Reference Manual

오디세우스/OOSQL

Version 5.0

Manual Release 1

2016 년 8 월

Advanced Information Technology Research Center (AITrc)

KAIST

목 차

1. OOSQL API

1.1. 시스템 관련 인터페이스

1.1.1. OOSQL_CreateSystemHandle

Syntax

Four OOSQL_CreateSystemHandle(OOSQL_SystemHandle* systemHandle, Four* procIndex)

Parameters

IN/OUT	이름	타입	설명
OUT	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
OUT	procIndex	Four*	process 의 식별자

Description

프로세스를 시작하고 OOSQL가 사용하는 내부 자료 구조를 초기화한다.

Return value

eNOERROR : OOSQL을 정상적으로 시작 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

Four              procIndex;
OOSQL_SystemHandle systemHandle;
Four              e;

e = OOSQL_CreateSystemHandle(&systemHandle, &procIndex);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_DestroySystemHandle(&systemHandle, procIndex);
if(e < eNOERROR) /* error 처리 */
……
```

1.1.2. OOSQL_DestroySystemHandle

Syntax

Four OOSQL_DestroySystemHandle(OOSQL_SystemHandle* systemHandle, Four procIndex)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	procIndex	Four	Process 의 식별자

Description

OOSQL이 사용하는 내부 자료 구조를 말기화하고 프로세스를 종료한다.

Return value

eNOERROR : OOSQL을 정상적으로 종료하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

Four               procIndex;
OOSQL_SystemHandle systemHandle;
Four               e;

e = OOSQL_CreateSystemHandle(&systemHandle, &procIndex);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_DestroySystemHandle(&systemHandle, procIndex);
if(e < eNOERROR) /* error 처리 */
……
```

1.2. 데이터베이스 및 볼륨 관련 인터페이스

1.2.1. OOSQL_Mount

Syntax

Four OOSQL_Mount(OOSQL_SystemHandle* systemHandle, Four numDevices, char** devNames, Four* volID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	numDevices	Four	볼륨을 구성하는 디바이스의 수
IN	devNames	char**	디바이스의 이름들의 배열

OUT	volID	Four*	마운트된 볼륨의 ID

Description

주어진 볼륨을 저장 시스템이 사용할 수 있도록 마운트한다. 하나의 볼륨은 하나이상의 디바이스에 의해 구성될 수 있기 때문에 입력으로 볼륨을 구성하는 디바이스의 개수와 디바이스의 이름들을 배열로 넘긴다. 디바이스의 이름은 UNIX 파일 시스템에서의 이름을 사용한다. 볼륨이 성공적으로 마운트되면 그 볼륨의 식별자를 반환해 준다.

Return value

eNOERROR : 볼륨을 마운트 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
char            devNameStrings[2][256];
char**          devNames;
Four            volID;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
e = OOSQL_Mount(&systemHandle, 2, devNames, &volID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.2. OOSQL_Dismount

Syntax

Four OOSQL_Dismount(OOSQL_SystemHandle* systemHandle, Four volID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자

IN	volID	Four	데이타베이스 볼륨의 ID

Description

마운트된 볼륨을 디스마운트한다. 디스마운트할 볼륨은 마운트시 반환해준 볼륨 식별자를 통하여 지정한다.

Return value

eNOERROR : 볼륨을 마운트 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
char            devNameStrings[2][256];
char**          devNames;
Four            volID;
……
strcpy(devNameStrings[0], "/device1-name");
strcpy(devNameStrings[1], "/device2-name");

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
e = OOSQL_Mount(&systemHandle, 2, devNames, &volID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.3. OOSQL_MountDB

Syntax

Four OOSQL_MountDB(OOSQL_SystemHandle* systemHandle, char* databaseName, Four* databaseID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseName	char*	데이터베이스의 이름
OUT	databaseID	Four*	마운트된 데이터베이스의 ID

Description

주어진 데이터베이스를 저장시스템에서 사용할 수 있도록 마운트한다. 하나의 데이터베이스는 하나이상의 볼륨으로 구성되며, 하나의 볼륨은 하나 이상의 디바이스로 구성된다. 데이터베이스는 OOSQL_CreateDB라는 유틸리티를 사용하여 작성하며 이때 주어지는 데이터베이스의 이름을 사용하여 마운트한다. 마운트된 데이터베이스에서 사용하고자 하는 볼륨의 ID를 얻어오기 위해서는 OOSQL_GetVolumeID를 사용하여 얻어온다. 데이터베이스는 시스템에서 단 한번만 마운트할 수 있다.

Return value

eNOERROR : 데이터베이스를 마운트 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;

……
e = OOSQL_MountDB(&systemHandle, "database-name", &databaseID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_DismountDB(&systemHandle, databaseID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.4. OOSQL_DismountDB

Syntax

Four OOSQL_DismountDB(OOSQL_SystemHandle* systemHandle, Four databaseID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseID	Four	데이타베이스의 ID

Description

마운트된 데이터베이스를 디스마운트한다. 디스마운트할 데이터베이스는 마운트시 반환해준 데이터베이스 식별자를 통하여 지정한다.

Return value

eNOERROR : 데이터베이스를 마운트 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;

……
e = OOSQL_MountDB(&systemHandle, "database-name", &databaseID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_DismountDB(&systemHandle, databaseID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.5. OOSQL_MountVolumeByVolumeName

Syntax

Four OOSQL_MountVolumeByVolumeName(OOSQL_SystemHandle* systemHandle, char* databaseName, char* volumeName, Four* volID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseName	char*	데이터베이스의 이름
IN	volumeName	char*	볼륨 이름
OUT	volID	Four*	마운트된 볼륨의 ID

Description

주어진 데이터베이스의 볼륨을 저장시스템에서 사용할 수 있도록 마운트한다. OOSQL_MountDB가 데이터베이스를 구성하는 모든 볼륨을 마운트하는 반면, OOSQL_MountVolumeByVolumeName은 특정 데이터베이스의 특정 볼륨을 개별적으로 마운트한다. 기능적으로는 OOSQL_Mount와 같은기능을 하나, 인수가 데이터베이스 이름, 볼륨이름을 준다는 측면에서 다르다.

OOSQL_Mount, OOSQL_MountVolumeByVolumeName으로 마운트된 볼륨은 OOSQL_Dismount를 통해 디스마운트를 해야 한다.

Return value

eNOERROR : 데이터베이스의 볼륨를 마운트 하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;

……
e = OOSQL_MountVolumeByVolumeName(&systemHandle, "database-name",
"volume-name", &volID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.6. OOSQL_GetVolumeID

Syntax

Four OOSQL_GetVolumeID(OOSQL_SystemHandle* systemHandle, Four databaseID, char* volumeName, Four* volumeID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseID	Four	데이타베이스의 ID
IN	volumeName	char*	볼륨의 이름
OUT	volumeID	Four*	볼륨의 ID

Description

마운트된 데이터베이스를 구성하는 볼륨들중에서 주어진 이름을 가지는 볼륨의 ID를 반환한다.

Return value

eNOERROR : 볼륨 ID를 받아왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;
Four            volID;

……
e = OOSQL_MountDB(&systemHandle, "database-name", &databaseID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_GetVolumeID(&systemHandle, databaseID, "volume-name",
          &volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.7. OOSQL_GetUserDefaultVolumeID

Syntax

Four OOSQL_GetUserDefaultVolumeID(OOSQL_SystemHandle* systemHandle, Four databaseID, Four* volumeID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseID	Four	데이타베이스의 ID
OUT	volumeID	Four*	볼륨의 ID

Description

마운트된 데이터베이스를 구성하는 볼륨들중에서 디폴트로 지정된 볼륨의 ID를 반환한다. 특정 볼륨을 디폴트로 지정하기 위해서는 OOSQL_SetUserDefaultVolumeID를 사용하며 데이터베이스를 마운트한후 자동적으로 첫번때 볼륨이 디폴트로 지정된다.

Return value

eNOERROR : 볼륨 ID를 받아왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"
```

```
OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;
Four            volID;

……
e = OOSQL_MountDB(&systemHandle, "database-name", &databaseID);
if(e < eNOERROR) /* error 처리 */
……
e    =    OOSQL_GetUserDefaultVolumeID(&systemHandle,    databaseID,
&volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.2.8. OOSQL_SetUserDefaultVolumeID

Syntax

Four OOSQL_SetUserDefaultVolumeID(OOSQL_SystemHandle* systemHandle, Four databaseID, Four volumeID)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	databaseID	Four	데이타베이스의 ID
IN	volumeID	Four	볼륨의 ID

Description

마운트된 데이터베이스를 구성하는 볼륨들중에서 주어진 볼륨을 디폴트로 지정한다.

Return value

eNOERROR : 볼륨 ID를 디폴트로 지정하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;
Four            volID;
……
e = OOSQL_MountDB(&systemHandle, "database-name", &databaseID);
```

```
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_GetVolumeID(&systemHandle, databaseID, "volume-name",
                      &volID);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_SetUserDefaultVolumeID(&systemHandle, databaseID, volID);
if(e < eNOERROR) /* error 처리 */
……
```

1.3. 트랜잭션 관련 인터페이스

1.3.1. OOSQL_TransBegin

Syntax

Four OOSQL_TransBegin(OOSQL_SystemHandle* systemHandle, XactID *xactId, ConcurrencyLevel cclevel)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
OUT	xactId	XactID*	트랜잭션 ID
IN	cclevel	CONCURRENCYLEVEL	주어진 트랜잭션이 사용할 동시성 수준

Description

새로운 트랜잭션을 초기화하고 트랜잭션의 시작임을 선언한다.생성된 트랜잭션은 이를 식별하기 위한 식별자가 부여되며 xactId로 반환된다.

cclevel은 주어진 트랜잭션이 사용할 동시성 수준이다. 동시성 수준은 여러 트랜잭션들이 동시에 수행될경우, 이를 어떻게 처리할 것인가를 결정한다. cclevel은 ConcurrencyLevel 타입으로 다음과 같이 정의된다. typedef enum { X_BROWSE_BROWSE, X_CS_BROWSE, X_CS_CS, X_RR_BROWSE, X_RR_CS, X_RR_RR } ConcurrencyLevel;

현 버전의 오디세우스/OOSQL은 X_BROWSE_BROWSE, X_RR_RR의 두가지 동시성 수준을 사용한다.

X_BROWSE_BROWSE는 no read lock, long write lock을 사용하는 수준으로 읽기를 주로 하는 트랜젝션에서 사용한다. X_BROWSE_BROWSE로 수행되는 트랜젝션은 다른 트랜젝션이 write연산을 하더래도 주어진 볼륨(데이터)에 대한 read 연산을 수행할 수 있으며 다른 트랜젝션이 write연산을 안할때, write 연산을

수행할 수 있다.

X_RR_RR은 long read lock, long write lock으로 쓰기를 주로하는 트랜젝션에서 사용한다. X_RR_RR은 다른 트랜젝션이 write연산을 수행할 경우, 주어진 볼륨(데이터)에 대한 read연산을 수행할 수 없으며, 다른 트랜젝션이 X_RR_RR 수준에서 read연산을 안할 때 write연산을 수행할수 있다. 또한 다른 트랜젝션이 write연산을 안할때, write 연산을 수행할 수 있다.

Return value

eNOERROR : 트랜잭션을 성공적으로 시작하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
XactID          xactID;

……
e = OOSQL_TransBegin(&systemHandle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_TransCommit(&systemHandle, &xactID);
if(e < eNOERROR) /* error 처리 */
……
```

1.3.2. OOSQL_TransCommit

Syntax

Four OOSQL_TransCommit(OOSQL_SystemHandle* systemHandle, XactID* xactId)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	xactId	XactID*	트랜잭션 ID

Description

주어진 트랜잭션을 완료한다. 트랜잭션이 완료되면 트랜잭션간에 수행된 데이터베이스 관련 연산이 실제로 데이터베이스에 반영된다.

Return value

eNOERROR : 트랜잭션을 성공적으로 완료하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
XactID          xactID;

……
e = OOSQL_TransBegin(&systemHandle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_TransCommit(&systemHandle, &xactID);
if(e < eNOERROR) /* error 처리 */
……
```

1.3.3. OOSQL_TransAbort

Syntax

Four OOSQL_TransAbort(OOSQL_SystemHandle* systemHandle, XactID* xactId)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	xactId	XactID*	트랜잭션 ID

Description

주어진 트랜잭션을 철회한다. 트랜잭션이 철회되면 트랜잭션간에 수행된 데이터베이스 관련 연산은 모두 취소되며 데이터베이스 상태는 트랜잭션 시작 이전 상태가 된다.

Return value

eNOERROR : 트랜잭션을 성공적으로 철회하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
```

```
XactID          xactID;

……
e = OOSQL_TransBegin(&systemHandle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_TransAbort(&systemHandle, &xactID);
if(e < eNOERROR) /* error 처리 */
……
```

1.4. 질의 관련 인터페이스

1.4.1. OOSQL_AllocHandle

Syntax

Four OOSQL_AllocHandle(OOSQL_SystemHandle* systemHandle, Four volID, OOSQL_Handle* handle)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	Four	데이타베이스 볼륨의 ID
OUT	handle	OOSQL_Handle*	할당된 핸들의 정보

Description

질의어 관련 OOSQL 연산을 수행하기 위한 핸들을 획득한다. 모든 질의어 관련 OOSQL 연산은 이 핸들을 통해 이루어진다.

Return value

eNOERROR : 핸들을 정상적으로 획득하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle       handle;
Four               volID;
Four               e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
```

```
if(e < eNOERROR) /* error 처리 */
……
```

1.4.2. OOSQL_FreeHandle

Syntax

Four OOSQL_FreeHandle(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	반환하고자 하는 핸들의 정보

Description

질의어 관련 OOSQL을 수행하기 위해 획득한 핸들을 반환한다.

Return value

eNOERROR : 핸들을 정상적으로 반환하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.3. OOSQL_Prepare

Syntax

Four OOSQL_Prepare(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, char* stmtText, OOSQL_SortBufferInfo* sortBuffInfo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들 정보
IN	stmtText	**CHAR***	수행하고자 하는 질의문
INOUT	sortBufferInfo	**OOSQL_SORTBUFFERINFO***	질의처리시 사용하는 Sort 의 버퍼에 관한 정보, NULL 을 건내주면 질의 수행중인 볼륨을 기반으로한 Disk Sort 를 한다.

Description

주어진 질의문에 구문 오류가 있는지 검사하고 이를 수행하기 위한 준비를 한다.

OOSQL에서는 질의를 처리하는 중에 필요에 따라 Sort를 하게 된다. Sort는 Memory 혹은 Disk를 기반으로 수행된다. Memory를 기반으로 Sort를 하는 것이 더 빠른 수행을 보인다. Sort를 사용하는 대표적인 질의로는 텍스트 정보 검색에서 키워드에 절단연산을 사용한 경우이다.

OOSQL_SortBufferInfo의 구조는 다음과 같다.

```
typedef struct {
    OOSQL_SortBufferMode            mode;
    OOSQL_DiskSortBufferInfo                diskInfo;
    OOSQL_MemorySortBufferInfo      memoryInfo;
} OOSQL_SortBufferInfo;
```

mode에는 Disk를 기반으로 Sort를 할지, Memory를 기반으로 Sort할지, Memory로 Sort하다가 모자라면 Disk로 Sort할지를 각각 OOSQL_SB_USE_DISK, OOSQL_SB_USE_MEMORY, OOSQL_SB_USE_MEMORY_WITH_DISK로 지정한다.

diskInfo는 모든 mode에 대해 채워줘야 하는 부분으로 어떤 볼륨에서 Sort를 할 지를 지정한다. OOSQL_DiskSortBufferInfo의 구조는 다음과 같다.

```
typedef struct {
    Four    sortVolID;
} OOSQL_DiskSortBufferInfo;
```

memory는 mode가 OOSQL_SB_USE_MEMORY, OOSQL_SB_USE_MEMORY_WITH_DISK일때 채워줘야 하는 부분으로 어떤 memory에서 Sort할 지를 지정한다. OOSQL_MemorySortBufferInfo의 구조는 다음과 같다.

```
typedef struct {
    void*                           sortBufferPtr;
    Four                            sortBufferLength;
    Four                            sortBufferUsedLength;
} OOSQL_MemorySortBufferInfo;
```

sortBufferPtr은 Sort를 할 memory가 있는 위치이고 sortBufferLength는 memory의 크기이다. Sort를 하기 위한 memory는 사용자가 설정해야 한다. sortBufferUsedLength는 실제로 사용한 memory의 크기이다.

mode가 OOSQL_SB_USE_MEMORY이고 sortBufferLength가 질의를 수행하기 위해 필요한 memory보다 작다면 eNEEDMORESORTBUFFERMEMORY_OOSQL이라는 에러가 반환된다. 이 에러가 반환되면 사용자는 memory의 크기를 늘려 다시 OOSQL_Prepare를 불러줘야 한다.

Return value

eNOERROR : 질의 수행을 위한 준비를 마쳤음

eNEEDMORESORTBUFFERMEMORY_OOSQL : memory sort를 하기 위한 메모리가 부족

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.4. OOSQL_Execute

Syntax

Four OOSQL_Execute(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	이름	타입	설명

IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들

Description

OOSQL_Prepare로부터 준비된 질의문을 수행하고 첫 질의 결과를 읽을 준비를 한다.

Return value

eNOERROR : 질의를 성공적으로 수행하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.5. OOSQL_ExecDirect

Syntax

Four OOSQL_ExecDirect(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, char* stmtText, OOSQL_SortBufferInfo* sortBuffInfo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자

IN	handle	OOSQL_Handle	OOSQL 핸들
IN	stmtText	char*	수행하고자 하는 질의문
INOUT	sortBufferInfo	**OOSQL_SORTBUFFERINFO***	질의처리시 사용하는 Sort 의 버퍼에 관한 정보, NULL 을 건내주면 질의 수행중인 볼륨을 기반으로한 Disk Sort 를 한다.

Description

주어진 질의문에 구문 오류가 있는지 검사하고 이를 수행하기 위한 준비를 한 다음 이를 수행하여 첫 질의 결과를 읽을 준비를 한다.

Return value

eNOERROR : 질의를 성공적으로 수행하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four              e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_ExecuteDirect(&systemHandle, handle, "select * from test-
table", NULL);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.6. OOSQL_Next

Syntax

Four OOSQL_Next(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	이름	타입	설명

IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들

Description

다음 질의 결과를 가져온다. 더 이상 가져올 질의 결과가 없을 경우 ENDOFEVAL을 반환한다.

Return value

ENDOFEVAL : 반환할 결과가 없음

eNOERROR : 질의 결과를 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

while((e = OOSQL_Next(&systemHandle, handle)) != ENDOFEVAL)
{
   if(e < eNOERROR) /* error 처리 */
   ……
}
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.7. OOSQL_GetData

Syntax

Four OOSQL_GetData(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two columnNumber, Four startPos, void* bufferPtr, Four

bufferLength, Four* returnLength)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	columnNumber	Two	질의 결과를 구성하는 컬럼의 번호
IN	startPos	Four	컬럼값 중 읽고자 하는 부분의 시작 위치
INOUT	bufferPtr	Void*	컬럼값을 받기 위한 버퍼
IN	bufferLength	Four	버퍼의 길이
OUT	returnLength	Four*	읽어들인 데이터의 길이

Description

질의 결과를 구성하는 하나의 컬럼의 값을 읽는다. columnNubmer는 읽고자하는 컬럼의 번호로 첫번째 컬럼은 0이 된다. startPos와 bufferLength는 질의 결과중에서 읽고자하는 부분을 지정하는 인수로 시작 위치와 길이를 나타낸다. 질의 결과는 bufferPtr이 가리키는 메모리에 저장된다. returnLength는 질의 결과로부터 실제로 읽어 들인 결과값의 길이다.

Return value

eNOERROR : 질의 결과 컴럼의 값을 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;
char             buffer[1024];
Four             returnLength;
……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */
```

```
e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

while((e = OOSQL_Next(&systemHandle, handle)) != ENDOFEVAL)
{
    if(e < eNOERROR) /* error 처리 */
    ……
e = OOSQL_GetData(&systemHandle, handle, 0, 0, buffer,
sizeof(buffer), &returnLength);
if(e < eNOERROR) /* error 처리 */
}
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.8. OOSQL_GetMultipleResults

Syntax

Four OOSQL_GetMultipleResults(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Four nResultsToRead, void* headerBuffer, Four headerBufferSize, void* dataBuffer, Four dataBufferSize, Four* nResultsRead);

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	NRESULTSTOREAD	FOUR	읽고자 하는 결과의 갯수, -1 이면 버퍼 크기가 되는 대로 읽음
IN	headerBuffer	void*	헤더 버퍼, NULL 로 주어질 경우, 헤더 정보를 생성하지 않음
IN	HEADERBUFFERSIZE	Four	헤더 버퍼의 크기

IN	dataBuffer	void*	데이터 버퍼, NULL 로 주어질 경우, 데이터 정보를 생성하지 않음
IN	dataBufferSize	Four	**데이터 버퍼의 크기**
OUT	nResultsRead	Four*	읽어들인 결과의 개수

Description

복수개의 OOSQL 질의 결과 객체를 한꺼번에 읽어온다. 이 함수는 OOSQL_Next 함수와 OOSQL_GetData 함수를 여러 번 호출하는 효과와 동일한 효과를 얻을 수 있으며 질의 결과 처리 속도를 훨씬 향상 시킬 수 있다. 복수개의 객체는 헤더 버퍼와 데이터 버퍼에 각각 읽혀진다. 헤더 버퍼에는 각 객체를 해석하는데 필요한 헤더 정보가 들어 있고 데이터 버퍼에는 실제 데이터가 들어간다. 만약 고정 길이 컬럼의 데이터만 얻어오는 질의식을 수행하였다면 데이터 버퍼에서 해당 데이터의 크기만큼 이동하면서 데이터를 해석할 수 있으므로 헤더 정보를 생성할 필요가 없다. 헤더 버퍼의 헤더 정보는 아래의 매크로를 통해 해석될 수 있다.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_OFFSET(headerBuffer, nColumns, i)

  i번째 결과객체의 dataBuffer에서의 위치를 반환한다.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_SIZE(headerBuffer, nColumns, i)

  i번째 결과객체의 dataBuffer에서의 크기를 반환한다.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_ISNULL(headerBuffer, nColumns, i, j)

  i번째 결과객체를 구성하는 j번째 컬럼이 Null인지 아닌지를 반환한다. j는 select절의 attribute 수까지 커질 수 있다.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_SIZE(headerBuffer, nColumns, i, j)

  i번째 결과객체를 구성하는 j번째 컬럼의 dataBuffer내에서의 크기를 반환한다. j는 select절의 attribute 수까지 커질 수 있다. dataBuffer내에서의 크기는 database내에서의 크기와 다를 수 있다. 왜냐하면, database내에서 매우 큰 객체가 있는 경우, 이는 결과 객체를 memory buffer에 다 올리지 못하기 때문이다. memory buffer에는 최대 8KB 까지만 결과가 저장된다.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_REALSIZE(headerBuffer,

nColumns, i, j)

i번째 결과객체를 구성하는 j번째 컬럼의 database내에서의 크기를 반환한다. j는 select절의 attribute 수까지 커질 수 있다.

□ OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_OID(headerBuffer, nColumns, i, j)

i번째 결과객체를 구성하는 j번째 컬럼이 저장된 객체의 OID를 반환한다. j는 select절의 attribute 수까지 커질 수 있다.

□ OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_COLNO(headerBuffer, nColumns, i, j)

i번째 결과객체를 구성하는 j번째 컬럼이 저장된 객체의 database에서의 컬럼 번호를 반환한다. j는 select절의 attribute 수까지 커질 수 있다.

Return value

eNOERROR : 질의 결과 값을 성공적으로 가져왔음

ENDOFEVAL : 더 이상 읽어 들일 질의 결과가 없음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
char*             dataBuffer;
Four              dataBufferSize;
Four              objectNum;
Four              length;
OID               oid;
Four              e, i;
……
dataBufferSize = 1024000;
dataBuffer = (char *)malloc(dataBufferSize);
……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select user_page from
user_page where match(description, \'한국\')>0", NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

char* pOidBuffer       = dataBuffer;
Four  oidBufferSize    = dataBufferSize;
Four  nResultsRead     = 0;
```

```
Four  nTotalResultsRead = 0;
Four  freeOidBufferSize = oidBufferSize;

while  ((e  =  OOSQL_GetMultipleResults(&systemHandle,  handle,  -1,
NULL, 0, pOidBuffer, freeOidBufferSize, &nResultsRead)) != ENDOFEV
{
   if (e < eNOERROR) /* error 처리 */

   nTotalResultsRead += nResultsRead;

   /* buffer의 80%를 채운 경우 doubling */
   if(nResultsRead >= ((freeOidBufferSize / sizeof(OID)) * 4 / 5))
   {
      oidBufferSize *= 2;
      dataBuffer =
         (char *)realloc(dataBuffer, oidBufferSize);
      pOidBuffer =
         (char*)dataBuffer + nTotalResultsRead * sizeof(OID);
      freeOidBufferSize =
         oidBufferSize - nTotalResultsRead * sizeof(OID);
   }
   else
   {
      pOidBuffer =
         (char*)dataBuffer + nTotalResultsRead * sizeof(OID);
      freeOidBufferSize =
         oidBufferSize - nTotalResultsRead * sizeof(OID);
   }
}

objectNum = nTotalResultsRead;
length    = nTotalResultsRead * sizeof(OID);
……
pOidBuffer = (char *)dataBuffer;
……
for (i = 0; i < objectNum; i++)
{
   /* dataBuffer에서 OID를 읽음 */
   memcpy((char *)&oid, pOidBuffer, sizeof(OID));
   /* OID 값을 사용하여 필요한 작업을 수행 */
   ……
   /* dataBuffer의 offset을 OID 크기만큼 증가시킴 */
   pOidBuffer += sizeof(OID);
}
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
free(dataBuffer);
……
```

1.4.9. OOSQL_GetMultiColumnData

Syntax

Four OOSQL_GetMultiColumnData(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Four nColumns, OOSQL_GetDataStruct* getDataStruct)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	nColumns	Four	읽어들일 컬럼의 개수
IN	getDataStruct	OOSQL_GetDataStruct*	각 컬럼별 읽을 내용을 정의한 구조체의 배열

Description

질의 결과를 구성하는 복수의 컬럼들을 읽는다. nColumns는 읽고자 하는 컬럼의 갯수이고 getDataStruct는 읽고자하는 컬럼들에 대한 정보를 배열 나타낸 것이다.

getDataStruct는 OOSQL_GetDataStruct로 다음과 같이 정의된다.

```
typedef struct {
    Two                         columnNumber;
    Four                        startPos;
    void*                       bufferPtr;
    Four                        bufferLength;
    Four                        returnLength;
} OOSQL_GetDataStruct;
```

OOSQL_GetDataStruct의 columnNumber은 읽고자 하는 컬럼의 번호이고 startPos는 일고자하는 컬럼에서의 데이터 시작 위치이다. bufferPtr는 읽은 데이터가 반환될 버퍼의 포인터이고 bufferLength은 bufferPtr이 가리키는 버퍼의 길이이다. returnLength는 읽혀진 데이터의 길이를 반환한다.

Return value

eNOERROR : 질의 결과 컬럼의 값을 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"
```

```
OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;
char             buffer1[1024], buffer2[1024];
OOSQL_GetDataStruct getData[2];
……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

while((e = OOSQL_Next(&systemHandle, handle)) != ENDOFEVAL)
{
   if(e < eNOERROR) /* error 처리 */
   ……

getData[0].columnNumber = 0;
getData[0].startPos = 0
getData[0].bufferPtr = buffer1;
getData[0].bufferLength = sizeof(buffer1);
getData[1].columnNumber = 1;
getData[1].startPos = 0
getData[1].bufferPtr = buffer2;
getData[1].bufferLength = sizeof(buffer2);
e = OOSQL_GetMultiColumnData(&systemHandle, handle, 2, getData)
if(e < eNOERROR) /* error 처리 */
}
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.10. OOSQL_PutData

Syntax

Four OOSQL_PutData(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two columnNumber, Four startPos, void* columnValuePtr, Four bufferLength)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자

IN	handle	OOSQL_Handle	OOSQL 핸들
IN	columnNumber	Two	인수의 번호
IN	startPos	Four	쓰고자 하는 부분의 시작 위치
IN	columnValuePtr	void*	컬럼값을 받기 위한 버퍼
IN	bufferLength	Four	버퍼의 길의

Description

질의식에 사용한 인수에 값을 지정한다. 인수는 질의식에서 '?'로 표시된다. 질의식에 직접 쓸 수 없는 바이너리 데이터나 큰 크기를 가지는 멀티미디어 데이터의 값을 지정하는데 유용하다. columnNumber는 인수의 번호로 질의식에 나타난 '?'의 순서에 따라 번호가 정해진다. 첫번째 인수는 0이다.

Return value

eNOERROR : 값을 성공적으로 수정하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four               volID;
Four             e;
Char             buffer[1024];
OOSQL_GetDataStruct getData[2];
……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle,
"insert into test-table values(?)", NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_PutData(&systemHandle, handle, 0, 0,
buffer, sizeof(buffer));
if(e < eNOERROR) /* error 처리 */

……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.11. OOSQL_GetOID

Syntax

Four OOSQL_GetOID(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two targetNumber, OID* oid);

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	targetNumber	Two	FROM 절에 사용된 테이블의 번호
OUT	oid	OID*	객체의 OID

Description

질의처리를 위해 읽어 들인 객체의 OID를 반환한다. targetNumber는 질의식의 FROM 절에 쓰인 테이블의 번호로 읽고자 하는 객체가 속한 테이블이다. 첫번째 테이블은 0번이다.

Return value

eNOERROR : OID를 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;
OID              oid;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
```

```
while((e = OOSQL_Next(&systemHandle, handle)) != ENDOFEVAL)
{
   if(e < eNOERROR) /* error 처리 */
   ……

e = OOSQL_GetOID(&systemHandle, handle, 0, &oid);
   if(e < eNOERROR) /* error 처리 */
}
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.12. OOSQL_GetNumResultCols

Syntax

Four OOSQL_GetNumResultCols(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two* nCols)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
OUT	nCols	Two*	컬럼의 개수

Description

질의 결과를 구성하는 컬럼의 갯수를 구해낸다.

Return value

eNOERROR : 질의 결과의 컬럼의 갯수를 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four             e;
Four             nCols;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
```

```
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_GetNumResultCols(&systemHandle, handle, &nCols)
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.13. OOSQL_GetResultColName

Syntax

Four OOSQL_GetResultColName(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two columnNumber, char* columnNameBuffer, Four bufferLength)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	columnNumber	Two	컬럼 번호
INOUT	**COLUMNNAMEBUFFER**	char*	컬럼의 이름을 받기 위한 버퍼
IN	bufferLength	Four	ColumnName 버퍼의 길이

Description

질의 결과를 구성하는 질의 결과 값의 이름을 가져온다.

Return value

eNOERROR : 질의 결과의 컬럼의 이름을 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four              e;
Four              nCols;
char              nameBuffer[1024];
……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_GetResultColName(&systemHandle, handle, 0, nameBuffer,
 sizeof(nameBuffer))
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.14. OOSQL_GetResultColType

Syntax

Four OOSQL_GetResultColType(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Two columnNumber, Four* columnType)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들
IN	columnNumber	Two	컬럼 번호
OUT	columnType	Four*	컬럼의 타입

Description

질의를 구성하는 컬럼의 타입을 가져온다. 가져온 타입은 다음과 같은 의미를 가진다.

반환값	SQL 타입
OOSQL_TYPE_SMALLINT	smallint
OOSQL_TYPE_INTEGER	integer
OOSQL_TYPE_REAL	real
OOSQL_TYPE_FLOAT	float
OOSQL_TYPE_DOUBLE	double precision
OOSQL_TYPE_CHAR	char
OOSQL_TYPE_VARCHAR	varchar
OOSQL_TYPE_OID	oid
OOSQL_TYPE_DATE	date
OOSQL_TYPE_TIME	time
OOSQL_TYPE_TIMESTAMP	timestamp

Return value

eNOERROR : 질의를 구성하는 컴럼의 타입을 성공적으로 가져왔음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four                    volID;
Four            e;
Four            type;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select * from test-table",
NULL);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Execute(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_GetResultColType(&systemHandle, handle, 0, &type)
if(e < eNOERROR) /* error 처리 */
```

```
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.4.15. OOSQL_EstimateNumResults

Syntax

Four OOSQL_EstimateNumResults(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, Four* nResults);

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	시스템 관리용 식별자
IN	handle	OOSQL_Handle	OOSQL 핸들 정보
OUT	nResults	**FOUR***	질의 결과 수 예측치

Description

텍스트 질의 결과 수를 추정한다. 단, 이 API는 OOSQL_Prepare() 수행 후에 호출되어야 한다.

Return value

eNOERROR : 질의 결과 수를 성공적으로 추정하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
OOSQL_Handle      handle;
Four              volID;
Four                    nResults;
Four              e;

……
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Prepare(&systemHandle, handle, "select …", NULL);
if(e < eNOERROR) /* error 처리 */
……
/* 질의 결과 수 예측치를 얻어온다. */
e = OOSQL_EstimateNumResults(&systemHandle, handle, &nResults);
```

```
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error 처리 */
……
```

1.5. 텍스트 관리 관련 인터페이스

1.5.1. OOSQL_Text_MakeIndex

Syntax

Four OOSQL_Text_MakeIndex(OOSQL_SystemHandle* systemHandle, Four volID, Four temporaryVolId, char* className)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	Four	데이타베이스 볼륨의 ID
IN	temporaryVolId	Four	임시 볼륨의 ID
IN	className	Char*	텍스트 인덱스를 최신 내용으로 바꾸고자 하는 클래스의 이름

Description

OOSQL에서는 텍스트를 데이타베이스에 넣으면서 텍스트 인덱스에 그 내용을 즉시 반영할 수 있거나 혹은 나중에 반영할 수 있다. 나중에 반영할 경우에는 이 함수를 통해 반영이 이루워 진다. 그러나 주어진 클래스의 텍스트 애트리뷰트를 DEFERED 모드로 일단 한번 설정한 경우 이 명령어를 수행하고 나서야 다시 IMMEDIATE 모드로 변경할 수 있다. 이 명령은 클래스의 모든 객체를 액세스 하므로 bulk로 데이터를 넣을 때만 사용하는 것이 바람직하다.

Return value

eNOERROR: 성공적으로 텍스트 인덱스를 구출하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
```

```
Four            volID;
……
e = OOSQL_Text_MakeIndex(&systemHandle, volID, "test-class");
if(e < eNOERROR) /* error 처리 */
……
```

1.5.2. OOSQL_Text_AddDefaultKeywordExtractor

Syntax

Four

OOSQL_Text_AddDefaultKeywordExtractor(OOSQL_SystemHandle* systemHandle, Four volID, char *keywordExtractor, Four version, char *keywordExtractorFilePath, char *keywordExtractorFunctionName, char *getNextPostingFunctionName, char *finalizeKeywordExtractorFunctionName)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	Four	데이타베이스 볼륨의 ID
IN	KEYWORDEXTRACTOR	**CHAR ***	추가할 default keyword 추출기의 이름
IN	version	Four	추가할 default keyword 추출기의 버전 번호
IN	KEYWORDEXTRACTORFILEPATH	char *	추가할 default keyword 추출기가 위치한 디렉토리 위치 정보
IN	keywordExtractorFunctionName	char *	keyword 추출기를 동작시키는 함수의 이름
IN	getNextPostingFunctionName	char*	keyword 추출기로부터 keyword 와 posting 정보를 얻어오는 함수의 이름

IN	FINALIZEKEYWORDEXTRACTORFUNCTIONNAME	char*	keyword 추출기의 동작을 끝내는 함수의 이름

Description

오디세우스/OOSQL에 default keyword 추출기를 등록한다. 사용자 정의 keyword 추출기가 별도로 정의되지 않은 모든 텍스트 컬럼에 대해서 이 default keyword 추출기가 적용된다.

Return value

eNOERROR: Default keyword 추출기를 등록하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;
……
e = OOSQL_Text_AddDefaultKeywordExtractor(&systemHandle, volID,
 "keyword-ext-name", 1, "/file-path", "openFuncName", "getFuncName",
 "closeFuncName");
if(e < eNOERROR) /* error 처리 */
……
```

1.5.3. OOSQL_Text_AddKeywordExtractor

Syntax

Four OOSQL_Text_AddKeywordExtractor(OOSQL_SystemHandle* systemHandle, Four volID, char *keywordExtractor, Four version, char *keywordExtractorFilePath, char *keywordExtractorFunctionName, char *getNextPostingFunctionName, char *finalizeKeywordExtractorFunctionName, Four *keywordExtractorNo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHA	시스템 관리용 식별자

		NDLE*	
IN	volID	Four	데이타베이스 볼륨의 ID
IN	KEYWORDEXTRACTOR	**CHAR ***	추가할 keyword 추출기의 이름
IN	version	Four	추가할 keyword 추출기의 버전 번호
IN	keywordExtractorFilePath	char *	추가할 keyword 추출기가 위치한 디렉토리 위치 정보
IN	keywordExtractorFuntionName	char *	keyword 추출기를 동작시키는 함수의 이름
IN	getNextPostingFunctionName	char*	keyword 추출기로부터 keyword 와 posting 정보를 얻어오는 함수의 이름
IN	FINALIZEKEYWORDEXTRACTORFUNTIONNAME	char*	keyword 추출기의 동작을 끝내는 함수의 이름
OUT	keywordExtractorNo	Four *	추가된 keyword 추출기의 번호

Description

오디세우스/OOSQL에 사용자 정의 keyword 추출기를 등록한다. keyword 추출기 등록후 사용자는 임의의 텍스트 컬럼에 대해 등록된 필터를 세트할 수 있다.

Return value

eNOERROR: Keyword 추출기를 등록하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;
Four            extNo;
……
e = OOSQL_Text_AddKeywordExtractor(&systemHandle, volID,
```

```
 "keyword-ext-name", 1, "/file-path", "openFuncName", "getFuncName",
 "closeFuncName", &extNo);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.4. OOSQL_Text_DropKeywordExtractor

Syntax

Four OOSQL_Text_DropKeywordExtractor(OOSQL_SystemHandle* systemHandle, Four volID, char *keywordExtractorName, Four version)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	FOUR	데이타베이스 볼륨의 ID
IN	KEYWORDEXTRACTORNAME	char *	추가할 keyword 추출기의 이름
IN	version	Four	추가할 keyword 추출기의 버전 번호

Description

오디세우스/OOSQL에 사용자 정의 keyword 추출기를 삭제한다.

Return value

eNOERROR: Keyword 추출기를 삭제하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;
Four            extNo;
……
e = OOSQL_Text_AddKeywordExtractor(&systemHandle, volID,
 "keyword-ext-name", 1, "/file-path", "openFuncName", "getFuncName",
 "closeFuncName", &extNo);
if(e < eNOERROR) /* error 처리 */
……
```

```
e = OOSQL_Text_DropKeywordExtractor(&systemHandle, volID, “keyword-
ext-name“, 1);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.5. OOSQL_Text_SetKeywordExtractor

Syntax

Four OOSQL_Text_SetKeywordExtractor(OOSQL_SystemHandle* systemHandle, Four volID, char* className, char* columnName, Four keywordExtractorNo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	FOUR	데이타베이스 볼륨의 ID
IN	className	char*	클래스 이름
IN	columnName	char*	텍스트 컬럼 이름
IN	KEYWORDEXTRACTORNO	Four	적용할 Keyword 추출기 번호

Description

주어진 텍스트 컬럼에 적용할 keyword 추출기를 set한다.

Return value

eNOERROR: Keyword 추출기를 주어진 텍스트 컬럼에 set하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;
Four            extNo;
……
e = OOSQL_Text_AddKeywordExtractor(&systemHandle, volID,
```

```
 "keyword-ext-name", 1, "/file-path", "openFuncName", "getFuncName",
 "closeFuncName", &extNo);
if(e < eNOERROR) /* error 처리 */

e = OOSQL_Text_SetKeywordExtractor(&systemHandle, volID, "class-
name", "column-name", extNo);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.6. OOSQL_Text_AddFilter

Syntax

Four OOSQL_Text_AddFilter(OOSQL_SystemHandle* systemHandle, Four volID, char *filterName, Four version, char *filterFilePath, char *filterFunctionName, Four *filterNo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	FOUR	데이타베이스 볼륨의 ID
IN	filterName	char *	추가할 filter 의 이름
IN	version	Four	추가할 filter 의 버전 번호
IN	filterFilePath	char *	추가할 filter 가 위치한 디렉토리 위치 정보
IN	filterFunction Name	char *	Filter 함수의 심볼 이름
OUT	filterNo	Four *	추가된 filter 의 번호

Description

오디세우스/OOSQL에 사용자 정의 filter를 등록한다. Filter 등록후 사용자는 임의의 텍스트 컬럼에 대해 등록된 필터를 세트할 수 있다.

Return value

eNOERROR: filter를 등록하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            volID;
Four            filterNo;
……
e = OOSQL_Text_AddFilter(&systemHandle, volID,
 "filter-name", 1, "/file-path", "funcName", &filterNo);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.7. OOSQL_Text_DropFilter

Syntax

Four OOSQL_Text_DropFilter(OOSQL_SystemHandle* systemHandle, Four volID, char *filterName, Four version)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	FOUR	데이타베이스 볼륨의 ID
IN	filterName	char *	삭제할 filter 의 이름
IN	version	Four	삭제할 filter 의 버전 번호

Description

오디세우스/OOSQL에 사용자 정의 filter를 삭제한다.

Return value

eNOERROR: filter를 삭제하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
```

```
Four            e;
Four            volID;
Four            filterNo;
……
e = OOSQL_Text_AddFilter(&systemHandle, volID,
 "filter-name", 1, "/file-path", "funcName", &filterNo);
if(e < eNOERROR) /* error 처리 */
……
e = OOSQL_Text_DropFilter(&systemHandle, volID,
 "filter-name", 1,);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.8. OOSQL_Text_SetFilter

Syntax

Four OOSQL_Text_SetFilter(OOSQL_SystemHandle* systemHandle, Four volID, char* className, char* columnName, Four filterNo)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	volID	FOUR	데이타베이스 볼륨의 ID
IN	className	char*	클래스 이름
IN	columnName	char*	텍스트 컬럼 이름
IN	filterNo	Four	적용할 filter 번호

Description

주어진 텍스트 컬럼에 적용할 필터를 set한다.

Return value

eNOERROR: filter를 주어진 컬럼에 set하였음

< eNOERROR : 오류 코드

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
```

```
Four            volID;
Four            filterNo;
……
e = OOSQL_Text_AddFilter(&systemHandle, volID,
 "filter-name", 1, "/file-path", "funcName", &filterNo);
if(e < eNOERROR) /* error 처리 */
……
e   =   OOSQL_Text_SetFilter(&systemHandle,   volID,   “class-name“,
“column-name", 1);
if(e < eNOERROR) /* error 처리 */
……
```

1.5.9. 키워드 추출 함수 prototype

오디세우스/OOSQL은 키워드 기반 검색을 하기 위해 주어진 문서로부터 키워드를 추출한다. 오디세우스/OOSQL은 키워드 추출을 위해 키워드 추출기를 사용한다. 키워드 추출기는 외부 동적 라이브러리(UNIX에서는 *.so, Windows에서는 *.dll)로 3개의 키워드 추출 함수들로 구성된다.

키워드 추출기의 키워드 추출함수는 키워드 추출기를 시작하고, 키워드 추출 결과를 반환하고, 키워드 추출기를 끝내는 기능을 수행한다. 이들 함수의 이름은 사용자가 원하는 어떤 이름을 사용해도 무방하다. 이들 이름을 사용자가 알기 위해, 이들 함수 이름들은 OOSQL_Text_AddDefaultKeywordExtractor 혹은 OOSQL_Text_AddKeywordExtractor 인터페이스를 이용하여 등록하거나 InstallDefaultKeywordExtractor, InstallKeywordExtractor 유틸리티를 사용하여 등록한다.

키워드 추출기를 구성하는 3가지 함수의 prototype은 다음과 같다. 이들 함수를 작성하는 방법은 OOSQL/example/null_keyword_extractor/에 있는 NullKeywordExtractor.c 에 잘 나타나 있다.

int openAndExecuteKeywordExtractor(Four locationOfContent, OOSQL_SystemHandle *handle, Four volId, char *className, OID *oid, Two colNo, char *inFileOrContent, Four *resultHandle)

locationOfContent : 키워드 추출을 할 내용이 들어 있는 위치를 가리킨다. OOSQL_TEXT_IN_FILE일 경우, 임시파일을 통해 그 내용이 주어 지며, 임시 파일의 이름은 inFileOrContent 인수를 통해 주어진다. OOSQL_TEXT_IN_MEMORY일 경우, 인수를 통해 그 내용이 주어 지며, 내용은 inFileOrcontent 인수를 통해 주어진다. OOSQL_TEXT_IN_DB일 경우, 그 내용을 저장하고 있는 데이터베이스 내에서의 위치가 handle, volId, className, oid, colNo를 통해 주어진다. 키워드 추출기는 이들 인수의 내용과 OOSQL_Text_FetchContent 함수를 사용하

여 읽는다.

handle : OOSQL API를 호출하기 위한 핸들로 locationOfContent가 OOSQL_TEXT_IN_DB일때만 유효한 값이 넘어온다.

volId : 키워드 추출 내용이 들어 있는 데이터베이스 볼륨의 식별자로 locationOfContent가 OOSQL_TEXT_IN_DB일때만 유효한 값이 넘어온다.

className : 키워드 추출 내용이 들어 있는 클래스의 이름으로 locationOfContent가 OOSQL_TEXT_IN_DB일때만 유효한 값이 넘어온다.

oid : 키워드 추출 내용이 들어 있는 객체의 식별자로 locationOfContent가 OOSQL_TEXT_IN_DB일때만 유효한 값이 넘어온다.

colNo : 키워드 추출 내용이 들어 있는 개체내의 컬럼 번호로 locationOfContent가 OOSQL_TEXT_IN_DB일때만 유효한 값이 넘어온다.

inFileOrContent : 키워드 추출 내용이 들어 있는 임시파일의 이름이거나 키워드 추출 내용자체로 locationOfContent가 OOSQL_TEXT_IN_FILE일때는 임시파일의 이름이고, locationOfContent가 OOSQL_TEXT_IN_MEMORY일때는 내용이 들어간다.

resultHandle : 키워드 추출 결과를 식별하는 식별자로 getAndNextKeywordExtractor와 closeKeywordExtractor의 인수로 사용된다.

이 함수는 내용으로부터 키워드를 추출하고 그 결과를 내부 메모리에 저장하는 기능을 수행한다. 내부 메모리에 저장된 결과는 resultHandle로 식별하며 이 식별자와 getAndNextKeywordExtractor함수를 통해 반환된다.

int getAndNextKeywordExtractor(Four handle, char *keyword, Four *nPositions, char *positionList)

handle : 결과를 식별하는 식별자로 openAndExecuteKeywordExtractor의 반환값이다.

keyword : 키워드 추출기로부터 추출된 키워드 문자열을 반환한다.

nPositions : 문서내에서 키워드가 나타난 횟수를 반환한다.

positionList : 문서내에서 키워드가 나타난 위치를 반환한다. 키워드 하나의 위치는 (문장 위치, 문장내 단어 위치) 로 반환되며 각각 int형을 가진다.

이 함수는 키워드 추출된 내용을 반환하는 함수로 키워드 문자열, 키워드 위치 정보를 반환한다. 모든 키워드를 다 읽은 경우, OOSQL_TEXT_DONE이 반환되고, 다 읽지 않은 경우에는 eNOERROR를 반환한다.

int closeKeywordExtractor(Four handle)

handle : 결과를 식별하는 식별자로 openAndExecuteKeywordExtractor의 반환 값이다.

이 함수는 키워드 추출된 내용을 다 읽어 더 이상 사용하지 않을 때 호출된다. 이 함수는 openAndExecuteKeywordExtractor에서 획득한 자원을 반환하는 기능을 수행한다.

1.5.10. 필터 함수 prototype

오디세우스/OOSQL은 키워드를 추출하기 이전에 키워드 추출 내용을 키워드 추출기가 읽을 수 있는 형태로 바꿔주는 필터를 사용한다. 필터는 다양한 포맷을 가지는 문서를 하나의 포맷으로 변환하는 기능을 수행하는 외부 동적 라이브러리(UNIX에서는 *.so, Windows에서는 *.dll)이다. 예를 들어, Microsoft Word나 PDF 문서를 텍스트 문서로 바꿔 주는 기능을 수행한다.

int filter(char *inFile, char *outFile)

inFile: filter 사용전 파일 이름

outFile: filter 사용후 파일 이름

여기서 filter라는 symbol은 사용자가 원하는 아무 심볼을 사용해도 무방함. OOSQL_Text_AddFilter 인터페이스를 이용하여 심볼 이름을 등록하거나 InstallFilter 유틸리티를 사용하여 등록할 수 있다.

1.6. 기타 인터페이스

1.6.1. OOSQL_GetErrorMessage

Syntax

Four OOSQL_GetErrorMessage(OOSQL_SystemHandle* systemHandle, Four errorCode, char* messageBuffer, Four bufferLength)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	errorCode	Four	에러 코드
INOUT	messageBuffer	char*	에러 메세지를 받기 위한 버퍼
IN	bufferLength	Four	MessageBuffer 의 길이

Description

주어진 에러코드를 에러메세지로 변환한다.

Return value

eNOERROR : 에러코드를 성공적으로 에러메세지로 변환함

< eNOERROR : 오류 코드

Example

```
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR)
{
   char errorMessage[4096];
OOSQL_GetErrorName(systemHandle, e, errorMessage,
                   sizeof(errorMessage));
printf("OOSQL ERROR(%s) : ", errorMessage);
OOSQL_GetErrorMessage(systemHandle, e, errorMessage,
sizeof(errorMessage));
puts(errorMessage);
return e;
}
……
```

1.6.2. OOSQL_GetErrorName

Syntax

Four OOSQL_GetErrorName(OOSQL_SystemHandle* systemHandle, Four errorCode, char* messageBuffer, Four bufferLength);

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	errorCode	Four	에러 코드
INOUT	messageBuffer	char*	에러 메세지를 받기 위한 버퍼
IN	bufferLength	Four	MessageBuffer 의 길이

Description

주어진 에러코드를 에러이름으로 바꾼다.

Return value

eNOERROR : 에러코드를 성공적으로 에러이름로 변환함

< eNOERROR : 오류 코드

Example

```
e = OOSQL_AllocHandle(&systemHandle, volID, &handle);
if(e < eNOERROR)
{
   char errorMessage[4096];
OOSQL_GetErrorName(systemHandle, e, errorMessage,
                   sizeof(errorMessage));
printf("OOSQL ERROR(%s) : ", errorMessage);
OOSQL_GetErrorMessage(systemHandle, e, errorMessage,
sizeof(errorMessage));
puts(errorMessage);
return e;
}
……
```

1.6.3. OOSQL_GetQueryErrorMessage

Syntax

Four OOSQL_GetQueryErrorMessage(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle, char* messageBuffer, Four bufferLength);

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	handle	OOSQL_Handle	질의 식별자
INOUT	messageBuffer	char*	에러 메세지를 받기 위한 버퍼
IN	bufferLength	Four	messageBuffer 의 길이

Description

주어진 질의어를 수행하면서 발생한 가장 최근 에러의 원인을 메시지로 변환하여 반환한다.

Return value

eNOERROR : 에러 메시지를 성공적으로 반환함

< eNOERROR : 오류 코드

Example

```
e = OOSQL_Prepare(&systemHandle, &handle, “select * from test-table”, NULL);
```

```
if(e < eNOERROR)
{
   char errorMessage[4096];
OOSQL_GetErrorName(systemHandle, e, errorMessage,
                   sizeof(errorMessage));
printf("OOSQL ERROR(%s) : ", errorMessage);
OOSQL_GetErrorMessage(systemHandle, e, errorMessage,
sizeof(errorMessage));
puts(errorMessage);
OOSQL_GetQueryErrorMessage(systemHandle, handle, errorMessage,
                           sizeof(errorMessage));
puts(errorMessage); \
return e;
}
……
```

1.6.4. OOSQL_OIDToOIDString

Syntax

Four OOSQL_OIDToOIDString(OOSQL_SystemHandle* systemHandle, OID* oid, char* oidString)

Parameters

IN/OUT	이름	타입	설명
IN	systemHandle	OOSQL_SYSTEMHANDLE*	시스템 관리용 식별자
IN	oid	OID*	OID
INOUT	oidString	char*	OID String

Description

주어진 OID를 string 형태로 바꾼다. oidString의 크기는 33 byte 이상이여야 한다. OID를 가지고 객체의 값을 알거나, 수정하거나 지우는 질의문인 SELECT FROM OBJECT, UPDATE OBJECT SET, DELETE FROM OBJECT에서 oid를 질의식에 넘겨줄때 이 인터페이스를 통해 oid string으로 전달한다.

Return value

eNOERROR : OID를 성공적으로 OIDString으로 변환하였음

< eNOERROR : 오류 코드

Example

```
OID  oid;
char  oidString[33];
```

```
e = OOSQL_OIDtoOIDString(&systemHandle, &oid, oidString);
if(e < eNOERROR) /* error 처리 */
```

2. 유틸리티

2.1. OOSQL_CreateDB

Usage

OOSQL_CreateDB *database_name* [*volume_name*] [-dbdir *database_directory*] {[-device *device_path numberOfPages*] [-device *device_path numberOfPages*] ...} [-extentSize *extent_size*] [-extentFillFactor *extent_fill_factor*] [-segmentSize *segmentSize*]

Description

OOSQL은 저장매체를 관리하는 단위로 데이터베이스를 사용한다. 데이터베이스는 하나 이상의 볼륨으로 구성되며, 각각의 볼륨 역시 하나 이상의 디바이스로 구성된다. 데이타베이스를 구축하기 위해서는 우선 데이타베이스를 생성하여야 한다. OOSQL_CreateDB는 데이타베이스를 생성해주는 유틸리티이다.

databse_name 은 사용자가 임의로 지정하는 데이타베이스의 이름으로, OOSQL의 데이타베이스 관련 인터페이스의 인수로 사용된다. *volume_name* 은 데이타베이스의 디폴트 볼륨의 이름이다. *volume_name* 은 생략될 경우 *database_name* 과 같은 이름을 갖는다. *database_directory* 는 생성될 데이타베이스가 위치할 운영체제 상의 디렉토리를 지정하는데, 생략할 경우 $ODYS_OODB 환경 변수가 지정하는 디렉토리로 설정된다.

하나의 볼륨은 여러 개의 디바이스 화일로 구성되며, 이들 디바이스들은 –device 옵션으로 지정한다. *device_path* 는 디바이스의 패스(path)이고, *numberOfPages* 는 디바이스가 몇 개의 디스크 페이지로 구성되어 있는지를 지정하는 값이다. 디바이스가 지정되지 않을 경우 volume_name 과 같은 이름의 디바이스가 16000 페이지의 크기(페이지 크기가 4Kbyte 인 경우 볼륨의 크기는 64Mbyte)로 데이타베이스 디렉토리에 생성된다. *extent_size* 는 데이타베이스로 사용하는 디스크의 크기에 따라 다른 값을 갖는다. *extent_fill_factor* 익스텐트의 몇 퍼센트 정도를 비워둘 것인지를 지정하는 값으로, 한꺼번에 대량의 데이타를 저장할 때에 익스텐트의 일부를 비워둠으로써 이후 연산의 효율성을 증대시키는데 사용된다. OOSQL 은 대용량 객체와 일반 객체를 볼륨내의 서로 다른 영역에 생성한다. 각각의 영역은 segment_size 페이지 단위로 할당되므로, segment_size 값이 클수록 객체간의 클러스터링 효과가 증대된다. 하지만, segment_size 값이 너무 큰 경우에는 필요없는 저장 공간의 낭비가 생길 수 있

다. 일반적으로 볼륨의 크기가 2GB 정도이면, 500MB 정도의 segment_size 를 지정하면 된다. Segment_size 를 지정하지 않을 경우 기본적으로 초기 디바이스 크기의 1/4 로 지정된다.

OOSQL_CreateDB는 데이타베이스 디렉토리를 생성하고 지정한 디바이스들에 대해 초기화를 수행한다. 또, $ODYS_OODB/OOSQL_SysDirFile 화일에 데이타베이스 이름, 디렉토리 경로, 볼륨 이름, 디바이스 패스등을 기록한다.

볼륨을 구성하는 디바이스 선택 시 주의 사항

볼륨을 구성하는 디바이스의 종류에 따라 데이타베이스의 성능이 달라진다. 그러므로, 볼륨의 종류에 따라 그에 알맞은 디바이스를 선택하여야, 최상의 성능을 유지할 수 있다. 데이타 볼륨, 임시 볼륨, 그리고 로그 볼륨을 구성하는 디바이스는 UNIX의 경우 raw device를 사용해야 하고, NT의 경우는 file system을 사용해야 한다. NT에서는 file system을 raw device처럼 다루는 기능이 있어, raw device를 사용하지 않아도 된다. 그러나, UNIX에서는 이러한 기능이 없기 때문에 반드시 raw device를 사용해야 한다. raw device를 사용하지 않을 경우에는, OS에서 수행하는 불필요한 버퍼링으로 인해 처리 시간과 메모리가 낭비되는 문제가 발생한다. 특히 버퍼링을 위해 사용되는 메모리의 크기는 디바이스의 크기(수백MB ~ 수GB) 만큼 커질 수도 있기 때문에 시스템에 과부하를 줄 수도 있다.

Example

다음은 testdb라는 이름의 데이터베이스를 생성하는 예이다. 볼륨 이름은 생략되었으므로 데이터베이스의 이름 testdb가 볼륨의 이름이 된다. 디바이스의 크기는 640,000KB (160,000 * 4KB) 이다.

- OOSQL_CreateDB testdb –extentSize 32 –segmentSize 6000 –device $ODYS_OODB/testdb/testdb 160000

2.2. OOSQL_DestroyDB

Usage

OOSQL_DestroyDB *database_name*

Description

데이타베이스 전체를 삭제하고자 할 경우 OOSQL_DestroyDB 유틸리티를 사용한다.

Example

다음은 testdb를 삭제하는 예이다.

- OOSQL_DestroyDB testdb

2.3. OOSQL_InitDB

Usage

OOSQL_InitDB *database_name*

Description

사용 중인 데이타베이스를 초기화하고자 할 경우, OOSQL_InitDB를 사용한다. OOSQL_InitDB는 사용자가 지정한 데이타베이스 내의 모든 객체 및 클래스를 삭제하고 데이타베이스 화일을 새로 생성한다. OOSQL_InitDB의 구문은 다음과 같다

Example

다음은 testdb를 초기화하는 예이다.

- OOSQL_InitDB testdb

2.4. OOSQL_AddVolume

Usage

OOSQL_AddVolume *database_name volume_name* {[-device *device_path numberOfPages*] [-device *device_path numberOfPages*] ...} [-extentSize *extent_size*] [-extentFillFactor *extent_fill_factor*] [-segmentSize *segmentSize*]

Description

OOSQL_AddVolume는 데이타베이스를 생성해주는 유틸리티이다.

databse_name 은 사용자가 임의로 지정하는 데이타베이스의 이름으로, OOSQL 의 데이타베이스 관련 인터페이스의 인수로 사용된다. *volume_name* 은 데이타베이스의 디폴트 볼륨의 이름이다. *volume_name* 은 생략될 경우 *database_name* 과 같은 이름을 갖는다. *database_directory* 는 생성될 데이타베이스가 위치할 운영체제 상의 디렉토리를 지정하는데, 생략할 경우 $ODYS_OODB 환경 변수가 지정하는 디렉토리로 설정된다.

하나의 볼륨은 여러 개의 디바이스 화일로 구성되며, 이들 디바이스들은 –device 옵션으로 지정한다. *device_path* 는 디바이스의 패스(path)이고, *numberOfPages* 는 디바이스가 몇 개의 디스크 페이지로 구성되어 있는지를 지정하는 값이다. 디바이스가 지정되지 않을 경우 volume_name 과 같은 이름의

디바이스가 16000 페이지의 크기로 데이타베이스 디렉토리에 생성된다. *extent_size* 는 데이타베이스로 사용하는 디스크의 크기에 따라 다른 값을 갖는다. *extent_fill_factor* 익스텐트의 몇 퍼센트 정도를 비워둘 것인지를 지정하는 값으로, 한꺼번에 대량의 데이타를 저장할 때에 익스텐트의 일부를 비워둠으로써 이후 연산의 효율성을 증대시키는데 사용된다. OOSQL 은 대용량 객체와 일반 객체를 볼륨내의 서로 다른 영역에 생성한다. 각각의 영역은 segment_size 페이지 단위로 할당되므로, segment_size 값이 클수록 객체간의 클러스터링 효과가 증대된다. 하지만, segment_size 값이 너무 큰 경우에는 필요없는 저장 공간의 낭비가 생길 수 있다. 일반적으로 볼륨의 크기가 2GB 정도이면, 500MB 정도의 segment_size 를 지정하면 된다. Segment_size 를 지정하지 않을 경우 기본적으로 초기 디바이스 크기의 1/4 로 지정된다.

OOSQL_AddVolume는 지정한 디바이스들에 대해 초기화를 수행한다. 또, $ODYS_OODB/OOSQL_SysDirFile 화일에 볼륨 이름, 디바이스 패스등을 기록한다.

Example

다음은 testdb 데이터베이스에 testdb2라는 볼륨을 추가하는 예이다. 디바이스의 크기는 640,000KB (160,000 * 4KB) 이다.

- OOSQL_AddVolume testdb testdb2 –extentSize 32 –segmentSize 6000 –device $ODYS_OODB/testdb/testdb2 160000

2.5. OOSQL_DropVolume

Usage

OOSQL_DropVolume *database_name volume_name*

Description

데이터베이스에서 하나의 볼륨을 삭제하고자 할 경우 OOSQL_DropVolume 유틸리티를 사용한다.

Example

다음은 testdb 데이터베이스에서 testdb2 볼륨을 삭제하는 예이다.

- OOSQL_DropVolume testdb testdb2

2.6. OOSQL_InitVolume

Usage

OOSQL_InitVolume *database_name volume_name* {[-device *device_path*

numberOfPages] [-device *device_path numberOfPages*] ...} [-extentSize *extent_size*] [-extentFillFactor *extent_fill_factor*] [-segmentSize *segmentSize*]

Description

OOSQL_InitVolume는 사용중인 볼륨을 초기화하는 유틸리티이다.

OOSQL_InitVolume은 볼륨을 초기화하면서 볼륨을 구성하는 디바이스들을 새로 지정할 수 있으며, extentSize, extentFillFactor와 segmentSize도 변경할 수 있다.

Example

다음은 testdb 데이터베이스의 testdb2라는 볼륨을 초기화하는 예이다.

- OOSQL_InitVolume testdb testdb2

2.7. OOSQL_AddDevice

Usage

OOSQL_AddDevice *database_name volume_name* -device *device_path numberOfPages* {[-device *device_path numberOfPages*] ...}

Description

OOSQL_AddDevice는 볼륨에 새로운 디바이스를 추가하는 유틸리티이다.

Example

다음은 testdb 데이터베이스의 testdb2라는 볼륨에 testdb2-1이라는 디바이스를 추가하는 예이다.

- OOSQL_AddDevice testdb testdb2 –device $ODYS_OODB/testdb/testdb2-1 16000

2.8. OOSQL_FormatLogVolume

Usage

OOSQL_FormatLogVolume *volume_name volume_id* -device *device_path numberOfPages* [-device *device_path numberOfPages*] {[-device *device_path numberOfPages*] ...} [-extentSize *extent_size*] [-extentFillFactor *extent_fill_factor*]

Description

OOSQL_FormatLogVolume은 로그를 위한 볼륨을 초기화하는 유틸리티이다. 로그는 데이터베이스의 연산을 기록하는 것으로 데이터베이스 시스템이 외부적 요

인등에 의해 비정상적인 종료를 하였을 경우, 데이터베이스의 내용을 원래대로 복귀시켜주는 역할을 한다. 트랜잭션의 철회(Roll Back)연산이나 파손회복 기능을 사용하기 위해서는 반드시 로그 볼륨을 생성해야 한다. 로그 볼륨이 없거나 지정되지 않은 경우에는 OOSQL_TransAbort 함수를 통해 트랜잭션 철회나 프로그램의 비정상적인 종료로 인한 데이터베이스 파손으로부터의 회복을 수행할 수 없다.

로그 볼륨은 하나 이상의 디바이스로 구성되고, *segment_size*의 지정이 필요 없다. *volume_id*는 사용자가 임의로 지정하는 볼륨의 식별자이고, 일반 볼륨들과 식별자가 겹치지 않도록 1000보다 작은 값으로 하는 것이 좋다.

로그 볼륨을 구성하는 디바이스는 UNIX의 경우 raw device를 사용해야 하고, NT의 경우는 file system을 사용해야 한다. NT에서는 file system을 raw device처럼 다루는 기능이 있어, raw device를 사용하지 않아도 된다. 그러나, UNIX에서는 이러한 기능이 없기 때문에 반드시 raw device를 사용해야 한다. raw device를 사용하지 않을 경우에는, OS에서 수행하는 불필요한 버퍼링으로 인해 처리 시간과 메모리가 낭비되는 문제가 발생한다.

OOSQL 응용프로그램이 생성된 로그를 사용하기 위해서는 $COSMOS_LOG_VOLUME이라는 환경변수를 로그 볼륨이 있는 위치를 가리키도록 설정해야 한다. $COSMOS_LOG_VOLUME에 두 개 이상의 디바이스를 설정할 때는 디바이스간의 구별자로 ;을 사용한다.

Example

다음은 testdb.log라는 이름으로 로그 볼륨을 초기화하는 예이다.

- OOSQL_FormatLogVolume testdb.log 200 –device /dev/rdsk/c0t1d0s3 500000 (UNIX csh)
- OOSQL_FormatLogVolume testdb.log 200 –device C:\log\testdb.log 500000 (Windows)

다음은 성생된 로그를 사용하도록 환경 변수를 설정하는 예이다.

setenv COSMOS_LOG_VOLUME /dev/rdsk/c0t1d0s3 (UNIX csh)

set COSMOS_LOG_VOLUME=C:₩log₩testdb.log (Windows)

2.9. OOSQL_FormatCoherencyVolume

Usage

OOSQL_FormatCoherencyVolume *volume_name volume_id* -device *device_path*

Description

OOSQL_FormatCoherencyVolume은 다중 서버 환경을 위한 coherency 볼륨을 초기화하는 유틸리티이다. 다중 서버 환경에서 한 프로세스 내의 버퍼 내용이 변경되면 이러한 변경 내용이 coherency 볼륨에 기록되고 다른 프로세스는 이

를 참조하여 버퍼의 내용을 일치시킨다.

Coherency 볼륨은 UNIX, NT 모두 파일 시스템에 생성해야 한다. *volume_id*는 사용자가 임의로 지정하는 볼륨의 식별자이고, 일반 볼륨들과 식별자가 겹치지 않도록 1000보다 작은 값으로 하는 것이 좋다. *device_path* 에는 반드시 OS 파일 시스템의 절대 패스를 적어주어야 한다.

OOSQL 응용프로그램이 생성된 coherency 볼륨을 사용하기 위해서는 $COSMOS_COHERENCY_VOLUME이라는 환경변수를 coherency 볼륨이 있는 위치를 가리키도록 설정해야 한다. 다중 서버 환경에서 자료의 갱신, 삽입, 삭제가 발생하면 반드시 coherency 볼륨을 사용해야 한다.

Example

다음은 coherency 라는 이름으로 coherency 볼륨을 초기화하는 예이다.

- OOSQL_FormatCoherencyVolume coherency 200 –device /temp/coherency.vol (UNIX csh)
- OOSQL_FormatCoherencyVolume coherency 200 –device C:\temp\coherency.vol (Windows)

다음은 성생된 coherency 볼륨을 사용하도록 환경 변수를 설정하는 예이다.

setenv COSMOS_COHERENCY_VOLUME /temp/coherency.vol (UNIX csh)

set COSMOS_COHERENCY_VOLUME=C:₩temp₩coherency.vol (Windows)

2.10. OOSQL_MakeTextIndex

Usage

OOSQL_MakeTextIndex *database_name volume_name class_name attribute_name1* [*attribute_name2* ...] *data_file_name* [*loaddb*]

Description

OOSQL_MakeTextIndex는 클래스의 한 텍스트 타입 속성에 배치 작업으로 텍스트 인덱스를 만든다.

*database_name*은 텍스트 인덱스를 만들 데이터베이스의 이름이고, *volume_name*은 텍스트 인덱스를 만들 볼륨의 이름이다. *class_name*은 텍스트 인덱스를 만들 클래스의 이름이다. *attribute_name1* ... 은 텍스트 인덱스를 구축하고자 하는 속성들의 이름이고, *data_file_name*은 텍스트 인덱스를 만들 문서의 내용이 저장되어 있는 파일이다. *data_file_name* 파일은 OOSQL_LoadDB의 입력파일 형식을 따른다. *loaddb*는 OOSQL_MakeTextIndex에서 OOSQL_LoadDB의 호출 여부를 지정한다. 실제 데이터의 로드는 하지 않고, 키워드 추출과 인덱스 구축만이 필요한 경우에 *loaddb*를 생략하고 사용한다.

Example

testdb 데이터베이스의 testdb 볼륨의 Newspaper 라는 클래스의 title 이라는 이름의 텍스트 속성에 대해서 test.in 이라는 데이터 파일로부터 텍스트 인덱스를 생성하려면 다음과 같은 명령을 수행한다.

```
> OOSQL_MakeTextIndex testdb testdb Newspaper title test.in loaddb
```

OOSQL_MakeTextIndex 는 몇 개의 작은 작업들로 구성된다. 제 2.10.1 절부터 2.10.6 절까지는 각각의 작은 작업들에 대한 설명이다.

2.10.1. OOSQL_ExtractKeyword

Usage

OOSQL_ExtractKeyword *database_name volume_name class_name attribute_name data_file_name*

Description

OOSQL_ExtractKeyword는 data_file_name이 가리키는 화일에서 텍스트 내용을 읽어 색인에 사용될 키워드들을 추출하여 화일로 저장하는 작업을 수행한다.

*database_name*은 텍스트 인덱스를 만들 데이터베이스의 이름이고, *volume_name*은 텍스트 인덱스를 만들 볼륨의 이름이다. *class_name*은 텍스트 인덱스를 만들 클래스의 이름이다. *attribute_name*은 텍스트 타입 속성 이름이고, *data_file_name*은 텍스트 인덱스를 만들 문서의 내용이 저장되어 있는 파일이다. *data_file_name* 파일은 OOSQL_LoadDB의 입력파일 형식을 따른다.

OOSQL_ExtractKeyword가 추출된 키워드를 저장하는 화일 이름은 다음과 같다.

$ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_Posting

Example

test.in 파일에서 testdb 데이터베이스의 testdb 볼륨의 Newspaper 클래스의 title 속성에 대한 키워드 추출을 하고 싶다면 다음과 같은 명령을 실행한다.

```
> OOSQL_ExtractKeyword testdb testdb Newspaper title test.in
```

2.10.2. OOSQL_SortPosting

Usage

OOSQL_SortPosting *input_file output_file*

Description

OOSQL_SortPosting은 OOSQL_ExtractKeyword에 의해 생성된 화일을 정렬하기 위해 사용되는 명령이다.

OOSQL_SortPosting은 input_file을 정렬하여 output_file로 쓴다. input_file과 output_file은 모두 posting file의 구조를 가진다. OOSQL_MakeTextIndex는 OOSQL_SortPosting의 input_file 이름을

$ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_Posting

로 지정하고, output_file은 다음과 같이 지정한다.

$ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_SortedPosting

2.10.3. OOSQL_LoadDB

Usage

OOSQL_LoadDB [-smallupdate | -largeupdate] *database_name* [*volume_name*] [-temporary *database_name* [*volume_name*] *data_file_name*

Description

OOSQL_LoadDB는 데이터 파일의 내용을 데이터베이스에 저장하고, 새로 생성된 객체들의 OID를 $ODYS_TEMP_PATH 디렉토리 아래의 TEXT_<class_name>_OID 파일에 저장한다.

OOSQL_LoadDB에 대한 자세한 설명은 2.11절에서 한다.

2.10.4. OOSQL_MapPosting

Usage

OOSQL_MapPosting *database_name volume_name class_name attribute_name posting_file_name new_posting_file_name oid_file_name*

Description

OOSQL_MapPosting은 OOSQL_LoadDB에서 만들어진 문서 번호/OID 테이블을 이용하여 OOSQL_SortPosting에 의해 만들어진 화일을 변환하는 작업을 수행한다.

*posting_file_name*은 변환하고자 하는 포스팅 파일의 이름이고, *new_posting_file_name*은 변환한 결과를 저장할 새로운 포스팅 파일의 이름이다. *oid_file_name*은 변환에 사용할 문서 번호/OID 테이블을 저장하고 있는 파일의 이름이다.

OOSQL_MapPosting은 매핑 테이블 화일을 환경 변수 ODYS_TEMP_PATH에 지정된

디렉토리에서 찾고, 변환된 화일을 같은 디렉토리에 생성한다. OOSQL_MakeTextIndex는 posting_file_name을

$ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_SortedPosting

으로 지정하고, new_posting_file_name을

$ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_SortedPosting_Mapped

으로 지정한다. new_posting_file_name 파일의 이름을

$ODYS_TEMP_PATH/TEXT_<class_name>_<attribute_name>_SortedPosting

과 같이 바꾼후에 다시 OOSQL_BuildTextIndex의 단계부터 작업을 계속한다.

2.10.5. OOSQL_BuildTextIndex

Usage

OOSQL_BuildTextIndex *database_name volume_name class_name attribute_name*

Description

OOSQL_BuildTextIndex는 OOSQL_MapPosting에 의해 생성된 화일들로부터 텍스트 인덱스를 생성한다.

*database_name은 데이터가 저장되어 있는 데이터베이스의 이름이고, volume_name*은 데이타가 저장되어 있는 볼륨의 이름이다. *class_name*은 텍스트 데이터를 가지고 있는 클래스 이름이다. *attribute_ name*은 텍스트 속성의 이름이다.

OOSQL_BuildTextIndex는 인덱스를 구축하고자 하는 화일을 환경 변수 ODYS_TEMP_PATH에 지정된 디렉토리에서 찾는다.

Example

/OOSQL/test.vol이라는 볼륨의 Newspaper 클래스의 title 속성으로부터 얻어진 화일인 TEXT_Newspaper_title_SortedPosting 화일에 대해서 인덱스를 구축하고 싶다면, 다음과 같은 명령을 실행한다.

```
> OOSQL_BuildTextIndex testdb testdb Newspaper title
```

2.10.6. OOSQL_UpdateTextDescriptor

Usage

OOSQL_UpdateTextDescriptor *database_name* [*volume_name*] *class_name*

Description

텍스트 타입의 애트리뷰트를 가지는 객체에는 텍스트 타입 애트리뷰트에 대한 정보인 텍스트 descriptor가 존재한다. OOSQL_UpdateTextDescriptor는 주어진 class에 존재하는 모든 객체들의 텍스트 descriptor에서 텍스트 인덱스가 구축되었는지를 나타내는 필드를 업데이트한다.

*database_name*은 데이터가 저장되어 있는 데이터베이스의 이름이고, *volume_name*은 데이타가 저장되어 있는 볼륨의 이름이다. *class_name*은 텍스트 데이터를 가지고 있는 클래스 이름이다.

Example

/OOSQL/test.vol이라는 볼륨의 Newspaper 클래스에 구축된 인덱스들을 업데이트하고 싶다면, 다음과 같은 명령을 실행한다.

```
> OOSQL_UpdateTextDescriptor testdb testdb Newspaper
```

2.11. OOSQL_LoadDB

Usage

OOSQL_LoadDB *database_name volume_name input_file_name*

Description

입력 파일에 있는 데이타를 데이터베이스에 삽입한다. *Database_name은 데이터를 삽입할 데이터베이스의 이름이고, volume_name*은 데이터가 삽입될 볼륨의 이름이다. *input_file_name*은 데이터를 가지고 있는 입력 파일의 이름이다. 이 파일은 입력 파일 포맷에 맞추어 저장된 텍스트 파일이다. 입력 파일 포맷은 다음 페이지에 자세히 나타나 있다.

OOSQL_LoadDB는 새로 생성된 객체들의 문서 번호/OID를 $ODYS_TEMP_PATH/TEXT_<class_name>_OID 파일에 기록한다.

OOSQL_LoadDB이 사용하는 입력 파일 포맷

- Comments

2 개의 hypen 으로 시작

예: -- This is a comment

- Command Lines

Syntax

%class class_name (attr_name [{attr_name}…])

Description

클래스에 정의되어 있는 속성의 이름을 명시한다. %class 는 뒤에서 설명할 모든 data line 의 앞에 나온다. %class 에 명시한 속성의 순서대로 데이터를 적어주어야 한다.

Example

%class person (name age)

person 이라는 class 에 name 과 age 라는 속성을 사용한다고 명시한다. 속성의 타입은 schema 에 저장되어 있는 정보를 사용한다.

- Data lines

Syntax

각 속성에 해당하는 데이터를 %class 에 명시한 순서대로 나열한다.
각각의 데이터 타입을 명시하는 방법은 다음 장에 명시되어 있다.

Description

class 의 속성에 채울 데이터를 나열한다.

Example

%class person (name age)

‘smith’	31
‘newman’	33
‘jones’	31
‘underwood’	47

■ 각 데이터 타입을 명시하는 방법

Data type	예
SHORT	2048
INTEGER	123456789
FLOAT	123.456

DOUBLE	1.45693e+20
STRING	‘this is a string’
VARSTRING	‘this is a char’
TEXT	“this is a text”
TIME	‘10:20:00’
DATE	‘7/4/1776’
TIMESTAMP	’10:20:00 7/4/1776’
SET	{ 1 2 3 }
BAG	{ 1 1 2 2 3 }
LIST (SEQUENCE)	{ ‘one’ ‘two’ ‘three’ }

- Object Reference

Syntax

@class_ref|instance_no

Description

객체간의 reference 를 제공한다. class_ref 에는 class_name 이나 class_id 가 올 수 있다. class_ref 는 @ 기호 다음에 바로 나와야 한다. | 기호가 class_ref 와 instance_no 를 구분한다. 이 사이에 공백은 허용되지 않는다.

Example 1

@person|28

person 클래스에서 instance 번호가 28 인 객체를 참조한다.

Example 2

%class person (name age)

1: ‘steve’	32
2: ‘joe’	33
3: ‘mary’	45
‘sarah’	23

instance 번호를 지정해주기 위해 양수 하나를 배당하고 뒤에 바로 :을 붙여준

다. 하나의 class 내에서 이 숫자는 유일해야 한다.

Example 3

%class automobile (make owner)

‘Ford’ @person|1

‘Mazda’ @person|3

‘Jeep’ @person|2

위의 예는 ‘Ford’의 owner 로 person class 의 1 번 instance 를 참조하고 ‘Mazda’의 owner 로 person class 의 3 번 instance 를 참조하고 ‘Jeep’의 owner 로 person class 의 2 번 instance 를 참조하라는 뜻이다. 실제 데이터베이스에는 참조하는 instnace 의 object id 가 저장된다.

2.12. OOSQL_CheckDataSyntax

Usage

OOSQL_CheckDataSyntax [-verbose] database_name volume_name input_file_name

Description

OOSQL_LoadDB 유틸리티의 입력 파일이 문법에 맞게 되어 있는지 검사하고 에러가 있는 경우 에러의 원인과 해결 방법을 표시해준다. *database_name*은 데이터를 삽입할 데이터베이스의 이름이고, *volume_name*은 데이터가 삽입될 볼륨의 이름이다. *input_file_name*은 데이터를 가지고 있는 입력 파일의 이름이다. –verbose 옵션을 사용하면 입력 파일에서 검사를 통과한 데이터를 *input_file_name*.tmp 라는 이름의 파일로 기록해준다. 에러가 발생 했을 때 이 파일을 조사하면 본 유틸리티가 어디까지 데이터를 검사하였는지 보다 쉽게 알 수 있다.

실행 예

```
% OOSQL_CheckDataSyntax sample sample source.txt
70번째 라인에서 에러가 발견되었습니다.
에러 원인:
큰 따옴표가 큰 따옴표 사이에 포함되어 있습니다.
=>
"아"주대학교 경영대학 홈페이지입니다."
해결 방법:
```


큰 따옴표 앞에 ₩를 붙여 주십시오.

367번째 라인에서 에러가 발견되었습니다.
에러 원인:
char, varchar, text 타입의 data line이 불완전하게 종료되었습니다.
=>
기초과목 전공과목 기초과목 과목명 구분
해결 방법:
(1) data가 한 line인 경우에는 ' 혹은 " 을 마지막에 추가하세요.
(2) data가 여러 line에 걸쳐있는 경우에는 ₩n 혹은 ₩ 을 마지막에 추가하세요.

총 123개의 객체에 대한 데이터를 검사하였습니다.
검사 결과 문법 에러가 2개 발견되었습니다.

2.13. OOSQL Python 유틸리티

Usage

OOSQL_LoaderExtract.py *database_name volume_name* [-temporary *database_name volume_name*] *data_file_name*

OOSQL_LoaderBuild.py *database_name volume_name* [-temporary *database_name volume_name*] *data_file_name*

OOSQL_Loader.py *database_name volume_name* [-temporary *database_name volume_name*] *data_file_name*

Description

OOSQL_MakeTextIndex 유틸리티는 이를 구성하는 서브 유틸리티를 수행하면서 각각의 서브 유틸리티마다 별도의 트랜잭션을 관리한다. 즉, 각각의 서브 유틸리티의 수행이 종료될 때마다 트랜잭션이 종료된다. 따라서, 데이터베이스 구축 과정의 일부가 수행되지 않은 중간 상태에서 트랜잭션이 종료되므로, 데이터베이스 구축 이전의 상태로 철회할 수 있는 방법이 없다.

이러한 단점을 해결하기 위해 Python 언어로 작성한 데이터베이스 구축용 유틸리티를 제공한다. 이들 유틸리티는 서브 유틸리티에서 수행하는 작업들을 하나의 트랜잭션에서 관리하여, 데이터베이스 구축이 완료된 시점에서 트랜잭션을 종료하도록 해준다.

- OOSQL_LoaderExtract.py: OOSQL_ExtractKeyword와 OOSQL_SortPosting 유

틸리티에서 수행하는 작업을 동일한 트랜잭션에 수행한다.

- OOSQL_LoaderBuild.py: OOSQL_LoadDB, OOSQL_MapPosting, OOSQL_BuildTextIndex 유틸리티에서 수행하는 작업을 동일한 트랜잭션에서 수행한다.

Python 유틸리티로 데이터베이스를 구축하기 위해서는,

1) OOSQL_LoaderExtract.py 유틸리티를 수행하고,

2) OOSQL_LoaderBuild.py 유틸리티를 수행하면 된다.

이 때, $ODYS_TEMP_PATH 환경 변수가 가리키는 디렉토리에 저장되는 과정 1)의 수행 결과를 복사해 놓으면, 다음 번에는 과정 1)을 매번 수행할 필요 없이 과정 2)부터 시작하여 데이터베이스를 구축할 수 있다. 그러므로, 과정 1)의 수행 결과를 반드시 복사해 놓길 바란다.

OOSQL_Loader.py는 OOSQL_LoaderExtract.py와 OOSQL_LoaderBuild.py를 연속적으로 수행하는 유틸리티이다. 위에서 언급했듯이 OOSQL_LoaderExtract.py 유티리티를 수행한 후 결과를 복사해 놓는 것이 바람직하므로, 가급적이면 OOSQL_LoaderExtract.py와 OOSQL_LoaderBuild.py를 연속적으로 수행하는 방식을 권장한다.

주의: 연속적으로 OOSQL_LoaderExtract.py와 OOSQL_LoaderBuild.py를 수행할 때, 두 유틸리티의 명령행에 명시한 데이터 파일을 가리키는 인수는 정확히 일치해야만 한다.

Python 유틸리티는 OOSQL_MakeTextIndex와 달리 자동적으로 데이터베이스 스키마를 읽어 테이블 및 컬럼 정보를 얻어오므로, 명령행에 테이블 이름과 컬럼 이름을 적어줄 필요가 없다.

Example

testdb 데이터베이스에 newspaper.txt 파일에 있는 데이터를 사용하여 데이터베이스 구축을 수행하려면, 다음과 같은 명령을 수행한다.

```
> OOSQL_LoaderExtract.py testdb testdb newspaper.txt
> OOSQL_LoaderBuild.py testdb testdb newspaper.txt
```

3. OOSQL 문법

3.1. OOSQL 전체 문법

다음은 OOSQL에서 사용가능한 질의문을 간략하게 정의한 문법이다.

```
statement ::=
        alter-table-statement
      | create-sequence-statement
      | create-table-statement
      | delete-statement
      | drop-sequence-statement
      | drop-table-statement
      | insert-statement
      | select-statement
      | update-statement
alter-table-statement ::=
      ALTER {TABLE | CLASS}   table-name
                                { ADD (column-identifier data-type) |
                                  DROP COLUMN column-identifier |
                                  DROP (column-identifier [,column-identifier]…) }
                              [, { ADD (column-identifier data-type) |
                                   DROP COLUMN column-identifier |
                                   DROP (column-identifier [,column-identifier]…) } ]…
create-sequence-statement ::=
      CRATE SEQUENCE sequence-identifier [START WITH start-value]
create-table-statement ::=
      CREATE [TEMPORARY] TABLE base-table-name [UNDER parent-table-name-list]
      (column-identifier data-type [,column-identifier data-type]...) |
      CREATE [TEMPORARY] CLASS base-class-name
      [AS SUBCLASS OF parent-class-name-list]
      (column-identifier data-type [,column-identifier data-type]...)
drop-sequence-statement ::=
      DROP SEQUENCE sequence-identifier
drop-table-statement ::=
      DROP { TABLE | CLASS} base-table-name
create-index-statement ::=
      CREATE [UNIQUE] [CLUSTER] INDEX index-name ON base-table-name
```

(column-identifier [,column-identifier]...)

drop-index-statement ::=

DROP INDEX index-name

delete-statement ::=

DELETE FROM table-name [WHERE search-condition] |

DELETE FROM OBJECT oid-string

insert-statement ::=

INSERT INTO table-name [(column-identifier [, column-identifier]...)]

VALUES (insert-value[, insert-value]...) |

INSERT INTO table-name [(column-identifier [, column-identifier]...)]

select-statement

update-statement ::=

UPDATE table-name

SET column-identifier = {expression | NULL }

[, column-identifier = {expression | NULL}]...

[WHERE search-condition] |

UPDATE OBJECT oid-string

SET column-identifier = {expression | NULL }

[, column-identifier = {expression | NULL}]...

select-statement ::=

SELECT [ALL | DISTINCT] select-list

FROM table-reference-list

[WHERE search-condition]

[group-by-clause]

[having-clause]

[order-by-clause]

[limit-clause] |

SELECT select-list

FROM OBJECT oid-string

3.2. Create Table 질의

Syntax

create-table-statement ::=

CREATE [TEMPORARY] TABLE base-table-name [UNDER parent-table-name-list]

(column-identifier data-type [,column-identifier data-type]...) |

CREATE [TEMPORARY] CLASS base-class-name

[AS SUBCLASS OF parent-class-name-list]

(column-identifier data-type [,column-identifier data-type]...)

Description

주어진 속성들을 가지는 새로운 테이블 혹은 클래스를 생성한다. 새로운 테이블 혹은 클래스는 기존의 테이블 혹은 클래스의 정의로 부터 상속 받아 정의될 수 있으며, 이 경우 parent-table-name-list혹은 parent-class-name-list에 나열하여 지정할 수 있다.

테이블에 임시로 데이타를 저장하고자 하는 경우에는 임시 테이블을 사용할 수 있다. 임시 테이블을 사용하고자 할 때는 정의시 temporary를 명시해준다. 임시 테이블은 트랜잭션 안에서만 존재하는 테이블로 트랜잭션이 끝나면 그 정의와 내용이 모두 없어진다. 임시 테이블은 일반 테이블과 달리 그 접근속도가 빠르기 때문에 질의 결과를 임시로 보관하는데 유용하다.

새로운 테이블 혹은 클래스를 구성하는 속성들은 column-identifier data-type의 나열로 지정한다. column-identifier에는 속성이 이름이 오며 data-type에는 속성의 타입이 온다. OOSQL에서 사용가능한 데이타 타입은 다음과 같다. OID 타입에서 클래스 이름은 생략 가능하며, 자세한 사용 예는 3.13절 Path Expressions을 참고하기 바란다.

type 이름	예제
CHAR(n)	employee_name char(10)
VARCHAR(n)	company_name varchar(20)
SMALLINT	age smallint
INTEGER	value integer
FLOAT	radius float
REAL	width real
DOUBLE PRECISION	volume double precision
OID[(class_name)]	object_id oid(employee)
TEXT	abstract text
DATE	date_released date
TIME	time_rented time

TIMESTAMP	time_row_accessed timestamp

Example

```
create table Employee (
      id integer, name char(20), age integer, fee integer)
```

3.3. Alter Table 질의

Syntax

alter-table-statement ::=

```
ALTER {TABLE | CLASS}  table-name
                        { ADD (column-identifier data-type) |
                          DROP COLUMN column-identifier |
                          DROP (column-identifier [,column-identifier]…) }
                    [, { ADD (column-identifier data-type) |
                          DROP COLUMN column-identifier |
                          DROP (column-identifier [,column-identifier]…) } ]…
```

Description

테이블 혹은 클래스의 정의를 변경한다. 상속 관계에서 말단 테이블 혹은 클래스만 변경이 가능하다.

새로운 컬럼을 추가할 때에는 ADD 절을 사용하며, 추가되는 컬럼의 속성들은 테이블이나 클래스를 생성할 때와 마찬가지로 column-identifier data-type의 리스트로 지정한다.

기존의 컬럼을 삭제할 때에는 DROP 절을 사용하며, 제거할 컬럼이 하나일 때에는 DROP COLUMN 절을 사용하여 지정하고 두 개 이상일 때에는 DROP 절 다음에 column-identifier의 리스트로 지정한다.

Example

```
alter table Employee add (address varchar(100), department char(20))
alter table Employee drop column fee
alter table Employee drop (age, fee)
```

3.4. Drop Table 질의

Syntax

drop-table-statement ::=

DROP TABLE base-table-name |

DROP CLASS base-class-name

Description

테이블의 정의를 삭제한다.

Example

```
drop table Employee
```

3.5. Create Index 질의

Syntax

create-index-statement ::=

CREATE [UNIQUE] [CLUSTER] [MLGF] INDEX index-name ON base-table-name (column-identifier [,column-identifier]...)

Description

테이블에 B^+-Tree 인덱스 또는 MLGF 인덱스를 생성한다. 인덱스를 구성하는 키값이 테이블내에서 유일한 경우 UNIQUE를 사용하며 인덱스에서의 키 순서대로 테이블의 순서를 지정하고자 할 경우 CLUSTER를 사용한다. MLGF 인덱스를 생성하고자 할 경우에는 키워드 MLGF를 사용한다. 키워드 MLGF가 생략되면 기본적으로 B^+-Tree 인덱스를 생성한다. 인덱스가 생성될 위치는 테이블의 이름과 속성의 이름들로 지정된다.

Example

```
create cluster index employee_id_index on Employee(id)
```

3.6. Drop Index 질의

Syntax

drop-index-statement ::=

DROP INDEX index-name

Description

인덱스 정의를 삭제한다.

Example

```
drop index employee_id_index
```

3.7. Create Sequence 질의

Syntax

create-sequence-statement ::=

CRATE SEQUENCE sequence-identifier [START WITH start-value]

Description

시퀀스를 생성한다. 시퀀스는 사용자가 고유한 정수를 생성할 수 있는 데이터베이스 객체이다. 시퀀스를 사용하여 자동으로 기본 키(Primary Key) 값을 생성할 수 있다. 시퀀스 번호는 테이블에 독립적으로 생성되며, 따라서 하나 또는 복수개의 테이블에서 동일한 시퀀스를 사용할 수 있다.

시퀀스가 생성되면 <*SEQUENCE_NAME*>.CURRVAL (시퀀스의 현재 값 반환) 또는 <*SEQUENCE_NAME*>.NEXTVAL (시퀀스를 증가하여 새로운 값 반환) 값을 사용하여 SQL 문에서 시퀀스 값을 엑세스할 수 있다.

START WITH 절로 시퀀스의 시작 값을 지정하지 않으면 시퀀스의 시작 값은 0으로 지정된다. 따라서 시퀀스 시작 값을 지정하지 않았을 경우 NEXTVAL에 대한 첫 번째 참조는 1을 반환하게 된다.

insert 질의와 update 질의 시 생성된 시퀀스를 이용하는 방법은 각각 3.10절과 3.11절을 참조하기 바란다.

Example

```
create sequence eseq

create sequence eseq start with 100

insert into Employee (id, name) values (eseq.nextval, '홍길동')

update Employee set id = eseq.nextval where name = '홍길동'
```

3.8. Drop Sequence 질의

Syntax

drop-sequence-statement ::=

DROP SEQUENCE sequence-identifier

Description

시퀀스의 정의를 삭제한다.

Example

```
drop sequence eseq
```

3.9. Select 질의

Syntax

select-statement ::=

SELECT [ALL | DISTINCT] select-list
FROM table-reference-list
[WHERE search-condition]
[group-by-clause]
[having-clause]
[order-by-clause]
[limit-clause] |
SELECT select-list
FROM OBJECT oid-string

Description

데이타베이스로 부터 주어진 조건을 만족하는 값들을 가져온다. 가져온 값들은 GOURP BY, HAVING, ORDER BY에 의해 그룹을 만들거나 순서를 정할 수 있다. SELECT절에는 데이타베이스로 부터 값을 가져오고자 하는 테이블의 속성 이름이나 AGGREGATE 함수들이 올 수 있다. FROM절에는 값을 가져오고자 하는 테이블의 이름이 온다. WHERE절에는 가져오고자 하는 데이타가 만족해야 할 조건이 온다. GROUP BY절에는 결과를 그룹으로 만드는 속성의 값이 오며 HAVING절에는 각각의 그룹이 만족해야 할 조건이 온다. ORDER BY절에는 질의 결과의 출력 순서를 정하는 속성의 이름이나 AGGREGATE 함수들이 올 수 있다. limit 절에는 질의 결과로 반환할 투플의 수가 오며, SQL 99 표준에는 없다.

주어진 OID를 가지는 객체로 부터 그 속성값을 읽고자 할때는 SELECT FROM OBJECT를 사용한다. oid-string은 값을 읽고자 하는 객체의 OID를 OOSQL_OIDToOIDString을 통해 만들어낸 문자열이다. 객체의 OID는 OOSQL_GetOID를 통해 얻어내거나 다음과 같은 질의문을 통해 얻어낼 수 있다.

```
select      e
from   Employee e
```

위 질의문을 통해 얻어낸 OID를 사용하여 다음과 같이 객체가 가지는 속성의 값을 구해 낼 수 있다.

```
select  *
from    object '00000560000A00000000000000000064'
```

OOSQL에서는 텍스트 정보 검색을 위해 MATCH라는 함수를 제공하다. MATCH는 키워드로 이루워진 정보 검색식을 만족하는 문서를 찾는데 사용되는 함수이다.

다음은 MATCH함수를 사용하여 content에 “컴퓨터”라는 키워드를 가지는 Newspaper를 찾아 그 content를 출력하는 질의문이다.

```
Select      content
From        Newspaper
Where       MATCH(content, “컴퓨터”) > 0
```

MATCH의 문법은 다음과 같다.

```
match-function ::=
        MATCH(column-identifier, ir-expression [, lable-id] [, scan-direction])
column-identifier ::= ID
ir-expression ::=
        keyword
        | ir-expression ir-binary-operator ir-expression
        | ir-expression ir-unary-operator INTEGER
        | (ir-expression)
keyword ::= “‘ ID “‘
ir-binary-operator ::= ‘&’ | ‘|’ | ‘-‘
ir-unary-operator ::= ‘>’ | ‘*’ | ‘:’
lable-id ::= INTEGER
scan-direction ::= FORWARD | BACKWARD
```

MATCH의 첫번째 인수인 column-identifier에는 검색하고자 하는 문서가 들어있는 속성의 이름이 들어가며 두번째 인수인 ir-expression에는 검색하고자 하는 키워드와 키워드간의 관계를 서술하는 연산자로 구성된 텍스트 정보식이 들어간다. 다음은 텍스트 정보 검색식에 들어 가는 연산자와 그 의미를 정리한 표이다.

연산자	의미
&	연산자의 양쪽에 나타난 두 키워드 모두 문서에 존재 해야한다는 의미로 두 키워드의 중요도의 최소값을 반환한다. 예) 멀티미디어 & 데이타베이스
|	연산자의 양쪽에 나타난 두 키워드중 하나 이상이 문서에 존재해야한다는 의미로 두 키워드의 중요도의 최대값을 반환한다.

예) 멀티미디어 | 데이타베이스

- 좌항의 키워드의 중요도를 우항의 키워드의 중요도로 뺀 값을 반환한다. 결과 값이 0이하이면 0을 반환한다.

  예) 멀티미디어 - 데이타베이스

* 키워드의 중요도에 상수값을 곱한 값을 반환한다.

  예) 멀티미디어 * 3

> 키워드의 중요도가 상수값이하인 경우 중요도를 0으로 반환한다.

  예) 멀티미디어 > 50

: 지정한 수만큼의 문서를 중요도가 높은 것 부터 반환한다.

  예) 멀티미디어 : 10

~n, ^n 양쪽에 나타난 키워드가 n으로 주어진 거리 이내에 있는지를 검사한다. ~는 양쪽에 나타난 키워드의 순서와 상관없이 거리를 검사하고, ^는 양쪽에 나타난 키워드의 순서대로 문서에 나타났는지와 거리를 함께 검사한다.

  연산자의 양쪽에는 키워드 이외에 and, or 연산자로 구성된 임의의 표현도 올 수 있다. 예를 들어 (A and B) ~2 C 와 같은 표현을 쓸 수 있다. 이 경우 (A ~2 C) and (B ~2 C)의 의미를 가진다.

  인수로 주어지는 키워드에는 star operator는 쓸 수 없다. 의미적으로 가능한 표현이지만, 내부 처리 시 많은 부담을 주기 때문에 처리를 하지 않는다.

  예) 멀티미디어 ^2 시스템

세번째 인수인 lable-id는 각각의 MATCH함수 마다 고유의 번호를 지정하는 것으로 MATCH함수의 값을 반환하는 WEIGHT함수의 인수로 사용된다. 다음 예는 두개의 MATCH 함수의 결과 값을 출력하는 질의예이다.

```
Select      WEIGHT(1), WEIGHT(2)
From        Newspaper
Where       MATCH(content, “컴퓨터”, 1) > 0 and
      MATCH(title, “인터넷”, 2) > 0
```

네번째 인수인 scan-direction은 검색 결과를 데이터베이스에 저장된 순서대

로 읽어올 것인지 혹은 역순으로 읽어올 것인지 지정한다. 순서대로 읽기 위해서는 FORWARD를 사용하고, 역순으로 읽기 위해서는 BACKWARD를 사용한다. 디폴트는 FORWARD 이다. 단, 하나의 질의식에서 여러 개의 MATCH() 함수가 사용될 경우, FORWARD와 BACKWARD를 혼용해서 사용할 수는 없다. 아래의 예는 역순으로 검색 결과를 읽어오는 예이다.

```
Select      Newspaper
From        Newspaper
Where       MATCH(content, "컴퓨터", BACKWARD) > 0
```

Example

```
select id, name from Employee where age > 20 order by age
```

3.10. Insert 질의

Syntax

insert-statement ::=

INSERT INTO table-name [(column-identifier [, column-identifier]...)]
VALUES (insert-value[, insert-value]...) |
INSERT INTO table-name [(column-identifier [, column-identifier]...)]
select-statement

Description

주어진 테이블에 주어진 값을 넣거나 주어진 질의 결과를 넣는다. table-name은 값을 넣고자 하는 테이블의 이름이고 column-identifier는 값이 들어갈 속성의 이름이다. insert-value는 실제 넣고자 하는 값이다. insert-value로서 특정 시퀀스의 CURRVAL (시퀀스의 현재 값 반환)과 NEXTVAL (시퀀스를 증가하여 새로운 값 반환) 값을 넣을 수도 있다.

넣고자 하는 값이 text 타입일 경우에는 반드시 'text'라는 keyword를 앞에 써줘야 한다. text를 넣으면서 바로 텍스트 관련 인덱스에 반영하고자 할 때에는 'text'혹은 'text immediate'를 적어 주며 나중에 텍스트 관련 인덱스에 반영하고자 할때에는 'text deferred'라고 적어준다. deferred로 들어간 텍스트에 대해 텍스트 관련 인덱스를 구출하고자 할 때에는 OOSQL_Text_MakeIndex를 사용한다.

넣고자 하는 값을 질의식에서 주지 않고 OOSQL_PutData에 의해 지정하고자 할 때에는 insert-value에 '?'를 사용한다.

Example

```
insert into Employee (id, name) values(10, '홍길동')
insert into Employee (id, name) values (eseq.nextval, '홍길동')
insert into Employee (id, name) values(?, ?)
insert into Newspaper (title) values(text 'OODBMS 개발 1')
insert into Newspaper (title) values(text deferred 'OODBMS 개발 2')
insert into Newspaper (title) values(text deferred ?)
```

3.11. Update 질의

Syntax

```
update-statement ::=
    UPDATE table-name
    SET column-identifier = {expression | NULL }
        [, column-identifier = {expression | NULL}]...
    [WHERE search-condition] |
    UPDATE OBJECT oid-string
    SET column-identifier = {expression | NULL }
        [, column-identifier = {expression | NULL}]...
```

Description

주어진 테이블에 주어진 조건을 만족하는 객체들의 값을 주어진 식에 의해 수정하거나 주어진 OID를 가지는 객체의 값을 주어진 식에 의해 수정한다. table-name은 수정하고자 하는 속성이 들어 있는 테이블의 이름이고 column-identifier는 수정하고자 하는 속성의 이름이다. expression은 수정될 값을 계산할 수 있는 수식이고 search-condition은 수정하고자 하는 객체가 만족해야 할 조건이다. expression으로서 특정 시퀀스의 CURRVAL (시퀀스의 현재 값 반환)과 NEXTVAL (시퀀스를 증가하여 새로운 값 반환) 값을 지정할 수도 있다. oid-string은 수정하고자 하는 객체의 OID를 OOSQL_OIDToOIDString을 통해 만들어낸 문자열이다.

수정하고자 하는 값을 질의식에서 주지 않고 OOSQL_PutData에 의해 지정하고자 할 때에는 expression에 '?'를 사용한다.

Example

```
update Employee set name = '홍길동' where id = 20
update Employee set id = eseq.nextval where name = '홍길동'
update Employee set name = ? where id = 20
```

```
update Newspaper set title = text 'OODBMS 개발' where id = 10
```

3.12. Delete 질의

Syntax

delete-statement ::=

DELETE FROM table-name [WHERE search-condition] |

DELETE FROM OBJECT oid-string

Description

주어진 조건을 만족하는 객체를 테이블로부터 삭제하거나 주어진 OID를 가지는 객체를 테이블로부터 삭제한다. table-name을 삭제하고자 하는 객체가 들이 있는 테이블의 이름이고 search-condition은 삭제하고자 하는 객체가 만족해야 할 조건이다. oid-string은 삭제하고자 하는 객체의 OID를 OOSQL_OIDToOIDString을 통해 만들어낸 문자열이다.

Example

```
delete from Employee where id = 20
```

3.13. Path Expressions

Description

OOSQL에서는 객체식별자(OID)로 연결된 복합 객체(complex object)의 각각의 객체를 항해할 수 있는 방법으로 path expression을 제공한다. Path expression이란 객체식별자 타입(OID type)을 가지는 속성 다음에 DOT(.) operator를 사용하여 그 속성이 가리키는 객체의 속성을 읽는 표현이다. Path expression은 따라가는 객체의 수만큼 그 길이가 길어진다.

다음은 "고용인의 배우자의 급여가 $10,000 이상인 배우자의 거주 도시 이름을 출력" 하는 의미를 가지는 질의문이다. 질의문에서 "e.spouse.address.city" 과 "e.spouse.salary"은 path expression 이다. Path expression "e.spouse.address.city"에서 e는 Employee 객체를 뜻한다. "e.spouse"는 Employee 객체의 배우자를 읽는 표현문이다. spouse 속성은 Employee 객체를 가리키는 객체식별자 타입으로 스키마에서는 OID(Employee)의 타입을 가진다. spouse 속성이 객체식별자 타입을 가지므로 path expression으로 spouse 속성이 가리키는 객체의 속성을 읽을 수 있다. "e.spouse.address"는 spouse 속성이 가리키는 객체의 address 속성을 읽는

표현이다. address 속성은 Address 객체를 가리키는 객체식별자 타입으로 스키마에서 OID(Address)의 타입을 가진다. address는 다시 객체식별자 타입을 가지므로 path exression을 사용하여 address가 가리키는 객체의 속성을 읽을 수 있으며 "e.spouse.address.city"는 address가 가리키는 객체의 city 속성을 읽는 표현이다. Path expression "e.spouse.salary" 역시 이와 마찬가지 방법으로 해석된다.

```
select e.spouse.address.city
from   Employee e
where  e.spouse.salary > 10000
```

Path expression에 사용된 객체식별자 속성이 가리킬 수 있는 클래스의 종류는 스키마 생성시 결정된다. 하지만, 질의의 경우에 따라서는 이를 무시하고 질의시에 클래스의 종류를 지정할 수도 있다. 이는 도메인 치환(domain substitution)이라 부른다. 도메인 치환을 사용하기 위해서는 해당 객체식별자 속성 다음에 가리키고자 하는 클래스의 이름을 "[<클래스 이름>]"의 형태로 지정한 다음에 path expression을 사용하면 된다. 예를 들어 "e.spouse.address.city"에 도메인 치환을 적용하면 "e.spouse[Employee].address[Address].city" 가 된다.

Example

■ **스키마 생성**

```
// Address 클래스를 정의한다.
create class Address (
      city     varchar(100),
      zip      char(10)
);

// Employee 클래스를 정의한다.
create class Employee (
      name     varchar(100),
      salary   integer,
      address  OID(Address)
);
```

```
// Employee 클래스에 spouse 속성을 추가한다. Employee를 가리키는 객체식별자 타입은
// Employee 클래스가 생성되기 전까지는 사용할 수 없으므로, Employee 클래스가 생성된
// 다음에 해당 속성을 추가한다.
alter class Employee add (spouse OID(Employee));
```

■ **데이터 입력**

```
// Employee "tom" 객체 생성
insert into Employee values (1, "tom", 100);

// Employee "jane" 객체 생성
insert into Employee values (2, "jane", 10000);

// Address "LA"
insert into Address values (1, "LA", "111-111");

// Employee "tom"의 객체 식별자를 구함
// 객체 식별자는 데이타베이스에 따라 다르게 나타나므로 여기서는 구해진
// 객체 식별자를 oid_string_of_tom로 가정한다.
select Employee from Employee where name = "tom";

// Employee "jane"의 객체 식별자를 구함 -> oid_string_of_jane
select Employee from Employee where name = "jane";

// Address "LA"의 객체 식별자를 구함 -> oid_string_of_LA
select Address from Address where city = "LA";

// Employee "tom"과 Employee "jane"을 배우자 관계로 설정
// Employee "tom"과 Address "LA"를 거주지 관계로 설정
update object 'oid_string_of_tom' set spouse='oid_string_of_jane',
                                      address = 'oid_string_of_LA';

// Employee "jane"과 Employee "tom"을 배우자 관계로 설정
// Employee "jane"과 Address "LA"를 거주지 관계로 설정
update object 'oid_string_of_jane' set spouse='oid_string_of_tom',
                                      address = 'oid_string_of_LA';
```

■ **질의**

```
select e.spouse.address.city
from   Employee e
```

```
where  e.spouse.salary > 10000
```

■ 질의 결과

```
+--------------------+
|e.spouse.address.city|
+--------------------+
|LA                  |
+--------------------+
```