GOLDILOCKS LITE 3.1 Manual (ko)

07217 서울시 영등포구 당산로 171 금강펜테리움IT타워 604호
전화: 02-322-6288 팩스: 02-322-6788
Webpage: www.sunjesoft.com
Support: technet@sunjesoft.com

GOLDILOCKS LITE 3.1 Manual (ko)

SUNJESOFT Inc

차례

1.

Getting Started

1.1 개요

GOLDILOCKS LITE는 트랜잭션 기능을 제공하는 Shared Memory 기반 데이터 관리 C library이다.
아래의 특징들을 갖는다.

- Shared Memory Direct Attached 방식으로 동작하는 C/C++ library
- Shared Memory 자동 확장 및 최대 크기 제한 기능 제공
- 장애 복구 및 안정성을 위한 Disk Logging 기능 제공
- C 언어 기반의 직관적인 API (Application Interface) 제공 (일부 ODBC APIs와 JNI 제공)
- RDBMS에 익숙한 사용자를 위해 SQL syntax 제공
- Array, B+tree, splay 등과 같은 탐색 방식 지원
- 트랜잭션 기능 제공 (Atomic, Concurrency, Durability 등의 기능 제공)

다음은 C 구조체를 어떤 방식으로 GOLDILOCKS LITE에 저장할 수 있는지 설명한다.

```
struct user_data
{
    int             mEmpNo;
    char            mEmpName[20];
    int             mDeptNo;
    struct timeval  mBirth
}
```

- 위와 같은 C 구조체를 GOLDILOCKS LITE에서는 다음과 같은 SQL syntax를 사용하여 테이블로 생성할 수 있다.

```
dbmMetaManager(DEMO)> create table user_data
(
    empNo       int,
    empName     char(20),
    deptNo      int,
    birth       date
)
success
dbmMetaManager(DEMO)>
```

- 다음과 같은 SQL syntax를 사용하여 인덱스를 생성할 수 있다.

```
dbmMetaManager(DEMO)> create unique index idx_user_data on user_data( empNo);
success
```

- 다음과 같은 SQL syntax를 데이터를 삽입/조회할 수 있다.

```
dbmMetaManager(DEMO)> desc user_data;
-------------------------------------------------------------------------------
Instance=(DEMO) Table=(USER_DATA) Type=(TABLE) RowSize=(40) LockMode(1)
-------------------------------------------------------------------------------
EMPNO                           int                 4          0
EMPNAME                         char                20         4
DEPTNO                          int                 4          24
BIRTH                           date                8          32
-------------------------------------------------------------------------------
IDX_USER_DATA                   unique     (EMPNO asc)
-------------------------------------------------------------------------------
success
dbmMetaManager(DEMO)> insert into user_data values (1, 'alice', 100, sysdate);
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select * from user_data;
-------------------------------------------------------------------------
EMPNO   : 1
EMPNAME : alice
DEPTNO  : 100
BIRTH   : 2025/07/30 08:08:15.484030
-------------------------------------------------------------------------
1 row selected
```

- 제공되는 API로 다음과 같이 개발할 수 있다.

```
struct user_data xData;
// 데이터 저장 API 예제
dbmInsertRow( Handle, "node", &xData, sizeof(struct user_data));
// 데이터 조회 API 예제
dbmSelectRow( Handle, "node", &xData );
...
```

노트

- dbmMetaManager는 사용자가 GOLDILOCKS LITE를 사용할 수 있게 제공되는 Interactive tool이다.

  (1.5 dbmMetaManager 참조)

- API의 상세 설명은 **2.2 C/C++ APIs** 참조한다.

Object

GOLDILOCKS LITE에서 Object란 Shared Memory 상에 생성되는 모든 유형의 Segment를 의미한다.

노트

Object 이름의 최대 길이는 64 byte이다.

DICTIONARY

Dictionary는 Object 정보를 관리하기 위해 Instance 생성 시점에 자동으로 생성되는 테이블 및 상태 정보를 볼 수 있도록 제공되는 내부 View들을 의미한다. (**1.4 DICTIONARY**를 참조)

INSTANCE

GOLDILOCKS LITE를 이용하는 세션의 정보와 트랜잭션 처리를 위한 공간으로 사용된다. 일반 RDBMS의 Undo Segment나 SGA/PGA와 유사한 의미를 갖는다.

저장 속성	설명
Session area	Instance 에 접근하는 세션의 정보를 저장하는 공간이다. (SessionID, PID, status 등)
Transaction area	Session에서 발생한 트랜잭션의 정보를 저장하는 공간이다. (트랜잭션 로깅 등)
Rollback image area	트랜잭션에 의해 변경되기 전의 record image를 저장하는 공간이다.

노트

- INSTANCE는 2가지 유형을 갖는다.
  - Dictionary Instance(이하 DICT) : **initdb**에 의해 생성되는 최상위 Instance
  - User Instance : 사용자가 생성하는 Instance
- 1개의 Instance에는 최대 1023 개의 세션이 동시에 접속할 수 있다.

TABLE

- TABLE은 data를 저장하는 공간이다.
- 한 개의 Instance 안에서 테이블 명은 고유해야 하며 테이블의 생성 개수에는 제한이 없다.

다음과 같은 유형의 테이블들을 지원한다.

테이블 유형	설명
Normal table	B tree indexing segment를 가질 수 있는 테이블
Direct table	한 개의 column value를 key로 사용하는 array 형태의 테이블
Splay table	Splay indexing 방식을 사용하는 테이블
Queue table	FIFO(First-In, First-Out) 방식 테이블
Store table	Char data type만으로 구성된 Key/Value 형식의 테이블
Sequence	채번을 위한 object

Column 타입으로 지정할 수 있는 데이터 유형은 다음 표와 같다.

Column type	설명
int	sizeof(int)
short	sizeof(short)
float	sizeof(float), 별도의 정밀도를 제공하지 않음
long	sizeof(long long)
char (size)	입력된 size 크기의 fixed 공간
double	sizeof(double), 별도의 정밀도를 제공하지 않음
date	sizeof(unsigned long long)

> 주의
> C struct의 default padding/packing 방식을 기반으로 offset과 size를 설정하기 때문에 #pragma pack 구문을 사용하여 default와 다를 경우 application은 정상 동작하지 않는다.

INDEX

데이터를 효율적으로 검색하기 위해 제공되는 object 이다.

- Unique 여부를 지정할 수 있다.
- (float, double)과 같은 유형은 index key column으로 사용하는 것을 권장하지 않는다.
- 하나 이상의 composite key를 구성할 수 있다.
- Index Key Column의 정렬 방식을 지정할 수 있다. (ASC, DESC)
- Normal Table만 Secondary Index를 생성할 수 있다.

노트

- DIRECT, SPLAY, QUEUE, STORE 테이블은 특성 상 1개의 INDEX만 생성할 수 있다.
- DIRECT Table은 1개의 정수형 데이터 타입을 가진 컬럼만 Index로 지정할 수 있다.
- STORE, QUEUE Table은 생성 시점에 자동으로 Btree 기반 Index가 생성되어임의로 변경할 수 없다.

DIRECT TABLE

테이블의 숫자형 데이터 타입 컬럼 중 하나를 index key로 지정하여 해당 컬럼의 데이터 값을 테이블 내 저장 위치로 사용한다. 예를 들어, 값이 1 이라면 table에 데이터를 저장할 수 있는 공간 중 1 번 위치에 데이터를 저장한다. (array indexing 저장 구조)

노트

Direct table은 create unique index 구문을 통해 반드시 하나의 column만 key로 지정해야 한다. Key로 지정할 수 있는 data type 타입은 long, short, int 이다.

SPLAY TABLE

Splay table은 splay indexing으로 정렬되는 테이블이다. Splay table은 이전에 조회하거나 조작한 데이터 근처의 데이터를 빠르게 탐색해야 하는 작업에 특화된 테이블 기능이다.

주의

Splay table은 데이터 조작 및 접근 패턴에 따라 성능이 달라질 수 있다.

STORE TABLE

테이블의 column을 별도로 정의하지 않고 key와 value 크기만 정의하여 생성하는 테이블이다.
SET 기능을 이용하여 데이터를 저장하고 GET 기능을 이용하여 조회한다.
Key의 최대 길이는 64byte, value의 최대 크기는 512K 이다. (Fixed Size로 동작)

QUEUE

First-In/First-Out (FIFO) 방식으로 사용할 수 있는 object이다.

Dequeue 동작은 다음과 같다. 데이터가 (1, 2, 3) 순서로 queue에 저장되어 있을 때, A 세션이 1을 dequeue 한 상태에서 아직 커밋하지 않았더라도 B 세션은 2를 dequeue 할 수 있다. 즉, A 세션이 커밋할 때까지 기다리지 않는다.

> 노트
> Priority는 숫자가 낮을수록 먼저 출력된다. 우선 순위를 높여서 처리하려면 priority 값을 낮은 숫자로 enque ue 해야 한다.

SEQUENCE

고유한 숫자를 순차적으로 생성하고자 할 때 사용할 수 있는 Object이다.

동시성 제어

GOLDILOCKS LITE에는 별도의 관리용 프로세스가 존재하지 않으며 접근하는 세션이 instance 일정 영역에 정보를 기록한다. 이를 기반으로 세션 간의 동시성 제어를 수행한다.

Row Level Lock

GOLDILOCKS LITE는 row level lock을 제공한다.

Read Committed

세션은 갱신 작업을 수행하기 전에 갱신할 image를 별도의 공간 (instance undo space)에 저장한다. 이 때 접근하는 조회 세션이 갱신 트랜잭션을 기다리지 않고 이전에 commit 된 image를 읽을 수 있도록 동시성을 제공한다.

> 노트
> 해당 동작은 DBM_MVCC_ENABLE property (**환경 변수 및 프로퍼티**참조) 에 의해 제어되는데, 사용자가 데이터를 조회하거나 갱신하는 동안 대기하도록 하려면 이 속성을 FALSE로 설정해야 한다.
> (기본 값은 TRUE 이다.)

Auto Dead Lock Detection

갱신 연산 간에 서로를 기다리는 상황이 발생할 수 있다. 예를 들어 T1 테이블에 (A, B) 레코드가 존재할 때 세션 1이 A를 갱신한 후 B를 갱신하려고 하고, 동시에 세션 2가 B를 갱신한 후 A를 갱신하려고 하면, 세션 1과 세션 2는 교착 상태에 빠진다.
이런 상황은 Library내에서 자동으로 감지되며, 세션 ID가 높은 값을 가진 세션에 오류가 발생하도록 동작한다.

> 주의
> Application이 교착 상태 오류를 반환받은 경우, 자동으로 이전 트랜잭션이 rollback 되지 않기 때문에, 사용자가 명시적으로 트랜잭션을 (commit 또는 rollback) 해야 교착 상태가 해소된다.

Delayed Recovery Concept

Delayed recovery는 GOLDILOCKS LITE에서 비정상적으로 종료된 트랜잭션을 Application 스스로 감지하고 복구하는 기능이다. GOLDILOCKS LITE는 세션이나 트랜잭션을 관리하는 별도의 프로세스를 가지고 있지 않아 Lock을 점유하려는 세션이 직전에 접근했던 세션의 비정상 종료 여부를 직접 감지하고 필요한 복구 작업을 수행한다. 이러한 방식의 복구를 Delayed Recovery 라고 한다.

GOLDILOCKS LITE에서 트랜잭션은 다음과 같은 방식으로 처리된다.

- 세션은 instance segment에 자신의 정보와 트랜잭션 정보를 기록한다.
- Lock이 필요한 자원에는 해당 세션의 정보를 기록한다.

Delayed recovery는 다음과 같이 감지된다.

- Lock 대상에 기록된 세션 (transaction ID)을 기준으로, 해당 트랜잭션이 정상적으로 종료되었는지 확인한다.
- 현재 lock을 점유하고 있는 세션의 유효성을 판단한다.

Delayed recovery가 필요한 경우, 다음과 같은 작업을 수행한다.

- 유효하지 않은 세션이 Instance 영역에 기록한 transaction log를 기반으로 데이터 복구, lock 해제, 자원 해제를 진행한다.
- 유효하지 않은 세션이 사용한 instance 영역을 해제한다.
- 이후 현재 세션은 자신의 트랜잭션을 정상적으로 진행한다.

> 노트
> 복구가 불가능한 경우에는 사용자에게 에러 메세지를 반환하고, 대상 테이블을 복구할 수 있는 방법을 제공한다. 자세한 내용은 **alter system refine [TableList]**을 참조한다.

Disk Logging Concept

GOLDILOCKS LITE는 디스크 로깅 기능을 제공한다. 동작 방식에 따라 다음과 같다.
* Parallel Logging Per Session : 세션 별로 각자의 로그파일에 기록하는 방식
* Log Cache (Redo Log buffer) : 공유되는 메모리 영역에 순차적으로 기록한 후 별도의 프로세스에 의해 로그파일에 기록하는 방식

디스크 로깅을 통해 사용자는 메모리 손실/장애에 대한 복구가 가능하다.
별도로 제공되는 체크포인트 수행을 통해 데이터파일을 생성한 후 메모리를 복원할 수 있다.

(1.6 복구 가이드 참조)

> 노트
> 디스크 로깅을 사용하는 경우 메모리 대비 성능감소를 고려해야 한다.

Replication Concept

GOLDILOCKS LITE는 network replication 방식으로 데이터를 동기화한다.

Network replication 방식의 특징은 다음과 같다.

- Master-slave 구조를 기반으로 한다.
- Commit 시점에 application이 트랜잭션 로그를 전송하는 방식이다.
- 설정에 따라 SYNC/ASYNC 방식을 선택할 수 있다.
  - SYNC 방식에서는 application이 commit을 수행할 때, 모든 slave에 반영이 완료되고 나서야 commit 이 완료된다.
  - ASYNC 방식에서는 application이 commit을 수행할 때, replication 전송 버퍼에 로그가 적재되면 commit 이 완료된 것으로 처리된다.
- Slave 측에서는 dbmReplica process를 기동해야 한다.
- 테이블 단위 이중화를 지원한다. (자세한 내용은 **create replication** 을 참조한다.)
- 모든 이중화 대상 테이블에는 unique index가 있어야 한다.
- 네트워크 장애가 발생하면 master는 미전송 로그를 파일로 저장한다. (자세한 내용은 **alter system replication sync** 를 참조한다.)
- Replication conflict가 발생하면 System Commit Number (SCN) 기준으로 가장 최신 데이터를 선택하여 데이터 정합성을 유지한다.

Replication 환경에서는 데이터 동기화 과정에서 데이터 간 불일치를 해소할 수 없는 상황이 발생할 수 있으며, 이러한 상태를 replication conflict 상태라고 정의한다.
GOLDILOCKS LITE는 이러한 상황을 다음과 같은 방식으로 처리한다.

Replication conflict 유형	Resolution on slave
Insert conflict • Duplicated record	Slave의 데이터가 정상인 경우에는 conflict 오류만 기록된다.
Update conflict • Not found	• 데이터가 존재할 경우, SCN이 더 높은 데이터를 기준으로 정합성을 맞춘다. • 데이터가 존재하지 않을 경우, 현재 로그를 기반으로 데이터를 삽입한다.
Delete conflict • Not found	Slave에서 데이터를 찾을 수 없는 경우, conflict 오류만 기록된다.

Application이 직접 이중화 전송을 수행할 때, network 전송 지연으로 인해 트랜잭션 순서가 뒤바뀔 수 있다. 예를 들어, master에서는 트랜잭션 TX-1 (insert → commit) 이후 TX-2 (update → commit) 순서로 수행되었지만, 장애/복구과정에서는 slave에는 Tx2가 먼저 전달되고 그 다음에 Tx1이 전달될 수 있다. 이러한 경우 데이터 동기화는 위의 conflict 정책을 통해 처리된다.

> 주의
> Slave에서는 데이터 변경/조회를 권장하지 않는다.

이중화 운영 중 network 장애가 발생하면, master 쪽의 application들은 전송해야 할 이중화 로그를 DBM_REPL_UNSENT_DIR에 지정된 경로에 파일 형태로 저장한다. 사용자는 network 장애가 복구된 후 "ALTER REPLICATION SYNC" 명령을 통해 미전송 로그를 전송하여 데이터를 동기화 할 수 있다.

1.2 Quick Start

GOLDILOCKS LITE의 설치와 기본 사용법에 대해 설명한다.

설치 전 작업

/dev/shm 공간 설정

Posix 방식의 shared memory를 이용함으로 /dev/shm에 충분한 공간이 확보되어야 한다.

커널 파라미터 설정

일부 Linux 커널 버전에서는 IPC 자원이 자동으로 삭제될 수 있어 이를 방지하기 위해 "RemoveIPC" 를 설정한다.

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

환경 변수 및 프로퍼티

GOLDILOCKS LITE를 사용하려면 사용자가 $DBM_HOME/conf/dbm.cfg 또는 사용자 환경 변수를 설정해야 한다.
(각 속성의 의미는 아래 표에 명시되어 있으며, 환경 변수가 설정 파일보다 우선 적용된다.)

환경 변수	설명
DBM_HOME	GOLDILOCKS LITE가 설치된 경로이다.
DBM_INSTANCE	사용할 Default Instance Name을 지정한다.
PATH	실행 파일을 수행할 수 있도록 사용자 환경 변수인 PATH에 $DBM_HOME/bin 을 추가한다.
LD_LIBRARY_PATH	실행 파일의 shared library를 탐색할 수 있도록 사용자 환경 변수인 LD_LIBRARY_PATH에 $DBM_HOME/lib 를 추가한다.
DBM_SHM_PREFIX	/dev/shm의 shared memory segment를 생성할 때 파일명의 접두어로 사용한다.
DBM_SHM_DIR	kernel버전에 따라 /dev/shm 하위의 디렉토리 생성이 허용될 경우 사용한다. (default(0): 사용하지 않음, (1): DBM_SHM_PREFIX 명으로 디렉토리 생성)

프로퍼티 이름	설명	옵션	Default 값	create instance 후 변경 가능 여부
DBM_DISK_LOG_ENABLE	DISK mode 사용 여부를 설정한다.	[0 | FALSE] = disable [1 | TRUE] = enable	0 /FALSE	X
DBM_DISK_LOG_DIR	DISK mode를 사용할 경우, 트랜잭션 로그 파일이 저장될 경로이다.		${DBM_HOME}/wal	X
DBM_DISK_LOG_FILE_SIZE	트랜잭션 로그 파일의 크기를 지정한다. 지정된 크기를 초과하면, 자동으로 다음 파일이 생성되어 트랜잭션 로그가 이어서 기록된다.	예시 1024M(1024megabyte) 1G (1gigabyte)	100M	X
DBM_DISK_DATA_FILE_DIR	DISK mode에서 CheckPoint가 수행될 때, 데이터 파일이 저장되는 경로이다.		${DBM_HOME}/dbf	X
DBM_DIRECT_IO_ENABLE	DIRECT I/O 사용 여부를 설정한다.	[0 | FALSE] = disable [1 | TRUE]= enable	[0 | FALSE]	X
DBM_DIRECT_IO_SIZE	DIRECT I/O를 사용할 수 있도록 disk의 sector size를 설정한다.	예시 512(byte)	512	X
DBM_LISTEN_PORT	dbmListener를 이용하여 원격으로 접속할 때 사용할 port를 지정한다.	예시 27584	27584	O
DBM_LISTEN_CONN_TIMEOUT	dbmListener와 연결되기까지의 timeout 시간을 설정한다.	예시 10000(ms)	10000	O
DBM_LISTEN_RECV_TIMEOUT	dbmListener에 요청을 전송한 후, 응답을 기다리는 시간을 설정한다.	예시 10000(ms)	10000	O
DBM_LOG_CACHE_MODE	Log cache 모드를 설정한다.	• 0: 사용 안 함 • 1:NVDIMM 사용	0	X

프로퍼티 이름	설명	옵션	Default 값	create instance 후 변경 가능 여부
		• 2:Shared memory 사용		
DBM_LOG_CACHE_SIZE	Log cache 크기를 지정한다.	예시 1G	1G	X
DBM_LOG_CACHE_FLUSH_INTERVAL	dbmLogFlusher의 동작 주기이다.	예시 3000(ms)	3000	O
DBM_DISK_COMMIT_WAIT	Commit 수행 시, redo 파일이 완전히 기록될 때까지 대기하는 모드를 설정한다. (이 기능은 데이터 안전성을 높일 수 있지만 성능 저하를 유발한다.)	[0 | FALSE] = disable [1 | TRUE]= enable	[0 | FALSE]	O
DBM_ARCHIVE_ENABLE	CheckPoint가 수행된 트랜잭션 로그 파일을 archive로 저장할지 여부를 설정한다.	[0 | FALSE] = disable [1 | TRUE] = enable	[0 | FALSE]	X
DBM_ARCHIVE_PATH	Archive로 저장할 디렉토리 경로이다.		${DBM_HOME}/arch	X
DBM_REPL_ENABLE	이중화 기능 사용 여부를 설정한다.	[0 | FALSE] = disable [1 | TRUE] = enable	[0 | FALSE]	X
DBM_REPL_ASYNC_DML	Async 이중화 사용 여부를 설정한다. 이중화 버퍼 크기만큼 트랜잭션을 모아서 일괄 전송된다. 이 방식은 전송 도중 장애가 발생할 경우 일부 트랜잭션 데이터가 동기화되지 않은 상태로 남을 수 있다.	[0 | FALSE] = disable [1 | TRUE] = enable	[0 | FALSE]	O
DBM_REPL_TARGET_PRIMARY_IP	dbmReplica가 구동하는 node의 IP를 설정한다.		127.0.0.1	O
DBM_REPL_T	dbmReplica의 listen port를 설정한다.(어플리케이션이 참조하는 값)		27584	O

프로퍼티 이름	설명	옵션	Default 값	create instance 후 변경 가능 여부
ARGET_PORT				
DBM_REPL_LISTEN_PORT	dbmReplica의 listen port를 설정한다.(dbmReplica가 참조하는 값)		27584	O
DBM_REPL_CONN_TIMEOUT	이중화 연결을 시도할 때의 timeout을 지정한다.	예시 10000(ms)	10000	O
DBM_REPL_RECV_TIMEOUT	이중화 동작 중에 응답 수신을 기다리는 시간이다.	예시 10000(ms)	10000	O
DBM_REPL_UNSENT_LOG_DIR	이중화 연결이 단절된 경우, 전송되지 않은 트랜잭션 로그를 임시로 저장할 디렉토리를 지정한다.		${DBM_HOME}/repl	X
DBM_REPL_UNSENT_LOGFILE_SIZE	미전송 트랜잭션 로그의 파일 크기를 지정한다.	예시 1024M(1024megabyte)	100M	X
DBM_NO_INDEX_ERROR_AT_PREPARE	dbmPrepareTable과 dbmPrepareTableHandle를 호출할 때, 대상 테이블에 index가 존재하지 않을 경우 오류를 반환할지 여부를 설정한다.	[0 | FALSE] = disable [1 | TRUE] = enable	[1 | TRUE]	O
DBM_LOADING_THREAD_COUNT	Startup, recovery 시 데이터를 로딩하는 thread의 개수를 지정한다.	1 이상의 수	8	O
DBM_PERF_ENABLE	Session의 acitivity를 count 할지 여부를 설정한다. • DML/DCL 수행 횟수 • Index operation 횟수 • Lock retry 횟수 • Delayed recovery 수행 횟수 관련된 내용은 V$SESS_STAT을 통해 조회 가능	[0 | FALSE] = disable [1 | TRUE] = enable	[0 | FALSE]	O
DBM_TRACE_LOG_FOR_PBT	상세한 trace log가 필요한 경우에 설정한다. (활성화하면 성능이 저하될 수 있다.)	[0 | FALSE] = disable [1 | TRUE] = enable	[0 | FALSE]	O
DBM_MVCC_ENABLE	조회 연산의 Lock 대기 여부를 설정한다. 변경 중인 레코드에 조회가 수행되어야 할 경우 이 설정에 따라 직전 commit된 데이터를 조회할 수 있다.	[0 | FALSE] = disable	[1 | TRUE]	

프로퍼티 이름	설명	옵션	Default 값	create instance 후 변경 가능 여부
		[1 | TRUE] = enable		O

다음은 bash 환경에서 환경 변수를 설정하는 예이다.

- GOLDILOCKS LITE 기본 설정

```
export DBM_HOME=/home/lim272/dbm_home     ## 설치한 경로
export PATH=${DBM_HOME}/bin:$PATH:.
export LD_LIBRARY_PATH=${DBM_HOME}/lib:$LD_LIBRARY_PATH:.
```

- 프로퍼티를 환경변수로 설정하는 예

```
export DBM_DISK_LOG_ENABLE=1
export DBM_DISK_LOG_DIR=${DBM_HOME}/wal
export DBM_DISK_DATA_FILE_DIR=${DBM_HOME}/dbf
```

설치 및 라이센스

본 절에서는 설치 방법과 라이센스 발급에 관한 내용을 기술한다.

설치

설치파일은 보통 다음과 같은 압축파일로 제공되기 때문에 사용자가 설정한 $DBM_HOME 경로에서 압축을 해제하면 설치가 완료된다.

패키지의 각 항목의 의미는 아래와 같다.

```
goldilocks_lite-3.2.rev6762-linux4.18.0-305.3.1.el8.x86_64.nopmem.noflt128-debug.tar.gz
* goldilocks_lite-3.2 : Product Major Version
* rev???? : Patch Version
* linux4.18.0.-305.3.1.el8 : Linux version (el8 호환)
* X86_64 : CPU / 64 bit
* nopmem : NVDIMM 지원 여부
* noflt128 : float128 지원 여부
* debug : debug / release 여부
```

패키지는 압축파일로 다음과 같이 해제한다.

```
shell> tar -xzf goldilocks_3.1.1.tar.gz
```

압축을 해제한 후 설치된 각 경로의 의미는 다음과 같다.

경로 이름	설명
arch	Archive log가 저장되는 기본 경로이다.
bin	각종 유틸리티 바이너리가 위치한 경로이다.
lib	API shared library 위치이다.
include	API header 파일 위치이다.
trc	Trace log가 저장되는 경로이다.
sample	API를 활용한 sample code 이다.
conf	dbm.cfg와 dbm.license 파일이 위치하는 곳이다.
dbf	Disk mode로 운영할 때 datafile이 저장되는 기본 경로이다.
repl	Replication mode로 운영 중 미전송 로그가 발생했을 때, 로그파일이 저장되는 기본 경로이다.
wal	Redo logfile이 저장되는 기본 경로이다.

bin 디렉토리의 각 binary는 각각 다음과 같은 기능을 수행한다.

Binary 이름	설명
dbmMetaManager	SQL/Internal Command를 실행할 수 있는 interactive 유틸리티 이다.
dbmListener	원격으로 접속할 경우, 타켓서버에서 구동해야 하는 유틸리티 이다.
dbmLogFlusher	Log Cache를 redo logfile로 flush 하는 유틸리티 이다.
dbmMonitor	현재 DB 상태를 모니터링 하는 유틸리티 이다.
dbmExp	Object 생성 script와 구분자를 포함하여 데이터를 추출하는 유틸리티 이다.
dbmImp	구분자를 가진 파일의 데이터를 적재하는 유틸리티 이다.
dbmErrorMsg	Error code를 출력하는 유틸리티 이다.
dbmCkpt	디스크 로깅방식에서 생성되는 redo logfile을 적용, 데이터 파일을 생성하는 유틸리티 이다.
dbmReplica	이중화 운영 모드에서 데이터를 수신/반영하는 유틸리티이다.
dbmDump	메모리 세그먼트와 파일을 추적하는 유틸리티 이다. 트랜잭션, 테이블, 인덱스, 데이터 파일, 로그 파일, anchor 파일 등의 정보를 출력할 수 있다.

라이센스

테스트 장비의 CPU core 개수와 hostname을 첨부하여 technet@sunjesoft.com 으로 라이선스를 요청해야 한다.

시작하기

dbmMetaManager를 구동한 후 initdb 명령을 수행하여 dictionary instance를 생성해야 한다. (자세한 내용은 **DICTIONARY**를 참조한다.)

```
[lim272@localhost 4th_iter]$ dbmMetaManager
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(unknown)> initdb;
success
```

> 노트
> Dictionary instance 이름은 "dict" 이다.

Dictionary instance에 접속한 상태에서만 사용자 instance를 생성할 수 있다.

```
dbmMetaManager(unknown)> set instance dict;
success
dbmMetaManager(DICT)> create instance demo;
success
dbmMetaManager(DICT)> set instance demo;
success
dbmMetaManager(DEMO)>
```

다음과 같이 사용자 Instance에서 table과 index를 생성하여 사용할 수 있다.

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 int);
success
dbmMetaManager(DEMO)> create unique index idx_t1 on t1 (c1);
success
dbmMetaManager(DEMO)> insert into t1 values (1, 1);
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> update t1 set c2 = 100 where c1 = 1;
1 row updated.
dbmMetaManager(DEMO)> select * from t1;
```

```
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 100
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> delete from t1;
1 row deleted.
dbmMetaManager(DEMO)> select * from t1;
--------------------------------------------------------------------------------
0 row selected
dbmMetaManager(DEMO)> rollback;
success
dbmMetaManager(DEMO)> select * from t1;
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 1
--------------------------------------------------------------------------------
1 row selected
```

1.3 구문

GOLDILOCKS LITE에서 사용 가능한 구문들을 기술한다.

Data Definition Language (DDL)

Instance, table, index 등과 같은 각 object 들을 생성하고 제거하는 구문들이다.
DDL에 사용되는 각 object의 이름의 길이는 최대 64 byte로 제한되며 반드시 문자로 시작해야 한다.

Table을 생성할 때 record 하나의 최대 크기는 1048247 byte이다.
Queue를 생성할 때 message의 최대 크기는 1048208 byte이다.
Index를 생성할 때 key columns의 최대 합산 크기는 1024 byte 이다.

> 주의
>
> - DDL 간에는 instance lock으로 동시성을 보장한다.
> - DDL 수행 시 commit/rollback 되지 않은 트랜잭션이 존재할 경우 수행되지 않는다.
> - DDL과 DML 간에는 동시성이 보장되지 않는다.
>   - DDL은 데이터가 변경되지 않는 상황에서 사용하는 것을 권장한다.

initdb

기능

GOLDILOCKS LITE를 처음 사용하기 위해 dictionary instance를 생성하는 명령이다.
생성된 dictionary instance의 이름은 "DICT" 이고 set instance 구문으로 접근할 수 있다.

구문

```
<initdb> ::= initdb
             ;
```

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(unknown)> initdb;
success
dbmMetaManager(unknown)> set instance dict;
success
dbmMetaManager(DICT)>
```

노트

- 이 명령으로 생성되는 object에 대한 설명은 **DICTIONARY**를 참조한다.
- DICT instance에서는 DICT_INST 테이블만이 유효한 정보를 가지고 있다. (기타 object 들은 deprecated 예정)

create instance

기능

사용자 Instance를 생성한다. 사용자 Instance를 생성한 후 테이블 등의 Object를 생성할 수 있다.
사용자 Instance는 Dictionary instance (DICT)에 접속한 상태에서만 생성하거나 제거할 수 있다.

노트

- 사용자 Instance 공간은 세션정보의 기록 및 트랜잭션 정보의 공간으로 사용된다.
- 세션의 최대 접속 가능한 개수는 1023개이다.

구문

```
<create instance> ::= CREATE INSTANCE instance_name
                      [ init <size> ]
                      [ extend <size> ]
                      [ max <size> ]
                      ;
```

- instance_name: 사용자 지정 이름이다.
- init <size>: 최초로 할당할 undo space의 크기를 지정한다. (한 개당 size 단위는 1M 이다.)
- extend <size>: init 된 공간이 모두 사용되어 확장될 때의 크기이다.
- max <size>: 최대로 확장할 수 있는 크기이다.

노트

Instance의 공간 1개는 1M로 사용자가 수행하는 트랜잭션의 이력을 저장하며 변경 연산을 수행할 때 롤백을 위한 Undo Image가 저장된다. 공간이 부족할 경우 자동으로 세션 내에 할당되며 최대 사용자가 지정한 MAX까지 확장된다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(unknown)> set instance dict;
success
dbmMetaManager(DICT)> create instance demo init 1024 extend 1024 max 10240;
success
dbmMetaManager(DICT)> select * from DIC_INST;
-------------------------------------------------------------------------------
INST_NAME   : DICT
INIT_SIZE   : 128
EXTEND_SIZE : 128
MAX_SIZE    : 1048576
-------------------------------------------------------------------------------
INST_NAME   : DEMO
INIT_SIZE   : 1024
EXTEND_SIZE : 1024
MAX_SIZE    : 10240
-------------------------------------------------------------------------------
2 row selected
```

노트

Instance를 생성하는 시점에 DBM_DISK_DATA_FILE_DIR 속성에 지정된 경로에 Dictionary table에 대한 datafile을 생성한다. 테이블의 생성/삭제/변경이 발생하면 메모리뿐만 아니라 datafile에도 내용을 반영한다. 이렇게 저장된 datafile은 디스크 모드에서 "startup" 복구를 수행할 때 복구해야 할 테이블 목록을 구성하는

과정에서 반드시 필요하기 때문에 datafile이 유실되지 않도록 주의가 필요하다.

노트

Disk mode로 설정한 경우, CREATE INSTANCE 시점에 DBM_DISK_LOG_DIR 에 설정된 경로에 <instance_name.anchor> 파일과 redo logfile이 생성되기 시작한다. Anchor file은 redo logfile의 메타 정보를 관리하므로 손상되지 않도록 주의해야 한다.

create table

기능

사용자 테이블을 생성한다. 테이블은 instance의 하위 개념이다.
Dictionary instance에는 사용자가 테이블을 생성할 수 없다.

구문

```
<create table> ::= CREATE [TABLE_TYPE] table_name
                       (
                          column_definition [, ...]
                       )
                       [ init <size> ]
                       [ extend <size> ]
                       [ max <size> ]
                       ;
```

- TABLE_TYPE ::= TABLE
  | DIRECT
  | SPLAY
- table_name: 사용자 지정 테이블 이름이다.
- column_definition ::= column_name data_type_definition
- column_name: 사용자 지정 column 이름이다.
- data_type_definition ::= short
  | int
  | long
  | float
  | double
  | char (size)
  | date

- init <size>: 최초로 할당될 레코드의 수를 의미한다. (Default: 1,024개)
- extend <size>: init 된 공간이 모두 사용된 뒤 공간 확장될 때 저장할 수 있는 row의 개수이다. (Default: 102,400 개)
- max <size>: 최대로 확장할 수 있는 row 개수이다. (Default: 4,096,000 개)

GOLDILOCKS LITE의 data type은 크기는 아래와 같다.

Data type	C type과 크기 (64 bit OS 기준)
short	short을 의미하며 2 byte 이다.
int	int를 의미하며 4 byte 이다.
long	long을 의미하며 8 byte 이다.
float	float을 의미하며 4 byte 이다.
double	double을 의미하며 8 byte 이다.
char	char와 동일하며 fixed size이다.
date	8 byte이며 사용자 구조체의 멤버 변수는 8byte 크기로 선언해야 한다.

> 노트
> 각 column의 offset은 c 언어의 struct의 default padding/packing 방식을 사용한다.

사용 예

```
**************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
**************************************************************
dbmMetaManager(DICT)> set instance demo;
success
dbmMetaManager(DEMO)> create table t1
                    2 (
                    3   c1 short,
                    4   c2 int,
                    5   c3 long,
                    6   c4 float,
                    7   c5 double,
                    8   c6 char(20),
                    9   c7 date
                   10 );
success
dbmMetaManager(DEMO)> desc t1;
--------------------------------------------------------------------------------
```

```
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(64)
-------------------------------------------------------------------------------
C1                              short               2          0
C2                              int                 4          4
C3                              long                8          8
C4                              float               4          16
C5                              double              8          24
C6                              char                20         32
C7                              date                 8         56
-------------------------------------------------------------------------------
Any index not created
-------------------------------------------------------------------------------
success
```

노트

테이블 형상에서 보여주는 (컬럼명, 데이터 타입) 이후의 항목은 각각 크기를 의미하는 byte수와 레코드내의 시작위치를 의미한다.

create index

기능

테이블에 속한 index를 생성한다.
* Index 생성 개수에는 제한이 없으나 삽입 성능에 영향을 준다.
* Key column size의 합이 1024 byte를 넘을 수 없다.
* Index key column은 int, short, long, char 네 가지 데이터 타입만 허용된다.
* Direct, Splay 테이블은 Index key column의 지정 용도로 이 구문을 사용한다.

구문

```
<create index> ::= CREATE [UNIQUE] INDEX index_name ON table_name
                   ( column_name <ordering> [, ... ] )
                   ;
```

- UNIQUE를 지정하면 동일한 Key 저장을 허용하지 않는다. (NULL저장 가능)
- index_name: 사용자 지정 index 이름이다.
- table_name: Index를 생성할 대상 table 이름이다.
- column_name: Key column으로 사용될 column 이름이다.
- ordering : [ASC | DESC] (정렬 순서를 명시한다. 지정하지 않는 경우 ASC로 설정된다.)

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> create unique index idx1_t1 on t1 (c1);
success
dbmMetaManager(DEMO)> create index idx2_t1 on t1 (c1, c2);
success
dbmMetaManager(DEMO)> desc t1;
--------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(64)
--------------------------------------------------------------------------------
C1                              short               2         0
C2                              int                 4         4
C3                              long                8         8
C4                              float               4         16
C5                              double              8         24
C6                              char                20        32
C7                              date                 8        56
--------------------------------------------------------------------------------
IDX1_T1                         unique     (C1 asc)
IDX2_T1                                    (C1 asc, C2 asc)
--------------------------------------------------------------------------------
success
```

노트

- 모든 테이블에는 하나 이상의 unique index가 반드시 존재해야 한다.
- Application은 API를 통해 사용할 index를 지정할 수 있다. (자세한 내용은 **dbmSetIndex**를 참조한다.)

create queue

기능

First-In/ First-Out (FIFO) 동작을 하는 queue 형태 테이블을 생성한다.

구문

```
<create queue> ::= CREATE QUEUE queue_name SIZE msg_size
                       [ init <size> ]
                       [ extend <size> ]
                       [ max <size> ]
                   ;
```

- queue_name: 대상 queue table 이름이다.
- msg_size: 테이블에 저장될 메시지의 최대 크기이다.
- init <size>: 최초로 저장 가능한 row 개수이다. (Default: 1,024개)
- extend <size>: init된 공간이 모두 사용되어 확장될 때 저장 가능한 row 개수이다. (Default: 102,400개)
- max <size>: 최대로 확장할 수 있는 row 개수이다. (Default: 4,096,000개)

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> create queue que1 size 1024;
success
dbmMetaManager(DEMO)> desc que1;
-------------------------------------------------------------------------------
Instance=(DEMO) Table=(QUE1) Type=(QUEUE) RowSize=(1056)
-------------------------------------------------------------------------------
PRIORITY                        int                 4          4
ID                              long                8          8
MSG_SIZE                        int                 4          16
IN_TIME                         date                8          24
MESSAGE                         char                1024       32
-------------------------------------------------------------------------------
IDX_QUE1                        unique     (PRIORITY asc, ID asc)
-------------------------------------------------------------------------------
success
```

- message의 ID는 enqueue 시점에 채번된다. (commit 시점이 아님)
- Dequeue는 Enqueue를 수행하는 세션이 커밋한 순서로 가져가나 동시에 커밋되는 경우 Message의 ID가 먼저인 경우를 읽게 된다.
  - A세션이 ID(1)을 채번하고 B세션이 ID(2)를 채번한 후 B세션이 먼저 커밋할 경우 Dequeue는 ID(2)인 데이터를 먼저 읽는다.
  - A, B 세션이 커밋한 상태에서 Dequeue가 수행되면 ID(1)인 데이터를 먼저 읽는다.
- Enqueue로 삽입된 데이터는 Commit이 완료된 이후 조회하거나 dequeue 할 수 있다.

    - Enqueue를 수행한 세션 역시 commit을 완료해야 자신이 Enqueue한 데이터를 dequeue 할 수 있다.
- Dequeue로 한 건을 가져온 이후 commit 하지 않는 세션이 존재하더라도 다른 dequeue를 수행하는 세션이 이를 기다리지 않고 다음 데이터를 dequeue 한다.

노트

- Queue를 생성할 때 메시지 크기는 사용자 입력값을 8바이트 단위로 정렬 (align)하여 생성한다. 따라서 API를 사용할 때 사용자 저장 변수의 크기는 테이블의 "MESSAGE" column 크기를 참고하여 할당해야 한다.
- Queue 테이블은 생성 시 자동으로 인덱스가 생성되며, 사용자가 임의로 조작하거나 변경할 수 없다.

create store

기능

String 또는, binary형태의 Key/value로 데이터를 저장/변경/조회하는 테이블을 생성한다.

구문

```
<create store> ::= CREATE STORE store_name KEY key_size VALUE value_size
                       [ init <size> ]
                       [ extend <size> ]
                       [ max <size> ]
                       ;
```

- store_name: 대상 store table의 이름이다.
- key_size: Key의 size 이다.
- value_size: Value의 size 이다.
- INIT <size>: 최초로 저장 가능한 record 개수이다. (Default: 1,024개)
- EXTEND <size>: init 된 공간이 모두 사용되어 확장될 때 저장 가능한 record 개수이다. (Default: 102,400개)
- MAX <size>: 최대로 확장할 수 있는 record 개수이다. (Default: 4,096,000개)

사용 예

```
************************************************************
* Copyright 2010. SUNJESOFT Inc. All rights reserved.
* Version (Debug 3.2-3.2.6 revision(6743))
* warning : open file limit 1024 is too low. : recommended 65536 or higher
************************************************************
```

```
dbmMetaManager(DEMO)> create store st1 key 32 value 1024;
success
dbmMetaManager(DEMO)> desc st1;
-------------------------------------------------------------------------------
Instance=(DEMO) Table=(ST1) Type=(STORE) RowSize=(1056) LockMode(1)
-------------------------------------------------------------------------------
ST_KEY                          char                32          0
ST_VALUE                        char                1024        32
-------------------------------------------------------------------------------
IDX_ST1                         unique      (ST_KEY asc)
-------------------------------------------------------------------------------
success
```

주의

- STORE 테이블의 column 이름을 임의로 변경하면 동작하지 않는다.
- STORE 테이블의 Key-Value 크기는 Fixed Size로 동작한다. (가변형 미지원)

create sequence

기능

Sequence 객체를 생성한다.

구문

```
<create sequence> ::= CREATE SEQUENCE sequence_name [options]
                    ;
<options> ::= START WITH <value>
              ¦ INCREMENT BY <value>
              ¦ MAXVALUE <value>
              ¦ MINVALUE <value>
              ¦ CYCLE ¦ NOCYCLE
```

- sequence_name: Sequence 이름이다.
- Sequence 생성 시점의 각 옵션들은 생략할 수 있다.
- <START WITH>에 설정된 값은 <MAXVALUE>를 초과하여 생성할 수 없다.
- <INCREMENT BY>를 생략할 경우 default는 1로 설정된다.

- <MINVALUE>를 생략할 경우 default는 0으로 설정된다.
- <MAXVALUE>를 생략할 경우 default는 LONG_MAX 값으로 설정된다.
- <CYCLE> 옵션을 지정했을 때 sequence의 current value가 MaxValue를 넘어서면 MinValue로 설정된다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> create sequence seq1 start with 10 increment by 2 maxvalue 15 cycle;
success
dbmMetaManager(DEMO)> select seq1.nextval from dual;
--------------------------------------------------------------------------------
NEXTVAL : 10
--------------------------------------------------------------------------------
```

> 주의
> Sequence는 트랜잭션 로깅 및 이중화가 되지 않는다. 따라서, Fail over등으로 서비스 재시작 전에 사용자가 적절하게 값을 변경하여 사용 해야 한다. (**alter sequence [currval]** 참조)

create user_type

기능

사용자 데이터의 형식을 생성한다. char 컬럼에 또다른 구조의 데이터를 저장하는 경우 해당 데이터를 출력 / 표현하기 위한 용도이다.

구문

```
<create type> ::= CREATE USER_TYPE type_name
                    (
                       column_definition [, ...]
                    )
                    ;
```

- type_name: 사용자 지정 형식 이름이다.
- column_definition ::= column_name data_type_definition
- column_name: 사용자 지정 column 이름이다.
- data_type_definition ::= short

| int
| long
| float
| double
| char (size)
| date

사용 예

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 char(100));
success
dbmMetaManager(DEMO)> insert into t1 values (1, 'ABCEFGHIJKLMN');
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> create user_type u1 (c1 char(3), c2 char(4));
success
dbmMetaManager(DEMO)> select user_type(c2, u1) from t1;
--------------------------------------------------------------------------------
USER_TYPE : C1=ABC C2=EFGH
--------------------------------------------------------------------------------
1 row selected
```

노트
USER_TYPE (**USER_TYPE(Column_Name, Type_Name)**참조) 함수를 이용하여 데이터를 사용자가 정의한 형식으로 변환하여 출력할 수 있다.

drop index

기능

지정된 index를 제거한다.

구문

```
<drop index> ::= DROP INDEX index_name
                 ;
```

index_name: 제거 대상 index의 이름이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> desc t1;
-------------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(64)
-------------------------------------------------------------------------------------
C1                              short               2         0
C2                              int                 4         4
C3                              long                8         8
C4                              float               4         16
C5                              double              8         24
C6                              char                20        32
C7                              date                 8        56
-------------------------------------------------------------------------------------
IDX1_T1                         unique     (C1)
IDX2_T1                                    (C1, C2)
-------------------------------------------------------------------------------------
success
dbmMetaManager(DEMO)> drop index idx2_t1;
success
dbmMetaManager(DEMO)> desc t1;
-------------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(64)
-------------------------------------------------------------------------------------
C1                              short               2         0
C2                              int                 4         4
C3                              long                8         8
C4                              float               4         16
C5                              double              8         24
C6                              char                20        32
C7                              date                 8        56
-------------------------------------------------------------------------------------
IDX1_T1                         unique     (C1)
-------------------------------------------------------------------------------------
success
```

> 주의
> STORE, Queue table에 생성된 index를 임의로 조작 하는 것은 권장하지 않는다.

drop table (queue, store)

기능

사용자가 지정한 table (queue, store)과 해당 object에 생성된 하위 index object를 모두 제거한다.

구문

```
<drop table> ::= DROP TABLE table_name <force>
                 |
                 DROP QUEUE table_name
                 |
                 DROP STORE store_name
                 ;
```

table_name: 제거 대상 table (queue, store)의 이름이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> desc t1;
-------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(64)
-------------------------------------------------------------------------------
C1                              short               2         0
C2                              int                 4         4
C3                              long                8         8
C4                              float               4         16
C5                              double              8         24
C6                              char                20        32
C7                              date                 8        56
-------------------------------------------------------------------------------
IDX1_T1                         unique     (C1)
IDX2_T1                                    (C1, C2)
```

```
-------------------------------------------------------------------------------
success
dbmMetaManager(DEMO)> drop table t1;
success
dbmMetaManager(DEMO)>
```

노트

FORCE 옵션은 동작 이상이나 사용자 실수로 인해 테이블 정보가 Dictionary table에서 정상적으로 정리되지 않은 상태에서 강제로 수행하고자 할 때 지정한다.
FORCE 옵션으로도 제거되지 않은 세그먼트 파일이 /dev/shm에 존재 하는 경우 사용자가 직접 삭제해야 한다.

drop sequence

기능

사용자가 지정한 sequence object를 제거한다.

구문

```
<drop sequence> ::= DROP SEQUENCE <sequence_name>
                    ;
```

sequence_name: 제거 대상 sequence의 이름이다.

사용 예

```
dbmMetaManager(DEMO)> drop sequence seq10;
success
```

drop user_type

기능

사용자가 지정한 user_type object를 제거한다.

구문

```
<drop type> ::= DROP USER_TYPE <type_name>
                ;
```

type_name: 삭제할 type name이다.

사용 예

```
dbmMetaManager(DEMO)> drop user_type u1;
success
```

drop instance

기능

사용자가 지정한 instance와 모든 하위 object를 제거한다.
Dictionary instance (dict)에 접속한 상태에서만 수행 가능하다.

구문

```
<drop instance> ::= DROP INSTANCE instance_name [FORCE]
                ;
```

instance_name: 제거 대상 instance의 이름이다.
FORCE: 복구 과정 등 특정 상황에서 강제로 제거를 수행해야 할 경우, 이 옵션을 지정할 수 있다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> set instance dict;
success
dbmMetaManager(DICT)> drop instance demo;
success
dbmMetaManager(DICT)> set instance demo;
ERR-22008] fail to attach a shared memory segment
Command] <set instance demo>
```

노트
FORCE 옵션은 동작 이상이나 사용자 실수로 인해 메타 정보가 정상적으로 정리되지 않은 상태에서 인스턴스를 강제로 삭제할 때 사용한다. FORCE 옵션으로 정리되지 않은 경우 사용자가 직접 삭제해야 한다.

truncate table (queue, store)

기능

사용자가 지정한 테이블(queue, store)을 초기화한다. 테이블 내의 데이터와 extend 된 segment는 모두 제거되고 테이블 생성 시점에 지정된 init size로 재생성된다.

구문

```
<truncate table> ::= TRUNCATE TABLE table_name
                     ¦
                     TRUNCATE QUEUE table_name
                     ¦
                     TRUNCATE STORE store_name
                     ;
```

table_name: Truncate 할 table(queue, store)의 이름이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(unknown)> set instance demo;
success
dbmMetaManager(DEMO)> truncate table t1;
success
```

주의
TRUNCATE 동작은 기존의 shared memory segment를 삭제한 후 새로운 세그먼트를 생성하는 과정이다.(DROP->CREATE) 이 과정에서 현재 진행 중인 트랜잭션과 동시성은 보장되지 않으며 commit 과정에서 오류로 처리될 수도 있다. 따라서 애플리케이션에 오류가 반환되면, 애플리케이션을 종료하고 새로 시작하는 것을 권장한다.

compact table

기능

사용자가 지정한 테이블에 대해 compaction을 수행한다. 확장된 segment에 속한 데이터를 삭제하여도 해당 메모리는 OS로 반납되지 않는다. 이 구문은 segment의 사용하지 않는 메모리 공간을 반납해야 할 경우 사용할 수 있다.

> 주의
> 이 명령은 내부적으로 (데이터 export → truncate → 데이터 import) 과정을 수행하는 DDL이다. 따라서 application을 모두 종료한 후 수행해야 한다.

구문

```
<compact table> ::= ALTER TABLE table_name COMPACT
                    ;
```

table_name: Compact 할 table의 이름이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(unknown)> set instance demo;
success
dbmMetaManager(DEMO)> ALTER TABLE t1 COMPACT;
success
```

> 노트
> Compact 기능은 Btree 테이블만 허용된다.

add column

기능

사용자가 테이블에 column을 추가할 경우에 사용한다. 위치를 지정할 수 없으며 항상 마지막 column으로 추가된다. add column 기능은 디스크 모드에서 사용할 수 없고, Normal 테이블 타입인 경우에만 사용할 수 있다.

> 주의
> 이 명령은 내부적으로 (데이터 export → 테이블 재생성 → 데이터 import) 과정을 수행하는 DDL이다. 따라서 application을 모두 종료한 후 수행해야 한다.

구문

```
<add column> ::= ALTER TABLE table_name ADD COLUMN ( column_name datatype [default_value] )
                 ;
```

- table_name: Column을 추가할 table의 이름이다.
- column_name: 추가할 column의 이름이다.
- datatype: Column의 데이터 타입 이름이다. (int, short, long, date, char, float, double)
- default_value: date type을 제외한 데이터 타입의 기본값을 지정할 경우에 정의한다. (상수만 허용)

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 > 0
--------------------------------------------------------------------------------
C1                  : 1
C2                  : a
C3                  : 1
C4                  : a
--------------------------------------------------------------------------------
C1                  : 2
C2                  : b
C3                  : 2
C4                  : b
--------------------------------------------------------------------------------
C1                  : 3
C2                  : c
C3                  : 3
C4                  : c
--------------------------------------------------------------------------------
3 row selected
```

```
dbmMetaManager(DEMO)> alter table t1 add column (c5 char(10) default 'zz' )
success
dbmMetaManager(DEMO)> select * from t1 where c1 > 0
--------------------------------------------------------------------------------
C1                  : 1
C2                  : a
C3                  : 1
C4                  : a
C5                  : zz
--------------------------------------------------------------------------------
C1                  : 2
C2                  : b
C3                  : 2
C4                  : b
C5                  : zz
--------------------------------------------------------------------------------
C1                  : 3
C2                  : c
C3                  : 3
C4                  : c
C5                  : zz
--------------------------------------------------------------------------------
3 row selected
```

노트

add column이 수행되면 $DBM_HOME/trc 경로에 <instance name>_<table name>_<scn>.dat 형식의 백업 데이터 파일이 생성된다. add column 작업이 실패하여 복구가 필요한 경우, 새로운 테이블을 생성한 후 다음 명령어를 사용하여 복구할 수 있다.

```
dbmImp -i demo -t t1 -d DEMO_T1_3.dat -b
```

주의

add/drop column 기능은 디스크 로깅 모드에서 지원하지 않는다.

drop column

기능

사용자가 테이블에서 column을 제거할 경우에 사용한다. drop column 기능은 Normal 테이블 타입인 경우에만 사용할 수 있고 column이 index key로 사용 중일 경우에는 수행할 수 없다.

> 주의
> 이 명령은 내부적으로 (데이터 export → 테이블 재생성 → 데이터 import) 과정을 수행하는 DDL이다. 따라서 application을 모두 종료한 후 수행해야 한다.

구문

```
<drop column> ::= ALTER TABLE table_name DROP COLUMN column_name
                  ;
```

- table_name: Column을 삭제할 table의 이름이다.
- column_name: 삭제할 column의 이름이다.

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 > 0;
--------------------------------------------------------------------------------
C1                  : 1
C2                  : a
C3                  : 1
C4                  : a
--------------------------------------------------------------------------------
C1                  : 2
C2                  : b
C3                  : 2
C4                  : b
--------------------------------------------------------------------------------
C1                  : 3
C2                  : c
C3                  : 3
C4                  : c
--------------------------------------------------------------------------------
3 row selected
```

```
dbmMetaManager(DEMO)> alter table t1 drop column c2;
success
dbmMetaManager(DEMO)> select * from t1 where c1 > 0;
--------------------------------------------------------------------------------
C1                  : 1
C3                  : 1
C4                  : a
--------------------------------------------------------------------------------
C1                  : 2
C3                  : 2
C4                  : b
--------------------------------------------------------------------------------
C1                  : 3
C3                  : 3
C4                  : c
--------------------------------------------------------------------------------
3 row selected
```

노트

drop column이 수행되면 $DBM_HOME/trc 경로에 <instance name>_<table name>_<scn>.dat 형식의 백업 데이터 파일이 생성된다. drop column 작업이 실패하여 복구가 필요한 경우, 새로운 테이블을 생성한 후 다음 명령어를 사용하여 복구할 수 있다.

```
dbmImp -i demo -t t1 -d DEMO_T1_3.dat -b
```

주의

add/drop column 기능은 디스크 로깅 모드에서 지원하지 않는다.

rename column

기능

사용자가 테이블에서 column의 이름만 변경할 경우에 사용한다.

구문

```
<rename column> ::= ALTER TABLE table_name RENAME COLUMN org_column_name TO new_column_name
                    ;
```

- table_name: Column 이름을 변경할 대상 table의 이름이다.
- org_column_name: 기존 column 이름이다.
- new_column_name: 새로 사용할 column 이름이다.

사용 예

```
dbmMetaManager(DEMO)> desc t1
-------------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(12)
-------------------------------------------------------------------------------------
C1                              int                 4          0
C2                              int                 4          4
C3                              int                 4          8
-------------------------------------------------------------------------------------
success
dbmMetaManager(DEMO)> alter table t1 rename column c2 to x359
success
dbmMetaManager(DEMO)> desc t1
-------------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(12)
-------------------------------------------------------------------------------------
C1                              int                 4          0
X359                            int                 4          4
C3                              int                 4          8
-------------------------------------------------------------------------------------
success
```

노트

Store, Queue 테이블은 rename 명령을 지원하지 않는다.

create replication

기능

Replication 대상 테이블을 DIC_REPL_TABLE 목록에 등록한다.

구문

```
<create replication> ::= CREATE REPLICATION TABLE [TableList]
                  ;
<TableList> ::= TableName [, tableName]
```

사용 예

```
dbmMetaManager(DEMO)> create replication table t1, t2;
success
```

이중화 대상 테이블은 다음과 같이 조회할 수 있다.

```
dbmMetaManager(DEMO)> select * from DIC_REPL_TABLE;
--------------------------------------------------------------------------------
INST_NAME  : DEMO
TABLE_NAME : T1
--------------------------------------------------------------------------------
INST_NAME  : DEMO
TABLE_NAME : T2
--------------------------------------------------------------------------------
2 row selected
```

> 노트
> 동일 트랜잭션이라도 DIC_REPL_TABLE에 등록되어 있지 않은 테이블은 이중화로 전송되지 않는다.

alter replication

기능

Replication 대상 테이블을 추가/삭제한다.

구문

```
<alter replication> ::= ALTER REPLICATION [ADD¦DROP] TABLE [TableList]
                  ;
<ADD> : 테이블을 이중화 대상으로 추가
<DROP> : 테이블을 이중화 대상에서 제거
<TableList> ::= TableName [, tableName]
```

사용 예

```
dbmMetaManager(DEMO)> alter replication add table t1, t2;
success
dbmMetaManager(DEMO)> alter replication drop table t1, t2;
success
```

노트

alter replication add/drop 구문은 현재 실행 중인 애플리케이션에 영향을 주지 않는다. 적용을 위해 애플리케이션 재기동을 해야 한다.

alter system replication sync

기능

Master 측에서 다음 목적을 위해 사용한다.
- 미전송 로그를 slave로 전송한다.
- 특정 대상 테이블을 동기화 하기 위해 레코드 전체를 Slave로 전송한다.

Slave 측에서 다음 목적을 위해 사용한다.
- Slave에서 미전송 로그를 읽어 동기화를 수행한다. (이 때, slave가 미전송 로그파일에 접근할 수 있어야 한다.)

구문

```
<alter system replication> ::= ALTER SYSTEM REPLICATION SYNC
                               [LOCAL ¦ ALL ¦ TableList]
                               ;
<LOCAL> : Slave 측에서 미전송 로그를 읽어 반영 가능한 경우
<ALL> : Master 측에서 모든 이중화 대상 테이블의 데이터를 slave로 전송하고자 할 경우
<TableList> ::= 모든 데이터를 전송할 대상 TableName [, tableName]
```

옵션을 지정하지 않은 경우 미전송 로그를 전송하는 방식으로 동작한다.

미전송 로그가 생성되는 위치와 관련해서는 **환경 변수 및 프로퍼티**의 이중화 관련 사항을 참조한다.

사용 예

```
dbmMetaManager(DEMO)> alter system replication sync;
success
```

drop replication

기능

Replication 대상 테이블을 DIC_REPL_TABLE 목록에서 제거한다.

구문

```
<drop replication> ::= DROP RPELICATION
                       ;
```

사용 예

```
dbmMetaManager(DEMO)> drop replication;
success
```

set instance

기능

사용자가 지정한 instance로 전환한다.

구문

```
<set instance> ::= SET INSTANCE instance_name
                   ;
```

instance_name: 전환할 instance의 이름이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
```

```
************************************************************
dbmMetaManager(unknown)> set instance dict;
success
dbmMetaManager(DICT)> set instance demo;
success
dbmMetaManager(DEMO)>
```

노트

set instance 구문은 dbmMetaManager에서만 동작한다.

alter sequence [currval]

기능

사용자가 생성한 sequence의 값을 지정된 값으로 변경한다.

구문

```
<alter sequence> ::= ALTER SEQUENCE sequence_name SET CURRVAL = value
                 ;
```

- sequence_name: 대상 sequence의 이름이다.
- value: 사용자가 변경할 current value 이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> select seq1.currval from dual;
-------------------------------------------------------------------------------
CURRVAL              : 3
-------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> alter sequence seq1 set currval = 1000;
success
dbmMetaManager(DEMO)> select seq1.currval from dual;
-------------------------------------------------------------------------------
CURRVAL              : 1000
```

```
--------------------------------------------------------------------------------
1 row selected
```

alter system reset checkpoint

기능

체크포인트를 특정 로그 파일 번호부터 수행할 수 있도록 해당 로그 파일 번호를 설정한다.

구문

```
<reset perf> ::= ALTER SYSTEM RESET checkpoint <instanceName> <logfile_number>
                 ;
<instanceName> ::        ❶ 대상 instance 이름 입력
<logfile_number> ::      ❷ 특정 로그파일 번호
```

사용 예

```
dbmMetaManager(DEMO)> alter system reset checkpoint demo -1;
success
```

> 노트
> 시스템 장애로 데이터파일이 삭제되고 Archive logfile을 통해 복구가 가능한 환경이라면 체크포인트 과정을 통해 다시 데이터 파일을 만들 수 있다. 이때 이 구문을 통해 체크포인트 시작 파일 번호를 지정한다.

alter system reset perf

기능

DBM_PERF_ENABLE 속성이 활성화 되면 (vsys_stat, vsess_stat)등에 통계정보가 누적된다. 이 명령은 이 정보들을 초기화한다.

구문

```
<reset perf> ::= ALTER SYSTEM RESET PERF
                 ;
```

사용 예

```
dbmMetaManager(DEMO)> alter system reset perf;
success
```

alter system refine [TableList]

기능

특정 테이블이나 인덱스에 대한 잠금을 유지한 상태에서 애플리케이션이 비정상적으로 종료된 경우, 복구하기 위한 긴급 명령이다.

구문

```
<reset perf> ::= ALTER SYSTEM REFINE [TableName, ...]
                 ;
```

TableName: 특정 테이블에 대해서만 작업을 수행하고자 할 경우에 명시하는 옵션이다.

사용 예

```
dbmMetaManager(DEMO)> alter system refine;
success
```

> 주의
> 이 명령은 내부적으로 index를 rebuild 하는 DDL이므로, 모든 application을 종료한 후 수행해야 한다.

Data Manipulation Language(DML)

DML에는 데이터 삽입, 갱신, 삭제, 조회, queue table에 대한 enqueue 및 dequeue 구문이 포함된다.

insert

기능

사용자가 지정한 테이블에 데이터를 삽입한다.

구문

```
<insert> ::= INSERT INTO table_name
             [ column_name [, ...] ]
             VALUES
             ( value_expression [, ...] )
             ;
```

- table_name: 데이터를 저장할 대상 테이블이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> desc t1;
-------------------------------------------------------------------------------
Instance=(DEMO) Table=(T1) Type=(TABLE) RowSize=(48)
-------------------------------------------------------------------------------
C1                              int                 4         0
C2                              double              8         8
C3                              char                20        16
C4                              date                 8        40
-------------------------------------------------------------------------------
IDX_T1                          unique     (C1)
-------------------------------------------------------------------------------
success
dbmMetaManager(DEMO)> insert into t1 (c1, c2, c3, c4) values (1, 1, 1, sysdate);
success
dbmMetaManager(DEMO)> insert into t1 (c1, c2) values (2, 2);
success
dbmMetaManager(DEMO)> insert into t1 values (3, 3, 3, sysdate);
success
dbmMetaManager(DEMO)> select * from t1;
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 1.000000
C3                  : 1
C4                  : 2019/01/02 16:44:17.227558
-------------------------------------------------------------------------------
C1                  : 2
```

```
C2                  : 2.000000
C3                  :
C4                  : 1970/01/01 09:00:00.000000
-------------------------------------------------------------------------------
C1                  : 3
C2                  : 3.000000
C3                  : 3
C4                  : 2019/01/02 16:44:37.283193
-------------------------------------------------------------------------------
3 row selected
```

노트

unique index가 걸린 테이블에 동일 key값으로 insert가 발생할 경우 선행 트랜잭션이 종료될 때까지 후행 트랜잭션은 대기한다.

update

기능

사용자가 지정한 테이블에서 한 건 이상의 데이터를 갱신한다.

구문

```
<update> ::= UPDATE table_name
             SET column_name = value_expression [, ...]
             [ WHERE cond_expression ]
             ;
```

- table_name: 데이터를 갱신할 대상 테이블이다.
- column_name: 테이블 내의 column 이름이다.
- value_expression: 갱신할 value 이다.
- cond_expression: 갱신할 조건절이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> select * from t1;
```

```
-------------------------------------------------------------------------------
C1                    : 1
C2                    : 1.000000
C3                    : 1
C4                    : 2019/01/02 16:44:17.227558
-------------------------------------------------------------------------------
C1                    : 2
C2                    : 2.000000
C3                    :
C4                    : 1970/01/01 09:00:00.000000
-------------------------------------------------------------------------------
C1                    : 3
C2                    : 3.000000
C3                    : 3
C4                    : 2019/01/02 16:44:37.283193
-------------------------------------------------------------------------------
3 row selected
dbmMetaManager(DEMO)> update t1 set c2 = 100 where c1 >= 1;
3 row updated.
dbmMetaManager(DEMO)> select * from t1;
-------------------------------------------------------------------------------
C1                    : 1
C2                    : 100.000000
C3                    : 1
C4                    : 2019/01/02 16:44:17.227558
-------------------------------------------------------------------------------
C1                    : 2
C2                    : 100.000000
C3                    :
C4                    : 1970/01/01 09:00:00.000000
-------------------------------------------------------------------------------
C1                    : 3
C2                    : 100.000000
C3                    : 3
C4                    : 2019/01/02 16:44:37.283193
-------------------------------------------------------------------------------
3 row selected
```

노트

Update로 갱신된 레코드는 잠금 (lock) 되어 다른 세션의 접근을 차단한다.
다른 세션에서 select 할 경우 update 이전의 커밋된 데이터를 조회한다.
(단, DBM_MVCC_ENABLE = FALSE 로 설정된 경우에는 update가 완료될 때까지 대기한다.)

주의

Index key Column은 변경할 수 없다.

delete

기능

사용자가 지정한 테이블에서 한 건 이상의 데이터를 삭제한다.

구문

```
<delete> ::= DELETE FROM table_name
             [ WHERE cond_expression ]
             ;
```

- table_name: 데이터를 갱신할 대상 테이블이다.
- cond_expression: 갱신할 조건절이다.

사용 예

```
*************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
*************************************************************
dbmMetaManager(DEMO)> select * from t1;
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 1.000000
C3                  : 1
C4                  : 2019/01/02 16:44:17.227558
-------------------------------------------------------------------------------
C1                  : 2
C2                  : 2.000000
```

```
C3                  :
C4                  : 1970/01/01 09:00:00.000000
-------------------------------------------------------------------------------
C1                  : 3
C2                  : 3.000000
C3                  : 3
C4                  : 2019/01/02 16:44:37.283193
-------------------------------------------------------------------------------
3 row selected
dbmMetaManager(DEMO)> delete from t1 where c1 >= 1;
3 row deleted.
dbmMetaManager(DEMO)> select * from t1;
-------------------------------------------------------------------------------
0 row selected
```

노트

Delete로 삭제할 레코드는 잠금 (lock) 되어 다른 세션의 접근을 차단한다.
다른 세션에서 select 할 경우 delete 이전의 커밋된 데이터를 조회한다.
(단, DBM_MVCC_ENABLE = FALSE 로 설정된 경우에는 delete가 완료될 때까지 대기한다.)

select 및 select for update

기능

사용자가 지정한 테이블에서 한 건 이상의 데이터를 조회한다.

구문

```
<select> ::= SELECT target_list FROM table_name
             [ WHERE cond_expression ]
             [ FOR UPDATE ]
             ;
```

- target_list ::= *
  | column_name [, ...]
- table_name: 데이터를 조회할 대상 테이블 이다.
- cond_expression: 조회할 조건절 이다.
- FOR UPDATE: Select 할 때 lock을 걸어 변경되는 것을 방지해야 할 경우에 사용한다.

사용 예

```
dbmMetaManager(DEMO)> select * from user_data;
--------------------------------------------------------------------------------
EMPNO   : 1
EMPNAME : alice
DEPTNO  : 100
BIRTH   : 2025/07/30 08:08:15.484030
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select * from user_data for update;
--------------------------------------------------------------------------------
EMPNO   : 1
EMPNAME : alice
DEPTNO  : 100
BIRTH   : 2025/07/30 08:08:15.484030
--------------------------------------------------------------------------------
1 row selected
```

enqueue

기능

사용자가 지정한 테이블에 한 건의 데이터를 enqueue 한다.

구문

```
<enqueue> ::= ENQUEUE INTO table_name
              [ target_list ]
              VALUES
              ( value_expression [, ...] )
              ;
```

- target_list ::= (column_name [, ...])
  - 사용 가능한 column은 priority, msg_size, message 이다.
- table_name: 데이터를 삽입할 대상 테이블이다.
- value_expression: 삽입할 value 이다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
dbmMetaManager(DEMO)> create queue que1 size 100;
success
dbmMetaManager(DEMO)> enqueue into que1 (priority, msg_size, message) values (90, 10, 'msg
1234567');
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select * from que1;
--------------------------------------------------------------------------------
PRIORITY : 90
ID       : 2
MSG_SIZE : 10
IN_TIME  : 2025/07/04 16:11:57.357598
MESSAGE  : msg1234567
--------------------------------------------------------------------------------
1 row selected
```

> 노트
> msg_size 와 message는 필수 항목이다.

insert 구문을 사용하여 강제로 수행할 경우 다음과 같이 오류가 발생한다.

```
dbmMetaManager(DEMO)> insert into que1 (priority, msg_size, message) values (90, 10, 'msg
1234567');
Command] <insert into que1 (priority, msg_size, message) values (90, 10, 'msg1234567')>
ERR-22073] a operation can not be executed on target-table (check table type or mode)
```

> 노트
> 내부적으로 관리하는 message의 ID는 enqueue 시점에 채번되며 커밋된 데이터 내에서는 Message ID순서로 dequeue된다.

dequeue

기능

사용자가 지정한 테이블에서 한 건의 데이터를 dequeue 한다.

구문

```
<dequeue> ::= DEQUEUE FROM table_name
              [ WHERE cond_expression ]
              [ TIMEOUT seconds ]
              ;
```

- table_name: 대상 테이블이다.
- cond_expression: Dequeue 할 조건절이다.
- seconds: 데이터가 없는 경우 기다리는 시간 (단위: 초)
  - 0: 데이터가 없는 경우 즉시 에러를 반환한다.
  - 음수: 데이터가 없는 경우 무한 대기한다.
  - 양수: 지정한 timeout 값만큼 대기한다.

사용 예

```
************************************************************
* Copyright © 2010 SUNJESOFT Inc. All rights reserved.
************************************************************
ddbmMetaManager(DEMO)> select * from que1;
------------------------------------------------------------------------------
PRIORITY : 0
ID       : 10
MSG_SIZE : 10
IN_TIME  : 2025/07/11 18:13:19.466761
MESSAGE  : msg1234
------------------------------------------------------------------------------
PRIORITY : 0
ID       : 11
MSG_SIZE : 10
IN_TIME  : 2025/07/11 18:13:19.466762
MESSAGE  : msg5678
------------------------------------------------------------------------------
2 row selected
dbmMetaManager(DEMO)> dequeue from que1 where ID = 11;
PRIORITY             : 0
```

```
ID                  : 11
MSG_SIZE            : 10
IN_TIME             : 2025/07/11 18:13:19.466762
MESSAGE             : msg5678
2 row selected
```

노트

- Priority가 낮은 값의 레코드부터 dequeue 된다.
- Priority가 같을 경우, commit 된 데이터 내의 Message ID 순서대로 dequeue 된다.
- Dequeue로 한 건을 가져온 이후 commit 하지 않는 세션이 존재하더라도 다른 세션은 이를 기다리지 않고 다음 데이터를 dequeue 한다.

set

기능

Store 타입 테이블에서 key에 대한 value를 저장하거나 갱신한다.
* 사용자가 지정한 테이블에 특정 key가 존재하지 않을 경우, value가 삽입된다.
* 사용자가 지정한 테이블에 특정 key가 존재하는 경우, value가 갱신된다.

구문

```
<set> ::= SET key value AT table_name
          [ nx ]
          ;
```

- key: Value 값을 저장할 key 이다.
- value: 해당 key에 저장할 값이다.
- nx: 중복을 허용하지 않는다.
- table_name: 데이터를 삽입하거나 갱신할 대상 Store 타입 테이블이다.

사용 예

다음은 Store 테이블에 저장하는 예이다.

```
dbmMetaManager(DEMO)> create store st1 key 32 value 100;
success
dbmMetaManager(DEMO)> set 'k1' 'v1' at st1;
```

```
success
dbmMetaManager(DEMO)> select * from st1;
--------------------------------------------------------------------------------
ST_KEY   : k1
ST_VALUE : v1
--------------------------------------------------------------------------------
1 row selected
```

> 노트
> 서로 다른 세션에서 같인 Key를 Set할 경우에는 Insert와 동일하게 선행 트랜잭션이 완료될 때까지 후행 트랜잭션은 대기한다. 동일 세션에서 같은 Key값으로 삽입을 수행하면 변경이 아닌 "duplicated" 오류가 발생한다.

중복을 허용하지 않는 "NX" 옵션을 사용하면 다음과 같이 중복된 Key가 존재할 때 에러로 반환된다.

```
dbmMetaManager(DEMO)> set 'k1' 'v99' at st1 nx;
Command] <set 'k1' 'v99' at st1 nx>
ERR-22055] key value duplicated (IDX_ST1)
```

다음은 기존 key에 대한 value를 갱신하는 예이다.

```
dbmMetaManager(DEMO)> set 'k1' 'v99' at st1;
success
dbmMetaManager(DEMO)> select * from st1;
--------------------------------------------------------------------------------
ST_KEY   : k1
ST_VALUE : v99
--------------------------------------------------------------------------------
1 row selected
```

get

기능

Store 타입 테이블에서 특정 key에 해당하는 값을 조회한다.

구문

```
<get> ::= GET key AT table_name
          ;
```

- key: value 값을 저장할 key 이다.
- table_name: 데이터를 조회할 대상 테이블이다.

사용 예

```
dbmMetaManager(DEMO)> get 'k1' at st1;
--------------------------------------------------------------------------------
ST_KEY   : k1
ST_VALUE : v1
--------------------------------------------------------------------------------
1 row selected
```

> 노트
> Select 문을 이용한 Store 테이블 조회도 가능하다.

Data Control Language (DCL)

DCL은 사용자가 수행한 각 트랜잭션을 commit/ rollback 하는 구문이다.

commit

기능

사용자가 발생시킨 트랜잭션을 영구적으로 반영한다.

구문

```
<commit> ::= COMMIT
             ;
```

사용 예

```
dbmMetaManager(DEMO)> commit;
success
```

rollback

기능

사용자가 발생시킨 트랜잭션을 이전 상태로 복원한다.

구문

```
<rollback> ::= ROLLBACK
               ;
```

사용 예

```
dbmMetaManager(DEMO)> rollback;
success
```

Built-in Function

- 사용상 편의를 위해 다음과 같은 built-in function을 제공한다.
- 반환되는 문자열의 크기는 최대 32Kbytes 이다.
- GOLDILOCKS LITE에서 제공하는 모든 숫자형 함수들은 overflow/ underflow와 관련된 에러를 체크하지 않는다.

Category	Function name	Return type	Desc
날짜/ 시간	sysdate	long (8 bytes)	내부 저장 용도의 8 byte long long 형태의 값으로 저장하고 출력한다.
	extract	int (4 bytes)	날짜형식의 값에서 지정된 항목의 값을 숫자형으로 반환한다.
	datetime_str	char (64 bytes 이내)	Date column을 문자열로 출력한다.
	to_date	date (8 bytes)	사용자 입력 문자열을 date type 값으로 변환한다.
			입력된 날짜 타입의 두 개인자

Category	Function name	Return type	Desc
	datediff	long (8 bytes)	사이의 간격을 초단위로 출력한다.
Dump	dump	char (최대 1Mbyte)	ascii를 출력한다. byte당 2~4 byte로 출력된다.
	hex	char (최대 1Mbyte)	hex를 출력한다. byte당 2 byte로 출력된다.
문자형	concat	char (최대 1Mbyte)	두 번째 인자의 문자열을 첫번째 인자에 붙여서 출력한다.
	instr	int (4 bytes)	소스문자열 내에 지정된 문자열이 존재할 경우 시작 위치를 1로 하여 offset을 출력한다.
	replace	char (최대 1Mbyte)	검색어를 찾아 지정된 문자열로 치환한다.
	substr	char (최대 1Mbyte)	문자열에서 지정된 위치부터 입력된 크기만큼 잘라낸다.
	length	int (4 bytes)	NULL 지점까지의 길이를 반환하며 NULL이 없을 경우 오류가 발생할 수 있다.
	ltrim	char (최대 1 Mbyte)	문자열의 왼쪽 공백 또는, 지정된 문자를 제거한다.
	rtrim	char (최대 1 Mbyte)	문자열의 오른쪽 공백, 또는 지정된 문자를 제거한다.
	lpad	char (최대 1 Mbyte)	문자열의 왼쪽에 지정한 문자열을 추가한다.
	rpad	char (최대 1 Mbyte)	문자열의 오른쪽에 지정한 문자열을 추가한다.
	upper	char (최대 1 Mbyte)	값을 대문자로 변환한다.
	lower	char (최대 1 Mbyte)	값을 소문자로 변환한다.
숫자형	abs	double (8 bytes)	절대값으로 변환한다.
	power	double (8 bytes)	입력된 수의 제곱을 출력한다.
	sqrt	double (8 bytes)	입력된 수의 제곱근을 출력한다.
	log	double (8 bytes)	base를 기준으로 하는 자연로그 결과를 출력한다.
	exp	double (8 bytes)	e의 제곱을 출력한다.
	mod	double (8 bytes)	나머지 연산을 한다.
	ceil	double (8 bytes)	소수점을 올림한다.
	floor	double (8 bytes)	소수점을 내림한다.
	round	double (8 bytes)	반올림 한다.
	trunc	double (8 bytes)	절사 연산을 한다.
	random	Int (4 bytes)	주어진 입력값 사이의 rando

Category	Function name	Return type	Desc
			m 정수를 출력한다.
Sequence	currval	long long (8 bytes)	-
	nextval	long long (8 bytes)	-
단방향 hash 암호화	digest	char (64 bytes)	알고리즘에 따라 반환되는 길이가 다르다.
Aggregation	min	double (8 bytes)	group by를 지원하지 않는다.
	max	double (8 bytes)	
	avg	double (8 bytes)	
	sum	double (8 bytes)	
JSON OBJECT	JSON_STRING	char	일반 테이블 결과를 JSON 형태로 반환한다.
기타	decode	char (최대 1 Mbyte)	결과와 일치하는 경우 사용자가 지정한 값을 반환한다.
	nvl	char (최대 1 Mbyte)	지정된 값이 NULL일 경우 사용자가 지정한 값을 반환한다.
	user_type	char (최대 1 Mbyte)	Column을 user_type으로 캐스팅하여 출력한다.

sysdate

현재 시스템 시간을 구하여 date type으로 반환한다.
(단, dbmMetaManager에서 조회할 때는 long long type이 아닌 string 형태로 반환한다.)

사용 예

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 date);
success
dbmMetaManager(DEMO)> insert into t1 values (1, sysdate);
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select * from t1;
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 2019/04/04 13:08:47.416266
-------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select sysdate from dual;
-------------------------------------------------------------------------------
SYSDATE             : 2021/02/01 15:32:50.110855
```

```
--------------------------------------------------------------------------------
1 row selected
```

노트
Sysdate은 unix time을 반환한다.

extract

주어진 날짜시간 타입의 데이터에서 입력된 항목의 필드를 숫자형으로 추출한다.
추출 가능한 항목은 다음과 같다.

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

사용 예

```
dbmMetaManager(DEMO)> select extract( 'year' from to_date('20211231115859', 'yyyymmddhhmiss')
) from dual;
--------------------------------------------------------------------------------
EXTRACT              : 2021
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select extract( 'month' from to_date('20211231115859', 'yyyymmddhhmiss')
) from dual;
--------------------------------------------------------------------------------
EXTRACT              : 12
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select extract( 'day' from to_date('20211231115859', 'yyyymmddhhmiss') )
from dual;
--------------------------------------------------------------------------------
EXTRACT              : 31
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select extract( 'hour' from to_date('20211231115859', 'yyyymmddhhmiss')
```

```
) from dual;
--------------------------------------------------------------------------------
EXTRACT              : 11
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select extract( 'minute' from to_date('20211231115859', 'yyyymmddhhmiss
') ) from dual;
--------------------------------------------------------------------------------
EXTRACT              : 58
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select extract( 'second' from to_date('20211231115859', 'yyyymmddhhmiss
') ) from dual;
--------------------------------------------------------------------------------
EXTRACT              : 59
--------------------------------------------------------------------------------
1 row selected
```

datetime_str

Date type의 column을 string으로 출력한다. 포맷은 YYYY/MM/DD H24:MI:SS.SSSSSS로 고정되어 있다.
인자로는 date type을 사용할 수 있다.

사용 예

```
dbmMetaManager(DEMO)> select datetime_str(sysdate) from dual;
--------------------------------------------------------------------------------
DATETIME_STR         : 2025/07/04 17:34:32.436717
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)>
```

to_date('문자열', 'Format')

사용자 입력 문자열을 일정한 형식의 date 타입 값으로 변환한다. Default 형식은 "YYYY/MM/DD H24:MI:SS.S6"로 고정되어 있다.

Format 지정은 다음 표를 참조한다.

Format	설명
YYYY	4 자리 연도이다.

Format	설명
MM	월의 단위로서 (01~12) 범위이다.
DD	일의 단위로서 (01~31) 범위이다.
HH	시간의 단위로서 (00~23) 범위이다.
MI	분의 단위로서 (00~59) 범위이다.
SS	초의 단위로서 (00~59) 범위이다.
S3	Millisecond까지 사용하는 경우로서 3 자리이다.
S6	Microsecond까지 사용하는 경우로서 6 자리이다.

사용 예

```
dbmMetaManager(DEMO)> select to_date( '2020/12/31 15:38:41.123456', 'yyyy/mm/dd hh:mi:ss.s6')
from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 15:38:41.123456
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12/31 15:38:41.123', 'yyyy/mm/dd hh:mi:ss.s3')
from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 15:38:41.123000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12/31 15:38:41', 'yyyy/mm/dd hh:mi:ss') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 15:38:41.000000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12/31 15:38', 'yyyy/mm/dd hh:mi') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 15:38:00.000000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12/31 15', 'yyyy/mm/dd hh') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 15:00:00.000000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12/31', 'yyyy/mm/dd') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/31 00:00:00.000000
```

```
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020/12', 'yyyy/mm') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/01 00:00:00.000000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '2020', 'yyyy') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/01 00:00:00.000000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select to_date( '11:31', 'hh:mi') from dual;
--------------------------------------------------------------------------------
TO_DATE              : 2020/12/01 11:31:00.000000
--------------------------------------------------------------------------------
1 row selected
```

Format이 지정되지 않는 경우에는 사용자의 문자열을 기본 포맷 형태로 일치시켜야 하는데 이 때 연/월/일을 포함해야 하며 그렇지 않은 경우에는 오류가 발생한다.

```
dbmMetaManager(DEMO)> select to_date('12:40') from dual;
Command] <select to_date('12:40') from dual>
ERR-22047] invalid expression type
ERR-22001] invalid parameters or usage at internal processing
```

다음과 같이 DML에 함수를 포함시켜서 사용할 수 있다.

```
dbmMetaManager(DEMO)> insert into t1 values (1, sysdate);
success
dbmMetaManager(DEMO)> insert into t1 values (1, to_date('2002/05/26 11:31:45.123456') );
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select * from t1;
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 2020/12/28 16:26:18.548964
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 2002/05/26 11:31:45.123456
--------------------------------------------------------------------------------
```

datediff(From, to)

입력된 날짜 타입의 인자 두 개 (from ~ to) 사이의 간격을 초단위로 환산하여 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select datediff( to_date('20201231', 'yyyymmdd'), to_date('20200101', '
yyyymmdd') ) from dual;
--------------------------------------------------------------------------------
DATEDIFF            : 31536000
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select datediff( to_date('20201231', 'yyyymmdd'), to_date('20200101', '
yyyymmdd') ) / (60 * 60 * 24) from dual;
--------------------------------------------------------------------------------
DIVIDE              : 365
--------------------------------------------------------------------------------
1 row selected
```

dump(value, [base])

지정된 문자열 value에 대해 byte 단위로 ascii 값으로 변환한 text를 반환한다.
Base를 별도로 지정하지 않을 경우 default로 10 진수 ascii로 동작하며 16 진수를 지원한다.

사용 예

```
dbmMetaManager(DEMO)> select c1 from t1;
--------------------------------------------------------------------------------
C1                  : a
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select dump(c1, 16) from t1;
--------------------------------------------------------------------------------
DUMP                : len=1: 61
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select dump(c1, 10) from t1;
--------------------------------------------------------------------------------
DUMP                : len=1: 97
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select dump(c1) from t1;
```

```
--------------------------------------------------------------------------------
DUMP                : len=1: 97
--------------------------------------------------------------------------------
1 row selected
```

Hex (Value)

인자로 주어진 문자열 value를 hex code로 출력한다.

사용 예

```
dbmMetaManager(DEMO)> select hex( 'abc' ) from dual;
--------------------------------------------------------------------------------
HEX                 : 616263
--------------------------------------------------------------------------------
1 row selected
```

concat(target, append)

target 문자열 뒤에 append 문자열을 추가하는 연산을 수행한다.

사용 예

```
dbmMetaManager(DEMO)> select concat( 'abc', 'def') from dual;
--------------------------------------------------------------------------------
CONCAT              : abcdef
--------------------------------------------------------------------------------
1 row selected
```

instr(source, keyword, [start, #appearance])

Source 문자열 내에서 keyword를 찾아 위치를 반환한다.
start가 생략되면 처음부터 탐색하고 음수인 경우 역순으로 검색한다. 만일 입력값이 0이면 출력값은 keyword에 상관없이 0으로 반환된다.
#appearance는 keyword가 source 내에 출현한 횟수가 일치하는 지점을 탐색한다.

사용 예

```
dbmMetaManager(DEMO)> select instr( 'abcdef abcdef', 'def' ) from dual;
--------------------------------------------------------------------------------
INSTR               : 4
```

```
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select instr( 'abcdef abcdef', 'def', 1, 2 ) from dual;
--------------------------------------------------------------------------------
INSTR                : 11
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select instr('abcd abcd abcd', 'bc', -5, 2) from dual;
--------------------------------------------------------------------------------
INSTR                : 2
--------------------------------------------------------------------------------
1 row selected
```

replace(source, keyword, replace)

Source 문자열 내에 keyword를 찾아 replace 문자열을 변경하는 연산을 수행한다.

사용 예

```
dbmMetaManager(DEMO)> select replace( 'abc, abc ing', 'abc', 'xxxxx' ) from dual;
--------------------------------------------------------------------------------
REPLACE              : xxxxx, xxxxx ing
--------------------------------------------------------------------------------
1 row selected
```

substr(source, start_position, count)

Source 문자열의 start_position부터 count 만큼 문자열을 잘라 반환한다.

- Count가 생략될 경우 start_position부터 남은 문자열을 모두 반환한다.
- Start_position이 source 문자열 길이를 초과할 경우, NULL을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select substr( 'abcdefghi', 4) from t1;
--------------------------------------------------------------------------------
SUBSTR               : defghi
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select substr( 'abcdefghi', 9, 5) from t1;
--------------------------------------------------------------------------------
```

```
SUBSTR              : i
-------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select dump(substr('abc', 5, 2)) from dual;
-------------------------------------------------------------------------------
DUMP : len=0:
-------------------------------------------------------------------------------
1 row selected
```

length(source)

Source 문자열 길이를 반환한다. 중간에 NULL이 있다면 그 위치까지의 길이만 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select length( 'abcdefg' ) from dual;
-------------------------------------------------------------------------------
LENGTH : 7
-------------------------------------------------------------------------------
1 row selected
```

ltrim / rtrim(source)

Source 문자열의 왼쪽 또는 오른쪽에 존재하는 공백 문자를 제거하여 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select ltrim( '   xyz' ) from dual;
-------------------------------------------------------------------------------
LTRIM               : xyz
-------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select rtrim( 'xyz   ' ) from dual;
-------------------------------------------------------------------------------
RTRIM               : xyz
-------------------------------------------------------------------------------
1 row selected
```

lpad / rpad(source, size, padding_string)

Source 문자열의 왼쪽 또는 오른쪽에 사용자가 입력한 padding_string을 size 이내로 추가하여 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select lpad('aa', 10, 'x') from dual;
--------------------------------------------------------------------------------
LPAD                : xxxxxxxxaa
--------------------------------------------------------------------------------
dbmMetaManager(DEMO)> select rpad('aa', 10, 'x') from dual;
--------------------------------------------------------------------------------
RPAD                : aaxxxxxxxx
--------------------------------------------------------------------------------
1 row selected
```

abs(value)

입력 항목의 절대값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select abs(-123) from dual;
--------------------------------------------------------------------------------
ABS                 : 123
--------------------------------------------------------------------------------
1 row selected
```

mod(value1, value2)

(value1 / value2) 연산으로 발생한 나머지 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select mod(14, 3) from dual;
--------------------------------------------------------------------------------
MOD                 : 2
--------------------------------------------------------------------------------
1 row selected
```

ceil(value)

Value 보다 큰 가장 가까운 정수를 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select ceil( 12.345) from dual;
--------------------------------------------------------------------------------
CEIL                : 13
--------------------------------------------------------------------------------
1 row selected
```

floor(value)

Value 보다 작은 가장 가까운 정수를 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select floor(13.678) from dual;
--------------------------------------------------------------------------------
FLOOR               : 13
--------------------------------------------------------------------------------
1 row selected
```

round(value)

Value를 반올림하여 정수를 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select round(12.345) from dual;
--------------------------------------------------------------------------------
ROUND               : 12
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select round(12.678) from dual;
--------------------------------------------------------------------------------
ROUND               : 13
--------------------------------------------------------------------------------
```

trunc(value, [pos])

Value가 소수점일 경우 지정된 pos 위치에서 소수점 이하의 수를 제거한 후에 반환한다.
지정되지 않은 경우 소수점 이하를 모두 버린 후에 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select trunc(13.34567) from dual;
--------------------------------------------------------------------------------
TRUNC                : 13
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select trunc(13.34567, 2) from dual;
--------------------------------------------------------------------------------
TRUNC : 13.34
--------------------------------------------------------------------------------
```

random(from, to)

From, To로 지정된 범위 내의 random 정수를 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select random(1, 10) from dual;
--------------------------------------------------------------------------------
RANDOM               : 2
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select random(100, 200) from dual;
--------------------------------------------------------------------------------
RANDOM               : 144
--------------------------------------------------------------------------------
1 row selected
```

nextval

지정된 sequence의 다음 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select seq1.nextval from dual;
--------------------------------------------------------------------------------
NEXTVAL              : 2
--------------------------------------------------------------------------------
1 row selected
```

currval

지정된 sequence의 현재 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select seq1.currval from dual;
--------------------------------------------------------------------------------
CURRVAL              : 2
--------------------------------------------------------------------------------
1 row selected
```

> 노트
> Sequence 객체에 대해 nextval이 호출되지 않았을 때 currval을 호출하면 오류를 반환한다.

Digest (Value, SHA-type)

입력된 문자열 value와 SHA-type로 암호화 된 결과를 출력한다. Digest 처리된 결과는 binary 형태로써 화면에 정상적으로 출력되지 않을 수도 있다. 이 경우, 다음과 같이 hex 함수 등을 통해 확인할 수 있다.

사용 예

```
dbmMetaManager(DEMO)> select hex( digest( 'my password', 'SHA256' ) ) from dual;
--------------------------------------------------------------------------------
HEX                  : BB14292D91C6D0920A5536BB41F3A50F66351B7B9D94C804DFCE8A96CA1051F2
--------------------------------------------------------------------------------
1 row selected
```

Min(Column)

Select 된 결과 집합에서 min 함수 내에 정의된 column 값 중 가장 작은 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 = 1;
--------------------------------------------------------------------------------
C1                    : 1
C2                    : -1
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 1
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 2
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 3
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 4
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 5
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 6
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 7
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 8
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 9
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 10
--------------------------------------------------------------------------------
11 row selected
```

```
dbmMetaManager(DEMO)> select min(c2) from t1 where c1 = 1;
--------------------------------------------------------------------------------
MIN_VALUE            : -1.000000000
--------------------------------------------------------------------------------
1 row selected
```

Max(Column)

Select 된 결과 집합에서 max 함수 내에 정의된 column 값 중 가장 큰 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 = 1;
--------------------------------------------------------------------------------
C1                   : 1
C2                   : -1
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 1
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 2
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 3
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 4
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 5
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 6
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 7
--------------------------------------------------------------------------------
C1                   : 1
C2                   : 8
--------------------------------------------------------------------------------
```

```
C1                  : 1
C2                  : 9
------------------------------------------------------------------------------
C1                  : 1
C2                  : 10
------------------------------------------------------------------------------
11 row selected
dbmMetaManager(DEMO)> select max(c2) from t1 where c1 = 1;
------------------------------------------------------------------------------
MAX_VALUE           : 10.000000000
------------------------------------------------------------------------------
1 row selected
```

Avg(Column)

Avg 함수는 column 값의 평균값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 = 1;
------------------------------------------------------------------------------
C1                  : 1
C2                  : -1
------------------------------------------------------------------------------
C1                  : 1
C2                  : 1
------------------------------------------------------------------------------
C1                  : 1
C2                  : 2
------------------------------------------------------------------------------
C1                  : 1
C2                  : 3
------------------------------------------------------------------------------
C1                  : 1
C2                  : 4
------------------------------------------------------------------------------
C1                  : 1
C2                  : 5
------------------------------------------------------------------------------
C1                  : 1
C2                  : 6
```

```
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 7
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 8
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 9
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 10
--------------------------------------------------------------------------------
11 row selected
dbmMetaManager(DEMO)> select avg(c2) from t1 where c1 = 1;
--------------------------------------------------------------------------------
AVG                 : 4.909090909
--------------------------------------------------------------------------------
1 row selected
```

Sum(Column)

Sum 함수는 column 값의 합계를 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select * from t1 where c1 = 1;
--------------------------------------------------------------------------------
C1                  : 1
C2                  : -1
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 1
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 2
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 3
--------------------------------------------------------------------------------
C1                  : 1
```

```
C2                  : 4
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 5
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 6
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 7
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 8
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 9
-------------------------------------------------------------------------------
C1                  : 1
C2                  : 10
-------------------------------------------------------------------------------
11 row selected
dbmMetaManager(DEMO)> select sum(c2) from t1 where c1 = 1;
-------------------------------------------------------------------------------
SUM                 : 54.000000000
-------------------------------------------------------------------------------
1 row selected
```

Decode(cond_expr, case_cond, value_expr, ..., [else_expr])

cond_expr이 가진 값과 일치하는 case_cond 다음에 기술된 value를 반환한다. 만일 일치하는 경우가 없을 경우, else_expr이 존재하면 해당 값을 반환하며 그렇지 않으면 길이가 0인 값을 반환한다.
cond_expr과 case_cond의 데이터 타입은 동일한데 double 또는 문자열을 사용할 수 있으며 반환되는 value_expr, else_expr은 문자열 데이터 타입이다.

사용 예

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 int);
success
dbmMetaManager(DEMO)> insert into t1 values (1, 10);
success
dbmMetaManager(DEMO)> insert into t1 values (2, 20);
```

```
success
dbmMetaManager(DEMO)> insert into t1 values (3, 30);
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select c1, c2, decode( c1, 1, 'a', 2, 'b', 3, 'c', 'k' ) from t1;
--------------------------------------------------------------------------------
C1                  : 1
C2                  : 10
DECODE              : a
--------------------------------------------------------------------------------
C1                  : 2
C2                  : 20
DECODE              : b
--------------------------------------------------------------------------------
C1                  : 3
C2                  : 30
DECODE              : c
--------------------------------------------------------------------------------
3 row selected
```

Upper(expr)

주어진 문자열 expr이 text일 경우, 이를 대문자로 변환하여 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select upper( 'aaaaabzc' ) from dual;
--------------------------------------------------------------------------------
UPPER               : AAAAABZC
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select upper( 123 ) from dual;
--------------------------------------------------------------------------------
UPPER               : 123
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select upper( 'a1b1C34' ) from dual;
--------------------------------------------------------------------------------
UPPER               : A1B1C34
--------------------------------------------------------------------------------
```

```
1 row selected
```

Lower(expr)

주어진 문자열 expr이 text일 경우, 이를 소문자로 변환하여 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select lower( 'AAAAABZC' ) from dual;
--------------------------------------------------------------------------------
LOWER                : aaaaabzc
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select lower( 123 ) from dual;
--------------------------------------------------------------------------------
LOWER                : 123
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select lower( 'A1B1c34' ) from dual;
--------------------------------------------------------------------------------
LOWER                : a1b1c34
--------------------------------------------------------------------------------
1 row selected
```

NVL(orgnExpr, valueExpr)

주어진 문자열 orgnExpr의 첫 번째 byte가 NULL인 경우 ValueExpr로 변환된 값을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> select nvl( 'x', 'not_null') from dual;
--------------------------------------------------------------------------------
NVL                  : x
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> select nvl( '', 'isnull') from dual;
--------------------------------------------------------------------------------
NVL                  : isnull
--------------------------------------------------------------------------------
1 row selected
```

JSON_STRING(Column_Name_List)

일반 테이블에 한해 전체 또는 주어진 column 목록에 해당하는 데이터들만 JSON 형태로 변환된 string을 반환한다.

사용 예

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 int, c3 char(20) );
success
dbmMetaManager(DEMO)> insert into t1 values (1, 2, 'xyz1' );
success
dbmMetaManager(DEMO)> insert into t1 values (100, 200, 'zyx2' );
success
dbmMetaManager(DEMO)> insert into t1 values (10, 20, 'ppppppppp1' );
success
dbmMetaManager(DEMO)> commit;
success
dbmMetaManager(DEMO)> select json_string(*) from t1;
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "1", "C2": "2", "C3": "xyz1" }
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "100", "C2": "200", "C3": "zyx2" }
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "10", "C2": "20", "C3": "ppppppppp1" }
--------------------------------------------------------------------------------
3 row selected
dbmMetaManager(DEMO)> select json_string(c1, c2, c3) from t1;
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "1", "C2": "2", "C3": "xyz1" }
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "100", "C2": "200", "C3": "zyx2" }
--------------------------------------------------------------------------------
JSON_STRING          : { "C1": "10", "C2": "20", "C3": "ppppppppp1" }
--------------------------------------------------------------------------------
3 row selected
```

노트

json_string에는 lite가 지원하는 타입만 제공되며 json_string과 같은 타입은 추가할 수 없다.

USER_TYPE(Column_Name, Type_Name)

User type으로 지정된 column을 캐스팅하여 값을 출력한다.

사용 예

```
dbmMetaManager(DEMO)> select * from t2;
--------------------------------------------------------------------------------
C1 : 100
C2 : 200
C3 :
--------------------------------------------------------------------------------
C1 : 200
C2 : 300
C3 :
--------------------------------------------------------------------------------
2 row selected
dbmMetaManager(DEMO)> select c1, c2, user_type(c3, u2) from t2;
--------------------------------------------------------------------------------
C1        : 100
C2        : 200
USER_TYPE : C1=-100 C2=-200
--------------------------------------------------------------------------------
C1        : 200
C2        : 300
USER_TYPE : C1=-200 C2=-300
--------------------------------------------------------------------------------
dbmMetaManager(DEMO)> select user_type(c3, u2) from t2 where user_type(c3, u2.c1 ) = -100;
--------------------------------------------------------------------------------
USER_TYPE : C1=-100 C2=-200
--------------------------------------------------------------------------------
1 row selected
```

1.4 DICTIONARY

Instance가 생성될 때 내부 meta 정보 관리를 위한 자동 생성되는 Built-in Table을 Dictionary로 정의한다.

- DICT (initdb 수행 시 생성)

    - DICT_INST : Instance meta 정보 관리
- 사용자 instance (create instance 수행 시 생성)
    - 사용자 Table등의 Meta 정보 관리
        - DIC_TABLE
        - DIC_INDEX
        - DIC_COLUMN
        - DIC_INDEX_COLUMN
        - DIC_SEQUENCE
        - DIC_REPL_INST
        - DIC_REPL_TABLE
    - Information View
        - V$INSTANCE
        - V$SESSION
        - V$TRANSACTION
        - V$SYS_STAT
        - V$SESS_STAT
        - V$TABLE_USAGE
        - V$REPL_STAT
        - V$LOG_STAT

노트

- dictionary table은 DDL/DML을 허용하지 않는다.
- 테이블 목록에 보이지만 설명이 없는 dictionary table은 향후 deprecated 예정이다.

DICTIONARY TABLES

DIC_INST

DICT instance 내에 생성되며 사용자 instance 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
INIT_SIZE	생성 시 초기 크기 (단위: undo page의 개수)
EXTEND_SIZE	공간 확장 시 크기
MAX_SIZE	최대 확장 가능 크기

DIC_TABLE

table 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
TABLE_NAME	Table name
TABLE_TYPE	TABLE 유형 • 1: TABLE • 2: QUEUE • 3: STORE • 4: SEQUENCE • 5: DIRECT • 7: SPLAY • 10: USER_TYPE
COLUMN_COUNT	TABLE을 구성하는 column의 개수
ROW_SIZE	TABLE에 저장되는 레코드 크기
LOCK_MODE	1: 동시성 제어모드 (0: deprecate 예정)
MSG_SIZE	QUEUE TABLE인 경우 message의 최대 크기
INDEX_COUNT	현재 테이블에 생성된 INDEX 개수
INIT_SIZE	생성 시 초기 segment의 크기 (단위: 레코드 개수)
EXTEND_SIZE	공간 확장 시 segment의 크기 (단위: 레코드 개수)
MAX_SIZE	최대 확장 가능한 segment의 크기 (단위: 레코드 개수)
INDEX_ID	Index 생성 시점마다 부여되는 index 고유 번호 채번 용도
ONLY_UPDATE_SELECT_MODE	테이블을 생성할 때 단일 프로세스만 접근할 수 있게 설정한 경우 (deprecate 예정)
CREATE_SCN	테이블 생성 시점의 SCN (system commit number)

DIC_COLUMN

Table을 구성하는 column 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
TABLE_NAME	Table name
COLUMN_NAME	Column name
USER_TYPE_NAME	User type으로 지정된 경우 해당 type name
DATA_TYPE	Column의 데이터 타입 • 1: short • 2: int • 3: double • 4: float • 5: long

Column name	설명
	• 6: char • 7: date • 15: USER_TYPE
COLUMN_OFFSET	레코드 전체 영역 중에 column이 저장되는 위치
COLUMN_SIZE	Column의 데이터 크기
COLUMN_ORDER	Column 순서

DIC_INDEX

Table에 생성한 index의 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
TABLE_NAME	Table name
INDEX_NAME	Index name
IS_UNIQUE	Unique 여부 • 1: unique • 0: non-unique
KEY_SIZE	Key column 크기의 합
KEY_COLUMN_COUNT	Key column의 총 개수
INDEX_ORDER	Index가 생성된 순서

DIC_INDEX_COLUMN

Index를 구성하는 key column의 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
TABLE_NAME	Table name
INDEX_NAME	Index name
COLUMN_NAME	Column name
ID	Index key column 의 고유 번호
COLUMN_JSON	json path
JSON_KEY_SIZE	json path 내의 key 크기
KEY_COLUMN_ORDER	Key column의 나열 순서 (Ordering 순서를 의미한다.)
COLUMN_ORDER	테이블 내의 column 순서
IS_ASC	오름차순 여부 (1: ASC, 0: DESC)

DIC_SEQUENCE

Sequence object를 구성하는 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
SEQUENCE_NAME	Sequence name
START_VALUE	초기값
INCREMENT_VALUE	증가값
CURRENT_VALUE	미사용 column
MIN_VALUE	MinValue
MAX_VALUE	MaxValue
IS_CYCLE	• 1: CYCLE • 0: NOCYCLE

DIC_REPL_INST

Create replication 수행 시 instance 이름을 저장한다.

Column name	설명
INST_NAME	Instance name

DIC_REPL_TABLE

Replication 대상 테이블 정보를 저장한다.

Column name	설명
INST_NAME	Instance name
TABLE_NAME	Table name

Information View

GOLDILOCKS LITE의 각종 정보를 table 형태로 제공하는 view이다.

노트

- Information view는 DDL/DML을 허용하지 않는다.

- 일부 Reserved column 항목은 향후 개발 계획으로 현재 출력 값에는 의미가 없다.

V$INSTANCE

현재 instance의 정보를 출력한다.

Column 이름	설명
CURR_SCN	현재 instance의 SCN 값
MIN_SCN	reserved
CURR_MIN_SCN	reserved
CURR_MIN_SCN_TRANS	reserved
ACTIVE_MODE	reserved
DISK_ENABLE	Disk LogFile 설정 여부
LOGFILE_SIZE	Disk LogFile 한 개의 크기
LOGCACHE_MODE	LogCache 설정 정보
LOGCACHE_CHUNK_SIZE	LogCache chunk 한 개의 크기
LOGCACHE_CHUNK_COUNT	LogCache chunk 최대 개수
LOGCACHE_RANGE	LogCache chunk의 사용 범위
CREATE_TIME	Instance를 생성한 시각

```
dbmMetaManager(DEMO)> select * from v$instance;
--------------------------------------------------------------------------------
SCN                 : 11
MIN_SCN             : 11
MIN_SCN_TRANS_ID    : 1
ACTIVE_MODE         : 1
DISK_ENABLE         : 0
LOGFILE_SIZE        : 0
LOGCACHE_MODE       : no cache mode
LOGCACHE_CHUNK_SIZE : 0
LOGCACHE_CHUNK_COUNT : 1
LOGCACHE_RANGE      : 0
CREATE_TIME         : 2020-03-25 12:57:02
--------------------------------------------------------------------------------
```

V$SESSION

현재 instance를 사용 중인 세션의 정보를 출력한다.

Column 이름	설명
ID	Session 고유 ID 이다.
TRANS_ID	Session이 가진 트랜잭션 식별 번호
PID	현재 Trans Id를 점유하고 있는 OS process ID 이다.
TID	현재 Trans Id를 점유하고 있는 OS thread ID 이다.
OLD_TID	비정상 종료 시점에 할당된 OS thread ID 이다.
CURR_UNDO_PAGE	트랜잭션 진행 중 현재 사용되는 undo page의 ID 이다.
FIRST_UNDO_PAGE	트랜잭션 진행 중에 사용된 첫 번째 undo page의 ID 이다.
LAST_UNDO_PAGE	트랜잭션 진행 중에 사용된 마지막 undo page의 ID 이다.
SAVEPOINT_UNDO_PAGE	트랜잭션 진행 중 implicit savepoint가 필요한 경우 해당 위치를 가리킨다.
SAVEPOINT_UNDO_OFFSET	트랜잭션 진행 중 implicit savepoint가 필요한 경우 해당 위치를 가리킨다.
WAIT_TRANS_ID	Lock 대기 중일 경우 LOCK을 점유한 상대 Trans ID 이다.
WAIT_OBJECT	Lock 대기 중일 경우 대상 테이블 이름이다.
WAIT_SLOT_ID	Lock 대기 중일 경우 데이터 공간의 고유 ID 이다.
SESSION_STATUS	트랜잭션의 현재 상태이다. • transaction: Transaction을 시작할 수 있거나 진행 중이다. • commit start: Commit이 시작되었다. • commit completed: Memory 까지 commit이 완료되었다. • rollback completed: Rollback이 완료되었다. • recovery completed: 비정상 종료로 인한 recovery가 완료되었다.
IS_LOGGING	트랜잭션의 메모리 logging mode 이다. (0: No Logging, 1: Logging) No logging으로 설정할 경우 트랜잭션을 복구하지 않는다. (Rollback이 불가능하다.)
LOGFILE_NO	현재 세션이 디스크 병렬 로깅 모드로 동작 중일 때, 세션이 사용 중인 logfile sequence 이다.
CKPT_NO	dbmCkpt에 의해 데이터 파일에 반영된 logfile sequence 이다.
IS_REPL	Reserved
REPL_SEND_SCN	세션이 마지막으로 전송한 이중화 트랜잭션의 SCN 이다.
REPL_RECV_ACK_SCN	세션이 상대편으로부터 반영 완료 통지를 받은 마지막 트랜잭션의 SCN 이다.
AUTOCOMMIT_MODE	AutoCommit Mode (0: Non-AutoCommit, 1:Auto-Commit)
BEGIN_TIME	세션의 접속 시각이다.
PROGRAM	프로그램 이름이다.
REMOTE_PID	원격 접속의 경우 원격 서버의 프로세스 ID 이다.
REMOTE_ADDR	원격 접속의 경우 원격 서버의 주소이다.
REMOTE_PROGRAM	원격 접속의 경우 원격 서버에서 구동 된 프로그램 이름이다.

```
dbmMetaManager(DEMO)> select * from v$session;
--------------------------------------------------------------------------------
TRANS_ID            : 1
PID                 : 35612
TID                 : 35612
OLD_TID             : 35612
CURR_UNDO_PAGE      : 6
```

```
FIRST_UNDO_PAGE      : 6
LAST_UNDO_PAGE       : 11
SAVEPOINT_UNDO_PAGE  : -1
SAVEPOINT_UNDO_OFFSE : -1
WAIT_TRANS_ID        : -1
WAIT_OBJECT          :
WAIT_SLOT_ID         : -1
STATUS               : transaction ready or running
IS_REPL              : 0
BEGIN_TIME           : 2024/10/18 08:46:47
PROGRAM              : dbmListener
REMOTE_PID           : 0001513772
REMOTE_ADDR          : 192.168.0.26
REMOTE_PROGRAM       : dbmMetaManager
--------------------------------------------------------------------------------
```

V$TRANSACTION

세션들의 트랜잭션 정보를 출력한다.

Column 이름	설명
TRANS_ID	transaction의 고유 번호
TRANS_SEQ	트랜잭션 내 발생한 순서
TRANS_TYPE	트랜잭션 유형
OBJECT_NAME	대상 테이블 이름
SLOT_ID	데이터 공간의 고유 ID
EXTRA_KEY	데이터 공간 내부 관리를 위한 고유 ID
COMMIT_FLAG	트랜잭션의 memory commit 여부 (1: Commit)
SKIP_FLAG	트랜잭션이 commit 되지 않도록 설정된 flag (1: Skip)
VALID_FLAG	트랜잭션 로그의 유효성 (1: 정상)

V$LOG_STAT

Disk log에 대한 전반적인 설정 정보를 보여주는 view이다.

Column 이름	설명
DISKLOG_ENABLE	Disk logging 설정 여부
CACHE_MODE	LOG CACHE MODE 설정값 (0: 병렬 로깅, 1: Cache, 2: NVDIMM)
DIRECT_IO_ENABLE	DIRECT IO 활성화 여부
ARCHIVE_ENABLE	Archive 설정 여부
CURR_FILE_NO	Reserved (Log cache 모드가 활성화 된 경우에만 의미가 있다.)

Column 이름	설명
CURR_FILE_OFFSET	Reserved
LAST_CKPT_FILE_NO	CheckPoint가 수행될 logfile의 시작 번호 (Log cache 모드가 활성화 된 경우에만 의미가 있다.)
LAST_ARCHIVE_FILE_NO	Archive가 시작될 대상 logfile 번호 (Log cache 모드가 활성화 된 경우에만 의미가 있다.)
LAST_CAPTURE_FILE_NO	Reserved
LOGCACHE_WRITE_IND	LogCache 공간 중 트랜잭션이 기록할 수 있는 현재 위치
LOGCACHE_READ_IND	LogCache 공간 중 flusher가 읽을 현재 위치
FLUSHER_FILE_NO	Flushser가 마지막으로 기록한 logfile의 번호 (Log cache 모드가 활성화 된 경우에만 의미가 있다.)
FLUSHER_FILE_OFFSET	Flushser가 마지막으로 기록한 logfile의 offset
LOG_DIR	Disk LogFile이 저장되는 경로
DATAFILE_DIR	dataFile이 저장되는 경로
ARCHIVE_DIR	Checkpoint 시점에 archiving 될 logfile이 저장될 경로

```
dbmMetaManager(DEMO)> select * from v$log_stat;
------------------------------------------------------------------------------
DISKLOG_ENABLE       : 0
CACHE_MODE           : 0
DIRECT_IO_ENABLE     : 0
ARCHIVE_ENABLE       : 0
CURR_FILE_NO         : -1
CURR_FILE_OFFSET     : -1
LAST_CKPT_FILE_NO    : -1
LAST_ARCHIVE_FILE_NO : -1
LAST_CAPTURE_FILE_NO : -1
LOGCACHE_WRITE_IND   : -1
LOGCACHE_READ_IND    : -1
FLUSHER_FILE_NO      : -1
FLUSHER_FILE_OFFSET  : -1
LOG_DIR              : /home/majaehwa/work/new_lite/pkg/wal
DATAFILE_DIR         : /home/majaehwa/work/new_lite/pkg/dbf
ARCHIVE_DIR          : /home/majaehwa/work/new_lite/pkg/arch
------------------------------------------------------------------------------
1 row selected
```

V$REPL_STAT

Replication 환경에서 이중화 상태 정보를 보여준다.

Column 이름	설명
TARGET_IP	Slave IP
TARGET_PORT	Slave port
LISTEN_PORT	Slave 측의 dbmReplica listen port number
SEND_SCN	Master 측에서 전송한 SCN (System Commit Number)
RECV_SCN	Master 측에서 전송한 SCN 중에 마지막으로 수신한 Ack SCN
UNSENT_START_FILENO	미전송 로그가 존재하는 경우, 해당 로그가 기록된 첫 번째 파일 번호
UNSENT_END_FILENO	미전송 로그가 존재하는 경우, 해당 로그가 기록된 마지막 파일 번호

```
dbmMetaManager(DEMO)> select * from v$repl_stat;
--------------------------------------------------------------------------------
TARGET_IP           : 127.0.0.1
TARGET_PORT         : 29002
LISTEN_PORT         : 29002
SEND_SCN            : -1
RECV_SCN            : -1
UNSENT_START_FILENO : 0
UNSENT_END_FILENO   : 0
--------------------------------------------------------------------------------
1 row selected
```

V$TABLE_USAGE

Instance 내에 생성된 모든 테이블의 사용량을 보여준다.
* SIZE는 row 개수를 가리킨다.

Column 이름	설명
OBJECT_NAME	테이블 이름
MAX_SIZE	최대 확장 가능한 크기
TOTAL_SIZE	현재까지 확장된 크기
USED_SIZE	확장된 크기 내에서 현재 사용 중인 공간의 크기
FREE_SIZE	최대 크기까지 남아 있는 여유 공간
USED_MEM	내부 헤더를 포함하여 사용량을 byte 단위로 환산한 크기

```
dbmMetaManager(DEMO)> select * from v$table_usage;
--------------------------------------------------------------------------------
OBJECT_NAME : T1
MAX_SIZE    : 4096000
TOTAL_SIZE  : 1024
USED_SIZE   : 0
```

```
FREE_SIZE    : 1024
USED_MEM     : 90488
-------------------------------------------------------------------------------
1 row selected
```

V$SYS_STAT

Instance에서 발생한 각 유형별 수행 횟수에 대한 정보를 제공한다.

Column 이름	설명
NAME	누적된 유형의 이름
ACCUM_COUNT	누적된 수치

```
dbmMetaManager(DEMO)> select * from v$sys_stat
-------------------------------------------------------------------------------
NAME                 : init_handle_op
ACCUM_COUNT          : 2
-------------------------------------------------------------------------------
NAME                 : free_handle_op
ACCUM_COUNT          : 1
-------------------------------------------------------------------------------
NAME                 : prepare_op
ACCUM_COUNT          : 20
-------------------------------------------------------------------------------
NAME                 : execute_op
ACCUM_COUNT          : 20
-------------------------------------------------------------------------------
NAME                 : insert_op
ACCUM_COUNT          : 2
-------------------------------------------------------------------------------
NAME                 : update_op
ACCUM_COUNT          : 1
-------------------------------------------------------------------------------
NAME                 : delete_op
ACCUM_COUNT          : 1
-------------------------------------------------------------------------------
NAME                 : scan_op
ACCUM_COUNT          : 3
-------------------------------------------------------------------------------
NAME                 : enqueue_op
ACCUM_COUNT          : 0
```

```
--------------------------------------------------------------------------------
NAME                 : dequeue_op
ACCUM_COUNT          : 0
--------------------------------------------------------------------------------
NAME                 : aging_op
ACCUM_COUNT          : 0
--------------------------------------------------------------------------------
NAME                 : commit_op
ACCUM_COUNT          : 11
--------------------------------------------------------------------------------
NAME                 : rollback_op
ACCUM_COUNT          : 1
--------------------------------------------------------------------------------
NAME                 : recovery_rollback_op
ACCUM_COUNT          : 0
--------------------------------------------------------------------------------
NAME                 : recovery_commit_op
ACCUM_COUNT          : 0
--------------------------------------------------------------------------------
```

누적 항목은 아래 표와 같으며 v$sess_stat의 항목도 이와 동일하다. 각 항목에 누적된 수치는 성공/실패 여부와 관계없이 내부 처리 과정에 진입한 횟수를 의미한다.

누적 항목	설명
init_handle_op	D/A mode로 instance에 접속한 횟수
free_handle_op	Instance에서 해제된 횟수
prepare_op	prepare statement가 호출된 횟수
execute_op	execute statement가 호출된 횟수
insert_op	insert가 수행된 횟수
update_op	update가 수행된 횟수
scan_op	select를 포함하여 대상을 scan 하는 모든 횟수
delete_op	delete가 수행된 횟수
enqueue_op	enqueue가 수행된 횟수
dequeue_op	dequeue가 수행된 횟수
aging_op	참조되지 않는 공간이 회수된 횟수
commit_op	commit이 호출된 횟수
rollback_op	rollback이 호출된 횟수
recovery_rollback_op	비정상 복구 (rollback과 동일한 과정)를 수행한 횟수
recovery_commit_op	비정상 복구 (기존에 commit 된 row에 대한 lock을 해제하는 과정)를 수행한 횟수

노트

DBM_PERF_ENALBE 속성이 활성화 된 경우에만 정보가 누적된다.

V$SESS_STAT

Instance가 생성된 이후에 발생한 세션 번호별로, 각 유형의 수행 횟수를 누적하여 출력한다.

Column 이름	설명
TRANS_ID	세션의 고유번호
STAT_NAME	누적 유형의 이름
ACCUM_COUNT	누적 수치

```
dbmMetaManager(DEMO)> select * from v$sess_stat
-------------------------------------------------------------------------------
TRANS_ID             : 1
NAME                 : init_handle_op
ACCUM_COUNT          : 2
-------------------------------------------------------------------------------
TRANS_ID             : 1
NAME                 : free_handle_op
ACCUM_COUNT          : 1
-------------------------------------------------------------------------------
TRANS_ID             : 1
NAME                 : prepare_op
ACCUM_COUNT          : 21
-------------------------------------------------------------------------------
TRANS_ID             : 1
NAME                 : execute_op
ACCUM_COUNT          : 21
-------------------------------------------------------------------------------
TRANS_ID             : 1
NAME                 : insert_op
ACCUM_COUNT          : 2
-------------------------------------------------------------------------------
TRANS_ID             : 1633972341
NAME                 : te_op
ACCUM_COUNT          : 0
-------------------------------------------------------------------------------
TRANS_ID             : 1
```

```
NAME                : delete_op
ACCUM_COUNT         : 1
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : scan_op
ACCUM_COUNT         : 3
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : enqueue_op
ACCUM_COUNT         : 0
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : dequeue_op
ACCUM_COUNT         : 0
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : aging_op
ACCUM_COUNT         : 0
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : commit_op
ACCUM_COUNT         : 11
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : rollback_op
ACCUM_COUNT         : 1
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : recovery_rollback_op
ACCUM_COUNT         : 0
------------------------------------------------------------------------------
TRANS_ID            : 1
NAME                : recovery_commit_op
ACCUM_COUNT         : 0
------------------------------------------------------------------------------
```

노트

DBM_PERF_ENALBE 속성이 활성화 된 경우에만 정보가 누적된다.
이 정보는 세션 접속 이후의 데이터가 아니라, 전체 누적된 값을 의미한다. 따라서 접속 이후의 변동량을 분석하려면, before/after 방식으로 스냅샷을 조회하여 비교해야 한다.

1.5 dbmMetaManager

개요

dbmMetaManager는 DDL/ DML/ DCL을 interactive하게 수행할 수 있는 utility이다.
dbmMetaManager를 수행할 때 사용할 수 있는 옵션은 아래 표와 같다.

입력 옵션	설명
-i <Instance name>	특정 instance name을 지정한다.
-f <script file>	특정 script file 내의 SQL을 순차적으로 실행한 후에 종료한다.
-p <process alias name>	dbmMetaManager가 식별 될 수 있도록 별칭을 지정할 수 있다.
-e	instance password를 설정한 경우 접근 시 암호를 입력한다.
-v	제품의 버전 정보를 출력한다.
-s	화면에 제품 사용 권한, 버전정보 등에 대한 출력을 생략한다.

```
[majaehwa@tech9 new_lite]$ dbmMetaManager -h
Usage] dbmMetaManager
-f : input a file-name to execute: -f <file name>
-p : specify a alias-name
-i : specify a instance name to attach: -i <instance name>
-e : specify a password
-v : print version info
-s : to silent option
-h : Display this information
```

Internal commands는 SQL과 달리 dbmMetaManager 내에서만 동작하는 명령이다.

Internal command	Description
initdb	최초로 DICTIONARY INSTANCE를 생성해야 할 때 수행하는 명령이다.
list	현재 instance에 생성된 object를 출력한다.
desc [table name]	입력된 테이블의 상세 정보를 출력한다.
h / hist / history	현재까지 실행했던 명령의 목록을 출력한다. (최대 20개)
ed [number]	직전에 수행한 명령 또는 history에 저장된 목록 중에 입력된 번호의 명령을 편집한다. 환경 변수인 DBM_EDITOR에 설정된 편집기가 구동된다. (기본은 "vi" 로 설정된다.)
/ [number]	직전에 수행한 명령 또는 history에 저장된 목록 중에 입력된 번호의 명령을 재수행한다.
q / quit / exit	dbmMetaManager를 종료한다. (수행한 트랜잭션은 모두 rollback 처리된다.)
set vertical [on | off]	결과가 column/ line 단위로 출력되도록 설정한다.
struct out [table name]	테이블의 형상을 C 구조체에 맞게 출력한다.

Internal command	Description
set instance [instance name]	Multi instance 환경에서 instance를 전환할 때 사용한다.
set password [old pwd] [new_pwd]	instance에 old password를 new password로 변경한다. • 최초 설정시 old password는 "null" 로 입력한다. • new password가 "null" 로 입력될 경우 암호는 해제된다. • password 변경을 할 경우 old password와 일치해야 한다.
set index [table name] [index name]	입력된 테이블의 index를 사용하도록 설정한다.
startup [instance_name]	dbmCkpt에 의해 생성된 데이터 파일을 사용하여 instance를 복구한다.

주의

dbmMetaManager의 Internal command는 소문자로 입력해야 한다.

노트

SQL방식 처리구조는 리턴할 레코드를 모두 임시 메모리를 할당하여 저장하기 때문에 오류가 발생할 수 있다. 이 경우 처리(조회)할 레코드의 범위를 나누어 수행해야 한다.

Internal Commands

dbmMetaManager에서만 수행가능한 명령들을 설명한다.

list

현재 접속된 instance 이하의 object 목록을 출력한다.

```
dbmMetaManager(DEMO)> list;
OBJECT                                    MAX       TOTAL        USED        FREE
================================================================================
DUAL                                   102400        1024           1        1023
REPL_LOG                             10000000        1024           0        1024
REPL_UNSENT                          10000000        1024           0        1024
SEQ1                                        1           1           0           1
success
```

> 노트
> list 명령으로 출력된 결과 중 DIRECT TABLE 유형은 사용량을 표현할 수 없는 구조임으로 필요한 경우 "sele ct count(*) " 구문을 통해 확인해야 한다.

desc

테이블의 생성 정보를 화면에 출력한다.

```
dbmMetaManager(DICT)> desc dic_index_column;
-------------------------------------------------------------------------------
Instance=(DICT) Table=(DIC_INDEX_COLUMN) Type=(TABLE) RowSize=(136) LockMode(1)
-------------------------------------------------------------------------------
INST_NAME                       char                32          0
TABLE_NAME                      char                32          32
INDEX_NAME                      char                32          64
COLUMN_NAME                     char                32          96
KEY_COLUMN_ORDER                int                 4           128
COLUMN_ORDER                    int                 4           132
-------------------------------------------------------------------------------
IDX_DIC_INDEX_COLUMN            unique      (INST_NAME asc, TABLE_NAME asc, INDEX_NAME asc,
COLUMN_NAME asc, COLUMN_JSON asc, JSON_KEY_SIZE asc, KEY_COLUMN_ORDER asc)
-------------------------------------------------------------------------------
success
```

set index

테이블을 조회할 때 사용 할 특정 index를 지정한다.

```
dbmMetaManager(DEMO)> create table t1 (c1 int, c2 int);
success
dbmMetaManager(DEMO)> create unique index idx1_t1 on t1 (c1);
success
dbmMetaManager(DEMO)> create unique index idx2_t1 on t1 (c2);
success
dbmMetaManager(DEMO)> insert into t1 values (1, 10);
success
dbmMetaManager(DEMO)> insert into t1 values (2, 20);
success
dbmMetaManager(DEMO)> select * from t1 where c2 = 20;
```

```
--------------------------------------------------------------------------------
C1                   : 2
C2                   : 20
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> set index t1 idx2_t1;
success
dbmMetaManager(DEMO)> select * from t1 where c2 = 20;
--------------------------------------------------------------------------------
C1                   : 2
C2                   : 20
--------------------------------------------------------------------------------
1 row selected
```

노트

GOLDILOCKS LITE는 별도의 SQL 최적화 기능을 제공하지 않는다. INDEX를 사용하는 경우는 SQL의 WHERE절에 index key column을 모두 포함한 EQUAL 조건인 경우에만 사용되며, 그 외의 경우에는 모두 full scan 방식으로 동작한다.

set vertical [on/off]

dbmMetaManager 조회 결과는 기본적으로 column 별로 한 줄씩 출력된다. set vertical option을 통해 한 개 레코드 단위로 출력할 수 있다.

```
dbmMetaManager(DEMO)> set vertical off
success
dbmMetaManager(DEMO)> select * from t1
--------------------------------------------------------------------------------
             C1              C2 C3                                   C4
--------------------------------------------------------------------------------
              1               1 sunjesoft                             1
           1024            1024 hey hey hey                        1024
--------------------------------------------------------------------------------
2 row selected
```

OS Command 수행

dbmMetaManager에서 OS Command 등을 수행할 필요가 있을 때는 +를 표기한 후에 기술한다. 다음 예제를 참고한다.

```
dbmMetaManager(unknown)> + ls -lrt ${DBM_HOME};
합계 12
drwxrwxr-x. 2 lim272 lim272 4096 12월  5 12:59 sample
drwxrwxr-x. 2 lim272 lim272  136 12월 11 15:58 conf
drwxrwxr-x. 2 lim272 lim272  177 12월 19 10:47 include
drwxrwxr-x. 2 lim272 lim272   27 12월 19 10:47 lib
drwxrwxr-x. 2 lim272 lim272 4096 12월 19 10:47 bin
drwxrwxr-x. 2 lim272 lim272 4096 12월 19 11:19 trc
drwxrwxr-x. 2 lim272 lim272    6 12월 19 11:37 arch
drwxrwxr-x. 2 lim272 lim272    6 12월 19 11:43 wal
drwxrwxr-x. 2 lim272 lim272  169 12월 19 11:48 dbf
drwxrwxr-x. 2 lim272 lim272   66 12월 19 11:49 repl
success
```

원격 연결

dbmMetaManager에서 원격으로 연결해야 할 경우 다음과 같이 수행한다. 원격지에 dbmListener가 구동된 상태이어야 한다.

```
Connect := CONNECT <remote ip> <remote listen port number> <remote instance name>
```

다음은 Listener에 접속하여 instance를 생성하는 예이다.

```
dbmMetaManager(unknown)> connect 127.0.0.1 27584 dict
success
dbmMetaManager(127.0.0.1:DICT)> create instance demo
success
```

struct out (구조체 출력)

현재 생성되어 있는 테이블에 대한 C Type 구조체 형태를 출력한다.

```
dbmMetaManager(DEMO)> create table t1
(c1 int, c2 short, c3 long, c4 float, c5 double, c6 date, c7 char(33) );
success
dbmMetaManager(DEMO)> struct out t1;
```

```
typedef struct T1
{
    int C1;
    short C2;
    long long C3;
    float C4;
    double C5;
    struct timeval C6;
    char C7[33];
} T1
success
```

history

현재까지 수행된 명령 중에 최근 20개를 출력한다.

```
dbmMetaManager(DEMO)> history;
   0: select 1 from dual
   1: select sysdate from dual
success
```

"/" (재수행 명령)

직전에 수행한 명령 또는 history에 저장된 목록 중에 입력된 번호의 명령을 재수행한다.

```
dbmMetaManager(DEMO)> history;
   0: select 1 from dual
   1: select sysdate from dual
success
dbmMetaManager(DEMO)> /
--------------------------------------------------------------------------------
SYSDATE : 2025/01/08 08:19:35.806824
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> /0
--------------------------------------------------------------------------------
1 : 1
--------------------------------------------------------------------------------
1 row selected
```

ed

직전에 수행한 명령 또는 history에 저장된 목록 중에 입력된 번호의 명령을 편집한다.

```
dbmMetaManager(DEMO)> history;
   0: select 1 from dual
   1: select sysdate from dual
success
dbmMetaManager(DEMO)> ed
success
####################################
## vi 모드에서 아래와 같이 편집한 경우
##  select sysdate from dual ==> select sysdate, sysdate from dual
####################################
dbmMetaManager(DEMO)> /
--------------------------------------------------------------------------------
SYSDATE : 2025/01/08 08:20:52.439592
SYSDATE : 2025/01/08 08:20:52.439592
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)> history
   0: select 1 from dual
   1: select sysdate from dual
   2: select sysdate, sysdate from dual
success
```

> 노트
> 편집기는 환경 변수인 DBM_EDITOR 로 설정할 수 있다. 기본 값은 "vi" 로 설정된다.

set instance

지정한 instance로 접속을 전환한다.

```
dbmMetaManager(DICT)> set instance demo;
success
dbmMetaManager(DEMO)>
```

노트

전환 이전의 Instance에서 수행한 트랜잭션은 자동으로 롤백처리 된다.

set password

현재 Instance에 접근 암호를 설정/해제한다.

```
dbmMetaManager(DEMO)> set password null lite_pwd;    # 암호 지정
success
dbmMetaManager(DEMO)>  set password lite_pwd null;   # 암호 해제
success
```

- set password 첫번째 인자는 현재의 암호를 의미하며 두번째 인자는 변경할 암호를 의미한다.
- null은 암호를 설정하지 않은 상태 또는, 활성화 하지 않을 경우 지정한다.

암호가 설정된 이후에는 패스워드가 일치하지 않을 경우 아래와 같이 접속/수행이 불가능하다.

```
$ dbmMetaManager
************************************************************
* Copyright 2010. SUNJESOFT Inc. All rights reserved.
* Version (Debug 3.2-3.2.6 revision(6754))
************************************************************
dbmMetaManager(DEMO)> set password null lite_pwd;
success
dbmMetaManager(DEMO)> quit
$ dbmMetaManager
************************************************************
* Copyright 2010. SUNJESOFT Inc. All rights reserved.
* Version (Debug 3.2-3.2.6 revision(6754))
************************************************************
ERR] invalid passwd, use '-e' option as instance need passwd
$ dbmMetaManager -e abcd
************************************************************
* Copyright 2010. SUNJESOFT Inc. All rights reserved.
* Version (Debug 3.2-3.2.6 revision(6754))
************************************************************
ERR] invalid passwd, use '-e' option as instance need passwd
```

노트

API 및 제공되는 모든 Utility에 동일하게 적용됨으로 password 설정은 주의가 필요하다.
인증 API는 **dbmAuthorize**를 참조한다.

startup

모든 instance 또는 지정된 특정 instance를 복구한다. 복구를 수행하려면 디스크 모드로 운영되어야 하며, dbmCkpt에 의해 생성된 데이터 파일이 필요하다.

```
dbmMetaManager(DICT)> startup DEMO
success
```

노트

startup 과정은 Instance가 갖고 있는 DIC_TABLE에 저장된 내용을 기반으로 테이블을 복구한다. 따라서, create instance 시점에 만들어진 DIC_TABLE.dbf가 존재해야 한다.

1.6 복구 가이드

GOLDILOCKS LITE의 데이터 복구가 필요할 경우를 대비에 제공하는 방안에 대해 설명한다.

스냅샷(SnapShot) 저장 및 복원

dbmExp를 이용하여 전체 object/ data를 Text로 저장하고 dbmImp를 통해 object/ data를 복원한다. 복원의 시점은 dbmExp를 수행한 시점이다.

다음은 전체 instance의 모든 object/ data를 내려받는 예이다.

```
[lim272@tech10 tmp]$ dbmExp -a -d
[ DEMO ] Instance start...
   + (T1) (10 rows) download
   + (T2) (10 rows) download
   + (T3) (10 rows) download
   + (T4) (10 rows) download
   + (T5) (12 rows) download
```

명령을 수행한 경로에 아래의 예시와 같이 파일이 생성되며 각 파일의 설명은 표와 같다.

```
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_T1.fmt
-rw-rw-r--. 1 lim272 lim272 10221 Jun 24 14:38 DEMO_T1.dat
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_T2.fmt
-rw-rw-r--. 1 lim272 lim272 10221 Jun 24 14:38 DEMO_T2.dat
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_T3.fmt
-rw-rw-r--. 1 lim272 lim272 10221 Jun 24 14:38 DEMO_T3.dat
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_T4.fmt
-rw-rw-r--. 1 lim272 lim272 10221 Jun 24 14:38 DEMO_T4.dat
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_T5.fmt
-rw-rw-r--. 1 lim272 lim272  1485 Jun 24 14:38 DEMO_create.sql
-rw-rw-r--. 1 lim272 lim272 12265 Jun 24 14:38 DEMO_T5.dat
-rw-rw-r--. 1 lim272 lim272   180 Jun 24 14:38 DEMO_in.sh
-rw-rw-r--. 1 lim272 lim272    20 Jun 24 14:38 DEMO_create_proc.sql
```

형식	설명
<INSTANCE_NAME>_<OBJECT_NAME>.dat	테이블 별 데이터가 저장된다.
<INSTANCE_NAME>_<OBJECT_NAME>.fmt	테이블 별 로딩을 위한 컬럼 구성 정보가 저장된다.
<INSTANCE_NAME>_create.sql	object 생성 구문이 저장된다.

형식	설명
<INSTANCE_NAME>_create_proc.sql	Procedure 생성 구문이 저장된다.
<INSTANCE_NAME>_in.sh	instance의 전체 테이블에 대해 dbmImp를 일괄 수행하는 스크립트가 저장된다.

노트

LITE<->LITE로의 dbmExp/dbmImp를 사용할 때 binary 옵션을 활성화하면, 데이터 변환 과정 없이 바로 적재할 수 있어 업로드 성능이 향상된다. (단, binary 옵션은 다른 DBMS와는 호환되지 않는다.)

디스크 로깅을 이용한 복원

GOLDILOCKS LITE의 디스크 로깅을 이용한 복원은 다음의 과정을 통해 가능하다.
* 모든 트랜잭션은 커밋과 동시에 Redo log를 파일로 기록한다.
* dbmCkpt 유틸리티를 이용하여 Redo LogFile을 반영한 데이터파일을 생성한다.
* dbmMetaManager에서 "startup" 명령을 수행한다.
디스크 로깅은 아래의 표와 같은 방식을 제공한다.

DBM_LOG_CACHE_MODE	설명
NONE (0)	세션별로 로그 파일에 기록한다.
NVDIMM (1)	NVDIMM 매체에 기록할 경우 설정한다. 디스크로 저장은 dbmLogFluhser를 구동해야 한다.
SHM (2)	Shared memory에 기록한다. 디스크로 저장은 dbmLogFluhser를 구동해야 한다.

주의

Log Cache 모드에서 dbmLogFlusher가 작동하지 않으면 설정된 Log Cache 메모리 공간이 비워질 수 없어 트랜잭션들은 모두 대기한다.

디스크 기록은 buffered i/o 및 direct i/o 방식이 있으며 DBM_DIRECT_IO_ENABLE 속성을 통해 제어할 수 있다. Commit시점에 디스크로 저장을 보장하려면 DBM_COMMIT_WAIT_MODE 속성을 통해 제어할 수 있다. (**환경 변수 및 프로퍼티** 참조)

노트
성능과 안전성은 trade-off 관계에 있기 때문에 DBM_COMMIT_WAIT_MODE 속성은 필요한 경우에만 설정해야 한다.

체크포인트

디스크 로그 파일을 반영하여 데이터 파일을 생성 및 갱신하는 과정을 체크포인트라고 한다.
반영된 로그파일은 DBM_ARCHIVE_LOG_ENABLE의 설정에 따라 삭제되거나 지정된 archive 경로로 이동된다.

노트

- 체크포인트가 수행되는 동안은 빈번한 I/O가 발생하여 System에 영향을 줄 수 있다.
- 로그 파일이 기록되는 공간은 체크포인트 수행 간격 동안 사용량이 증가함으로 적정 공간 할당이 요구된다.

복구

디스크 로깅 및 체크포인트에 의해 복구하는 과정은 다음과 같이 수행한다.

```
(1) shell> dbmCkpt -i demo -f
(2) dbmMetaManager(unknown)> startup;
success
```

dbmCkpt는 기록이 완료된 로그파일만을 대상으로 데이터파일에 반영하기 때문에
(1)의 예시처럼 "-f" option을 이용해 현재 기록 중인 로그파일도 데이터파일에 반영하도록 한다.
(2)의 예시처럼 dbmMetaManager에서 "startup" 명령을 통해 복구를 수행한다.
Startup 과정은 Dictionary table을 기반으로 Object를 생성하고 데이터파일을 기반으로 데이터를 복원한다.

노트
운영 중 복구 과정에서 현재 로그파일을 반영하지 않은 상태에서 Startup 했다면 기존의 shared memory만을 제거한 후 다시 체크포인트 수행하고 startup 해야 한다.

Archive 로그를 가지고 체크포인트를 수행 할 경우에 반영할 로그파일의 시작번호를 아래와 같이 지정할 수 있다.

```
dbmMetaManager> alter system reset checkpoint demo -1;
```

1.7 Utility

GOLDILOCKS LITE에서 제공하는 사용자 utility에 대해 설명한다.

dbmExp

GOLDILOCKS LITE 내의 object 생성과 관련된 script와 데이터를 추출하는 utility 이다.

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-a	모든 instance를 export 할 경우에 지정하며 -i 옵션과 함께 지정할 수 없다.
-i <instance Name>	특정 instance를 지정할 경우에 입력한다.
-t <table Name>	특정 table을 지정할 경우에 입력한다. 입력하지 않을 경우 전체를 추출한다.
-r <delimeter>	데이터 추출 옵션이 활성화 된 경우 레코드 단위 구분자를 지정한다.
-c <delimeter>	데이터 추출 옵션이 활성화 된 경우 column 단위 구분자를 지정한다.
-d	데이터를 추출하고자 할 경우에 입력한다.
-b	Binary 형태로 데이터를 추출할 경우에 입력한다. 추후 dbmImp로 로딩할 때 parsing 비용을 줄일 수 있다. (타 DBMS와는 호환 불가한 형식이다.)
-n	column 타입이 char 형식일 경우, 데이터 내에서 null 이전까지의 값만 출력하고자 할 때 사용된다.
-p	instance에 password 가 설정된 경우 입력한다.

노트

- dbmExp에 의해 추출된 데이터는 text로 저장되며 다른 DBMS로 데이터를 이관하여 사용할 수도 있다.
- 별도 옵션 없이 사용자가 row와 column 구분자를 지정하지 않으면 csv 형식으로 저장한다.
- Column과 Row의 delimeter는 다른 값이여야 하고, 데이터에도 포함되서는 안된다.
- Date column은 기본적으로 microseconds까지 출력된다. 다른 DBMS로 이관 (적재) 할 경우 해당 DBMS에 제공하는 적절한 데이터 타입과 포맷을 사용해야 한다.

사용 예 (특정 instance의 하위 object와 데이터 추출)

```
$ dbmExp -i demo -d
[ DEMO ] Instance start...
   + (QUE1) (1 rows) download
   + (T1) (1 rows) download
```

- 생성되는 각 파일 이름은 <InstanceName>_<ObjectName> 형식이다.
- Sequence는 current 값을 start with 값으로 설정하여 출력한다.

dbmExp 추출 결과물

dbmExp를 통해 아래의 표와 같은 script 결과물이 생성된다.
생성되는 script 파일 형식은 <INSTANCE_NAME>_xxx.sh(sql) 이다.

File name	설명
DEMO_create.sql	Instance에 존재하는 object 생성 스크립트를 저장한다.
DEMO_create_proc.sql	Procedure 생성 스크립트를 저장한다.
DEMO_in.sh	모든 테이블의 데이터를 로딩하는 명령어를 저장한다.

dbmImp

dbmExp를 통해 추출된 데이터 또는 다른 DBMS로부터 추출된 구분자를 가진 text 형태의 데이터를 분석하여 적재하는 utility 이다.

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-i <instance Name>	대상 instance name을 지정한다.
-t <table Name>	대상 table name을 지정한다.
-r <delimeter>	레코드 단위 구분자를 지정한다.
-c <delimeter>	Column 단위 구분자를 지정한다.
-d <data file name>	데이터 파일 이름을 지정한다.
-f <form file name>	Form file 이름을 지정한다.
-p	instance에 password가 설정된 경우 입력한다.
-b	데이터 파일이 dbmExp에 의해 만들어진 binary format일 경우에 지정한다. (Parsing 비용을 줄일 수 있다.)

노트

Form file 정보를 수정하여 특정 column을 올리지 않거나 순서를 변경하여 데이터를 업로드할 수 있다.

구분자를 특수문자로 사용할 경우 아래의 입력방식을 사용하도록 한다.

구분자	Ascii 값	입력방식 (on dbmImp)
\t (tab)	9	\\t
\r (carriage return)	13	\\r
\n (line feed)	10	\\n

사용 예

```
$ dbmImp -i demo -t t1 -d DEMO_T1.dat
(DEMO.T1) Import Success. (ReadCount=1, Inserted=1)
$ dbmImp -i demo -t que1 -d DEMO_QUE1.dat
(DEMO.QUE1) Import Success. (ReadCount=1, Inserted=1)
```

추출된 form file을 이용하여 데이터에 존재하는 특정 column을 올리지 않으려면 Y 항목을 N으로 변경한다.

```
$ cat DEMO_T1.fmt
C1 Y
C2 Y
C3 Y
C4 Y
C5 Y
C6 Y    ❶ N 으로 변경
C7 Y
C8 Y
C9 Y
C10 Y
```

- 데이터 파일에 C2, C3 column이 없다면 위 예제에서는 C2, C3 line을 제거한 후에 수행한다.
- 데이터 파일에 C2, C3 column의 순서가 바뀌어 저장된 경우, C2, C3 line을 서로 바꾸어 처리한다.

노트

DATE 타입은 기본적으로 YYYY/MM/DD HH:MI:SS.SSSSSS 형태의 문자열을 기반으로 데이터를 해석하기 때문에 다른 DBMS에서 데이터를 추출하여 올려야 할 경우 해당 DBMS에서 동일한 형식으로 DATE를 문자

열로 추출해야 한다.

dbmCkpt

체크포인트를 수행하는 utility이다.

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-i <instance name>	대상 instance name을 지정한다.
-v	한 번만 수행한 후에 종료한다.
-f	체크포인트 대상 logfile이 다 쓰여지지 않은 상태라도 강제로 체크포인트를 수행할지 여부를 지정한다. (Startup 직전 수행시점에 반드시 옵션을 포함한 체크포인트가 필요하다.)
-s	체크포인트를 수행하는 간격 (단위: 초)

dbmDump

dbmDump는 Shared Memory의 내용 및 디스크에 저장된 파일의 내용을 출력하는 tool이다.
입력 옵션과 파일 유형에 해당되는 내용을 출력한다.

Dump Memory

현재 메모리에 있는 Instance, table, index의 주요 정보들을 dbmDump를 통해 확인할 수 있다.

연관 object	입력 가능 옵션	설명
ALL	-h	도움말을 출력한다.
instance, table, index	-i <instance name>	instance name을 지정한다.
instance	-x <session ID>	Instance dump 수행 시 입력한 session ID와 관련된 정보만 출력할 경우에 지정한다.
table	-t <table name>	대상 table name을 지정한다.
table	-s	slot-area 정보를 출력한다.
index	-d <index name>	대상 index name을 지정한다.

dbmDump (Instance)

Instance의 헤더 정보 및 각 세션 별 주요 정보를 dump한다.

사용 예

```
[majaehwa@tech9 new_lite]$ dbmDump -i demo;
InstName=DEMO, TransId=-1
Segment Init=128, extend=128, max=1048576
SegmentNo=0, Alloc=12, Free=139, Gap=128
Lock=-1, SCN=29, ObjectId=12, Name=DEMO, Active=1,DiskLogMode=1, LogFileSize=104857600,
CreateTime=2025-07-24 15:17:12
================================================================================================
==============
TxInfo(T=1, P:3560962, T=3560962), Stat=transaction, First/Last(11:11), SavePoint(-1:-1)
WaitTrans=-1(,-1), SCN=9223372036854775807, Repl=0, BeginTime=2025/07/28 15:50:27.098864
Page=11, PrevOffset=-1, Offset=32, NextLogPage=11, NextLogOffset=32, Size=0, LogType=INSERT_
TABLE, RelSlot=0, RelExtra=1, ObjectId=(12)
Page=11, PrevOffset=32, Offset=256, NextLogPage=11, NextLogOffset=256, Size=80, LogType=UPDATE
_TABLE, RelSlot=0, RelExtra=1, ObjectId=(12)
================================================================================================
==============
```

dbmDump (Instance)는 instance header 정보와 각 session 별 트랜잭션 정보를 출력한다.
Instance header에 해당하는 정보들은 다음 표와 같다.

항목	설명
Lock	Instance lock 설정 여부
SCN	System commit number 용도
ObjectId	Reserved
Name	Instance 이름
Active	Reserved
DiskLogMode	디스크 모드 활성화 여부
LogFileSize	디스크 모드 일때, 로그를 기록하는 파일 사이즈
CreateTime	Instance 생성 시각

Session 별로 기본 출력 정보는 다음 표와 같다.

항목	설명
TxInfo(T: P: T)	Session 정보 • T: 내부에서 할당된 session ID • P: Process ID • T: Thread ID

항목	설명
Stat	트랜잭션 상태 (Stat=transaction말고는 본적이 없음.) • transaction: 트랜잭션 진행 가능 또는 진행 중 • commit start: 메모리 커밋 시작 • commit completed: 메모리 커밋 완료 • rollback completed: 메모리 롤백 완료 • recovery completed: 메모리 비정상 복구 완료
First/ Last	Session이 사용 중인 undo 공간의 첫 번째와 마지막 PageId
SavePoint	Reserved
WaitTrans	트랜잭션이 LockWait 상태일 경우에 대기하는 상대편 transactionId(대상 테이블, 대상 slot Id)
SCN	Commit 시점에 채번된 SCN
Repl	이중화 모드 설정 여부
BeginTime	세션이 할당된 시간

Session 현재 진행 중인 트랜잭션이 있을 경우 출력되는 항목은 다음 표와 같다.

항목	설명
Page	현재 트랜잭션 로그가 기록된 현재 page ID
PrevOffset	이전 트랜잭션 로그가 기록된 page 내의 offset 정보
Offset	현재 트랜잭션 로그가 기록된 page 내의 offset 정보
NextLogPage	다음 트랜잭션 로그가 기록된 page ID
NextLogOffset	다음 트랜잭션 로그가 기록될 page 내의 offset 정보
Size	Rollback image의 크기
LogType	트랜잭션 로그 유형
RelSlot	Table내에 slot ID
RelExtra	Table내의 고유한 record ID
ObjectId	Object에 부여되는 고유한 ID

dbmDump (Table)

테이블의 Header 상태 정보 및 레코드가 저장된 Slot영역의 Row Header와 데이터를 Dump한다.

사용 예

```
[majaehwa@tech9 wal]$ dbmDump -i demo -t t1;
InstName=DEMO, ObjectId=11, TableName=T1, Lock=-1, RowSize=4, CreateTime=2025-07-24 15:17:24,
CreateSCN=22, ExtraKey=2, Root=-1
1] C1 int(Type=2) 4 0
--------------------------------------------------------------------------------
SlotID=0, sExtraKey=1, Lock=13313, Size=4, SCN=24
C1 = [4]
```

```
SlotID=1, sExtraKey=2, Lock=14337, Size=4, SCN=25
C1 = [2]
```

출력 상단은 Table Header를 표현하며 의미는 아래와 같다.

항목	설명
InstName	테이블이 속한 Instance
ObjectId	테이블의 Object ID
Lock	테이블 Lock 정보
RowSize	테이블에 저장되는 레코드 크기
CreateTime	생성 시각
CreateSCN	생성 시점의 SCN
ExtraKey	레코드 저장시점마다 채번되는 고유번호
Root	Splay Table인 경우 Root Slot Id로 사용

레코드 별로 출력 되는 항목은 다음과 같다.

항목	설명
SlotID	테이블 내의 위치 정보이다. 테이블은의 레코드는 고정크기로 저장된 배열과 같으며 segment내의 N번째 저장 위치를 의미한다.
sExtraKey	레코드의 고유ID
Lock	현재 혹은, 직전에 Lock을 점유한 TransID
Size	레코드 크기
SCN	레코드의 커밋번호

노트

Splay table의 경우 Row header에 tree 구조를 저장하고 있다. Table Header에 출력된 "Root" 항목에 저장된 부분이splay tree 구조 내에 최상위(시작점)를 의미한다.

```
[majaehwa@tech9 src]$ dbmDump -i demo -t t4;
InstName=DEMO, ObjectId=14, TableName=T4, Lock=-1, RowSize=4, CreateTime=2025-07-29 14:40:34,
CreateSCN=41, ExtraKey=1, Root=0
1] C1 int(Type=2) 4 0
--------------------------------------------------------------------------------
SlotID=0, ExtraKey=1, Lock=31745, Size=4, SCN=45, parent=-1, left=-1, right=-1
C1 = [5]
```

dbmDump (Index)

Btree Index의 헤더정보 및 Index가 저장된 형태를 dump한다.

사용 예

```
[majaehwa@tech9 dbf]$ dbmDump -i demo -t t1 -d idx_t1;
InstName=DEMO, TableName=T1, IndexName=IDX_T1
try to attach a index segment
Lock=-1, RootNodeId=0, Unique=0, CompareCount=1, Depth=0, SplitCount=0, RefCount=0, CreateTime
=2025-07-24 15:21:12
==== NODE (0) : Valid=1 Cnt:2 (Prev:-1, Next:-1)
SlotNo=0] DataSlotId=1 ExtraKey=2 (Left=-1, Right=-1)
Key(C1) Val=[2]
SlotNo=1] DataSlotId=0 ExtraKey=1 (Left=-1, Right=-1)
Key(C1) Val=[4]
```

Index Header의 주요 항목은 아래와 같다.

항목	설명
Lock	현재 Index Header에 Lock을 점유한 TransId 이며 이 값이 지속적으로 "-1"이 아닌 특정 값으로 반복된다면 Index를 재구축해야 한다.
RootNodeId	root node의 ID이다.
Unique	0 : non-unique 1 : unique
CompareCount	Index가 사용될 때마다 증가한다.
Depth	Btree의 깊이를 표현한다.
SplitCount	Leaf Node가 Split될 때마다 증가한다.
RefCount	현재 Index를 탐색 중인 트랜잭션의 개수
CreateTime	Index가 생성된 시간

> 노트
> DataSlotId 에 해당하는 데이터를 보기 위해서는 **dbmDump (Table)**를 참조한다.

Dump File

Anchor, Data, log 파일을 Dump 하여 주요 정보를 확인할 수 있다.

입력 옵션	설명
-h	도움말을 출력한다.
-f <anchor filename, data filename, log filename>	대상 filename을 지정한다

dbmDump (Anchor)

Log anchor file을 dump한다.

사용 예

```
[majaehwa@tech9 wal]$ dbmDump -f DEMO.anchor
File Info : Anchor, version 1
MemAnchor(Trans=1): mLogFileNo=0, LogFileOffset=9728, CkptFileNo=-1, ArchiveFileNo=-1,
CaptureFileNo=-1
MemCacheInd: CacheWriteInd=0, CacheReadInd=0, LogFileNo=0, LogFileOffset=0
LogAnchor(Trans=1): mLogFileNo=0, LogFileOffset=9728, CkptFileNo=-1, ArchiveFileNo=-1,
CaptureFileNo=-1
LogCacheInd: CacheWriteInd=0, CacheReadInd=0, LogFileNo=0, LogFileOffset=0
```

Log Anchor 파일은 세션 별 로깅방식과 Log Cache방식에 대한 정보를 기록하고 있다.
출력 결과의 해석은 각 디스크 로깅 방식 설정에 따라 다르다.

DBM_LOG_CACHE_MODE가 0 인 경우 MemAnchor의 주요 항목들은 다음과 같다.

항목	설명
MemAnchor(Trans=N)	세션 ID "N"을 의미한다.
mLogFileNo	세션이 사용 중인 로그 파일의 번호 이다.
LogFileOffset	세션이 사용 중인 로그 파일의 Offset이다.
CkptFileNo	체크포인트가 수행해야 하는 파일의 시작 번호 이다.
ArchiveFileNo	아카이빙을 수행해야 하는 파일의 시작 번호 이다.
CaptureFileNo	Reserved

DBM_LOG_CACHE_MODE가 (1, 2) 인 경우 MemCacheInd의 주요 항목들은 다음과 같다.

항목	설명
CacheWriteInd	Log Cache공간내에 기록할 수 있는 위치 정보
CacheReadInd	dbmLogFlusher 가 Flush 할 때 읽어야 하는 위치 정보
LogFileNo	dbmLogFlusher가 기록 중인 파일 번호 이다.
LogFileOffset	dbmLogFlusher가 기록 중인 파일의 위치이다.

> 노트
> Log Anchor는 mmap방식으로 메모리에 공유되며 파일에도 기록된다. "Mem" prefix가 붙은 출력 라인은 메모리를 dump 한 결과이며 prefix가 없는 라인은 실제 파일에 저장된 정보를 출력한 것이다.

dbmDump (Datafile)

Datafile을 dump한다.

사용 예

```
[majaehwa@tech9 dbf]$ dbmDump -f DEMO_T1.dbf
File Info : Data, version 1
InstName=DEMO, TableName=T1, SlotSize=76, CreateSCN=22
ColumnName (C1), Order=1, Type=int, Offset=0, Size=4
0) Size=4, SCN=24                          ## 0) 0의 의미는 SlotID이다.
(C1=4)
1) Size=4, SCN=25
(C1=2)
Dump (DEMO_T1.dbf) completed
```

dbmDump (Logfile)

Redo logfile의 주요 정보를 dump한다.

사용 예

```
[majaehwa@tech9 wal]$ dbmDump -f DEMO.1.0 | tail
   + logType(CREATE_TABLE), object(V$TYPE_INFO), SlotId(-1), sPos(48)
sFileOffset=8192, BlockSCN=22, BlockCount=1, logCount=1, Time=2025/07/24 15:17:24.641676
   + logType(CREATE_TABLE), object(T1), SlotId(-1), sPos(48)
sFileOffset=8704, BlockSCN=24, BlockCount=1, logCount=1, Time=2025/07/24 15:17:32.444694
   + logType(INSERT_TABLE), object(T1), SlotId(0), sPos(48) RowHdr(scn=24, size=4) :
sFileOffset=9216, BlockSCN=25, BlockCount=1, logCount=1, Time=2025/07/24 15:17:38.542943
   + logType(INSERT_TABLE), object(T1), SlotId(1), sPos(48) RowHdr(scn=25, size=4) :
sFileOffset=9728, BlockSCN=26, BlockCount=1, logCount=1, Time=2025/07/24 15:21:12.871522
   + logType(CREATE_INDEX), object(IDX_T1), SlotId(-1), sPos(48)
sReadSz=0, LastOffset=10240, errno=0 at reading blockHdr
```

Redo Log는 커밋시점마다 저장해야 할 트랜잭션 내의 모든 로그를 1개의 Block이라 불리는 공간에 기록하여 저장된다. 1개의 Block 크기는 DBM_DISK_BLOCK_SIZE 속성으로 지정된다.

각 Block마다 Header를 가지며 주요 항목은 아래와 같다.

항목	설명
sFileOffset	log가 기록된 파일 내 위치
BlockSCN	Commit SCN
BlockCount	Block의 개수
logCount	Block내의 세부 트랜잭션 로그의 개수
Time	트랜잭션 로그를 디스크에 기록하기 직전 시각

Block 헤더 정보 이후의 로그들은 트랜잭션 내의 상세 로그를 의미한다.

항목	설명
logType	트랜잭션 로그의 상세 유형
object	트랜잭션이 일어난 대상 Object
SlotId	대상 Object 내의 Slot ID
sPos	현재 읽은 Block의 Header 크기
RowHdr(SCN, Size)	커밋된 레코드의 SCN 및 크기 정보

dbmListener

원격 노드에 접속을 수행할 경우 대상 서버에서 구동해야 하는 프로세스이다.
* 원격 노드에서 접속하는 세션 관리
* 원격 노드에 요청되는 트랜잭션 처리

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-i <instance name>	dbmListener가 사용할 Property Section을 지정한다.
-v	verbose mode로 동작한다.

dbmListener를 구동한 후에 사용자가 접속하려면 다음과 같다.

```
dbmMetaManager> connect <ip> <port> <instanceName> ;
```

또는

dbmConnect API 사용(**dbmConnect** 참조)

dbmLogFlusher

디스크 로깅모드에서 Log cache 방식을 사용할 때 Cache내의 트랜잭션 로그를 디스크로 저장하는 프로세스이다.

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-i <instance name>	대상 instance name을 지정한다

dbmMonitor

GOLDILOCKS LITE의 상태를 모니터링 하는 프로세스이다.

Input Option

입력 옵션	설명
-h	도움말을 출력한다.
-i <instance name>	대상 instance name을 지정한다
-s <interval time>	모니터링 결과 출력 간격을 설정한다. (second 단위)
-u	Usage Info 출력값이 입력된 값(백분율)을 넘는 경우의 테이블만 표시한다.
-v	verbose mode로 구동된다.

사용 예

```
[nh39@tech9 new_lite]$ dbmMonitor -i demo
2025/07/28 20:11:29
[Table Usage]
TableName                                MAX           TOTAL            USED
FREE  Usage Info
-------------------------------------------------------------------------------------------------
--------------
DIC_CAPTURE_HOST                      4096000            1024               0
1024   0.00
DIC_CAPTURE_TABLE                     4096000            1024               0
1024   0.00
```

```
DIC_COLUMN                              4096000           1024            176
848   0.00
DIC_INDEX                               4096000           1024             15
1009   0.00
DIC_INDEX_COLUMN                        4096000           1024             41
983   0.00
DIC_INST                                4096000           1024              1
1023   0.00
DIC_PROCEDURE                           4096000           1024              0
1024   0.00
DIC_PROCEDURE_TEXT                      4096000           1024              0
1024   0.00
DIC_REPL_INST                           4096000           1024              0
1024   0.00
DIC_REPL_TABLE                          4096000           1024              0
1024   0.00
DIC_SEQUENCE                            4096000           1024              0
1024   0.00
DIC_TABLE                               4096000           1024             25
999   0.00
DIC_USER_LIBRARY                        4096000           1024              0
1024   0.00
DIC_USER_LIBRARY_FUNCTION               4096000           1024              0
1024   0.00
DIC_USER_LIBRARY_PARAMETER              4096000           1024              0
1024   0.00
-------------------------------------------------------------------------------------------------
---------
[Lock Status]
 WaitSessionId   WaitTransPID   LockTransId   LockTransPID LockObject   LockSlot
-------------------------------------------------------------------------------
            2         599736         31745         599672 USER_DATA           0
-------------------------------------------------------------------------------
```

Lock Status 각 항목은 다음과 같다.

항목	설명
WaitSessionId	Lock wait 중인 세션ID
WaitTransPID	Lock wait 중인 프로세스 ID
LockTransID	Lock을 점유한 트랜잭션 ID
LockTransPID	Lock을 점유한 프로세스 ID

항목	설명
LockObject	대상 테이블
LockSlot	대상 테이블 내의 SlotID

dbmReplica

Slave Node에서 데이터를 수신/반영하기 위해 구동하는 프로세스이다.

Input Option

입력 옵션	설명
-i	dbmReplica를 여러 개 구동해야 할 경우 프로세스 구분 목적의 Alias를 지정한다.
-h	도움말을 출력한다.
-v	verbose mode로 구동한다.

사용 예

```
$ dbmRepica -i demo
```

노트

dbmReplica는 구동 시점에 dbm.cfg내의 "COMMON" section 이나 환경변수에 설정된 속성을 사용한다.

1.8 Sizing

GOLDILOCKS LITE를 사용하기 전에 필요한 Memory Segment에 대한 Sizing 방안을 설명한다.

> 주의
>
> - 각 설명은 Segment 생성옵션의 init size만 설명하며 extend/max를 고려해서 추가 산정이 필요할 수 있다. (extend 단위의 계산식은 동일하다.)
> - Sizing 방식은 패치 및 신규 버전 릴리즈 시점에 개선 및 기능 추가로 인해 변경될 수 있다.

Instance Sizing

Instance segment는 세션 정보 및 Undo Logging 공간으로 사용된다.
* 세션은 헤더 공간에 고정 크기로 포함된다.
* Undo Logging 공간은 1개당 1M의 크기를 가진다.
* 세션은 dbmInitHandle 시점에 1개의 Undo Logging공간을 할당 받는다.

```
Instance Sizing =  sizeof(segment Header area) +          // 240 byte
                   sizeof(session area) +                 // 1392640 byte
                   sizeof(long long) * (init_size + 1) +
                   (1M * init_size)
                  ;
```

Table Sizing

Table의 크기를 계산할 때 table header, row header 등을 고려해야 한다.

```
Table Sizing = sizeof(segment header)  +          // 240 byte
               sizeof(table header)    +          // 136 byte
               sizeof(long long) * (init_size + 1) +
               ( (Record Header=72byte) + RecordSize ) * init_size
               ;
```

주의
디스크 모드 사용 시 생성될 데이터 파일의 크기는 table segment의 크기이다.

Index Sizing

테이블 타입별로 아래의 표와 같다.

Table 유형	설명
Btree	아래 식 참조
Splay	별도의 공간 산정을 하지 않음
Store	Btree Index와 동일
Queue	Btree Index와 동일
Direct	별도의 공간 산정을 하지 않음

Btree Index의 크기는 아래와 같이 산정한다.

```
init size = Table Segment Init Size * 4 + 4
Index Sizing = sizeof(segment header)  +          // 240 byte
               sizeof(index header) +             // 1192 byte
               ( sizeof(indexNodeHeader) +        // 32 byte
                 sizeof(index slot Header) +      // 32 byte
                 index key column size의 합 ) * 128 * init size
```

노트
Splay, direct table 유형은 dummy index segment 만 생성하며 크기는 6K이다.

디스크 공간 산정

디스크 모드로 운영할 경우 다음의 파일이 생성된다.
* 트랜잭션 로그 파일
* 데이타 파일
이중화 운영모드인 경우 master 노드에서는 다음의 파일이 생성될 수 있다.
* 이중화 미전송 로그

트랜잭션 로그파일

트랜잭션 로그파일은 체크포인트 과정을 통해 삭제되거나 Archive 경로로 이동된다.
따라서, 체크포인트 수행 간격 동안 디스크 공간이 부족하지 않도록 산정한다.
디스크 공간이 부족하여 트랜잭션 로그가 Commit시점에 기록되지 못할 경우 트랜잭션은 실패로 종료하며 모두 롤백 처리된다.
1개의 트랜잭션 로그파일 최대 크기는 DBM_DISK_LOG_FILE_SIZE 속성으로 정의한다.
트랜잭션 처리량, 체크포인트 간격, 디스크 장비의 I/O 처리속도에 따라 크기를 산정해야 한다.

데이터파일

데이터파일은 Table Segment가 모두 Extend되었을 때를 가정하여 최대 크기가 저장 가능한 공간으로 산정한다.

이중화 미전송 로그파일

이중화 모드로 운영 시 네트워크 장애에 의해 master 노드에 다음의 파일이 생성된다.
* 미전송 트랜잭션 로그 파일

미전송 트랜잭션 로그파일은 사용자에 의해 전송 처리가 완료되면 제거된다. (**alter system replication sync** 참조)
로그 파일의 최대 크기는 DBM_REPL_UNSENT_LOGFILE_SIZE 속성으로 정의한다.
사용자는 네트워크 장애 처리가 완료될 수 있을 것으로 예상되는 시간을 고려하여 공간을 산정한다.

1.9 Monitoring

본 장에서는 GOLDILOCKS LITE을 모니터링하는 방법들에 대해 설명한다.

LOCK 정보 확인

모든 변경 연산들에 대해 record 단위로 lock을 점유하여 다른 세션으로부터의 데이터가 변경되지 않도록 보호한다. 세션이 장시간 대기하는 경우에 원인을 찾기 위해 다음 방법을 사용한다.

```
dbmMetaManager(DEMO)> select * from v$session;
--------------------------------------------------------------------------------
ID                    : 1
TRANS_ID              : 26625
PID                   : 3388915
TID                   : 3388915
OLD_TID               : 3388915
VIEWSCN               : 9223372036854775807
CURR_UNDO_PAGE        : 4
FIRST_UNDO_PAGE       : 4
LAST_UNDO_PAGE        : 4
SAVEPOINT_UNDO_PAGE   : -1
SAVEPOINT_UNDO_OFFSET : -1
WAIT_TRANS_ID         : 2                     <<= Lock을 점유한 트랜잭션 ID
WAIT_OBJECT           : T1                    <<= Lock OBJECT
WAIT_SLOT_ID          : 0                     <<= Object 내의 레코드 위치
SESSION_STATUS        : transaction
IS_LOGGING            : 0
LOGFILE_NO            : -1
CKPT_NO               : -1
IS_REPL               : 0
REPL_SEND_SCN         : -1
REPL_RECV_ACK_SCN     : -1
AUTOCOMMIT_MODE       : 0
BEGIN_TIME            : 2025/07/18 06:14:09
PROGRAM               : dbmMetaManager
REMOTE_PID            :
REMOTE_ADDR           :
REMOTE_PROGRAM        :
```

```
--------------------------------------------------------------------------------
ID                    : 2
TRANS_ID              : 2
PID                   : 3389780
TID                   : 3389780
OLD_TID               : 3389780
VIEWSCN               : 9223372036854775807
CURR_UNDO_PAGE        : 5
FIRST_UNDO_PAGE       : 5
LAST_UNDO_PAGE        : 5
SAVEPOINT_UNDO_PAGE   : -1
SAVEPOINT_UNDO_OFFSET : -1
WAIT_TRANS_ID         : -1
WAIT_OBJECT           :
WAIT_SLOT_ID          : -1
SESSION_STATUS        : transaction
IS_LOGGING            : 0
LOGFILE_NO            : -1
CKPT_NO               : -1
IS_REPL               : 0
REPL_SEND_SCN         : -1
REPL_RECV_ACK_SCN     : -1
AUTOCOMMIT_MODE       : 0
BEGIN_TIME            : 2025/07/18 06:26:58
PROGRAM               : dbmMetaManager
REMOTE_PID            :
REMOTE_ADDR           :
REMOTE_PROGRAM        :
--------------------------------------------------------------------------------
2 row selected
```

위 결과에서 WAIT_TRANS_ID 항목이 -1이 아닌 경우 대상 TransID를 가진 세션을 기다리고 있는 상태를 의미한다. 위의 예제에서는 ID=1 세션이 ID=2 세션을 대기하는 것을 의미한다. 대기 중인 대상 테이블은 T1 이고 0번 slot을 가진 record에서 대기하고 있다는 의미이다.

위의 예에서처럼 TransID=2인 세션이 무엇을 하고 있는지는 다음과 같이 조회할 수 있다.

```
dbmMetaManager(DEMO)> select * from v$transaction where trans_id = 2;
--------------------------------------------------------------------------------
TRANS_ID    : 2
TRANS_SEQ   : 1
TRANS_TYPE  : UPDATE_TABLE
```

```
OBJECT_NAME : T1
SLOT_ID     : 0
EXTRA_KEY   : 1
COMMIT_FLAG : 0
SKIP_FLAG   : 0
VALID_FLAG  : 1
-------------------------------------------------------------------------------
1 row selected
```

SlotID=0에 해당하는 record를 변경하고 있으며 commit/ rollback을 하지 않은 상태가 유지되고 있다. TransID=2를 가진 세션이 어떤 프로세스인지 v$session의 PID, PROGRAM 컬럼을 통해 확인할 수 있다.
(원격노드에서 접속한 경우 REMOTE_PID, REMOTE_PROGRAM 항목을 통해 확인한다.)

> 노트
> 각 항목의 의미는 **V$SESSION** 및 **V$TRANSACTION**을 참조한다.

처리량 확인

GODILOCKS LITE는 각 주요 operation에 대한 누적 정보를 기록할 수 있다. 다음과 같이 전체 누적량을 확인할 수 있다.

```
dbmMetaManager(DEMO)> select * from v$sys_stat;
-------------------------------------------------------------------------------
STAT_NAME   : init_handle_op
ACCUM_COUNT : 0
-------------------------------------------------------------------------------
STAT_NAME   : free_handle_op
ACCUM_COUNT : 0
-------------------------------------------------------------------------------
STAT_NAME   : prepare_op
ACCUM_COUNT : 0
-------------------------------------------------------------------------------
STAT_NAME   : execute_op
ACCUM_COUNT : 0
-------------------------------------------------------------------------------
STAT_NAME   : insert_op
ACCUM_COUNT : 0
```

```
--------------------------------------------------------------------------------
STAT_NAME   : update_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : delete_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : scan_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : enqueue_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : dequeue_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : aging_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : commit_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : rollback_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : recovery_rollback_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : recovery_commit_op
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : split_index_node
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
STAT_NAME   : retry_lock_count
ACCUM_COUNT : 0
--------------------------------------------------------------------------------
17 row selected
```

노트
각 항목의 의미는 V$SYS_STAT을 참조한다.

Log Cache 및 Checkpoint 상태

디스크 로깅과 관련된 설정 및 동작 상태를 출력한다.

```
dbmMetaManager(DEMO)> select * from v$log_stat;
-------------------------------------------------------------------------------
DISKLOG_ENABLE       : 0
CACHE_MODE           : 0
DIRECT_IO_ENABLE     : 0
ARCHIVE_ENABLE       : 0
CURR_FILE_NO         : -1
CURR_FILE_OFFSET     : -1
LAST_CKPT_FILE_NO    : -1
LAST_ARCHIVE_FILE_NO : -1
LAST_CAPTURE_FILE_NO : -1
LOGCACHE_WRITE_IND   : -1
LOGCACHE_READ_IND    : -1
FLUSHER_FILE_NO      : -1
FLUSHER_FILE_OFFSET  : -1
LOG_DIR              : /mnt/md1/ssd_home/lim272/new_lite/pkg/wal
DATAFILE_DIR         : /mnt/md1/ssd_home/lim272/new_lite/pkg/dbf
ARCHIVE_DIR          : /mnt/md1/ssd_home/lim272/new_lite/pkg/arch
-------------------------------------------------------------------------------
1 row selected
```

노트
각 컬럼의 의미는 V$LOG_STAT 을 참조한다.

2.

API Reference

2.1 API 공통사항

- 모든 API 호출은 정상 처리 결과를 "0"으로 반환하며 그 외에는 에러코드를 반환한다.
- Compile 할 때 $DBM_HOME/include/dbmUserAPI.h을 사용자 소스코드에 포함해야 한다.
- Linking 할 때 $DBM_HOME/lib/libdbmCore.so을 사용자가 작성한 Makefile 내에 포함해야 하며 사용자 환경 변수인 LD_LIBRARY_PATH에도 추가해야 한다.
- 모든 API 동작은 non-autoCommit 모드이기 때문에 select를 제외한 트랜잭션 마지막에는 dbmCommit 또는 dbmRollback을 호출해야 반영이 완료된다.
- API는 TableName을 이용하거나 dbmTableHandle을 이용하는 두 가지 유형이 있으므로 API 상세 spec을 참고하여 적절한 유형을 사용해야 한다.

2.2 C/C++ APIs

dbmInitHandle

기능

API 사용을 위한 handle 초기화 작업을 수행한다. 모든 API를 사용하기 위해 반드시 선행되어 호출되어야 한다. dbmInitHandle 내에서는 아래의 과정을 내부적으로 수행한 후 성공/실패를 반환한다.

- Process 정보 저장
- Undo space 할당
- Dictionary 준비
- 트랜잭션 처리를 위한 내부 공간 할당

인자

```
int dbmInitHandle( dbmHandle     ** aHandle,
                   char           * aInstanceName )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle **	In/ out	변수는 NULL로 초기화한 후에 사용해야 한다.
aInstanceName	char *	In	-

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );
}
```

> 노트
> dbmHandle 변수는 (void *) 형태로 dbmInitHandle을 통해 내부적으로 필요한 공간을 할당하여 사용자에게 반환한다.

dbmConnect

원격 노드 접속을 수행한다.

인자

```
int dbmConnect( dbmHandle     ** aHandle,
                const char     * aTargetIP,
                int              aTargetPort,
                const char     * aInstName )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle **	In/ out	변수는 NULL로 초기화한 후에 사용해야 한다.
aTargetIP	const char *	In	접속할 원격노드의 IP
aTargetPort	int	In	접속할 원격노드에 구동된 dbmListener의 PortNo
aInstanceName	const char *	In	접속할 원격노드의 대상 Instance name

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
```

```
    int             rc;
    rc =  dbmConnect( &sHandle, "127.0.0.1", 27584, "demo" )
}
```

노트

- 원격노드에서 dbmListener 가 구동된 상태여야 한다.
- Handle의 해제는 dbmFreeHandle을 이용한다.

dbmFreeHandle

기능

dbmInitHandle에 의해 할당된 자원을 모두 해제한다.
만일 변경 트랜잭션을 commit 하지 않은 상태에서 dbmFreeHandle이 호출되면 자동으로 rollback을 먼저 수행한다.
dbmFreeHandle을 호출하지 않고 프로그램을 종료할 수 있지만 dbmFreeHandle 없이 dbmInitHandle를 계속 호출하면 메모리가 증가하며 동시에 접속 가능한 공간이 부족한 오류가 발생할 수 있다.

인자

```
int dbmFreeHandle( dbmHandle     ** aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle **	In/ out	해제 후 변수는 NULL로 초기화 된다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );
    // 사용자코드
    rc = dbmFreeHandle( &sHandle );
}
```

dbmPrepareTable

기능

사용자가 지정한 table을 handle에 준비시킨다. (Shared memory attach 등을 수행)

인자

```
int dbmPrepareTable( dbmHandle       * aHandle,
                     char            * aTableName );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수가 사용되어야 한다.
aTableName	char *	In	Prepare할 TableName을 입력한다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTable( sHandle,
                          "table1" );
}
```

dbmPrepareTableHandle

기능

사용자가 지정한 table에 대해서 TableHandle을 생성하여 반환한다. (Shared memory attach 등을 수행한다.)

인자

```
int dbmPrepareTableHandle( dbmHandle       * aHandle,
                           const char      * aTableName,
                           dbmTableHandle  ** aTableHandle );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수가 사용되어야 한다.
aTableName	char *	In	Prepare 할 TableName을 입력한다.
aTableHandle	dbmTableHandle **	out	Prepare 된 table의 handle 이다.

사용 예

```
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "table1",
                                &sTableHandle );
}
```

노트

만일 대상 table에 DDL이 발생하면 내부적으로 prepare를 재수행한다. 이 때 table의 변경 정보를 재구축하는 과정에서 메모리 사용량이 일부 증가할 수 있으며 성능의 jitter가 발생할 수 있다.

dbmAuthorize

기능

Instance에 암호가 설정된 경우 접근 허용을 위해 암호를 확인하는 API이다.

인자

```
int dbmAuthorize( dbmHandle     * aHandle,
                  const char    * aPassword )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In/ out	dbmInitHandle에 의해 리턴된 Handle 변수
aPassword	const char *	In	Instance에 설정된 암호

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmAuthorize( sHandle, "lite_pwd" );
}
```

> 노트
> dbmAuthorize API는 dbmInitHandle 이후 호출해야 다른 API 사용이 가능하다.

dbmPrepareStmt

기능

사용자가 수행할 SQL을 실행 직전 단계로 만든다.

- 구문 분석
- Object 정합성 등의 validation 수행
- DDL/ DML을 모두 사용할 수 있다.

인자

```
int dbmPrepareStmt( dbmHandle       * aHandle,
                    char            * aSQLString,
                    dbmStmt        ** aStmt )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수가 사용되어야 한다.
aSQLString	char *	In	Prepare 할 SQL string 이다.
aStmt	dbmStmt **	In/ out	Prepared 된 Stmt가 반환된다. (NULL로 초기화 된 변수여야 한다.)

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select * from t1 where c1 = :v1",
                         & sStmt );
}
```

노트

- SQL 내에서 parameter를 기술하는 방법은 다음 두 가지인데 섞어서 쓸 수는 없으며 한 가지 형태로만 나열해야 한다.
  - Marker (?) 를 이용하여 순서대로 나열하는 방법
  - :v1 과 같이 이름을 명시하는 방법
- dbmStmt는 (void *) 형태 변수로 dbmPrepareStmt는 prepared 된 결과를 dbmStmt 변수에 반환한다.

Procedure는 다음과 같은 방식으로 호출한다. Procedure를 호출할 때는 out mode parameter인 경우에도 BindParamById와 같은 BindParam 함수를 사용해야 한다.

```
rc = dbmPrepareStmt( sHandle, "exec proc1(?, ?, ?, ?)", &sStmt );
    TEST_ERR( sHandle, rc, "prepareStmt" );
    rc = dbmBindParamById( sHandle, sStmt, 1, DBM_BIND_DATA_TYPE_INT, &sData.c1, NULL );
    TEST_ERR( sHandle, rc, "bindParam1" );
    rc = dbmBindParamById( sHandle, sStmt, 2, DBM_BIND_DATA_TYPE_INT, &sData.c2, NULL );
    TEST_ERR( sHandle, rc, "bindParam2" );
    rc = dbmBindParamById( sHandle, sStmt, 3, DBM_BIND_DATA_TYPE_INT, &sData.c3, NULL );
    TEST_ERR( sHandle, rc, "bindParam3" );
    rc = dbmBindParamById( sHandle, sStmt, 4, DBM_BIND_DATA_TYPE_INT, &sData.c4, NULL );
    TEST_ERR( sHandle, rc, "bindParam4" );
    for( i = 0 ; i < 10 ; i ++ )
    {
        sData.c1 = i;
        sData.c2 = i;
        sData.c3 = i;
        rc = dbmExecuteStmt( sHandle, sStmt );
        TEST_ERR( sHandle, rc, "executeStmt" );
        fprintf( stdout, "c4=%d\n", sData.c4 );
    }
```

주의

dbmPrepareStmt에서 생성되는 dbmStmt 객체는 공유할 수 없다. 따라서 서로 다른 handle을 쓸 경우, dbmPrepareStmt를 통해 각 handle별 dbmStmt 객체를 생성해야 한다.

dbmFreeStmt

기능

dbmStmt Handle의 자원을 해제시킨다.

인자

```
int dbmFreeStmt( dbmHandle       * aHandle,
                 dbmStmt        ** aStmt )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수가 사용되어야 한다.
aStmt	dbmStmt **	In/ out	Prepared 된 Stmt Pointer를 입력한다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select * from t1 where c1 = :v1",
                         & sStmt );
    // ....
    rc = dbmFreeStmt( sHandle, &sStmt );
}
```

dbmBindParamById

기능

dbmPrepareStmt에서 사용된 사용자 parameter marker (?)에 대응되는 변수를 binding 한다.

- 사용자 변수는 input mode로만 사용할 수 있다.
- Binding 되는 변수의 pointer를 지정해야 하며 dbmPrepareStmt와 dbmExecuteStmt 사이에 호출한다.
- 길이 정보를 포함하는 변수를 지정하지 않을 경우, CHAR 타입은 실행 시점에 바인딩 된 변수의 string length를 구해 실행한다. 따라서 NULL을 포함하지 않는 변수일 경우 알 수 없는 오류가 발생할 수 있다. 그 외의 타입은 NULL로 지정할 수 있다.

인자

```
int dbmBindParamById( dbmHandle          * aHandle,
                      dbmStmt            * aStmt,
                      int                  aBindId,
                      dbmBindDataType      aBindType,
                      void               * aData,
                      long               * aSizePtr );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Prepared 된 stmt 변수이다.
aBindId	int	-	Prepare 시점에 나열된 parameter의 순서 (base=1) 이다.
aBindType	dbmBindDataType	In	Binding 변수의 데이터 타입이다. - DBM_BIND_DATA_TYPE_SHORT - DBM_BIND_DATA_TYPE_INT - DBM_BIND_DATA_TYPE_DOUBLE - DBM_BIND_DATA_TYPE_FLOAT - DBM_BIND_DATA_TYPE_LONG - DBM_BIND_DATA_TYPE_CHAR - DBM_BIND_DATA_TYPE_DATE - DBM_BIND_DATA_TYPE_TIMESTAMP
aData	void *	In	사용자 변수 포인터이다.
aSizePtr	long *	In	데이터 크기를 저장하는 8 byte 변수 포인터

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            sVar;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select * from t1 where c1 = ?",
                         & sStmt );
    rc = dbmBindParamById( sHandle,
                           sStmt,
                           1,
                           DBM_BIND_DATA_TYPE_INT,
                           & sVar,
                           NULL );
}
```

노트

dbmBindParamById는 사용자 데이터만 연결할 수 있고 테이블이나 column 이름은 binding 할 수 없다.

dbmBindParamByName

기능

dbmPrepareStmt에서 사용자가 기술한 변수 이름을 기반으로 사용자 변수를 binding 한다.

- 사용자 변수는 input mode로만 사용할 수 있다.
- Binding 되는 변수의 pointer를 지정해야 하며 dbmPrepareStmt와 dbmExecuteStmt 사이에 호출한다.

인자

```
int dbmBindParamByName( dbmHandle           * aHandle,
                        dbmStmt             * aStmt,
                        char                * aVarName,
                        dbmBindDataType       aBindType,
                        void                * aData,
                        long                * aSizePtr );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Prepared 된 stmt 변수이다.
aVarName	char *	In	Prepare 시점에 나열된 parameter 이름이다.
aBindType	dbmBindDataType	In	Binding 변수의 데이터 타입이다. - DBM_BIND_DATA_TYPE_SHORT - DBM_BIND_DATA_TYPE_INT - DBM_BIND_DATA_TYPE_DOUBLE - DBM_BIND_DATA_TYPE_FLOAT - DBM_BIND_DATA_TYPE_LONG - DBM_BIND_DATA_TYPE_CHAR - DBM_BIND_DATA_TYPE_DATE - DBM_BIND_DATA_TYPE_TIMESTAMP
aData	void *	In	사용자 변수 포인터이다.
aSizePtr	long *	In	데이터 크기를 저장하는 8 byte 변수 포인터이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
```

```
    int            sVar;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select * from t1 where c1 = :v1",
                         & sStmt );
    rc = dbmBindParamByName( sHandle,
                             sStmt,
                             "v1",
                             DBM_BIND_DATA_TYPE_INT,
                             & sVar,
                             NULL );
}
```

노트

dbmBindParamByName은 사용자 데이터만 연결할 수 있고 테이블이나 column 이름은 binding 할 수 없다.

dbmBindCol

기능

Select 구문에 의해 수행된 결과를 저장하기 위한 사용자 변수와 매핑하는 역할을 수행한다.

- dbmExecuteStmt에 의해 수행된 질의가 select 구문이어야 한다.
- 실제 select target 절의 데이터 크기와 사용자 변수의 크기가 다르므로 결과를 복사할 때 오류가 발생하지 않도록 주의해야 한다.

인자

```
int dbmBindCol( dbmHandle          * aHandle,
                dbmStmt            * aStmt,
                int                  aBindIndex,
                dbmBindDataType      aBindDataType,
                void               * aData,
                int                  aMaxSize,
                long               * aSizePtr )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Prepared 된 stmt 변수이다.
aBindIndex	int	In	Target column이 출현하는 순서이다. (Base=1)
aBindDataType	dbmBindDataType	In	Target 변수의 데이터 타입이다.
aData	void *	In	사용자 변수 포인터이다.
aMaxSize	long	In	데이터 크기의 최대값이다
aSizePtr	long *	In	데이터 크기를 저장하는 8 byte 변수 포인터이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            sVar;
    int            sCol1;
    int            sCol2;
    int            rc;
```

```
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select c1, c2 from t1 where c1 = :v1",
                         & sStmt );
    rc = dbmBindParamByName( sHandle,
                             sStmt,
                             "v1",
                             DBM_BIND_DATA_TYPE_INT,
                             & sVar,
                             NULL );
    rc = dbmBindCol( sHandle,
                     sStmt,
                     1,
                     DBM_BIND_DATA_TYPE_INT,
                     & sCol1,
                     sizeof(int),
                     NULL );

    rc = dbmBindCol( sHandle,
                     sStmt,
                     2,
                     DBM_BIND_DATA_TYPE_INT,
                     & sCol2,
                     sizeof(int),
                     NULL );
}
```

dbmBindColStruct

기능

Select 구문에 의해 수행된 결과를 저장하기 위해 사용자 변수와 매핑한다는 점에서 dbmBindCol과 동일하다. 다만 사용자가 정의한 구조체 변수로 반환받고자 할 때 사용한다는 차이가 있다.

- dbmExecuteStmt에 의해 수행된 질의가 select 구문이어야 한다.
- 구조체를 한 개만 binding 할 수 있고 dbmBindCol과 혼용하여 사용할 수 없다.
- 실제 select target 절의 데이터 크기와 사용자 변수의 크기가 다른 경우 메모리 침범등의 오류가 발생할 수 있다.

인자

```
int dbmBindColStruct( dbmHandle           * aHandle,
                      dbmStmt             * aStmt,
                      void                * aData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Prepared 된 stmt 변수이다.
aData	void *	In	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
    int c3;
} DATA;
int main( int argc, char *argv[] )
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt = NULL;
    DATA           sData;
    char           sErrMsg[1024];
    int c1;
    int rc;
    rc = dbmInitHandle( &sHandle, "demo" );
```

```
    TEST_ERR( sHandle, rc, "initHandle" );
    c1 = 10;
    rc = dbmPrepareStmt( sHandle,
                         "select * from t1 where c1 = ?",
                         & sStmt );
    TEST_ERR( sHandle, rc, "prepareStmt" );
    //중략
    rc = dbmBindColStruct( sHandle,
                           sStmt,
                           &sData );
    TEST_ERR( sHandle, rc, "selectTargetBind" );
    rc = dbmExecuteStmt( sHandle, sStmt );
    if( rc )
    {
        dbmGetErrorData( sHandle, &rc, sErrMsg, sizeof(sErrMsg) );
        printf("ERR-%d] %s\n", rc, sErrMsg );
    }
    rc = dbmFetchStmt( sHandle, sStmt );
    TEST_ERR( sHandle, rc, "fetchStmt" );
    printf( "Out c1=%d, c2=%d, c3=%d\n", sData.c1, sData.c2, sData.c3 );
    rc = dbmFreeStmt( sHandle, &sStmt );
    TEST_ERR( sHandle, rc, "FreeStmtInsert" );
    return 0;
}
```

dbmExecuteStmt

기능

dbmPrepareStmt에 의해 처리된 statement를 실행한다.

인자

```
int dbmExecuteStmt( dbmHandle           * aHandle,
                    dbmStmt             * aStmt )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Prepared 된 stmt 변수이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            sVar;
    int            sCol1;
    int            sCol2;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select c1, c2 from t1 where c1 = :v1",
                         & sStmt );
    rc = dbmBindParamByName( sHandle,
                             sStmt,
                             "v1",
                             DBM_BIND_DATA_TYPE_INT,
                             & sVar,
                             NULL );
    rc = dbmBindCol( sHandle,
                     sStmt,
                     1,
```

```
                    DBM_BIND_DATA_TYPE_INT,
                    & sCol1,
                    sizeof(int),
                    NULL );
    rc = dbmBindCol( sHandle,
                    sStmt,
                    2,
                    DBM_BIND_DATA_TYPE_INT,
                    & sCol2,
                    sizeof(int),
                    NULL );
    rc = dbmExecuteStmt( sHandle,
                         sStmt );
}
```

dbmFetchStmt

기능

dbmExecuteStmt에 의해 수행된 select 문 조회 결과를 한 건씩 가지고 온다.

- Select 문이 아닌데 호출될 경우 오류가 발생한다.
- dbmBindCol에 의해 미리 사용자 변수가 지정되어야 하며 잘못 설정된 경우 오류가 발생한다.

인자

```
int dbmFetchStmt( dbmHandle          * aHandle,
                  dbmStmt            * aStmt )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Executed 된 stmt 변수이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    int            sVar;
    int            sCol1;
    int            sCol2;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select c1, c2 from t1 where c1 = :v1",
                         & sStmt );
    rc = dbmBindParamByName( sHandle,
                             sStmt,
                             "v1",
                             DBM_BIND_DATA_TYPE_INT,
                             & sVar,
                             NULL );
```

```
    rc = dbmBindCol( sHandle,
                     sStmt,
                     1,
                     DBM_BIND_DATA_TYPE_INT,
                     & sCol1,
                     sizeof(int),
                     NULL );
    rc = dbmBindCol( sHandle,
                     sStmt,
                     2,
                     DBM_BIND_DATA_TYPE_INT,
                     & sCol2,
                     sizeof(int),
                     NULL );
    rc = dbmExecuteStmt( sHandle,
                         sStmt );
    while(1)
    {
        rc = dbmFetchStmt( sHandle,
                           sStmt );

        if( rc != 0 )
        {
            break;
        }
    }
}
```

dbmFetchStmt2Json

기능

dbmExecuteStmt가 select 문을 조회한 결과를 한 건씩 JSON format text로 가져온다.

- Select 문이 아닌데 호출될 경우 오류가 발생한다.
- 입력된 버퍼의 크기는 충분히 크게 할당하여 사용해야 한다.

인자

```
int dbmFetchStmt2Json( dbmHandle          * aHandle,
                       dbmStmt            * aStmt,
                       char               * aDataPtr )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	Execute 된 stmt 변수이다.
aDataPtr	char *	In/Out	JSON format 결과가 저장될 사용자 변수 포인터이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt   = NULL;
    char           sText[1024];
    int            sVar;
    int            rc;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareStmt( sHandle,
                         "select c1, c2 from t1 where c1 = :v1",
                         & sStmt );
    rc = dbmBindParamByName( sHandle,
                             sStmt,
                             "v1",
                             DBM_BIND_DATA_TYPE_INT,
                             & sVar,
```

```
                              NULL );
    while( dbmFetchStmt2Json( sHandle, sStmt, sText ) == 0 )
    {
        fprintf( stdout, "%s\n", sText );
    }
}
```

dbmInsertRow

기능

한 개의 사용자 데이터를 지정된 테이블에 추가한다.

인자

```
int dbmInsertRow( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData,
                  int                  aDataSize )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In	사용자 변수 포인터이다.
aDataSize	int	In	데이터 크기이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle,
                       "t1",
                       & sData,
                       sizeof(DATA) );
}
```

dbmInsert

기능

한 개의 사용자 데이터를 table handle에 지정된 테이블에 추가하고 row의 slot ID를 반환한다.

인자

```
int dbmInsert( dbmHandle       * aHandle,
               dbmTableHandle  * aTableHandle,
               void            * aUserData,
               int               aDataSize,
               long            * aSlotId );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블 핸들이다.
aUserData	void *	In	사용자 변수 포인터이다.
aDataSize	int	In	데이터 크기이다.
aSlotId	long *	out	NULL이 아닌 경우 slot ID를 반환한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    long             sSlotId;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                &sTableHandle );
    sData.c1 = 1;
```

```
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle,
                       sTableHandle,
                       & sData,
                       sizeof(DATA),
                       & sSlotId );
}
```

dbmUpdateRow

기능

한 건 이상의 사용자 데이터를 갱신한다.

- Key를 갱신하지 않고 데이터만 갱신할 경우에 빠르게 처리하기 위해 호출한다.
- Before/ after의 update로 인해 key column의 값이 갱신될 경우, dbmPrepareStmt/ dbmExecuteStmt를 통해 수행해야 한다.

인자

```
int dbmUpdateRow( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData,
                  int                * aAffectedCount )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In	사용자 변수 포인터이다.
aAffectedCount	int *	Out	Updated 된 row 개수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    sData.c2 = 100;
```

```
    rc = dbmUpdateRow( sHandle,
                       "t1",
                       & sData,
                       & sRowCount );
}
```

노트

Unique index의 경우 dbmUpdateRow는 한 건만 갱신되지만 non-unique index의 경우 여러 건이 갱신될 수 있는데 이 경우 해당 건수는 반환되는 aAffectedCount를 통해 확인할 수 있다.

dbmUpdate

기능

지정된 table handle의 테이블에서 주어진 데이터에 해당하는 key를 가진 데이터를 변경하며, 필요한 경우 변경 전의 data를 반환한다.

aSlotID 인자가 -1 이 아닌 경우, 해당 slot의 데이터를 변경한다.

인자

```
int dbmUpdate( dbmHandle      * aHandle,
               dbmTableHandle * aTableHandle,
               void           * aUserData,
               long             aSlotId,
               int            * aRowCount,
               void           * aReturnOldData );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	In	사용자 변수 포인터이다.
aSlotId	long	in	-1 이 아닐 경우, 해당 slot의 데이터를 변경한다.
aRowCount	int *	Out	Updated 된 row 개수이다.
aReturnOldData	void *	Out	NULL이 아닐 경우 이전 data를 반환한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    DATA             sOldData;
```

```
    int              sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 1;
    sData.c2 = 100;
    rc = dbmUpdate( sHandle,
                    sTableHandle,
                    & sData,
                    -1,
                    & sRowCount,
                    & sOldData );
}
```

dbmUpsert

기능

지정된 table handle의 테이블에서 주어진 데이터에 해당하는 key를 가진 데이터가 있으면 변경하고 없으면 insert를 수행한다. Update 되었을 때 필요한 경우 변경되기 전의 data를 반환한다.

변경 전의 data가 필요 없으면 aReturnOldData를 NULL로 설정해야 하지만, aInsertCnt는 NULL이 아닌 경우 insert로 동작이 성공하면 1을 반환하고 update로 성공하면 0을 반환한다.

변경 전의 data가 필요하면 aReturnOldData를 NULL이 아닌 값으로 설정해야 하고, 해당 변수값은 update 일 때만 유효하므로 이 변수가 유효한지 여부를 판단하기 위해 aInsertCnt도 NULL이 아닌 값으로 설정해야 한다.

인자

```
int dbmUpsert( dbmHandle        * aHandle,
               dbmTableHandle   * aTableHandle,
               void             * aUserData,
               int                aDataSize,
               void             * aReturnOldData,
               int              * aInsertCnt );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	In	사용자 변수 포인터이다.
aDataSize	int	in	aUserData의 크기
aReturnOldData	void *	Out	NULL이 아닐 경우 이전 data를 반환한다.
aInsertCnt	int *	Out	NULL이 아닐 경우 • Insert로 성공하면 1을 반환한다. • Update로 성공하면 0을 반환한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
```

```
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    DATA             sOldData;
    int              sInsertCnt = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 1;
    sData.c2 = 100;
    rc = dbmUpsert( sHandle,
                    sTableHandle,
                    & sData,
                    sizeof(DATA),
                    & sOldData,
                    & sInsertCnt);
}
```

dbmDeleteRow

기능

한 건 이상의 사용자 데이터를 삭제한다.

인자

```
int dbmDeleteRow( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData,
                  int                * aAffectedCount )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In	사용자 변수 포인터이다.
aAffectedCount	int *	Out	Delete 된 row 개수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    sData.c2 = 100;
    rc = dbmDeleteRow( sHandle,
                       "t1",
                       & sData,
                       & sRowCount );
```

```
}
```

dbmDelete

기능

주어진 table handle의 테이블에서 한 건 이상의 사용자 데이터를 삭제한다.

인자

```
int dbmDelete( dbmHandle      * aHandle,
               dbmTableHandle * aTableHandle,
               void           * aUserData,
               long             aSlotId,
               int            * aRowCount,
               void           * aReturnOldData );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	In	사용자 변수 포인터이다.
aSlotId	long	in	-1 이 아닌 경우 해당 slot의 데이터를 삭제한다.
aRowCount	int *	Out	NULL이 아닌 경우 delete 된 row 개수이다.
aReturnOldData	void *	Out	NULL이 아닌 경우 delete 된 row의 데이터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    DATA             sOldData;
    int              sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
```

```
    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 1;
    sData.c2 = 100;
    rc = dbmDeleteRow( sHandle,
                       sTableHandle,
                       & sData,
                       -1,
                       & sRowCount,
                       & sOldData );
}
```

주의

Splay table type에서 delete 하면 즉시 key를 삭제한다. 따라서 트랜잭션이 진행되는 도중에 삭제된 key는 조회되지 않는다. 또한 삭제된 트랜잭션을 롤백하려고 할 때 이미 다른 세션에 의해 삽입되었을 경우에는 삭제된 트랜잭션이 롤백되지 않고 해당 레코드는 유실된다.

dbmBindColumn

기능

dbmUpdateRowByCols를 호출할 때 특정 column에 사용자 데이터를 저장하기 위해 사용한다.
dbmBindColumn을 수행하는 시점에 내부 임시 버퍼에 사용자 데이터가 복제된다. 따라서 execution하기 전에 반드시 dbmBindColumn을 호출해야 한다.
동일한 column을 대상으로 반복적으로 dbmBindColumn을 호출할 경우 마지막에 호출한 값이 저장되며 존재하지 않는 column에 대해 수행할 경우 오류가 발생한다.

인자

```
int dbmBindColumn( dbmHandle     * aHandle,
                   const char    * aTableName,
                   const char    * aColumnName,
                   void          * aUserData,
                   int             aDataSize );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aColumnName	char *	In	대상 column 이름이다.
aUserData	void *	In	사용자 데이터 포인터이다.
aDataSize	int	In	aUserData의 크기 (byte) 이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    char           sData[1024];
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmBindColumn( sHandle,
                        "t1",
                        "col1",
                        sData,
                        sizeof(sData) );
}
```

dbmClearBind

기능

dbmBindColumn을 수행한 정보를 삭제한다. 특정 column만 지정할 수 있으며 column을 명시하지 않고 NULL로 지정할 경우 이전에 binding 된 모든 정보를 삭제한다.

인자

```
int dbmClearBind( dbmHandle    * aHandle,
                  const char   * aTableName,
                  const char   * aColumnName );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aColumnName	char *	In	대상 column 이름이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    char           sData[1024];
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmClearBind( sHandle,
                       "t1",
                       "col1" );
    rc = dbmClearBind( sHandle,
                       "t1",
                       NULL );
}
```

dbmUpdateRowByCols

기능

특정 column만 갱신하고자 할 경우에 사용한다. 호출하기 전에 미리 dbmBindColumn API를 이용하여 key value를 포함하여 갱신하려는 column에 값을 설정해야 한다.

인자

```
int dbmUpdateRowByCols( dbmHandle      * aHandle,
                        const char     * aTableName,
                        int            * aRowCount );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aRowCount	int *	Out	갱신된 row count를 반환한다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    char           sData[1024];
    int            sRowCount;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmBindColumn( sHandle,
                        "t1",
                        "col1",
                        sData,
                        sizeof(sData) );
    rc = dbmUpdateRowByCols( sHandle,
                             "t1",
                             &sRowCount );
}
```

dbmSelectRow

기능

한 건의 사용자 데이터를 조회하고 fetch 한다.
여러 건이 조회되더라도 첫 번째 한 건만 가지고오는데 다음 건을 가져와야 할 경우, dbmFetchNext 계열의 API를 사용해야 한다.

인자

```
int dbmSelectRow( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    rc = dbmSelectRow( sHandle,
                       "t1",
                       & sData );
}
```

> 노트
> 대상 레코드를 조회하려면 dbmSelectRow, dbmUpdateRow, dbmDeleteRow 모두 실행 시점에 사용자 데이터에 key를 포함한 상태여야 한다. dbmSelectRow는 사용자 데이터를 조회한 후에 사용자 변수에 저장하기까지 한다.

Date 타입을 반환받아야 할 경우에는 사용자가 unsigned long long 형태의 8 byte 변수를 지정하여 값을 저장할 수 있다.

다음 예제를 참고한다.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <time.h>
#include <dbmUserAPI.h>
typedef struct
{
    int c1;
    long long c2;
} DATA;
main()
{
    dbmHandle *sHandle = NULL;
    struct timeval ss;
    time_t now;
    struct tm *nowtm;
    char buf[200];
    DATA sData;
    int sRowCount;
    int rc;
    rc = dbmInitHandle( &sHandle, "demo" );
    if( rc )
    {
        printf( "test fail1\n" );
    }
    sData.c1 = 1;
    rc = dbmSelectRow( sHandle, "t1", &sData );
    if( rc )
```

```
    {
        printf( "test fail2\n" );
    }
```

- Date 값을 string format으로 변환한다.

```
ss.tv_sec = sData.c2 / 1000000.0;
ss.tv_usec = sData.c2 % 1000000;
now = ss.tv_sec;
nowtm = localtime(&now);
strftime( buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", nowtm);
printf( "c1=%d, c2=%s.%ld\n", sData.c1, buf, ss.tv_usec );
```

- 현재 시각으로 변경할 경우

```
gettimeofday( &ss, NULL );
sData.c2 = ss.tv_sec * 1000000LL + ss.tv_usec;
rc = dbmUpdateRow( sHandle, "t1", &sData, &sRowCount );
if( rc )
{
    printf( "test fail3\n" );
}
rc = dbmSelectRow( sHandle, "t1", &sData );
if( rc )
{
    printf( "test fail4\n" );
}
ss.tv_sec = sData.c2 / 1000000.0;
ss.tv_usec = sData.c2 % 1000000;
now = ss.tv_sec;
nowtm = localtime(&now);
strftime( buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", nowtm);
printf( "c1=%d, c2=%s.%ld\n", sData.c1, buf, ss.tv_usec );
dbmCommit( sHandle );
}
================================
== Test
================================
shellPrompt> ./testPgm
c1=1, c2=2020-12-23 17:48:38.913314
c1=1, c2=2020-12-23 17:49:33.214220
```

API에 의해 변경된 시간정보가 table에도 동일하게 반영되어 있다는 것을 확인할 수 있다.

```
dbmMetaManager(DEMO)> select * from t1;
--------------------------------------------------------------------------------
C1                    : 1
C2                    : 2020/12/23 17:49:33.214220
--------------------------------------------------------------------------------
1 row selected
dbmMetaManager(DEMO)>
```

dbmSelect

기능

주어진 table handle의 테이블에서 주어진 조건에 맞는 사용자 데이터를 조회하기 시작하고 fetch 한다.
한 건 이상의 데이터를 조회하려면 계속해서 dbmFetch 함수를 사용해야 한다.

인자

```
int dbmSelect( dbmHandle         * aHandle,
               dbmTableHandle    * aTableHandle,
               void              * aUserData,
               void              * aUntilData,
               dbmScanDirection    aScanDir,
               dbmScanType         aScanType );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	In/ out	시작 조건이 되는 사용자 변수 포인터이다. 검색된 결과 데이터가 저장되는 버퍼 공간이기도 하다.
aUntilData	void *	in	종료 조건이 되는 사용자 변수 포인터 이다. 종료 조건이 없을 경우 NULL을 명시한다
aScanDir	dbmScanDirection	in	• DBM_SCAN_DIR_BACKWARD: 인덱스의 역방향 탐색 • DBM_SCAN_DIR_FORWARD: 인덱스의 정방향 탐색 • DBM_SCAN_DIR_EQUAL: 같은 값을 가지는 key 탐색
aScanType	dbmScanType	in	• DBM_SCAN_TYPE_RDONLY: read-only 로 검색 • DBM_SCAN_TYPE_FOR_UPDATE: FOR UPDATE 모드로 검색

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
```

```
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    int              sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 1;
    rc = dbmSelect( sHandle,
                    sTableHandle,
                    & sData,
                    NULL,
                    DBM_SCAN_DIR_EQUAL,
                    DBM_SCAN_TYPE_RDONLY );
}
```

노트

DBM_SCAN_DIR_FORWARD는 aUserData 보다 큰 다음 (index order 기준) 데이터를 반환하고, UntilData의 key 보다 큰 경우에는 NOT_FOUND를 반환한다.
DBM_SCAN_DIR_BACKWARD는 aUserData 보다 작은 다음 (index order 기준) 데이터를 반환하고, UntilData의 key 보다 작은 경우에는 NOT_FOUND를 반환한다.
DBM_SCAN_DIR_EQUAL은 aUserData와 값이 같은 다음 데이터를 반환한다.

dbmSetIndex

기능

지정한 테이블에 사용할 index를 지정한다.

인자

```
int dbmSetIndex( dbmHandle           * aHandle,
                 const dbmChar       * aTableName,
                 const dbmChar       * aIndexName )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	const char *	In	테이블 이름을 입력한다.
aIndexName	const char *	In	사용할 index 이름을 입력한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    DATA             sUntilData;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    rc = dbmSetIndex( sHandle, "T1", "IDX2_T1" );
}
```

dbmFetch

기능

주어진 table handle의 테이블에서 수행한 이전 검색에 이어서 다음 데이터를 조회한다.

반드시 dbmSelect가 먼저 호출된 이후에 사용해야 한다.

인자

```
int dbmFetch( dbmHandle         * aHandle,
              dbmTableHandle    * aTableHandle,
              void              * aUserData,
              void              * aUntilData,
              dbmScanDirection    aScanDir,
              dbmScanType         aScanType );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	out	검색된 결과 데이터가 저장되는 버퍼 공간이다.
aUntilData	void *	in	종료 조건이 되는 사용자 변수 포인터 이다. 종료 조건이 없을 경우 NULL을 명시한다
aScanDir	dbmScanDirection	in	• DBM_SCAN_DIR_BACKWARD : 인덱스의 역방향 탐색 • DBM_SCAN_DIR_FORWARD : 인덱스의 정방향 탐색 • DBM_SCAN_DIR_EQUAL : 같은 값을 가지는 key 탐색
aScanType	dbmScanType	in	• DBM_SCAN_TYPE_RDONLY : read-only 로 검색 • DBM_SCAN_TYPE_FOR_UPDATE : FOR UPDATE 모드로 검색

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
```

```
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    DATA             sUntilData;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 10;
    sUntilData.c1 = 100;
    rc = dbmSelect( sHandle,
                    sTableHandle,
                    & sData,
                    & sUntilData,
                    DBM_SCAN_DIR_FORWARD,
                    DBM_SCAN_TYPE_RDONLY );
   while( 1 )
   {
       rc = dbmFetch( sHandle,
                      sTableHandle,
                      & sData,
                      & sUntilData,
                      DBM_SCAN_DIR_FORWARD,
                      DBM_SCAN_TYPE_RDONLY );
       if( rc != 0 ) break;
   }
}
```

노트

DBM_SCAN_DIR_FORWARD는 이전에 반환한 데이터보다 큰 다음 (index order 기준) 데이터를 반환하고, UntilData의 key 보다 큰 경우에는 NOT_FOUND를 반환한다.
DBM_SCAN_DIR_BACKWARD는 이전에 반환한 데이터보다 작은 다음 (index order 기준) 데이터를 반환하고, UntilData의 key 보다 작은 경우에는 NOT_FOUND를 반환한다.
DBM_SCAN_DIR_EQUAL은 이전에 반환한 데이터와 값이 같은 다음 데이터를 반환하고, 해당 인덱스가 non-unique 인덱스 속성일 때만 의미가 있다.

dbmSelectMax

기능

주어진 table handle의 direct 테이블에 대해서만 key로 설정된 데이터 중 최대값을 조회한다.

테이블이 가진 segment들 중에 데이터가 하나라도 들어간 마지막 segment를 찾아서 이진 탐색을 통해 최대값을 가진 데이터를 반환한다.

이진 탐색 중에 데이터가 들어 있지 않은 slot을 발견할 경우, 잘못된 결과를 반환할 수 있다.

인자

```
int dbmSelectMax( dbmHandle      * aHandle,
                  dbmTableHandle * aTableHandle,
                  void           * aUserData );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    DATA             sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    sData.c1 = 1;
```

```
    rc = dbmSelectMax( sHandle,
                       sTableHandle,
                       & sData );
}
```

노트

dbmSelectMax API는 index key의 값이 sequential 하게 증가하는 형태로 데이터가 삽입되는 경우에만 지원되며, 이는 오직 direct table에서만 적용된다.

dbmSelectCount

기능

입력된 데이터와 일치하는 레코드의 개수를 반환한다.

인자

```
int dbmSelectRow( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData,
                  int                * aRetCount )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.
aRetCount	int *	out	데이터의 건수를 반환 받을 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    rc = dbmSelectCount( sHandle,
                         "t1",
                         & sData,
                         & sRowCount );
}
```

노트

일치하는 대상이 없을 경우 RowCount는 0으로 반환된다.

dbmSelectRowGT

기능

입력된 데이터의 key 값보다 큰 데이터를 조회한다.

인자

```
int dbmSelectRowGT( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 1;
    rc = dbmSelectRowGT( sHandle,
                         "t1",
                         & sData );
}
```

노트

위의 사용 예에서 dbmSelectRowGT를 수행하여 1보다 큰 데이터가 존재할 경우, 이를 가지고 온다.

dbmSelectRowLT

기능

입력된 데이터의 key 값보다 작은 데이터를 조회한다.

인자

```
int dbmSelectRowLT( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
    rc = dbmSelectRowLT( sHandle,
                         "t1",
                         & sData );
}
```

노트

위의 예에서 dbmSelectRowLT를 수행하여 10보다 작은 데이터가 존재할 경우, 이를 가지고 온다.

dbmFetchNext

기능

입력된 데이터의 key 값과 동일한 다음 데이터를 조회한다.
Non-unique index에 여러 건의 데이터가 존재하는 까닭에 동일한 key 값을 갖는 여러 레코드를 가져오기 위해 dbmSelectRow와 조합하여 fetch를 수행해야 할 경우에 사용한다.

인자

```
int dbmFetchNext( dbmHandle          * aHandle,
                  char               * aTableName,
                  void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
    rc = dbmSelectRow( sHandle,
                       "t1",
                       & sData );
   while( 1 )
   {
       rc = dbmFetchNext( sHandle,
```

```
                        "t1",
                        &sData );
        if( rc != 0 ) break;
    }
}
```

dbmFetchNextGT

기능

입력된 데이터의 key 값보다 큰 데이터를 조회한다. dbmSelectRow 또는 dbmSelectRowGT에서 시작된 값을 기준으로 조회한다.

인자

```
int dbmFetchNextGT( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
    rc = dbmSelectRow( sHandle,
                       "t1",
                       & sData );
   while( 1 )
   {
       rc = dbmFetchNextGT( sHandle,
                            "t1",
```

```
                            &sData );
        if( rc != 0 ) break;
    }
}
```

dbmFetchNextLT

기능

입력된 데이터의 key 값보다 작은 데이터를 조회한다.

인자

```
int dbmFetchNextLT( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
    rc = dbmSelectRow( sHandle,
                       "t1",
                       & sData );
   while( 1 )
   {
       rc = dbmFetchNextLT( sHandle,
                            "t1",
                            &sData );
```

```
        if( rc != 0 ) break;
    }
}
```

dbmSelectForUpdateRow

기능

dbmSelectRow와 동일하며 조회된 레코드를 LOCK 한다.

인자

```
int dbmSelectForUpdateRow( dbmHandle          * aHandle,
                           char               * aTableName,
                           void               * aUserData )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
    rc = dbmSelectForUpdateRow( sHandle,
                                "t1",
                                & sData );
}
```

노트
다른 갱신 DML들과 마찬가지로, dbmSelectForUpdateRow를 수행한 경우 반드시 dbmCommit 이나 dbmRollback을 호출하여 트랜잭션을 종료해야 한다.

레코드 lock을 유지하는 API 종류는 다음과 같다.

- dbmSelectForUpdateRowGT
- dbmSelectForUpdateRowLT
- dbmFetchNextUpdateRowGT
- dbmFetchNextUpdateRowLT

dbmInsertArray

기능

N 개의 레코드를 삽입할 때 사용한다. N 개의 operation 중에 오류가 한 건 이상 발생하면 에러를 반환하는데 오류가 발생한 경우에도 operation은 진행되며 모든 operation이 완료된 후에 결과가 반환된다.

인자

```
int dbmInsertArray( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData,
                    int                  aDataSingleSize,
                    int                  aDataCount,
                    int                  aRetArr[] )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.
aDataSingleSize	int	In	데이터 한 개의 크기를 지정한다.
aDataCount	int	In	aUserData에 담긴 데이터 개수를 지정한다.
aRetArr	int	Out	각 operation의 에러코드를 순서대로 담는다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
    int c3;
} DATA;
int main( int argc, char *argv[] )
{
    dbmHandle    * sHandle = NULL;
    DATA           sData[10];
    int sErrCode[10];
    int i;
    int rc;
```

```
    rc = dbmInitHandle( &sHandle, "demo" );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
        sData[i].c2 = i;
        sData[i].c3 = i;
    }
```

- 정상 처리

```
rc = dbmInsertArray( sHandle, "t1", sData, sizeof(DATA), 10, sErrCode );
    if( rc ) exit(-1);
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
```

- 에러 처리

```
rc = dbmInsertArray( sHandle, "t1", sData, sizeof(DATA), 10, sErrCode );
    for( i = 0 ; i < 10; i ++ )
    {
        printf( "%d] retCode = %d\n", i, sErrCode[i] );
    }
    dbmFreeHandle( &sHandle );
    return 0;
}
```

주의

사용자가 입력한 데이터 한 개의 크기와 개수에 따라 설정해야 하는 사용자 버퍼가 잘못 설정되면 올바르지 않은 포인터가 접근할 수 있으므로 주의해서 사용해야 한다.

dbmUpdateArray

기능

N 개의 레코드를 변경할 때 사용한다. N 개의 operation 중에 오류가 한 건 이상 발생하면 에러를 반환하는데 오류가 발생한 경우에도 operation은 진행되며 모든 operation이 완료된 후에 결과가 반환된다.

인자

```
int dbmUpdateArray( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData,
                    int                  aDataCount,
                    int                * aAffectedRowCount,
                    int                  aRetArr[] )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.
aDataCount	int	In	aUserData에 담긴 개수를 지정한다.
aAffectedRowCount	int	Out	전체 처리된 건수를 반환한다.
aRetArr	int	Out	각 operation의 에러코드를 순서대로 담는다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
    int c3;
} DATA;
int main( int argc, char *argv[] )
{
    dbmHandle    * sHandle = NULL;
    DATA           sData[10];
    int sErrCode[10], sRowCount;
    int i;
    int rc;
```

```
    rc = dbmInitHandle( &sHandle, "demo" );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
        sData[i].c2 = i;
        sData[i].c3 = i;
    }
    rc = dbmInsertArray( sHandle, "t1", sData, sizeof(DATA), 10, sErrCode );
    if( rc ) exit(-1);
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
        sData[i].c2 = i*10;
        sData[i].c3 = i*20;
    }
    rc = dbmUpdateArray( sHandle, "t1", sData, 10, &sRowCount, sErrCode );
    if( rc ) exit(-1);
    printf( "AffectedRow = %d\n", sRowCount );
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
    dbmFreeHandle( &sHandle );
    return 0;
}
```

주의

사용자가 입력한 데이터 한 개의 크기와 개수에 따라 설정해야 하는 사용자 버퍼가 잘못 설정되면 올바르지 않은 포인터가 접근할 수 있으므로 주의해서 사용해야 한다.

dbmSelectArray

기능

N 개의 레코드를 조회할 때 사용한다. N 개의 operation 중에 오류가 한 건 이상 발생하면 에러를 반환하는데 오류가 발생한 경우에도 operation은 진행되며 모든 operation이 완료된 후에 결과가 반환된다.

인자

```
int dbmSelectArray( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData,
                    int                  aDataCount,
                    int                * aAffectedRowCount,
                    int                  aRetArr[] )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.
aDataCount	int	In	aUserData에 담긴 개수를 지정한다.
aAffectedRowCount	int	Out	전체 처리된 건수를 반환한다.
aRetArr	int	Out	각 operation의 에러코드를 순서대로 담는다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
    int c3;
} DATA;
int main( int argc, char *argv[] )
{
    dbmHandle    * sHandle = NULL;
    DATA           sData[10];
    int sErrCode[10], sRowCount;
    int i;
    int rc;
```

```
    rc = dbmInitHandle( &sHandle, "demo" );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
        sData[i].c2 = i;
        sData[i].c3 = i;
    }
    rc = dbmInsertArray( sHandle, "t1", sData, sizeof(DATA), 10, sErrCode );
    if( rc ) exit(-1);
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
    }
    rc = dbmSelectArray( sHandle, "t1", sData, 10, &sRowCount, sErrCode );
    if( rc ) exit(-1);
    printf( "AffectedRow = %d\n", sRowCount );
    dbmFreeHandle( &sHandle );
    return 0;
}
```

주의

사용자가 입력한 데이터 한 개의 크기와 개수에 따라 설정해야 하는 사용자 버퍼가 잘못 설정되면 올바르지 않은 포인터가 접근할 수 있으므로 주의해서 사용해야 한다.

dbmDeleteArray

기능

N 개의 레코드를 삭제할 때 사용한다. N 개의 operation 중에 오류가 한 건 이상 발생하면 에러를 반환하는데 오류가 발생한 경우에도 operation은 진행되며 모든 operation이 완료된 후에 결과가 반환된다.

인자

```
int dbmDeleteArray( dbmHandle          * aHandle,
                    char               * aTableName,
                    void               * aUserData,
                    int                  aDataCount,
                    int                * aAffectedRowCount,
                    int                  aRetArr[] )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 테이블 이름이다.
aUserData	void *	In/ out	사용자 변수 포인터이다.
aDataCount	int	In	aUserData에 담긴 개수를 지정한다.
aAffectedRowCount	int	Out	전체 처리된 건수를 반환한다.
aRetArr	int	Out	각 operation의 에러코드를 순서대로 담는다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
    int c3;
} DATA;
int main( int argc, char *argv[] )
{
    dbmHandle    * sHandle = NULL;
    DATA           sData[10];
    int sErrCode[10], sRowCount;
    int i;
    int rc;
```

```
    rc = dbmInitHandle( &sHandle, "demo" );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
        sData[i].c2 = i;
        sData[i].c3 = i;
    }
    rc = dbmInsertArray( sHandle, "t1", sData, sizeof(DATA), 10, sErrCode );
    if( rc ) exit(-1);
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
    for( i = 0; i < 10; i ++ )
    {
        sData[i].c1 = i;
    }
    rc = dbmDeleteArray( sHandle, "t1", sData, 10, &sRowCount, sErrCode );
    if( rc ) exit(-1);
    printf( "AffectedRow = %d\n", sRowCount );
    rc = dbmCommit( sHandle );
    if( rc ) exit(-1);
    dbmFreeHandle( &sHandle );
    return 0;
}
```

주의

사용자가 입력한 데이터 한 개의 크기와 개수에 따라 설정해야 하는 사용자 버퍼가 잘못 설정되면 올바르지 않은 포인터가 접근할 수 있으므로 주의해서 사용해야 한다.

dbmEnqueue

기능

사용자 데이터를 queue 형식의 테이블에 삽입한다.

- 한 개의 테이블에 여러 형태의 메시지를 이용하려면 msg-type을 임의의 숫자로 지정한다.
- Priority가 더 낮을 수록 우선순위가 높다.
- Msg-type 이나 priority 등이 설정되지 않은 경우에는 모두 0으로 설정된다.

인자

```
int dbmEnqueue( dbmHandle          * aHandle,
                char               * aTableName,
                int                  aPriority,
                char               * aUserData,
                int                  aDataSize )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 queue 테이블 이름이다.
aPriority	int	In	0 이상의 사용자 정의 priority 이다.
aUserData	void *	In	사용자 변수 포인터이다.
aDataSize	int	In	aUserData의 크기이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    rc = dbmInitHandle( &sHandle, "demo" );

    sData.c1 = 10;
```

```
    sData.c2 = 100;
    rc = dbmEnqueue( sHandle,
                     "que1",
                     1,
                     & sData,
                     sizeof(DATA) );

}
```

dbmDequeue

기능

Queue table로부터 한 건의 사용자 데이터를 추출한다.
"Priority" 를 입력하면 해당 값 보다 큰 데이터를 반환한다. Priority 순서로 반환하려면 (-1)로 입력한다.
Dequeue 된 데이터는 자동으로 삭제되며 rollback이 호출된 경우 원본 데이터 형태로 그대로 삽입된다. (Priority는 유지되고 삽입 시간만 rollback time으로 변경된다.)

인자

```
int dbmDequeue( dbmHandle          * aHandle,
                char               * aTableName,
                int                  aInPriority,
                int                * aOutPriority,
                char               * aUserData,
                int                * aDataSize,
                int                  aTimeout )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 queue 테이블 이름이다.
aInPriority	int	In	Dequeue를 수행할 priority를 특정할 경우 해당 값을 입력하고 그렇지 않을 경우 -1을 입력한다. (입력된 priority와 같거나 큰 대상을 반환한다.)
aOutPriority	int *	Out	해당 메시지의 우선순위 정보를 반환한다. (NULL일 경우에는 반환하지 않는다.)
aUserData	void *	Out	사용자 변수 포인터이다.
aDataSize	int *	Out	aUserData의 크기이다.
aTimeout	int	In	Queue에 데이터가 없을 경우 대기하는 시간을 지정 (us단위) 한다.

aTimeout의 설정값에 따라 아래 표와 같이 동작한다.

aTimeout 설정값	동작 방식
-1	Queue에 데이터가 없을 경우, NOT_FOUND 에러를 반환한다.
0	Queue에 데이터가 없을 경우, 무한 대기한다.
0 보다 큰 값	aTimeout 시간 동안 queue에 데이터가 없을 경우, TIMEOUT 에러를 반환한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sDataSize = 0;
    int            sPriority;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmDequeue( sHandle,
                     "que1",
                     -1,
                     &sPriority,
                     & sData,
                     & sDataSize,
                     1000 );

}
```

노트

- Queue에 저장된 데이터는 (Priority, MsgID) 순으로 조합되어 정렬된다.
- Priority가 지정되지 않으면 MsgID 순서대로 출력된다.
- Priority가 동일한 경우, MsgID 순서대로 출력된다.
- Priority 값이 입력되면, 해당 값 이상인 항목만 출력된다.

dbmGetCurrVal

기능

미리 생성되어 있는 sequence 객체의 현재값을 반환한다.

인자

```
int dbmGetCurrVal( dbmHandle          * aHandle,
                   char               * aSequenceName,
                   long long          * aCurrVal )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 queue 테이블 이름이다.
aCurrVal	long long int	Out	반환받을 포인터 (8 byte 변수 필요) 이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    long long      sCurrVal;
    int            sDataSize = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmGetCurrVal( sHandle,
                        "seq1",
                        & sCurrVal );
}
```

주의

Sequence 객체에 대해 nextval이 호출되지 않은 상태에서 currval을 호출할 경우 오류가 발생한다.

dbmGetNextVal

기능

미리 생성되어 있는 sequence 객체의 다음 값을 반환한다.

인자

```
int dbmGetNextVal( dbmHandle          * aHandle,
                   char               * aSequenceName,
                   long long          * aNextVal )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	char *	In	대상 queue 테이블 이름이다.
aNextVal	long long int	Out	반환받을 포인터 (8 byte 변수 필요) 이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    long long      sNextVal;
    int            sDataSize = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmGetNextVal( sHandle,
                        "seq1",
                        & sNextVal);
}
```

dbmCommit

기능

사용자가 수행한 모든 트랜잭션을 영구적으로 반영한다.

인자

```
int dbmCommit( dbmHandle           * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    sData.c1 = 10;
    sData.c2 = 200;
    rc = dbmUpdateRow( sHandle, "t1", &sData, &sRowCount );
    rc = dbmCommit( sHandle );
}
```

dbmRollback

기능

사용자가 수행한 트랜잭션을 모두 rollback 한다.

인자

```
int dbmRollback( dbmHandle           * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    sData.c1 = 10;
    sData.c2 = 200;
    rc = dbmUpdateRow( sHandle, "t1", &sData, &sRowCount );
    rc = dbmRollback( sHandle );
}
```

dbmDeferCommit

기능

사용자가 수행한 모든 트랜잭션을 메모리에만 반영한다. 이 상태에서는 disk logging mode와 관계없이 반드시 dbmDeferSync를 호출하여야만 트랜잭션이 정리된다.

인자

```
int dbmDeferCommit( dbmHandle           * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    sData.c1 = 10;
    sData.c2 = 200;
    rc = dbmUpdateRow( sHandle, "t1", &sData, &sRowCount );
    rc = dbmDeferCommit( sHandle );
}
```

주의

dbmDeferCommit 후에는 해당 세션에서 수행한 내역이 unlock 상태가 된다. 이 상태에서는 추가적으로 DML을 변경할 수 없으며 반드시 dbmDeferSync를 호출해야 한다. 또한 이 상태에서 비정상적으로 종료될 경우 해당 트랜잭션의 WAL 로그는 복구할 수 없다.

dbmDeferSync

기능

dbmDeferCommit에 의해 호출된 트랜잭션 내역은 디스크에 로깅된다.

인자

```
int dbmDeferSync( dbmHandle          * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sRowCount = 0;
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    sData.c1 = 10;
    sData.c2 = 200;
```

```
    rc = dbmUpdateRow( sHandle, "t1", &sData, &sRowCount );
    rc = dbmDeferCommit( sHandle );
    rc = dbmDeferSync( sHandle );
}
```

dbmRefineSystem

기능

사용자로 하여금 instance-level, table-level에서 table/index lock을 해제하거나 index를 재구축하는 복구 기능을 실행하도록 한다.

인자

```
int dbmRefineSystem( const char    * aInstName,
                     const char    * aTableName )
```

인자 항목	타입	In/ out	비고
aInstName	const char *	In	instance 이름을 입력한다.
aTableName	const char *	In	특정 table name을 입력한다.

- Instance 이름은 필수이다.
- TableName을 지정하면 대상 테이블만 복구하고, NULL을 입력하면 instance와 관련된 모든 테이블을 복구한다.

사용 예

```
if( argc > 1 )
    {
        if( dbmRefineSystem( "demo", NULL ) != 0 )
        {
            printf( "failed to refine all\n" );
            exit(-1);
        }
    else
    {
        if( dbmRefineSystem( "demo", "t1" ) != 0 )
        {
            printf( "failed to refine t1\n" );
            exit(-1);
        }
    }
```

노트

Instance-level에서 수행할 때 대상 테이블이 아니면 skip 한다.

dbmGetRowCount

기능

UPDATE/ DELETE/ SELECT 기능을 prepare/ execute 한 후에 대상 row의 개수를 계산한다.

인자

```
int dbmGetRowCount( dbmHandle          * aHandle,
                    dbmStmt            * aStmt,
                    int                * aRowCount )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aStmt	dbmStmt *	In	dbmExecuteStmt으로 실행된 dbmStmt 변수이다.
aRowCount	int *	Out	반환받을 변수 포인터 (4 byte 크기의 변수) 이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmStmt      * sStmt = NULL;
    int            sRowCount;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmPrepareStmt( sHandle, "...", &sStmt );
    rc = dbmExecuteStmt( sHandle, sStmt );

    rc = dbmGetRowCount( sHandle, sStmt, &sRowCount );
}
```

dbmGetRowSize

기능

입력한 테이블이 생성된 시점의 row size를 반환한다.

인자

```
int dbmGetRowSize( dbmHandle          * aHandle,
                   char               * aObjectName,
                   int                * aRowSize )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aObjectName	char *	In	조회할 object name 이다.
aRowSize	int *	Out	반환받을 변수 포인터 (4 byte 크기의 변수) 이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            sRowSize;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmGetRowSize( sHandle, "Table1", &sRowSize );
}
```

dbmGetTableName

기능

입력된 테이블 핸들로부터 table name을 반환한다.

인자

```
const char * dbmGetTableName( dbmTableHandle          * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmTableHandle *	In	dbmPrepareTableHandle 로 처리된 변수이다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    char  * sTableName;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmPrepareTableHandle( &sHandle, "t1", &sTableHandle );
    sTableName = dbmGetTableHandle ( sTableHandle );
}
```

노트

TableHandle이 성공한 상태이어야 한다.

dbmGetTableType

기능

입력된 테이블 핸들로부터 table type을 반환한다.

인자

```
dbmTableType  dbmGetTableType( dbmTableHandle          * aHandle )
```

인자 항목	타입	In/ out	비고
aHandle	dbmTableHandle *	In	dbmPrepareTableHandle 로 처리된 변수이다.

dbmTableType의 정의는 아래와 같다.

```
typedef enum
{
    DBM_TABLE_TYPE_INVALID = 0,
    DBM_TABLE_TYPE_TABLE,                 // BTree Index Table
    DBM_TABLE_TYPE_QUEUE,                 // N/A
    DBM_TABLE_TYPE_SEQUENCE,              // Sequence Object
    DBM_TABLE_TYPE_DIRECT_TABLE,          // Direct Table
    DBM_TABLE_TYPE_DIRECT_QUEUE,          // N/A
    DBM_TABLE_TYPE_SPLAY_TABLE,           // Splay Index Table
    DBM_TABLE_TYPE_LIST_TABLE,            // N/A
    DBM_TABLE_TYPE_HASH_TABLE,            // Hash Table
    DBM_TABLE_TYPE_PERF_VIEW,
    DBM_TABLE_TYPE_MAX
} dbmTableType;
```

노트

본 매뉴얼의 작성 시점인 3.2 버전 기준으로 사용자가 생성 가능한 유형은 다음과 같다.

- DBM_TABLE_TYPE_TABLE
- DBM_TABLE_TYPE_DIRECT_TABLE
- DBM_TABLE_TYPE_SPLAY_TABLE
- DBM_TABLE_TYPE_HASH_TABLE

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    dbmTableType  sTableType;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmPrepareTableHandle( &sHandle, "t1", &sTableHandle );
    sTableType = dbmGetTableType ( sTableHandle );
}
```

dbmSetSplayMode4DML

기능

Splay table handle인 경우, 입력된 option에 따라 insert 할 때 tree를 splay 할 지 여부를 설정한다.
0 으로 설정하면 splay 하지 않는다.

인자

```
int dbmSetSplayMode4DML( dbmTableHandle          * aHandle,
                         int                       aMode )
```

인자 항목	타입	In/ out	비고
aHandle	dbmTableHandle *	In	dbmPrepareTableHandle 로 처리된 변수이다.
aMode	int	int	0 : no splay (default) 1 : do splay

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmPrepareTableHandle( &sHandle, "t1", &sTableHandle );
    dbmSetSplayMode4DML( sTableHandle, 1 );
}
```

노트

입력된 TableHandle은 SPLAY_TABLE_TYPE일 때만 동작한다.

dbmGetErrorData

기능

API 호출로 발생한 각 오류 코드에 대한 상세 에러 메시지를 확인한다.
DBM 내에는 에러가 stack 형태로 축적되어 있기 때문에 에러가 날 때까지 호출하면 최초 오류부터 발생 원인을 추적할 수 있다.

인자

```
int dbmGetErrorData( dbmHandle          * aHandle,
                     int                * aErrorCode,
                     char               * aErrorMsg )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aErrorCode	int *	Out	에러 코드가 담길 변수 포인터이다.
aErrorMsg	char *	Out	에러 메시지가 저장될 변수 포인터이다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sErrCode;
    char           sErrMsg[1024];
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    if( rc )
    {
        while( dbmGetErrorData( sHandle, &sErrCode, sErrMsg ) == 0 )
```

```
        {
             fprintf( stdout, "ERR-%d] %s\n", sErrCode, sErrMsg );
        }
    }
}
```

노트

dbmGetErrorData를 통해 반환받는 에러 메시지 버퍼의 크기는 최소 512 byte 이상이어야 한다.

dbmGetErrorMsg

기능

API 호출로 발생한 각 오류 코드에 대한 상세 에러 메시지를 확인한다.
에러 stack에 저장된 에러의 원본 유형을 출력하기 때문에 상세한 부가 오류사항은 출력되지 않는다.

인자

```
void dbmGetErrorMsg( int         aErrorCode,
                     char      * aErrorMsg,
                     int         aErrorMsgSize )
```

인자 항목	타입	In/ out	비고
aErrorCode	int	In	조회할 에러 코드이다.
aErrorMsg	char *	Out	에러 코드가 담길 변수 포인터이다.
aErrorMsgSize	int	In	에러 메시지가 저장될 변수의 크기를 지정한다.

사용 예

```
typedef struct
{
    int c1;
    int c2;
} DATA;
main()
{
    dbmHandle    * sHandle = NULL;
    DATA           sData;
    int            sErrCode;
    char           sErrMsg[1024];
    rc = dbmInitHandle( &sHandle, "demo" );
    sData.c1 = 10;
    sData.c2 = 100;
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(DATA) );
    if( rc )
    {
        dbmGetErrorMsg( rc, sErrMsg, sizeof(sErrMsg) );
    }
```

```
}
```

dbmGetTableUsage

기능

지정한 테이블 segment 의 사용량 정보를 반환한다.

인자

```
int dbmGetTableUsage( dbmHandle       * aHandle,
                      const char      * aTableName,
                      long            * aMaxSize,
                      long            * aTotalSize,
                      long            * aUsedSize,
                      long            * aFreeSize )
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	const char *	In	대상 테이블 이름이다.
aMaxSize	long *	Out	테이블을 생성할 때 지정한 최대 row 개수이다.
aTotalSize	long *	Out	테이블의 확장된 상태를 포함하여 현재 저장 가능한 최대 row 개수이다.
aUsedSize	long *	Out	테이블에 현재 사용 중인 row 개수이다. (Commit 되지 않은 row를 포함할 수 있음)
aFreeSize	long *	Out	테이블 가용 공간의 개수이다. (Commit 되지 않은 row를 포함할 수 있음)

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    long           sTotal;
    long           sMax;
    long           sUsed;
    long           sFree;
    int            rc;
    rc = dbmInitHandle( &sHandle,
                        "demo" );
    rc = dbmGetTableUsage( sHandle,
                           "t1",
                           &sMax,
```

```
                        &sTotal,
                        &sUsed,
                        &sFree );
}
```

dbmGetTableUsageByHandle

기능

입력된 table handle에 해당하는 테이블의 segment 사용량 정보를 반환한다.

Direct 테이블의 경우, 각 segment 별로 데이터를 가진 최대 slot ID를 구하여 계산되므로 중간에 빈 slot이 있을 경우에도 사용 중인 slot으로 계산될 수도 있다.

인자

```
int dbmGetTableUsageByHandle( dbmHandle       * aHandle,
                              dbmTableHandle  * aTableHandle,
                              long            * aMaxSize,
                              long            * aTotalSize,
                              long            * aUsedSize,
                              long            * aFreeSize );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.
aMaxSize	long *	Out	테이블을 생성할 때 지정한 최대 row 개수이다.
aTotalSize	long *	Out	테이블의 확장된 상태를 포함하여 현재 저장 가능한 최대 row 개수이다.
aUsedSize	long *	Out	테이블에 현재 사용 중인 row 개수이다. (Commit 되지 않은 row를 포함할 수 있음)
aFreeSize	long *	Out	테이블 가용 공간의 개수이다. (Commit 되지 않은 row를 포함할 수 있음)

사용 예

```
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    long             sTotal;
    long             sMax;
    long             sUsed;
    long             sFree;
    int              rc;
```

```
    rc = dbmInitHandle( &sHandle,
                        "demo" );
    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    rc = dbmGetTableUsage( sHandle,
                           sTableHandle,
                           &sMax,
                           &sTotal,
                           &sUsed,
                           &sFree );
}
```

노트

해당 값은 스냅샷이므로 테이블에 삽입/ 삭제가 발생하는 도중에는 그 값이 정확하지 않다.
DIRECT TABLE은 마지막으로 입력된 최종 위치만 기록하기 때문에 중간에 삭제될 경우에는 값이 정확하지 않을 수 있다.

dbmExtendTable

기능

주어진 table handle에 해당하는 테이블에 새 segment를 추가한다.
새 segment의 크기는 CREATE TABLE 시 지정된 extend size 이다.

인자

```
int dbmExtendTable( dbmHandle       * aHandle,
                    dbmTableHandle  * aTableHandle );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableHandle	dbmTableHandle *	In	대상 테이블의 핸들이다.

사용 예

```
main()
{
    dbmHandle      * sHandle = NULL;
    dbmTableHandle * sTableHandle = NULL;
    rc = dbmInitHandle( &sHandle,
                        "demo" );
    rc = dbmPrepareTableHandle( sHandle,
                                "t1",
                                & sTableHandle );
    rc = dbmExtendTable( sHandle,
                         sTableHandle );
}
```

노트

- Extend에 의해 확장 가능한 segment의 최대 개수는 999 개이다.
- Extend가 빈번하게 발생할 경우 삽입 성능이 저하될 수 있으므로 table 생성 시점에 init 크기를 적절하게 설정해야 한다.

dbmExistDataInQue

기능

지정된 queue table에 데이터가 존재하는지 여부를 반환한다.

인자

```
int dbmExistDataInQue( dbmHandle       * aHandle,
                       const char      * aTableName,
                       int             * aExists );
```

인자 항목	타입	In/ out	비고
aHandle	dbmHandle *	In	dbmInitHandle로 처리된 변수이다.
aTableName	const char *	In	대상 queue table 이름이다.
aExists	int *	Out	• 0: 존재하지 않는다. • 1: 한 개 이상의 데이터가 존재한다.

사용 예

```
main()
{
    dbmHandle    * sHandle = NULL;
    int            i;
    rc = dbmInitHandle( &sHandle, "demo" );
    rc = dbmExistDataInQue( sHandle, "que1", &i );
}
```

노트

일반 테이블에 대해서는 사용할 수 없고 queue type 테이블만 지원한다.

dbmNow

기능

현재 시각을 반환한다.

인자

```
unsigned long dbmNow( void )
```

사용 예

```
#include <dbmUserAPI.h>
#include <common.h>
typedef struct
{
    int c1;
    unsigned long c2;
} DATA;
int main()
{
    DATA   sData;

    sData.c1 = 1;
    sData.c2 = dbmNow();
    rc = dbmInsertRow( sHandle, "t1", &sData, sizeof(sData) );
    TEST_ERR( sHandle, rc, "InsertFail" );
    sData.c1 = 1;
    rc = dbmSelectRow( sHandle, "t1", &sData );
    TEST_ERR( sHandle, rc, "SelectFail" );
    tt =sTimeVal.tv_sec;
    nowtm = localtime( &tt );
    strftime( buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", nowtm);
    printf( "fetchData: c1=%d, c2=%s.%06ld\n", sData.c1, buf, sTimeVal.tv_usec );
}
```

2.3 Error Message

GOLDILOCKS LITE에 정의된 에러 메시지는 다음 표와 같다.

Error defined name	Error code	Detail message	Description
DBM_ERRCODE_INVALID_ARGS	22001	fail to validate some parameters at internal processing	-
DBM_ERRCODE_MEMORY_NOT_SUFFICIENT	22002	fail to alloc memory from OS (errno=%d)	-
DBM_ERRCODE_FAIL_TO_ALLOC_MEMORY	22003	fail to alloc memory from dbmAllocator	-
DBM_ERRCODE_NOT_IMPL	22004	not implemented	-
DBM_ERRCODE_ALREADY_SHM_EXIST	22005	a shared memory already exists	-
DBM_ERRCODE_CREATE_SHM_FAIL	22006	fail to create a shared memory segment	-
DBM_ERRCODE_INIT_SHM_FAIL	22007	fail to initialize a shared memory segment	-
DBM_ERRCODE_ATTACH_SHM_FAIL	22008	fail to attach a shared memory segment	-
DBM_ERRCODE_SHM_OPEN_FAIL	22009	fail to open a shared memory	-
DBM_ERRCODE_SHM_FSTAT_FAIL	22010	fail to get a information of shm	-
DBM_ERRCODE_SHM_INVALID_SIZE	22011	invalid segment size to attach a shm	-
DBM_ERRCODE_MMAP_FAIL	22012	fail to call a mmap to attach a shared memory segment	-
DBM_ERRCODE_DETACH_SHM_FAIL	22013	fail to detach a shared memory segment	-
DBM_ERRCODE_DROP_FAIL	22014	fail to drop a shared segment memory	-
DBM_ERRCODE_CREATE_SHM_DIR_FAIL	22015	fail to create a directory for shared-memory	-
DBM_ERRCODE_INVALID_SLOT_NO	22016	invalid slot number (SlotId=%ld)	-
DBM_ERRCODE_NO_EXIST_DIC	22017	fail to attach dictionary (execute initdb)	-
DBM_ERRCODE_NOT_DEF_INSTANCE	22018	a operation not allowed without instance	-
DBM_ERRCODE_NOT_EXIST_TABLE	22019	(%s) table not exists	-
DBM_ERRCODE_NOT_EXIST_COLUMN	22020	(%s) Column not exists	-

Error defined name	Error code	Detail message	Description
DBM_ERRCODE_MAX_SEGMENT	22021	a segment has no space to extend because of reached max_segment	-
DBM_ERRCODE_NO_SPACE	22022	a segment has no space to extend because of reached max_size	-
DBM_ERRCODE_CONNECT_FAIL	22023	fail to connect target server	-
DBM_ERRCODE_SEND_FAIL	22024	fail to send a packet	-
DBM_ERRCODE_RECV_FAIL	22025	fail to receive a packet	-
DBM_ERRCODE_HB_FAIL	22026	fail to send or receive a packet for HB	-
DBM_ERRCODE_INIT_HANDLE_FAIL	22027	fail to initialize a handle	-
DBM_ERRCODE_ALLOC_HANDLE_FAIL	22028	fail to alloc a memory for handle	-
DBM_ERRCODE_NEED_VALUE_NULL	22029	a pointer have to be set null to initialize a handle	-
DBM_ERRCODE_FREE_HANDLE_FAIL	22030	fail to free a handle	-
DBM_ERRCODE_ALLOC_STMT_FAIL	22031	fail to alloc a memory for statement	-
DBM_ERRCODE_INIT_PARSE_CTX_FAIL	22032	fail to alloc a memory for parser-context	-
DBM_ERRCODE_EXECUTE_FAIL	22033	fail to execute a statement	-
DBM_ERRCODE_INVALID_STMT_TYPE	22034	invalid stmt type	-
DBM_ERRCODE_INVALID_PLAN_TYPE	22035	invalid plan type	-
DBM_ERRCODE_INVALID_DATA_TYPE	22036	invalid data type	-
DBM_ERRCODE_INVALID_TABLE_SIZE_OPTION	22037	invalid table size option	-
DBM_ERRCODE_PREPARE_FAIL	22038	"fail to prepare a statement	-
DBM_ERRCODE_FREE_STMT_FAIL	22039	fail to finalize a stmt	-
DBM_ERRCODE_INVALID_EXPR_TYPE	22040	invalid expr type	-
DBM_ERRCODE_ALLOC_MEM_FAIL	22041	fail to alloc a memory for something	-
DBM_ERRCODE_INVALID_BUILT_FUNC	22042	invalid built-in function	-
DBM_ERRCODE_INVALID_SEGMENT	22043	invalid segment	-
DBM_ERRCODE_ALLOC_TRANS	22044	fail to alloc a trans for current-session	-

Error defined name	Error code	Detail message	Description
_FAIL			
DBM_ERRCODE_DATA_COUNT_MISMATCH	22045	the number of binding-data mismatch to target-list	-
DBM_ERRCODE_INVALID_COLUMN	22046	(%s) column not exists	-
DBM_ERRCODE_INVALID_EXPR	22047	invalid expression type	-
DBM_ERRCODE_CONVERT_DATA_FAIL	22048	fail to convert a data as invalid data-type or value-size or origin-value etc.	-
DBM_ERRCODE_BINDING_COL_FAIL	22049	fail to bind a column (%s)	-
DBM_ERRCODE_CONVERT_OVERFLOW	22050	fail to convert data as overflow	-
DBM_ERRCODE_NO_MORE_DATA	22051	no more data to fetch	-
DBM_ERRCODE_DIVIDE_BY_ZERO	22052	a operation can not be executed because of divide by zero	-
DBM_ERRCODE_INVALID_GROUP_BY	22053	invalid group-by or target-list to execute group-by	-
DBM_ERRCODE_NOT_EXIST_INDEX	22054	index not exist (%s)	-
DBM_ERRCODE_INDEX_DUPLICATED	22055	index key value duplicated (%s)	-
DBM_ERRCODE_INDEX_KEY_NOT_FOUND	22056	index key not found (%s)	-
DBM_ERRCODE_INVALID_LOG_TYPE	22057	invalid log type	-
DBM_ERRCODE_DUP_COLUMN_NAME	22058	(%s) column duplicated	-
DBM_ERRCODE_INVALID_DATA_SIZE	22059	invalid data size (Limit=%d : InputSize=%d)	-
DBM_ERRCODE_CHANGE_SCN_FAIL	22060	fail to change SCN of row (Segment=%s, SlotId=%ld)	-
DBM_ERRCODE_INVALID_SCN	22061	invalid scn (SCN=%ld)	-
DBM_ERRCODE_COMMIT_PROC_FAIL	22062	fail to process a function to commit (log=%s)	-
DBM_ERRCODE_ROLLBACK_PROC_FAIL	22063	"fail to process a function to rollback (log=%s)	-
DBM_ERRCODE_DUP_INDEX_KEY_COLUMN	22064	a index with same ordering key was already created (%s)	-
DBM_ERRCODE_DUP_COLUMN_DEFINED	22065	a column definition duplicated (%s)	-
DBM_ERRCODE_OPEN_DISK_LO			

Error defined name	Error code	Detail message	Description
G_FAIL	22066	fail to open a disk logfile (%s) (errno=%d)	-
DBM_ERRCODE_LSEEK_DISK_LOG_FAIL	22067	fail to locate a position of disk logfile (%s) (errno=%d)	-
DBM_ERRCODE_SWITCH_DISK_LOG_FAIL	22068	fail to switch a disk logfile	-
DBM_ERRCODE_WRITE_DISK_LOG_FAIL	22069	fail to write a disk logfile (errno=%d)	-
DBM_ERRCODE_FSYNC_DISK_LOG_FAIL	22070	fail to sync a disk logfile (errno=%d)	-
DBM_ERRCODE_READ_DISK_LOG_FAIL	22071	fail to read from a disk logfile (errno=%d)	-
DBM_ERRCODE_INVALID_DISK_LOG	22072	invalid disk log block	-
DBM_ERRCODE_INVALID_TABLE_TYPE	22073	a operation can not be executed on target-table (check table type)	-
DBM_ERRCODE_INVALID_INDEX_STAT	22075	a index (%s) invalid stat, need to rebuild index	-
DBM_ERRCODE_INVALID_TRY	22076	not supported transaction	-
DBM_ERRCODE_INST_ALREADY_EXISTS	22077	a instance already exists	-
DBM_ERRCODE_INDEX_ALREADY_EXISTS	22078	a index already exists	-
DBM_ERRCODE_ALREADY_EXISTS_TABLE	22079	a table already exists	-
DBM_ERRCODE_DEAD_LOCK_DETECT	22080	a dead-lock detection	-
DBM_ERRCODE_TOO_LONG_NAME	22081	a length of object too long (max %d bytes)	-
DBM_ERRCODE_INVALID_BIND_PARAM	22082	invalid binding parameters (index or name not exist)	-
DBM_ERRCODE_MISMATCH_BIND_COL	22083	invalid binding column count	-
DBM_ERRCODE_NEED_DICT_HANDLE	22084	this operation can be executed by a dictionary handle.	-
DBM_ERRCODE_NOT_EXISTS_INST	22085	a instance not exists	-
DBM_ERRCODE_INVALID_KEY_DATA_TYPE	22086	a index key column must have a data type as (long, char, int, short)	-
DBM_ERRCODE_TIMEOUT	22087	a timeout raised on this operation	-
DBM_ERRCODE_NOT_ALLOWED_OPERATION	22088	this operation not allowed at current-instance	-
DBM_ERRCODE_TOO_BIG_ROW			

Error defined name	Error code	Detail message	Description
SIZE	22089	a total size of columns is too big to create	-
DBM_ERRCODE_NEED_COMMIT_OR_ROLLBACK	22090	fail to free a statement variable as transaction not completed	-
DBM_ERRCODE_TOO_BIG_TO_WRITE_LOG	22091	a log-size is too big to write a transaction log	-
DBM_ERRCODE_FAIL_TO_PARSE	22092	fail to parse a syntax	-
DBM_ERRCODE_NEED_INDEX	22093	a operation via API need a index	-
DBM_ERRCODE_INVALID_SEQ_OPTION	22094	a invalid number or range for sequence	-
DBM_ERRCODE_SEQUENCE_MAXVALUE	22095	a sequence reached at max-value	-
DBM_ERRCODE_SEQUENCE_NOT_DEF_CURRVAL	22096	a currval of sequence not yet defined (need to call nextval)	-
DBM_ERRCODE_NOT_ENOUGH_BUFF	22097	not enough buffer size	-
DBM_ERRCODE_INVALID_LICENSE	22098	invalid license	-
DBM_ERRCODE_INVALID_OFFSET	22099	invalid offset	-
DBM_ERRCODE_TOO_MANY_ROWS	22100	too many rows	-
DBM_ERRCODE_CHECK_DIC_FAIL	22101	fail to check dictionary"	-
DBM_ERRCODE_THREAD_FAIL	22102	fail to invoke a thread	-
DBM_ERRCODE_FILE_READ_FAIL	22103	fail to read	-
DBM_ERRCODE_FILE_WRITE_FAIL	22104	fail to write	-
DBM_ERRCODE_NOT_ACTIVE_INSTANCE	22105	a instance not active-mode	-
DBM_ERRCODE_DIRECT_INVALID_KEY_DATA_TYPE	22106	a index key column must have a data type as (long, int, short)	-
DBM_ERRCODE_DIRECT_NEED_INDEX	22107	at first, need to create a index to use a direct table	-
DBM_ERRCODE_FAIL_TO_PREPARE_DISK_LOG	22108	fail to prepare a disk logfile	-
DBM_ERRCODE_FAIL_TO_PREPARE_REPL	22109	fail to prepare replication	-
DBM_ERRCODE_FAIL_TO_PREPARE_TABLE	22110	fail to prepare a table	-
DBM_ERRCODE_NOT_FOUND	22111	no data found	-

Error defined name	Error code	Detail message	Description
DBM_ERRCODE_REPL_NOT_CONNECTED	22112	a replication-session not connected	-
DBM_ERRCODE_TOO_MANY_RESULT	22113	a result-set has too many rows to process	-
DBM_ERRCODE_NOT_EXIST_PROC	22114	a procedure not found	-
DBM_ERRCODE_ALREADY_EXIST_PROC	22115	a procedure already exists	-
DBM_ERRCODE_INVALID_IDENTIFIER	22116	invalid identifier	-
DBM_ERRCODE_CASE_NOT_FOUND	22117	case not found	-
DBM_ERRCODE_CURSOR_ALREADY_OPENED	22118	a cursor already opened	-
DBM_ERRCODE_CURSOR_NOT_OPENED	22119	a cursor not opened	-
DBM_ERRCODE_EXCEPTION_DUPLICATED	22120	a exception duplicated(Line=%d,Column=%d)	-
DBM_ERRCODE_RAISE_USER_EXCEPTION	22121	a user exception raised	-
DBM_ERRCODE_UNHANDLE_EXCEPTION	22122	unhandled exceptions	-
DBM_ERRCODE_PREPARE_PROCEDURE	22123	fail to prepare a object/statement of procedure (Line=%d, Column=%d)	-
DBM_ERRCODE_EXIT_ONLY_AT_LOOP	22124	a exit/continue statement is able to be used in loop-statement	-
DBM_ERRCODE_EXECUTE_PROC_FAIL	22125	fail to execute a procedure statement (Line=%d)	-
DBM_ERRCODE_CHANGED_PLAN	22126	changed index after dbmPrepareStmt	-
DBM_ERRCODE_ALREADY_ATTACH_TID	22127	current thread-id already attached at (Trans=%d)	-
DBM_ERRCODE_TOO_MANY_SEGMENT_EXTEND	22128	a count of segment expected too many chunk. (need less than 999)	-
DBM_ERRCODE_GET_SEMAPHORE	22129	error get semaphore (id=%ld)	-
DBM_ERRCODE_CURSOR_API_ALREADY_OPENED	22130	open cursor api already executed	-
DBM_ERRCODE_CURSOR_API_ALREADY_CLOSED	22131	close cursor api already executed	-
DBM_ERRCODE_DDL_RAISED	22132	a handle of table re-prepared as ddl executed	-

Error defined name	Error code	Detail message	Description
DBM_ERRCODE_BEGIN_TRANS_STAT	22133	a operation can not be executed as other transaction (transId=%d) already began	-
DBM_ERRCODE_NEED_NO_TX_AT_DDL	22134	a operation can not be executed as previous transaction need commit or rollback	-
DBM_ERRCODE_ALREADY_EXIST_LIB	22135	a library already exists	-
DBM_ERRCODE_NOT_EXIST_LIB	22136	a function not exists	-
DBM_ERRCODE_EXECUTE_USER_FUNC_FAIL	22137	fail to execute a user function (%s:RetCode=%d)	-
DBM_ERRCODE_INVALID_TIME_OPTION	22138	invalid time option	-
DBM_ERRCODE_PORT_OUT_OF_RANGE	22139	port out of range	-
DBM_ERRCODE_CLIENT_MAX_OUT_OF_RANGE	22140	client max out of range	-
DBM_ERRCODE_PROCESS_MAX_OUT_OF_RANGE	22141	process max out of range	-
DBM_ERRCODE_PROCESS_MIN_OUT_OF_RANGE	22142	process min out of range	-
DBM_ERRCODE_PROCESS_CNT_OUT_OF_RANGE	22143	process count out of range	-
DBM_ERRCODE_GSB_CREATE_FAIL	22145	create gsb failed	-
DBM_ERRCODE_GSB_DROP_FAIL	22146	drop gsb failed	-
DBM_ERRCODE_INVALID_JSON_KEY_VALUE	22147	a key value has not to be json-object or array	-
DBM_ERRCODE_INVALID_JSON_VALUE	22148	invalid json key-string or valueOrType	-
DBM_ERRCODE_ALREADY_EXISTS_REPL	22149	a replication name already exists	-
DBM_ERRCODE_INVALID_DIRECT_TABLE_INDEX	22150	a column is not valid as index in direct-table	-
DBM_ERRCODE_INVALID_REPL_DIR	22151	a value of unsent_dir property is not matched between anchor-file and property-file	-
DBM_ERRCODE_INVALID_PROP	22152	a property(%s) is not found or invalid value	-
DBM_ERRCODE_NEED_JOIN_INDEX	22153	a join table need index	-
DBM_ERRCODE_DDL_NOT_ALLOWED_IN_REPL	22154	a DDL not allowed as a table involved in replication	-
DBM_ERRCODE_NEED_INDEX_ON_OPERATION	22155	a operation can not executed as some table need unique-index"	-

Error defined name	Error code	Detail message	Description
DBM_ERRCODE_ODBC_CALL_FAIL	22156	fail to call ODBC_LIB (Detail:%s)	-
DBM_ERRCODE_NOT_EXIST_DSN	22157	a dsn not exists	-
DBM_ERRCODE_ODBC_LIB_OPEN_FAIL	22158	fail to open odbc-library	-
DBM_ERRCODE_ODBC_GET_SYMBOL_FAIL	22159	fail to get a function symbol of mapping ODBC API	-
DBM_ERRCODE_INVALID_JSON_KEY_SIZE	22160	a json key is too long	-
DBM_ERRCODE_ERROR_HTTP	22161	http failed	-
DBM_ERRCODE_INVALID_LIMIT_OPTION	22162	invalid limit option	-
DBM_ERRCODE_NOT_ALLOWED_UPDATE	22163	a update operation not allowed on a record with expired-time	-
DBM_ERRCODE_NOT_INVALID_CREATE_TIME	22164	a create-time of segment is invalid	
DBM_ERRCODE_CANNOT_UPDATE_KEY_COLUMN	22165	cannot update key column value	