삭제하고 새로 기록을 시작한다. 만약 기존 file이 없을 경우, CREATE와 동일하게 동작한다.
- APPEND: 기존 file의 가장 뒷부분에 이어서 내용을 기록한다. 만약 기존 file이 없을 경우, CREATE와 동일하게 동작한다.

만약 이런 option이 주어지지 않았을 경우, 기본적으로 REPLACE 모드로 동작한다.

노트

Spool 기능이 동작하는 도중에 새로운 spool이 시작되면 기존 spool은 종료되고 새 spool이 시작된다.

사용 예

다음은 spool 기능을 시작하는 예이다.

- result.txt file을 생성하여 gsql 실행 결과를 기록한다.

```
gSQL> \SP00L 'result.txt' CREATE
gSQL> SELECT * FROM T1 WHERE C1 < 10;
gSQL> \SP00L OFF
```

- 기존 result.txt file의 뒷부분에 결과를 이어서 기록한다.

```
gSQL> \SP00L 'result.txt' APPEND
gSQL> SELECT * FROM T1 WHERE C1 >= 10;
gSQL> \SP00L OFF
```

- 기존 result.txt file의 내용을 삭제하고 새로 결과를 기록한다.

```
gSQL> \SP00L 'result.txt' REPLACE
gSQL> SELECT * FROM T1;
gSQL> \SP00L OFF
```

\spool OFF

구문

```
\spool OFF
\spo OFF
```

설명

현재 spool을 종료한다. 만약 spool을 사용하고 있는 중이 아니라면 아무런 동작도 일어나지 않는다.

사용 예

다음은 spool을 시작한 뒤에 종료하는 예이다.

```
gSQL> \SP00L 'result.txt'
gSQL> SELECT * FROM T1;
```

- Spool 종료

```
gSQL> \SP00L OFF
```

\spool

구문

```
\spool
```

```
\spo
```

설명

현재 spool 상태를 나타낸다. Spool을 사용하는 중이라면 어떤 file이 spool 중인지 알려주고, spool을 사용하고 있지 않은 경우에는 spool을 사용하고 있지 않다는 사실을 알린다.

사용 예

다음은 gsql 이력에 저장된 SQL 구문을 편집하는 예이다.

```
gSQL> \SP00L 'a.txt'
```

- a.txt에 spool 중이라는 사실을 알린다.

```
gSQL> \SP00L
```

```
currently spooling to a.txt
```

```
gSQL> \SP00L OFF
```

- 현재 spool을 사용하고 있지 않다는 사실을 알린다.

```
gSQL> \SP00L
```

```
not spooling currently
```

\startup

구문

```
\startup
```

```
\startup nomount
```

```
\startup mount
```

```
\startup open
```

설명

GOLDILOCKS 서버를 구동한다.

\startup 명령을 수행하려면 SYSDBA 또는 ADMIN role로 접속해야 한다. SYSDBA로 접속하는 방법은 **서버 구동 및 종료**를 참조한다.

- \startup nomount
 - 서버를 NOMOUNT 단계로 시작한다.
- \startup mount
 - 서버를 MOUNT 단계로 시작한다.
- \startup open
 - 서버를 OPEN 단계로 시작한다.
- \startup
 - \startup open과 동일하다.

서버의 구동 단계는 NOMOUNT, MOUNT, OPEN 단계로 구분되며 자세한 내용은 **다단계 시작**을 참조한다.

\startup 명령을 수행한 후에 다음 단계로 전이하려면 **ALTER SYSTEM {MOUNT | OPEN} DATABASE** 구문을 수행해야 한다.

사용 예

다음은 GOLDILOCKS를 구동하는 예이다.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
Startup success
```

\var

구문

```
\var variable_name data_type
```

설명

호스트 변수를 선언한다.

variable_name과 동일한 호스트 변수가 선언되어 있을 경우, 기존의 호스트 변수를 삭제하고 새로 선언한다. variable_name의 최대 길이는 128 byte 이다.

호스트 변수의 data_type은 GOLDILOCKS의 데이터 타입과 동일한데 자세한 내용은 **Data Type**을 참조한다.

호스트 변수는 `\exec :var := value` 명령을 이용하여 값을 대입하며 `\print` 명령을 통해 값을 조회할 수 있다.

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
gSQL> \print v1
V1
--
1
```

호스트 변수는 SQL 구문의 입력 또는 출력 인자로 사용할 수 있으며 SQL 구문이나 `\exec :var := value` 명령에서 사용할 때는 colon (:) 기호를 호스트 변수 앞에 붙여 호스트 변수임을 명시해야 한다.

사용 예

다음은 호스트 변수를 선언하고 SQL 구문의 입력 인자로 사용하는 예이다.

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
gSQL> SELECT * FROM t1 WHERE id = :v1;
ID NAME    ADDR
--
1 leekmo Seoul, Korea
1 row selected.
```

다음은 호스트 변수를 SQL 구문의 출력 인자로 사용하는 예이다.

```
gSQL> \var v_name VARCHAR(128)
gSQL> SELECT name INTO :v_name FROM t1 WHERE id = 1;
V_NAME
-----
leekmo
1 row selected.
```

다음은 호스트 변수를 선언하고 SQL 구문의 입력과 출력 인자로 사용하는 예이다.

UPDATE name RETURNING .. INTO 구문을 수행하는 다음 예에서 SET 절과 WHERE 절의 호스트 변수 `:v_id`는 입력 인자로 사용되었으며 INTO 절의 `:v_id`는 출력 인자로 사용되었다.

```
gSQL> \var v_name VARCHAR(128)
gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> UPDATE t1 SET id = 100 + :v_id WHERE id = :v_id RETURNING id INTO :v_id;
```

```
V_ID
----
101
1 row updated.
gSQL> \print v_id
V_ID
----
101
```


34.

gloader/gloadernet (Upload/download Tool)

34.1 gloder 개요

gloder는 GOLDILOCKS 내에 존재하는 데이터를 테이블 단위로 다운로드 또는 업로드 할 수 있는 유틸리티이다.

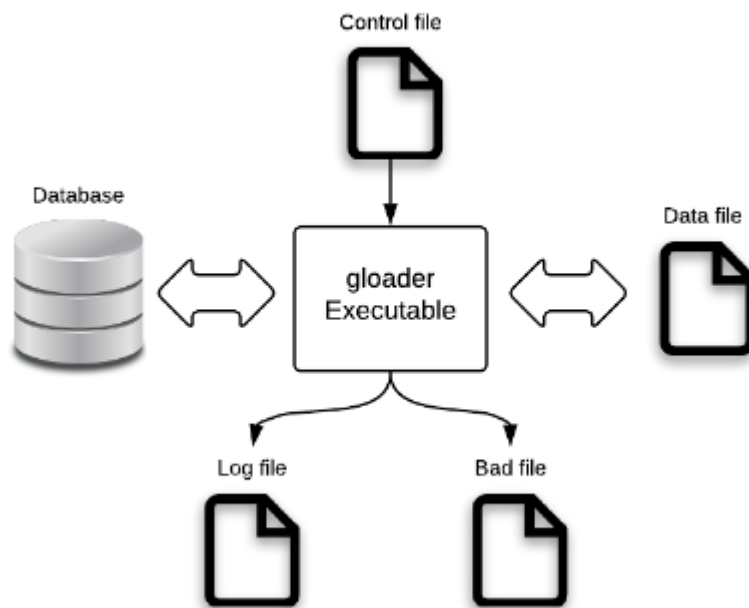
표 34-1 실행파일 구분

실행파일 이름	구분
gloder	Direct attach (D/A) 환경에서 사용된다.
gloadernet	Client/ server (C/S) 환경에서 사용된다.

Environment

gloder는 database와 연결이 되어야 하고 gloder를 사용할 때 필요한 모든 파일에 주의를 기울여야 한다.

그림 1 gloder environment



데이터를 업로드하려면 control file과 datafile이 필요하고 업로드 결과로써 log file이 생성된다.

데이터를 다운로드하려면 control file이 필요하고 다운로드 결과로써 datafile과 log file, bad file이 생성된다.

Control File

Control file은 gloder가 동작하는데 필요한 파일로써 다음과 같은 정보를 가지고 있다. (Control File 구문을 참조한다.)

- Table 이름
- Schema 이름
- Row의 column 사이의 구분자
- Column 데이터의 시작과 끝을 알리는 qualifier
- Row 사이의 구분자
- Character set
- 공백 문자에 대한 trim 여부
- Where 절

Data File

gloader가 데이터를 업로드 할 때는 data file을 미리 준비해야 하고 데이터를 다운로드 할 때는 datafile이 생성된다. Data file은 텍스트 형식과 바이너리 형식을 지원한다.

- 텍스트 형식 데이터 파일은 파일의 내용을 확인하여 직접 수정할 수 있다는 장점이 있다.
- 바이너리 형식 데이터 파일은 텍스트 형식과 비교하여 빠르게 수행할 수 있다.

노트

gloader는 data file에 대해 default로 direct IO를 사용한다. Direct IO를 사용하여 data file을 업로드 할 때 파일 크기가 direct IO에 적합한 정렬이 아닐 경우, gloader가 임의로 파일 크기를 조절한다.

Log File

Log file은 gloader가 동작하면서 발생하는 다음과 같은 에러와 결과를 기록하는 파일이다.

- 에러가 발생한 row 번호와 원인
- gloader 운영에 대한 결과

Bad File

Bad file은 gloader가 데이터를 업로드하는 동안 에러가 발생한 row를 기록하는 파일이다. Bad file을 기록할 때 column, row 사이의 구분자와 qualifier는 사용자가 지정한 것을 사용한다.

사용 예

다음은 gloader를 이용하여 데이터를 다운로드하고 업로드하는 예이다.

다음과 같이 SQL statement를 이용하여 table을 생성한다.

```
$ cat test.sql
CREATE TABLE TEST
(
  TEST_NAME VARCHAR(60),
  TEST_NUM  INTEGER,
  TEST_TIME TIMESTAMP(0) WITH TIME ZONE
);
INSERT INTO TEST VALUES
( 'NAME', 1, '1999-01-08 04:05:06.789 -8:00' );
INSERT INTO TEST VALUES
( 'NAME', 2, '1999-01-08 04:05:06.789 -8:00' );
INSERT INTO TEST VALUES
( 'NAME', 3, '1999-01-08 04:05:06.789 -8:00' );
COMMIT;
```

다음과 같이 control file을 이용한다.

```
$ cat test.ctl
TABLE TEST
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

- Export: 데이터를 다운로드 한다.

```
$ gloder test test --export --control test.ctl --data test.dat --no-copyright
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS
$ cat test.dat
"NAME","1","1999-01-08 04:05:07. -08:00"
"NAME","2","1999-01-08 04:05:07. -08:00"
"NAME","3","1999-01-08 04:05:07. -08:00"
$ cat test.log
cat test.log
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS [ Start Time: 2010-1-1 01:01:01 End Time:
2010-1-1 01:01:01 Taken Time: 56496 micro-sec ]
```

- Import: 데이터를 업로드 한다.

```
$ cat import.dat
"NAME","1","1999-01-08 04:05:07. -08:00"
"NAME","2","1999-01-08 04:05:07. -08:00"
"NAME","3","1999-01-08 04:05:07. -08:00"
```

```
"FAIL","FAIL","FAIL"
```

```
$ gloader test test --import --control test.ctl --data import.dat --no-copyright
```

```
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 4 RECORDS, SUCCEEDED 3 RECORDS
```

```
$ gsql test test
```

```
gSQL> select * from test;
```

```
TEST_NAME TEST_NUM TEST_TIME
```

```
-----
```

NAME	1	1999-01-08 04:05:07.000000 -08:00

NAME	2	1999-01-08 04:05:07.000000 -08:00

NAME	3	1999-01-08 04:05:07.000000 -08:00

NAME	1	1999-01-08 04:05:07.000000 -08:00

NAME	2	1999-01-08 04:05:07.000000 -08:00

NAME	3	1999-01-08 04:05:07.000000 -08:00

```
6 rows selected.
```

```
$ cat import.log
```

```
Err Rec(4) Col(2): 22018(12006): data value is not a numeric literal
```

```
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 4 RECORDS, SUCCEEDED 3 RECORDS [ Start Time:
```

```
2010-1-1 01:01:01 End Time: 2010-1-1 01:01:01 Taken Time: 56496 micro-sec ]
```

```
$ cat import.bad
```

```
"FAIL","FAIL","FAIL"
```

34.2 gloder 사용

Datafile Type

Text Datafile

사용자가 눈으로 확인하고 수정할 수 있는 문자열로 표현된다. 직접 파일을 생성하거나 수정할 수 있고 database에 있는 기존 테이블의 데이터를 다운로드 받을 수도 있다. 또한 다른 DBMS 제품에서 다운로드 받은 텍스트 형식의 데이터를 이용할 수도 있다.

텍스트 타입 데이터 파일의 표현에 대한 설명은 control file에 기록된다.

Binary Datafile

이진 파일로 되어 있다. 사용자는 binary datafile을 직접 생성하거나 수정할 수 없고 database에 있는 기존 테이블의 데이터를 다운로드 받을 때 생성된다.

Binary 타입 파일의 데이터는 GOLDILOCKS에 정의된 각 데이터 타입의 구조에 적합하게 기록되기 때문에 텍스트 타입보다 빠르게 업로드 할 수 있다.

노트

GOLDILOCKS database 버전이 서로 다른 경우에는 binary 파일을 이용한 데이터 업로드/ 다운로드를 권하지 않는다. 또한 문자열 집합이 다른 database 사이에 업로드/ 다운로드 할 경우, column의 크기를 변경해야 할 수도 있다.

데이터 다운로드

Text File로 다운로드

Simple Download

다음은 다운로드 할 테이블의 구조와 데이터를 보여준다.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, 'LKH', X'10' );
INSERT INTO TEST VALUES( 2, 'KMM', X'A0' );
```

```

INSERT INTO TEST VALUES( 3, 'ksj', X'CD' );
COMMIT;
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2 I3
-- --- --
1 LKH 10
2 KMM A0
3 ksj CD
3 rows selected.

```

다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.

```

$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','

```

다음과 같이 gloader를 통해 T1 테이블의 데이터를 다운로드한다.

```

$ gloader test test --export --control test.ctl --data test.dat
COMPLETED IN EXPORTING TABLE: PUBLIC.test, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS

```

다음과 같이 TEST 테이블의 데이터가 데이터 파일에 다운로드 되었다.

```

$ ls
test.ctl test.dat test.log
$ cat test.dat
1,LKH,10
2,KMM,A0
3,ksj,CD

```

공백 문자 처리

다음은 다운로드 할 테이블의 구조와 데이터를 보여준다.

```

$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, ' L K H ', X'10' );
INSERT INTO TEST VALUES( 2, 'KIM
MM', X'A0' );
INSERT INTO TEST VALUES( 3, ' KIM S
J ', X'CD' );

```

```

COMMIT;
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2      I3
--
1  L K H  10
2  KIM     A0
MM
3  KIM S   CD
J
3 rows selected.

```

- OPTIONALLY ENCLOSED BY 구문을 사용하지 않은 control 파일
 - 다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.

```

$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'

```

- 다음은 TEST 테이블의 데이터가 데이터 파일에 다운로드된 모습이다.

```

$ ls
test.ctl test.dat test.log
$ cat test.dat
1, L K H ,10
2,KIM
MM,A0
3, KIM S
J ,CD

```

노트

데이터에 공백 문자가 있는 경우 control 파일에 OPTIONALLY ENCLOSED BY 구문을 사용해야 다운로드 할 때와 업로드 할 때 모두 동일한 데이터를 유지할 수 있다.

- OPTIONALLY ENCLOSED BY 구문을 사용한 control 파일
 - 다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.


```
$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

- 다음은 TEST 테이블의 데이터가 데이터 파일에 다운로드된 모습이다.

```
$ ls
test.ctl test.dat test.log
$ cat test.dat
"1"," L K H ","10"
"2","KIM
MM","A0"
"3"," KIM S
J ","CD"
```

시간 데이터 타입

DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE을 다운로드하면 프로퍼티 기본 포맷으로 출력된다.

다음은 DATE, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE의 데이터 형식을 보여준다.

```
$ gsql test test
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE 'NLS_DATE_
FORMAT';
PROPERTY_VALUE                                INIT_VALUE
-----
YYYY-MM-DD                                    YYYY-MM-DD
1 row selected.
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE 'NLS_TIME_
WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE                                INIT_VALUE
-----
HH24:MI:SS.FF6 TZH:TZM                        HH24:MI:SS.FF6 TZH:TZM
1 row selected.
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE
'NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE                                INIT_VALUE
-----
YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM
```

1 row selected.

다음은 다운로드 할 테이블의 구조와 데이터를 보여준다.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER, I2 DATE, I3 TIME WITH TIME ZONE, I4 TIMESTAMP WITH TIME ZONE
);
INSERT INTO TEST VALUES ( 1, '1999-12-31', '01:01:01', '1999-12-31 01:01:01.789 -8:00' );
INSERT INTO TEST VALUES ( 2, '2000-01-01', '23:12:12', '2000-01-01 23:12:06.0 +8:00' );
INSERT INTO TEST VALUES ( 3, '2000-12-31', '23:12:12', '2000-12-31 23:12:12.0 -8:00' );
COMMIT;
$ gsql test test
gSQL> SELECT * FROM T1;
I1 I2          I3                      I4
--
1 1999-12-31 01:01:01.000000 +09:00 1999-12-31 01:01:01.789000 -08:00
2 2000-01-01 23:12:12.000000 +09:00 2000-01-01 23:12:06.000000 +08:00
3 2000-12-31 23:12:12.000000 +09:00 2000-12-31 23:12:12.000000 -08:00
3 rows selected.
```

다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.

```
$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

다음은 TEST 테이블의 데이터를 데이터 파일에 다운로드한 모습이다.

```
$ gloder test test -e -c test.ctl -d test.dat
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS
$ ls
test.ctl test.dat test.log
$ cat test.dat
"1","1999-12-31 00:00:00","01:01:01.000000 +09:00","1999-12-31 01:01:01.789000 -08:00"
"2","2000-01-01 00:00:00","23:12:12.000000 +09:00","2000-01-01 23:12:06.000000 +08:00"
"3","2000-12-31 00:00:00","23:12:12.000000 +09:00","2000-12-31 23:12:12.000000 -08:00"
```

Binary File로 다운로드

Simple Download

다음은 다운로드 할 테이블의 구조와 데이터를 보여준다.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, 'LKH', X'10' );
INSERT INTO TEST VALUES( 2, 'KMM', X'A0' );
INSERT INTO TEST VALUES( 3, 'ksj', X'CD' );
COMMIT;
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2 I3
-- -- --
1 LKH 10
2 KMM A0
3 ksj CD
3 rows selected.
```

다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.

```
$ cat testctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
```

TEST 테이블의 데이터는 다음과 같이 gloader를 통해 다운로드된다.

```
$ gloader test test --export --control testctl --data test.dat --format binary
COMPLETED IN EXPORTING TABLE: PUBLIC.test, 3 RECORDS
```

다음은 gloader를 수행한 후에 생성된 파일이다.

```
$ ls
testctl test.dat test.log
```

노트

Binary 파일의 내용은 직접 확인하거나 수정할 수 없다.

Complex Download

다음은 다운로드 할 테이블의 구조와 데이터를 보여준다.

```
$ gsql test test
gSQL>\desc TEST
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 DATE TRUE
I3 TIME(6) WITH TIME ZONE TRUE
I4 TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
-----
1572864
1 row selected.
```

다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.

```
$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
```

- 하나의 파일로 다운로드
 - 다음과 같이 gloder를 통해 TEST 테이블의 데이터를 다운로드한다.

```
$ gloder test test --export --control test.ctl --data test.dat --format binary
loaded 1000 records into PUBLIC.TEST
loaded 2000 records into PUBLIC.TEST
... 중략 ...
loaded 1571000 records into PUBLIC.TEST
loaded 1572000 records into PUBLIC.TEST
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, TOTAL 1572864 RECORDS
```

- 다음은 gloder를 수행한 후에 생성된 파일이다.

```
$ ll test.*
-rw-r--r-- 1 test test 71 2014-08-28 12:13 t1.ctl
-rw-r--r-- 1 test test 59930624 2014-08-28 12:49 t1.dat
-rw-r--r-- 1 test test 159 2014-08-28 12:49 t1.log
```

- 여러 파일로 다운로드
 - 다음과 같이 gloader를 통해 TEST 테이블의 데이터를 다운로드한다.

```
$ gloader test test --export --control test.ctl --data test.dat --format binary --filesize
31461376
loaded 1000 records into PUBLIC.TEST
loaded 2000 records into PUBLIC.TEST
... 중략 ...
loaded 1571000 records into PUBLIC.TEST
loaded 1572000 records into PUBLIC.TEST
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, TOTAL 1572864 RECORDS
```

- 다음은 gloader를 수행한 후에 생성된 파일이다.

```
$ ll test.*
-rw-r--r-- 1 test test      71 2014-08-28 12:13 test.ctl
-rw-r--r-- 1 test test 31461376 2014-08-28 13:02 test.dat
-rw-r--r-- 1 test test 28474880 2014-08-28 13:02 test.dat.001
-rw-r--r-- 1 test test    159 2014-08-28 12:49 test.log
```

데이터 업로드

Text File 업로드

Simple Upload

다음은 업로드 대상인 테이블을 보여준다.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 DATE TRUE
I3 TIME(6) WITH TIME ZONE TRUE
I4 TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

다음은 업로드 할 데이터 파일을 보여준다.

```
$ cat test.dat
1,1999-12-31 00:00:00,01:01:01.000000 +09:00,1999-12-31 01:01:01.789000 -08:00
2,2000-01-01 00:00:00,23:12:12.000000 +09:00,2000-01-01 23:12:06.000000 +08:00
3,2000-12-31 00:00:00,23:12:12.000000 +09:00,2000-12-31 23:12:12.000000 -08:00
```

다음과 같이 gloder로 데이터 파일을 업로드하고 결과를 확인한다.

```
$ gloder test test --import --control test.ctl --data test.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2 I3 I4
-----
1 1999-12-31 01:01:01.000000 +09:00 1999-12-31 01:01:01.789000 -08:00
2 2000-01-01 23:12:12.000000 +09:00 2000-01-01 23:12:06.000000 +08:00
3 2000-12-31 23:12:12.000000 +09:00 2000-12-31 23:12:12.000000 -08:00
3 rows selected.
```

공백 문자 처리

다음은 업로드 대상인 테이블을 보여준다.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(10) TRUE
I3 BINARY VARYING(10) TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

- Control 파일에서 OPTIONALLY ENCLOSED BY 구문을 사용하지 않고 다운로드한 데이터 파일
 - 다음은 control 파일에서 OPTIONALLY ENCLOSED BY 구문을 사용하지 않고 다운로드한 데이터 파일이다. (Text File로 다운로드를 참조한다.)

```
$ cat test.dat
1, L K H ,10
2,KIM
MM,A0
3, KIM S
J ,CD
```

- 다음과 같이 gloader로 데이터 파일을 업로드하고 결과를 확인한다. 두 번째와 세 번째 레코드의 데이터 내에 New Line('\n')이 존재하는데도 불구하고 column 데이터의 시작과 끝을 알리는 qualifier가 설정되지 않아 이를 row 구분자로 인식하게 된다.

```
$ gloader test test --import --control test.ctl --data test.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 5 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2 I3
--
1 L K H 10
2 KIM null
3 KIM S null
3 rows selected.
```

- Control 파일에서 OPTIONALLY ENCLOSED BY 구문을 사용하여 다운로드한 데이터 파일
 - 다음은 control 파일에서 OPTIONALLY ENCLOSED BY 구문을 사용하여 다운로드한 데이터 파일이다. (T

ext File로 다운로드를 참조한다.) 위의 결과와는 달리 아래의 결과를 보면 New Line('\n')이 데이터의 일부로 취급되었다.

```
$ cat test.dat
"1"," L K H ","10"
"2","KIM
MM","A0"
"3"," KIM S
J ","CD"
```

- 다음과 같이 gloder로 데이터 파일을 업로드하고 결과를 확인한다.

```
$ gloder test test --import --control test.ctl --data test.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 5 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2      I3
--
1  L K H  10
2 KIM      A0
MM
3  KIM S   CD
J
3 rows selected.
```

Binary File 업로드

Simple Upload

다음은 업로드 할 테이블의 구조를 보여준다.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE                IS_NULLABLE
-----
I1          NUMBER(10,0)                TRUE
I2          CHARACTER VARYING(10)    TRUE
I3          BINARY VARYING(10)         TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

다음은 테이블 데이터를 다운로드하기 위해 작성한 control 파일의 내용이다.


```
$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
```

다음은 gloader를 이용하여 업로드 할 파일이다. test.dat는 **Binary File**로 다운로드 할 때 미리 다운로드하였다.

```
$ ls
test.dat
```

다음과 같이 TEST 테이블에 데이터를 업로드하고 결과를 보여준다.

```
$ gloader test test --import --control test.ctl --data test.dat --format binary
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2 I3
-- --- --
1 LKH 10
2 KMM A0
3 ksj CD
3 rows selected.
```

Complex Upload

다음은 업로드 할 테이블의 구조를 보여준다.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(10) TRUE
I3 BINARY VARYING(10) TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

다음은 업로드하기 위한 데이터 파일이다.

```
$ ll test.*
-rw-r--r-- 1 test test 71 2014-08-28 12:13 test.ctl
-rw-r--r-- 1 test test 31461376 2014-08-28 13:02 test.dat
-rw-r--r-- 1 test test 28474880 2014-08-28 13:02 test.dat.001
-rw-r--r-- 1 test test 159 2014-08-28 12:49 test.log
```

--filesize로 download 한 여러 파일을 upload 하려면 각각의 데이터 파일에 대해 개별적으로 gloder를 수행해야 한다.

```
$ gloder test test --import --control test.ctl --data test.dat --format binary
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 860728 RECORDS, SUCCEEDED 2285000 RECORDS,
ERRORED 0 RECORDS
$ gloder test test --import --control test.ctl --data test.dat.001 --format binary
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 860728 RECORDS, SUCCEEDED 860728 RECORDS,
ERRORED 0 RECORDS
```

다음은 업로드한 결과이다.

```
gSQL> select count(*) from test;
COUNT(*)
-----
3145728
1 row selected.
```

Upload 처리 단위 제어

데이터를 업로드 할 때 성능과 관련된 옵션을 사용하면 더 빠르게 실행할 수 있다.

--array, --commit, --atomic을 참조한다.

다음은 약 38만 레코드를 가진 데이터 파일이다.

```
$ ll test.dat
-rw-r--r-- 1 test test 75092480 2014-08-28 15:59 test.dat
```

다음은 업로드에 사용되는 control 파일이다.

```
$ cat test.ctl
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY ''
LINES TERMINATED BY '\n'
```

Array 바인딩과 Commit 주기

다음과 같이 gloder 명령을 실행하면 바인드하는 레코드를 5000 개, COMMIT하는 레코드는 20000 개로 하여 업로드한다.

```
$ gloader test test --import --control test.ctl --data test.dat --array 5000 --commit 20000
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... 중략 ...
loaded 3810000 records into PUBLIC.TEST
loaded 3815000 records into PUBLIC.TEST
loaded 3818244 records into PUBLIC.TEST
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818243 RECORDS
```

gloader를 실행한 결과는 다음과 같다.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
-- -----
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
-----
3818246
1 row selected.
```

Array 바인딩과 Atomic 옵션

다음과 같이 gloader 명령을 실행하면 바인드하는 레코드를 5000 개씩 atomic INSERT로 업로드하고, 1000 개의 레코드가 실패한다.

```
$ gloader test test --import --control test.ctl --data test.dat --array 5000 --atomic
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... 중략 ...
loaded 38175000 records into PUBLIC.TEST
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3817243 RECORDS
```

다음은 gloader로 실행한 결과이다.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
```

```

-- -----
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
-----
3817243
1 row selected.

```

다음은 업로드에 실패한 원인을 로그 파일에 기록한 결과인데 첫 번째 레코드의 첫 column에 숫자가 아닌 데이터가 저장되었기 때문에 업로드에 실패하였다.

```

$ cat test.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(1) Col(-1): HY000(19041): Failed to atomic execution
Err Rec(1001) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(1001) Col(-1): HY000(19041): Failed to atomic execution
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3817243 RECORDS [
Start Time: 2014-8-28 16:46:42 End Time: 2014-8-28 16:46:52 Taken Time: 10084582 micro-sec ]

```

다음은 업로드에 실패한 레코드를 bad file에 기록한 결과인데 500 개의 array 단위로 업로드하기 때문에 1000 개의 레코드가 저장되었다.

```

"s1"," L K H ","10"
"2","KIM","A0"
"3"," KIM S J ","CD"
... 생략 ...

```

노트

Array, commit 옵션은 수행 결과에 영향을 주지 않지만 atomic 연산의 경우 INSERT 성공이나 실패가 array 단위로 처리되기 때문에 하나의 레코드라도 업로드에 실패하면 해당 레코드가 속한 array의 모든 레코드가 INSERT 실패로 처리된다.

로그 파일에는 array에서 처음 실패한 레코드의 원인만 기록되고 이후 레코드의 실패에 대해선 기록되지 않는다.

Parallel Upload

gloader가 업로드 작업을 수행할 때 작업을 부분별로 나누어 여러 thread로 처리할 수 있어 성능 향상에 도움이 된다. (--parallel을 참조한다.)

다음은 약 38만 레코드를 가진 데이터 파일이다.

```
$ ll test.dat
-rw-r--r-- 1 test test 75092480 2014-08-28 15:59 test.dat
```

다음은 업로드에 사용되는 control 파일이다.

```
$ cat test.ct1
TABLE PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

다음과 같이 gloader 명령을 실행하면 네 개의 업로드를 수행하는 thread로 바인드하는 레코드를 5000 개씩 INSERT로 업로드하고, 100 개의 레코드가 실패한다.

```
$ gloader test test --import --control test.ct1 --data test.dat --array 5000 --parallel 4
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... 중략 ...
loaded 3810000 records into PUBLIC.TEST
loaded 3815000 records into PUBLIC.TEST
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818143 RECORDS
```

다음은 gloader를 실행한 결과이다.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
--
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
-----
3818143
```

1 row selected.

다음은 업로드에 실패한 원인을 로그 파일에 기록한 결과이다.

```
$ cat test.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(3) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(5) Col(1): 22018(12006): data value is not a numeric literal
... 중략 ...
Err Rec(1001) Col(1): 22018(12006): data value is not a numeric literal
... 중략 ...
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818143 RECORDS [
Start Time: 2014-8-28 16:46:42 End Time: 2014-8-28 16:46:52 Taken Time: 10084582 micro-sec ]
```

다음은 업로드에 실패한 레코드를 bad file에 기록한 결과인데 100 개의 레코드가 저장되었다.

```
"s1"," L K H ","10"
"s2","KIM","A0"
"s3"," KIM S J ","CD"
... 중략 ...
```

노트

다중 thread로 데이터를 업로드 할 때 INSERT하고 에러가 발생한 레코드에 대한 내용을 로그 파일과 bad file에 기록하는데, 이 때 레코드의 순서는 데이터 파일에서의 순서와 다를 수 있다.

Upload 문제 해결

레코드 업로드 실패에는 다양한 원인이 있고 업로드에 실패한 레코드는 bad file에 기록된다. 또한 실패 원인에 대한 정보는 로그 파일에 기록된다.

중복 제약으로 인한 실패

제약 조건이 있는 업로드 할 테이블에 이미 중복된 데이터가 존재 (primary key 또는 unique index)

다음은 업로드 할 테이블 대상의 구조이다.

```
gSQL>\desc T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(10) TRUE
I3 BINARY VARYING(10) TRUE
gSQL> SELECT * FROM T1;
I1 I2 I3
-- -- --
1 LKH 10
2 KMM A0
3 ksj CD
3 rows selected.
```

다음은 업로드 할 데이터 파일이다.

```
$ cat t1.dat
"1","LKH"
"4","SOS"
"5","OK0"
```

다음은 gloader로 업로드를 실행한 결과 및 생성된 로그 파일, bad file이다.
레코드가 기본 key 제약 조건을 위반하였기 때문에 업로드에 실패하였다.

```
$ gloader test test -i -c t1.ct1 -d t1.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 2 RECORDS
$
$ cat t1.log
```

```

Err Rec(1) Col(-1): 40002(16057): unique constraint (PUBLIC.T1_PRIMARY_KEY) violated
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 2 RECORDS [ Start Time:
2014-8-26 16:19:51 End Time: 2014-8-26 16:19:51 Taken Time: 15455 micro-sec ]
$
$ cat t1.bad
"1","LKH"

```

날짜/ 시간 타입 포맷으로 인한 실패

DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE 데이터 타입에 대해 포맷을 설정할 수 있고 이로 인해 gloder를 이용한 업로드에 실패할 수 있다.

다음은 업로드 할 테이블을 보여준다.

```

gSQL> \DESC T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT * FROM T1;
no rows selected.

```

다음은 서버에서 TIMESTAM WITH TIME ZONE의 포맷을 보여준다.

```

gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE 'NLS_
TIMESTAMP_WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE INIT_VALUE
-----
YYYY-MM-DD HH24:MI:SS.FF6 TZh:TzM YYYY-MM-DD HH24:MI:SS.FF6 TZh:TzM

```

다음은 업로드 할 데이터 파일이다.

```

$ cat t1.dat
"4","20000108 00:00:00"
"5","20000108 04:05:06"
"6","20000108 04:05:06"

```

다음은 gloder로 업로드한 결과이다.

```

$ gloder test test -i -c t1.ct1 -d t1.dat

COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 0 RECORDS

```


다음은 업로드에 실패한 원인과 레코드가 기록된 로그 파일과 bad file이다.

```
$ cat t1.log
Err Rec(1) Col(2): HY000(12136): literal does not match format string
Err Rec(2) Col(2): HY000(12136): literal does not match format string
Err Rec(3) Col(2): HY000(12136): literal does not match format string
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 0 RECORDS [ Start Time:
2014-8-27 13:56:29 End Time: 2014-8-27 13:56:29 Taken Time: 20670 micro-sec ]
$ cat t1.bad
"4","20000108 00:00:00"
"5","20000108 04:05:06"
"6","20000108 04:05:06"
```

문제 해결

데이터 파일과 서버에서 사용하는 데이터 타입의 포맷이 달라서 생긴 문제이므로 데이터 파일의 데이터 타입 포맷을 설정해줘야 한다. 이 때 .odbc.ini를 사용한다.

다음은 데이터 타입의 포맷을 설정한 .odbc.ini 파일이다.

```
$ cat .odbc.ini
[GOLDILOCKS]
HOST = 127.0.0.1
PORT = 21123
DATE_FORMAT = SYYYYMMDD
TIME_FORMAT = HH24MISS
TIME_WITH_TIME_ZONE_FORMAT = HH24MISS TZHTZM
TIMESTAMP_FORMAT = SYYYYMMDD HHMISS
TIMESTAMP_WITH_TIME_ZONE_FORMAT = SYYYYMMDD HH:MI:SS
```

다음은 .odbc.ini를 설정하고 다시 gloader를 수행한 결과이다.

```
$ gloader test test -i -c t1.ct1 -d t1.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
```

노트

- .odbc.ini에 설정하는 데이터 타입의 포맷은 전체 column에 적용되고 각각의 column에 대해 개별적으로 설정할 수는 없다.
- .odbc.ini에 데이터 타입의 포맷을 설정하면 gloader로 다운로드 할 때도 적용된다.

용량 부족으로 인한 실패

다음은 gloder를 수행하고 실패한 결과를 보여준다.

```
$ gloder test test -i -c t1ctl -d t2.dat
loaded 4000 records into PUBLIC.T1
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 4200 RECORDS, SUCCEEDED 0 RECORDS
```

다음은 테이블스페이스의 데이터 파일에 용량이 부족하여 실패했다는 것을 보여준다.

```
$ cat t1.log
Err Rec(1) Col(-1): HY000(14015): there is no extendible datafile in tablespace 'MEM_DATA_TBS'
Err Rec(2) Col(-1): HY000(14015): there is no extendible datafile in tablespace 'MEM_DATA_TBS'
... 중략 ...
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 4200 RECORDS, SUCCEEDED 0 RECORDS [ Start Time:
2014-8-27 15:7:28 End Time: 2014-8-27 15:7:29 Taken Time: 317885 micro-sec ]
```

문제 해결

테이블스페이스의 데이터 파일을 확장하거나 추가한다.

자세한 내용은 ALTER TABLESPACE를 참조한다.

데이터 파일 분석 실패

Control 파일에서 기술한 field terminator와 qualifier, line terminator가 텍스트 데이터 파일 내에서 올바르게 사용되지 않아 일치하지 않을 수 있다.

다음은 업로드 할 테이블 대상의 구조이다.

```
gSQL>\desc T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(10) TRUE
I3 BINARY VARYING(10) TRUE
```

다음은 업로드 할 control 파일의 내용이다.

```
$ cat t1.ctl
TABLE T1
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

다음은 업로드 할 데이터 파일이다.

```
$ cat t1.dat
'1',"LKH","aa"
"2","SOS","aa"
"5","OKO","00"
```

다음은 gloader로 업로드한 결과이다.

```
$ gloader test test --import --control t1.ct1 --data t1.dat
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 1 RECORDS
```

다음은 업로드에 실패한 원인과 레코드를 보여준다.

```
$ cat t1.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 1 RECORDS [ Start Time:
2014-8-28 18:6:13 End Time: 2014-8-28 18:6:13 Taken Time: 18679 micro-sec ]
$
$ cat t1.bad
"2","SOS","aa"
'1',"LKH","aa"
```

문제 해결

데이터 파일 분석 실패의 주된 원인은 field terminator, qualifier, 또는 line terminator를 적절하게 사용하지 못한 것이다. 따라서 control 파일과 데이터 파일의 field terminator, qualifier와 line terminator를 확인하고 bad file에 기록된 레코드를 이용하여 데이터 파일을 수정해야 한다.

노트

데이터 파일 분석 과정에서 파싱에 실패한 레코드는 로그 파일에 기록되지 않고 bad file에 기록된다.

Character Set이 다른 Database에 업로드

Character set이 다른 database 사이에서 데이터를 binary 타입으로 다운로드/업로드 할 경우, column의 크기를 변환해야 할 수도 있다.

다음은 업로드 할 테이블 대상의 구조이다.

```
gSQL>\desc T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(36) TRUE
```

다음은 테이블의 데이터인데 VARCHAR 타입이고 크기는 36 이며 모두 채워져 있다.

```
SELECT * FROM T1;
I1 I2
-----
1 일이삼사오육칠팔구십일이삼사오육칠팔
```

위의 데이터를 UHC database에서 다운로드한 후에 동일한 스키마를 갖는 UTF8 database로 업로드하면 다음과 같은 에러가 발생한다.

```
$ gloder test test -i -f binary -T T1 -d t1.dup
ERR-HY000(42023): byte length of data greater than column length.
ERROR: FAILED TO IMPORT TABLE PUBLIC.T1
```

문제 해결

다음과 같이 column I2의 길이를 변경하면 데이터가 정상적으로 업로드된다.
I2를 VARCHAR(18 CHAR)로 선언하거나 I2를 VARCHAR(54) 크기로 선언한다.

```
gSQL>\desc T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(18 CHAR) TRUE
```

```
gSQL>\desc T1
COLUMN_NAME TYPE IS_NULLABLE
-----
I1 NUMBER(10,0) TRUE
I2 CHARACTER VARYING(54) TRUE
```

34.3 Control File 구문

```
TABLE [schema_name.]table_name[domain_name.]
FIELDS TERMINATED BY 'Field Terminator'
[OPTIONALLY ENCLOSED BY 'Open Qualifier' [AND 'Close Qualifier']]
[LINES TERMINATED BY 'Line Terminator']
[Character set character set_name]
[RTRIM [ON|OFF]]
[LTRIM [ON|OFF]]
[WHERE="conditional statement"]
```

주의

한 개의 control file은 한 개의 table에 대해 설명한다. 따라서 control file에는 위의 각 항목들이 한 번씩만 기술되어야 하며 중복하여 기술될 경우 control file 파싱 에러가 발생한다.

CHARACTERSET

구문

```
CHARACTERSET character set_name
```

설명

다운로드 또는 업로드 할 데이터 파일의 character set을 의미한다.

Character set이 표현되지 않았을 경우, GOLDILOCKS의 property 중에 CHARACTER_SET의 기본 character set 인 UTF8이 사용된다.

사용 예

다음은 다운로드 또는 업로드 할 데이터 파일의 character set이 ASCII 코드인 예이다.

```
% cat samplectl
CHARACTERSET ASCII
```

TABLE

구문

```
TABLE table_name
TABLE schema_name.table_name
TABLE table_name@domain_name
TABLE schema_name.table_name@domain_name
```

설명

데이터를 업로드 또는 다운로드 할 대상 테이블 이름과 테이블이 속한 스키마 이름, 도메인 이름을 기술한다.

도메인 이름은 데이터 다운로드에만 사용할 수 있는데 해당 멤버의 데이터만 다운로드 한다.

테이블 이름에는 문자열 또는 double quote (") 문자열이 사용된다.

테이블 이름이 gloder의 명령행 인자로도 주어질 경우, 명령행 인자로 주어진 값이 control file의 값보다 우선적으로 사용된다.

사용 예

다음은 테이블 이름만 기술하는 예이다.

```
% cat sample.ctl
TABLE lineitem
```

다음은 PUBLIC 스키마와 테이블 이름을 기술하는 예인데 위 예와 같은 의미를 갖는다.

```
% cat sample.ctl
TABLE PUBLIC.lineitem
```

다음은 G1N1 멤버의 PUBLIC 스키마와 테이블 이름을 기술하는 예인데 위 예와 같은 의미를 갖는다.

```
% cat sample.ctl
TABLE PUBLIC.lineitem@G1N1
```

다음은 delimited identifier로 생성한 테이블 이름을 기술하는 예이다.

```
gSQL> CREATE TABLE "Tab*&^" ( id INTEGER );
```

```
% cat sample.ctl
TABLE "Tab*&^"
```

FIELDS TERMINATED BY

구문

```
FIELDS TERMINATED BY 'Field Terminator'
```

설명

Field terminator는 데이터 파일의 record 데이터에서 각 column들 사이의 구분자로 사용된다. Control 파일에서 field terminator 설정은 생략할 수 없다.

Field terminator는 하나 이상의 문자열로 설정할 수 있으며, qualifier 및 line terminator와 중복되거나 서로 간의 부분 집합 문자열을 사용해서는 안된다.

Field terminator로 설정할 수 있는 문자열에는 제한이 없다. 다만 NEW LINE이나 TAB, CARRIAGE RETURN 문자를 설정하려면 각각 n, t, r 문자 앞에 역슬래시 (\) 또는 %를 추가해야 한다.

Field terminator가 gloder의 명령행 인자로도 주어질 경우, 명령행 인자로 주어진 값이 control file의 값보다 우선적으로 사용된다.

사용 예

다음은 field terminator를 COMMA와 NEW LINE으로 설정하는 예이다.

```
% cat samplectl
FIELDS TERMINATED BY ',\n'
```

```
% cat samplectl
FIELDS TERMINATED BY ',%n'
```

OPTIONALLY ENCLOSED BY

구문

```
OPTIONALLY ENCLOSED BY 'Open Qualifier' [AND 'Close Qualifier']
```

설명

Qualifier는 column의 시작과 끝을 표현하는 구분자이다.

Qualifier는 하나의 문자로만 설정할 수 있으며 첫 번째 qualifier는 open qualifier, 마지막 qualifier는 close qualifier라고 불린다. Open qualifier와 close qualifier를 같은 문자로 설정할 수도 있지만 각기 다른 문자로 설정하여 col

umn의 시작과 끝을 표현할 수도 있다.

Qualifier로 설정한 문자는 field terminator와 line terminator에서 사용할 수 없다.

Close qualifier가 문자 그대로 column 데이터에 포함될 경우, 두 개의 close qualifier를 사용하여 하나의 유효한 데이터임을 표현한다.

Close qualifier를 생략하고 open qualifier만 설정할 경우, close qualifier에는 open qualifier와 동일한 문자를 사용한다.

OPTIONALLY ENCLOSED BY 구문이 없는 경우, column의 데이터는 field terminator에 의해 구분된다.

Qualifier가 gloder의 명령행 인자로도 주어질 경우, 명령행 인자로 주어진 값이 control file의 값보다 우선적으로 사용된다.

노트

OPTIONALLY ENCLOSED BY 구문이 없는 경우, 데이터 다운로드와 업로드 결과가 서로 다를 수 있다. (Upload 문제 해결을 참조한다.)

사용 예

다음은 open qualifier와 close qualifier를 각각 double quote (")와 single quote (')로 표현한 예이다.

```
% cat sample.ctl
OPTIONALLY ENCLOSED BY '""' AND '"""'
```

LINES TERMINATED BY

구문

```
LINES TERMINATED BY 'Line Terminator'
```

설명

Line terminator는 데이터 파일에서 각 record들의 구분자로 사용된다.

Line terminator는 하나 이상의 문자열로 설정할 수 있으며, qualifier 및 field terminator와 중복되거나 서로 간의 부분집합 문자열을 사용해서는 안된다.

Line terminator 설정을 생략하면 NEW LINE('\n' 또는 '%n')이 기본값으로 사용된다.

Line terminator가 gloder의 명령행 인자로도 주어질 경우, 명령행 인자로 주어진 값이 control file의 값보다 우선적으로 사용된다.

노트

- Unix와는 달리 Windows OS에서 export 받은 데이터 파일에는 하나의 NEW LINE 대신에 CARRIAGE RETURN과 NEW LINE이 함께 기록된다. 이런 데이터파일을 이용하여 import 할 때는 '\r\n'과 같이 control 파일의 LINES TERMINATED BY를 명확히 설정해 주어야 정상적으로 데이터를 import할 수 있다.
- Field terminator와 line terminator는 각각 다른 문자열로 설정할 것을 권장한다. 첫 문자를 포함하여 일치하는 문자열이 많을수록 내부 비교 비용으로 인해 import 성능이 저하될 수 있다. 즉, field terminator와 line terminator를 각각 다른 문자열로 하되 해당 문자열의 길이가 짧을수록 성능이 향상될 수 있다.

사용 예

다음은 '^^\t\r\n'를 line terminator로 사용하는 예이다.

```
% cat sample.ctl
LINES TERMINATED BY '^^\t\r\n'
```

LTRIM

구문

```
LTRIM ON|OFF
```

설명

Left trim을 결정한다.

Default 값은 OFF이며, OFF로 설정할 경우 왼쪽 WHITESPACE를 data로 간주한다.

ON으로 설정할 경우 왼쪽 WHITESPACE는 무시한다.

노트

OPTIONALLY ENCLOSED BY 구문이 없을 경우에만 적용된다.

OPTIONALLY ENCLOSED BY 구문이 사용되고 데이터 파일에서 column이 구분자로 둘러싸여 있을 경우, RTRIM과 LTRIM은 OFF이다.

사용 예

다음은 LTRIM을 ON하는 예이다.

```
% cat sample.ctl  
LTRIM ON
```

RTRIM

구문

```
RTRIM ON|OFF
```

설명

Rigth trim을 결정한다.

Default 값은 OFF이며, OFF로 설정할 경우 오른쪽 WHITESPACE를 data로 간주한다.

ON으로 설정할 경우 오른쪽 WHITESPACE는 무시한다.

노트

OPTIONALLY ENCLOSED BY 구문이 없을 경우에만 적용된다.

OPTIONALLY ENCLOSED BY 구문이 사용되고 데이터 파일에서 column이 구분자로 둘러싸여 있을 경우, RTRIM과 LTRIM은 OFF이다.

사용 예

다음은 RTRIM을 ON 하는 예이다.

```
% cat sample.ctl  
RTRIM ON
```

WHERE

구문

```
WHERE="conditional_statement"
```

설명

데이터를 다운로드 할 경우 조건절을 사용한다.

사용 예

다음은 WHERE를 사용하는 예이다.

```
% cat sample.ctl  
WHERE="I2 > 3"
```

34.4 gloader Argument References

사용법

```
$ gloader --help
```

Usage

```
gloader user password mode data [control] [format] [options]
```

```
user          user name
```

```
password      password
```

mode: gloader's mode.

```
--export      export data
```

```
--import      import data
```

data:

```
--data        data file
```

options:

```
--control      control file
```

```
--format       file format(text|binary, Default text)
```

```
--log          log file
```

```
--bad          bad file
```

```
--dsn          dsn string
```

```
--array        number of rows in bind array(Default 1000)
```

```
--filesize     max file size
```

```
--commit       number of commit unit(Default 5000)
```

```
--comment      commenting on commit
```

```
--atomic       use atomic function
```

```
--parallel     use parallel in import
```

```
--propagation  enabling or disabling a redo log propagation(ON|OFF)
```

```
--errors       number of error count to allow(Default 100)
```

```
--AsTIMESTAMP  bind DATE as TIMESTAMP
```

```
--buffered     buffered disk io(Default direct io)
```

```
--tablename    [schema_name.]table_name[@domain_name]
```

```
--fieldterm    field terminator
```

```
--lineterm     line terminator
```

```
--qualifier    qualifier(column data encloser)
```

```
--where        export only rows selected by given WHERE condition
```

```
--group-id     importing distributed data by group id using global connections in a
```

clustered environment

```
--directio-size direct io size(Default 512)
```

<code>--no-copyright</code>	suppresses the display of the banner
<code>--silent</code>	suppresses the display of the result message
<code>--help</code>	print help message

필수 인자

Database에 연결하기 위해 user name, password의 순서로 인자가 입력된다.

gloader의 작업에 대한 mode, control file과 datafile이 인자로 입력된다.

인자	설명
user_name	User name이다. User name의 최대 길이는 128 이다.
password	Password이다. Password의 최대 길이는 128 이다.

--export

설명

gloader가 database에서 데이터를 다운로드하도록 기술한다.

사용 예

```
$ gloader test test --export -c sample.ctl -d sample.dat
```

--import

설명

gloader가 database에서 데이터를 업로드 하도록 기술한다.

사용 예

```
$ gloader test test --import -c sample.ctl -d sample.dat
```

--control

설명

Control file 경로를 기술한다.

--tablename이 argument 로 주어질 경우 control file을 생략할 수 있다. gloader를 수행하려면 테이블 이름이 필

수 요소이므로 반드시 control file이나 argument를 통해 테이블 이름이 주어져야 한다. 만일 테이블 이름이 argument로 주어지지 않을 경우, control file을 통해 반드시 TABLE 항목을 설정해야 한다.

Control file이 생략되면 field terminator, qualifier, line terminator를 argument로 부여할 수 있다. 만약 이러한 구분자들이 argument로도 주어지지 않을 경우, CSV 형식의 구분자들을 기본으로 사용한다.

사용 예

다음은 sample.ctl 이라는 이름을 가진 control 파일을 사용하는 예이다.

```
$ gloder test test -i --control sample.ctl -d sample.dat
```

--data

설명

다운로드/ 업로드 할 data file 경로를 기술한다.

사용 예

다음은 sample.dat 라는 이름을 가진 데이터 파일을 업로드하는 예이다.

```
$ gloder test test -i -c sample.ctl --data sample.dat
```

옵션 인자

--tablename

설명

[Schemaname.]Tablename[@domain_name] 형식으로 테이블 이름을 기술한다.

Argument 또는 control file의 TABLE 설정을 통해 테이블 이름을 반드시 부여해야 한다. 즉, control file이 생략되었을 경우, 테이블 이름 argument가 필수이다.

만약 control file의 TABLE 항목이 설정되어 있는 상태에서 tablename argument도 주어졌다면 argument 값을 우선적으로 사용한다.

사용 예

다음은 control file argument가 생략되어 tablename argument를 부여함으로써 CSV 형식으로 업로드하는 예이다.

```
$ gloader test test -i --tablename PUBLIC.T1 --data sample.dat
```

다음은 control file에 TABLE 항목이 설정되어 tablename argument를 생략한 예이다.

```
$ cat sample.ctl | grep TABLE
TABLE T1
$ gloader test test -i -c sample.ctl --data sample.dat
```

--format

설명

Datafile의 형식에 대해서 기술한다.

Datafile에는 text 형식과 binary 형식이 있고, 설정되지 않았을 경우 text 형식이 기본으로 사용된다.

사용 예

다음과 같이 데이터가 sample.dat 파일에 binary 형식으로 다운로드 된다.

```
$ gloader test test --export --control sample.ctl --data sample.dat --format binary
```

다음은 위에서 gloader를 실행한 후에 생성된 sample.dat와 sample.log 파일을 보여준다.

```
$ ls
sample.ctl sample.dat sample.log
```

--log

설명

Logfile 경로를 기술한다.

gloader를 수행하면서 발생하는 에러와 작업 결과를 기록한다.

설정되지 않았을 경우 확장자를 log로 바꾸어 datafile 이름과 동일한 이름으로 생성된다.

사용 예

다음은 --log 옵션을 사용하여 로그 파일 이름을 명시하는 예이다.

```
$ gloader test test --export --control sample.ctl --data sample.dat --log SAMPLE.log
```

다음은 위에서 gloader를 실행한 후 생성된 sample.dat와 SAMPLE.log 파일을 보여준다.

```
$ ls
sample.ctl sample.dat SAMPLE.log
```

다음은 --log 옵션을 사용하여 로그 파일 이름을 명시하지 않은 예이다.

```
$ gloder test test --export --control sample.ctl --data sample.dat
```

다음은 위에서 gloder를 실행한 후 생성된 sample.dat와 sample.log 파일을 보여준다.

```
$ ls
sample.ctl sample.dat sample.log
```

--bad

설명

Bad file 경로를 기술한다.

gloder가 import 작업을 할 때만 유효하며 오류가 발생하여 업로드 되지 못한 row를 저장한다.

설정하지 않았을 경우, 확장자를 bad로 바꾸어 datafile 이름과 동일한 이름으로 생성된다.

사용 예

다음은 --bad 옵션을 사용하여 bad file 이름을 명시하는 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --bad SAMPLE.bad
```

다음은 위에서 gloder를 실행한 후에 생성된 sample.dat, sample.log와 SAMPLE.bad 파일을 보여준다.

```
$ ls
SAMPLE.bad sample.ctl sample.dat sample.log
```

다음은 --bad 옵션을 사용하여 bad file 이름을 명시하지 않은 예이다.

```
$ gloder test test --export --control sample.ctl --data sample.dat
```

다음은 위에서 gloder를 실행한 후 생성된 sample.dat, sample.log와 sample.bad 파일을 보여준다.

```
$ ls
sample.bad sample.ctl sample.dat sample.log
```


--dsn

설명

dsn string이며 최대 길이는 128이다.

다운로드/업로드 할 데이터 파일에 표현된 시간 관련 데이터 타입의 형식에 대해 기술해야 할 때 사용된다.

Client/Server (C/S) 환경에서 gloadernet을 이용하여 서버를 지정할 때 사용된다.

자세한 내용은 **odbc.ini** 파일을 참조한다.

사용 예

다음은 --dsn 옵션을 사용하기 위해 기술된 .odbc.ini의 예이다.

```
$ cat .odbc.ini
[GOLDILOCKS]
HOST = 127.0.0.1
PORT = 21123
DATE_FORMAT = SYYYYYMMDD
TIME_FORMAT = HH24MISS
TIME_WITH_TIME_ZONE_FORMAT = HH24MISS TZHTZM
TIMESTAMP_FORMAT = SYYYYYMMDD HHMISS
TIMESTAMP_WITH_TIME_ZONE_FORMAT = SYYYYYMMDD HH:MI:SS
```

다음은 이름이 goldilocks인 서버에 --dsn 옵션을 사용하여 데이터를 업로드하는 예이다.

```
$ gloadernet test test --dsn goldilocks --import --control sample.ctl --data sample.dat
```

다음은 사용자가 --dsn 옵션을 사용하여 데이터 파일의 시간 관련 데이터 타입의 형식을 정의하여 다운로드하는 예이다.

```
$ gloader test test --dsn goldilocks --export --control sample.ctl --data sample.dat
```

자세한 내용은 **날짜/ 시간 타입 포맷으로 인한 실패**를 참조한다.

--array

설명

데이터를 import 할 때 bind하는 row의 개수를 기술한다.

Array의 개수에 비례하여 메모리의 사용량이 필요한데 설정되지 않았을 경우 1,000 (rows)가 사용된다.

이 옵션은 데이터를 업로드 할 때만 적용된다.

사용 예

다음은 데이터 파일을 2,000 개의 레코드 단위로 업로드하는 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --array 2000
```

--filesize

설명

gloder가 데이터를 다운로드 할 때 파일의 최대 크기를 설정할 수 있고, 데이터가 기술한 최대 크기를 초과하면 파일 확장자 뒤에 순열 번호를 더하여 파일을 생성한다.

설정하지 않았을 경우, 파일의 최대 크기는 무한대이고, 최소값은 31,461,376 (30 Mbytes)이다.

Binary 타입 데이터 파일에만 유효한 옵션이다.

사용 예

명령어를 실행한 결과 파일의 최대 크기를 초과하여 생성된 결과물은 다음과 같다.

```
$ gloder test test --export --control sample.ctl --data sample.dat --filesize 31461376
```

```
$ls -al
-rw-r--r-- 1 test test 31461376 sample.dat
-rw-r--r-- 1 test test 31461376 sample.dat.001
-rw-r--r-- 1 test test 6715392 sample.dat.002
```

--commit

설명

gloder가 데이터를 업로드 할 때 트랜잭션은 no commit 상태이다. --commit option을 사용하여 레코드의 commit 주기를 설정할 수 있다.

설정하지 않았을 경우 5000 (rows)가 사용된다.

사용 예

다음은 --commit 옵션을 사용하는 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --commit 10000
```

--comment

설명

gloader가 트랜잭션을 commit 할 때 트랜잭션에 주석을 지정한다.

사용 예

다음은 --comment 옵션을 사용하는 예이다.

```
$ gloader test test --import --control sample.ctl --data sample.dat --comment import_sample
```

--atomic

설명

Array INSERT를 수행하는 옵션으로써 데이터를 업로드 할 때만 유용하다.

Atomic array INSERT는 array 크기만큼의 insert 문을 하나의 트랜잭션으로 처리하기 때문에 기존의 array insert 보다 빠르게 처리된다.

사용 예

다음은 --atomic 옵션을 사용하는 예이다.

```
$ gloader test test --import --control sample.ctl --data sample.dat --array 1000 --atomic
```

--parallel

설명

병렬 처리할 thread의 개수를 기술한다.

gloader가 업로드 작업을 수행할 때만 유용하며 --parallel 옵션을 조정하여 각 부분 작업을 진행하는 thread의 개수를 늘리면 빠른 처리가 가능하다.

Thread의 개수를 늘려서 빠르게 작업할 수 있지만 thread 개수는 작업환경에 따라 정해야 한다.

기본값은 1이고, 최대값은 32이다.

사용 예

다음은 데이터를 업로드 할 때 여덟 개의 thread를 사용하는 예이다. 여덟 개의 업로드 thread 이외에 데이터 파일 분석하는 thread와 읽기 전용 thread를 포함하여 총 열 개의 thread가 동작한다.

```
$ gloader test test --import --control sample.ctl --data sample.dat --parallel 8
```

--propagation

설명

업로드 트랜잭션의 로그를 다른 복제 서버에 전파할지 여부를 결정한다.
ON, OFF를 설정할 수 있다.

사용 예

다음은 업로드 트랜잭션 로그를 다른 복제 서버에 전파하지 않을 경우의 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --propagation off
```

--errors

설명

gloder가 데이터를 업로드 할 때 허용되는 에러 개수를 설정한다.
설정하지 않을 경우, 100 (rows)가 사용되고 0으로 설정할 경우, --errors 옵션을 무시한다.
--array 옵션의 크기보다 작은 경우 허용되는 에러 개수는 array 개수이다.

사용 예

다음은 --errors 옵션을 사용하는 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --errors 10000
```

--AsTIMESTAMP

설명

하위 호환을 위해 TIMESTAMP 형식으로 저장된 DATE 타입 데이터를 TIMESTAMP 형식으로 동작하도록 한다.
--AsTIMESTAMP 옵션이 사용되면 DATE 데이터에도 TIMESTAMP_FORMAT이 적용된다.

사용 예

다음은 --AsTIMESTAMP 옵션을 사용하는 예이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --AsTIMESTAMP
```

--buffered

설명

Datafile에 한해 direct IO를 대신 buffered IO를 사용하도록 한다.

Log file과 bad file은 buffered IO를 사용한다.

사용 예

다음은 --buffered 옵션을 사용하는 예이다.

```
$ gloader test test --import --control sample.ctl --data sample.dat --buffered
```

--fieldterm

설명

Field terminator를 부여한다.

Control file에도 설정되어 있을 경우 fieldterm argument 값이 우선적으로 사용된다.

NEW LINE, CARRIAGE RETURN, TAB의 경우에는 각각 n, r, t 앞에 %를 붙여서 사용한다.

Shell meta character로 사용되는 ', ", ₩, & 등의 문자는 되도록 사용하지 않아야 한다.

사용 예

다음은 --fieldterm 옵션을 사용하는 예이다.

```
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm "..."
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm ...
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm '...'
```

--lineterm

설명

Line terminator를 부여한다.

Control file에도 설정되어 있을 경우 lineterm argument 값이 우선적으로 사용된다.

자세한 입력 방법은 fieldterm argument와 같다.

사용 예

다음은 --lineterm 옵션을 사용하는 예이다.

```
$ gloder test test --i -T PUBLIC.test -d sample.dat --lineterm "..."
$ gloder test test --i -T PUBLIC.test -d sample.dat --lineterm ...
$ gloder test test --i -T PUBLIC.test -d sample.dat --lineterm '...'
```

--qualifier

설명

Column 데이터의 시작과 끝에 추가할 qualifier를 부여한다. Qualifier는 한 개 문자만 설정할 수 있다. Control file에도 설정되어 있을 경우 qualifier argument 값이 우선적으로 사용된다. 자세한 입력 방법은 fieldterm argument와 같다.

사용 예

다음은 --qualifier 옵션을 사용하는 예이다.

```
$ gloder test test --i -T PUBLIC.test -d sample.dat --qualifier "|"
$ gloder test test --i -T PUBLIC.test -d sample.dat --qualifier ''
```

--where

설명

export 동작에서 조건절을 설정하여 데이터를 다운로드 한다. Control file에도 where절이 설정되어 있을 경우 where argument 값이 우선적으로 사용된다.

사용 예

다음은 --where 옵션을 사용하는 예이다.

```
$ gloder test test --i -T PUBLIC.test -d sample.dat --where "I2 > 4"
```

--group-id

설명

클러스터 환경에서 sharded 테이블의 데이터만 그룹별로 업로드 한다. 이 옵션을 non-sharded 테이블에 데이터를 업로드 할 때 사용할 경우, 해당 옵션은 유효하지 않다. 이 옵션은 CS 환경에서만 작동되므로 gloadernet에서만 유효하며 텍스트 파일을 업로드 할 때만 사용할 수 있다.

--group-id 옵션은 ODBC의 GLOBAL CONNECTION을 사용하므로 데이터 원본 구성에 관련 내용을 설정해야 한다.

사용 예

다음은 global connection을 사용하기 위한 odbc.ini 설정 파일이다.

```
$cat .odbc.ini
[GOLDILOCKS]
HOST=127.0.0.1
PORT = 22581
LOCALITY_AWARE_TRANSACTION=1
LOCATOR_DSN = LOCATOR
[LOCATOR]
LOCATOR_FILE=.locator.ini
```

다음은 gloadernet에서 --group-id 옵션을 사용하는 예이다.

```
$ gloadernet test test --i -T PUBLIC.test -d sample.dat --group-id
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 20 RECORDS, SUCCEEDED 20 RECORDS
```

다음은 gloader에서 --group-id 옵션을 사용하여 에러가 발생한 예이다.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --group-id
ERR-HY010(19009): Function sequence error : The function should be called only when the SQL_
ATTR_LOCALITY_AWARE_TRANSACTION connection attribute is set.
```

--directio-size

설명

gloader는 기본적으로 direct IO를 사용한다. Direct IO 크기의 기본값은 512 이지만 이 크기는 --directio-size를 사용하여 변경할 수 있다. 크기값은 512의 2 거듭 제곱 값이어야 한다.

사용 예

다음은 gloader에서 direct IO 크기를 변경하는 예이다.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --directio-size 1024
```

--no-copyright

설명

Copyright와 version을 출력하지 않는다.

사용 예

다음은 --no-copyright 옵션과 함께 gloder를 수행한 결과이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --no-copyright  
COMPLETED IN EXPORTING TABLE: PUBLIC.t1, 3 RECORDS  
$
```

--silent

설명

gloder 수행 결과를 출력하지 않는다.

사용 예

다음은 --silent 옵션과 함께 gloder를 수행한 결과이다.

```
$ gloder test test --import --control sample.ctl --data sample.dat --silent  
$
```

--help

설명

도움말을 출력한다.

자세한 내용은 **사용법**을 참조한다.

35.

gdump

35.1 gdump 개요

정의

gdump는 GOLDDILOCKS에서 제공하는 유틸리티로써 database가 관리하는 다음과 같은 binary file들을 text 형태로 dump한다.

- Control file
- Datafile
- Log file
- Incremental backup file
- Property file
- Commit log file
- Log buffer file
- Pending log buffer file

인자

gdump는 필수 인자와 옵션 인자로 구분되어 사용되고 옵션 인자는 덤프하려는 파일의 유형에 따라 다르게 사용된다

```
gdump file_type file_name [options]
```

필수 인자

필수 인자는 파일 유형, 파일 경로 순으로 사용된다.

```
file_type: CONTROL | LOG | DATA | PROPERTY | BACKUP | COMMIT_LOG | LOG_BUFFER | PEND_BUFFER
file_name: file name to dump
```

file_type 인자

- CONTROL: Control file이다. <GOLDDILOCKS_DATA>/wal 에 위치하고, 확장자는 .ctl 이다.
- Log: Log file (online/ archived redo log file) 이다. <GOLDDILOCKS_DATA>/wal 또는 <GOLDDILOCKS_DATA>/archive_log 에 위치하고, 확장자는 .log 이다.
- DATA: Datafile 이다. <GOLDDILOCKS_DATA>/db에 위치하고 확장자는 .dbf 이다.
- PROPERTY: Binary property file을 의미한다. <GOLDDILOCKS_DATA>/conf 에 위치하고 파일의 기본 이름은 goldilocks.properties.binary 이다.
- BACKUP: Incremental backup file을 의미한다. <GOLDDILOCKS_DATA>/backup 에 위치하고, 확장자는 .inc

이다.

- COMMIT_LOG: <GOLDILOCKS_DATA>/wal 에 위치한 commit.log 파일이다.
- LOG_BUFFER: 메모리에 있는 log buffer를 file로 저장한 것이다. gsyncher를 실행하면 log buffer를 file로 저장한다.
- PEND_BUFFER: 메모리에 있는 pending log buffer를 file로 저장한 것이다. gsyncher를 실행하면 pending log buffer를 file로 저장한다.

주의

깨진 control file에 대해 덤프를 시도하면 실패한다.

옵션 인자

```
-S --silent                silent
file_type = CONTROL
    -s, --section sys | log | db | backup | all dump controlfile section(default all)
file_type = DATA
    -h, --header                dump datafile header
    -n, --number INTEGER ( >= 0 )    (log or page) sequence number
    -f, --fetch  INTEGER ( >= 1 )    dump as much as count
file_type = LOG
    -h, --header
    -n, --number  INTEGER ( >= 0 )    (log or page) sequence number
    -o, --offset  INTEGER              offset of lsn(usable if set lsn)
    -f, --fetch   INTEGER ( >= 1 )    dump as much as count
    -a, --all
file_type = BACKUP
    -b, --body    header | all        dump incremental body(default all)
    -t, --tbs     dump specific tablespace body(usage if set body)
    -n, --number  INTEGER ( >= 0 )    (log or page) sequence number
    -f, --fetch   INTEGER ( >= 1 )    dump as much as count
```

설명

인자 이름	사용 가능한 파일 유형	설명
--silent	All	Copyright와 version, 수행 시간 등을 표시하지 않는다.
		덤프할 control file의 섹션을 의미한다. • sys: System section • log: Log section

인자 이름	사용 가능한 파일 유형	설명
--section	Control file	- db: Database section - backup: Incremental backup section - all: All sections (기본값)
--header	Datafile	Datafile을 덤프할 때 헤더만 덤프한다. 설정되지 않으면 헤더를 표시하지 않는다.
	Log file	Log file을 덤프할 때 헤더만 덤프한다.
--number	Datafile	덤프할 page 번호를 나타낸다. (기본값은 0이다.)
	Log file	덤프할 log 번호를 나타낸다. (기본값은 0이고 지정한 번호보다 큰 log를 덤프한다.)
	Incremental back up file	덤프할 page 번호를 나타낸다. (기본값은 0이고 지정한 번호보다 큰 page를 덤프한다.)
--fetch	Datafile	덤프할 page 개수이다. (기본값은 1이다.)
	Log file	덤프할 log 개수이다. (기본값은 전체 log이다.)
	Incremental back up file	덤프할 page 개수이다. (기본값은 전체 page이다.)
--offset	Log file	--number 옵션 인자가 설정되었을 때만 유효하고, number 인자에 설정된 값에서 offset만큼 떨어진 위치를 의미한다. (기본값은 0이다.)
--body	Incremental back up file	Page의 헤더 (header) 또는 헤더와 페이지 (all)를 표시한다. 설정하지 않으면 아무것도 표시하지 않는다. (기본값은 none이다.)
--tbs	Incremental back up file	--body 옵션 인자가 설정되었을 때만 유효하다. Incremental backup file에서 특정 테이블스페이스 번호를 지정한다. 설정하지 않으면 모든 테이블스페이스를 의미한다.
--all	Log file	유효하지 않은 log를 포함하여 덤프한다.

노트

프로퍼티 파일을 덤프할 때는 옵션을 사용하지 않는다.

35.2 사용 예

Control File

- 시스템 영역 덤프

```
$ gdump control control_0.ctl --section sys --silent
[SYSTEM SECTION]
```

```
-----
SERVER STATE           : SERVICE
DATA STORE MODE       : TDS
LAST CHECKPOINT LSN   : 136976
INCREMENTAL BACKUP CHUNK COUNT : 1
INCREMENTAL BACKUP SECTION OFFSET : 512
LOG SECTION OFFSET    : 8704
DB SECTION OFFSET     : 13312
```

- 로그 영역 덤프

```
$ gdump control control_0.ctl --section log --silent
[LOG SECTION]
```

```
-----
DATABASE CREATION TIME : 2014-08-25 18:37:11.299610
[CHECKPOINT]
LID           : 0,54834,13
LSN           : 136976
RECOVERY LSN  : 136976
ARCHIVELOG MODE : ARCHIVELOG
LAST INACTIVATED LOGFILE SEQUENCE : -1
[LOG STREAM]
STATE         : ACTIVE
GROUP COUNT   : 4
BLOCK SIZE    : 512
FILE SEQUENCE : 0
[LOG GROUP #0]
STATE         : CURRENT
SIZE          : 104857600
MEMBER COUNT  : 1
FILE SEQUENCE : 0
```

PREV LAST LSN : -1

MEMBER #0 : "/home/lkh/work/product/Gliese/home/wal/redo_0_0.log"

- Database 영역 덤프

```
$ gdump control control_0.ctl --section db --silent
```

[DB SECTION]

[DATABASE]

TRANSACTION_TABLE_SIZE : 1024

UNDO_RELATION_COUNT : 128

TABLESPACE COUNT : 4

NEW TABLESPACE ID : 4

[TABLESPACE #0]

NAME : DICTIONARY_TBS

ATTRIBUTES : MEMORY | PERSISTENT | DICT

STATE : CREATED

LOGGING STATE : LOGGING

ONLINE STATE : ONLINE

EXTENT_SIZE : 8

RELATION_ID : 0

[DATAFILE #0]

SIZE : 134209536

STATE : CREATED

NAME : "/home/lkh/work/product/Gliese/home/db/system_dict.dbf"

- Incremental backup 영역 덤프

```
$ gdump control control_0.ctl --section backup --silent
```

[INCREMENTAL BACKUP SECTION]

[BACKUP #0]

FILE PATH : /home/lkh/work/product/Gliese/home/backup/databaseD20140825T183902L0S0.inc

BACKUP LSN : 136976

BACKUP LEVEL : 0

BACKUP OBJECT : database

[BACKUP #1]

FILE PATH : /home/lkh/work/product/Gliese/home/backup/controlD20140825T183904L0S0.inc

BACKUP LSN : 136976

BACKUP LEVEL : 0

BACKUP OBJECT : control

Datafile

- 데이터 파일 system_dict.dbf의 첫 번째 page를 덤프한다.

```
$ gdump data system_dict.dbf --silent
TABLESPACE ID : 0
DATAFILE ID : 0
PAGE SEQUENCE ID : 0

-----

[PHYSICAL HEADER] TYPE(EXT_BLOCK_MAP), FREENESS(FREE), LSN(107938), TIMESTAMP
(1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID
(0,0)

-----

0000    02000000004000000 A2A5010000000000 4643D5EE70010500 0000FFFF00000000
0020    0000000000000000 0000000000000000 0000000000000000 0300000000000000
0040    0000000000000000 0000000000000000 0000000000000000 0200000001000000
0060    0200000001000000 0320000001000000 0000000000000000 0000000000000000
0080    0000000000000000 0000000000000000 0000000000000000 0000000000000000
00A0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
00C0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
....
```

- 데이터 파일 system_dict.dbf의 100 번째 page부터 세 개의 page를 덤프한다.

```
$ gdump data system_dict.dbf --number 100 --fetch 3 --silent
TABLESPACE ID : 0
DATAFILE ID : 0
PAGE SEQUENCE ID : 100

-----

[PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(348), TIMESTAMP
(1408959438472006), PARENT RID(0,57,4), SEGMENT ID(4294967296), MAX VIEW SCN(0), AGABLE SCN
(0), SELF ID(0,1
00)

-----

0000    04000000003000000 5C01000000000000 4643D5EE70010500 0000390004000000
0020    00000000001000000 0000000000000000 0000000000000000 80C0317800000000
0040    0000000000000000 0000000000000000 0000000064000000 0100000001000002
0060    FFFFFFFFFFFFFFFF 0100000000000000 0000000000000000 0000000000000000
0080    0000000000000000 0200000001000000 7303000072030000 6E00010000000000
00A0    37000000000000000 0000000000000000 0000000000000000 0000000000000000
```

```
00C0      0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

```
....
```

```
1FC0      0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

```
1FE0      0000000000000000 0000000000000000 0000000000000000 5C01000000000000
```

```
TABLESPACE ID : 0
```

```
DATAFILE ID : 0
```

```
PAGE SEQUENCE ID : 101
```

```
-----
[PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(353), TIMESTAMP
(1408959438472006), PARENT RID(0,58,4), SEGMENT ID(4294967296), MAX VIEW SCN(0), AGABLE SCN
(0), SELF ID(0,1
01)
```

```
-----
0000      04000000003000000 61010000000000000 4643D5EE70010500 00003A0004000000
```

```
0020      00000000001000000 00000000000000000 00000000000000000 00BF317800000000
```

```
0040      00000000000000000 00000000000000000 00000000065000000 0100000001000002
```

```
0060      FFFFFFFFFFFFFFFF 01000000000000000 00000000000000000 0000000000000000
```

```
0080      00000000000000000 02000000010000000 8303000082030000 7000010000000000
```

```
00A0      38000000000000000 00000000000000000 00000000000000000 0000000000000000
```

```
00C0      00000000000000000 00000000000000000 00000000000000000 0000000000000000
```

```
....
```

```
1FC0      00000000000000000 00000000000000000 00000000000000000 0000000000000000
```

```
1FE0      00000000000000000 00000000000000000 00000000000000000 6101000000000000
```

```
TABLESPACE ID : 0
```

```
DATAFILE ID : 0
```

```
PAGE SEQUENCE ID : 102
```

```
-----
[PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(358), TIMESTAMP
(1408959438472006), PARENT RID(0,59,4), SEGMENT ID(4294967296), MAX VIEW SCN(0), AGABLE SCN
(0), SELF ID(0,1
02)
```

```
-----
0000      04000000003000000 66010000000000000 4643D5EE70010500 00003B0004000000
```

```
0020      00000000001000000 00000000000000000 00000000000000000 C0BD317800000000
```

```
0040      00000000000000000 00000000000000000 00000000066000000 0100000001000002
```

```
0060      FFFFFFFFFFFFFFFF 01000000000000000 00000000000000000 0000000000000000
```

```
0080      00000000000000000 02000000010000000 9303000092030000 7200010000000000
```

```
00A0      39000000000000000 00000000000000000 00000000000000000 0000000000000000
```

```
00C0      00000000000000000 00000000000000000 00000000000000000 0000000000000000
```

```
....
```


- 데이터 파일 system_dict.dbf의 header를 덤프한다.

```
$ gdump data system_dict.dbf --header --silent
FILE                      : system_dict.dbf
Tablespace Physical Id : 0
Datafile Id              : 0
Last Checkpoint Lsn      : 136976
Creation TIME            : 2014-08-25 18:37:18.472006
```

Log File

- 온라인 redo log file인 redo_0_0.log 전체를 덤프한다.

```
$ gdump log redo_0_0.log --silent
=====
[LOG FILE HEADER]
-----
LOG_GROUP_ID      : 0
BLOCK_SIZE        : 512
FILE_SIZE         : 104857600
FILE_SEQUENCE     : 0
PREV_LAST_LSN     : -1
CREATION TIME     : 2014-08-25 18:37:13.315283
=====
[LOG #0] : BLOCK(0), LSN(0), SIZE(1132), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001), RID
(0,-1,0)
[PIECE #0] : TYPE(MEMORY_FILE_CREATE), SIZE(1116), CLASS(DATAFILE), REDO_TYPE(CONTROL_FILE),
PROPAGATE_LOG(YES), RID(0,0,0)
FFFFFFFF2F686F6D 652F6C6B682F776F 726B2F70726F6475 63742F476C696573    ..../home/lkh/work/
product/Glies
652F686F6D652F64 622F73797374656D 5F646963742E6462 6600000000000000    e/home/db/system _
dict.dbf.....
0000000000000000 0000000000000000 0000000000000000 0000000000000000    .....
.....
[LOG #1] : BLOCK(3), LSN(1), SIZE(232), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001), RID(0,0,0)
[PIECE #0] : TYPE(MEMORY_TBS_CREATE), SIZE(216), CLASS(DATAFILE), REDO_TYPE(CONTROL_FILE),
PROPAGATE_LOG(YES), RID(0,0,0)
... 중략 ...
```

- redo_0_0.log의 100번 log sequence number에서 다섯 개의 offset만큼 떨어진 log로부터 세 개의 log를 덤프

프한다.

```
$ gdump log redo_0_0.log --number 100 --offset 5 --fetch 3 --silent
```

[LOG FILE HEADER]

```
LOG_GROUP_ID      : 0
BLOCK_SIZE        : 512
FILE_SIZE         : 104857600
FILE_SEQUENCE     : 0
PREV_LAST_LSN     : -1
CREATION TIME     : 2014-08-25 18:37:13.315283
```

```
[LOG #0] : BLOCK(105), LSN(105), SIZE(24), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001), RID
(0,0,55)
```

```
[PIECE #0] : TYPE(INIT_RELATION_HEADER), SIZE(8), CLASS(ACCESS), REDO_TYPE(PAGE), PROPAGATE_
LOG(YES), RID(0,160,55)
```

0A0000000000000000

```
[LOG #1] : BLOCK(105), LSN(106), SIZE(240), PIECE_COUNT(4), TRANS_ID(FFFFFFFFFFFF0001), RID
(0,0,56)
```

```
[PIECE #0] : TYPE(INIT_PAGE), SIZE(88), CLASS(PAGE_ACCESS), REDO_TYPE(PAGE), PROPAGATE_LOG(YES
), RID(0,0,56)
```

```
0400000004000000 000000000000000000 4643D5EE70010500 00000D0004000000 ..... FC...
```

p.

```
00000000001000000 00000000000000000 00000000000000000 00000000000000000
```

.....

000000000000000000 000000000000000000 000000000380000000

8

```
[PIECE #1] : TYPE(MEMORY_BITMAP_UPDATE_LEAF_STATUS), SIZE(24), CLASS(SEGMENT), REDO_TYPE(PAGE
), PROPAGATE LOG(YES), RID(0.13.4)
```

04000000003000000 00000000000000000 00000000000000000

• • • • •

```
[PIECE #2] : TYPE(BYTES), SIZE(8), CLASS(RECOVERY), REDO_TYPE(PAGE), PROPAGATE_LOG(YES), RID
(0,4,56)
```

040000000030000000

```
[LOG #2] : BLOCK(105), LSN(107), SIZE(832), PIECE_COUNT(8), TRANS_ID(FFFFFFFFFFFF0001), RID
(0,-1,0)
```

```
[PIECE #0] : TYPE(INIT_PAGE), SIZE(88), CLASS(PAGE_ACCESS), REDO_TYPE(PAGE), PROPAGATE_LOG(YES
). RID(1.0.178)
```

```
010000000040000000 000000000000000000 BED7F8EE70010500 0000FFFF00000000
```

p.

```
000000000000000000 000000000000000000 000000000000000000 000000000000000000 .....
.....
000000000000000000 000000000000000000 01000000B2000000 .....
.....
```

Incremental Backup File

- Incremental backup 정보를 덤프한다.
 - 다음과 같이 databaseD20140825T183902L0S0.inc에서 파일의 헤더와 테일만 덤프한다.

```
$ gdump backup databaseD20140825T183902L0S0.inc --silent
INCREMENTAL FILE HEADER
-----
OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
LSN: PREV(0), MAX (136976), CHKPT(136976)
CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-----
INCREMENTAL FILE TAIL
-----
TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
```

- Incremental backup 전체 페이지에 대해 페이지 헤더와 페이지 바디를 덤프한다.
 - 다음과 같이 databaseD20140825T183902L0S0.inc에서 모든 테이블스페이스의 전체 페이지를 덤프한다.

```
$ gdump backup databaseD20140724T123414L0S0.inc --body all
INCREMENTAL FILE HEADER
-----
OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
LSN: PREV(0), MAX (136976), CHKPT(136976)
CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-----
INCREMENTAL FILE TAIL
-----
TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
TABLESPACE ID : 0
```

DATAFILE ID : 0

PAGE SEQUENCE ID : 0

 [PHYSICAL HEADER] TYPE(EXT_BLOCK_MAP), FREENESS(FREE), LSN(107938), TIMESTAMP
 (1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID
 (0,0)

```
0000    020000000040000000 A2A501000000000000 4643D5EE70010500 0000FFFF00000000
0020    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0040    000000000000000000 000000000000000000 000000000000000000 020000000100000000
0060    020000000010000000 032000000010000000 000000000000000000 000000000000000000
0080    000000000000000000 000000000000000000 000000000000000000 000000000000000000
```

... 중략 ...

TABLESPACE ID : 0

DATAFILE ID : 0

PAGE SEQUENCE ID : 1

 [PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(FREE), LSN(100016), TIMESTAMP
 (1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(4294967296), MAX VIEW SCN(0), AGABLE SCN
 (0), SELF ID(0,1)

```
0000    040000000040000000 B08601000000000000 4643D5EE70010500 0000FFFF00000000
0020    000000000010000000 000000000000000000 000000000000000000 802A327800000000
0040
```

... 중략 ...

TABLESPACE ID : 2

DATAFILE ID : 0

PAGE SEQUENCE ID : 24580

 [PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(24), TIMESTAMP(1408959441642963),
 PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,24580)

```
0000    030000000040000000 180000000000000000 D3A505EF70010500 0000FFFF00000000
0020    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0040    000000000000000000 000000000000000000 020000000046000000 1E001E0000001E00
```

... 중략 ...

- Incremental backup에 지정된 수보다 큰 페이지에 대해 페이지 헤더와 페이지 바디를 덤프한다.
 - 다음과 같이 databaseD20140825T183902L0S0.inc에서 모든 테이블스페이스의 1000번 이상인 페이지를 덤프한다.

```
$ gdump backup databaseD20140724T123414L0S0.inc --number 10000 --body all
```

```
INCREMENTAL FILE HEADER
```

```
-----
OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
LSN: PREV(0), MAX (136976), CHKPT(136976)
CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-----
```

```
INCREMENTAL FILE TAIL
```

```
-----
TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
TABLESPACE ID : 2
DATAFILE ID : 0
PAGE SEQUENCE ID : 16387
-----
```

```
[PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(22), TIMESTAMP(1408959441642963),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,16387)
```

```
-----
0000    030000000040000000 160000000000000000 D3A505EF70010500 0000FFFF00000000
0020    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0040    000000000000000000 000000000000000000 020000000034000000 00010001000000001
0060    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0080    000000000000000000 000000000000000000 000000000000000000 000000000000000000
-----
```

... 중략 ...

```
TABLESPACE ID : 2
DATAFILE ID : 0
PAGE SEQUENCE ID : 24580
-----
```

```
[PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(24), TIMESTAMP(1408959441642963),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,24580)
```

```
-----
0000    030000000040000000 180000000000000000 D3A505EF70010500 0000FFFF00000000
0020    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0040    000000000000000000 000000000000000000 020000000046000000 1E001E0000001E00
0060    000000000000000000 000000000000000000 000000000000000000 000000000000000000
0080    000000000000000000 000000000000000000 000000000000000000 000000000000000000
-----
```

... 이하 생략 ...

- Incremental backup의 지정된 테이블스페이스의 지정된 수에 대한 페이지 헤더와 바디를 덤프한다.
 - 다음과 같이 database D20140825T183902L0S0.inc에서 테이블스페이스 0번의 1000번 페이지를 덤프

한다.

```
gdump backup databaseD20140825T183902L0S0.inc --tbs 0 --number 1000 --body all --fetch 1 --
silent | more
```

```
INCREMENTAL FILE HEADER
```

```
-----
OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
LSN: PREV(0), MAX (136976), CHKPT(136976)
CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-----
```

```
INCREMENTAL FILE TAIL
```

```
-----
TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
TABLESPACE ID : 0
DATAFILE ID : 0
PAGE SEQUENCE ID : 1000
-----
```

```
[PHYSICAL HEADER] TYPE(UNFORMAT), FREENESS(FREE), LSN(6307), TIMESTAMP(1408959438472006),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(0,1000)
```

```
-----
0000      010000000040000000 A31800000000000000 4643D5EE70010500 0000FFFF00000000
0020      000000000000000000 000000000000000000 000000000000000000 000000000000000000
0040      000000000000000000 000000000000000000 0000000000E8030000 000000000000000000
0060      000000000000000000 000000000000000000 000000000000000000 000000000000000000
... 이하 생략 ...
```

Property File

- 다음과 같이 gsql을 이용하여 property 중 하나인 PAGE_CHECKSUM_TYPE을 설정한다.

```
gSQL> ALTER SYSTEM SET PAGE_CHECKSUM_TYPE = 1 SCOPE = BOTH;
System altered.
```

- 다음과 같이 GOLDILOCKS의 binary 프로퍼티 파일을 덤프한다.

```
$ gdump property goldilocks.properties.binary
```

```
=====
FILE: goldilocks.properties.binary
```

```
TYPE: PROPERTY-BINARY
TIME: 2014-08-29 12:45:13.853448
=====
PAGE_CHECKSUM_TYPE = 1
=====
TIME: 2014-08-29 12:45:13.853627
=====
```

Commit Log

다음과 같이 commit log file을 덤프한다.

```
$ gdump commit_log commit.log
=====
FILE: commit.log
TYPE: COMMIT LOG FILE
TIME: 2017-03-23 11:47:42.555368
=====
[COMMIT LOG FILE HEADER]
-----
FILE_SEQUENCE   : 0
FILE_SIZE       : 104857600
MAX_BLOCK_COUNT : 204800
SIGNATURE       : 5DAABFECBDCD11E68785A3701A679D47
=====
[BLOCK(0), LOG_COUNT(1)]
TRANS_ID(1.0.59899956), COMMIT_SCN(16603.645.17371), INDOUBT_BEHAVIOR(FORGET), SYNC_GLOBAL_
TABLE_SCN(TRUE), PREV_LOG_FILE_SEQ(-1), PREV_LOG_BLOCK_SEQ(-1), PREV_LOG_SLOT_SEQ(-1)
```

Log Buffer File

다음과 같이 gsyncher를 실행한 후에 생성된 log buffer file을 덤프한다.
자세한 내용은 **gsyncher**를 참조한다.

```
$ gdump log_buffer logbuffer.log
=====
FILE: logbuffer.log
```

```
TYPE: LOG BUFFER FILE
```

```
TIME: 2017-03-23 11:10:12.787400
```

```
=====
```

```
=====
```

```
[LOG BUFFER FILE HEADER]
```

```
-----
```

```
REAR_FILE_BLOCK_SEQ_NO   : 203410
```

```
FRONT_FILE_BLOCK_SEQ_NO  : 203410
```

```
REAR_SBSN                : 797213
```

```
REAR_LSN                 : 375933
```

```
REAR_LID                 : (3, 3254496, 161)
```

```
FRONT_SBSN               : 797212
```

```
FRONT_LSN                : 375933
```

```
FILE_SEQ_NO              : 3
```

```
BUFFER_SIZE              : 10485760
```

```
BUFFER_BLOCK_COUNT       : 20480
```

```
BLOCK_OFFSET             : 13
```

```
GROUP_COMMIT_LSN         : 336688
```

```
LOG_SWITCHING_SBSN       : -1
```

```
=====
```

Pending Log Buffer File

다음과 같이 gsyncher를 실행한 후에 생성된 pending log buffer file을 덤프한다.
자세한 내용은 **gsyncher**를 참조한다.

```
$ gdump pend_buffer pendbuffer.log
```

```
=====
```

```
FILE: pendbuffer.log
```

```
TYPE: PEND BUFFER FILE
```

```
TIME: 2017-03-23 11:10:29.603646
```

```
=====
```

```
=====
```

```
[PEND LOG BUFFER FILE HEADER]
```

```
PEND_LOG_BUFFER COUNT : 4
```

```
-----
```

```
PEND_LOG_BUFFER NO. : 0
```

```
PEND_BUFFER_SIZE       : 1048576
```

```
BUFFER_BLOCK_COUNT     : 16384
```

```
FRONT_PBSN             : 102
```



```

REAR_PBSN      : 102
FRONT_LSN      : -1
REAR_LSN       : 327055

```

```

-----
PEND_LOG_BUFFER NO. : 1
PEND_BUFFER_SIZE    : 1048576
BUFFER_BLOCK_COUNT  : 16384
FRONT_PBSN          : 0
REAR_PBSN           : 0
FRONT_LSN           : -1
REAR_LSN            : -1

```

```

-----
PEND_LOG_BUFFER NO. : 2
PEND_BUFFER_SIZE    : 1048576
BUFFER_BLOCK_COUNT  : 16384
FRONT_PBSN          : 0
REAR_PBSN           : 0
FRONT_LSN           : -1
REAR_LSN            : -1

```

```

-----
PEND_LOG_BUFFER NO. : 3
PEND_BUFFER_SIZE    : 1048576
BUFFER_BLOCK_COUNT  : 16384
FRONT_PBSN          : 0
REAR_PBSN           : 0
FRONT_LSN           : -1
REAR_LSN            : -1

```

```

=====

```


36.

tablediff

36.1 tablediff 개요

배경

GOLDILOCKS 시스템 운영자는 예상치 못한 장애로 인해 cyclone이나 LogMirror 툴로 동기화된 GOLDILOCKS의 두 테이블의 동기화가 어긋나는 경우에 대비할 필요가 있다. 동기화가 어긋났다는 것은 특정 row가 한 쪽 테이블에만 존재하거나 row의 column 내용 일부가 서로 다른 경우를 말한다. 혹은 이런식으로 동기화가 어긋나지 않았더라도 cyclone과 LogMirror는 테이블의 동기화 여부를 알려주지 않기 때문에 운영 시간을 피해서 테이블의 동기화 여부를 검증하고 비동기화된 row에 대한 동기화를 수행해야 한다.

기능

tablediff는 CDC 등의 툴로 동기화된 GOLDILOCKS의 두 테이블을 row 단위로 비교하는 툴이다. 특정 row가 한 테이블에만 있거나 row의 내용이 다를 경우 리포팅하고 동기화 해준다. 각종 동작을 제어할 수 있는 설정 파일을 입력 받고 비동기 row를 알리는 log 파일과 동기화 수행 결과를 알려주는 log 파일을 출력한다.

두 테이블의 스키마는 동일해야 하며 primary key를 반드시 포함하고 있어야 한다는 제약 조건이 있다. 실시간으로 테이블의 내용이 갱신될 수 있으므로 현재 트랜잭션이 발생하고 있는 테이블에 대해 이 툴을 수행하는 것은 권장하지 않는다. 다만 조회 (SELECT 구문)가 발생하고 있을 때는 문제가 되지 않는다. 이 툴은 GOLDILOCKS의 테이블에 대해서만 사용할 수 있다.

이 툴은 TableDiff와 TableSync라는 두 개의 실행 명령어를 가진다.

TableDiff

```
java sunje.goldilocks.tool.diff.TableDiff [configure file]
```

TableDiff 프로그램은 두 테이블의 비동기화 여부를 검증하고 옵션에 따라 비동기 row를 즉시 동기화하거나 동기화를 나중에 수행할 수 있도록 동기화 정보 (binary 파일)를 남기도록 할 수 있다. 즉시 동기화하는 thread는 다중으로 실행되도록 할 수 있다.

TableSync

```
java sunje.goldilocks.tool.diff.TableSync [configure file]
```

TableSync 프로그램은 앞서 TableDiff가 남긴 동기화 정보를 이용하여 동기화를 수행한다. (Row 비교 작업은 하지 않는다.) 멀티 thread를 구동하여 동시에 진행할 수 있다.

노트

TableDiff에서 남긴 bin 파일에는 configure 파일이 적용되지 않은 두 table의 비동기화된 row가 저장되어 있는데 TableSync에서는 이 bin 파일을 이용하여 두 table을 강제로 동기화한다.

특징

이 툴은 Java 프로그램으로써 jar 파일 형태로 제공된다. 따라서 java(1.6)가 있어야 수행할 수 있다. 또한 GOLDILOCKS JDBC 드라이버를 사용하기 때문에 goldilocks6.jar가 필요하다. TCP/ IP로 GOLDILOCKS에 접속하기 때문에 원격으로 실행할 수 있고 독자적인 프로토콜을 사용하여 JDBC나 ODBC로 테이블을 검증하는 것보다 훨씬 빠르게 실행된다.

Row 비교 작업과 동기화 작업은 멀티 thread로 동시에 작업할 수 있다. Row 비교 작업을 다중화하기 위해서는 테이블의 key의 범위를 균등하게 나누는 작업을 수동으로 해줘야 한다. 사용자가 열 개의 범위로 나눠주면 (설정 파일에 정의할 수 있다.) 열 개의 thread가 테이블 비교 작업을 수행한다. 이에 반해 동기화 작업은 지정된 thread 개수만큼의 thread로 수행된다.

파일 구성

tablediff 프로그램은 \$GOLDILOCKS_HOME/bin/tablediff.jar라는 단 하나의 파일로 구성된다. 실행하려면 \$GOLDILOCKS_HOME/lib/goldilocks6.jar 파일도 필요하다. 그리고 입력 인자로 설정 파일이 필요한데 \$GOLDILOCKS_DATA/conf/tablediff.conf 샘플 파일을 참조하면 된다.

36.2 사용법

커맨드 사용법

tablediff는 Java 프로그램이기 때문에 Java (JRE 1.6 또는 JDK1.6)가 필요하다. tablediff.jar 파일과 goldilocks6.jar 파일을 CLASSPATH에 포함시키거나 Java의 -classpath 옵션으로 지정해주면 된다. 다음과 같은 커맨드로 tablediff 프로그램을 구동할 수 있다.

```
export CLASSPATH=$CLASSPATH:$GOLDILOCKS_HOME/bin/tablediff.jar:$GOLDILOCKS_HOME/lib/goldilocks6.jar
java sunje.goldilocks.tool.diff.TableDiff [configure file]
```

또는

```
java -classpath $GOLDILOCKS_HOME/bin/tablediff.jar:$GOLDILOCKS_HOME/lib/goldilocks6.jar sunje.goldilocks.tool.diff.TableDiff [configure file]
```

다음은 간단한 샘플 테이블에 대해 TableDiff를 실행한 결과이다.

```
gSQL> create table tab1 ( c1 integer primary key, c2 char(10) );
gSQL> create table tab2 ( c1 integer primary key, c2 char(10) );
gSQL> insert into tab1 values ( 1, 'HELLO');
gSQL> insert into tab2 values ( 1, 'HELLO');
gSQL> insert into tab1 values ( 2, 'WORLD');
gSQL> insert into tab2 values ( 2, 'world');
gSQL> insert into tab1 values ( 3, 'good');
gSQL> insert into tab2 values ( 4, 'good');
gSQL> commit;
shell> java sunje.goldilocks.tool.diff.TableDiff tablediff.conf
Total 4 rows processed
> row diff          : 1, update target(success/failure): 1/0
> key diff source only: 1, insert into target(success/failure): 1/0
> key diff target only: 1, delete from target(success/failure): 1/0
TableDiff completed
elapsed time = 0.229 sec
```

- tablediff.conf의 내용

```

SOURCE_URL      = jdbc:goldilocks://127.0.0.1:22581/test
SOURCE_USER     = TEST
SOURCE_PASSWORD = test
SOURCE_SCHEMA   = PUBLIC
SOURCE_TABLE    = TAB1
TARGET_URL      = jdbc:goldilocks://127.0.0.1:22581/test
TARGET_USER     = TEST
TARGET_PASSWORD = test
TARGET_SCHEMA   = PUBLIC
TARGET_TABLE    = TAB2
OPERATION       = SYNC
TARGET_INSERT   = ON
TARGET_UPDATE   = ON
TARGET_DELETE   = ON
SOURCE_INSERT    = OFF

```

Property Option

tablediff의 설정 파일에서 사용할 수 있는 옵션들을 설명한다.

SOURCE, TARGET 테이블 설정 옵션

이들은 source와 target 테이블을 정의하는 옵션들로서 반드시 정의되어야 한다. Source와 target은 각각 비교 대상이 되는 테이블을 지칭한다.

- SOURCE_URL: Source 테이블이 존재하는 GOLDDILOCKS의 JDBC connection URL이다.
- SOURCE_USER: Source 테이블이 존재하는 GOLDDILOCKS의 계정이다.
- SOURCE_PASSWORD: Source 테이블이 존재하는 GOLDDILOCKS 계정의 password이다.
- SOURCE_SCHEMA: Source 테이블의 스키마이다.
- SOURCE_TABLE: Source 테이블 이름이다.
- TARGET_URL: Target 테이블이 존재하는 GOLDDILOCKS의 JDBC connection URL이다.
- TARGET_USER: Target 테이블이 존재하는 GOLDDILOCKS의 계정이다.
- TARGET_PASSWORD: Target 테이블이 존재하는 GOLDDILOCKS 계정의 password이다.
- TARGET_SCHEMA: Target 테이블의 스키마이다.
- TARGET_TABLE: Target 테이블 이름이다.

다음 예제를 참조한다.

```

SOURCE_URL      = jdbc:goldilocks://192.168.0.100:22581/test
SOURCE_USER     = TEST
SOURCE_PASSWORD = test
SOURCE_SCHEMA   = PUBLIC
SOURCE_TABLE    = T1
TARGET_URL      = jdbc:goldilocks://192.168.0.101:22581/test
TARGET_USER     = TEST
TARGET_PASSWORD = test
TARGET_SCHEMA   = PUBLIC
TARGET_TABLE    = T2

```

Operation

TableDiff의 동작을 결정한다. DIFF로 동작하면 동기화 정합성 검증만 수행하여 결과를 알리고, SYNC로 동작하면 정합성을 검증하면서 동기화 작업을 함께 수행한다.

DIFF로 동작하면 (DIFF_BIN_FILE 속성으로 명시한 파일명으로) 정합성 결과 파일이 생성되는데 이 파일을 가지고 TableSync를 수행할 수 있다.

동기화 정책 설정

다음과 같이 동기화 정책을 제어하는 네 가지 속성이 있다. 모두 ON이나 OFF 값을 가질 수 있다.

- TARGET_INSERT: Source에 있는 key가 target에 없으면 target에 그 key를 삽입한다.
- TARGET_UPDATE: Key가 아닌 column의 내용이 다르면 target의 row를 갱신한다.
- TARGET_DELETE: Source에 없는 key가 target에 있으면 target의 row를 삭제한다.
- SOURCE_INSERT: Source에 없는 key가 target에 있으면 source에 그 row를 삽입한다.

노트

TARGET_DELETE와 SOURCE_INSERT가 모두 ON일 수는 없다.

EXCLUDED_COLUMNS

비교 대상에서 제외할 column을 명시한다. 콤마 (,)를 구분자로 사용하여 column 이름을 명시하면 된다. Key column은 제외할 수 없다.

WHERE_CLAUSE

테이블의 비교 대상 row에 대한 조건을 설정할 수 있다. 예를 들어 WHERE_CLAUSE = SALARY >= 1000000 이라고 명시하면 테이블의 row 중에 SALARY column 값이 1,000,000 이상인 row들에 대해서만 정합성 검사가 이루어진다.

DISPLAY_ROW_UNIT

TableDiff 프로그램은 테이블 비교 작업의 진행 상황을 표시하는데 특정 개수의 row를 처리할 때마다 콘솔로 몇 개의 row가 처리되었는지 출력한다. 이 속성은 몇 개의 row마다 출력할 것인지를 설정한다. 기본값은 100,000인데 생략할 수 있다.

SYNC_OUT_FILE

동기화 결과를 기록할 파일 이름을 설정한다. 정의되지 않으면 tablesync.log에 기록한다. 만일 동기화 thread가 여러 개인 경우, 파일 이름 뒤에 숫자가 붙는다.

DIFF_OUT_FILE

Row 불일치 결과를 기록할 파일 이름을 설정한다. 정의되지 않으면 tablediff.log에 기록한다. 이 파일은 사람이 읽을 수 있는 텍스트 파일 형태이다.

DIFF_BIN_FILE

OPERATION 속성을 DIFF로 설정하면 TableDiff는 동기화 정보 파일을 남기는데 이 속성이 정의한 파일명으로 남긴다. TableSync는 이 파일을 입력 인자로 사용한다.

PROPAGATE_REDO_LOG

Row 동기화 작업 로그를 다른 복제 서버에 전파할지 여부를 결정한다. 기본값은 OFF 이다.

LOGGING_ON_SUCCESS

Row 동기화 작업을 할 때 사용한 DML (INSERT, UPDATE, DELETE)이 성공했을 경우에도 기록을 남길 것인지 여부를 설정한다. 기본값은 OFF 이며 이 값이 설정되어 있으면 동기화 작업 속도가 떨어진다. DML이 실패할 경우에는 이 속성과 관계없이 로깅 정보를 남긴다.

LOGGING_ON_DIFF

Row 정합성을 비교하여 불일치가 발생했을 때 로깅을 남길 것인지 여부를 설정한다. 기본값은 OFF 이다.

JOB_QUEUE_SIZE

Row 동기화 작업은 작업 thread들에 의해 이루어지는데 동기화 작업 thread들은 JOB QUEUE에 들어온 작업들을 하나씩 가져가서 작업을 수행한다. JOB QUEUE에 작업을 넣는 thread는 TableDiff나 TableSync의 메인 thread인데 동기화 thread의 작업 속도가 더디면 JOB QUEUE가 가득차게 된다.

JOB_QUEUE_SIZE 속성은 JOB QUEUE의 크기를 결정하는데 이 값이 크면 queue가 가득차는 경우는 자주 발생하지 않겠지만 메모리를 많이 사용하게 된다. 기본값은 100인데 100 정도면 queue가 가득차지 않은 상태로 사용할 수 있다.

이 속성은 특별한 경우가 아니면 바꾸지 않는 것이 좋다.

JOB_THREAD

동기화 작업 thread 수를 지정한다. 정의하지 않을 경우, 기본값은 1이다. 동기화 할 row가 많을 경우 CPU의 개수를 고려하여 이 값을 적당한 수로 지정하는게 좋다. 당연히 이 값이 클수록 동기화 속도가 빨라진다.

JOB_UNIT_SIZE

동기화 작업은 이 속성이 명시한 크기만큼 batch job으로 수행한다. 기본값은 100이다.

DISPLAY_CALL_STACK

에러가 발생할 경우 call stack을 표시할 것인지 여부를 설정한다. 기본값은 OFF이다.

테이블 비교의 다중화 설정

일반적으로 TableDiff의 row 비교 작업은 단일 thread로 수행된다. 하지만 조건절을 명시하면 여러 thread로 비교 작업을 수행할 수 있다. 속성 이름을 PARTITION_RANGE[n]으로 하여 n 개의 속성에 대해 WHERE 조건절처럼 범위를 설정하면 n 개의 thread가 각각 비교 검사를 수행한다.

예를 들어 테이블에 월별 데이터가 들어 있는 경우에 12 개의 thread가 각각 비교 검사를 수행하도록 하려면 다음 속성들을 정의하면 된다.

```
PARTITION_RANGE1 = MONTH=1
PARTITION_RANGE2 = MONTH=2
PARTITION_RANGE3 = MONTH=3
PARTITION_RANGE4 = MONTH=4
PARTITION_RANGE5 = MONTH=5
```

```
PARTITION_RANGE6 = MONTH=6  
PARTITION_RANGE7 = MONTH=7  
PARTITION_RANGE8 = MONTH=8  
PARTITION_RANGE9 = MONTH=9  
PARTITION_RANGE10 = MONTH=10  
PARTITION_RANGE11 = MONTH=11  
PARTITION_RANGE12 = MONTH=12
```

모든 조건들은 서로 중복되지 않아야 하며 (disjoint해야 하며) union한 것은 전체 set와 동일해야 한다. 또한 조건에 사용되는 column은 반드시 primary key의 앞부분이어야 한다. (즉, 명시한 조건이 primary index를 사용할 수 있어야 한다.)

TableDiff에만 적용되는 속성이므로 TableSync는 이 속성을 무시한다.

37.

gsyncher

37.1 gsyncher 소개

정의

gsyncher는 GOLDILOCKS에서 제공하는 shared memory 상의 log와 disk의 logfile을 동기화하는 유틸리티이다. gsyncher는 서버가 운영되는 중에 비정상적으로 종료되면 shared memory에 남겨진 최신 log를 동기화하여 logfile에 기록한다.

특징

- 서버가 운영되는 동안에는 gsyncher를 실행할 수 없다.
- gsyncher가 운영될 수 있는 서버의 STARTUP 단계는 OPEN 이다.
- gsyncher를 실행하면 shared memory에 결합된 다른 응용 프로그램들을 모두 종료시킨다.
- gsyncher를 실행하는 중에 logfile이 스위치 될 수 있고, 이 경우에는 controlfile을 작성한다.
- gsyncher는 controlfile과 log가 반영되는 logfile을 사전에 backup한다.
- 훼손된 controlfile이 복원되어야 gsyncher를 실행할 수 있다.

사용법

```
gsyncher [options]
```

Option

-l --log	Log trace msg
-s --silent	Do not print message
-f --home	home directory
-c --copy-right	Do not print copy right
-b --backup-path	Backup directory
-h --help	Print help message

37.2 사용 예

- 예제 1: 서버가 정상적으로 운영되는 도중에 gsyncher를 실행하였다.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[SHARED MEM] Detached from shm.
ERR-HY000(53002): gmaster is active
```

- 예제 2: gsyncher가 실행될 수 있는 서버의 STARTUP 단계가 아니다.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[SHARED MEM] Detached from shm.
ERR-HY000(53000): invalid phase(MOUNT): executable phase is OPEN
```

- 예제 3: gsyncher를 실행하면 진행 과정이 출력된다. 실행 결과는 total buffer (0) bytes로써 모든 log가 logfile에 이미 flush 되어 동기화 할 log가 없다.

```
$ gsyncher --log
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[FLUSH] Log buffer flushed - Log group from id (0) to (0), lsn from (130784) to (130784),
total buffer (0) bytes
[SHARED MEM] Detached from shm.
[FINI] Log sync complete.
```

- log 옵션을 사용하여 gsyncher 동작을 로그 파일에 기록한다.

```

$ cat $GOLDILOCKS_DATA/trc/gsyncher.trc
=====
gsyncher start
TIME      : 2015-08-05 16:12:15.776633
=====
[2015-08-05 16:12:15.776763 THREAD(29199,139846497756928)]
[INIT] Log started.
[2015-08-05 16:12:15.777113 THREAD(29199,139846497756928)]
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[2015-08-05 16:12:15.777208 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[2015-08-05 16:12:15.777297 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[2015-08-05 16:12:15.777358 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[2015-08-05 16:12:15.777416 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.777917 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.777993 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778051 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778107 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778577 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778652 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778710 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778766 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778822 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.818007 THREAD(29199,139846497756928)]
[SHARED MEM] Detached from shm.
[2015-08-05 16:12:15.818100 THREAD(29199,139846497756928)]
[FINI] Log sync complete.

```

- 예제 4: Lsn 130786부터 Lsn 131012까지의 log가 log group 1에 flush 되었다.


```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gsq1' is cleared.
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[FLUSH] Log buffer flushed - Log group from id (1) to (1), lsn from (130786) to (131012),
total buffer (84480) bytes
[SHARED MEM] Detached from shm.
[FINI] Log sync complete.
```

- 예제 5: gsyncher가 동작하는 도중에 logfile switch가 발생하면 controlfile에 반영된다.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gsql' is cleared.
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gsql' is cleared.
[CLEAR PROCESS] Process 'gsql' is cleared.
```

[FLUSH] Log buffer flushed - Log group from id (2) to (3), lsn from (156412) to (178129), total buffer (16644096) bytes

[CONTROLFILE] Saving controlfile caused by logfile switching.

[CONTROLFILE] '/media/solid/ssd_home/work/product/Gliese/home/wal/control_0.ctl' control file is saved.

[CONTROLFILE] '/media/solid/ssd_home/work/product/Gliese/home/wal/control_1.ctl' control file is saved.

[SHARED MEM] Detached from shm.

[FINI] Log sync complete.

38.

gmon

38.1 gmon 소개

정의

gmon은 GOLDILOCKS가 제공하는 데이터베이스(gmaster) 프로세스 감지 유틸리티이다.

기능

gmon은 데이터베이스 프로세스 (gmaster)의 동작 여부를 모니터링하다가 해당 프로세스가 관리자의 SHUTDOWN에 의해 종료되지 않고 비정상적으로 종료되면 **gsyncher**를 실행하고 종료한다. 이는 gsyncher를 실행하여 log 손실을 최대한 줄이기 위함이다.

gmon은 서버 프로퍼티 GMON_AUTOSTART를 1로 설정하여 자동 실행되도록 하거나 사용자가 직접 실행할 수 있다.

노트

하나의 gmaster 프로세스는 하나의 gmon 프로세스만 감지할 수 있다.

사용법

```
gmon [options]
```

Option

-s --start	Start gmon
-t --stop	Stop gmon
-u --status	Get gmon status
-o --home	gmaster home path
-l --silent	Suppress display message
-r --no-copyright	Suppress display copy right and version
-h --help	Print help messages

38.2 사용 예

- 예제 1: 기본 경로 (\$GOLDILOCKS_DATA)를 사용하여 gmon을 실행한다.

```
$ gmon --start  
gmon is started.
```

- 예제 2: home 경로를 지정하여 gmon을 실행한다. 상대 경로를 사용할 경우 \$GOLDILOCKS_DATA를 베이스로 하여 home 경로가 지정된다.

```
$ gmon --start --home g1n1_home  
gmon is started.
```

- 예제 3: home 경로를 절대 경로로 지정하여 gmon을 실행한다. 절대 경로를 사용할 경우, 지정된 경로로 home 경로가 지정된다.

```
$ gmon --start --home /g1n1_home  
gmon is started.
```

- 예제 4: gmon process의 상태를 확인한다.

```
$ gmon --status --home /g1n1_home  
gmon is running(19119).
```

- 예제 5: gmon process를 종료한다.

```
$ gmon --stop --home /g1n1_home  
gmon is stopped.
```


39.

gtrclogger

39.1 gtrclogger 소개

정의

gtrclogger는 GOLDILOCKS의 log를 원격으로 수집하는 유틸리티이다.

GOLDILOCKS는 기본적으로 프로퍼티의 SYSTEM_LOGGER_DIR에 지정된 디렉토리 (기본값은 \$GOLDILOCKS_DATA/trc)의 system.trc 파일에 서버에서 발생하는 이벤트와 trace log를 기록한다.

만약 다수의 서로 다른 instance를 운영하고 있는 환경이라면 각각의 서버에 발생하는 trace log를 별도로 모니터링 해야 하는데 이에 대한 편의를 제공하고자 서로 다른 GOLDILOCKS instance의 trace log를 한 곳에 집중하여 수집 할 수 있게 해 준다. gtrclogger는 다른 장비에서 실행 중인 여러 GOLDILOCKS의 trace log를 udp로 받아 파일에 저장한다.

gtrclogger는 원격 GOLDILOCKS로부터 trace log를 받아서 파일에 기록하는데, 이를 위해 원격 GOLDILOCKS는 다음과 같은 전송 프로퍼티를 가져야 한다.

- TRACE_LOGGER
- TRACE_LOGGER_REMOTE_HOST
- TRACE_LOGGER_REMOTE_PORT

특징

- gtrclogger는 서버 운영과는 관계없이 실행할 수 있다.
- 여러 서버로부터 trace log를 받는 경우에도 받는 순서대로 하나의 파일에 저장한다.
- Trace log는 기본적으로 \$(GOLDILOCKS_DATA)/trc/system.rmt.trc로 저장된다.
- system.rmt.trc가 10 Mbyte 보다 클 경우, system.rmt.trc_날짜_Index로 백업된다.

사용법

| gtrclogger [options]

Options

-s --start	start gtrclogger
-q --stop	stop gtrclogger
-p --port	udp port for receive log (1024 ~ 49151): default 21470
-d --dir	write file directory: default \$(GOLDILOCKS_DATA)/system.rmt.trc
-h --help	show gtrclogger help messages

39.2 사용 예

- 예제 1: 기본 port (21470)와 기본 디렉토리 (\$GOLDILOCKS_DATA/trc)를 사용하여 gtrclogger를 실행한다.

```
$ gtrclogger --start  
gtrclogger is started successfully.
```

- 예제 2: port 21471, 21472 두 개로 trace log를 받고 디렉토리는 \$GOLDILOCKS_DATA/rmt_trc로 설정하여 gtrclogger를 실행한다.

```
$ gtrclogger -s -p 21471 -p 21472 -d $GOLDILOCKS_HOME/rmt_trc  
ERR-HY000(11040): No such object  
(/home/lym1/workspace/product/Gliese/home/rmt_trc/system.rmt.trc)  
$ mkdir $GOLDILOCKS_HOME/rmt_trc  
$ gtrclogger -s -p 21471 -p 21472 -d $GOLDILOCKS_HOME/rmt_trc  
gtrclogger is started successfully.
```

- 예제 3: gtrclogger를 종료한다.

```
$ gtrclogger -q  
Stop Done.
```

40.

glocator

40.1 glocator 소개

정의

glocator는 GOLDILOCKS cluster system에서 client에게 location을 제공하고 관리하는 유틸리티이다. glocator 프로그램은 **gloctl**과 **gagent**를 통해 cluster member 노드들의 location 정보를 제공받는다. glocator는 User Datagram Protocol (UDP)을 사용하여 client 및 내부 프로세스와 통신한다.

노트

glocator가 사용하는 location 정보는 cluster member 노드가 사용하는 location과 차이가 있다. glocator는 member 노드의 host, listener port, db home path, agent port를 필요로 한다. glocator는 gagent 보다 먼저 실행되어 gagent로부터 이런 location 정보를 제공받거나 gloctl을 통해 제공받아야 한다.

사용법

```
glocator [options]
```

Option

help

설명

Help 메시지를 출력한다.

사용 예

```
$ glocator --help
Usage:
  glocator [options]
Options:
  -c  --create          Create glocator environment
```

```
-s --start      Start glocator
-t --stop      Stop glocator
-f --conf      Set configure file
-u --status    Get glocator status
-y --sync      Synchronize with alternate glocator(BOTH, SOURCE)
-l --silent    Suppress display message
-r --no-copyright Suppress display copy right and version
-h --help      Print help message
```

create

설명

glocator의 데이터 파일을 생성한다.

사용 예

```
$ glocator --create
glocator is created.
```

노트

glocator 데이터 파일은 <GOLDILOCKS_DATA>/db 디렉토리에 생성된다.

```
$ ls
README glocator.dat system_data.dbf system_dict.dbf system_undo.dbf
```

start

설명

glocator를 구동한다. 같은 port를 갖는 glocator가 이미 구동 중이라면 에러가 발생한다.

사용 예

```
$ glocator --start
glocator is started.
```

stop

설명

구동 중인 glocator를 정지시킨다. 구동 중인 glocator를 멈추려면 같은 port가 설정되어야 한다.

사용 예

```
$ glocator --stop  
glocator is stopped.
```

conf

설명

glocator를 구동할 때 configure file을 설정한다.

사용 예

```
$ glocator --start --conf goldilocks.glocator.conf  
glocator is started.
```

status

설명

구동 중인 glocator의 상태 메시지를 출력한다. 구동 중인 glocator 상태를 확인하려면 같은 port가 설정 되어야 한다.

사용 예

```
$ glocator --status  
Process ID: 26058  
Configuration file: goldilocks.glocator.conf  
Unix domain path: /tmp/unix-glocator.42581  
Udp listen host: 0.0.0.0, Port: 42581  
glocator is running.
```

sync

설명

glocator를 구동하기 전에 ALTERNATE_LOCATORS에 설정된 glocator의 데이터를 동기화를 한다.

데이터 동기화에는 ALTERNATE_LOCATORS에서 데이터를 가지고 오는 SOURCE 방식과 두 glocator를 병합하는 BOTH 방식이 있다.

사용 예

다음은 sync 옵션에 성공하는 예이다.

```
$ glocator --start --sync
glocator is started.
```

다음은 sync 옵션에 실패하는 예이다. ALTERNATE_LOCATORS 속성에 값이 설정되지 않았다.

```
$ glocator --start --sync
ERR-HY000(60016): Need more alternate locator host information.
```

다음은 sync 옵션에 실패하는 예이다. ALTERNATE_LOCATORS에 설정된 glocator로부터 응답을 받지 못한 상황이다.

```
$ glocator --start --sync
ERR-HY000(60016): Need more alternate locator host information.
```

silent

설명

실행에 대한 glocator의 메시지를 출력하지 않는다.

사용 예

```
$ glocator --start --silent
```

no-copyright

설명

실행에 대한 glocator의 copyright와 버전 메시지를 출력하지 않는다.

사용 예

```
$ glocator --start --no-copyright  
glocator is started.
```


40.2 glocator 사용

데이터 파일

glocator를 구동하려면 시작하기 전에 glocator 데이터 파일을 먼저 생성해야 한다.

```
$ glocator --create
glocator is created.
```

glocator 데이터 파일이 저장되는 디렉토리는 configuration `LOCATION_FILE_DIR`을 수정하여 변경할 수 있다. 기본값은 `<GOLDILOCKS_DATA>/db`에 생성된다. 데이터 파일은 `LOCATION_FILE_NAME`을 수정하여 변경할 수 있다. 기본값은 `glocator.dat`이다.

glocator 데이터 파일의 최대 크기와 초기 크기는 configuration `LOCATION_FILE_MAX_SIZE`와 `LOCATION_FILE_SIZE`를 수정하여 변경할 수 있다.

CSTARTUP과 CSHUTDOWN

glocator는 GOLDILOCKS 서버의 CSTARTUP과 CSHUTDOWN에 사용될 수 있다.

이를 위해서는 glocator가 구동 중이고 gsqlnet을 통해 CSTARTUP 또는 CSHUTDOWN 명령을 실행해야 한다.

단, gsqlnet을 실행하는 장비의 `odbc.ini`에 `LOCATOR_DSN`과 속성값이 설정되어 있어야 한다.

자세한 내용은 표 25-1 GOLDILOCKS UNIX ODBC driver 라이브러리를 참조한다.

다음은 glocator를 사용하기 위해 `odbc.ini`에 설정한 내용이다.

```
[GOLDILOCKS]
HOST=127.0.0.1
PORT=20101
UID=sys
PWD=gliese
LOCATOR_DSN=GLOCATOR

[GLOCATOR]
HOST=127.0.0.1
PORT=42581
```

노트

DSN [GLOCATOR]에 FILE 속성이 있으면 glocator 대신 FILE 속성에 명시된 파일을 우선해서 사용하므로 FILE 속성을 제외해야 한다.

CSTARTUP과 CSHUTDOWN 명령을 실행하려면 glocator가 member 노드들에 대한 location 정보를 알고 있어야 한다.

다중화

개요

glocator 다중화는 데이터를 일관성있게 유지하여 장애가 발생할 경우 원격 glocator를 사용한 지속적인 서비스를 가능하게 한다.

설정

다중화를 사용하려면 각각의 glocator가 configure file에 ALTERNATE_LOCATORS 속성을 설정해야 한다. ALTERNATE_LOCATORS 속성에는 Locator_name이 설정되어야 하고 설정된 Locator_name은 HOST, PORT 속성과 함께 configure 파일에 기록되어야 한다.

ALTERNATE_LOCATORS 속성은 다음과 같이 설정할 수 있다.

```
[LOCATOR]
ALTERNATE_LOCATORS = (locator_name1,locator_name2)
[locator_name1]
HOST= ip_address
PORT = port_num
[locator_name2]
HOST= ip_address
PORT = port_num
```

다음은 ALTERNATE_LOCATORS 속성을 설정하는 configure 파일의 예이다.

```
[LOCATOR]
PORT=42581
SYNC_RESPONSE_TIMEOUT = 2
SYNC_RETRY_COUNT = 2
LOCATION_FILE_NAME='glocator_1.dat'
```

```

ALTERNATE_LOCATORS=(LOCATOR_2,LOCATOR_3)
[LOCATOR_2]
HOST=127.0.0.1
PORT=42582
[LOCATOR_3]
HOST=127.0.0.1
PORT=42583

```

노트

- [LOCATOR]는 glocator가 기본값으로 읽어들이는 DSN이다.
- glocator와 관련된 ODBC와 gagent 설정에 대한 자세한 내용은 **odbc.ini 파일**, **ALTERNATE_LOCATORS**을 참조한다.

데이터 동기화

다중화된 glocator를 서비스할 때 데이터는 일관되게 유지되어야 한다. 기존에 작동 중인 glocator가 없을 경우, 모든 glocator는 일반 시작을 하면 된다. 기존에 동작 중인 glocator가 있을 경우, **sync** 옵션을 사용하여 데이터를 동기화하는 방법으로 alternate glocator를 시작할 수 있다.

sync 옵션에는 glocator 자신의 데이터와 alternate glocator의 데이터를 병합하는 BOTH와 상대 glocator로부터 데이터를 가지고 오는 SOURCE가 있다. glocator의 sync 작업은 glocator와 alternate glocator 사이의 1 : 1 대응 작업이다. 따라서 A, B, C 세 개의 glocator를 다중화하려고 할 때 A와 B가 이미 구동 중이고 C를 BOTH 방식으로 sync하여 구동하면 A 또는 B glocator의 데이터가 서로 다를 수 있다.

서비스 중인 glocator는 변경된 내용만 동기화한다. 패킷 손실 등의 이유로 데이터 동기화에 실패하는 경우, 데이터가 서로 다를 수 있다. 이 경우, **gloctl** 프로그램을 사용하여 해당 glocator에서 직접 데이터를 수정하거나 glocator를 sync 옵션으로 다시 시작할 수 있다.

40.3 기능

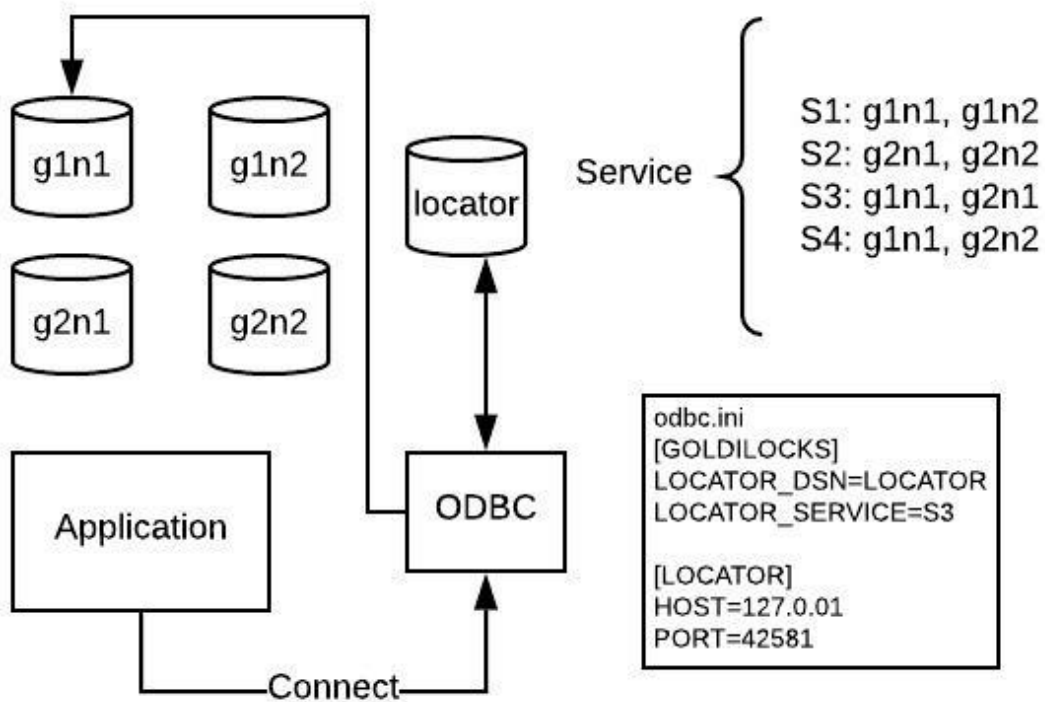
Connection Service

Cluster 환경에서 특정 노드에 대한 location 정보를 알지 못해도 사용자가 정한 몇 개의 노드 중에 임의로 접속할 수 있도록 service 기능을 제공한다.

Service 기능은 glocator에서 관리하는 노드를 사용자가 임의의 그룹으로 지정하는 것이다. 이 service는 gloctl 프로그램을 사용하여 등록할 수 있다. glocator에서 관리하는 서비스 목록 역시 gloctl 프로그램을 사용하여 확인할 수 있다.

응용 프로그램은 ODBC 드라이버를 사용하여 접속해야 하며 **odbc.ini 파일**에 LOCATOR_DSN과 LOCATOR_SERVICE 속성을 지정해야 한다.

그림 1 locator_service



위 그림은 응용 프로그램 (application)이 service s3에 속한 g1n1에 접속한 내용이다. ODBC 드라이버가 service를 이용하여 연결하는 순서는 service에 등록된 노드 순서와 동일하다. 위의 예에서 ODBC 드라이버가 g1n1에 연결하는데 실패하면 다음 순서인 노드 g2n1에 연결을 시도할 것이다.

노트

Service 기능을 사용하려면 glocator에 노드의 유효한 location 정보가 먼저 입력되어야 한다.

Cluster Failover

glocator는 server가 failover를 진행할 때 도움이 된다.

Server 프로퍼티 **CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY** 값이 1 또는 2로 설정되고, cluster 환경에서 노드 간에 연결이 끊어져 failover가 발생하면 각 노드는 cluster failover를 진행하여 gagent를 통해 glocator에게 자신의 viability를 질의한다.

질의를 받은 glocator는 연결이 끊어진 두 노드의 viability를 판단하고 gagent를 통해 각 노드에 결과를 전달한다.

Viability 결과를 받은 노드는 종료되거나 failover를 진행한다.

40.4 glocator Configuration

Configuration File 및 환경 변수

glocator는 configuration을 설정하기 위해 file이나 환경 변수를 사용할 수 있다.

환경 변수는 configuration property name에 prefix로 'LOCATOR_'를 추가한 name을 사용하여 지정할 수 있다. 예를 들어 configuration file에 HOST를 설정하면 \$LOCATOR_HOST를 지정하는 것과 같은 효과가 있다.

Configuration file 내용은 환경 변수 설정값보다 우선한다.

glocator는 DSN을 [LOCATOR] 기본값으로 하여 configuration file을 읽는다.

glocator는 \$GOLDILOCKS_DATA/conf/goldilocks.glocator.conf 파일을 자신의 환경 파일로 가지고 있다. glocator의 구동 환경을 변경하려면 해당 파일의 내용을 변경하거나 환경 변수를 설정하고 glocator를 구동하면 된다.

Configuration Property

HOST

항목	설명
이름	HOST
설명	glocator가 bind 하는 IP address이다.
Data type	ip address (ip v4)
기본값/ 범위	0.0.0.0

glocator가 UDP 통신을 위해 bind하는 IP address이다.

IP v4 형식의 IP address를 사용한다.

PORT

항목	설명
이름	PORT
설명	glocator가 Recv를 기다리는 port이다.
Data type	INT
기본값/ 범위	42581 / 1024~49151

glocator가 UDP 통신으로 packet을 받는 port이다.
Port는 1024부터 49151까지 사용할 수 있다.

WORKER_COUNT

항목	설명
이름	WORKER_COUNT
설명	Job을 처리하는 thread의 개수이다.
Data type	INT
기본값/ 범위	1 / 1~8

glocator가 client나 내부 프로세스로부터 받은 packet을 처리하는 thread의 개수이다.

SESSION_QUEUE_SIZE

항목	설명
이름	SESSION_QUEUE_SIZE
설명	수신한 packet을 처리하기 전까지 저장하는 queue size이다.
Data type	INT
기본값/ 범위	33554432 / 10485760~2147483648

glocator가 client나 내부 프로세스로부터 받은 packet을 처리하기 전에 저장하는 queue의 size이다.
Queue에는 packet이 session 단위의 단일 item으로 저장된다.

SESSION_ALLOCATOR_SIZE

항목	설명
이름	SESSION_ALLOCATOR_SIZE
설명	Session queue에 저장되는 item을 할당하는 allocator size이다.
Data type	INT
기본값/ 범위	33554432 / 10485760~2147483648

Session queue에 저장되는 item (session)을 할당할 때 사용되는 allocator의 size이다.

PACKET_ALLOCATOR_SIZE

항목	설명
이름	PACKET_ALLOCATOR_SIZE
설명	UDP 통신으로 packet을 받을 때 packet을 할당하는 allocator size이다.

항목	설명
Data type	INT
기본값/ 범위	33554432 / 10485760~2147483648

glocator가 packet을 받기 위해 할당하는 buffer를 위한 allocator의 size이다.

SYSTEM_LOGGER_DIR

항목	설명
이름	SYSTEM_LOGGER_DIR
설명	glocator trace log file의 directory path이다.
Data type	String
기본값/ 범위	'<GOLDILOCKS_DATA>/trc'

glocator의 system trace log file이 저장되는 directory path이다. 기본값의 <GOLDILOCKS_DATA>는 \$GOLDILOCKS_DATA 환경 변수 값으로 대체된다.

SYSTEM_UDS_DIR

항목	설명
이름	SYSTEM_UDS_DIR
설명	glocator에서 사용되는 unix domain socket 파일이 저장되는 directory path이다.
Data type	String
기본값/ 범위	'/tmp' / 최대 60 byte

glocator에서 사용되는 unix domain socket 파일이 저장되는 directory path를 설정한다. Directory 최대 길이는 60 byte 이내로 설정해야 한다.

LOCATION_FILE_DIR

항목	설명
이름	LOCATION_FILE_DIR
설명	glocator에서 사용되는 location 파일이 저장되는 directory path이다.
Data type	String
기본값/ 범위	'<GOLDILOCKS_DATA>/db'

glocator에서 사용되는 location 파일이 저장되는 directory path를 설정한다.

LOCATION_FILE_NAME

항목	설명
이름	LOCATION_FILE_NAME
설명	glocator에서 사용되는 location 파일 이름이다.
Data type	String
기본값/ 범위	'glocator.dat'

glocator에서 사용되는 location 파일 이름을 설정한다.

기본값은 glocator.dat 이다.

LOCATION_FILE_SIZE

항목	설명
이름	LOCATION_FILE_SIZE
설명	Location 파일의 초기 size이다.
Data type	Int
기본값/ 범위	1048576 / 104576~2147483648

glocator에서 사용되는 location 파일의 초기 size를 설정한다.

LOCATION_FILE_MAX_SIZE

항목	설명
이름	LOCATION_FILE_MAX_SIZE
설명	Location 파일의 최대 size이다.
Data type	Int
기본값/ 범위	10485760 / 104576~2147483648

glocator에서 사용되는 location 파일의 최대 size를 설정한다.

SESSION_TIMEOUT

항목	설명
이름	SESSION_TIMEOUT
설명	glocator가 packet을 수신하기 위해 최대로 대기하는 시간을 설정한다.
Data type	Int
기본값/ 범위	100 / 0 ~2147483648 (second 단위)

glocator가 packet을 수신하기 위해 최대로 대기하는 시간 (초)을 설정한다. 처리되지 못하고 시간 초과된 packet은 버려진다.

FAILOVER_TIMEOUT

항목	설명
이름	FAILOVER_TIMEOUT
설명	glocator가 failover 요청을 받고 작업할 때의 최대 대기 시간을 설정한다.
Data type	Int
기본값/ 범위	18 / 0 ~ 2147483648

glocator가 요청받은 failover 작업을 하는 중에 packet을 기다리는 최대 시간을 설정한다. 시간을 초과하면 요청한 gagent에 에러를 반환한다.

ALTERNATE_LOCATORS

항목	설명
이름	ALTERNATE_LOCATORS
설명	glocator의 alternate locator, 다중화 설정을 한다.
Data type	String
기본값/ 범위	empty / 0 ~ 1024 bytes

glocator의 **다중화**를 설정한다.

SYNC_RETRY_COUNT

항목	설명
이름	SYNC_RETRY_COUNT
설명	glocator가 alternate locator와의 동기화 작업에 실패할 경우 다시 시도하는 횟수를 설정한다.
Data type	Int
기본값/ 범위	1 / 0 ~ 5

glocator 동기화 작업의 재전달 횟수를 설정한다.

데이터가 변경되었을 때 glocator는 alternate locator에 변경된 데이터를 전달하는데, 데이터를 전달한 후에 응답을 받지 못하면 다시 전달한다.

SYNC_RESPONSE_TIMEOUT

항목	설명
이름	SYNC_RESPONSE_TIMEOUT
설명	glocator 동기화 작업에 대한 응답 대기 시간을 설정한다.
Data type	Int
기본값/ 범위	5 / 1 ~ 20

glocator가 전달한 동기화 데이터에 대한 응답을 기다리는 시간을 설정한다.

41.

gagent

41.1 gagent 소개

정의

gagent는 GOLDDLOCKS cluster system에서 각 멤버 노드와 함께 실행되어 **glocator**와 communication 하는 유틸리티이다.

노트

gagent는 glocator에게 local 노드의 location 정보를 제공할 수 있다. Location 정보의 정합성을 위해 glocator가 gagent 보다 먼저 실행되어야 한다.

사용법

```
gagent [options]
```

Option

help

설명

Help 메시지를 출력한다.

사용 예

```
$ gagent --help
Usage:
  gagent [options]
Options:
  -s --start          Start gagent
  -t --stop           Stop gagent
  -u --status         Get gagent status
  -f --conf           Set configure file
```

-o --home	gmaster home path
-l --silent	Suppress display message
-r --no-copyright	Suppress display copy right and version
-h --help	Print help message

start

설명

gagent를 구동한다. 같은 home 디렉토리를 갖는 gagent가 이미 구동 중이라면 에러가 발생한다.

사용 예

```
$ gagent --start
gagent is started.
```

stop

설명

구동 중인 gagent를 멈추게 한다. 구동 중인 gagent를 멈추려면 동일한 home 디렉토리가 설정되어야 한다.

사용 예

```
$ gagent --stop
gagent is stopped.
```

status

설명

구동 중인 gagent의 상태 메시지를 출력한다. 구동 중인 gagent 상태를 확인하려면 동일한 home directory가 설정되어야 한다.

사용 예

```
$ gagent --status
gagent(31938) is running.
```

conf

설명

gagent를 구동할 때 configure file을 설정한다.

사용 예

```
$ gagent --start --conf goldilocks.gagent.conf  
gagent is started.
```

home

설명

gagent를 구동할 때 GOLDDILOCKS의 서버 홈 디렉토리를 설정한다.

상대 경로를 사용할 경우 <GOLDDILOCKS_DATA>를 기준으로 홈 디렉토리를 찾는다. <GOLDDILOCKS_DATA>는 \$ GOLDDILOCKS_DATA 환경 변수의 값으로 대체된다.

사용 예

```
$ gagent --start --home g1n1_home --conf goldilocks.gagent.conf  
gagent is started.
```

위의 예에서 gagent는 <GOLDDILOCKS_DATA>/g1n1_home을 홈으로 설정하여 구동된다.

silent

설명

실행에 대한 gagent의 메시지를 출력하지 않는다.

사용 예

```
$ gagent --start --silent  
Copyright (C) 2010 SUNJESOFT Inc. All rights reserved.
```

no-copyright

설명

실행에 대한 gagent의 copyright와 버전 메시지를 출력하지 않는다.

사용 예

```
$ gagent --start --no-copyright  
gagent is started.
```

41.2 gagent Configuration

Configuration File 및 환경 변수

gagent는 configuration을 설정하기 위해 file 또는 환경 변수를 사용할 수 있다.

환경 변수는 configuration property name에 prefix로 'AGENT_'를 추가한 name을 사용하여 지정할 수 있다. 예를 들어 configuration file에 HOST를 설정하면 \$AGENT_HOST를 지정하는 것과 같은 효과가 있다.

Configuration file의 내용은 환경 변수 설정값보다 우선한다.

gagent는 \$GOLDILOCKS_DATA/conf/goldilocks.gagent.conf 파일을 자신의 환경 파일로 가지고 있다. gagent의 구동 환경을 변경하려면 해당 파일의 내용을 변경하거나 환경 변수를 설정하고 gagent를 구동하면 된다.

Configuration Property

HOST

항목	설명
이름	HOST
설명	gagent가 bind 하는 IP address이다.
Data type	ip address (ip v4)
기본값/범위	0.0.0.0

gagent가 UDP 통신을 위해 bind하는 IP address이다.

IP v4 형식의 IP address를 사용한다.

PORT

항목	설명
이름	PORT
설명	gagent가 Recv를 기다리는 port이다.
Data type	INT
기본값/범위	43581 / 1024~49151

gagent가 UDP 통신으로 packet을 받는 port이다.

Port는 1024부터 49151까지 사용할 수 있다.

LOCATOR_HOST

항목	설명
이름	LOCATOR_HOST
설명	glocator의 IP address이다.
Data type	ip address(ip v4)
기본값/ 범위	0.0.0.0

glocator의 IP address이다.

LOCATOR_PORT

항목	설명
이름	LOCATOR_PORT
설명	glocator가 Recv를 기다리는 port이다.
Data type	INT
기본값/ 범위	42581 / 1024~49151

gagent가 glocator에게 packet을 보낼 때 glocator의 port이다.

Port는 1024부터 49151까지 사용할 수 있다.

COMMAND_QUEUE_SIZE

항목	설명
이름	COMMAND_QUEUE_SIZE
설명	수신한 packet을 처리하기 전까지 저장하는 queue size이다.
Data type	INT
기본값/ 범위	1048576 / 1048576~2147483648

gagent가 glocator로부터 받은 packet을 처리하기 전에 저장하는 queue의 size이다.

Queue에는 packet이 command 단위의 단일한 item으로 저장된다.

COMMAND_ALLOCATOR_SIZE

항목	설명
이름	COMMAND_ALLOCATOR+SIZE
설명	Command queue에 저장되는 item을 할당하는 allocator size이다.
Data type	INT
기본값/ 범위	1048576 / 1048576~2147483648

Command queue에 저장되는 item (command)을 할당할 때 사용되는 allocator의 size이다.

PACKET_ALLOCATOR_SIZE

항목	설명
이름	PACKET_ALLOCATOR+SIZE
설명	UDP 통신으로 packet을 받을 때 packet을 할당하는 allocator size이다.
Data type	INT
기본값/ 범위	1048576 / 1048576~2147483648

gagent가 packet을 받기 위해 할당하는 buffer를 위한 allocator의 size이다.

SYSTEM_LOGGER_DIR

항목	설명
이름	SYSTEM_LOGGER_DIR
설명	gagent의 trace log file의 directory path이다.
Data type	String
기본값/ 범위	'<GOLDILOCKS_DATA>/trc'

gagent의 system trace log file이 저장되는 directory path이다. 기본값의 <GOLDILOCKS_DATA>는 \$GOLDILOCKS_DATA 환경 변수 값으로 대체된다.

UPDATE_LOCATION_TIME

항목	설명
이름	UPDATE_LOCATION_TIME
설명	gagent가 glocator에 자신의 location 정보를 업데이트 하는 주기를 설정한다.
Data type	Int
기본값/ 범위	120 / 30~2147483648 (second 단위)

gagent에서 glocator에게 자신의 노드 location 정보를 업데이트 하는 주기를 설정한다.

SESSION_TIMEOUT

항목	설명
이름	SESSION_TIMEOUT
설명	gagent가 packet을 수신하기 위해 최대로 대기하는 시간을 설정한다.
Data type	Int

항목	설명
기본값/ 범위	60 / 0 ~ 2147483648 (second 단위)

glocator가 packet을 수신하기 위해 최대로 대기하는 시간 (초)을 설정한다. 처리되지 못하고 시간 초과된 packet은 버려진다.

ALTERNATE_LOCATORS

항목	설명
이름	ALTERNATE_LOCATORS
설명	alternate locators를 설정한다.
Data type	String
기본값/ 범위	empty / 0 ~ 1024

gagent에 설정된 glocator로부터 응답을 받지 못하는 경우, alternate locator로 대체하여 작업을 진행한다.

다음은 ALTERNATE_LOCATOR를 사용한 configure 파일의 예이다.

```
[AGENT]
PORT=43581
LOCATOR_HOST=127.0.0.1
LOCATOR_PORT=42581
ALTERNATE_LOCATORS='LOCATOR1,LOCATOR2'
[LOCATOR1]
HOST=127.0.0.1
PORT=42582
[LOCATOR2]
HOST=127.0.0.1
PORT=42583
```


42.

gloctl

42.1 gloctl 소개

정의

gloctl은 GOLDILOCKS에서 제공하는 대화형 유틸리티로써 **glocator**에 location을 제공하고 편집한다. gloctl은 User Datagram Protocol (UDP) 통신 프로토콜을 사용하여 glocator와 통신한다.

사용법

```
gloctl [options]
```

Option

help

설명

Help 메시지를 출력한다.

사용 예

```
$ gloctl --help
Usage:
  gloctl [options]
Options:
  -c --conf          User configure file
  -i --ip            Locator host ip
  -p --port          Locator port number
  -o --import        Import control FILE
  -l --silent        Suppress the display of result message and echoing commands
  -r --no-copyright  Suppress display copy right and version
  -h --help          Print help message
```


conf

설명

glocator와 통신하기 위해 **Configuration** 설정 파일을 지정한다.

지정하지 않을 경우, \$GOLDILOCKS_DATA/conf/goldilocks.gloctl.conf가 사용된다.

사용 예

```
$ gloctl --conf gloctl.conf
gLoctl>
```

- 위의 예에서 사용한 gloctl.conf 파일의 내용은 다음과 같다.

```
[GLOCTL]
# Port number (1024 ~ 49151)
PORT = 44581
# Locator address
LOCATOR_HOST = 127.0.0.1
# Locator port number (1024 ~ 49151)
LOCATOR_PORT = 42581
# Time out to receive message from glocator (second)
# second ( 0 ~ 2147483647 )
MESSAGE_TIMEOUT = 10
```

ip

설명

gloctl를 구동할 때 glocator의 IP address를 설정한다. DSN과 함께 사용될 때는 IP 설정값이 우선한다.

사용 예

```
$ gloctl --ip 127.0.0.1
gLoctl>
```

위의 예에서 glocator의 port는 odbc.ini에서 DSN 기본값 LOCATOR를 이용하여 얻는다.

port

설명

gloctl를 구동할 때 glocator의 port를 설정한다. DSN과 함께 사용될 때는 port 설정값이 우선한다.

사용 예

```
$ gloctl --port 42581
gLoctl>
```

위의 예에서 glocator의 IP는 odbc.ini에서 DSN 기본값 LOCATOR를 이용하여 얻는다.

import

설명

대화형 모드가 아닌 파일 내의 gloctl 명령을 batch로 수행한다.

사용 예

```
$ gloctl --import import.txt
HELP
QUIT
IMPORT      {'FILE'}          Upload FILE to locations
EXPORT      {'FILE'}          Download locations to FILE
ADD MEMBER   {DSN|{member_name {'host; port; db_home; agent_port;'}}} Add member location
DROP MEMBER  {member_name}     Drop member location
SET TIMEOUT  {second}          Set time for session timeout
Add member succeeded.
```

- 위의 예에서 사용한 import.txt의 내용은 다음과 같다.

```
HELP
ADD MEMBER G1N3 'HOST=127.0.0.1;PORT=24581;DB_HOME=g1n3_home;AGENT_PORT= 44581;'
```

silent

설명

실행에 대한 gloctl의 메시지를 출력하지 않는다.

사용 예

```
$ gloctl --silent --import import.txt
```

no-copyright

설명

실행에 대한 gloctl의 copyright과 버전 메시지를 출력하지 않는다.

사용 예

```
$ gloctl --no-copyright  
gLoctl>
```

42.2 Interactive Command Reference

ADD MEMBER

구문

```
ADD MEMBER {DSN | member_name {'host; port; db_home; agent_port;'}}
```

설명

glocator에 member 하나를 추가한다. odbc.ini에 존재하는 DSN 또는 member name과 필요한 location 정보를 입력해야 한다.

노트

- Port는 glsnr의 port이다.
- ADD MEMBER는 기존 member의 정보를 갱신할 때 사용할 수 있다. 이 때 해당 member가 failover 하는 중이라면 갱신에 실패한다.

사용 예

odbc.ini의 DSN을 이용하여 ADD MEMBER를 수행한다.

```
gLoctl> ADD MEMBER G1N1
Add member succeeded.
gLoctl>
```

- 다음은 odbc.ini의 내용이다.

```
[G1N1]
HOST=127.0.0.1
PORT=20101
DB_HOME= g1n1_home
AGENT_PORT=43581
LOCATOR_DSN=LOCATOR
```

- Member의 location을 서술하여 추가한다.

```
gLoc1> ADD MEMBER G1N2 'HOST=127.0.0.1; PORT=20201; DB_HOME=g1n2_home; AGENT_PORT= 43581'
Add member succeeded.
gLoc1>
```

ADD SERVICE

구문

```
ADD SERVICE service_name {'member_name; ... '}
```

설명

glocator에 service 하나를 추가한다. Service에 속한 node는 **glocator**에서 관리되어야 하며 glocator에 동일한 node가 없으면 무시된다.

사용 예

ADD SERVICE로 service s1을 추가한다.

```
gLoc1> ADD SERVICE S1 'G1N1;G1N2;G2N1'
Add service succeeded.
gLoc1>
```

- 다음은 odbc.ini의 내용이다.

```
[GOLDILOCKS]
HOST=127.0.0.1
PORT=22581
LOCATOR_DSN=LOCATOR
LOCATOR_SERVICE=S1
[LOCATOR]
HOST=127.0.0.1
PORT=42581
```

노트

odbc.ini 파일에 glocator 속성을 기술하고 **glocator**를 사용하면 gsqlnet과 기타 응용 프로그램은 service 목록 중 하나의 서버에 접속할 수 있다.

DROP MEMBER

구문

```
DROP MEMBER member_name
```

설명

member_name에 해당하는 member를 glocator에서 삭제한다.

노트

삭제하려는 member가 failover 과정 중에 있다면 삭제할 수 없다.

사용 예

```
gLoctl> DROP MEMBER G1N2
Drop member succeeded.
gLoctl>
```

DROP SERVICE

구문

```
DROP MEMBER service_name
```

설명

service_name에 해당하는 service를 glocator에서 삭제한다.

사용 예

```
gLoc1> DROP SERVICE S1
Drop service succeeded.
gLoc1>
```

EXPORT

구문

```
EXPORT {'FILE'}
```

설명

glocator로부터 location 정보를 받아 파일에 저장한다.

사용 예

Location.txt라는 파일에 location 정보를 다운로드 한다.

```
gLoc1> EXPORT 'Location.txt'
Export file succeeded.
gLoc1>
```

- 위 예의 Location.txt 파일 내용은 다음과 같다.

```
[G1N1]
PORT = 20101
AGENT_PORT = 0
HOST = 127.0.0.1
DB_HOME = g1n1_home
[G1N3]
PORT = 24581
AGENT_PORT = 44581
HOST = 127.0.0.1
```

```
DB_HOME = g1n3_home
```

HELP

구문

```
HELP
```

설명

gloctl의 대화형 모드에서 명령어 목록을 보여준다.

사용 예

```
gLoctl> HELP
HELP
QUIT
IMPORT {'FILE'} Upload FILE to locations
EXPORT {'FILE'} Download locations to FILE
ADD MEMBER {DSN!{member_name {'host; port; db_home; agent_port;'}}} Add member location
DROP MEMBER {member_name} Drop member location
SET TIMEOUT {second} Set time for session timeout
ADD SERVICE {service_name {'member_name; ... '}} Add service
DROP SERVICE {service_name} Drop service
gLoctl>
```

IMPORT

구문

```
IMPORT {'FILE'}
```

설명

ini 형식의 파일을 이용하여 glocator에 location을 전송한다.

사용 예

Location.txt 파일에 있는 내용을 glocator로 보낸다.

```
gLoct1> IMPORT 'Location.txt'
Import file succeeded.
gLoct1>
```

- 위 예의 Location.txt 파일 내용은 다음과 같다.

```
[G1N1]
PORT = 20101
AGENT_PORT = 0
HOST = 127.0.0.1
DB_HOME = g1n1_home
[G1N3]
PORT = 24581
AGENT_PORT = 44581
HOST = 127.0.0.1
DB_HOME = g1n3_home
```

QUIT

설명

gloct1을 종료한다.

구문

```
QUIT
```

SET TIMEOUT

설명

glocator의 TIMEOUT을 설정한다.

구문

```
SET TIMEOUT {second}
```

사용 예

```
gLoctl> set timeout 120
Set timeout.
gLoctl>
```

42.3 Location File

Location file은 gloctl에서 glocator로 노드의 location 정보를 업로드하거나 다운로드한다.

설명

Location file 형식은 데이터 원본 스펙 파일의 형식과 유사하다.

```
[node_name]
HOST = host_address
PORT = port_no
DB_HOME = db_home_path
AGENT_PORT = agent_port_no
[SERVICE]
service_name = node_name {, node_name}*

```

Location file의 location 키워드는 다음 표와 같다.

표 42-1 Location information

키워드	설명
node_name	노드 이름이다.
HOST	노드의 IP 주소이다.
PORT	노드에서 실행된 glsnr의 port 번호이다.
DB_HOME	노드의 홈 디렉토리를 설정한다.
AGENT_PORT	노드에서 실행된 gagent의 port 번호이다.

[SERVICE]는 서비스 힌트 목록을 나열하는 고정 키워드이다. glocator에 등록되어 있거나 등록할 서비스 힌트 목록은 SERVICE 키워드 아래에 나열된다.

HOST와 PORT 키워드는 반드시 입력해야 하고 누락할 경우 에러가 발생한다.

다음은 location file을 구성하는 예이다.

```
[G1N1]
HOST = 192.168.0.101
PORT = 20101
DB_HOME = g1n1_home
AGENT_PORT = 40101
[G1N2]
```

```
HOST = 192.168.0.102
PORT = 20102
DB_HOME = g1n2_home
AGENT_PORT = 40102
[G2N1]
HOST = 192.168.0.201
PORT = 20201
DB_HOME = g2n1_home
AGENT_PORT = 40201
[G1N2]
HOST = 192.168.0.202
PORT = 20202
DB_HOME = g2n1_home
AGENT_PORT = 40202
[SERVICE]
service_1 = G1N1,G2N1
service_2 = G1N1,G1N2
```

42.4 Configuration

gloctl의 환경 파일은 \$GOLDILOCKS_DATA/conf/goldilocks.gloctl.conf이다. gloctl의 구동 환경을 변경하려면 해당 파일의 내용을 변경하거나 **conf** 옵션으로 환경 파일을 지정하여 gloctl을 구동하면 된다.

gloctl이 실행 중인 상태에서 gloctl의 환경을 변경하여 적용하려면 gloctl을 멈춘 후에 구동 환경을 변경 하고 다시 시작해야 한다.

PORT

항목	설명
이름	PORT
설명	gloctl이 사용하는 port이다.
Data type	INT
기본값/ 범위	44581 / 1024 ~ 49151

gloctl이 사용하는 소켓의 port를 설정한다.

LOCATOR_HOST

항목	설명
이름	PORT
설명	gloctl과 통신하는 glocator의 host address이다.
Data type	STRING
기본값/ 범위	127.0.0.1

glocator의 host address를 설정한다.

LOCATOR_PORT

항목	설명
이름	PORT
설명	gloctl과 통신하는 glocator의 port이다.
Data type	INT

항목	설명
기본값/ 범위	42581 / 1024 ~ 49151

glocator의 port를 설정한다.

MESSAGE_TIMEOUT

항목	설명
이름	MESSAGE_TIMEOUT
설명	gloctl이 glocator로부터 응답을 기다리는 시간을 설정한다. (second)
Data type	INT
기본값/ 범위	10 / 0 ~ 2147483647

gloctl이 glocator로 패킷을 보낸 후에 응답을 기다리는 시간을 설정한다.

파트 VII.

Replication

43. 개요	3,489
43.1 GOLDDILOCKS Replication 소개	3,490
43.2 특징	3,491
CYCLONE	3,491
LOGMIRROR	3,491
44. CYCLONE	3,493
44.1 CYCLONE	3,494
개요	3,494
운영상 특징	3,494
운영 시 제약사항	3,494
이중화 운영 중 DDL 처리	3,496
다른 DBMS와 연동할 때 Datatype 호환성	3,497
기타	3,498
44.2 준비사항	3,499
GOLDDILOCKS 준비사항	3,499
사용자 등록 및 권한 설정	3,501
44.3 환경설정	3,504
환경설정 파일	3,504
환경설정 옵션	3,505
44.4 운영하기	3,522
GOLDDILOCKS 접속 정책	3,522
실행 옵션	3,524
44.5 CYCLONE 운영 예	3,527
운영 순서	3,527
원본 GOLDDILOCKS 환경 설정	3,528
원격 GOLDDILOCKS 환경 설정	3,528
CYCLONE MASTER 환경 설정	3,528
CYCLONE SLAVE 환경 설정	3,529
실행 및 운영	3,529
데이터 동기화	3,530
이중화 정보 초기화	3,531
Recovery	3,532
44.6 Cluster 환경에서 CYCLONE 운영	3,534
준비사항	3,534
운영	3,534
수행	3,535
SYNC	3,537
기타	3,538
44.7 모니터링 (CYMON)	3,539

환경 설정 파일	3,539
모니터링 내용	3,539
실행 및 모니터링	3,540
45. LOGMIRROR	3,545
45.1 LOGMIRROR	3,546
개요	3,546
운영상 특징	3,546
LOGMIRROR 운영시 GOLDILOCKS 성능저하 요인	3,546
45.2 준비사항	3,547
GOLDILOCKS 준비사항	3,547
45.3 환경설정	3,549
환경설정 파일	3,549
환경설정 옵션	3,549
45.4 운영	3,553
LOGMIRROR 운영	3,553
실행 옵션	3,554
45.5 CYCLONE과 연동 예	3,556
운영 순서	3,557

43.

개요

43.1 GOLDDILOCKS Replication 소개

데이터베이스 이중화 (replication)는 데이터를 효율적이고 일관되게 분배하여 원본 데이터베이스 (local database)에서 장애가 발생할 경우, 원격 데이터베이스 (remote database)를 사용하여 데이터를 복구하여 지속적으로 서비스를 유지할 수 있도록 한다. 또한 동일한 데이터를 가지는 여러 개의 데이터베이스를 구축하기 위해서도 이중화가 사용된다.

GOLDDILOCKS의 이중화는 Change Data Capture(CDC) 방식을 사용하는 톨인 CYCLONE을 지원한다.

노트

Change Data Capture (CDC)

데이터베이스 운영 중에 생성되는 redo log를 캡처하고 분석하여 이중화를 수행하는 방식이다.

데이터베이스에 갱신된 내용이 redo log file에 저장되고 난 이후에 CYCLONE이 처리할 수 있기 때문에 비동기 (asynchronous) 방식만 지원한다.

CDC 방식은 비동기 방식만 지원하기 때문에 원본 데이터베이스와 그를 이중화하는 원격 데이터베이스 사이에는 데이터 반영에 대한 interval이 발생할 수 밖에 없다. 만약 원격 데이터베이스로 이중화를 완료하지 않은 상태에서 원본 데이터베이스나 장비에 장애가 발생할 경우, 원격 데이터베이스는 interval을 반영하지 못한 상태로 남아있게 된다.

이러한 비동기 방식의 interval로 인해 데이터가 미반영되는 점을 보완하기 위해 LOGMIRROR를 사용할 수 있다.

노트

LOGMIRROR는 redo log file을 이중화하는 톨로써 원본 데이터베이스에서 생성되는 redo log를 손실없이 원격 장비에 전송하고 저장한다. 이를 사용하여 데이터의 손실을 방지할 수 있다.

43.2 특징

CYCLONE

CYCLONE의 특징은 다음과 같다.

- Change Data Capture (CDC) 방식을 사용하는 이중화 툴이다.
- 원본 데이터베이스의 redo log file 갱신 사항을 인식하고 분석하여 적용한다.
- Master와 slave로 구분되어 운영된다.
- Active-active, active-standby를 지원한다.
- Table 단위의 replication이 가능하다.
- 성능 향상을 위해 다양한 옵션을 설정할 수 있다.
- 독립적인 프로세스 (process)로 실행되기 때문에 장애가 발생해도 GOLDDLOCKS에 영향을 미치지 않는다.
- 운영을 구성할 때 H/W를 추가하거나 변경할 필요가 없다.
- 다양한 이중화 구성 (topology)이 가능하다.
- Standalone 환경에서 master와 slave 간의 운영은 1 : 1 또는 N : N을 지원한다.
- Cluster 환경에서 master와 slave간의 운영은 N : 1을 지원한다.

LOGMIRROR

LOGMIRROR의 특징은 다음과 같다.

- 원본 데이터베이스의 redo log를 손실없이 원격 장비에 전송하고 저장한다.
- CYCLONE과 연동하여 데이터 손실없이 이중화할 수 있다.
- 한 개의 데이터베이스에서 한 개의 LOGMIRROR만 운영할 수 있다.
- Master와 slave로 구분되어 운영된다.

44.

CYCLONE

44.1 CYCLONE

CYCLONE은 Change Data Capture (CDC) 방식을 사용하는 이중화 툴이다.

개요

데이터베이스는 복구작업을 위해 운영 중에 발생하는 데이터 갱신사항을 redo log file에 기록한다. CDC는 이런 redo log file에 기록되는 내용을 분석하여 이중화를 수행하는 방식이다.

CYCLONE은 master와 slave로 나뉘어져 구동된다. Master는 원본 데이터베이스의 redo log file의 갱신사항을 인식하고 분석하여 slave로 전송한다. Slave는 수신된 데이터를 분석하고 ODBC를 이용하여 이중화를 수행한다.

운영상 특징

- Master와 slave로 구분되어 수행되며 group 단위의 master/ slave로 동작한다.
- CYCLONE의 master와 slave는 TCP/ IP 통신을 사용한다.
- Group 단위로 실행/ 종료할 수 있다.
- 이중화는 table 단위로 이루어지며 group 내에는 한 개 이상의 table이 포함될 수 있다.
- 하나의 table은 여러 개의 group에 포함되어 운영될 수 있다.
- 원본 데이터베이스에는 반드시 redo log file이 있어야 한다. 즉, DATA_STORE_MODE는 TDS로 운영되어야 한다.
- 원본 데이터베이스에는 반드시 SUPPLEMENTAL LOGGING이 있어야 한다.
 - SUPPLEMENTAL LOGGING은 CYCLONE 이중화를 위해 redo log file에 추가적인 정보를 추가한다.
- 원본 데이터베이스는 ARCHIVE LOG 모드로 운영되어야 한다.
 - GOLDILOCKS는 redo log file을 recursive하게 재사용한다. 만약 이중화가 완료되지 않은 상황에서 redo log file이 재사용될 경우, 이중화는 종료되며 현재 시점부터 기존의 이중화 상태를 포기하고 재시작해야 한다. 따라서 ARCHIVE LOG 모드로 운영하여 해당 redo log file이 아카이빙되도록 해야 한다.
- GOLDILOCKS와 독립적인 process로 동작하므로 이중화 장애가 발생하더라도 GOLDILOCKS에 영향을 미치지 않는다.

운영 시 제약사항

- 이중화에 참여하는 table은 반드시 PRIMARY KEY를 가져야 한다.
- Commit 된 transaction만 이중화한다. 따라서 commit 전에는 slave에서 해당 내용을 확인할 수 없다.
- Primary key update는 지원하지 않는다.
 - primary key의 값이 갱신되는 경우, 해당 table은 give up 되며 더 이상 이중화 대상이 되지 않는다.

- 이중화에 참여하는 table은 Generated Always As Identity 속성을 갖는 column을 사용할 수 없다.
- 이중화가 진행 중인 table에 Data Definition Language (DDL)이 수행된 경우 give up 될 수 있다.
 - Table1의 DDL 구문에 따른 give up 발생 및 절차에 따른 허용 여부를 참조한다.
 - CYCLONE의 table DDL 처리 절차에 따르지 않은 경우, 허용 가능한 DDL이라 하더라도 give up 된다.
 - 다른 table의 이중화에는 영향을 미치지 않는다.
 - Truncate table 또한 마찬가지이다.
- Give up 된 table 중에 --reset TABLE_NAME으로 give up된 table만 reset 할 수 있다.
- 이중화 하는 table을 구성하는 column은 동일한 구조 (데이터 타입, 순서)를 가져야 한다.
- 이중화를 수행하는 데이터베이스는 동일한 character encoding을 가져야 한다.

표 44-1 DDL 구문에 따른 give up 발생 및 절차에 따른 허용 여부

DDL 분류	Give up 발생 여부	절차에 따른 DDL 허용 여부	DDL 구문
Table DDL	X	-	CREATE TABLE
	O	X	DROP TABLE
	O	X	TRUNCATE TABLE
	O	X	ALTER TABLE .. RENAME
	X	-	ALTER TABLE .. STORAGE
	X	-	ALTER TABLE .. ADD SUPPLEMENTAL LOG
	O	X	ALTER TABLE .. DROP SUPPLEMENTAL LOG
	X	-	ALTER TABLE .. READ { ONLY WRITE }
Column DDL	O	O	ALTER TABLE .. ADD COLUMN
	O	X	ALTER TABLE .. SET UNUSED COLUMN
	X	-	ALTER TABLE .. RENAME COLUMN
	X	-	ALTER TABLE .. ALTER COLUMN .. SET DEFAULT
	X	-	ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT
	O	X	ALTER TABLE .. ALTER COLUMN .. SET NOT NULL
	O	X	ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL
	O	X	ALTER TABLE .. ALTER COLUMN .. ALTER IDENTITY
	O	X	ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY
	O	O	ALTER TABLE .. ALTER COLUMN .. SET DATATYPE
Constraint DDL	O	X	ALTER TABLE .. ADD CONSTRAINT
	O	X	ALTER TABLE .. DROP CONSTRAINT
	O	X	ALTER TABLE .. ALTER CONSTRAINT
	X	-	ALTER TABLE .. RENAME CONSTRAINT
Index DDL	O	X	CREATE UNIQUE INDEX
	X	-	CREATE INDEX
	O	X	DROP INDEX unique_index
	X	-	DROP INDEX non_unique_index
	X	-	ALTER INDEX .. STORAGE
	X	-	ALTER INDEX .. RENAME

DDL 분류	Give up 발생 여부	절차에 따른 DDL 허용 여부	DDL 구문
Table의 상위 객체	O	X	DROP USER
	O	X	DROP SCHEMA
	O	X	DROP TABLESPACE

노트

사용자 실수를 방지하기 위해 서버 프로퍼티 **DISABLE_DDL_CDC_GIVEUP**를 사용하여 give up을 유발하는 DDL을 수행하지 못하도록 제어할 수 있다.

이중화 운영 중 DDL 처리

이중화가 수행되고 있는 table에 허용 가능한 DDL을 수행할 경우 반드시 다음의 절차에 따라야 한다.

- Master 측에서 DDL을 수행하기 전에 master와 slave에서 운영 중인 CYCLONE을 모두 종료한다.
 - 진행 중인 업무 process는 종료할 필요없다.
 - 예: cyclone --master --stop/ cyclone --slave --stop
- Master와 slave 양쪽에 DDL을 수행한다. (허용 가능한 DDL이어야만 한다.)
 - 자세한 내용은 DDL 구문에 따른 give up 발생 및 절차에 따른 허용 여부를 참조한다.
- Master와 slave의 CYCLONE을 다시 기동한다.
 - cyclone --master --start .../ cyclone --slave --start ...
 - reset 옵션은 필요하지 않다.
 - 다시 구동할 때 1단계에서 종료된 이후부터 recovery를 수행하고 DDL을 처리한다. (Trace log를 참조한다.)

표 44-2 DDL 적용 절차

구분	MASTER CYCLONE	MASTER DB	SLAVE CYCLONE	SLAVE DB
1	CYCLONE STOP	-	-	-
2	-	-	CYCLONE STOP	-
3	-	DDL 수행	-	-
4	-	-	-	DDL 수행
5	CYCLONE START	-	-	-
6	-	-	CYCLONE START	-

노트

DDL 적용 절차를 정상적으로 수행한 후에 CYCLONE master에서 DDL을 수행한 TABLE이 give up 되었다면 다음 두 가지를 확인해야 한다.

1. Master DB와 slave DB에 수행한 DDL이 모두 동일하고, 수행 후 table의 구조가 동일한지 확인한다. (DDL 수행 순서는 관계없다.)
2. 수행한 DDL이 허용 가능한 것인지 확인한다.

위와 같은 이유로 table이 give up되었을 경우, table 단위의 reset (예: cyclone --start --master --reset TABLE_NAME)을 사용하여 다시 구동하는 방법으로 give up된 table을 현재 시점부터 다시 이중화해야 한다.

다른 DBMS와 연동할 때 Datatype 호환성

- CYCLONE slave가 target database를 GOLDDILOCKS가 아닌 다른 DBMS로 연동할 때 source database인 GOLDDILOCKS 테이블의 column 데이터 타입이 다른 DBMS 테이블의 column 데이터 타입과 정상적으로 매핑되어야 데이터 손실이나 오류를 방지할 수 있다.
- 현재는 Oracle과 연동할 수 있다.

Oracle

표 44-3 Oracle 연동 datatype

GOLDDILOCKS	Oracle	비고
Boolean	X	해당 데이터 타입이 없다.
NATIVE_SMALLINT	NUMBER(5)	-
NATIVE_INTEGER	NUMBER(10)	-
NATIVE_BIGINT	NUMBER(19)	-
NATIVE_REAL	BINARY_FLOAT	-
NATIVE_DOUBLE	BINARY_DOUBLE	-
FLOAT	FLOAT	-
SMALLINT	NUMBER(5,0)	-
INTEGER	NUMBER(10,0)	-
BIGINT	NUMBER(19,0)	-
INT2	NUMBER(5,0)	-
INT4	NUMBER(10,0)	-

GOLDILOCKS	Oracle	비고
INT8	NUMBER(19,0)	-
REAL	FLOAT(24)	-
DOUBLE	FLOAT(53)	-
DOUBLE PRECISION	FLOAT(53)	-
FLOAT4	FLOAT(24)	-
FLOAT8	FLOAT(53)	-
DECIMAL	NUMERIC	-
NUMBER	NUMBER	-
NUMERIC	NUMERIC	-
CHAR	CHAR	-
VARCHAR	VARCHAR, VARCHAR2	-
BINARY	RAW	-
VARBINARY	RAW	-
DATE	DATE	-
TIME	X	해당 데이터 타입이 없다.
TIMESTAMP	TIMESTAMP	-
TIMESTAMP WITH TIMEZONE	X	ODBC driver를 지원하지 않는다.
INTERVAL	X	ODBC driver를 지원하지 않는다.
LONG VARCHAR	LONG VARCHAR	-
LONG VARBINARY	LONG RAW	-

- 위의 표에 설명된 것과 같이 다음 네 개 데이터 타입은 이중화할 수 없다.
 - BOOLEAN
 - TIME
 - TIMESTAMP WITH TIMEZONE
 - INTERVAL

기타

이중화 시점은 다음과 같다.

- 최초로 실행할 때 master와 slave가 실행되고 초기화 과정이 끝난 이후부터 이중화가 시작된다.
- 이중화가 수행되는 도중에 종료 후 다시 구동하더라도 기존 종료 시점 이후부터 계속해서 이중화를 수행한다. (Recovery 기능)
- 기존 이중화를 포기하고 현재 시점부터 재시작하려면 --reset 옵션을 사용해야 한다.

44.2 준비사항

원본 GOLDILOCKS에서는 GOLDILOCKS 준비사항과 사용자 등록 및 권한 설정을 모두 수행해야 한다.
원격 GOLDILOCKS에서는 사용자 등록 및 권한 설정만 수행하면 된다.

GOLDILOCKS 준비사항

CYCLONE을 이용한 이중화를 시작하기 전에 GOLDILOCKS에는 다음과 같은 사항이 설정되어 있어야 한다.

SUPPLEMENTAL LOGGING

SUPPLEMENTAL LOGGING은 CYCLONE의 이중화를 위해 redo log file에 부가 정보를 함께 저장한다. 이미 운영 중인 데이터베이스의 해당 설정을 변경하려면 데이터베이스를 다시 시작해야 하는데 특정 테이블에만 SUPPLEMENTAL LOGGING을 설정할 경우에는 데이터베이스를 다시 시작할 필요가 없다.

데이터베이스에 SUPPLEMENTAL LOGGING 설정

- GOLDILOCKS의 프로퍼티로 SUPPLEMENTAL LOGGING을 설정할 경우, 모든 테이블에 대해 SUPPLEMENTAL LOGGING이 기록된다.
- GOLDILOCKS를 다시 시작해야 한다.
- 프로퍼티 파일에 해당 내용을 추가 또는 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: SUPPLEMENTAL LOG_DATA_PRIMARY_KEY = YES

이중화에 참여하는 특정 table에 SUPPLEMENTAL LOGGING 설정

```
<add table supplemental log statement> ::=
  ALTER TABLE table_name
    ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
  ;
```

ARCHIVE LOG

GOLDILOCKS는 redo log file을 순환하며 재사용한다. 만약 CYCLONE이 처리 중인 redo log file을 GOLDILOCKS가 재사용하면 CYCLONE은 더 이상 진행되지 못하고 종료된다. 이런 이중화의 지속적인 운영을 보장하려면 반드시 GOLDILOCKS를 ARCHIVE LOG 모드로 운영해야 한다.

운영 중인 데이터베이스를 ARCHIVE LOG 모드로 변경

- GOLDDILOCKS를 다시 시작해야 한다.
- 데이터베이스를 종료한 후에 sysdba로 접속하여 mount 상태에서 ARCHIVE LOG 모드로 변경한다.

```
gSQL> \startup mount
Startup success
gSQL> alter database archivelog;
Database altered.
```

데이터베이스 생성 시 ARCHIVE LOG 모드 설정

- 데이터베이스를 생성하기 전에 프로퍼티 파일을 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: ARCHIVELOG_MODE = 1

노트

ARCHIVE LOG 파일이 저장되는 경로는 'ARCHIVELOG_DIR'로 확인하고 변경할 수 있다.

Cluster 환경에서의 설정

SLAVE가 운영되는 GOLDDILOCKS가 cluster로 동작할 경우에는 TRANSACTION_COMMIT_WRITE_MODE 가 '1' (Wait) 이거나 CLUSTER_ASYNC_COMMIT이 'NO'로 설정되어야 한다. 둘 중의 하나만 설정하면 된다.

이는 cluster의 transaction 실행 정책과 Cyclone의 distributor의 정책으로 인하여 데이터가 충돌 (conflict)하지 않도록 하기 위함이다.

TRANSACTION_COMMIT_WRITE_MODE 변경

- GOLDDILOCKS를 다시 시작해야 한다.
- 프로퍼티 파일에 해당 내용을 추가하거나 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: TRANSACTION_COMMIT_WRITE_MODE = 1

노트

TRANSACTION_COMMIT_WRITE_MODE의 값이 '0'일 경우 NO-WAIT이고, '1'일 경우 WAIT를 나타낸다.

CLUSTER_ASYNC_COMMIT 변경

- 프로퍼티 파일에 해당 내용을 추가하거나 변경한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: CLUSTER_ASYNC_COMMIT = NO
- 다음과 같이 SQL 문을 실행하여 운영 중인 GOLDDILOCKS의 프로퍼티를 변경할 수 있다.

```
gSQL> alter system set CLUSTER_ASYNC_COMMIT=NO;
System altered.
```

DATA_STORE_MODE

CYCLONE은 GOLDDILOCKS의 redo log file을 읽어서 이중화를 수행한다. 따라서 GOLDDILOCKS는 Transactional Data Store (TDS) 모드로 동작해야 한다.

DATA_STORE_MODE 변경

- 데이터베이스를 다시 시작해야 한다.
- 프로퍼티 파일에 해당 내용을 추가하거나 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: DATA_STORE_MODE = 2

노트

DATA_STORE_MODE의 값이 1일 경우 Concurrent Data Store (CDS)를, 2일 경우 Transactional Data Store (TDS)를 의미한다.

사용자 등록 및 권한 설정

CYCLONE은 운영 중에 필요한 정보를 검색하고 조작한다. 따라서 CYCLONE 운영을 위한 사용자가 있어야 하며 해당 사용자에게 특정한 권한을 설정해 주어야 한다.

데이터베이스 User 생성

CYCLONE을 운영하려면 특정 사용자를 추가해야 하는데, CYCLONE이 master, slave 모드로 동작하는 GOLDDILOCKS 모두에 해당 사용자가 추가되어야 한다.

```

<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE {tablespace_name|NULL} ]
    [ <schema clause> ]
    ;
<schema clause> ::=
    WITH SCHEMA [schema_name]
    | WITHOUT SCHEMA

```

다음은 이름이 cdc_user이고 password가 cdc_password인 사용자를 추가하는 예이다.

```
gSQL> CREATE USER cdc_user IDENTIFIED BY cdc_password;
```

데이터베이스 권한

사용자의 접속 권한을 설정

다음은 cdc_user에게 접속 권한을 설정하는 예이며 master/ slave 모드로 동작하는 GOLDDILOCKS 모두에 설정해야 한다.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO cdc_user;
```

TABLE 변경 권한 설정

다음은 cdc_user에게 테이블 변경 권한을 설정하는 예이며 slave 모드로 동작하는 GOLDDILOCKS에 설정한다.

```
gSQL> GRANT INSERT ANY TABLE, DELETE ANY TABLE, UPDATE ANY TABLE ON
DATABASE TO cdc_user;
```

테이블스페이스 권한

데이터 테이블스페이스와 임시 테이블스페이스에 권한을 설정해야 한다. 다음은 cdc_user에게 기본 테이블스페이스의 사용 권한을 설정하는 예이며 slave 모드로 동작하는 GOLDDILOCKS에 설정한다.

```
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO cdc_user;
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_temp_tbs TO cdc_user;
```


스키마 권한

CYCLONE에서 관리하는 meta를 생성하고 관리하기 위한 스키마 권한을 설정해야 한다. 다음은 cdc_user 스키마 권한을 설정하는 예이며 slave 모드로 동작하는 GOLDDILOCKS에 설정한다.

```
gSQL> GRANT CREATE TABLE, CREATE INDEX, CREATE SEQUENCE, CREATE VIEW,  
ADD CONSTRAINT ON SCHEMA cdc_user TO cdc_user;
```

44.3 환경설정

환경설정 파일

CYCLONE을 실행할 때 환경 설정 파일을 사용하여 운영에 필요한 정보와 옵션을 설정할 수 있다.

- --conf 옵션을 사용하여 특정 환경설정 파일을 설정하지 않을 경우, \$GOLDILOCKS_DATA/conf 디렉토리에서 특정 파일을 읽어들인다. Master로 동작할 경우에는 cyclone.master.conf 파일을 읽고, slave로 동작할 경우에는 cyclone.slave.conf 파일을 읽는다.

표 44-4 설정 내용

이름	설명	적용범위
COMM_CHUNK_COUNT	통신에 사용될 BUFFER의 크기를 설정한다.	Master/ slave
DSN	Data Source Name을 설정한다.	Master/ slave
GROUP_NAME	그룹 이름을 설정한다.	Master/ slave
HOST_IP	GOLDILOCKS가 운영 중인 host IP address를 설정한다.	Master/ slave
HOST_EXTERNAL_IP	cyclone master가 접속할 slave GOLDILOCKS IP가 HOST_IP와 다를 때 사용한다. (Master, slave가 wan 구간에 있는 경우)	Slave
HOST_PORT	GOLDILOCKS가 운영 중인 host port를 설정한다.	Master/ slave
PORT	Master/ slave 통신에 사용될 port를 설정한다.	Master/ slave
USER_ID	사용자 이름을 설정한다.	Master/ slave
USER_PW	사용자 암호를 설정한다.	Master/ slave
USER_ENCRYPT_PW	암호화 된 사용자 암호를 설정한다.	Master/ slave
CAPTURE_TABLE	이중화할 테이블을 설정한다.	Master
LOG_PATH	LOGMIRROR와 연동할 때 사용되며 redo log file의 위치를 설정한다.	Master
PROTOCOL	GOLDILOCKS에 연결하는 connection type을 설정한다. (D/A or TCP)	Master/ slave
READ_LOG_BLOCK_COUNT	CAPTURE가 동작할 때 한 번에 읽어들이 데이터의 양을 설정한다.	Master
TRANS_SORT_ARRAY_SIZE	CAPTURE에 할당될 BUFFER의 크기를 설정한다.	Master
TRANS_FILE_PATH	CAPTURE 할 때 임시로 생성되는 file이 저장될 위치를 설정한다.	Master
SYNCHER_COUNT	SYNC 기능을 사용할 때 적용되며, 데이터 INSERT를 동시에 수행하는 SYNCHER의 개수를 설정한다.	Master
SYNC_ARRAY_SIZE	SYNC 기능을 사용할 때 적용되며, 동시에 데이터를 INSERT하는 array의 사이즈를 설정한다.	Master
GIVEUP_INTERVAL	이중화를 수행하는 속도가 GOLDILOCKS 속도를 따라가지 못할 경우 이중화를 포기하도록 설정한다.	Master

이름	설명	적용범위
APPLIER_COUNT	이중화를 적용할 때 동시에 수행하는 APPLIER의 개수를 설정한다.	Slave
APPLY_COMMIT_SIZE	이중화를 적용할 때 COMMIT 하기 위한 최대 사이즈를 설정한다.	Slave
APPLY_TABLE	이중화 테이블이 적용될 테이블을 설정한다.	Slave
MASTER_IP	CYCLONE master가 운영되고 있는 장비의 IP address를 설정한다.	Slave
PROPAGATE_MODE	CYCLONE으로 적용한 데이터를 PROPAGATE 할지 여부를 설정한다.	Slave
UPDATE_APPLY_MODE	Update 할 때 동작을 구분한다. (기본값은 0 이다) 0: Primary key만 동일할 경우에 UPDATE를 수행한다. 1: Primary key와 update 수행 이전 값이 동일할 경우에 UPDATE를 수행한다. 2: Primary key만 동일한 경우에 UPDATE를 수행하는데 update 수행 이전/이후 값을 비교하여 다를 경우 로그에 남긴다.	Slave
TCP_NODELAY	Socket의 TCP_NODELAY 옵션을 설정한다. (기본값은 1 이다) 0: TCP_NODELAY off 1: TCP_NODELAY on	Master
HEARTBEAT_TIMEOUT	이중화 연결 후에 network가 끊기거나 시스템 장애로 인해 연결이 원활하지 않을 경우, 연결을 유지하는 최대 시간 (초)을 설정한다.	Master/ slave
LOG_CAPTURE_INTERVAL_1	Capture의 수행 주기를 설정한다. 해당 값으로 10 회 수행한 후에 변경 사항이 없을 경우에는 LOG_CAPTURE_INTERVAL_2의 값으로 전환되어 capture를 수행한다. (기본값은 0.2 초이다.)	Master
LOG_CAPTURE_INTERVAL_2	Capture의 수행 주기를 설정한다. LOG_CAPTURE_INTERVAL_1의 값으로 수행한 후에 변경 사항이 없을 경우 capture 수행 주기를 설정한다. (기본값은 1초이다.)	Master
CLUSTER	Master가 cluster 환경일 경우 master의 접속 정보를 기술한다. 한 개의 master는 다음과 같은 세 개의 정보로 구성된다. - ID (1 이상의 값으로 설정되며 구분자이다.) - MASTER_IP - PORT	Slave
ORACLE_DRIVER	Oracle에서 제공하는 Oracle driver의 파일 위치를 기술한다.	Slave
PACKET_COMPRESSION_MODE	Master와 Slave 통신 데이터의 압축 여부를 설정한다. (1: Enable, 0: Disable, Default: Enable)	Master

환경설정 옵션

COMM_CHUNK_COUNT

- CYCLONE의 master와 slave의 데이터 통신에 사용될 buffer (chunk) 개수를 설정한다.
- 16M * N (설정값)으로 자원을 할당한다.
- 기본값은 32이며 실제 크기는 16M * 32 = 512M 이다.
- 최소값은 10 이다.

- Master와 slave에서 설정할 수 있다.
 - 너무 작은 값으로 설정하여 buffer가 모자랄 경우에는 성능이 느려진다.
- 모든 그룹에 적용되는 설정

```
COMM_CHUNK_COUNT = 10
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    COMM_CHUNK_COUNT=20
    ....
    ....
}
```

DSN

- GOLDILOCKS에 접속할 때 필요한 Data Source Name을 설정한다.
- Master와 slave에서 설정할 수 있다.
- Default 값은 GOLDILOCKS 이다.
- 모든 그룹에 적용되는 설정

```
DSN=GOLDILOCKS
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    DSN=GOLDILOCKS
    ....
    ....
}
```

GROUP_NAME

- 장비 내에서 CYCLONE의 운영을 구분하기 위해 필요하며 운영 process가 생성되는 단위이다.
- CYCLONE을 그룹 단위로 시작하거나 종료할 때 구분자가 된다.
- 한 번 설정하면 이후에 변경하지 않아야 한다. 변경할 경우 새로운 그룹으로 인식된다.
- Master와 slave에서 설정할 수 있다.

- Master와 slave 간에 접속할 때는 GROUP_NAME이 아니라 PORT를 이용하여 구분한다.
- 동일한 장비에서 GROUP_NAME은 중복되지 않아야 한다.
- 중괄호 {}를 사용해야 한다.

```
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

HOST_IP

- CYCLONE이 접속하고자 하는 GOLDDILOCKS의 IP address를 설정한다.
- Master와 slave에서 설정할 수 있다.
 - PROTOCOL이 TCP로 설정된 경우에만 유효하다.
- 모든 그룹에 적용되는 설정

```
HOST_IP = 127.0.0.1
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    HOST_IP = 127.0.0.1
    ....
    ....
}
```

HOST_EXTERNAL_IP

- Slave에서 설정할 수 있다.
- sync로 동작할 때 CYCLONE master에서 slave측 GOLDDILOCKS에 접속하는데, 이 때 CYCLONE master에서 접속할 slave측 GOLDDILOCKS IP가 HOST_IP와 다를 때 사용한다.
 - Master, slave가 wan 구간에 있어서 LAN 상의 IP와 wan 상의 IP가 다를 때 사용한다.
 - CYCLONE slave에서 이 값을 CYCLONE master로 전달하고 CYCLONE master에서 사용한다.
- 모든 그룹에 적용되는 설정

```
HOST_EXTERNAL_IP = 192.168.0.120
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    HOST_EXTERNAL_IP = 192.168.0.120
    ....
    ....
}
```

HOST_PORT

- CYCLONE이 접속하고자 하는 GOLDDILOCKS의 port를 설정한다.
- HOST_IP와 함께 설정해야 한다.
- Master와 slave에서 설정할 수 있다.

- 모든 그룹에 적용되는 설정

```
HOST_PORT = 22531
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    HOST_PORT = 22531
    ....
    ....
}
```

PORT

- Master와 slave 간에 통신할 때 사용되는 PORT를 설정한다.
- GROUP 간에는 중복된 PORT를 설정하면 안되고 GROUP 마다 유일한 값을 사용해야 한다.
- GROUP_NAME 내에서만 설정할 수 있으며, 반드시 설정되어야 한다.

- 그룹 내에서만 설정할 수 있다.

```
GROUP_NAME = testGROUP
{
    PORT = 21102
    ....
    ....
}
```

```
}
```

USER_ID

- GOLDILOCKS 접속에 필요한 사용자 ID를 설정한다.
- Master와 slave에서 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
USER_ID = testID
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    USER_ID = testID
    ....
    ....
}
```

USER_PW

- GOLDILOCKS 접속에 필요한 사용자 암호를 설정한다.
- Master와 slave에서 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
USER_PW = testPW
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    USER_PW = testPW
    ....
    ....
}
```

USER_ENCRYPT_PW

- GOLDILOCKS 접속에 필요한 사용자 패스워드를 encrypt하여 설정한다.
- USER_PW를 대신하여 사용한다.
- Encrypt 된 사용자 패스워드는 [cyclone --encrypt 사용자패스워드 --key 암호화할key]로 생성한다.
- 해당 설정값을 사용한 경우 cyclone을 실행할 때 --key 옵션을 사용해야 한다. (이 때 --encrypt로 생성한 key와 동일한 key 값을 사용해야 한다.)
- 모든 그룹에 적용되는 설정

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
    ....
    ....
}
```

CAPTURE_TABLE

- 이중화할 테이블을 설정한다.
 - *스키마이름.테이블 이름* 형식으로 설정한다.
- Master에서만 설정할 수 있다.
- 그룹 내에서만 설정할 수 있다.
- 여러개의 테이블을 명세할 경우, 소괄호 ()를 사용해야 한다.

```
GROUP_NAME = testGROUP
{
    CAPTURE_TABLE =
    (
        testSchema1.testTable1,
        testSchema1.testTable2,
        testSchema2.testTable1
    )
}
```


LOG_PATH

- LOGMIRROR와 연동해서 수행할 때 사용된다.
- LOGMIRROR가 저장하는 redo log file의 경로를 설정한다.
 - 절대 경로를 사용해야 한다.
 - 경로에는 single quote (')를 사용해야 한다.
- 모든 그룹에 적용되는 설정

```
LOG_PATH = '/data/wal/'
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    LOG_PATH = '/data/wal/'
    ....
    ....
}
```

PROTOCOL

- 운영 중인 GOLDDLOCKS에 접속할 type을 설정한다.
- DA 또는 TCP로 설정할 수 있다.
- PROTOCOL이 DA로 설정되는 경우, GOLDDLOCKS에 접속할 때 HOST_IP와 HOST_PORT를 사용하지 않는다.
 - 단, DA라도 slave에서는 SYNC를 위해 HOST_IP와 HOST_PORT를 사용할 수 있다.
- Default 값은 DA 이다.
- 모든 그룹에 적용되는 설정

```
PROTOCOL = DA
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    PROTOCOL = DA
    ....
    ....
}
```

READ_LOG_BLOCK_COUNT

- Redo log file을 캡처할 때, 한 번에 읽어들이는 로그 블록의 개수를 설정한다.
- Master에서만 설정할 수 있다.
- 로그 블록 한 개 사이즈는 512 bytes 이다.
- 기본값은 40960 이며, 실제 읽어들이는 사이즈는 20 Mbytes 이다.
 - 최소값은 100 이다.
- 모든 그룹에 적용되는 설정

```
READ_LOG_BLOCK_COUNT = 1024
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    READ_LOG_BLOCK_COUNT = 1024
    ....
    ....
}
```

TRANS_SORT_AREA_SIZE

- Redo log file을 캡처할 때 필요한 메모리 공간을 설정한다.
- Master에서만 설정할 수 있다.
- 단위는 MB (Megabyte) 이다.
- 기본값은 500 MB 이다.
 - 최소값은 10 MB 이다.
- 너무 작은 값으로 설정할 경우, 성능이 저하된다.
- 모든 그룹에 적용되는 설정

```
TRANS_SORT_AREA_SIZE = 300
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    TRANS_SORT_AREA_SIZE = 300
    ....
    ....
}
```

TRANS_FILE_PATH

- TRANS_SORT_AREA_SIZE 이상의 공간이 필요할 경우 임시 파일을 생성하는데, 이 때 해당 임시파일이 저장되는 경로를 설정한다.
 - 절대 경로를 사용해야 한다.
 - 경로에는 single quote (')를 사용해야 한다.
- Master에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
TRANS_FILE_PATH = '/data/TmpTrans'
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    TRANS_FILE_PATH = '/data/TmpTrans'
    ....
    ....
}
```

SYNCHER_COUNT

- 동기화를 수행할 때 사용된다.
 - 동기화 수행에 참여하는 SYNCHER의 개수를 설정한다.
 - 단위는 개수로 입력해야 한다.
- Master에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
SYNCHER_COUNT = 8
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    SYNCHER_COUNT = 8
    ....
    ....
}
```

SYNC_ARRAY_SIZE

- 동기화를 수행할 때 사용된다.
 - 동기화를 수행할 때 한 번에 삽입(INSERT)되는 레코드의 단위를 설정한다.
 - 단위는 개수로 입력해야 한다.
- Master에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
SYNC_ARRAY_SIZE = 1000
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    SYNC_ARRAY_SIZE = 1000
    ....
    ....
}
```

GIVEUP_INTERVAL

- GOLDILOCKS와 CYCLONE의 INTERVAL이 설정값보다 클 경우에 이중화를 포기하고 CYCLONE을 종료한다.
 - 단위는 REDO LOG BLOCK 개수로 입력해야 한다.
- Master에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
GIVEUP_INTERVAL = 10000
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    GIVEUP_INTERVAL = 10000
    ....
    ....
}
```

APPLIER_COUNT

- 이중화를 적용하는 APPLIER의 개수를 설정한다.
 - Parallel factor를 나타낸다.
- 기본값은 6이며, 여섯 개의 session이 생성된다.
 - 최대값에 제한은 없지만 너무 큰 값을 설정할 경우 APPLIER 사이에 contention이 발생할 수 있다.
 - 설정값에 따라 성능에 중요한 영향을 미친다.
 - 해당 설정값만큼 session이 생성되며 동시에 이중화를 적용한다.
- Slave에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
APPLIER_COUNT = 16
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    APPLIER_COUNT = 16
    ....
    ....
}
```

APPLY_COMMIT_SIZE

- 이중화를 수행할 때 COMMIT을 수행하는 트랜잭션의 개수를 설정한다.
 - 원본 데이터베이스에서 COMMIT 된 트랜잭션을 원격 데이터베이스에 이중화할 때 한 번에 트랜잭션을 수행한 후 COMMIT 한다.
 - 설정값은 최대값을 나타낸다.
- 설정값에 따라 성능에 영향을 미친다.
- Slave에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```
APPLY_COMMIT_SIZE = 1000
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    APPLY_COMMIT_SIZE = 1000
    ....
}
```

```

    ....
}

```

APPLY_TABLE

- 이중화 테이블이 적용될 테이블을 설정한다.
 - *이중화 테이블이름 TO 적용될 테이블* 형식으로 설정한다.
 - 테이블 간의 이름은 서로 같지 않아도 된다.
- Slave에서만 설정할 수 있다.
- 그룹 내에서만 설정할 수 있다.
- 여러 개의 테이블을 기술할 경우, 소괄호 ()를 사용해야 한다.

```

GROUP_NAME = testGROUP
{
    APPLY_TABLE =
    (
        testSchema1.testTable1 TO testSchema1.testTable1,
        testSchema1.testTable2 TO testSchema2.testTable3,
        testSchema2.testTable3 TO testSchema2.testTable4
    )
}

```

MASTER_IP

- CYCLONE master가 운영 중인 장비의 IP address를 설정한다.
- Slave에서만 설정할 수 있다.
- 모든 그룹에 적용되는 설정

```

MASTER_IP = 192.168.0.100

```

- 특정 그룹에 적용되는 설정

```

GROUP_NAME = testGROUP
{
    MASTER_IP = 192.168.0.100
    ....
    ....
}

```

PROPAGATE_MODE

- CYCLONE이 CIRCULAR하게 구성되었을 경우, 한 CYCLONE이 적용한 트랜잭션을 다른 CYCLONE이 적용할 지 여부를 설정한다.
- Slave에서만 설정할 수 있다.
 - 기본값은 0으로써 이 경우 PROPAGATE 하지 않는다.
 - PROPAGATE 할 경우 1, 그렇지 않을 경우 0 으로 설정한다.
- 모든 그룹에 적용되는 설정

```
PROPAGATE_MODE = 1
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    PROPAGATE_MODE = 1
    ....
    ....
}
```

UPDATE_APPLY_MODE

- Slave에서 update 할 때 사용된다.
 - 0: Primary key만 동일할 경우에 UPDATE를 수행한다. (Default)
 - 1: Primary key와 update 수행 이전 값이 동일할 경우에 UPDATE를 수행한다.
 - 2: Primary key만 동일한 경우에 UPDATE를 수행하지만, update 수행 전/ 후 값을 비교하여 다를 경우에 로그를 남긴다.
- 0으로 설정된 경우, 이전 값을 확인하지 않고 update한다.
- 1로 설정된 경우, 이전 값이 다르면 update에 실패하고 conflict log를 남긴다.
- 2로 설정된 경우, 이전 값이 다르면 update는 성공하고 conflict log를 남긴다.
- 모든 그룹에 적용되는 설정

```
UPDATE_APPLY_MODE = 1
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    UPDATE_APPLY_MODE = 1
    ....
}
```

```
.....
}
```

노트

데이터 타입이 long varchar 또는 long varbinary인 column의 경우, 성능상의 이유로 길이만 비교한다.

TCP_NODELAY

- Master에서만 사용된다.
- CDC 전송 socket에 대한 TCP_NODELAY 옵션을 설정한다. (Sync는 이 옵션의 영향을 받지 않는다. TCP_NODELAY on 으로 고정된다.)
 - 0: socket TCP_NODELAY 옵션을 off 한다.
 - 1: socket TCP_NODELAY 옵션을 on 한다. (Default)
- 모든 그룹에 적용되는 설정

```
TCP_NODELAY = 1
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    TCP_NODELAY = 1
    ....
    ....
}
```

HEARTBEAT_TIMEOUT

- Master와 slave에서 설정할 수 있다.
- Master와 slave의 이중화 연결 후에 network가 끊기거나 시스템 장애에 의해 연결이 원활하지 않을 경우, 연결 유지를 지속하는 최대 시간 (초)을 설정한다.
- 기본값은 30 (초)이다.
 - 최소값은 10 (초)이다.
- 모든 그룹에 적용되는 설정


```
HEARTBEAT_TIMEOUT = 40
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    HEARTBEAT_TIMEOUT = 40
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_1

- Master에서 설정할 수 있다.
- Master에서 동작하는 capture의 수행 주기를 설정한다. 수행 주기는 밀리초 단위로 설정한다
- Redo log file이 갱신되는 것을 빠른 시간내에 감지하기 위해서 사용된다.
- 해당 설정값으로 10 회 수행한 후에 redo log file이 갱신되지 않을 경우에는 LOG_CAPTURE_INTERVAL_2로 전환하여 수행한다.
- 기본값은 200 (0.2초)이며 단위는 밀리초 (ms)이다.
- 모든 그룹에 적용되는 설정

```
LOG_CAPTURE_INTERVAL_1 = 200
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_1 = 200
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_2

- Master에서 설정할 수 있다.
- Master에서 동작하는 capture의 수행 주기를 설정한다. 수행 주기는 밀리초 단위로 설정한다
- Redo log file이 갱신되는 것을 빠른 시간내에 감지하기 위해서 사용된다.
- LOG_CAPTURE_INTERVAL_1 설정값으로 10 회 수행한 뒤에 redo log file이 갱신되지 않을 경우에는 해당 값으로 전환하여 수행한다.
- 기본값은 1000 (1초)이며 단위는 밀리초 (ms)이다.

- 모든 그룹에 적용되는 설정

```
LOG_CAPTURE_INTERVAL_2 = 1000
```

- 특정 그룹에 적용되는 설정

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_2 = 1000
    ....
    ....
}
```

CLUSTER

- Slave에서 설정할 수 있다.
- Cluster 환경에서 운영되는 N 개 master의 접속정보를 설정한다. (ID, MASTER_IP, PORT 정보로 구성된다.)
 - ID: Slave에서 master를 구분하는 구분자이며 숫자만 입력할 수 있다. 해당 값은 한 번 설정한 이후에는 변경되지 않아야 한다.
 - MASTER_IP: Master가 운영되고 있는 IP를 설정한다.
 - PORT: Master가 운영되고 있는 PORT를 설정한다.
- Cluster는 group 내부에서만 설정할 수 있다.

```
GROUP_NAME = testGROUP
{
    CLUSTER = (ID=1, MASTER_IP=192.0.0.100, PORT=21102),
              (ID=2, MASTER_IP=192.0.0.101, PORT=21103)
    ....
    ....
}
```

ORACLE_DRIVER

- Slave에서 설정할 수 있다.
- Oracle에서 제공하는 ODBC driver 파일의 위치와 이름을 기술한다.
 - 현재는 Oracle 11g 버전만 지원한다.
- 모든 그룹에서 설정할 수 있다.

```

ORACLE_DRIVER = '/app/oracle/product/11.2.0/db_1/lib/libsqora.so.11.1'
GROUP_NAME = testGROUP
{
    ....
    ....
}

```

PACKET_COMPRESSION_MODE

- Master에서 설정할 수 있다.
- Master에서 slave로 전송되는 데이터의 압축 여부를 설정한다.
 - 1: Enable (Default)
 - 0: Disable
- 특정 그룹에서 설정할 수 있다.

```

GROUP_NAME = testGROUP
{
    PACKET_COMPRESSION_MODE = 1
    ....
    ....
}

```

44.4 운영하기

CYCLONE은 GOLDDILOCKS의 D/A 또는 C/S 환경에서 운영할 수 있다.

운영 환경에 따라 다음과 같이 CONFIG 파일 또는 ODBC.INI의 PROTOCOL 설정을 사용해야 한다.
CONFIG 설정은 ODBC.INI 설정보다 우선한다.

표 44-5 CONFIG, ODBC.INI 구분

설정	구분
PROTOCOL=DA	D/A 환경에서 사용된다. (Default)
PROTOCOL=TCP	C/S 환경에서 사용된다.

운영 중에 수행되는 내용은 trace log를 통해 볼 수 있다.

구분	파일
Cyclone master	\$GOLDDILOCKS_DATA/trc/cyclone_master.trc
Cyclone slave	\$GOLDDILOCKS_DATA/trc/cyclone_slave.trc

노트

Trace log에 저장되는 에러 메시지와 대처 방안은 **CYCLONE 에러 메시지와 해결 방법**을 참조한다.

GOLDDILOCKS 접속 정책

CYCLONE에서 GOLDDILOCKS에 접속하는 정책이다.

GOLDDILOCKS에 접속하는 기능은 두 가지가 있다. 하나는 CYCLONE에서 local GOLDDILOCKS에 접속하는 기능이고 다른 하나는 SYNC를 수행할 때 CYCLONE master에서 slave 측 GOLDDILOCKS에 원격으로 접속하는 기능이다. 두 가지 모두 CYCLONE config 설정값이 우선적으로 사용되고 그 다음에 odbc.ini를 사용한다. 단, SYNC를 수행할 때는 PROTOCOL 값은 무시되고 무조건 TCP로 설정된다.

GOLDDILOCKS 접속과 관련된 config property는 DSN, PROTOCOL, HOST_IP, HOST_EXTERNAL_IP, HOST_PORT, USER_ID, USER_PW 이다.

(USER_ENCRYPT_PW는 USER_PW를 대신하므로 USER_PW로 총칭한다.)

올바르게 설정된 예

- D/A로 접속하였기 때문에 USER_ID, USER_PW만으로도 가능하다.

구분	파일
CONFIG	USER_ID=test, USER_PW=test
odbc.ini의 GOLDDILOCKS 설정	-

- CONFIG나 odbc.ini 둘 중 한 군데에만 정보가 있으면 된다. D/A로 동작하기 때문에 HOST_IP, HOST_PORT는 무시된다.

구분	파일
CONFIG	HOST_IP=127.0.0.1, HOST_PORT=22581, USER_ID=test
odbc.ini의 GOLDDILOCKS 설정	USER_PW=test

- TCP로 접속하였고 CONFIG가 우선하므로 USER_ID는 test 계정으로 동작한다.

구분	파일
CONFIG	PROTOCOL=TCP, USER_ID=test, USER_PW=test
odbc.ini의 GOLDDILOCKS 설정	HOST_IP=127.0.0.1, HOST_PORT=22581, USER_ID=test2, USER_PW=test2

- D/A로 접속하더라도 slave에는 HOST_EXTERANL_IP를 설정할 수 있다.

구분	파일
CONFIG	PROTOCOL=DA, HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test
odbc.ini의 GOLDDILOCKS 설정	HOST_IP=127.0.0.1, HOST_PORT=22581, USER_PW=test

잘못 설정된 예

- D/A로 접속하는데 USER_PW가 없다.

구분	파일
CONFIG	HOST_EXTERNAL_IP=192.168.0.10
odbc.ini의 GOLDDILOCKS 설정	PROTOCOL=DA, USER_ID=test

- TCP로 접속하는데 HOST_IP가 없다.

구분	파일
CONFIG	HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test

구분	파일
odbc.ini의 GOLDDILOCKS 설정	PROTOCOL=TCP, HOST_PORT=22581, USER_PW=test

실행 옵션

CYCLONE을 실행할 때는 다음 옵션과 함께 사용해야 한다.

표 44-6 실행 옵션

옵션	설명	비고
--start -s	CYCLONE을 실행한다.	--master --slave와 함께 사용해야 한다.
--stop -t	CYCLONE을 종료한다.	--master --slave와 함께 사용해야 한다.
--master -m	Master 모드로 수행한다.	--start --stop과 함께 사용해야 한다.
--slave -l	Slave 모드로 수행한다.	--start --stop과 함께 사용해야 한다.
--status -u	CYCLONE의 운영상태를 보여준다.	--master --slave와 함께 사용해야 한다.
--conf -c	CYCLONE을 실행할 때 필요한 환경파일의 경로를 설정한다.	--conf CONFIG_FILE 형식으로 입력한다. --start와 함께 사용해야 한다. 명시적으로 설정하지 않을 경우, master는 \$GOLDDILOCKS_DATA/cyclone.master.conf, slave는 \$GOLDDILOCKS_DATA/cyclone.slave.conf를 사용한다.
--silent -i	메시지를 출력하지 않도록 한다.	-
--reset -r	이중화 운영정보를 초기화한다.	--reset TABLE_NAME 또는 --reset all 형식으로 입력한다. 여러 개의 table을 reset할 경우에는 single quote (') 안에 기술한다.
--group -g	특정 그룹을 설정한다.	--group GROUP_NAME 형식으로 입력한다.
--help -h	도움말을 출력한다.	-
--sync -n	데이터 동기화를 수행한다.	--master --slave와 함께 사용해야 한다.
--encrypt -e	주어진 key로 사용자 패스워드를 암호화한다.	-
--key -k	--encrypt 옵션을 실행할 때 암호화 key를 설정한다. config에 USER_ENCRYPT_PW가 사용된 경우 복호화 key를 설정한다.	-
--info -o	현재 이중화 되고 있는 테이블 상태를 보여준다.	--master, --group과 함께 사용해야 한다.
--recovery -v	Cluster 환경에서 단독모드로 수행 중이던 이중화를 다른 cluster member로 이관할 경우에 사용한다.	--recovery GROUP_NAME 형식으로 사용하며, GROUP_NAME은 이전에 수행되던 slave의 CYCLONE GROUP_NAME을 기술한다.
--stand-alone -S	Cluster 환경에서 단독모드로 동작하도록 한다. (Cluster환경에서는 기본적으로 cluster 모드로 동작한다.)	Master에서만 유효하다. (Slave에서는 환경설정파일에서 설정한다.)

옵션	설명	비고
--local -a	Cluster 환경에서 sharding 된 table을 동기화할 경우에 사용한다. 해당 cluster 그룹의 데이터만 동기화한다.	--sync와 함께 사용해야 한다.

- 기본 환경 파일을 사용하여 모든 그룹을 master 모드로 실행한다.

```
prompt> cyclone --master --start
```

- Master 모드로 모든 그룹을 종료한다.

```
prompt> cyclone --master --stop
```

- 기본 환경 파일을 사용하여 slave 모드로 모든 그룹을 실행한다.

```
prompt> cyclone --slave --start
```

- Slave 모드로 모든 그룹을 종료한다.

```
prompt> cyclone --slave --stop
```

- Master 모드로 TEST_GROUP 그룹만 실행한다.

```
prompt> cyclone --master --start --group TEST_GROUP
```

- Master 모드로 운영 중인 그룹 중에 TEST_GROUP 그룹만 종료한다.

```
prompt> cyclone --master --stop --group TEST_GROUP
```

- Slave 모드로 TEST_CONFIG 파일을 설정하여 실행한다.

```
prompt> cyclone --slave --start --conf TEST_CONFIG
```

- GOLDILOCKS 사용자 패스워드를 암호화한다.

```
prompt> cyclone --encrypt test --key 1234
Cyclone Encrypted Passwd : 'YFH+bpBNvk='
```

- config에 USER_ENCRYPT_PW가 설정되었다.

```
prompt> cyclone --master --start --key 1234
```

- Master와 slave의 기존 이중화를 제거하고 새로 시작하게 한다.

```
prompt> cyclone --master --start --reset all
prompt> cyclone --slave --start --reset all
```

- Master에서 T1, T2 테이블의 기존 이중화만 제거하고 현재 시점부터 새로 시작하게 한다.

```
prompt> cyclone --master --start --reset 'T1, T2'
prompt> cyclone --slave --start
```

- Master 측에서 현재 이중화 상태를 테이블별로 보여준다.

```
prompt> cyclone --master --info --group GROUP1
=====
GROUP NAME = GROUP1
=====
SCHEMA NAME      : PUBLIC
TABLE NAME       : T1 (GIVE-UP (DDL-LSN:129541))
PHYSICAL ID      : 5299989643264
=====
SCHEMA NAME      : PUBLIC
TABLE NAME       : T2 (ACTIVE (CAPTURE-START-LSN:128922))
PHYSICAL ID      : 35549444308992
=====
SCHEMA NAME      : PUBLIC
TABLE NAME       : T3 (ACTIVE (CAPTURE-START-LSN:128922))
PHYSICAL ID      : 35558034243584
=====
SCHEMA NAME      : PUBLIC
TABLE NAME       : T4 (ACTIVE (CAPTURE-START-LSN:128922))
PHYSICAL ID      : 35566624178176
=====
TOTAL   COUNT : 4
GIVE-UP COUNT : 1
=====
```

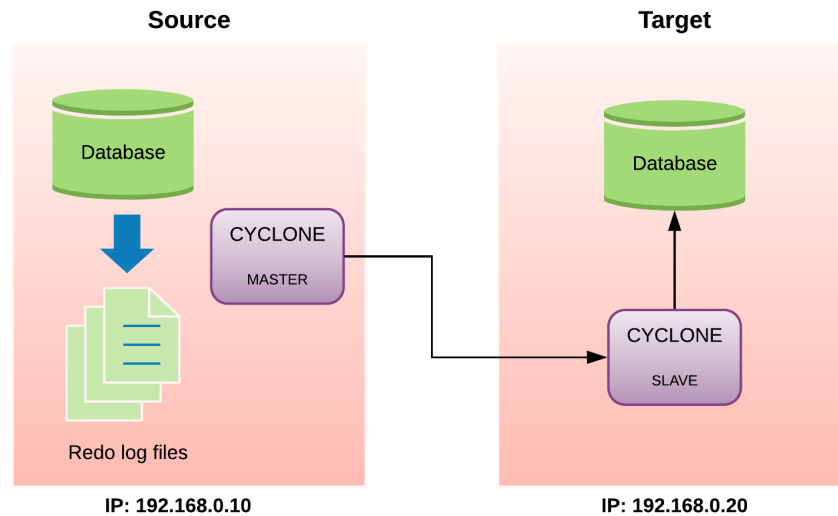
노트

그룹 또는 노드의 추가/삭제와 관련된 예는 **노드 추가와 삭제**를 참조한다.
 이중화 초기화와 관련된 예는 **이중화 초기화**를 참조한다.

44.5 CYCLONE 운영 예

CYCLONE 운영 구조는 다음과 같다.

그림 1 CYCLONE 운영 구조도



- 장비 구조
 - 데이터 원본이 되는 source 장비
 - GOLDDILOCKS IP: 192.168.0.10
 - GOLDDILOCKS port: 22581
 - 이중화할 테이블: T1, T2
 - 이중화를 위한 원격 target 장비
 - GOLDDILOCKS IP: 192.168.0.20
 - GOLDDILOCKS port: 22581

운영 순서

1. 원본 GOLDDILOCKS 환경 설정
2. 원격 GOLDDILOCKS 환경 설정
3. CYCLONE MASTER 환경 설정
4. CYCLONE SLAVE 환경 설정
5. 실행 및 운영

원본 GOLDBLOCKS 환경 설정

기술된 준비사항을 모두 수행한다.

테스트를 위해 T1, T2 table을 생성한다.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

원격 GOLDBLOCKS 환경 설정

사용자 등록 및 권한 설정을 수행한다.

CYCLONE MASTER 환경 설정

- CYCLONE MASTER 환경 설정 파일
 - 저장위치: \$GOLDBLOCKS_DATA/conf/cyclone.master.conf
 - Source 장비이다.
- 동일한 장비의 D/A로 접속하기 때문에 HOST 정보는 없다.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = GROUP1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

CYCLONE SLAVE 환경 설정

- CYCLONE SLAVE 환경 설정 파일
 - 저장위치: \$GOLDILOCKS_DATA/conf/cyclone.slave.conf
 - Target 장비이다.
- 동일한 장비의 D/A로 접속하지만 SYNC를 위해 HOST_IP와 HOST_PORT를 사용한다.

```

USER_ID = cdc_user
USER_PW = cdc_password
HOST_IP = 192.168.0.20
HOST_PORT = 22581
GROUP_NAME = GROUP1
{
    PORT = 21102
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}

```

실행 및 운영

- CYCLONE master 실행
 - Source 장비에서 실행해야 한다.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf
```

- CYCLONE SLAVE 실행
 - Target 장비에서 실행해야 한다.

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf
```

데이터 동기화

- 데이터 동기화
 - Master의 데이터를 slave에 모두 복사한 뒤에 이중화한다.
 - 동기화를 수행할 때 이중화에 참여하는 master의 table 데이터를 slave에서 운영 중인 table에 복사한다.
- 운영을 시작할 때 master와 slave 모두에 --sync 옵션을 적용해야 한다.
 - 모든 table을 sync할 경우에는 --sync all을 형식을 사용하고 특정 테이블을 sync할 경우에는 --sync TABLE_NAME 형식을 사용한다.
 - TABLE_NAME에는 master의 table 이름을 기술한다. (Slave로 실행하더라도 TABLE_NAME에는 master의 table 이름을 기술해야 한다.)
- D/A라도 slave의 CONFIG 또는 odbc.ini에 HOST_IP (또는 HOST_EXTERNAL_IP), HOST_PORT가 설정되어 있어야 한다.

노트

- 이중화에 참여하는 slave의 table에 데이터가 있을 경우 PK 중복 등으로 인해 데이터가 정상적으로 동기화되지 않을 수 있으므로 동기화를 수행하기 전에 slave의 table 데이터를 사용자가 직접 제거해야 한다.
- --sync 옵션을 사용하여 데이터를 동기화할 경우, 기존의 이중화 정보는 삭제된다. 즉, --sync 옵션을 사용하면 내부적으로 --reset 옵션이 강제로 설정된다.
- Master로 운영 중인 cyclone이 slave로 운영 중인 GOLDDILOCKS에 직접 접속하여 데이터를 동기화한다.

- CYCLONE master 실행
 - Source 장비에서 실행해야 한다.

```
cyclone --master --start --conf $GOLDDILOCKS_DATA/conf/cyclone.master.conf --sync all
```

- CYCLONE SLAVE 실행
 - Target 장비에서 실행해야 한다.

```
cyclone --slave --start --conf $GOLDDILOCKS_DATA/conf/cyclone.slave.conf --sync --all
```

- 다음은 sync 에러가 발생하는 예이다.
 - 에러를 방지하기 위해 slave의 테이블 데이터 삭제를 권장한다.
 - MASTER 실행

```
gSQL> CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL> INSERT INTO T1 VALUES( 1, 'HELLO' );
```

- SLAVE 실행

```
gSQL> CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL> INSERT INTO T1 VALUES( 1, 'BYE' );
```

- MASTER 실행

```
cyclone --master --start --sync all
```

- SLAVE 실행

```
cyclone --slave --start --sync all
SLAVE cyclone log
[RECEIVER] start the data synchronization.
[RECEIVER] Error Occurred.
STOP Done.
```

Primary key에 동일한 값을 사용하려고 했기 때문에 에러가 발생했으며 sync가 작동하지 않는다.

이중화 정보 초기화

- 이중화 정보 초기화
 - 이중화에 참여하는 테이블 전체 또는 특정 테이블에 대한 이중화를 현재 시점부터 다시 시작하도록 한다.
 - --reset TABLE_NAME 또는 --reset ALL 옵션을 사용하여 수행한다.
- Master와 slave에서 --reset 수행
 - Master에서는 이중화에 참여하는 table 정보와 column 정보등을 meta로 저장하고 있다. Master에서 --reset ALL 또는 Reset Table_name으로 RESET을 수행할 경우 해당 테이블의 meta를 제거하고 현재 시점의 정보를 사용하여 meta를 다시 구성한다.
 - DDL을 수행하여 이중화 과정에서 give up 된 table을 다시 이중화에 참여시키려면 반드시 master에서 --reset을 수행해야 한다.
 - Slave에서는 이중화를 재시작할 때 recovery 하기 위한 정보를 저장하고 있다. Master를 실행할 때와 마찬가지로 slave를 실행할 때도 --reset ALL이나 --reset TABLE_NAME을 기술해야 한다.

노트

- 이중화에 참여하는 모든 table을 초기화 하려면 master/ slave를 실행할 때 --reset ALL을 추가한다.

```
prompt> cyclone --master --start --reset ALL
prompt> cyclone --slave --start --reset ALL
```

- 특정 테이블의 이중화 정보를 초기화하려면 master를 실행할 때 --reset TABLE_NAME을 추가한다.
 - Slave를 수행할 때의 테이블 이름은 master를 실행할 때 사용된 테이블 이름과 동일하게 기술해야 한다.

```
prompt> cyclone --master --start --reset T1
prompt> cyclone --slave --start --reset T1
```

- 두 개 이상의 테이블을 초기화 하려면 single quote (')를 사용한다.

```
prompt> cyclone --master --start --reset 'T1 T2'
prompt> cyclone --master --start --reset 'T1, T2'
```

Recovery

- Recovery를 사용하는 경우
 - Cluster 환경에서 standalone 옵션을 사용하여 이중화하는 도중에 master의 장애로 인해 동일한 group의 다른 cluster member로 이중화를 이관하여 계속 진행할 경우에 사용된다.
 - Master에 --recovery GROUP_NAME으로 수행하며 GROUP_NAME은 기존에 이중화를 수행했던 slave의 group 이름을 기술한다.

노트

이중화 정보 초기화 절차

1. Cluster 환경에서 G1N1 member로 (standalone) 이중화를 수행한다.

```
prompt> cyclone --master --start --reset ALL --group g1n1_master --stand-alone
prompt> cyclone --slave --start --reset ALL --group g1n1_slave
```

2. 해당 cluster member를 종료하거나 서비스를 이관하기 위해 기존 cyclone 운영을 중단한다.

```
prompt> cyclone --master --stop
prompt> cyclone --slave --stop
```

3. 기존 이중화 정보 (G1N1)를 사용하여 새로운 이중화 (G1N2)를 서비스한다.

```
prompt> cyclone --master --start --reset all --group g1n2_master --stand-alone --recovery g1n1_slave  
prompt> cyclone --master --start --reset all --group g1n2_slave
```

44.6 Cluster 환경에서 CYCLONE 운영

준비사항

- Cluster 환경에서 동일한 cluster group 내의 member를 이중화하기 위한 방법을 제공한다.
 - Standalone 환경에서 master와 slave는 1:1 구조로 운영된다.
 - Cluster 환경에서 master와 slave는 N:1 구조로 운영된다.
- 실행 가능 환경
 - Master는 cluster 환경이어야 한다. (자동으로 인식한다.)
 - Slave는 standalone, cluster 환경 모두에서 실행할 수 있다.
- Slave와 접속된 master의 상태를 node라고 한다.

노트

- 하나의 slave에 접속하는 master는 동일한 cluster group내의 member여야 한다.
 - 다른 cluster group의 member는 다른 slave에서 운영되어야 한다.
- Master가 cluster 환경일 경우, CYCLONE이 기본적으로 cluster 모드로 작동하므로 standalone 모드로 동작해야 할 경우에는 반드시 --stand-alone 옵션을 사용해야 한다.
 - Master를 수행할 때만 --stand-alone 옵션을 사용한다.
 - 환경 설정 파일에 CLUSTER 옵션을 추가하여 master의 cluster 정보를 기술하면 slave가 cluster 모드로 동작한다.

운영

실행과 종료

- Standalone 환경과 cluster 환경은 master와 slave의 종료 시점이 다르다.
 - Standalone 환경
 - Master가 종료될 경우, slave도 자동 종료된다.
 - Slave가 종료될 경우, master는 대기상태로 전환되지만 종료되지는 않는다.
 - Cluster 환경
 - Master가 종료되더라도 slave는 종료되지 않는다. 만약 이미 접속한 다른 master가 있다면 지속적으로 이중화가 운영된다. 만약 접속된 master가 없을 경우에는 slave가 대기 상태로 전환된다.
 - Slave가 종료될 경우, 접속되어 있던 master는 모두 자동으로 종료된다.

Node Type

Cluster 환경에서 CYCLONE은 slave와 접속하는 master를 trust node와 non-trust node로 구분한다.

표 44-7 Cluster 환경에서 node 운영 방식

Node	설명	이중화 가능 여부
Trust node	이중화에 참여하는 모든 table이 online 상태인 node이다.	O
Non-trust node	이중화에 참여하는 table 중 하나 이상이 offline 상태인 node이다.	X

Node 상태변화에 따른 운영

- Non-trust node에서 trust node로 변경
 - Offline table이 존재하는 상태에서 해당 cluster member에 rebalance를 수행하여 모든 table이 online 상태로 변경될 경우에 수행된다.
 - 해당 node는 non-trust node에서 trust node로 자동으로 변경되며 이중화 운영에 참여한다.
- Trust node에서 non-trust node로 변경
 - 해당 cluster member가 종료되어 동작하지 않는 상황에서 운영 중인 다른 cluster member에서 이중화되고 있는 table이 변경될 경우에 수행된다.
 - Non-trust node는 cluster member가 조인할 때 자동으로 변경되며 이중화 운영에서 제외된다.

노트

- 이중화에 참여하는 master 중에 trust node가 하나 이상 있어야 이중화가 유지된다.
- Trust node는 특정 node에 국한되지 않으며, node의 종류에 관계없이 trust node이기만 하면 된다.
- 이중화에 참여하는 모든 master가 non-trust node가 될 경우, 이중화는 더 이상 진행되지 않으며, 이후 trust node가 되더라도 이중화가 다시 진행되지 않는다.
- Cluster 환경에서 지속적으로 이중화를 진행하기 위해서는 항상 trust node가 하나 이상 있어야 한다.

수행

환경 설정 파일

Master의 환경 설정 파일은 standalone과 동일하다.

다음은 G1N1의 master 환경을 설정하는 예로써 해당 장비의 IP는 192.168.0.10 이라고 가정한다.

- Cluster 환경이라 하더라도 master에서 변경할 부분은 없다.

```

USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1N1_Master
{
    PORT = 21011
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}

```

다음은 G1N2의 master 환경을 설정하는 예로써 해당 장비의 IP는 192.168.0.20 이라고 가정한다.

- Cluster 환경이라 하더라도 master에서 변경할 부분은 없다.

```

USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1N2_Master
{
    PORT = 21012
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}

```

Slave의 환경설정 파일에는 CLUSTER를 추가하여 master의 정보를 기술한다.

- CLUSTER 항목은 ID, MASTER_IP, PORT로 구성된다. (순서는 관계없다)
 - ID는 숫자로 입력하는데 slave에서 master를 구분하기 위한 내부 구분자로 사용된다. 한 번 설정한 이후에는 변경할 수 없다. (변경할 경우, 다른 master로 인식된다.)
 - MASTER_IP는 CLUSTER 멤버에서 CYCLONE MASTER가 운영되는 IP이다.
 - PORT는 CLUSTER 멤버에서 CYCLONE MASTER가 운영되는 PORT이다.
- 하나의 CLUSTER 정보는 괄호 안에 기술하고 괄호 사이에 comma (,)를 넣어 구분한다.

CLUSTER의 master 정보인 MASTER_IP, PORT는 CLUSTER 환경 설정에 기술한다.

```

USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1_Slave

```

```
{
  CLUSTER = ( ID = 1, MASTER_IP = 192.168.0.10, PORT = 21011 ),
             ( ID = 2, MASTER_IP = 192.168.0.20, PORT = 21012 )
  APPLY_TABLE =
  (
    T1 T0 T1,
    T2 T0 T2
  )
}
```

실행 방법

Master에서 수행

- G1N1에서 수행한다.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --group G1N1_Master
```

- G1N2에서 수행한다.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --group G1N2_Master
```

Slave에서 수행

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf --group G1_Slave
```

SYNC

Master에서 수행

G1N1에서 수행한다. (SYNC를 수행할 master로써 --sync 옵션을 사용한다.)

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --sync all
```

Slave에서 수행

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf --sync all
```

노트

Sharded table이 이중화에 참여할 경우 sync 할 때 --local 옵션을 사용해야 한다. 이는 sharded table의 데이터가 각 cluster group에만 존재하고 --local 옵션없이 sync를 수행할 경우에 다른 cluster group의 데이터까지 동기화되기 때문이다. 그러나 cluster환경에서 이중화에 참여하는 모든 table이 cloned일 경우에는 동기화할 때 --local 옵션을 사용하지 않아도 된다

기타

--reset 옵션은 standalone 환경에서 쓰이는 것과 동일하게 사용할 수 있다.

44.7 모니터링 (CYMON)

CYMON (CYclone MONitor)은 CDC 이중화 톨인 CYCLONE을 모니터링하기 위한 톨이다. CYCLONE 모니터링 정보를 master로 운영 중인 GOLDDILOCKS에 주기적으로 갱신한다.

표 44-8 실행 파일

파일 이름	구분
cymon	cyclone용 monitor

환경 설정 파일

cymon의 환경 설정 파일은 cyclone을 master로 실행할 때 사용한 환경 파일을 동일하게 사용한다. --conf 옵션을 사용하여 특정 환경 설정 파일을 설정하지 않을 경우, cyclone을 master로 실행할 때 기본값으로 \$GOLDDILOCKS_D ATA/conf/cyclone.master.conf 파일을 읽는다.

노트

CYMON은 CYCLONE이 master로 운영 중인 장비와 동일한 장비에서 실행되어야 한다.

모니터링 내용

CYMON

CYMON은 CYCLONE의 운영 정보를 GOLDDILOCKS의 CYCLONE_MONITOR_INFO 테이블에 주기적으로 갱신한다. 모니터링 내용은 다음과 같다.

표 44-9 CYCLONE_MONITOR_INFO

Column	설명
GROUP_NAME	CYCLONE이 실행된 group 이름이다.
TIME	해당 정보가 갱신된 시간 정보 (YYYY-MM-DD HH24:MI:SS)이다.
MASTER_STATE	Master로 운영 중인 CYCLONE의 상태이다. - N/A: 상태를 알 수 없다. - READY: Slave가 접속하기를 기다리는 중이다. - RUNNING: Slave가 접속하여 이중화를 수행하는 중이다.
	Slave로 운영 중인 CYCLONE의 상태이다.

Column	설명
SLAVE_STATE	- N/A: 상태를 알 수 없다. - RUNNING: 이중화를 수행 중이다.
MASTER_PORT	Master로 운영 중인 CYCLONE이 slave를 기다리고 있는 port 정보이다.
SLAVE_IP	CYCLONE slave로 접속한 장비의 IP address 정보이다.
REDO_LOG_FILESEQ	운영 중인 GOLDBLOCKS redo log file의 sequence 번호이다.
REDO_LOG_BLOCKSEQ	운영 중인 GOLDBLOCKS redo log file의 block sequence 번호이다.
CAPTURE_FILESEQ	CYCLONE master가 capture 중인 redo log file의 sequence 번호이다.
CAPTURE_BLOCKSEQ	CYCLONE master가 capture 중인 redo log file의 block sequence 번호이다.
CAPTURE_INTERVAL	CYCLONE master가 처리해야 할 (남아있는) redo log block의 개수이다. 1 block = 512 bytes (Redo log block size = 512 bytes)
CAPTURE_INTERVAL_SIZE	CYCLONE master가 처리해야 할 (남아있는) redo log file의 크기이다.
TOTAL_TX_COUNT	CYCLONE master에서 capture한 모든 transaction의 개수이다.
CAPTURE_TX_COUNT	CYCLONE master에서 capture한 transaction 중에 이중화할 대상이 포함된 transaction의 개수이다.
CAPTURE_COMMIT_LSN	CYCLONE master에서 capture한 마지막 transaction의 commit log 번호이다. 더 이상 capture할 transaction이 없으면 변경되지 않는다.

노트

INTERVAL 정보는 master로 운영 중인 CYCLONE의 redo log file을 분석하는 CAPTURE 정보이며, CYCLONE SLAVE에서 APPLIER에 의해 반영된 정보는 아니다.

실행 및 모니터링

실행 옵션

옵션	설명	비고
--conf -c	환경파일의 경로를 설정한다.	--conf CONFIG_FILE 형식으로 입력한다. CYCLONE을 master 환경에서 실행할 때 사용한 환경 파일과 동일한 파일로 설정해야 한다.
--start -s	CYMON을 실행한다.	-
--stop -t	CYMON을 종료한다.	-
--status -u	CYMON의 운영상태를 보여준다.	-
--cycle -y	모니터링 정보의 갱신 주기를 설정한다.	--cycle X 형식으로 입력한다. 초단위로 입력해야 한다.
--key -k	config에 USER_ENCRYPT_PW가 사용된 경우 복호화 key를 설정한다.	-

옵션	설명	비고
--trace -r	Capture의 모니터링 정보를 trace 로그에도 동시에 기록한다.	Trace log는 \$GOLDILOCKS_DATA/trc/cymon.trc에 저장된다.
--silent -i	메시지를 출력하지 않도록 한다.	-
--help -h	도움말을 출력한다.	-

사용 예

- CYMON 실행 (cyclone_monitor_info 테이블 모니터링 정보를 갱신한다.)
 - CYMON을 실행하여 1초 주기로 모니터링 정보를 갱신한다. CYMON을 종료할 때까지 지속적으로 정보를 갱신한다.

```
cymon --start --cycle 1
```

- 모니터링 정보 확인

```
gSQL> \set vertical on
gSQL> select * from cyclone_monitor_info;
      GROUP_NAME # GROUP1
           TIME # 2015-01-13 17:34:53
MASTER_STATE # READY
SLAVE_STATE # N/A
MASTER_PORT # 21102
      SLAVE_IP # null
REDO_LOG_FILESEQ # 0
REDO_LOG_BLOCKSEQ # 52392
CAPTURE_FILESEQ # 0
CAPTURE_BLOCKSEQ # 0
CAPTURE_INTERVAL # 0
CAPTURE_INTERVAL_SIZE # 0
```

- 위의 정보를 통해 CYCLONE MASTER만 동작하고 있고 SLAVE는 대기 중임을 확인할 수 있다.

```
gSQL> \set vertical on
gSQL> select * from cyclone_monitor_info;
      GROUP_NAME # GROUP1
           TIME # 2015-01-13 17:36:17
MASTER_STATE # RUNNING
SLAVE_STATE # RUNNING
MASTER_PORT # 21102
      SLAVE_IP # 127.0.0.1
```

```

REDO_LOG_FILESEQ # 0
REDO_LOG_BLOCKSEQ # 52811
CAPTURE_FILESEQ # 0
CAPTURE_BLOCKSEQ # 52811
CAPTURE_INTERVAL # 0
CAPTURE_INTERVAL_SIZE # 0

```

- 위의 정보를 통해 CYCLONE MASTER와 SLAVE가 운영 중임을 확인할 수 있다.
- CYMON 실행 (Trace log에 모니터링 정보를 저장한다.)
 - CYMON을 실행한 후 master와 정상적으로 실행되었을 때부터 모니터링 table 뿐만 아니라 trace 로그에도 모니터링 정보가 저장된다.
 - 일반적으로 master DB 측의 오류로 인해 cyclone_monitor_info 테이블을 조회할 수 없을 때 사용한다. 이 경우, master DB의 정보가 되는 REDO_LOG_FILESEQ, REDO_LOG_BLOCKSEQ에 대한 정보는 포함하지 않는다.
 - Trace log의 위치는 \$GOLDILOCKS_DATA/trc/cymon.trc이다.

```
cymon --start --trace
```

- 모니터링 정보 확인 (--trace 옵션을 사용했을 경우)
 - Cyclone master가 정상적으로 실행되었을 때부터 모니터링 정보를 저장한다.

GROUP_NAME	TIME	MASTER_STATE	SLAVE_STATE	MASTER_PORT	SLAVE_IP
CAPTURE_FILESEQ	CAPTURE_BLOCKSEQ	TOTAL_TX_COUNT	CAPTURE_TX_COUNT	CAPTURE_COMMIT_LSN	
GROUP1	2016-11-02 15:43:03	READY	N/A	21102	null
0	0	0	0	0	
GROUP2	2016-11-02 15:43:03	READY	N/A	21103	null
0	0	0	0	0	

- 위의 정보를 통해 CYCLONE MASTER만 동작하고 있고 SLAVE는 대기 중임을 확인할 수 있다.

GROUP_NAME	TIME	MASTER_STATE	SLAVE_STATE	MASTER_PORT	SLAVE_IP
CAPTURE_FILESEQ	CAPTURE_BLOCKSEQ	TOTAL_TX_COUNT	CAPTURE_TX_COUNT	CAPTURE_COMMIT_LSN	
GROUP1	2016-11-02 15:43:11	RUNNING	RUNNING	21102	192.168.0.206
7	52779	0	0	15346	
GROUP2	2016-11-02 15:43:11	READY	N/A	21103	null
0	0	0	0	0	

- 위의 정보를 통해 CYCLONE MASTER (group1)와 SLAVE가 운영 중임을 확인할 수 있다.

- CYMON 운영상태 확인

```
$ cymon --status
```

```
=====
|          CYMON STATUS          |
=====
|    Service is running...      |
-----
```

- CYMON 종료

```
$ cymon --stop
```

```
stop done.
```


45.

LOGMIRROR

45.1 LOGMIRROR

LOGMIRROR는 GOLDILOCKS가 생성하는 redo log를 원격지에 복제하여 동일한 redo log file을 구성하는 redo log 이중화 툴이다.

개요

CDC 방식을 사용하는 CYCLONE은 GOLDILOCKS가 운영 중에 생성하는 redo log file을 분석하여 이중화한다. 따라서 redo log file을 분석하지 못한 상태에서 운영 서버에 장애가 발생할 경우, 분석하지 못한 redo log 만큼의 데이터는 이중화하지 못하고 이는 데이터 손실로 이어진다.

그러나 LOGMIRROR를 이용하여 손실없이 원격지에 redo log를 전송하고 redo log file을 구성할 수 있다면 CYCLONE의 데이터 손실 문제를 해결할 수 있다.

즉, LOGMIRROR는 CYCLONE이 ASYNC 방식을 사용함으로써 발생하는 데이터 손실을 완벽하게 방지하기 위한 툴이며 CYCLONE과 함께 동작해야 한다.

운영상 특징

- Master와 slave로 구분되어 수행되며 GOLDILOCKS 하나당 LOGMIRROR 한 개만 운영할 수 있다.
- Master와 slave는 TCP/IP와 Infiniband 통신을 사용할 수 있다.
 - Network 속도는 GOLDILOCKS 운영 속도에 많은 영향을 준다.
- 이중화하는 대상은 GOLDILOCKS의 redo log file이다.
- 원본 데이터베이스는 반드시 Transactional Data Store (TDS) 모드로 운영해야 한다.
 - CYCLONE을 함께 운영해야하므로 CYCLONE의 운영상 특징을 모두 동일하게 갖는다.

LOGMIRROR 운영시 GOLDILOCKS 성능저하 요인

LOGMIRROR는 GOLDILOCKS가 운영 중에 생성하는 redo log를 파일로 저장하기 전에 원격지로 보내고 정상적으로 처리된 이후에 다음 처리과정을 수행한다. 따라서 이중화하는 장비간의 네트워크 속도가 성능의 주요 요인이며 LOGMIRROR 없이 운영되는 GOLDILOCKS보다 성능이 저하될 수 있다. 이에, 네트워크 속도가 빠른 infiniband까지 지원함으로써 성능 저하를 최소화하였다.

45.2 준비사항

GOLDILOCKS 준비사항

LOGMIRROR를 수행하기 전에 GOLDILOCKS에 다음과 같은 사항이 설정되어 있어야 한다.

LOG_MIRROR_MODE

LOG_MIRROR_MODE 프로퍼티는 GOLDILOCKS에서 LOGMIRROR를 사용하기 위한 설정이다. 이 설정이 활성화 되면 GOLDILOCKS가 생성하는 redo log를 LOGMIRROR가 원격지로 전송하기 전에 임시로 저장할 자원이 할당된다.

- LOG_MIRROR_MODE 프로퍼티를 변경하면 GOLDILOCKS가 재시작된 후에 적용된다.
- 프로퍼티 파일에 해당 내용을 추가하거나 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: LOG_MIRROR_MODE = 1

또는 gSQL에서 다음을 수행한다.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_MODE=1 SCOPE=FILE;
System altered.
```

LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE

LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE는 LOGMIRROR에서 사용할 임시 저장 공간을 설정한다.

- Redo log buffer에서 redo log file로 저장하기 전에 사용되는 임시 저장 공간이다.
 - 설정값은 redo log file로 저장할 때 사용되는 MAXIMUM_FLUSH_LOG_BLOCK_COUNT와 관련있다.
- 기본값은 100 M 이다.
 - 최소값은 10 M이며, 최대값은 1 G이다.
 - 설정값이 너무 작을 경우 성능에 영향을 줄 수 있다.
- LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE 프로퍼티를 변경할 경우, GOLDILOCKS가 재시작된 후에 적용된다.
- 프로퍼티 파일에 해당 내용을 추가하거나 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE = 설정값

또는 gSQL에서 다음을 수행한다.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE = 200M SCOPE=FILE;
System altered.
```

LOG_MIRROR_TIMEOUT

GOLDILOCKS는 LOGMIRROR와 연동될 때 LOGMIRROR의 응답을 기다리는 과정을 포함한다. 이 응답이 너무 느리면 이 과정을 포함하는 GOLDILOCKS 또한 느려진다. LOG_MIRROR_TIMEOUT은 응답 대기시간을 지정하고 이 시간을 경과할 때까지 응답이 오지 않을 경우 LOGMIRROR 서비스를 중단한 채 GOLDILOCKS의 서비스만 계속 수행한다.

LOGMIRROR를 다시 수행하려면 LOGMIRROR를 재구동하여 recovery 과정을 거치면 정상적으로 다시 운영할 수 있다.

- 기본값은 0 이다.
 - 0은 무한대기를 나타낸다.
 - 초 단위로 설정할 수 있다
- LOG_MIRROR_TIMEOUT 프로퍼티를 변경할 경우, 그 즉시 반영된다.
- 프로퍼티 파일에 해당 내용을 추가하거나 갱신한다.
 - 프로퍼티 파일: goldilocks.properties.conf
 - 프로퍼티 설정: LOG_MIRROR_TIMEOUT = 설정값

또는 gSQL에서 다음을 수행한다.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_TIMEOUT = 20;
System altered.
```

노트

CYCLONE을 운영할 때 필요한 GOLDILOCKS 준비사항 역시 적용되어 있어야 한다.

45.3 환경설정

환경설정 파일

LOGMIRROR를 실행할 때 환경 설정 파일을 사용하여 운영에 필요한 정보와 옵션을 설정할 수 있다.

- --conf 옵션을 사용하여 특정 환경 설정 파일을 설정하지 않을 경우, \$GOLDILOCKS_DATA/conf 디렉토리에 서 특정 파일을 읽게되는데, master로 동작할 경우에는 logmirror.master.conf 파일을 읽고, slave로 동작할 경우에는 logmirror.slave.conf 파일을 읽는다.

표 45-1 설정 내용

이름	설명	적용범위
PORT	Master/ slave 통신에 사용할 port를 설정한다.	Master/ slave
DSN	Data Source Name을 설정한다.	Master
HOST_IP	GOLDILOCKS가 운영 중인 host IP address를 설정한다.	Master
HOST_PORT	GOLDILOCKS가 운영 중인 host port를 설정한다.	Master
USER_ID	사용자 이름을 설정한다.	Master
USER_PW	사용자 암호를 설정한다.	Master
USER_ENCRYPT_PW	암호화된 사용자 암호를 설정한다.	Master/ slave
LOG_PATH	이중화된 redo log file이 저장될 경로를 설정한다.	Slave
MASTER_IP	LogMirror master가 운영 중인 장비의 IP address를 설정한다.	Slave
HEARTBEAT_TIMEOUT	이중화 연결 후에 network가 끊기거나 시스템 장애에 의해 연결이 원활하지 않을 경우, 연결을 유지하는 최대 시간 (초)을 설정한다.	Master/ slave
TCP_NODELAY	Socket의 TCP_NODELAY 옵션을 설정한다. (Default는 1 이다) • 0: TCP_NODELAY off • 1: TCP_NODELAY on	Master

환경설정 옵션

PORT

- Master와 slave간에 통신할 때 사용되는 port를 설정한다.
- Master와 slave에서 설정할 수 있다.

PORT=21106

DSN

- GOLDDILOCKS에 접속할 때 필요한 Data Source Name을 설정한다.
- Master에서 설정할 수 있다.

DSN = GOLDDILOCKS

HOST_IP

- LOGMIRROR가 운영될 GOLDDILOCKS의 IP address를 설정한다.
- HOST_PORT와 함께 설정해야 한다.
- Master에서 설정할 수 있다.

HOST_IP = 127.0.0.1

HOST_PORT

- LOGMIRROR가 운영될 GOLDDILOCKS의 port를 설정한다.
- HOST_IP와 함께 설정해야 한다.
- Master에서 설정할 수 있다.

HOST_PORT = 22531

USER_ID

- GOLDDILOCKS 접속에 필요한 사용자 ID를 설정한다.
- Master에서 설정할 수 있다.

USER_ID = testID

USER_PW

- GOLDDILOCKS 접속에 필요한 사용자 암호를 설정한다.
- Master에서 설정할 수 있다.

USER_PW = testPW

USER_ENCRYPT_PW

- GOLDILOCKS 접속에 필요한 사용자 패스워드를 encrypt하여 설정한다.
- USER_PW를 대신하여 사용한다.
- Encrypt 된 사용자 패스워드는 [logmirror --encrypt 사용자패스워드 --key 암호화할key] 로 생성한다.
- 해당 설정값을 사용한 경우 logmirror를 실행할 때 --key 옵션을 사용해야 한다. (이 때 --encrypt로 생성한 key 와 동일한 key 값을 사용해야 한다.)

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

LOG_PATH

- 이중화되는 redo log file이 저장될 경로를 설정한다.
 - 절대 경로를 사용해야 한다.
 - 경로에는 single quote (')를 사용해야 한다.
- Slave에서 설정할 수 있다.

```
LOG_PATH = '/data/wal'
```

MASTER_IP

- LOGMIRROR master가 운영 중인 장비의 IP address를 설정한다.
- Slave에서 설정할 수 있다.

```
MASTER_IP = 192.168.0.100
```

HEARTBEAT_TIMEOUT

- Master와 slave에서 설정할 수 있다.
- Master와 slave의 이중화 연결 후에 network가 끊기거나 시스템 장애에 의해 연결이 원활하지 않을 경우, 연결 유지를 지속하는 최대 시간 (초)을 설정한다.
- 기본값은 30 (초)이다.
 - 최소값은 10 (초)이다.

```
HEARTBEAT_TIMEOUT = 40
```

TCP_NODELAY

- Master에서만 사용된다.
- LogMirror 전송 socket에 대한 TCP_NODELAY 옵션을 설정한다.
 - 0: Socket TCP_NODELAY 옵션을 off 한다.
 - 1: Socket TCP_NODELAY 옵션을 on 한다. (Default)

TCP_NODELAY = 1

45.4 운영

LOGMIRROR의 master/ slave 실행 환경은 다음과 같다.

표 45-2 실행 환경

구분	GOLDILOCKS 운영여부	설명
Master	O	Master로 운영할 때는 반드시 GOLDILOCKS가 운영되고 있는 장비에서 LogMirror를 수행해야 한다.
Slave	X	Slave로 운영할 때 GOLDILOCKS는 운영하지 않고 있어도 되지만 redo log file이 저장될 디스크 여유공간은 반드시 필요하다.

운영 중에 수행되는 내용은 trace log를 통해 확인할 수 있다.

구분	파 일
Master	\$GOLDILOCKS_DATA/trc/LogMirror_master.trc
Slave	\$GOLDILOCKS_DATA/trc/LogMirror_slave.trc

노트

Trace log에 저장되는 에러 메시지와 해결 방법은 **LOGMIRROR 에러 메시지와 해결 방법**을 참조한다.

LOGMIRROR 운영

LOGMIRROR는 GOLDILOCKS의 환경 설정 및 master/ slave 초기화와 운영 이후에도 다음 과정이 완료되어야만 정상적으로 redo log file을 이중화할 수 있다.

노트

이중화를 시작하려면 redo log file을 SWITCH 해야 한다. 즉, 새로운 로그 파일이 생성되어야만 LOGMIRROR의 정상적 운영이 시작된다. 따라서 다음 과정을 수행해야 한다.

```
gSQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

노트

LOGMIRROR가 이중화하는 redo log file은 지속적으로 저장된다. 즉, 자동으로 지워지지 않는다. 따라서 운영 장비의 환경에 따라 주기적으로 파일을 삭제하거나 이동하는 등의 관리 작업이 필요하다

실행 옵션

표 45-3 실행 옵션

옵션	설명	비고
--start -s	LOGMIRROR를 실행한다.	--master --slave 와 함께 사용해야 한다.
--stop -t	LOGMIRROR를 종료한다.	--master --slave 와 함께 사용해야 한다.
--master -m	Master 모드로 수행한다.	--start --stop 과 함께 사용해야 한다.
--slave -l	Slave 모드로 수행한다.	--start --stop 과 함께 사용해야 한다.
--conf -c	환경 파일의 경로를 설정한다.	--conf CONFIG_FILE 형식으로 입력한다.
--infiniband -f	infiniband 네트워크 환경을 사용한다.	TCP/ IP 환경을 사용할 경우, 입력하지 않는다.
--silent -i	메시지를 출력하지 않도록 한다.	-
--help -h	도움말을 출력한다.	-

- 기본 환경 파일을 사용하여 master 모드로 실행한다.

```
prompt> logmirror --master --start
```

- Master 모드로 운영 중인 LOGMIRROR를 종료한다.

```
prompt> logmirror --master --stop
```

- 기본 환경 파일을 사용하여 slave 모드로 실행한다.

```
prompt> logmirror --slave --start
```

- Slave 모드로 운영 중인 LOGMIRROR를 종료한다.

```
prompt> logmirror --slave --stop
```

- TEST_CONFIG 파일, infiniband network를 사용하여 master 모드로 실행한다.

```
prompt> logmirror --master --start --conf TEST_CONFIG --infiniband
```

- TEST_CONFIG 파일, infiniband network를 사용하여 slave 모드로 실행한다.

```
prompt> logmirror --slave --start --conf TEST_CONFIG --infiniband
```

노트

LOGMIRROR를 초기화하는 예는 **LOGMIRROR**를 참조한다.

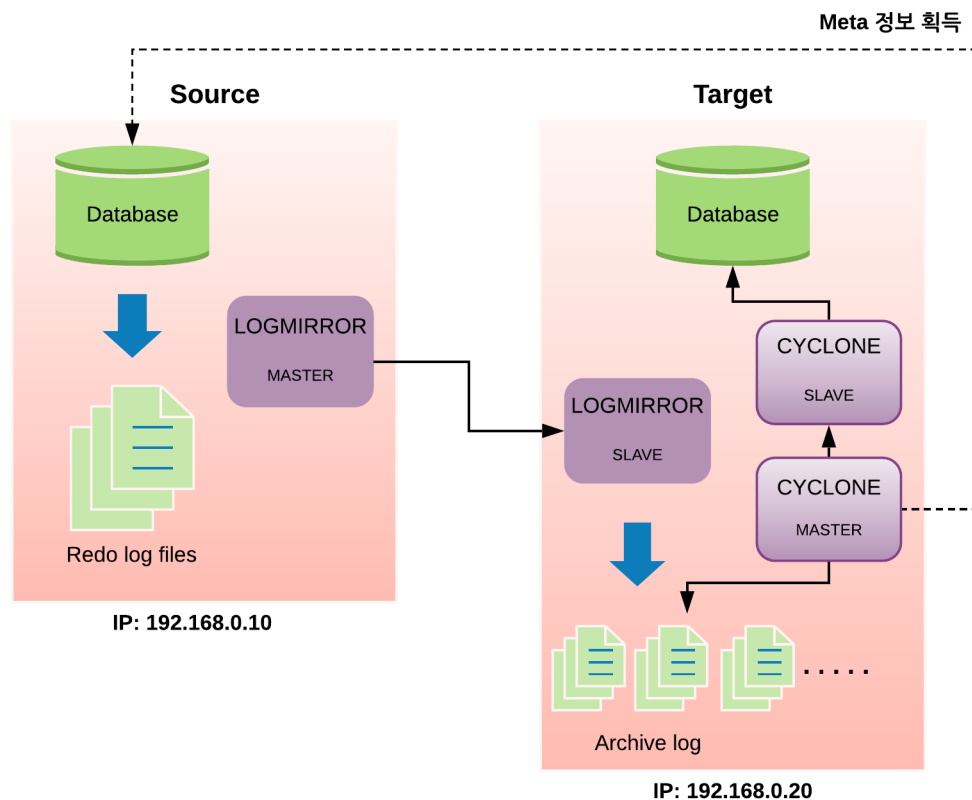
45.5 CYCLONE과 연동 예

CDC 이중화 툴인 CYCLONE을 redo log file 이중화 툴인 LOGMIRROR와 연동하여 사용할 경우 이중화 데이터 손실을 방지할 수 있다.

TOOL	기능	연동
CYCLONE	CDC 이중화	독립적으로 운영할 경우 데이터가 손실될 수 있다.
LOGMIRROR	REDO LOG FILE 이중화	연동하여 운영할 경우 데이터가 손실되지 않는다.

다음은 연동을 위한 운영 구조의 예이다.

그림 1 운영 구조도 예제



- 장비 구조
 - 데이터의 원본이 되는 source 장비
 - IP: 192.168.0.10
 - GOLDILOCKS listen port: 22581
 - 이중화할 테이블: T1, T2
 - 이중화를 위한 원격 target 장비
 - IP: 192.168.0.20
- 두 개의 장비는 데이터 손실 없는 이중화를 위해 LOGMIRROR와 CYCLONE을 연동하여 운영한다.

- Source 장비: LOGMIRROR (master)
- Target 장비: LOGMIRROR (slave) + CYCLONE (master, slave)

운영 순서

1. 원본 GOLDDILOCKS에 CYCLONE 및 LOGMIRROR 환경 설정
2. LOGMIRROR MASTER/ SLAVE 환경 설정
3. LOGMIRROR MASTER/ SLAVE 실행
4. LOGMIRROR를 정상적으로 운영하기 위해 원본 GOLDDILOCKS의 LOGFILE SWITCH를 수행
5. 원격 GOLDDILOCKS 환경 설정
6. CYCLONE MASTER/ SLAVE 환경 설정
7. CYCLONE MASTER/ SLAVE 실행

원본 GOLDDILOCKS에 CYCLONE 및 LOGMIRROR 환경 설정

다음을 참조한다.

- CYCLONE 환경설정: [준비사항](#)
- LOGMIRROR 환경설정: [준비사항](#)

환경 설정을 마친 후, 테스트를 위해 T1, T2 table을 생성한다.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

LOGMIRROR MASTER/ SLAVE 환경 설정

- LOGMIRROR MASTER 환경 설정 파일
 - 저장위치: \$GOLDDILOCKS_DATA/conf/logmirror.master.conf
 - Source 장비이다.

노트

LOGMIRROR MASTER는 반드시 원본 GOLDDILOCKS가 운영되는 장비에서 실행되어야 한다.

- DSN 또는 HOST 정보를 입력한다. 여기서는 HOST 정보를 입력한다.

```
HOST_IP = 192.168.0.10
```

```
HOST_PORT = 22581
```

- 위 단계에서 생성한 CYCYLONE USER를 사용한다.

```
USER_ID = cdc_user
```

```
USER_PW = cdc_password
```

- LOGMIRROR 통신에 사용할 PORT를 설정한다.

```
PORT = 21106
```

- LOGMIRROR SLAVE 환경 설정 파일
 - 저장위치: \$GOLDILOCKS_DATA/conf/logmirror.slave.conf
 - Target 장비이다.
- LOGMIRROR MASTER가 운영되고 있는 장비의 IP를 입력한다.

```
MASTER_IP = 192.168.0.10
```

- LOGMIRROR MASTER와 동일한 PORT를 입력해야 한다.

```
PORT = 21106
```

- 이중화된 redo log file이 저장될 절대 경로를 입력한다.

```
LOG_PATH = '/data/LogMirrorWAL'
```

LOGMIRROR MASTER/ SLAVE 실행

- LOGMIRROR MASTER 실행
 - Source 장비에서 실행해야 한다.

```
logmirror --master --start --conf $GOLDILOCKS_DATA/conf/logmirror.master.conf
```

- LOGMIRROR SLAVE 실행
 - Target 장비에서 실행해야 한다.

```
logmirror --slave --start --conf $GOLDILOCKS_DATA/conf/logmirror.slave.conf
```


정상적인 LOGMIRROR 운영을 위해 원본 GOLDDILOCKS의 LOGFILE SWITCH 수행

- LOGFILE SWITCH를 수행하여 LOGMIRROR 파일을 기록하기 시작한다.
 - Source 장비에서 실행해야 한다.

```
gSQL> ALTER SYSTEM SWITCH LOGFILE;
```

노트

위와 같이 실행할 경우 SLAVE 장비의 설정된 경로에 redo log file이 생성된다. 만약 redo log file이 생성되지 않았으면 디렉토리 권한이나 디렉토리 생성 여부를 확인해봐야 한다.

원격 GOLDDILOCKS 환경 설정

Target 장비에서 운영되는 GOLDDILOCKS의 정상적인 운영 및 이중화를 위한 테이블 구성, 스키마 정보를 확인한다.

여기에서는 테스트를 위해 T1, T2 table을 생성한다.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

CYCLONE MASTER/ SLAVE 환경 설정

- CYCLONE MASTER 환경 설정 파일
 - 저장위치: \$GOLDDILOCKS_DATA/conf/cyclone.master.conf
 - Target 장비이다.
- Source 장비의 DSN 또는 HOST 정보를 입력한다. 여기서는 HOST 정보를 입력한다.

```
HOST_IP    = 192.168.0.10
HOST_PORT  = 22581
```

- CYCLONE USER를 사용한다.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = GROUP1
{
```

```

    PORT = 21102
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}

```

- CYCLONE SLAVE 환경 설정 파일
 - 저장위치: \$GOLDILOCKS_DATA/conf/cyclone.slave.conf
 - Target 장비이다.
- Target 장비에서 운영되는 GOLDILOCKS에 D/A로 접속한다. (따라서 DSN, HOST 정보는 없다)

```

USER_ID = cdc_user
USER_PW = cdc_password

```

- CYCLONE master는 동일한 장비에서 수행되고 있다.

```

MASTER_IP = 192.168.0.20
GROUP_NAME = GROUP1
{

```

- LogMirror slave가 설정한 경로를 입력해야 한다.

```

LOG_PATH = '/data/LogMirrorWAL'
    PORT = 21102
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}

```

CYCLONE MASTER/ SLAVE 실행

- CYCLONE MASTER 실행
 - D/A로 접속하는 경우 target 장비에서 실행해야 한다.

```

cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf

```

- CYCLONE SLAVE 실행

- D/A로 접속하는 경우 target 장비에서 실행해야 한다.

```
| cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf
```


부록 A.

Error Codes

GOLDILOCKS에서 제공하는 에러 code는 다음과 같다. 해당 내용은 **V\$ERROR_CODE** view로 조회할 수 있다.

A.1 OS 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-1 OS 관련 에러

Error code	SQLSTATE	Message
11000	HY000	Invalid argument
11001	HY000	The operation was incomplete although some processing was performed and the results are partially valid
11002	HY000	Permission denied '%s'
11003	HY000	Standard Layer has encountered the end of the file
11004	HY000	Unable to duplicate a file
11005	HY000	File Lock Error
11006	HY000	File Unlock Error
11007	HY000	Unable to open the file '%s'
11008	HY000	Unable to close the file
11009	HY000	Unable to remove the file
11010	HY000	Unable to link the file
11011	HY000	Unable to read the file
11012	HY000	Unable to write the file
11013	HY000	Unable to sync the file
11014	HY000	Unable to seek the file
11015	HY000	Unable to truncate the file
11016	HY000	Unable to get the status of the file
11017	HY000	Unable to create a pipe
11018	HY000	Unable to wait for the given process
11019	HY000	Insufficient resource
11020	HY000	Unable to send signal to given process
11021	HY000	Unable to set a signal handler
11022	HY000	Unable to block a signal
11023	HY000	Unable to set thread attribute
11024	HY000	Unable to create a thread
11025	HY000	Unable to join with a terminated thread
11026	HY000	Unable to set thread once code
11027	HY000	Truncation occurred on I/O operation
11028	HY000	The given name(%s) is too long
11029	HY000	shared memory segment exists
11030	HY000	Unable to allocate a shared memory segment

Error code	SQLSTATE	Message
11031	HY000	Unable to attach the shared memory segment
11032	HY000	Unable to detach the shared memory segment
11033	HY000	Unable to destroy the shared memory segment
11034	HY000	Unable to create a semaphore
11035	HY000	Unable to acquire the given semaphore
11036	HY000	Unable to release the given semaphore
11037	HY000	Unable to destroy the given semaphore
11038	HY000	The given object was busy
11039	HY000	Operation timed out
11040	HY000	No such object (%s)
11041	HY000	Two objects do not match
11042	HY000	Not enough memory
11043	HY000	File system is full
11044	HY000	Unable to execute the program
11045	HY000	not supported function
11046	HY000	given buffer does not have enough space
11047	HY000	invalid network address
11048	HY000	sendto() system call returned failure
11049	HY000	recvfrom() system call returned failure
11050	HY000	fail to get address from socket with getsockname()
11051	HY000	fail to get peer address from socket with getpeername()
11052	HY000	fail to manipulate file descriptor with fcntl()
11053	HY000	fail to manipulate socket descriptor with setsockopt()
11054	HY000	not implemented feature
11055	HY000	fail to manipulate device parameters of special file with ioctl()
11056	HY000	fail to get hostname
11057	HY000	fail to close a socket
11058	HY000	fail to shutdown a socket
11059	HY000	fail to create a socket
11060	HY000	fail to get property of a file descriptor
11061	HY000	fail to set property of a file descriptor
11062	HY000	fail to bind a socket to address
11063	HY000	fail to listen on a socket
11064	HY000	fail to accept a connection request with a socket
11065	HY000	system call was interrupted by a signal
11066	HY000	fail to get options of a socket
11067	HY000	fail to connect to an host with a socket
11068	HY000	fail to wait for some file descriptors with poll()
11069	HY000	internal error
11070	HY000	sendmsg() system call returned failure

Error code	SQLSTATE	Message
11071	HY000	recvmsg() system call returned failure
11072	HY000	socketpair() system call returned failure
11073	HY000	converted value is out of range (overflow or underflow)
11074	HY000	given string is not number
11075	HY000	not a valid file descriptor
11076	HY000	resource temporarily unavailable
11077	HY000	given address is already in use
11078	HY000	fail to initialize communication context
11079	HY000	fail to finalize communication context
11080	HY000	fail to connect communication context
11081	HY000	fail to poll communication context
11082	HY000	fail to write packet through communication context
11083	HY000	fail to read packet through communication context
11084	HY000	fail to send descriptor through communication context
11085	HY000	fail to receive descriptor through communication context
11086	HY000	fail to get current file path
11087	HY000	fail to open library (%s)
11088	HY000	fail to close library
11089	HY000	fail to get symbol address (%s)
11090	HY000	not supported query
11091	HY000	fail to check file (%s)
11092	HY000	input is too long
11093	HY000	system call error (function : %s, error no : %d)
11094	HY000	divide zero
11095	HY000	Unable to open a semaphore
11096	HY000	Unable to close the given semaphore
11097	HY000	Unable to unlink the given semaphore
11098	HY000	permission denied - '%s'
11099	HY000	not supported os
11100	HY000	Environment Variable "%s" is not defined.
11101	HY000	License file(%s) is not exist
11102	HY000	License is out of date. License is valid after %4d-%02d-%02d
11103	HY000	License date is expired. License is expired in %4d-%02d-%02d
11104	HY000	License core count mismatch.
11105	HY000	License key is corrupted.
11106	HY000	License host name mismatch.
11107	HY000	failed to set thread affinity.
11108	HY000	failed to create timer.
11109	HY000	failed to set timer.
11110	HY000	failed to destroy timer.

Error code	SQLSTATE	Message
11111	HY000	address is not given.
11112	HY000	Unable to access the message queue.
11113	HY000	A message queue exists.
11114	HY000	Unable to create a message queue.
11115	HY000	The message queue was removed.
11116	HY000	Unable to destroy the message queue.
11117	HY000	Unable to send message.
11118	HY000	Unable to receive message.
11119	HY000	No message was available in the message queue.
11120	HY000	fail to get property of a file status
11121	HY000	fail to set property of a file status
11122	HY000	infiniband ibv/rdma interface returned error
11123	HY000	no infiniband device was found
11124	HY000	cannot find specified infiniband device
11125	HY000	cannot find suitable Memory Region for specified ptr and size
11126	HY000	fail to get semaphore value
11127	HY000	fail to control pollset
11128	HY000	fail to create pollset
11129	HY000	overflow fd
11130	HY000	invalid INI file format
11131	HY000	path '%s' does not exist
11132	HY000	Unable to get maximum undo semaphores per array
11133	HY000	Unable to create a undo semaphore
11134	HY000	Unable to bind the given undo semaphore
11135	HY000	Unable to unbind the given undo semaphore
11136	HY000	Unable to get the given undo semaphore state
11137	HY000	Unable to destroy the given undo semaphore
11138	HY000	Unable to suspend asynchronous io
11139	HY000	Unable to control the shared memory segment
11140	HY000	Unable to detach a thread
11141	O8S01	Communication link failure
11142	HY000	No such process

A.2 타입과 연산 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-2 타입과 연산 관련 에러

Error code	SQLSTATE	Message
12000	HY000	failed to initialize datatype layer
12001	HYC00	not implemented feature, in a function %s
12002	22001	byte length of data greater than column length
12003	22001	character length of data greater than column length
12004	22001	data converted with truncation of fractional digits
12005	22001	conversion of data would result in loss of whole digits
12006	22018	data value is not a numeric literal
12007	22003	data is outside the range of the data type to which the number is being converted
12008	22001	data is greater than 0, less than 2, and not equal to 1
12009	22003	data is less than 0 or greater than or equal to 2
12010	22001	(byte length of data)/2 is greater than column byte length
12011	22003	data value is not a hexadecimal value
12012	22008	data value is a valid timestamp literal; time portion is nonzero
12013	22018	data value is not a valid date literal or timestamp literal
12014	22008	data value is a valid timestamp literal; fractional seconds portion is nonzero
12015	22018	data value is not a valid time literal or timestamp literal
12016	22008	data value is a valid timestamp literal; fractional seconds portion truncated
12017	22018	data value is not a valid date literal or time literal or timestamp literal
12018	22015	data value is a valid interval value; the value in one of the fields is truncated
12019	22018	data value is not a valid interval literal
12020	22001	number of digits greater than column byte length
12021	22001	number of characters greater than column byte length
12022	22003	data converted with truncation of whole digits
12023	22003	data is outside the range of the data type to which the number is being converted
12024	22001	data is greater than 0, less than 2, and not equal to 1
12025	22003	data is less than 0 or greater than or equal to 2
12026	22015	interval data truncated
12027	22001	byte length of data greater than column length
12028	22001	character length of data greater than column length
12029	22003	byte length of data not equal to SQL data length

Error code	SQLSTATE	Message
12030	22001	column byte length is less than %d
12031	22008	data value is not a valid date
12032	22007	data value is not a valid date format
12033	22001	column byte length is less than %d
12034	22018	data value is not a valid GUID
12035	22001	column byte length is less than %d
12036	22001	data value is not a valid time
12037	22007	data value is not a valid time format
12038	22001	column byte length is between %d and character byte length
12039	22001	column byte length is less than %d
12040	22008	data value is not a valid timestamp
12041	22008	time fields are nonzero
12042	22007	data value does not contain a valid date
12043	22008	fractional seconds fields are nonzero
12044	22007	data value does not contain a valid time
12045	22008	fractional seconds fields are truncated
12046	22007	data value does not contain a valid timestamp
12047	22001	column byte length is less than character byte length
12048	22015	data value is not a valid interval literal
12049	22003	conversion resulted in truncation of whole digits
12050	22015	one or more fields of data value were truncated during conversion
12051	22001	column byte length is less than character byte length
12052	22015	data value is not a valid interval literal
12053	22003	conversion resulted in truncation of whole digits
12054	22015	one or more fields of data value were truncated during conversion
12055	01004	byte length of data greater equal than buffer length
12056	01004	character length of data greater equal than buffer length
12057	01S07	data converted with truncation of fractional digits
12058	22003	conversion of data would result in loss of whole digits
12059	22018	data is not numeric literal
12060	22003	data is outside the range of the data type to which the number is being converted
12061	01S07	data is greater than 0, less than 2, and not equal to 1
12062	22003	data is less than 0 or greater than or equal to 2
12063	01S07	data value is a valid timestamp-value; time portion is nonzero
12064	22018	data value is not a valid date-value or timestamp-value
12065	01S07	data value is a valid timestamp-value; fractional seconds portion is nonzero
12066	22018	data value is not a valid time-value or timestamp-value
12067	22018	data value is not a valid date-value or time-value or timestamp-value
12068	01S07	data value is a valid interval value; truncation of one or more trailing fields

Error code	SQLSTATE	Message
12069	22015	data value is a valid interval; leading field significant precision is lost
12070	22018	data value is not a valid interval value
12071	01004	number of whole digits is less than buffer length
12072	22003	number of whole digits is greater than buffer length
12073	01S07	data converted with truncation of fractional digits
12074	22003	conversion of data would result in loss of whole digits
12075	22003	data is outside the range of the data type to which the number is being converted
12076	01S07	data is greater than 0, less than 2, and not equal to 1
12077	22003	data is less than 0 or greater than or equal to 2
12078	22003	byte length of data is greater than buffer length
12079	01S07	fractional seconds portion truncated
12080	22015	whole part of number truncated
12081	22003	buffer length is less equal than 1
12082	22003	buffer length is less than 1
12083	01004	(byte length of data)/2 is greater equal than buffer length
12084	01004	(character length of data)/2 is greater equal than buffer length
12085	01004	byte length of data is greater than buffer length
12086	01004	buffer length is between %d and character byte length
12087	22003	buffer length is less than %d
12088	22003	byte length of data is greater than buffer length
12089	22003	buffer length is less than %d
12090	22003	byte length of data is greater than buffer length
12091	01004	buffer length is between %d and character byte length
12092	22003	buffer length is less than %d
12093	22003	byte length of data is greater than buffer length
12094	01004	buffer length is between %d and character byte length
12095	22003	buffer length is less than %d
12096	01004	buffer length is between %d and character length
12097	22003	byte length of data is greater than buffer length
12098	01S07	time portion of timestamp is nonzero
12099	01S07	fractional seconds portion of timestamp is nonzero
12100	01S07	fractional seconds portion of timestamp is truncated
12101	01S07	trailing fields portion truncated
12102	22015	leading precision of target is not big enough to hold data from source
12103	22003	interval precision was a single field and truncated whole
12104	22015	interval precision was not a single field
12105	22003	byte length of data is greater than buffer length
12106	01004	number of whole digits is less than buffer length
12107	22003	number of whole digits is greater equal than buffer length

Error code	SQLSTATE	Message
12108	01S07	trailing fields portion truncated
12109	22015	leading precision of target is not big enough to hold data from source
12110	01S07	interval precision was a single field and truncated fractional
12111	22003	interval precision was a single field and truncated whole
12112	07006	interval precision was not a single field
12113	22003	byte length of data is greater than buffer length
12114	01004	number of whole digits is less than buffer length
12115	22003	number of whole digits is greater equal than buffer length
12116	22009	invalid time zone displacement value
12117	42000	invalid number of arguments
12118	42000	inconsistent datatype of function parameters: %s
12119	42000	comparison is not applicable: %s and %s
12120	42000	conversion is not applicable: from %s to %s
12121	22021	invalid character value in character set repertoire
12122	22012	divisor is equal to zero
12123	22018	data is not boolean literal
12124	HY000	invalid argument for factorial function
12125	22027	trim set should have only one character
12126	22008	datetime field overflow
12127	HY000	invalid ROWID
12128	22019	invalid escape character
12129	HY000	invalid character set identifier
12130	HY000	invalid extract field for extract source
12131	42000	inconsistent datatypes: argument[%d] expected %s
12132	HY000	date format value exceeds maximum length (%d)
12133	HY000	date format not recognized
12134	HY000	format code appears twice
12135	HY000	%s conflicts with %s
12136	HY000	literal does not match format string
12137	HY000	input value not long enough for date format
12138	HY000	date format picture ends before converting entire input string
12139	HY000	year must be between %s and %s, and not be 0
12140	2200C	missing or illegal character following the escape character
12141	HY000	not a valid month
12142	HY000	a non-numeric character was found where a numeric was expected
12143	HY000	day of month must be between 1 and last day of month
12144	HY000	not a valid day of the week
12145	HY000	day of year must be between 1 and 365 (366 for leap year)
12146	HY000	format code cannot appear in date input format
12147	HY000	BC/B.C. or AD/A.D. required

Error code	SQLSTATE	Message
12148	HY000	julian date must be between 0 and 5373484
12149	HY000	%s may only be specified once
12150	HY000	Julian date precludes use of day of year
12151	HY000	Not enough format info buffer
12152	HY000	hour must be between %d and %d
12153	HY000	minute must be between 0 and 59
12154	HY000	second must be between 0 and 59
12155	HY000	the fractional seconds must be between 0 and 999999
12156	HY000	time zone hour must be between -14 and 14
12157	HY000	time zone minute must be between 0 and 59
12158	HY000	seconds in day must be between 0 and 86399
12159	HY000	AM/A.M. or PM/P.M. required
12160	HY000	HH24 precludes use of meridian indicator
12161	HY000	%s conflicts with use of %s
12162	HY000	invalid XID string
12163	HY000	date not valid for month specified
12164	HY000	signed year precludes use of BC/AD
12165	HY000	number format value exceeds maximum length (%d)
12166	HY000	invalid number format model
12167	HY000	cannot use %s twice
12168	HY000	cannot use %s and %s together
12169	22001	string data, right truncation
12170	HY000	argument '%d' is out of range
12171	HY000	invalid datepart for data type %s
12172	42000	maximum number of arguments exceeded
12173	42000	invalid char length units(%s): use OCTETS or CHARACTERS
12174	22023	The argument of %s function is invalid
12175	22023	The argument [%d] of %s function is invalid
12176	HY000	internal error, in a function %s
12177	HY000	not supported data type (%s)
12178	HY000	data type specifier is out of range (type : %d, range : %d to %d)
12179	HY000	precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12180	HY000	leading precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12181	HY000	scale specifier is out of range(type : %s, scale : %d, range : %d to %d)
12182	HY000	fractional second precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12183	HY000	character length unit specifier is out of range(type : %s, character length unit : %d, range : %d to %d)

Error code	SQLSTATE	Message
12184	HY000	interval indicator specifier is out of range(type : %s, indicator : %d, range : %d to %d)
12185	42000	%s and %s cannot be matched
12186	0A000	not supported function %s

A.3 자원 관리 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-3 자원 관리 관련 에러

Error code	SQLSTATE	Message
13000	HY000	Failed to initialize kernel layer
13001	HY000	kernel layer is not initialized yet
13002	HY000	Invalid argument
13003	HY000	Invalid shared memory segment type
13004	HY000	static shared memory segment is corrupted
13005	HY000	maximum number of shared memory segment num exceeded
13006	HY000	kernel layer internal error - %s
13007	42000	identifier is too long
13008	HY000	comment is too long
13009	42S02	fixed table does not exist
13010	HY000	Insufficient static area
13011	HY000	Unable to extend memory: [MAX: %ld, TOTAL: %ld, ALLOC: %ld] DESC: %s
13012	HY000	too many files are opened
13013	HY000	Property(%s) does not exist
13014	HY000	Unable to change property at this startup phase.(%s)
13015	0U000	Unable to change read-only property.(%s)
13016	22000	Invalid property value.(%s)
13017	0U000	Specified property cannot be modified.(%s)
13018	22000	Specified property cannot be modified with this SCOPE option.(%s)
13019	22000	Specified property cannot be modified.(%s)
13020	22000	Invalid GOLDILOCKS_DATA Directory
13021	22000	Invalid SID
13022	HY004	Invalid property data type
13023	22000	Invalid Property ID
13024	HY000	exceeded maximum number of processes
13025	HY000	File does not exist (%s)
13026	HY000	File exist (%s)
13027	HY000	File Process Error
13028	HY000	Property is not ReadOnly
13029	HY000	ReadOnly Property Already Setted
13030	HY000	Property File Parsing Error Line (%d)
13031	0A000	Unsupported return type is used

Error code	SQLSTATE	Message
13032	0A000	Function does not exist
13033	HY000	It can not make range
13034	RD000	Service is not available
13035	HY000	Specified Breakpoint name is not registered yet
13036	HY000	General abort error of Breakpoint
13037	HY000	Invalid Session Name
13038	HYT00	Exceeded maximum idle time
13039	HYT00	Exceeded maximum query time
13040	08S01	Communication link failure
13041	RD000	User session ID does not exist
13042	08004	Server rejected the connection
13043	HY008	operation canceled
13044	RD000	invalid process identifier
13045	08004	mismatched binary version - server(%s.%d.%d.%d.%d), client(%s.%d.%d.%d.%d)
13046	HY000	session cannot be killed now
13047	22003	data is outside the range of the data type to which the number is being converted
13048	22015	one or more fields of data value were truncated during conversion
13049	HYC00	not implemented feature, in a function
13050	HY000	insufficient plan cache memory
13051	HY000	Unable to attach the shared memory segment. File does not exist(%s)
13052	42000	invalid dump option string: %s
13053	42000	dump object modified by concurrent statement
13054	42000	invalid numa map (%s)
13055	HY000	AS ADMIN must be executed in dedicated session
13056	HY000	unable to create file '%s' - already in use
13057	HY000	Property(%s) is desupported
13058	HY000	memory alloc size is too large: [MAX: %ld, ALLOC: %ld] DESC: (%d)%s
13059	HYT00	Exceeded maximum packet allocation time

A.4 저장 관리 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-4 저장 관리 관련 에러

Error code	SQLSTATE	Message
14000	HY000	Failed to initialize storage manager layer
14001	HY000	storage manager layer is not initialized yet
14002	HY000	Tablespace(%d) exceed datafile limit
14003	HY000	Tablespace(%d) is full
14004	HY000	Tablespace(%d) is already dropped
14005	HY000	System Tablespace is exists
14006	42000	Table is mutating, trigger/function may not see it
14007	RD000	deadlock detected
14008	24000	fetch out of sequence
14009	2200H	sequence generator limit exceeded
14010	22023	MAXVALUE cannot be made to be less than the current value
14011	22023	MINVALUE cannot be made to exceed the current value
14012	22023	cannot be made to be less than the current value
14013	22023	cannot be made to exceed the current value
14014	HY000	Fail to extend datafile(%u) of tablespace(%u) in Transaction(%ld)
14015	HY000	there is no extendible datafile in tablespace '%s'
14016	23001	some rows of base table violate uniqueness of index
14017	42000	invalid operation on fixed table or dump table
14018	42000	the operation is disallowed before %s startup phase
14019	HY000	log stream number string is invalid
14020	HY000	log stream %d already exists
14021	HY000	log group number string is invalid
14022	HY000	log group state must be INACTIVE or UNUSED
14023	HY000	log group %d already exists
14024	HY000	log member %s already exists
14025	42000	invalid dump option string: %s
14026	HYT00	resource busy or timeout expired
14027	HY000	datafile(%d,%d) is already dropped
14028	HY000	file '%s' is already exist
14029	HY000	sequence is not yet defined
14030	RD000	maximum index depth(%d) exceeded
14031	HY000	maximum key length(%d) exceeds

Error code	SQLSTATE	Message
14032	42000	can't serialize access for this transaction
14033	42000	unable to checkpoint in cds mode
14034	42000	unable to recover database in cds mode
14035	42000	some rows of base table violate not null constraint
14036	42000	invalid ROWID
14037	42000	Database is READ ONLY
14038	42000	unable to recover database in READ ONLY mode
14039	HY000	exceed segment MAXSIZE(%ld)
14040	HY000	Target object of dump table is mutating
14041	42000	cannot access the OFFLINE tablespace '%s'
14042	42000	cannot find the row (%ld)
14043	42000	file is already used in database '%s'
14044	42000	datafile not empty
14045	42000	cannot drop datafile of OFFLINE tablespace '%s'
14046	42000	file does not exist - '%s'
14047	42000	file is READ ONLY - '%s'
14048	42000	file is invalid or old version - '%s'
14049	42000	tablespace is not online - '%s'
14050	42000	tablespace is not offline - '%s'
14051	42000	media recovery required - '%s'
14052	42000	segment is unusable (physical id : %ld)
14053	RD000	insufficient resource for sort
14054	HY000	internal error (%s, %d)
14055	HY000	segment is corrupted (physical id : %ld)
14056	HY000	failed to apply redo log (lsn : %ld, lpsn : %d, physical id : %ld)
14057	HY000	dropping log group makes total log group count less than minimum log group count(%d)
14058	42000	log member is not exist - '%s'
14059	42000	log member is already used - '%s'
14060	HY000	CURRENT logfile is empty - '%s'
14061	HY000	next logfile to switch should be INACTIVE or UNUSED - '%s'
14062	42000	INITRANS must be less equal than MAXTRANS
14063	42000	cannot be added log member to exceed maximum log member count(%d)
14064	42000	the operation is disallowed at %s phase
14065	RD000	the instant table row length is too large
14066	RD000	the sort instant table key row length is too large
14067	HY000	cannot media recover; archive log mode not enabled
14068	HY000	logfile does not exist - '%s'
14069	HY000	cannot support ARCHIVELOG in cds mode
		control file is not recent; do media recovery with USING BACKUP CONTROLFILE

Error code	SQLSTATE	Message
14070	HY000	option
14071	HY000	cannot BACKUP; tablespace(%s) is offline
14072	HY000	cannot BACKUP; tablespace(%s) is temporary
14073	HY000	cannot BACKUP; tablespace is dropped
14074	HY000	cannot BACKUP; tablespace(%s) is already in backup
14075	HY000	cannot execute; tablespace(%s) is proceeding backup
14076	HY000	cannot END BACKUP; tablespace(%s) is not in backup
14077	HY000	cannot END BACKUP; none of the tablespaces are in backup
14078	HY000	cannot BACKUP; tablespace(%s) is proceeding other operation
14079	HY000	cannot BACKUP; database is already in backup
14080	HY000	cannot END BACKUP; database is not in backup
14081	HY000	cannot BACKUP; database is going shutdown
14082	HY000	(%s) is not an appropriate logfile to recover
14083	HY000	must use RESETLOGS option for database open
14084	HY000	must use NORESETLOGS option for database open
14085	HY000	(%s) was not a sufficiently old backup
14086	HY000	media recovery canceled
14087	HY000	cannot SHUTDOWN; backup in progress
14088	HY000	control file is corrupted
14089	HY000	LEVEL 0 INCREMENTAL BACKUP does not exist
14090	HY000	cannot flush logs; disabled log flushing
14091	HY000	(%s) is not valid logfile
14092	HY000	not exist incremental backup
14093	HY000	not exist valid incremental backup for tablespace(%s)
14094	HY000	datafile recovery required - datafile(%s) of tablespace(%s) corrupted
14095	HY000	backup file(%s) is corrupted
14096	23001	fail to build index (%ld)
14097	HY000	control file is corrupted - '%s'
14098	HY000	maximum record length(%d) exceeds
14099	HY000	log member state must be UNUSED or INACTIVE
14100	HY000	dropping log member makes log member count of group(%d) less than minimum log member count(%d)
14101	HY000	log group id does not exist - '%d'
14102	HY000	cannot execute; recovery is in progress
14103	01000	Warning: media recovery needs a logfile including log (Lsn %ld)
14104	01000	Warning: suggestion '%s'
14105	HY000	control file '%s' is inconsistent with primary
14106	HY000	tablespace (%s) is taken offline as the result of a write error
14107	HY000	given LOG_BLOCK_SIZE(%d) is not a suitable value for a size of log block; LOG_BLOCK_SIZE should be one of 512, 1024, 2048 or 4096.

Error code	SQLSTATE	Message
14108	HY000	cannot disable archivelog - exist tablespace needed media recovery.
14109	HY000	OFFLINE NORMAL disallowed; cannot be taken offline consistently.
14110	HY000	OFFLINE IMMEDIATE disallowed; noarchivelog mode
14111	HY000	cannot create COLUMNAR TABLE; given MIN ROW COUNT is too big to put into one page
14112	HY000	record size is too large for columnar table
14113	HY000	datafile '%s' is corrupted
14114	HY000	datafile '%s' is more recent than redo logfile
14115	HY000	transaction undo limit exceeded
14116	HY000	failed to journal log
14117	HY000	cannot end incomplete recovery; incomplete recovery never been begun
14118	HY000	property TRANSACTION_TABLE_SIZE value must be equal to or greater than '%d'
14119	HY000	property UNDO_RELATION_COUNT value must be equal to or greater than '%ld'
14120	HY000	sequence is dropped
14121	HY000	file '%s' does not match with database - file signature '%s', database signature '%s'
14122	HY000	there is a possibility that a deadlock occurs
14123	22023	INCREMENTBY sign cannot be changed in the cluster system
14124	HY000	RESTORE in progress
14125	HY000	exceed maximum journal file size
14126	HY000	there must be at least one recoverable member in the group
14127	HY000	object no longer exists
14128	HY000	invalid remote view scn
14129	HYT00	transaction allocation time exceeded
14130	HYT00	undo relation allocation time exceeded

A.5 딕셔너리 캐시 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-5 딕셔너리 캐시 관련 에러

Error code	SQLSTATE	Message
15000	HY000	failed to initialize execution library layer
15001	HY000	execution library layer is not initialized yet
15002	42000	%s: schema does not exist
15003	42000	%s: table or view does not exist
15004	42000	%s: index does not exist
15005	42000	%s: column does not exist
15006	23000	%s: dictionary integrity constraint violation by concurrent DDL execution
15007	HY090	not enough stack space to add entries
15008	HY000	table related objects are modified while validation
15009	HY000	Invalid argument
15010	42000	object modified by concurrent DDL execution
15011	HYC00	not implemented feature, in a function %s
15012	42000	user dropped by other session
15013	42000	user default schema dropped
15014	42000	user default data tablespace dropped
15015	42000	user default temporary tablespace dropped
15016	0Y002	limit(%d) on the number of named cursor exceeded
15017	42000	DDL not allowed for supplemental log table
15018	42000	cluster member '%s' has already joined in an other cluster system
15019	42000	the %s of cluster member '%s' is not compatible with the cluster system
15020	42000	tablespace '%s' of the cluster system does not exist at cluster member '%s'
15021	42000	tablespace '%s' of the cluster member '%s' does not exist at the cluster system
15022	42000	the %s of the tablespace '%s' at cluster member '%s' is not compatible with the cluster system
15023	HY000	limit(%d) on the number of named instant table exceeded
15024	42000	the cluster member '%s' does not join in a cluster system
15025	42000	startup phase of new cluster member should be OPEN
15026	HY000	internal error, in a function %s
15027	42000	attempt to create, alter or drop an index on temporary table already in use
15028	42000	user default index tablespace dropped

A.6 SQL 구문 처리 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-6 SQL 구문 처리 관련 에러 (16000~16099)

Error code	SQLSTATE	Message
16000	HY000	failed to initialize sql processor layer
16001	HYC00	not implemented feature, in a function %s
16002	HY000	invalid function sequence of sql processor layer
16003	42000	identifier is too long
16004	28000	invalid username/password; logon denied
16005	42000	name '%s.%s' is already used by an existing object
16006	42000	schema '%s' does not exist
16007	42000	schema '%s' is not writable
16008	42000	tablespace '%s' does not exist
16009	42000	tablespace '%s' is not writable
16010	42000	invalid value for PCTFREE or PCTUSED
16011	42000	invalid INITRANS option value
16012	42000	duplicate column name
16013	HY000	invalid value is prevented in validation step
16014	22023	INCREMENT must be a non-zero integer
16015	22023	START WITH cannot be less than MINVALUE
16016	22023	START WITH cannot be more than MAXVALUE
16017	22023	the number of values to CACHE must be greater than 1
16018	22023	number to CACHE must be less than one cycle
16019	22023	MINVALUE must be less than MAXVALUE
16020	22023	INCREMENT must be less than MAXVALUE minus MINVALUE
16021	22023	descending sequences that CYCLE must specify MINVALUE
16022	22023	ascending sequences that CYCLE must specify MAXVALUE
16023	22023	duplicate or conflicting CACHE/NO CACHE specifications
16024	42000	numeric precision specifier is out of range(%d to %d)
16025	42000	numeric scale specifier is out of range(%d to %d)
16026	42000	character length specifier is out of range(1 to %d)
16027	42000	character varying length specifier is out of range(1 to %d)
16028	42000	binary length specifier is out of range(1 to %d)
16029	42000	binary varying length specifier is out of range(1 to %d)
16030	42000	name already used by an existing constraint
		auto-generated name already used by an existing constraint, specify unique con...

Error code	SQLSTATE	Message
16031	42000	straint name
16032	42000	table can have only one primary key
16033	42000	such unique or primary key already exists in the table
16034	42000	name already used by an existing index
16035	42000	auto-generated name already used by an existing index, specify non-duplicate index name
16036	42000	'%s': invalid identifier
16037	42000	maximum number of key columns exceeded
16038	23000	integrity constraint violation
16039	3B000	savepoint '%s' never established in this session or is invalid
16040	42000	table or view does not exist
16041	42000	table '%s' is not writable
16042	42000	unique/primary keys in table referenced by foreign keys
16043	42000	table referenced by views
16044	42000	sequence does not exist
16045	42000	duplicate options
16046	42000	such column list already indexed
16047	22003	numeric value out of range
16048	42000	specified index does not exist
16049	42000	index '%s' is not writable
16050	42000	cannot modify index used for enforcement of unique/primary/foreign key
16051	HY000	not applicable
16052	42000	column not allowed here
16053	42000	not enough values
16054	42000	too many values
16055	22004	NULL value not allowed
16056	22004	cannot insert NULL into "%s"."%s"."%s"
16057	23000	unique constraint (%s.%s) violated
16058	42000	not applicable hint
16059	42000	not applicable datatype
16060	07002	bind variable does not exist
16061	22004	cannot update ("%s"."%s"."%s") to NULL
16062	42000	syntax error
16063	42000	fractional seconds precision specifier is out of range(%d to %d)
16064	42000	invalid interval indicator
16065	42000	interval leading precision specifier is out of range(%d to %d)
16066	42000	interval fractional seconds precision specifier is out of range(%d to %d)
16067	42000	invalid INITIAL storage option value
16068	42000	invalid NEXT storage option value
16069	42000	invalid MINSIZE storage option value

Error code	SQLSTATE	Message
16070	42000	invalid MAXSIZE storage option value
16071	42000	storage option MAXSIZE is less than MINSIZE
16072	42000	duplicate or conflicting NULL and/or NOT NULL specifications
16073	42000	duplicate UNIQUE/PRIMARY KEY specifications
16074	42000	sequence number not allowed here
16075	42000	inconsistent datatypes : expected %s
16076	42000	invalid DATAFILE clause for alter of %s %s TABLESPACE
16077	42000	invalid DATAFILE clause for alter of %s %s TABLESPACE
16078	42000	invalid TEMPFILE clause for alter of %s %s TABLESPACE
16079	42000	file path is too long (< %d)
16080	RD000	file size is out of range(%ld to %ld)
16081	42000	failed to open database : system threads are already exist.
16082	42000	invalid MAXTRANS option value
16083	42000	INITRANS must be less equal than MAXTRANS
16084	42000	result OFFSET must be 0 or positive numeric value
16085	42000	result LIMIT/FETCH must be positive numeric value
16086	42000	related objects are modified by concurrent DDL execution
16087	42000	boolean expression expected on where clause
16088	HY000	session cannot be killed now
16089	28000	built-in authorization(user or role) '%s' is not modifiable
16090	42000	built-in tablespace '%s' is not modifiable
16091	42000	built-in schema '%s' is not modifiable
16092	42000	built-in table or view '%s' is not modifiable
16093	42000	built-in index '%s' is not modifiable
16094	42000	built-in constraint '%s' is not modifiable
16095	42000	built-in sequence '%s' is not modifiable
16096	28000	authorization(user or role) '%s' does not exist
16097	42000	constraint '%s' does not exist
16098	07006	bind type mismatch of parameter number (%d)
16099	42000	the number of elements in the INTO list is mismatch

표 A-7 SQL 구문 처리 관련 에러 (16100~16199)

Error code	SQLSTATE	Message
16100	HY000	single-row subquery returns more than one row
16101	42000	parallel factor is out of range(0 to %d)
16102	42000	invalid MEMORY clause for alter of %s %s TABLESPACE
16103	3C000	cursor '%s' is already declared
16104	42000	invalid cursor query: '%s'
16105	42000	invalid cursor property

Error code	SQLSTATE	Message
16106	34000	cursor '%s' does not exist
16107	34000	cursor '%s' is already open
16108	34000	cursor '%s' is not open
16109	42000	invalid use: cursor '%s' is not a declared cursor
16110	42000	invalid use: cursor '%s' is not an ODBC/JDBC cursor
16111	42000	cursor '%s' is not scrollable
16112	42000	query expression is not updatable
16113	42000	invalid phase(%s): executable phase of '%s' is between %s and %s
16114	42000	database is already mounted
16115	42000	database is already open
16116	42000	transaction is READ ONLY
16117	42000	database not mounted
16118	42000	supplemental logging attribute primary key exists
16119	42000	cannot drop nonexistent primary key supplemental logging
16120	42000	column referenced by views
16121	42000	column has a check constraint defined
16122	42000	pseudo column not allowed here
16123	42000	cannot select ROWID from not writable table
16124	42000	table can have only one identity column
16125	42000	column being added already exists in table
16126	42000	table must be empty to add mandatory (%s) column ('%s')
16127	42000	cannot modify INITIAL storage option
16128	42000	name '%s' is already used by an existing tablespace
16129	42000	duplicate datafile name
16130	42000	file is already exist - '%s'
16131	42000	cannot rename the built-in tablespace
16132	42000	cannot rename the OFFLINE tablespace
16133	42000	cannot drop the built-in tablespace
16134	42000	cannot drop constraint - nonexistent constraint
16135	42000	cannot drop constraint - nonexistent primary key
16136	42000	cannot drop constraint - nonexistent unique key
16137	42000	the CDS database cannot be closed with ABORT option
16138	42000	the DS database can be closed with NORMAL option - deprecated
16139	42000	multiple ORDER BY clauses not allowed
16140	42000	multiple OFFSET clauses not allowed
16141	42000	multiple LIMIT clauses not allowed
16142	42000	column ambiguously defined
16143	42000	column part of USING clause cannot have qualifier
16144	42000	invalid OFFSET option value
16145	42000	invalid LIMIT option value

Error code	SQLSTATE	Message
16146	42000	common column name '%s' appears more than once in %s table
16147	42000	file is already used in database
16148	42000	tablespace not empty, use INCLUDING CONTENTS option
16149	42000	datafile does not exist in tablespace
16150	42000	cannot drop datafile of OFFLINE tablespace
16151	42000	cannot drop first datafile of tablespace
16152	42000	memory is already used in database
16153	42000	memory does not exist in tablespace
16154	42000	cannot drop first memory of tablespace
16155	42000	tablespace '%s' is not writable for logging index
16156	42000	tablespace '%s' is not writable for nologging index
16157	42000	ORDER BY item must be the number of a SELECT-list expression
16158	42000	group function is not allowed here
16159	42000	invalid number of arguments
16160	42000	group function is nested too deeply
16161	42000	not a single-group group function
16162	42000	mismatched datafile pairs
16163	42000	cannot rename datafile of ONLINE tablespace
16164	42000	file does not exist
16165	42000	tablespace is not online
16166	42000	tablespace is not offline
16167	42000	cannot offline temporary tablespace
16168	42000	cannot offline dictionary tablespace
16169	42000	cannot offline undo tablespace
16170	42000	invalid WAIT interval
16171	42000	column is not updatable
16172	22003	data is outside the range of the data type to which the number is being converted
16173	22015	one or more fields of data value were truncated during conversion
16174	42000	system session cannot be killed
16175	42000	hint not allowed in positioned DELETE
16176	42000	hint not allowed in positioned UPDATE
16177	34000	cursor '%s' is not updatable
16178	34000	cursor cannot identify the underlying table
16179	HY109	cursor was not positioned on a certain row
16180	HY109	cursor was positioned on a row that had been deleted
16181	42000	the CDS database cannot be closed with OPERATIONAL option
16182	42000	invalid segment identifier
16183	42000	FETCH position must be numeric value
16184	42000	cannot drop all columns in a table

Error code	SQLSTATE	Message
16185	42000	column already has a NOT NULL constraint
16186	42000	column does not have a NOT NULL constraint
16187	42000	null values found
16188	HY000	result set does not exist
16189	42000	column is not an identity column
16190	42000	column cannot be cast to incompatible type
16191	42000	column cannot be cast to decrease precision or scale
16192	42000	column cannot be cast to decrease string length
16193	42000	column cannot be cast to incompatible char length unit
16194	42000	column cannot be cast to incompatible interval indicator
16195	42000	float precision specifier is out of range(%d to %d)
16196	42000	file is already used in log stream - '%s'
16197	42000	duplicate logfile name
16198	42000	size of log file is smaller than minimum size of log file(%ld bytes).
16199	42000	can not find specified log member.

표 A-8 SQL 구문 처리 관련 에러 (16200~16299)

Error code	SQLSTATE	Message
16200	42000	the number of source logfiles is different from the number of targets
16201	42000	can not find specified log file.
16202	42000	name '%s' is already used by an existing schema
16203	42000	schema not empty, use CASCADE option
16204	RD000	another checkpoint is in progress
16205	42000	boolean expression expected
16206	RD000	maximum number of arguments exceeded
16207	42000	%s and %s cannot be matched
16208	42000	insufficient privileges
16209	42000	user %s lacks CREATE SESSION privilege; logon denied
16210	42000	lacks privilege (%s ON DATABASE)
16211	42000	lacks privilege (%s ON TABLESPACE "%s")
16212	42000	lacks privilege (%s ON SCHEMA "%s")
16213	42000	user name '%s' conflicts with another user or role name
16214	42000	tablespace '%s' is not data tablespace
16215	42000	tablespace '%s' is not temporary tablespace
16216	28000	user '%s' does not exist
16217	42000	user has own schema: DROP SCHEMA first
16218	42000	CASCADE must be specified to drop user '%s'
16219	42000	you may not GRANT/REVOKE privileges to/from yourself
16220	42000	duplicate authorization name in grantee/revokee list

Error code	SQLSTATE	Message
16221	42000	user default schema dropped
16222	42000	user default data tablespace dropped
16223	42000	user default temporary tablespace dropped
16224	42000	no privileges to GRANT ALL PRIVILEGES
16225	42000	insufficient privilege to GRANT
16226	42000	duplicate privilege listed
16227	42000	cannot drop a user that is currently connected
16228	42000	fixed table not grantable/revokable
16229	42000	invalid old password
16230	42000	schema element does not match schema identifier
16231	28000	cannot GRANT/REVOKE privileges to/from built-in authorization
16232	28000	cannot REVOKE privileges you did not grant to '%s'
16233	HY000	OFFLINE IMMEDIATE disallowed; noarchivelog mode
16234	HY000	OFFLINE IMMEDIATE disallowed; no logging tablespace
16235	2B000	dependent privilege descriptors still exist
16236	0A000	not supported query%s
16237	42000	a subquery block has incorrect number of result columns
16238	42000	ORDER BY item must be the number of a SELECT-list expression
16239	42000	expression must have same datatype as corresponding expression
16240	42000	subquery expressions not allowed here
16241	42000	invalid number of column names specified
16242	42000	must name this expression with a column alias
16243	01000	Warning: View definition has compilation errors
16244	42000	bind variables not allowed for data definition operations
16245	42000	view "%s.%s" has errors
16246	42000	illegal use of %s data type
16247	HY000	cannot BACKUP; noarchivelog mode
16248	HY000	cannot RECOVER TABLESPACE; tablespace is in use
16249	HY000	cannot SHUTDOWN; backup in progress
16250	HY000	RECOVER UNTIL CANCEL can be issued by directly attached session only - deprecated
16251	42000	single-row subquery returns more than one row
16252	42000	circular view definition encountered
16253	42000	only base table columns are allowed to GRANT/REVOKE
16254	42000	lacks privilege (%s ON TABLE "%s"."%s")
16255	42000	lacks privilege (USAGE ON SEQUENCE "%s"."%s")
16256	07001	wrong number of parameters
16257	HY000	argument of CASE/WHEN must be type boolean
16258	HY000	the XID does not exist
16259	HY000	the XID string is not valid

Error code	SQLSTATE	Message
16260	HY000	the global transaction is in progress
16261	42000	invalid INCREMENTAL BACKUP LEVEL specified : %d
16262	42000	INCREMENTAL BACKUP LEVEL 0 cannot have any BACKUP options
16263	HY000	cannot RESTORE TABLESPACE; tablespace is in use
16264	HY000	transaction is in progress
16265	42000	outer join operator (+) cannot be used with ANSI joins
16266	42000	outer join operator (+) not allowed in %s
16267	42000	an outer join cannot be specified on a correlation column
16268	42000	a predicate may reference only one outer-joined table
16269	42000	a table may be outer joined to at most one other table
16270	42000	two tables cannot be outer-joined to each other
16271	42000	RESTART value cannot be made to be less than the MINVALUE
16272	42000	RESTART value cannot be made to exceed the MAXVALUE
16273	42000	MINVALUE cannot be made to exceed the current value
16274	42000	MAXVALUE cannot be made to be less than the current value
16275	42000	MINVALUE cannot be made to exceed the START WITH value
16276	42000	MAXVALUE cannot be made to be less than START WITH value
16277	42000	log group id is already used
16278	42000	identity column cannot have a default value
16279	42000	DEFAULT expression has errors
16280	42000	the statement is disallowed
16281	42000	resource manager doing work outside transaction
16282	42000	MEMORY tablespace does not support an AUTOEXTEND option
16283	HY000	invalid ROWID
16284	42000	unable to execute in CDS mode
16285	RP000	failed to create system threads
16286	HY000	valid incremental backup does not exist
16287	42000	log group id does not exist
16288	42000	invalid string value specified
16289	42000	into clause can have only one row
16290	42000	SHUTDOWN must be executed in dedicated session
16291	40002	transaction rollback: integrity constraint violation
16292	42000	column length specifier is out of range(1 to %d) for columnar table
16293	42000	columnar table does not support LONG VARCHAR and LONG VARBINARY
16294	42000	cannot defer a constraint that is not deferrable
16295	42000	invalid NOT NULL constraint specified on an identity column
16296	42000	constraint '%s' is not deferrable
16297	42000	duplicate constraint name specified
16298	42000	cannot make primary key supplemental log data with primary key constraint violation

Error code	SQLSTATE	Message
16299	42000	datafile does not exist in database

표 A-9 SQL 구문 처리 관련 에러 (16300 ~ 16399)

Error code	SQLSTATE	Message
16300	42000	too many SQL resource
16301	42000	name '%s' is already used by an existing public synonym
16302	42000	private synonym '%s' to be droppped does not exist
16303	42000	public synonym '%s' to be droppped does not exist
16304	42000	looping chain of synonyms
16305	42000	synonym translation is no longer valid
16306	42000	expression is out of iteration time scope
16307	42000	constant value expression expected
16308	42000	invalid or redundant resource
16309	42000	the account is locked
16310	42000	the password will expire in %ld days
16311	42000	the password will expire soon
16312	42000	the password has expired
16313	42000	the password cannot be reused
16314	42000	cannot lock "SYS" account
16315	42000	profile '%s' already exists
16316	42000	function '%s' not found
16317	42000	profile '%s' has users assigned, cannot drop without CASCADE
16318	42000	profile '%s' does not exist
16319	42000	cannot assign PROFILE to "SYS" account
16320	42000	cannot assign DEFAULT parameter value to "DEFAULT" profile
16321	42000	password length less than '%d'
16322	42000	password must contain at least '%d' letter(s)
16323	42000	password must contain at least uppercase '%d' character(s)
16324	42000	password must contain at least lowercase '%d' character(s)
16325	42000	password must contain at least '%d' digit(s)
16326	42000	password must contain at least '%d' special character(s)
16327	42000	password same as or similar to user name
16328	42000	password contains the user name reversed
16329	42000	password same as or similar to database name
16330	42000	password too simple
16331	42000	Password should differ from the old password by at least '%d' characters
16332	42000	constant value does not support LONG VARCHAR and LONG VARBINARY
16333	42000	value of parameter number (%d) must be consistent datatype
16334	42000	invalid value is prevented in parameter number (%d)

Error code	SQLSTATE	Message
16335	42000	UPDATE primary key not allowed for supplemental log table
16336	42000	cluster group '%s' does not exist
16337	42000	duplicate cluster object listed
16338	42000	cluster domain '%s' does not exist
16339	42000	invalid hash shard count (must between %d and %d)
16340	42000	not found available sharding key column
16341	42000	cannot specify sharding strategy and placement for a table on dictionary schema '%s'
16342	HYC00	not implemented feature for cluster system, in a function %s
16343	42000	invalid syntax for a stand-alone system
16344	42000	database is not local opened
16345	42000	database is already opened
16346	42000	name '%s' is already used by an existing cluster object
16347	42000	cluster port is out of range(%d to %d)
16348	42000	duplicate cluster connection listed
16349	42000	the first cluster group does not contain local cluster member '%s'
16350	42000	maximum number of cluster members exceeded
16351	08000	failed to connect to the cluster member '%s'
16352	42000	mismatch local cluster member with location file
16353	42000	conflict between new cluster member and location file
16354	HY000	connection of member '%s' is broken
16355	42000	non-deterministic function not allowed here
16356	42000	version conflict
16357	42000	must be accessible to at least one member of group '%s'
16358	42000	accessible member does not exist
16359	42000	no inactive members
16360	42000	cloned table "%s"."%s" must be accessible to at least one member
16361	42000	sharded table "%s"."%s" must be accessible to at least one member of group '%s'
16362	42000	NODE SHARDED table cannot be rebalanced
16363	42000	the cluster member '%s' does not join in a cluster system
16364	42000	cluster group has sharded data
16365	42000	schema name of instant table must be SESSION_SCHEMA
16366	42000	hash shard count smaller than AT CLUSTER GROUP list
16367	42000	cannot drop sharding key column
16368	42000	not a fixed table
16369	42000	not a statistics table
16370	42000	have to create one or more datafiles to each member
16371	42000	conversion is not applicable
16372	42000	duplicate shard name

Error code	SQLSTATE	Message
16373	42000	the number of elements in shard bound mismatch the sharding key columns
16374	42000	range shard bound value must be one of: valid constant or MAXVALUE
16375	42000	only MAXVALUE allowed after MAXVALUE
16376	42000	multiple MAX shard defined
16377	42000	MAX shard not defined
16378	42000	duplicate shard bound value
16379	42000	failed to rebalance some tables
16380	HYC00	UNIQUE or PRIMARY KEY must include all sharding key columns for cluster system
16381	42000	invalid shard count (must between %d and %d)
16382	42000	cluster member '%s' does not exist
16383	42000	domain '%s' is unknown
16384	42000	multiple DEFAULT shard defined
16385	42000	DEFAULT shard not defined
16386	42000	DEFAULT cannot be specified with other values
16387	42000	list shard bound value must be one of: valid constant or NULL or DEFAULT
16388	HYC00	does not support deferrable constraints in the cluster system
16389	42000	domain names of the datafile for renaming don't mismatch
16390	42000	invalid table for sharding strategy and placement specified
16391	42000	the number of sharding key in function is mismatch
16392	HYC00	does not support ROWID in the cluster system
16393	01000	Warning: accessible member does not exist
16394	01000	Warning: connection of member '%s' is broken
16395	01000	Warning: must be accessible to at least one member of group '%s'
16396	42000	exceeds the maximum number of nodes
16397	42000	exceeds the maximum number of groups
16398	42000	the domain of property does not match with domain '%s'
16399	42000	cannot insert or update a generated always identity column, value shall be a DEFAULT

표 A-10 SQL 구문 처리 관련 에러 (16400 ~ 16499)

Error code	SQLSTATE	Message
16400	HY000	cannot RECOVER UNTIL CHANGE SCN; invalid SCN format
16401	42000	invalid SAMPLE size specified
16402	42000	duplicate index name
16403	42000	failed to join some of nodes to the global database
16404	42000	cannot execute the statement with OS user permissions
16405	42000	some tables in the database need to be rebalanced
16406	42000	invalid routine name
16407	42000	procedure or function does not exist

Error code	SQLSTATE	Message
16408	42000	bind variables not allowed for procedures or functions
16409	01000	Warning: Routine definition has compilation errors
16410	42000	Startup driver node must have the latest data - a suitable startup driver node is '%s' member
16411	42000	maximum number of recursive SQL levels (%d) exceeded.
16412	42000	all of shards in a table must be online.
16413	42000	LOCAL_OFFLINE domain must be specified to base table
16414	42000	domain option not allowed here
16415	42000	LOCAL_OFFLINE domain cannot be used with other domains
16416	42000	lacks privilege (EXECUTE ON PROCEDURE "%s"."%s")
16417	42000	active member '%s' cannot be offlined
16418	42000	cluster member '%s' does not exist in cluster group '%s'
16419	42000	object identifier not allowed here
16420	0A000	does not support adding identity column in the cluster system
16421	0A000	does not support deferred unique integrity in the cluster system
16422	0A000	does not support SERIALIZABLE in the cluster system
16423	42000	does not support non-deterministic DML in the cluster system : global secondary index expected
16424	42000	function generating different values is not allowed here
16425	42000	procedure, function, package, or type is not allowed here
16426	42000	does not support ADD COLUMN DEFAULT generating different values in the cluster system
16427	42000	invalid events (%s)
16428	42000	cluster system is modified by concurrent cluster DDL execution
16429	42000	does not support cursor in the cluster system : global secondary index expected
16430	42000	the cluster database can be closed only with ABORT or NORMAL option
16431	42000	non-deterministic expression not allowed here : may lead to a replica synchronization failure
16432	42000	global secondary index already exists in table
16433	42000	global secondary index does not exist in table
16434	HY000	invalid grid is prevented in cluster operation, in a function %s
16435	HY000	duplicate source datafile name
16436	HY000	duplicate target datafile name
16437	HY000	cannot execute on cloned tables
16438	HY000	invalid shard name - '%s'
16439	HY000	cannot move shard to the same group
16440	HY000	cannot execute on cluster wide sharded tables
16441	42000	not analyzed table
16442	42000	invalid analyzed file format (%s)
16443	42000	old plan

Error code	SQLSTATE	Message
16444	42000	invalid cluster group name
16445	42000	REBALANCE EXCLUDE can be executed only on cluster wide sharded tables
16446	42000	REBALANCE EXCLUDE target group is empty
16447	HY000	internal error (%s, %d)
16448	42000	of the total '%d' tables, '%d' tables failed to move shard
16449	42000	cluster group does not have sharded data
16450	42000	exceeded global property lock timeout
16451	42000	rownum not allowed here
16452	42000	rownum not allowed in sensitive cursor
16453	42000	rownum not allowed in positioned UPDATE
16454	42000	function '%s' not allowed here
16455	42000	routine "%s.%s" has errors
16456	42000	the startup phase '%s' of cluster member '%s' is mismatch with the required startup phase '%s'
16457	42000	the connection map of cluster member '%s' is mismatch with local connection map
16458	42000	reserved domain cannot be used with other domains
16459	42000	cannot execute on hash sharded tables
16460	42000	cannot execute on range sharded tables
16461	42000	invalid shard values
16462	42000	cannot split shard - source shard cannot be empty
16463	42000	first CREATE CLUSTER GROUP must include only one cluster member
16464	HY000	unsupported feature with temporary table
16465	42000	cannot drop local cluster group
16466	42000	cannot execute with SERIALIZABLE
16467	42000	table '%s' is READ ONLY
16468	HY000	cannot create TEMPORARY object in a NON-TEMPORARY tablespace
16469	42000	cannot execute on global connection
16470	42000	datafile size specified is smaller than minimum required '%ld'
16471	42000	audit policy '%s' does not exists
16472	42000	audit policy cannot be dropped as it is currently enabled
16473	42000	audit policy '%s' already exists
16474	42000	not auditable option
16475	42000	audit policy already applied with the BY clause
16476	42000	audit policy already applied with the EXCEPT clause
16477	42000	not auditable object
16478	42000	tablespace '%s' is not temporary tablespace or data tablespace
16479	42000	failed to update system version - %s
16480	42000	cannot set irrecoverable on the local cluster member
16481	42000	global tempory table segment no longer exists

Error code	SQLSTATE	Message
16482	42000	database is not accessible; '%s' has detached from the cluster

A.7 PSM 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-11 PSM 관련 에러

Error code	SQLSTATE	Message
17000	2F000	failed to initialize PSM processor layer
17001	2F000	not implemented feature (%s)
17002	2F000	duplicated label name
17003	2F000	duplicated exception name
17004	2F000	duplicated variable name
17005	2F000	not applicable expression type
17006	2F000	unknown variable or column name (%s)
17007	2F000	invalid expression
17008	2F000	wrong number of parameters
17009	2F000	NOT NULL constraint violation
17010	2F000	can not find label name
17011	2F000	duplicated field name
17012	2F000	unknown type name
17013	2F000	the type of record field should be scalar
17014	2F000	target host parameter of assignment is not bound in OUT mode
17015	2F000	cannot set constraint or initial value on non-scalar-type variable
17016	2F000	variables which have NOT NULL constraint should have initial values
17017	2F000	unhandled user exception
17018	2F000	pre-defined exception name not allowed in declare section
17019	2F000	OTHERS exception must be last among the exceptions of a block
17020	2F000	unknown exception
17021	2F000	Invalid error number for PRAGMA EXCEPTION_INIT
17022	2F000	A CASE statement must either list all possible cases or have an else clause
17023	2F000	duplicated error-code in same exception handler
17024	2F000	(%s) cannot be used as assignment target
17025	2F000	The command cannot be executed when global transaction is in the ACTIVE state
17026	2F000	Transaction is READ ONLY
17027	2F000	duplicated identifier
17028	2F000	out of scope
17029	2F000	too many depths of scope (limit=%d)
17030	2F000	invalid reference field (%s)

Error code	SQLSTATE	Message
17031	2F000	mismatch identifier type
17032	2F000	PSM compilation error
17033	2F000	duplicated cursor name
17034	2F000	return type of cursor is invalid
17035	2F000	invalid expression type
17036	2F000	cursor is not defined
17037	2F000	cursor is already open
17038	2F000	cursor is not open
17039	2F000	unknown cursor
17040	2F000	fetch target count mismatch
17041	2F000	execution fail
17042	2F000	bind variables not allowed for data definition operations
17043	2F000	no target-list for into clause
17044	2F000	returning-into need to bind variable with returning-into or using OUT clause
17045	2F000	no data found
17046	2F000	variables of record type can not be mixed with other variable
17047	2F000	only record-type variable allowed
17048	2F000	can not be used with INTO-clause and USING clause having OUT-modes
17049	2F000	can not use a expression in USING OUT binding
17050	2F000	query need to bind with INTO-clause for targets
17051	2F000	invalid statement-type
17052	2F000	Alias required in list of cursor to avoid duplicated target names
17053	2F000	schema or table object does not exist
17054	2F000	only record-type variable allowed
17055	2F000	this cursor must be declared with FOR-UPDATE to use with CURRENT OF
17056	2F000	invalid syntax error
17057	2F000	only select-into statement allowed
17058	2F000	source-name of return-type is not record-type or record-type variable
17059	2F000	source-name of rowtype must be a table, view, cursor, cursor-variable
17060	2F000	a cursor-variable with having return-type cannot be used in OPEN FOR dynamic-sql
17061	2F000	it can be allowed with only IN-Binding type
17062	2F000	duplicated routine name
17063	2F000	unknown routine name
17064	2F000	invalid routine name
17065	2F000	invalid routine object status
17066	2F000	a RAISE statement with no exception name must be inside an exception handler
17067	2F000	return types of Result Set variables or query do not match
17068	2F000	wrong number or types of arguments
17069	2F000	invalid variable usage

Error code	SQLSTATE	Message
17070	2F000	return statement in procedures or anonymous blocks cannot have <value> clause
17071	2F000	return statement in functions should have <value> clause
17072	2F000	existing state of routine has been discarded
17073	2F003	DDLs or DCLs cannot be invoked during DMLs or SELECT statements are executing in "%s"."%s"
17074	2F003	cannot begin transaction inside a query in "%s"."%s"
17075	2F003	cannot commit or rollback transaction inside a query or DML in "%s"."%s"
17076	2F000	application error "%s"
17077	2F000	the argument of collection type variable is not a constant expression
17078	2F000	a function with OUT-Parameter not allowed in expression
17079	2F000	a nested function not allowed in executing SQL
17080	2F002	cannot execute DML jobs inside a query
17081	2F000	program body of the routine is not defined
17082	HY000	bind parameter in target or return clause is not allowed
17083	2F000	Function "%s" returned without value

A.8 세션 관련 에러

- Error code: GOLDBLOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-12 세션 관련 에러

Error code	SQLSTATE	Message
18000	HY000	Invalid Statement
18001	HY024	Invalid attribute value
18002	HY092	Invalid attribute/option identifier
18003	42000	Transaction is READ ONLY
18004	42000	Statement cannot execute as atomic action
18005	42000	Preparable phase is open phase
18006	08S01	Communication link failure
18007	HYC00	not implemented feature, in a function %s
18008	HY014	Limit on the number of SQL statement exceeded
18009	42000	The command cannot be executed when global transaction is in the ACTIVE state
18010	07009	Invalid descriptor index
18011	HY000	Character set not equal(%s and %s)
18012	HY010	fetch out of sequence
18013	HYC00	Optional feature not implemented
18014	07001	Wrong number of parameters
18015	24000	Invalid cursor state

A.9 ODBC 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-13 ODBC 관련 에러

Error code	SQLSTATE	Message
19000	HY009	Invalid use of null pointer
19001	HY092	Invalid attribute/option identifier
19002	08003	Connection not open
19003	IM001	Driver does not support this function
19004	IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
19005	IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed
19006	08002	Connection name in use
19007	HY090	Invalid string or buffer length
19008	28000	Invalid authorization specification
19009	HY010	Function sequence error
19010	24000	Invalid cursor state
19011	HY012	Invalid transaction operation code
19012	HYC00	Optional feature not implemented
19013	HY003	Invalid application buffer type
19014	07009	Invalid descriptor index
19015	HY004	Invalid SQL data type
19016	HY105	Invalid parameter type
19017	HY009	Invalid argument value
19018	HY011	Attribute cannot be set now
19019	HY024	Invalid attribute value
19020	25000	Invalid transaction state
19021	HY000	General error
19022	07005	Prepared statement not a cursor-specification
19023	22001	String data, right truncated
19024	HY007	Associated statement is not prepared
19025	HY092	Option type out of range
19026	22002	Indicator variable required but not supplied
19027	01S01	Error in row
19028	HY016	Cannot modify an implementation row descriptor
19029	HY091	Invalid descriptor field identifier
19030	07006	Restricted parameter value violation
19031	22015	Interval field overflow

Error code	SQLSTATE	Message
19032	22003	Numeric value out of range
19033	01004	String data, right truncated
19034	22018	Invalid character value for cast specification
19035	01S07	Fractional truncation
19036	22008	Datetime field overflow
19037	22007	Invalid datetime format
19038	08S01	Communication link failure
19039	HY000	An invalid SQL type has occurred
19040	34000	Invalid cursor name
19041	HY000	Failed to atomic execution
19042	HY110	Invalid driver completion
19043	IM012	DRIVER keyword syntax error
19044	08001	Client unable to establish connection
19045	HY096	Information type out of range
19046	IM002	Data source not found and no default driver specified
19047	HY106	Fetch type out of range
19048	HY100	Uniqueness option type out of range
19049	HY101	Accuracy option type out of range
19050	HY097	Column type out of range
19051	HY098	Scope type out of range
19052	HY099	Nullable type out of range
19053	HY021	Inconsistent descriptor information
19054	01S02	Option value changed
19055	HY109	Invalid cursor position
19056	HY107	Row value out of range
19057	HYT00	Timeout expired
19058	2C000	Invalid or unknown NLS parameter value specified
19059	HY020	Attempt to concatenate a null value
19060	HY019	Non-character and non-binary data sent in pieces
19061	HY000	Invalid 'odbc.ini' specification. property : [%s]
19062	IM008	Dialog failed
19063	40001	Transaction rollback
19064	40003	Statement completion unknown
19065	08S01	Communication link failure
19066	01000	General warning
19067	HY104	Invalid precision or scale value
19068	40003	Retry the transactional operations
19069	08S01	Failed to communicate with locator

A.10 JDBC 관련 에러

- Error code: GOLDBLOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-14 JDBC 관련 에러

Error code	SQLSTATE	Message
21000	HV000	Not used error
21001	46001	Invalid connection url[%s]: [%s]
21002	46110	Unsupported feature: [%s]
21003	08S01	Invalid protocol: [%s]
21004	HV000	Invalid method call for read-only ResultSet
21005	08003	The Connection is already closed
21006	08003	The Statement is already closed
21007	08003	The ResultSet is already closed
21008	HV000	Invalid method call for forward-only ResultSet
21009	HV000	Invalid Wrapper type: [%s]
21010	HV000	Invalid argument: [%s]
21011	08000	Cannot connect to the server: %s
21012	08S01	Communication link failure: %s
21013	HV000	A mandatory property is omitted: [%s]
21014	HV000	Not a select query for executeQuery method
21015	HV000	The statement has produced a ResultSet
21016	HV000	This method cannot be called for a PreparedStatement
21017	HV000	Parameter type[%s] is mismatch with previous type[%s] during batch
21018	HV000	Lack of parameter: [%s]'th parameter
21019	HV000	No batch job added
21020	HV000	Batch job added
21021	22003	The value[%s] is out of range of [%s] type
21022	22026	The value length is mismatch[%s != %s]
21023	HV000	The value([%s] type) cannot be converted to [%s] type
21024	08000	The connection timeout expired: [%s millis]
21025	HV000	The column is not found: [%s]
21026	3B000	Savepoint error: %s
21027	HV000	Cannot execute the query in auto-commit mode
21028	24000	Invalid cursor position
21029	HV000	No column has been read
21030	HV000	Encoding error: [%s]
21031	HV000	Decoding error: [%s]

Error code	SQLSTATE	Message
21032	HV000	Stream error: [%s]
21033	HV000	Trying to read the deleted row
21034	HV000	Invalid interval format string: [%s]
21035	HV000	The Statement has an opened ResultSet
21036	HV000	The RowId object is unknown
21037	08000	The failover failed
21038	07009	The parameterIndex is not valid: [%s]
21039	HY000	No data read
21040	HY000	Invalid locator protocol version
21041	HY000	not matched locator protocol command sequence number
21042	HY000	Unable to open the file '%s'
21043	HY000	Unable to read the file '%s'
21044	HY000	Unable to close the file '%s'
21045	HY000	invalid INI file format
21046	HY000	Parameter[%s]'s input data type is mismatched with output data type

A.11 Embedded SQL 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-15 Embedded SQL run-time library 관련 에러

Error code	SQLSTATE	Message
23000	42000	syntax error
23001	0A000	feature not supported
23002	HY000	Bind variable "%s" was not declared
23003	HY000	Application can't be initialized
23004	HY000	Application can't be finalized
23005	RD000	Service is not available
23006	RD000	Statement did not prepared
23007	08001	Client can't establish SQL-connection
23008	HY000	cursor "%s" is not open
23009	HY000	Application can't create semaphore
23010	HY000	Application can't destroy semaphore
23011	HY000	Application can't create allocator
23012	HY000	Application can't destroy allocator
23013	HY000	Application can't create symbol table
23014	HY000	Application can't destroy symbol table
23015	IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
23016	IM001	Driver does not support ODBC 3.0
23017	HY000	Application can't allocate memory
23018	42000	Statement "%s" was not prepared
23019	42000	Descriptor name is too long
23020	HYC00	Not implemented feature, in a function %s
23021	07004	USING clause required for dynamic parameters
23022	07001	USING clause: illegal variable name/number
23023	07001	USING clause: not all variables bound
23024	07007	INTO clause required for result fields
23025	07001	INTO clause: illegal variable name/number
23026	07001	INTO clause: not all variables bound
23027	22002	null value, no indicator parameter
23028	07005	cursor "%s" statement is not a query
23029	08003	connection not open
23030	34000	cursor "%s" is not updatable
23031	08002	connection name "%s" in use

Error code	SQLSTATE	Message
23032	08002	duplicate connection context

표 A-16 Embedded SQL precompile 관련 에러

Error code	SQLSTATE	Message
41000	42000	syntax error
41001	0A000	feature not supported
41002	42000	Host variable "%s" not declared
41003	42000	Cursor "%s" not declared
41004	42000	"%s": file not exist
41005	42000	expression type does not match usage
41006	42000	Fatal error while doing embedded SQL preprocessing
41007	HY000	Precompiler can't create symbol table
41008	HY000	Precompiler can't destroy symbol table
41009	42000	identifier is too long
41010	42000	token "%s" is not valid in preprocessor expressions
41011	42000	multi-character character constant
41012	42000	floating constant in preprocessor expression
41013	42000	invalid suffix "%s" on integer constant
41014	42000	cursor name "%s" already declared
41015	07004	cursor "%s" is not a dynamic cursor
41016	42000	"%s" has no member named "%s"
41017	42000	storage size of "%s" isn't known
41018	42000	variable "%s" must be integer type
41019	42000	variable "%s" must be string type
41020	42000	invalid indicator variable type: indicator variable "%s" must have numeric type(char/short/int/long)
41021	42000	duplicate "%s"
41022	42000	both "%s" and "%s" in declaration specifiers
41023	42000	two or more data types in declaration specifiers
41024	42000	C++ punctuation sequences are not permitted
41025	42000	host variable expression has invalid type
41026	42000	typedef "%s" has invalid
41027	42000	too many 'define' macro (%d)
41028	42000	'%s' macro is already defined at line %d, in file %s
41029	42000	missing host variable
41030	42000	invalid getting groupid variable type: variable "%s" must have signed numeric type
41031	42000	invalid sql statement: getting groupid is available with select/update/delete/insert statement

A.12 Communication 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-17 Communication 관련 에러

Error code	SQLSTATE	Message
24000	08S01	Communication link failure
24001	08S01	Invalid communication protocol
24002	22000	Invalid GOLDDILOCKS_DATA Directory
24003	22000	not supported function
24004	22000	Invalid interprocess protocol
24005	22000	Unable to attach the shared memory segment. unable to access the file '%s'
24006	HV090	Invalid string length or buffer length
24007	22000	not matched command sequence number

표 A-18 Server communication 관련 에러

Error code	SQLSTATE	Message
52000	08S01	Communication link failure
52001	08000	Invalid communication protocol
52002	HY000	Not enough cm unit memory
52003	HY000	exceeded maximum packet allocation timeout

A.13 ServerLibrary 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-19 ServerLibrary 관련 에러

Error code	SQLSTATE	Message
25000	0A000	feature not supported
25001	HY000	Server is not running
25002	HY000	Environment Variable "%s" is not defined
25003	08000	Invalid protocol
25004	HY000	STARTUP must be executed in dedicated session
25005	HY000	session killed
25006	HY000	remote session is not opened
25007	42000	the property '%s' of cluster member '%s' is not compatible with the cluster system
25008	42000	the scn '%s' of cluster member '%s' is mismatch
25009	HY000	number of shared servers must be greater than or equal to number of request queues
25010	42000	invalid startup phase

A.14 gsql/ gsqlnet 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-20 gsql/ gsqlnet 관련 에러

Error code	SQLSTATE	Message
40000	42000	syntax error
40001	0A000	feature not supported
40002	HY000	Bind variable "%s" not declared
40003	42000	"%s": table or view does not exist.
40004	42000	"%s": index does not exist.
40005	RD000	Service is not available
40006	RD000	Statement did not prepared
40007	42000	'%s': duplicate statement
40008	42000	'%s': statement does not exist
40009	42000	'%s': statement is not prepared
40010	42000	insufficient privileges
40011	42000	driver does not support %s function
40012	42000	invalid object for describe
40013	42000	driver does not support the requested connection mode(%s)
40014	HY000	the rollback was caused by an unspecified reason
40015	HY000	the rollback was caused by a communication failure
40016	HY000	a deadlock was detected
40017	HY000	a condition that violates the integrity of the resources was detected
40018	HY000	the resource manager rolled back the transaction branch for a reason not on this list
40019	HY000	a protocol error occurred in the resource manager
40020	HY000	a transaction branch took too long
40021	HY000	may retry the transaction branch
40022	HY000	the inclusive upper bound of the rollback codes
40023	HY000	resumption must occur where suspension occurred
40024	HY000	the transaction branch may have been heuristically completed
40025	HY000	the transaction branch has been heuristically committed
40026	HY000	the transaction branch has been heuristically rolled back
40027	HY000	the transaction branch has been heuristically committed and rolled back
40028	HY000	routine returned with no effect and may be re-issued
40029	HY000	the transaction branch was read-only and has been committed
40030	HY000	asynchronous operation already outstanding

Error code	SQLSTATE	Message
40031	HY000	a resource manager error occurred in the transaction branch
40032	HY000	the XID is not valid
40033	HY000	invalid arguments were given
40034	HY000	routine invoked in an improper context
40035	HY000	resource manager unavailable
40036	HY000	the XID already exists
40037	HY000	resource manager doing work outside transaction
40038	HY000	unknown xa error
40039	HY000	marked the transaction branch rollback-only for unspecified reason
40040	HY000	cannot open the resource
40041	HY000	cannot close the resource
40042	HY000	the sql is empty
40043	HY000	invalid host variable type
40044	08003	connection does not exist
40045	HY000	invalid NUMSIZE value: NUMSIZE must be between %d and %d
40046	HY000	invalid PAGESIZE value: PAGESIZE must be between %d and %d
40047	HY000	invalid LINESIZE value: LINESIZE must be between %d and %d
40048	HY000	invalid COLSIZE value: COLSIZE must be between %d and %d
40049	HY000	invalid HISTORY value: HISTORY must be less equal than %d
40050	HY000	invalid DDLSIZE value: DDLSIZE must be between %d and %d
40051	HY000	invalid object identifier
40052	HY000	not enough DDLSIZE. use command: Wset ddsize {n}
40053	HY000	confirmation password mismatch
40054	HY000	invalid locator ini property - '%s'
40055	HY000	not exist member '%s' in file - '%s'
40056	HY000	some of nodes were failed to startup
40057	HY000	not specified valid location information
40058	HY000	some of nodes were failed to shutdown
40059	HY000	failed to connect to original driver node '%s' after startup
40060	HY000	cannot execute 'cstartup' or 'cshutdown' command in DA mode
40061	HY000	currently connected node is inactive

A.15 gloader/ gloadernet 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-21 gloader/ gloadernet 관련 에러

Error code	SQLSTATE	Message
42000	RD000	Service is not available
42001	HY000	invalid control file format
42002	HY000	buffer exceeded overflow area
42003	HY000	open same file(%s)
42004	HY000	occur invalid handle error
42005	HY000	occur error while operating SQLGetDiagRec/SQLGetDiagField
42006	HY000	using same character for delimiter and qualifier
42007	HY000	invalid thread unit count.(1~32)
42008	HY000	invalid array size.(1~65535)
42009	HY000	unavailable multithreading in export mode.
42010	08S01	Communication link failure
42011	HY000	invalid file size (minimum size : %d)
42012	HY000	unavailable file size in text mode.
42013	HY000	invalid column type.
42014	HY000	invalid column count.
42015	HY000	invalid data size.
42016	HY000	invalid header of binary file.
42017	HY000	file is empty.
42018	HY000	Qualifier must be 1 length character.
42019	HY000	LF and CR can not be used for delimiter or qualifier.
42020	HY000	qualifiers and terminators must not be subsets of each other
42021	HY000	table name must be given in case of absence of control file.
42022	HY000	can not fetch schema name.
42023	22001	byte length of data greater than column length.
42024	HY000	invalid direct io size

A.16 gmaster 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-22 gmaster 관련 에러

Error code	SQLSTATE	Message
44000	HY000	database is already mounted
44001	HY000	database is already local opened
44002	HY000	database is already opened
44003	HY000	system thread is already created
44004	HY000	failed to create system threads (%s)
44005	HY000	member '%s' does not exist
44006	HY000	fail to send packet through communication context
44007	HY000	failed to start up a process '%s'

A.17 glsnr 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-23 glsnr 관련 에러

Error code	SQLSTATE	Message
45000	HY000	Listener configuration file "%s" doesn't exist.
45001	HY000	Environment Variable "%s" is not defined.
45002	HY000	Listener is already started.
45003	HY000	overflow fd
45004	08000	Invalid communication protocol
45005	08004	access denied
45006	HY000	Operation timed out
45007	HY000	shared mode inactive
45008	HY000	Invalid property value [DEDICATED SHARED] : %s
45009	HY000	Invalid property value [NO INVITED EXCLUDED] : %s
45010	HY000	Invalid property value [YES NO 1 0]: %s
45011	HY000	Invalid property value [%ld ~ %ld]: %s
45012	HY000	Invalid property value [%ld ~ %ld]: %s
45013	HY000	buffer over flow %s

A.18 cyclone 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-24 cyclone 관련 에러

Error code	SQLSTATE	Message
46000	RD000	Service is not available
46001	HY000	"%s": file does not exist
46002	HY000	configure file processing error
46003	HY000	"%s": table does not exist
46004	HY000	"%s": schema does not exist
46005	HY000	"%s.%s": previously Added. Maybe duplicated
46006	HY000	"%s%s": table must have a primary key
46007	HY000	internal error occurred (%s)
46008	HY000	invalid Datatype(%d)
46009	HY000	"%s.%s" table must set supplemental log or primary key.
46010	HY000	group [%s] is already running
46011	HY000	cyclone is not running
46012	HY000	GOLDDILOCKS_DATA system environment is invalid
46013	HY000	"%s" log file reused or invalid. restart cyclone with '--reset' option
46014	HY000	fail to analyze flow[%s]
46015	HY000	Invalid Port Number (1024 ~ 49151) [%s]
46016	HY000	Communication link failure
46017	HY000	Invalid Protocol value (%s)
46018	HY000	Master disconnect abnormally
46019	HY000	Protocol Error Occurred (%s)
46020	HY000	Already Slave Connected
46021	HY000	Invalid Transaction Slot Id
46022	HY000	Invalid Supplemental Log (%d)
46023	HY000	Invalid operation for DDL
46024	HY000	Invalid value to analyze
46025	HY000	Invalid group name
46026	HY000	Not set master ip
46027	HY000	Invalid capture information
46028	HY000	There is no group in configure
46029	HY000	Duplicate group name (%s)
46030	HY000	Invalid Value : %s(%ld)
46031	HY000	Too low value : %s(%ld)

Error code	SQLSTATE	Message
46032	HY000	Invalid Host Port(%d)
46033	HY000	Redo Log file Read Timeout (%s)
46034	HY000	Invalid Archive Log File (%s)
46035	HY000	Invalid Temporary File (%s)
46036	HY000	Fail to write file (%s)
46037	HY000	Fail to Update State (%s)
46038	HY000	Fail to set Propagate mode at Session
46039	HY000	There is no active redo log file.
46040	HY000	There is no restart information. Master Database must be started first.
46041	HY000	Can Not Add New Table. Master Database must be started first.(%s,%s)
46042	HY000	Master Database must be started for the first time.
46043	HY000	Invalid Meta File(%s).
46044	HY000	Redo Log File does not exist(%s).
46045	HY000	USER_ID or USER_PW does not exist.
46046	HY000	Slave is not running with --sync option.
46047	HY000	Invalid Connect Information for sync (%s)
46048	HY000	(%s) value is too long
46049	HY000	Invalid encrypt/decrypt key
46050	HY000	no '--key' option
46051	HY000	password already exists (use one of USER_PW or USER_ENCRYPT_PW)
46052	HY000	reset or sync table (%s,%s) has never participated in replication
46053	HY000	reset or sync argument is invalid (%s)
46054	HY000	Table meta file has broken. please restart with '--reset all'
46055	HY000	protocol timeout
46056	HY000	Slave stopped normally
46057	HY000	Cannot use reset and sync option at the same time.
46058	HY000	There is no (%s,%s) table to sync (--sync option)
46059	HY000	(%s,%s) table (%s) column is not acceptable(GENERATED ALWAYS AS IDENTITY)
46060	HY000	driver does not support %s function
46061	HY000	data type is not supported.
46062	HY000	Cluster information does not matched(before groupid(%d), memberid(%d)/current groupid(%d), memberid(%d))
46063	HY000	Invalid cluster information at configure file(%s)
46064	HY000	Fail to rollback transaction

A.19 LogMirror 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-25 LogMirror 관련 에러

Error code	SQLSTATE	Message
47000	RD000	Service is not available
47001	HY000	Invalid Protocol value (%s)
47002	HY000	"%s": file does not exist
47003	HY000	configure file processing error
47004	HY000	invalid Control file
47005	HY000	Communication link failure
47006	HY000	LogMirror is already running
47007	HY000	GOLDILOCKS_DATA system environment is invalid
47008	HY000	There is no Shared Memory Area for LogMirror
47009	HY000	Invalid Value : %s(%s)
47010	HY000	Invalid port : (%d)
47011	HY000	Master disconnect abnormally
47012	HY000	Invalid Log File(%s)
47013	HY000	Archive Log File does not exist
47014	HY000	Archive Log file Read Timeout (%s)
47015	HY000	LogMirror is not running
47016	HY000	Connection Information does not exist
47017	HY000	Infiniband Receive Fail. (%ld)
47018	HY000	GOLDILOCKS Shutdown or Give-up LogMirror Service.
47019	HY000	Invalid encrypt/decrypt key
47020	HY000	no '--key' option
47021	HY000	password already exists(use one of USER_PW or USER_ENCRYPT_PW)
47022	HY000	protocol heartbeat timeout

A.20 cymon 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-26 cymon 관련 에러

Error code	SQLSTATE	Message
48000	HY000	GOLDDILOCKS_DATA system environment is invalid
48001	HY000	"%s": file does not exist
48002	RD000	Service is not available
48003	HY000	configure file processing error
48004	HY000	Duplicate group name (%s)
48005	HY000	Invalid Value : %s(%ld)
48006	HY000	Invalid group name
48007	HY000	Invalid Port Number (1 ~ 65535)
48008	HY000	There is no group in configure
48009	HY000	cymon is already running
48010	HY000	Communication link failure
48011	HY000	Protocol Error Occurred (%s)
48012	HY000	Already Monitor Connected
48013	HY000	cymon is not running
48014	HY000	internal error occurred (%s)
48015	HY000	Fail to update monitoring info
48016	HY000	USER_ID or USER_PW does not exist.
48017	HY000	Invalid encrypt/decrypt key
48018	HY000	no '--key' option
48019	HY000	password already exists(use one of USER_PW or USER_ENCRYPT_PW)

A.21 gdispatcher 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-27 gdispatcher 관련 에러

Error code	SQLSTATE	Message
50000	HY000	Dispatcher is already started
50001	HY000	overflow fd

A.22 gbalancer 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-28 gbalancer 관련 에러

Error code	SQLSTATE	Message
51000	HY000	overflow fd

A.23 gsyncher 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-29 gsyncher 관련 에러

Error code	SQLSTATE	Message
53000	HY000	invalid phase(%s): executable phase is %s
53001	HY000	GOLDILOCKS_DATA system environment is invalid
53002	HY000	gmaster is active
53003	HY000	gsyncher is already running
53004	HY000	Shared memory is cleared

A.24 Cluster 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-30 Cluster 관련 에러

Error code	SQLSTATE	Message
56000	HY000	location information of member '%s' already exists
56001	HY000	location information of member '%s' does not exist
56002	HY000	location information of member '%s' is modified by concurrent execution
56003	HY000	has inactive member (%s)
56004	HY000	location file is corrupted - '%s'
56005	HY000	connections of some cluster members are broken
56006	40000	transaction rollback: failed to prepare global transaction
56007	HY000	must be accessible to at least one of replicas
56008	40000	transaction rollback: failed to synchronize replicas
56009	HY000	remote server (member ID: %ld) is abnormally terminated
56010	40000	transaction rollback: remote session is abnormally terminated
56011	HY000	a shared session cannot modify NUMA property
56012	HY000	no connection was established to recover in-doubt transactions
56013	HY000	cannot resolve in-doubt transaction '%d.%d.%d' because '%d' members of the total '%d' remote cluster members were disconnected - connection map was '%s'
56014	HY000	MAX_NODE_COUNT property value '%d' must be equal to or less than '%d'
56015	HY000	'%s' property value must be greater than or equal to '%s' property value '%d'
56016	HY000	all members affected by the transaction are inactive - cannot guarantee durability of the transaction
56017	40000	transaction rollback: failed to open remote session
56018	HY000	protocol payload size(%ld) exceed maximum packet size(%ld)
56019	HY000	failover busy
56020	HY000	internal error (%s, %d)
56021	HY000	exceeded maximum response timeout
56022	HY000	failed to connect to an host

A.25 cserver 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-31 cserver 관련 에러

Error code	SQLSTATE	Message
57000	HY000	sample error

A.26 cdispatcher 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-32 cdispatcher 관련 에러

Error code	SQLSTATE	Message
58000	HY000	the sender failed to connect to the member(%d)
58001	HY000	the socket of target member '%s' is already disconnected
58002	HY000	the location information for member '%s' is not found
58003	HY000	the connection information is invalid; %s %d is either equal to or greater than %d
58004	HY000	the connection timeout from remote(%d)
58005	HY000	the disconnection timeout from remote(%d)
58006	HY000	invalid connection state transition
58007	HY000	dispatcher thread counts of two hosts do not match
58008	HY000	failed to start '%s' thread of cluster dispatcher
58009	HY000	internal error (%s, %d)
58010	HY000	broken socket
58011	HY000	old cluster connection is still alive
58012	HY000	failover event is in progress
58013	HY000	the connection can be established at LOCAL OPEN or OPEN phase

A.27 gtrclogger 관련 에러

- Error code: GOLDDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-33 gtrclogger 관련 에러

Error code	SQLSTATE	Message
59000	HY000	exceeded port cnt
59001	HY000	gtrclogger is already running
59002	HY000	gtrclogger is not running
59003	HY000	Operation timed out
59004	22000	Invalid GOLDDILOCKS_DATA directory

A.28 glocator 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-34 glocator 관련 에러

Error code	SQLSTATE	Message
60000	HY000	syntax error
60001	HY000	file does not exist - '%s'
60002	HY000	GOLDILOCKS_DATA system environment is invalid
60003	HY000	Invalid value - %s(%s)
60004	HY000	file name is too long - '%s'
60005	HY000	glocator is already started.
60006	HY000	Location file is corrupted - '%s'
60007	HY000	Property value is out of length - '%s'
60008	HY000	Invalid property value [YES NO 1 0] - '%s'
60009	HY000	Invalid property value [%ld ~ %ld] - '%s'
60010	HY000	Invalid property value [%ld ~ %ld] - '%s'
60011	HY000	File is already exist - '%s'
60012	HY000	Location file exceeded max size(%ld) - over size(%ld)
60013	HY000	Not exist node information - '%s'
60014	HY000	Exceeded response time for failover.
60015	HY000	Node is on failover - '%s'
60016	HY000	Need more alternate locator host information.
60017	HY000	Invalid configure file specifcation - property:[%s]
60018	HY000	Failed to propagate - '%s:%d'

A.29 gloctl 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-35 gloctl 관련 에러

Error code	SQLSTATE	Message
61000	HY000	Syntax error
61001	HY000	File is not exist - '%s'
61002	HY000	Can not find any ini object in file - '%s'
61003	HY000	Dsn is not exist - '%s'
61004	HY000	ADD MEMBER keyword syntax is error - '%s'
61005	HY000	ADD SERVICE keyword syntax is error - '%s'
61006	HY000	Invalid property value [YES NO 1 0] - '%s'
61007	HY000	Invalid property value [%ld ~ %ld] - '%s'
61008	HY000	Invalid property value [%ld ~ %ld] - '%s'
61009	HY000	STL_ENV_DB_DATA system environment is invalid
61010	HY000	Keyword syntax is invalid - '%s'
61011	HY000	Data value is invalid - '%s'

A.30 gmon 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-36 gmon 관련 에러

Error code	SQLSTATE	Message
62000	HY000	gmon is already running
62001	HY000	process(%ld) did not respond signal
62002	HY000	GOLDILOCKS_DATA system environment is invalid

A.31 gagent 관련 에러

- Error code: GOLDILOCKS 자체 에러 코드
- SQLSTATE: SQL 표준 상태 코드
- Message: 에러 메시지

표 A-37 gagent 관련 에러

Error code	SQLSTATE	Message
63000	HY000	GOLDILOCKS_DATA system environment is invalid
63001	HY000	Property value is out of length - '%s'
63002	HY000	Invalid property value [YES NO 1 0] - '%s'
63003	HY000	Invalid property value [%ld ~ %ld] - '%s'
63004	HY000	Invalid property value [%ld ~ %ld] - '%s'
63005	HY000	gagent is already started.
63006	HY000	file name is too long - '%s'
63007	HY000	file does not exist - '%s'
63008	HY000	Database is not cluster.
63009	HY000	Invalid configure file specifcation - property:[%s]
63010	HY000	Invalid phase(%s): executable phase is between %s and %s
63011	HY000	Property PORT of dsn(%s) is not set in file 'odbc.ini'

부록 B.

Wait Event

B.1 Wait Event

Wait event와 관련된 performance view는 다음과 같다.

- v\$system_event V\$SYSTEM_EVENT
- v\$session_event V\$SESSION_EVENT
- v\$session_wait V\$SESSION_WAIT
- v\$wait_event_name V\$WAIT_EVENT_NAME
- v\$wait_event_class_name V\$WAIT_EVENT_CLASS_NAME

B.2 Wait Event의 Class

- Administrative: Waits resulting from DBA commands that cause users to wait (for example, an index rebuild)
- Application: Waits resulting from user application code (for example, lock waits caused by row level locking or explicit lock commands)
- Cluster: Waits related to cluster resources (for example, global cache resources)
- Commit: This wait class only comprises one wait event - wait for redo log write confirmation after a commit (that is, 'log file sync')
- Concurrency: Waits for internal database resources (for example, latches)
- Configuration: Waits caused by inadequate configuration of database or instance resources (for example, undersized log file sizes, shared pool size)
- Idle: Waits that signify the session is inactive, waiting for work (for example, 'gsql message from client')
- Network: Waits related to network messaging
- Other: Waits which should not typically occur on a system
- Scheduler: Resource manager related waits
- System IO: Waits for background process IO
- User IO: Waits for user IO

B.3 Wait Event 항목

ENQUEUE: GDISPATCHER REQUEST

Shared mode에서 GDISPATCHER가 request를 enqueue하는 시간이다.

Parameter: None

ENQUEUE: SHARED-SERVER RESPONSE

Shared mode에서 shared server가 request를 dequeue하는 시간이다.

Parameter: None

DEQUEUE: SHARED-SERVER REQUEST

Shared mode에서 shared server가 request를 dequeue하는 시간이다.

Parameter: None

DEQUEUE: GDISPATCHER RESPONSE

Shared mode에서 GDISPATCHER가 response를 dequeue하는 시간이다.

Parameter: None

SEND: DEDICATE-SERVER SPOOLED RESPONSE

Dedicate mode에서 dedicate server가 spool 된 response를 client로 send하는 시간이다.

Parameter	설명
send data size	Send 할 data byte

SEND: DEDICATE-SERVER RESPONSE

Dedicate mode에서 dedicate server가 response를 client로 send하는 시간이다.

Parameter	설명
send data size	Send 할 data byte

RECV: DEDICATE-SERVER REQUEST

Dedicate mode에서 dedicate server가 client의 request를 receive하는 시간이다.

Parameter: None

SEND: GDISPATCHER RESPONSE

Shared mode에서 GDISPATCHER가 response를 client로 send하는 시간이다.

Parameter	설명
send data size	Send 할 data byte

RECV: GDISPATCHER REQUEST

Shared mode에서 GDISPATCHER가 client의 request를 receive하는 시간이다.

Parameter: None

ENQUEUE: CLUSTER REQUEST

Cluster의 request를 enqueue하는 시간이다.

Parameter: None

ENQUEUE: CLUSTER BROADCAST REQUEST

Cluster의 broadcast request를 enqueue하는 시간이다.

Parameter: None

DEQUEUE: CLUSTER RESPONSE

Cluster의 response를 dequeue하는 시간이다.

Parameter: None

SEND: CDISPATCHER

Cluster CDISPATCHER에서 send하는 시간이다.

Parameter: None

RECV: CDISPATCHER

Cluster CDISPATCHER에서 receive 하는 시간이다.

Parameter: None

GMASTER: ARCHIVE LOG

gmaster에서 archive log를 처리하는 시간이다.

Parameter: None

GMASTER: CHECKPOINT

gmaster에서 checkpoint를 처리하는 시간이다.

Parameter: None

GMMASTER: IO SLAVE

gmaster에서 IO slave를 처리하는 시간이다.

Parameter: None

GMMASTER: LOG FLUSH

gmaster에서 archive log를 처리하는 시간이다.

Parameter: None

GMMASTER: PAGE FLUSH

gmaster에서 page flush를 처리하는 시간이다.

Parameter: None

WRITE: TRACE LOG

Trace log를 write하는 시간이다.

Parameter: None

WRITE: COPY ARCHIVING LOG

Archiving log를 복사하는 시간이다.

Parameter: None

WRITE: BACKUP CTRL FILE

Control file을 백업하는 시간이다.

Parameter: None

WRITE: RESTORE CTRL FILE

Control file을 restore하는 시간이다.

Parameter: None

READ: ARCHIVE LOG

Archive log file을 read하는 시간이다.

Parameter: None

READ: CTRL FILE

Control file을 read하는 시간이다.

Parameter: None

WRITE: LOG FILE

Log file을 write하는 시간이다.

Parameter: None

WRITE: PAGE FILE

Page file을 write하는 시간이다.

Parameter: None

WRITE: CTRL FILE

Control file을 write하는 시간이다.

Parameter: None

WRITE: REMOVE DATA FILE

Data file을 삭제하는 시간이다.

Parameter: None

WRITE: JOURNAL BUFFER

Journal buffer를 write하는 시간이다.

Parameter: None

READ: JOURNAL BUFFER

Journal buffer를 read하는 시간이다.

Parameter: None

WAIT TRANSACTION

Transaction을 wait하는 시간이다.

Parameter	설명
wait transaction id	기다릴 transaction id

WAIT OTHER TRANSACTION

다른 transaction이 종료되기를 wait하는 시간이다.

Parameter	설명
wait transaction id	기다리는 transaction id
target transaction id	종료 대상 transaction id

WAIT ENABLE LOGGING

Logging이 가능할 때까지 wait하는 시간이다.

Parameter: None

WAIT LOG FLUSHER

Log flusher를 wait하는 시간이다.

Parameter	설명
send data size	Send 할 data byte

WAIT PAGE FLUSHER

Page flusher를 wait하는 시간이다.

Parameter	설명
send data size	Send 할 data byte

WAIT XA CONTEXT

XA context를 wait하는 시간이다.

Parameter: None

LATCH: LOG BUFFER

Log buffer latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PROCESS MANAGER

Process manager latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ENV MGR

Env manager latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SESSION ENV MGR

Session env manager latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PCH

Page Control Header (PCH) latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PAGE

Page latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PENDING LOG

Pending log latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ALLOC TRANS

Allocate transaction latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: UNDO SEGMENT

Undo segment latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: CLUSTER LOCATION

Cluster location latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DICT HASH ELEMENT AGING

Dict hash element aging latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DICT HASH RELATED AGING

Dict hash related aging latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: FILE MANAGER

File manager latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRACE LOG

Trace log latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: STATIC HASH

Static hash latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: STATIC HASH BUCKET

Static hash bucket latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SQL HANDLE

SQL handle latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: XA CONTEXT HASH

XA context hash latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PLAN CLOCK

Plan clock latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: XA CONTEXT

XA context latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: MEM CONTROLLER

Memory controller latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DYNAMIC MEM

Dynamic memory latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PROPERTY

Property latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ATTACH SHM

Attach shared memory latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BACKUP TBS

Backup tablespace latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DATABASE COMPONENT

Database component latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TABLESPACE

Tablespace latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BACKUP DATABASE

Backup database latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL BUFFER

Journal buffer latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL BUFFER ENTRY

Journal buffer entry latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL WRITE BUFFER

Journal write buffer latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: LOCK ITEM

Lock item latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: RECORD HASH

Record hash latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DEADLOCK

Deadlock latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE

Sequence latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: LOG STREAM

Log stream latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BUILD AGABLE SCN

Build agable SCN latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRANSACTION TABLE

Transaction table latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SESSION LINK HASH

Session link hash latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ALLOC XA CONTEXT

Allocate XA context latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE GLOBALX

Sequence global latch X를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE GLOBALY

Sequence global latch Y를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRANSACTION LOG FILE

Transaction logfile latch를 기다리는 시간이다.

Parameter	설명
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

부록 C.

Open Source License

아파치 라이선스 버전 2.0

사용, 복제 및 배포에 관한 규정과 조건

버전 2.0, 2004년 1월

제1조. 정의

"라이선스"란 본 문서의 제1조에서 제9조에 걸쳐 정의되고 있는 사용과 복제 및 배포에 관한 규정과 조건을 의미한다.

"라이선서"란 저작권 소유자 혹은 저작권 소유자의 위임을 받아 라이선스를 부여하는 사람을 지칭한다.

"법인"이란, 행위자와 그 행위자를 관리하거나, 행위자에 의해 관리되거나, 행위자와 공동 관리 하에 있는 모든 주체로 구성된 연합체를 가리킨다. 이 정의에서 "관리"란 (i) 계약 또는 그 외의 방식으로, 직접 또는 간접적으로 이 법인의 지휘나 경영을 야기하는 권한, 또는 (ii) 이 법인에 대해 50% 이상의 주식 소유권, 혹은 (iii) 수익 소유권을 가지는 것을 지칭한다.

"당신"은 본 라이선스에 의해 부여되는 권리를 행사하는 개인이나 법인을 지칭한다.

"소스" 형식이란 소프트웨어의 소스 코드, 문서화 소스, 설정 파일 등에 대해 변경을 추가하는데 용이한 형식을 의미한다.

"오브젝트" 형식이란 소스 형식에 대한 기계적인 변환이나 번역을 통해 생성된 모든 형식으로, 컴파일된 오브젝트 코드, 생성된 문서, 기타 매체 형식으로서의 전환 등을 포함하나 여기에 국한되지는 않는다.

"저작물"이란 소스 형식이나 오브젝트 형식을 불문하고 저작물에 포함되거나 첨부된 저작권 문구에 의해 표시된 것처럼 (아래의 부록 사례를 참조) 본 라이선스에 의해 사용이 허가되는 저작물을 의미한다.

"파생 저작물"이란 소스 형식이나 오브젝트 형식을 불문하고 저작물에 기반하거나 저작물로부터 파생된 모든 저작물로, 이에 대한 편집상의 개정이나 주석, 퇴고 등의 수정행위가 전체적으로 원 저작물을 나타내는 것을 의미한다. 본 라이선스에서의 파생 저작물은 저작물 및 그것의 파생 저작물의 인터페이스와 별도로 독립적으로 존재하거나 단순히 링크나 이름으로만 연결된 저작물은 포함되지 않는다.

"기여"란 저작물의 원본을 비롯하여 저작물이나 그것의 파생 저작물에 대한 수정이나 추가를 포함하는 모든 저작물로서, 저작권자 혹은 저작권자로부터 위임을 받은 개인이나 법인이 저작물에 반영시키기 위한 의도로 라이선서에게 제출한 것을 의미한다.

본 정의에 있어서 "제출"이란 저작물을 논하거나 개선하기 위한 목적으로 라이선서나 그의 대리인에 의해 관리되는 전자메일링 리스트, 소스 코드 관리 시스템, 이슈 추적 시스템 등을 사용한 전자적 방식이나 구두, 혹은 서면 등의 방식으로 소통하는 모든 형태를 의미한다. 단, 저작권자가 "기여가 아님" 이라고 명확하게 표시하거나 서면으로 표현한 방법은 제외된다.

"기여자"란 자신의 기여가 라이선서에 의해 받아들여지고 결과적으로 저작물에 반영된 라이선서 및 모든 개인이나 법인을 의미한다.

제2조. 저작권 라이선스의 부여

본 라이선스의 규정과 조건에 따라, 각 기여자는 전세계 어디에서나 사용 가능한 영구적이고 번복이 불가능한 비독점 저작권 라이선스를 당신에게 무료로 부여한다. 이 라이선스는 저작물이나 파생저작물을 소스 형식이나 오브젝트 형식으로 복제, 파생 저작물 생성, 전시, 공연, 서브라이선스 설정, 재배포 하는 행위에 대한 사용허가이다.

제3조. 특허 라이선스의 부여

본 라이선스의 규정과 조건에 따라 각 기여자는 당신에게 저작물을 작성 및 사용, 판매, 판매를 위한 청약, 수입, 그 외 방법으로 이전할 수 있는, 영구적이면서 전 세계에서 사용 가능하고 (본 조항에 명시된 경우를 제외하고) 번복이 불가능한 비독점 특허 라이선스를 무료로 부여한다. 단, 이러한 특허 라이선스는 기여자에 의해 라이선스 허용이 가능한 특허 청구항 중에서 해당 기여자의 기여가 단독으로, 혹은 기여가 반영된 저작물과의 조합을 통해 필연적으로 특허를 침해하는 경우에만 적용된다. 만약 당신이 어떤 개인이나 법인을 상대로, 저작물이나 저작물 내에 반영된 기여가 직접적 혹은 간접적으로 특허를 침해한다고 주장하는 특허 소송(교차 청구나 반소 포함)을 걸면, 본 라이선스에 근거해 당신에게 부여된 특허 라이선스는 해당 소송이 제기되는 시점에서 전부 종료된다.

제4조. 재배포

당신은 저작물이나 파생 저작물의 사본을 원본 그대로, 혹은 수정한 상태로 소스 형식이나 오브젝트 형식으로 복제하거나 배포할 수 있다. 이는 어떤 매체로든지 가능하며, 다음과 같은 조건들을 충족시켜야 한다.

1. 저작물이나 파생 저작물을 양도받는 모든 사람에게 본 라이선스의 사본을 제공해야 한다. 또한,
2. 수정된 파일에 대해서는 수정을 했다는 사실을 설명하는 명확한 안내문구를 첨부해야 한다. 또한,
3. 당신은 저작물의 소스 형식에 포함된 모든 저작권, 특허, 상표, 귀속에 관한 고지 내용을 당신이 배포하는 모든 파생 저작물의 소스 형식 내에 존속시켜야 한다.
4. 저작물이 배포되었을 때 “NOTICE” 에 해당하는 텍스트 파일을 포함하는 경우에는, 당신이 배포하는 파생 저작물에는 그러한 NOTICE 파일 내에 포함되어 있는 귀속에 관한 고지 사항을 읽을 수 있는 사본을 포함시켜야 한다. 단, 저작물의 어떤 부분에도 관련되지 않는 고지 사항들은 제외해야 하며, 다음 중 적어도 하나의 위치에 삽입해야 한다. 즉, (i) 파생 저작물의 일부로 배포되는 “NOTICE” 텍스트 파일, 또는 (ii) 파생 저작물과 함께 소스 형식이나 문서가 제공되는 경우 그 소스 형식이나 문서, 또는 (iii) 파생 저작물에 제3자 고지 사항이 통상적으로 나타나는 경우, 그것이 표시되는 모든 위치에 삽입해야 한다. NOTICE 파일의 내용은 어디까지나 정보 전달을 위한 것으로, 라이선스를 수정하지 않는다. 당신은 자체적으로 작성한 귀속 사항을 당신이 배포하는 파생 저작물 내에 저작물의 NOTICE 텍스트에 연결되거나 추가된 형식으로 추가할 수 있다. 단, 이는 이러한 추가적인 귀속 사항이 라이선스를 수정한다고 해석될 우려가 없는 경우에만 가능하다.

당신은 자신의 수정 물에 자체적으로 저작권 표시를 추가할 수 있으며, 수정물의 사용, 복제, 배포, 혹은 그러한 파생 저작물 전체에 대해 추가적인 라이선스 조항이나 별도의 라이선스 조항들을 만들 수 있다. 단, 이는 저작물에 대한 당신의 사용, 복제 및 배포 행위가 그 외의 사항들에서는 본 라이선스에 명시된 조건들을 준수한다는 것을 전제로 한다.

제5조. 기여의 제출

명확한 이견을 제시하지 않는 한, 당신이 저작물에 반영시키기 위해 라이선서에게 제출한 모든 기여는 다른 추가적인 규정이나 조건 없이 본 라이선스의 규정과 조건을 따라야 한다. 위의 내용과 관계없이, 그러한 기여와 관련하여 당신이 라이선서와 체결한 별도의 라이선스 계약의 조항들이 이 사실로 인해 대체되거나 수정되지는 않는다.

제6조. 상표

본 라이선스는 저작물의 출처를 설명하거나 NOTICE 파일의 내용을 복제하는데 필요한 합당하고 관례적인 경우를 제외하고는 라이선서의 상호나 상표, 서비스 표, 제품명을 사용할 권한을 부여하지 않는다.

제7조. 보증의 부인

관련 법규에 의한 요청이나 서면으로 동의한 경우가 아니라면, 라이선서는 저작물을 (그리고 기여자는 각자의 기여를) 타이틀이나 비침해성, 특정한 목적에 대한 적합성이나 상품성 여부에 대한 보증이나 조건을 포함한 그 어떠한 형태의 보증이나 조건도 명시적이나 묵시적으로 설정되지 않은 “있는 그대로의” 상태로 제공한다. 당신은 저작물의 사용이나 재배포의 적절성을 스스로 판단할 책임을 지며, 본 라이선스에 의해 부여된 권리를 행사하는 것에 따르는 모든 위험성을 부담하게 된다.

제8조. 면책

기여자가 손해발생 가능성을 사전에 알고 있었다 하더라도 관련 법규(고의적 과실이나 중과실 등)나 서면 동의에 의해 구속되지 않는 한, 법이론이나 계약, 불법행위 (과실 포함) 및 기타 이유로 인한 어떤 경우에도, 기여자는 본 라이선스나 저작물의 사용이나 작동 불능으로 인해 발생한 직접적이거나 간접적인 손해, 특수하거나 우발적이거나 결과적 손해 등 어떤 손해에 대해서도 책임지지 않는다. (이러한 조건은 영업권의 상실, 업무 정지, 컴퓨터 장애나 오작동 및 그 외의 모든 상업상의 손해나 손실에 대해서도 적용되나 이에만 국한되는 것은 아니다.)

제9조. 보증이나 추가적 책임의 인수

저작물이나 그로부터 파생된 저작물을 재배포할 때, 당신은 지원이나 보증, 손해보상, 또는 그 외의 책임이나 본 라이선스와 모순되지 않는 권리를 제공할 수 있으며, 이에 대해 수수료를 받을 수 있다. 단, 이러한 의무사항들을 받아들이는데 있어, 당신은 다른 기여자를 대신할 수 없으며 단독으로 책임을 져야 한다. 또한 당신이 이러한 보증이나 추가적 책임을 인수한 것으로 인해, 다른 기여자에게 어떤 책임이 부과되거나 배상이 요구되더라도 그러한 기여자에게 피해가 미치지 않도록 보호하는데 동의해야 한다.

Copyright 2019-2022 (주)선재소프트

Apache License 버전 2.0(본 라이선스)의 적용을 받음.

이 파일을 사용하기 위해서는 반드시 본 라이선스를 따라야 합니다. 본 라이선스의 사본은 다음 사이트에서 구할 수 있습니다. <http://www.apache.org/licenses/LICENSE-2.0>

관련 법규나 서면 동의에 의해 구속되지 않는 한, 본 라이선스에 따라 배포되는 소프트웨어는 어떠한 보증이나 조건도 명시적으로나 묵시적으로 설정되지 않는 “있는 그대로”의 상태로 배포됩니다. 본 라이선스가 허용하거나 제한하는 사항을 규정한 문언에 대해서는 라이선스를 참조하십시오.

