Reference Manual

ODYSSEUS/OOSQL

Version 5.0

Manual Release 1

Aug. 2016

Advanced Information Technology Research Center (AITrc)

KAIST

Contents

1. OOSQL API

1.1. Interface for System Management

1.1.1. OOSQL_CreateSystemHandle

Syntax

Four OOSQL_CreateSystemHandle(OOSQL_SystemHandle* systemHandle, Four* procIndex)

Parameters

IN/OUT	Name	TYPE	Description
OUT	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
OUT	procIndex	Four*	Process Identifier

Description

Starts the process and initializes the internal data structure used by OOSQL.

Return value

eNOERROR : OOSQL has been started successfully

< eNOERROR : Error Code

Example

```
#include "OOSQL_APIs.h"

Four               procIndex;
OOSQL_SystemHandle systemHandle;
Four               e;

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

1.1.2. OOSQL_DestroySystemHandle

Syntax

Four OOSQL_DestroySystemHandle(OOSQL_SystemHandle* systemHandle, Four procIndex)

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	procIndex	Four	Process Identifier

Description

Finalizes the internal data structure used by OOSQL and terminates the process.

Return value

eNOERROR	: OOSQL has been terminated successfully
< eNOERROR	: Error Code

Example

```
#include "OOSQL_APIs.h"

Four               procIndex;
OOSQL_SystemHandle systemHandle;
Four               e;

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

1.2. Interface to Manage Databases and Volumes

1.2.1. OOSQL_Mount

Syntax

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

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for System Management
IN	numDevices	Four	Number of devices organizing volume

IN	devNames	char**	Array of device names
OUT	volID	Four*	Volume ID mounted

Description

Mounts the given volume to make it available for the storage system. Since a single volume can be composed of one more devices, you should give the number and array of device names to this function as parameters. It uses the name in UNIX file system as the device name. If the volume is mounted successfully, it returns the identifier of the volume.

Return value

eNOERROR	: Volume has been mounted
< eNOERROR	: Error Code

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 handling */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error handling */
……
```

1.2.2. OOSQL_Dismount

Syntax

```
Four OOSQL_Dismount(OOSQL_SystemHandle* systemHandle, Four volID)
```

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHA	Identifier for system management

		NDLE*	
IN	volID	Four	Database Volume ID

Description

Dismounts the mounted volume. You can specify the volume to be dismounted through a volume identifier, which is returned when the volume is mounted.

Return value

eNOERROR : Volume has been mounted

< eNOERROR : Error Code

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 handing */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error handling */
……
```

1.2.3. OOSQL_MountDB

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	databaseName	char*	Database Name

OUT	databaseID	Four*	Database ID mounted

Description

Mounts the given database to make it available for the storage system. One database is composed of one more volumes, and one volume is composed of one more devices. Database is created using the utility of OOSQL_CreateDB, and is mounted using the database name given at this time. You can use OOSQL_GetVolumeID to get the volume ID from the mounted database. You can mount database only once in the system.

Return value

eNOERROR	: Database has been mounted
< eNOERROR	: Error Code

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four             e;
Four             databaseID;

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

1.2.4. OOSQL_DismountDB

Syntax

```
Four OOSQL_DismountDB(OOSQL_SystemHandle* systemHandle, Four databaseID)
```

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	databaseID	Four	Database ID

Description

Dismounts the mounted database. The database to be dismounted is specified through the database identifier, which is returned when mounting.

Return value

eNOERROR : Database has been mounted

< eNOERROR : Error Code

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four            e;
Four            databaseID;

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

1.2.5. OOSQL_MountVolumeByVolumeName

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	databaseName	char*	Database Name
IN	volumeName	char*	Volume Name
OUT	volID	Four*	Volume ID mounted

Description

Mounts the given volume of database to make it available for a storage system. While OOSQL_MountDB mounts all the volumes organizing database, OOSQL_MountVolumeByVolumeName separately mounts a specific volume of a specific database. Though performing the same operation as OOSQL_Mount, it can be distinguished in the aspect that it has the name of database and volume as arguments.

The mounted volumes of OOSQL_Mount, and OOSQL_MountVolumeByVolumeName have to be dismounted through OOSQL_Dismount.

Return value

eNOERROR	: Database has been mounted
< eNOERROR	: Error Code

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 handling */
……
e = OOSQL_Dismount(&systemHandle, volID);
if(e < eNOERROR) /* error handling */
……
```

1.2.6. OOSQL_GetVolumeID

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	databaseID	Four	Database ID
IN	volumeName	char*	Volume Name
OUT	volumeID	Four*	Volume ID

Description

Returns ID of the volume having the given name among the volumes organizing the mounted database.

Return value

eNOERROR : Volume ID has been retrieved

< eNOERROR : Error Code

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

1.2.7. OOSQL_GetUserDefaultVolumeID

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	databaseID	Four	Database ID
OUT	volumeID	Four*	Volume ID

Description

Returns ID of the volume that is appointed as a default among the volumes organizing the mounted database. You can appoint a specific volume as a default using OOSQL_SetUserDefaultVolumeID. After mounting database, the first volume is automatically appointed as a default.

Return value

eNOERROR	: volume ID has been retrieved
< eNOERROR	: Error Code

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 handling */
……
e   =   OOSQL_GetUserDefaultVolumeID(&systemHandle,   databaseID,
&volID);
if(e < eNOERROR) /* error handling */
……
```

1.2.8. OOSQL_SetUserDefaultVolumeID

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	databaseID	Four	Database ID
IN	volumeID	Four	Volume ID

Description

Appoints the given volume as a default among volumes organizing the mounted database.

Return value

eNOERROR : Volume having the given volume ID has been appointed as a default

< eNOERROR : Error Code

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

1.3. Interface for Transactions

1.3.1. OOSQL_TransBegin

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
OUT	xactId	XactID*	Transaction ID
IN	cclevel	**CONCURRENCY LEVEL**	Concurrency level to be used by the given transaction

Description

Initializes a new transaction and declares the start of the transaction. An identifier is given to identify the created transaction, and it is returned by xactId.

cclevel is a concurrency level to be used by the given transaction. When a few transactions are executed at the same time, the concurrency level determines how to process it. cclevel is a type of ConcurrencyLevel that is defined as follows. typedef enum { X_BROWSE_BROWSE, X_CS_BROWSE, X_CS_CS, X_RR_BROWSE, X_RR_CS, X_RR_RR } ConcurrencyLevel;

The current version of ODYSSEUS/OOSQL uses two type of the concurrency level: X_BROWSE_BROWSE and X_RR_RR.

X_BROWSE_BROWSE is a level using no read lock and long write lock, and it is used in the transaction that mainly reads. The transaction, which is executed with X_BROWSE_BROWSE, can perform the read operation for the given volume (data) though the other transactions perform the write operation. And, it can execute the write operation when the other transactions do not execute the write operation.

X_RR_RR is a level using long read lock and long write lock, and it is used in the transaction that mainly writes. The transaction with X_RR_RR cannot execute the read operation for the given volume (data) when the other transactions execute the write operation. It can execute the write operation when the other transactions do not execute the write operation on the level of X_RR_RR. And, it can execute the write operation when the other transaction does not execute the write operation.

Return value

eNOERROR	: Transaction has been successfully started up
< eNOERROR	: Error Code

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four              e;
XactID            xactID;

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

1.3.2. OOSQL_TransCommit

Syntax

Four OOSQL_TransCommit(OOSQL_SystemHandle* systemHandle, XactID* xactId)

Parameters

IN/OUT	**Name**	TYPE	**Description**

IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	xactId	XactID*	Transaction ID

Description

Completes the given transaction. When the transaction is completed, the operations on database are practically reflected on the database.

Return value

eNOERROR	: Transaction has been successfully completed
< eNOERROR	: Error Code

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four             e;
XactID            xactID;

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

1.3.3. OOSQL_TransAbort

Syntax

Four OOSQL_TransAbort(OOSQL_SystemHandle* systemHandle, XactID* xactId)

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	xactId	XactID*	Tansaction ID

Description

Aborts the given transaction. When the transaction is aborted, all operations executed among transactions on the database are aborted, and the database state becomes the state before the transaction start up.

Return value

eNOERROR   : Transaction has been successfully aborted

< eNOERROR : Error Code

Example

```
#include "OOSQL_APIs.h"

OOSQL_SystemHandle systemHandle;
Four             e;
XactID            xactID;

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

1.4. Interface for Query Processing

1.4.1. OOSQL_AllocHandle

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	volID	Four	Database Volume ID
OUT	handle	OOSQL_Handle*	Information on the handle allocated

Description

Gets the handle for executing OOSQL operations on a query. All the OOSQL operations on the query are executed through this handle.

Return value

eNOERROR   : Handle has been successfully retrieved

< eNOERROR : Error Code

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

1.4.2. OOSQL_FreeHandle

Syntax

Four OOSQL_FreeHandle(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	handle	OOSQL_Handle	Information on handle to be returned.

Description

Releases the handle obtained for executing OOSQL on a query.

Return value

eNOERROR : handle has been successfully released.

< eNOERROR : Error Code

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 handling */
……
```

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

1.4.3. OOSQL_Prepare

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle information
IN	stmtText	**CHAR***	Query to be executed
INOUT	sortBufferInfo	**OOSQL_SORTBUFFERINFO***	Information on a buffer of a sort used during executing query. NULL given, a disk sort is executed based on the volume, in which the query is being procesed.

Description

Prepares for processing the given query after it checks if there is any syntax error in the query.

OOSQL execute a sort at need during executing the query. a sort is executed on the basis of memory or disk. The sort based on memory shows faster execution. A representative query using a sort is one executing truncation operation for keywords in text information retrieval.

The structure of OOSQL_SortBufferInfo is as follows:

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

For the mode, you can appoint whether the sort should be executed on the basis of Disk or Memory, or on Disk in case of insufficient memory and on Memory in the other case, respectively with OOSQL_SB_USE_DISK, OOSQL_SB_USE_MEMORY, OOSQL_SB_USE_MEMORY_WITH_DISK.

Being the part to be filled for all the modes, diskInfo appoints the volume in which the sort should be executed. The structure of OOSQL_DiskSortBufferInfo is as follows:

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

memoryInfo is the part to be filled when the mode is OOSQL_SB_USE_MEMORY or OOSQL_SB_USE_MEMORY_WITH_DISK, and it determines the memory in which the sort is executed. The structure of OOSQL_MemorySortBufferInfo is as follows:

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

sortBufferPtr is the position where the memoty to be sorted is located, and sortBufferLength is the memory size. The users should determine the memory to be sorted. sortBufferUsedLength is the size of the memory used actually.

When mode is OOSQL_SB_USE_MEMORY, and sortBufferLength is smaller than the memory necessary for executing a query, the error, eNEEDMORESORTBUFFERMEMORY_OOSQL, is returned. In this case, a user has to execute OOSQL_Prepare once and again after increasing the memory size.

Return value

eNOERROR : It has been ready for executing the query.

eNEEDMORESORTBUFFERMEMORY_OOSQL : Insufficient memory for the memory sort.

< eNOERROR : Error Code

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 handling */
```

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

1.4.4. OOSQL_Execute

Syntax

Four OOSQL_Execute(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle

Description

Executes the query prepared from OOSQL_Prepare, and then, gets ready for reading the first result of the query.

Return value

eNOERROR	: The query has been successfully executed.
< eNOERROR	: Error Code

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 handling */

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

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

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

1.4.5. OOSQL_ExecDirect

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	stmtText	char*	Query to be executed
INOUT	sortBufferInfo	**OOSQL_SORTBUFFERINFO***	Information on a buffer of a sort to be used during executing a query. If NULL is passed over, It executes the disk sort based on the volume in which the query is executing.

Description

Checks if there is any syntax error in the given query and prepares for executing this query. Then, it gets ready for reading the first result of the query.

Return value

eNOERROR : Query has been successfully executed.

< eNOERROR : Error Code

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 handling */

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

1.4.6. OOSQL_Next

Syntax

Four OOSQL_Next(OOSQL_SystemHandle* systemHandle, OOSQL_Handle handle)

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle

Description

Loads the next result of the query. When there is no result to be loaded, it returns ENDOFEVAL.

Return value

ENDOFEVAL	: No result to be returned has been found
eNOERROR	: Query result has been successfully loaded
< eNOERROR	: Error Code

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 handling */

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

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

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

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	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	columnNumber	Two	**NUMBER OF A COLUMN ORGANIZING THE QUERY RESULT**
IN	startPos	Four	Location of the part to be read in the column value
INOUT	bufferPtr	Void*	Buffer for taking the column value
IN	bufferLength	Four	Length of buffer
OUT	returnLength	Four*	Length of the read data

Description

Reads the value of a single column organizing the query results. columnNubmer is the number of a column to be read, and becomes 0 on the first column. startPos and bufferLength are arguments appointing the part to be read, and they indicate the start position and length. The query results are stored in the memory appointed by bufferPtr. returnLength is the length of the result value taken actually from the query result.

Return value

eNOERROR	: Column value has been successfully retrieved
< eNOERROR	: Error Code

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 handling */

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

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

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

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***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	**NRESULTSTOREAD**	**FOUR**	Number of results to be read, if -1 is passwd over, read the data as much as possible
IN	headerBuffer	void*	Header buffer, if -1 is passed over, the information of header is not created
IN	**HEADERBUFFERSIZE**	Four	Size of header buffer
IN	dataBuffer	void*	Data buffer, if NULL is passed over, the information of data is not created
IN	dataBufferSize	Four	**SIZE OF DATA BUFFER**
OUT	nResultsRead	Four*	Number of read results

Description

Read multiple query results at a time. Calling this API can replace calling OOSQL_Next and OOSQL_GetData multiple times and improve performance. Multiple objects are read into the header buffer and data buffer. The header buffer reads the information for interpretating each object; data buffer reads real data. If the query formular that reads fixed length data is excuted, the header buffer does not have to be created. The header information of the header buffer can be interpretated using the following macro.

□ OOSQL_MULTIPLERESULT_NTH_OBJECT_OFFSET(headerBuffer, nColumns, i)

Returns the position of ith result object in dataBuffer.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_SIZE(headerBuffer, nColumns, i)

  Returns the size of the ith result object in dataBuffer.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_ISNULL(headerBuffer, nColumns, i, j)

  Returns whether the jth column constituting the ith result object is NULL or not. j can be as large as the number of attributes in the select clause.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_SIZE(headerBuffer, nColumns, i, j)

  Returns the size of the jth column constituting the ith result object in dataBuffer. j can be as large as the number of attributes in the select clause. The size in databBuffer can be different from the size in the database. The reason is that, if there is a very big size of object in the database, the memory buffer cannot accomodate it. The memory buffer can read up to 8KB.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_REALSIZE(headerBuffer, nColumns, i, j)

  Returns the size of the jth column constituting the ith result object in the database. j can be as large as the number of attributes in the select clause.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_OID(headerBuffer, nColumns, i, j)

  Returns the OID of the object containing the jth column constituting the ith result object in database. j can be as large as the number of attributes in the select clause.

- OOSQL_MULTIPLERESULT_NTH_OBJECT_ITH_COLUMN_COLNO(headerBuffer, nColumns, i, j)

  Returns the column number of the jth column constituting the ith result object in the database. j can be as large as the number of attributes in the select clause

Return value

eNOERROR	: Query results have been successfully read
ENDOFEVAL	: No result has been found

< eNOERROR : Error Code

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 handling */

e  =  OOSQL_Prepare(&systemHandle,  handle,  "select  user_page  from
user_page where match(description, \‘Korea\’)>0", NULL);
if(e < eNOERROR) /* error handling */

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

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 handling */

   nTotalResultsRead += nResultsRead;

   /* In case 80% of buffer is filled, 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++)
    {
        /* read OID from dataBuffer */
        memcpy((char *)&oid, pOidBuffer, sizeof(OID));
        ……
        /* increase offset of databuffer by size of OID */
        pOidBuffer += sizeof(OID);
    }
    ……
    e = OOSQL_FreeHandle(&systemHandle, handle);
    if(e < eNOERROR) /* error handling */
    ……
    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	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	nColumns	Four	Number of columns to be read
IN	getDataStruct	OOSQL_GetDataStruct*	Array of the structures that define the contents to be read for each column

Description

Read several columns organizing the query results. nColumns is the number of columns to be read, and getDataStruct is the array of information on the columns to be read.

getDataStruct is OOSQL_GetDataStruct, which is defined as follows:

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

columnNumber of OOSQL_GetDataStruct is the number of a column to be read, and startPos is the start position of dada in the column to be read. bufferPtr is the pointer of buffer where the read data will be returned, and bufferLength is the length of buffer which bufferPtr appoints. returnLength returns the length of data that has been read.

Return value

eNOERROR : Columns' Values have been successfully taken after the query executed.

< eNOERROR : Error Code

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 handling */

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

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

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

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	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	columnNumber	Two	Parameter number
IN	startPos	Four	Start position of the part to be used
IN	columnValuePtr	void*	Buffer for taking the column value
IN	bufferLength	Four	Length of the buffer

Description

Appoints the value of argument used in the query. An argument is expressed as '?' in the query formula. It is useful for the binary data that cannot be directly described in the query formula or for the large-sized multimedia data to appoint. columnNumber is the argument number, which is determined according to the order of '?' in the query formula. The first argument number is 0.

Return value

eNOERROR : Value has been successfully modified

< eNOERROR : Error Code

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 handling */

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

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

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

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

1.4.11. OOSQL_GetOID

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	targetNumber	Two	Number of a table used in FROM clause

OUT	oid	OID*	OID of the object

Description

Returns OID of the object that has been read to execute the query. targetNumber is the number of a table used in FROM clause, and the table is one including the object to be read. The first table number is 0.

Return value

eNOERROR	: OID has been successfully taken
< eNOERROR	: Error Code

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 handling */

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

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

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

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

1.4.12. OOSQL_GetNumResultCols

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
OUT	nCols	Two*	Number of Columns

Description

Retrieves the number of columns organizing the query results.

Return value

ENOERROR : The number of columns organizing the query results has been successfully taken

< eNOERROR : Error Code

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 handling */

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

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

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

1.4.13. OOSQL_GetResultColName

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	columnNumber	Two	Column Number
INOUT	**COLUMNNAMEBUFFER**	char*	Buffer for taking the column name
IN	bufferLength	Four	Length of buffer ColumnName

Description

Retrieves the name of the column organizing a query result.

Return value

eNOERROR	: Column name in the query result has been successfully taken
< eNOERROR	: Error Code

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 handling */

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

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

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

1.4.14. OOSQL_GetResultColType

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
IN	columnNumber	Two	Column Number
OUT	columnType	Four*	Column Type

Description

Retrieves the type of the column organizing a query. The retrieved type has a meaning as follows:

Retrieved Value	SQL Type
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: The type of the column organizing a query has been successfully retrieved

< eNOERROR : Error Code

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 handling */

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

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

e = OOSQL_GetResultColType(&systemHandle, handle, 0, &type)
if(e < eNOERROR) /* error handling */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error handling */
……
```

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***	Identifier for system management
IN	handle	OOSQL_Handle	OOSQL handle
OUT	nResults	**FOUR***	Number of estimated query results

Description

Estimate the number of query results. This API must be called after OOSQL_Prepare() is executed.

Return value

eNOERROR	: The number of query results has been successfully estimated.
< eNOERROR	: Error code

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 handling */

e = OOSQL_Prepare(&systemHandle, handle, "select …", NULL);
if(e < eNOERROR) /* error handling */
……
e = OOSQL_EstimateNumResults(&systemHandle, handle, &nResults);
if(e < eNOERROR) /* error handling */
……
e = OOSQL_FreeHandle(&systemHandle, handle);
if(e < eNOERROR) /* error handling */
……
```

1.5. Interface for Text Management

1.5.1. OOSQL_Text_MakeIndex

Syntax

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

Parameters

IN/OUT	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	volID	Four	Database volume ID
IN	temporaryVolId	Four	Temporary volume ID
IN	className	Char*	Name of class to update text index

Description

In OOSQL, while inserting a text into database, you can reflect it on the text index immediately or later. In case of a later reflection, it is made through this function. However, when the text attribute of the given class is set up once to be DEFERED mode, it cannot be changed into IMMEDIATE mode unless this command is executed. Since this command accesses all objects of the class, it is recommendable to execute this command only when you insert the bulk of data.

Return value

eNOERROR: Text index has been successfully constructed.

< eNOERROR : Error Code

Example

```
#include "OOSQL_APIs.h"

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

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	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	volID	Four	Database Volume ID
IN	KEYWORDEXTRACTOR	**CHAR ***	Name of a default keyword extractor to be added
IN	version	Four	Version number of a default keyword extractor to be added.
IN	KEYWORDEXTRACTORFILEPATH	char *	Information on the location of the directory having a default keyword extractor to be added
IN	keywordExtractorFunctionName	char *	Name of the function initiating the keyword extractor
IN	getNextPostingFunctionName	char*	Name of the function retrieving keywords and posting information from the extractor
IN	FINALIZEKEYWORDEXTRACTORFUNCTIONNAME	char*	Name of the function stopping the operation of the keyword extractor

Description

Registers a default keyword extractor on ODYSSEUS/OOSQL. This default keyword extractor is applied to all the text columns where any custom keyword extractor is not additionally defined.

Return value

eNOERROR: Default keyword extractor has been registered

< eNOERROR : Error Code

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 handling */
……
```

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	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	Identifier for system management
IN	volID	Four	Database Volume ID
IN	KEYWORDEXTRACTOR	**CHAR ***	Name of a keyword extractor to be added
IN	version	Four	Version number of a keyword extractor to

			be added
IN	keywordExtractorFilePath	char *	Position the information on the directory having a keyword extractor to be added
IN	keywordExtractorFuntionName	char *	Name of the function initiating the keyword extractor
IN	getNextPostingFunctionName	char*	Name of the function retrieving keywords and posting information from the keyword extractor
IN	**FINALIZEKEYWORDEXTRACTORFUNTIONNAME**	char*	Name of the function stopping operation of keyword extractor
OUT	keywordExtractorNo	Four *	Number of the added keyword extractor

Description

Registers a custom keyword extractor on ODYSSEUS/OOSQL. After registering the keyword extractor, a user can set the registered filter for any text column.

Return value

eNOERROR	: Keyword extractor has been registered
< eNOERROR	: Error Code

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 handling */
……
```

1.5.4. OOSQL_Text_DropKeywordExtractor

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	volID	**FOUR**	Database Volume ID
IN	KEYWORDEXTRACTORNAME	Char *	Name of a keyword extractor to be deleted
IN	version	Four	Version number of a keyword extractor to be addeddeleted

Description

Deletes a custom keyword extractor from ODYSSEUS/OOSQL.

Return value

eNOERROR: Keyword extractor has been deleted

< eNOERROR : Error Code

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

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	**Name**	TYPE	**Description**
IN	systemHandle	**OOSQL_SYSTEMHANDLE***	Identifier for system management
IN	volID	**FOUR**	Database Volume ID
IN	className	Char*	Class Name
IN	columnName	Char*	Text Column Name
IN	KEYWORDEXTRACTORNO	Four	Number of a keyword extractor to be applied

Description

Set a keyword extractor to be applied to the given text column.

Return value

eNOERROR : Keyword extractor has been set for the given text column

< eNOERROR : Error Code

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 handling */
```

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

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	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for system management
IN	volID	FOUR	Database Volume ID
IN	filterName	char *	Name of filter to be added
IN	version	Four	Version number of filterto be added
IN	filterFilePath	char *	Position information on directory where the filter to be added is located
IN	filterFunctionName	char *	Symbol name of filter function
OUT	filterNo	Four *	Number of filter added

Description

Register the custom filter on ODYSSEUS/OOSQL. After registering the filter, a user can set the registered filter for a random text column.

Return value

eNOERROR	: filter registered
< eNOERROR	: Error Code

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 handling */
……
```

1.5.7. OOSQL_Text_DropFilter

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for System Management
IN	volID	**FOUR**	Database Volume ID
IN	filterName	char *	Name of filter to be deleted
IN	version	Four	Version number of filter to be deleted

Description

Deletes the custom filter from ODYSSEUS/OOSQL.

Return value

eNOERROR	: filter has been deleted
< eNOERROR	: Error Code

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 handling */
……
e = OOSQL_Text_DropFilter(&systemHandle, volID,
  "filter-name", 1,);
if(e < eNOERROR) /* error handling */
……
```

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	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for System Management
IN	volID	**FOUR**	Database Volume ID
IN	className	char*	Class Name
IN	columnName	char*	Text Column Name
IN	filterNo	Four	Number of filter to be applied

Description

Set the filter to be applied to the given text column.

Return value

eNOERROR	: filter has been set to the given column
< eNOERROR	: Error Code

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 handling */
……
e = OOSQL_Text_SetFilter(&systemHandle, volID, “class-name“, “column-name", 1);
if(e < eNOERROR) /* error handling */
……
```

1.5.9. Keyword Extracting Function Prototype

ODYSSEUS/OOSQL extracts a keyword from the given text to perform the keyword base search. ODYSSEUS/OOSQL uses a keyword extractor to extract the keyword. A keyword extractor is composed of three keyword extracting functions being an external dynamic library (*.so in UNIX and *.dll in Windows).

The keyword extracting function starts a keyword extractor, returns the results of keyword extraction, and stops the keyword extractor. A user can name these functions at his/her random. The names of these functions are registered by using the interfaces of OOSQL_Text_AddDefaultKeywordExtractor and OOSQL_Text_AddKeywordExtractor , or utilities of InstallKeywordExtractor and InstallKeywordExtractor to recognize them.

Three function prototypes of a keyword extractor are as follows. Refer to NullKeywordExtractor.c in OOSQL/example/null_keyword_extractor/ for how to create these functions.

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

locationOfContent : It indicates the position of the contents from which keywords are extracted. In case of OOSQL_TEXT_IN_FILE, the contents are given through a temporary file, and the name of this file is given through the argument of inFileOrContent. In case of OOSQL_TEXT_IN_MEMORY, the contents are given through an argument, and the contents are given through the argument of inFileOrcontent. In case of OOSQL_TEXT_IN_DB, the position in the DB storing the contents is given through handle, volId, className, oid, and colNo. A keyword extractor reads using these arguments and the function of OOSQL_Text_FetchContent.

handle : It is the handle for calling OOSQL API. A valid value is obtained only in case locationOfContent is OOSQL_TEXT_IN_DB.

volId : It is an identifier of DB volume that contains the contents of keyword extraction. A valid value is obtained only in case locationOfContent is OOSQL_TEXT_IN_DB.

className : It is the name of the class that stores contents of keyword extraction. A valid value is obtained only in case locationOfContent isOOSQL_TEXT_IN_DB.

oid : It is an identifier of the object that contains the contents of keyword extraction. A valid value is obtained only in case locationOfContent is OOSQL_TEXT_IN_DB.

colNo : It is the column number that contains the contents of keyword extraction. A valid value is obtained only in case locationOfContent is OOSQL_TEXT_IN_DB.

inFileOrContent : It is the name of a temporary file containing the contents of keyword extraction or the content itself. If locationOfContent is OOSQL_TEXT_IN_FILE, it is the name of a temporary file, and if locationOfContent is OOSQL_TEXT_IN_MEMORY, it is the content.

resultHandle : It is an identifier of the results of keyword extraction. It is used as arguments of getAndNextKeywordExtractor and closeKeywordExtractor.

This function extracts keywords from contents, and stores the results in the internal memory. The results, stored in the internal memory, can be identified with resultHandle and returned through getAndNextKeywordExtractor.

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

handle : It is an identifier of the results. It is the return value of openAndExecuteKeywordExtractor.

keyword : It returns the keyword string that has been extracted by the keyword extractor.

nPositions : It returns the number of times that the keyword is found in the text.

positionList : It returns the position where the given keyword is found in the text. The form of the position of a single keyword is (the position of the sentence, the position of the word in the sentence), and each element of the form is int type.

This function returns the keyword extracted contents, such as keyword strings and position informations of the keywords. In case all keywords have been read, OOSQL_TEXT_DONE is returned. In the other case, eNOERROR is returned.

int closeKeywordExtractor(Four handle)

handle : As an identifier to identify the results, it is the return value of openAndExecuteKeywordExtractor

This function is called in case the contents, from which the keyword has been extracted, are thoroughly read. This function returns the resources that have been obtained by openAndExecuteKeywordExtractor.

1.5.10. Filter Function prototype

Before extracting keywords, ODYSSEUS/OOSQL uses a filter that transforms the contents into the format that the keyword extractor can read. A filter is an external dynamic library (*.so in UNIX and *.dll in Windows) that transforms the various formats of texts into a single format.

For example, it transforms Microsoft Word and PDF text into a text.

int filter(char *inFile, char *outFile)

inFile: File name before using the filter

outFile: File name after using the filter

A user can use any symbol for a filter he/she wants. Using the interface of OOSQL_Text_AddFilter or the utility of InstallFilter, the user can register the symbol name.

1.6. Other Interfaces

1.6.1. OOSQL_GetErrorMessage

Syntax

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

Parameters

IN/OUT	**Name**	TYPE	**Description**
IN	systemHandl	OOSQL_SYSTEMHA	ID for System Management

	e	NDLE*	
IN	errorCode	Four	Error Code
INOUT	messageBuffer	char*	Buffer for getting error message
IN	bufferLength	Four	Size of MessageBuffer

Description

Changes the given error code into the appropriate error message.

Return value

eNOERROR: Given error code has been successfully transformed into the appropriate error message

< eNOERROR : Error Code

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	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for System Management
IN	errorCode	Four	Error Code

INOUT	messageBuffer	char*	Buffer for getting error message
IN	bufferLength	Four	Size of MessageBuffer

Description

Changes the given error code into the appropriate error name.

Return value

eNOERROR : Given error code has been successfully transformed into the appropriate error name

< eNOERROR : Error Code

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	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for System Management
IN	handle	OOSQL_Handle	Query identifier
INOUT	messageBuffer	char*	Buffer for receiving an error message

IN	bufferLength	Four	Size of message Buffer

Description

Converts the latest error that has occurred while executing the given query into the appropriate error message and returns it.

Return value

eNOERROR	: Error message has been successfully returned
< eNOERROR	: Error Code

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	Name	TYPE	Description
IN	systemHandle	OOSQL_SYSTEMHANDLE*	ID for System Management
IN	oid	OID*	OID
INOUT	oidString	char*	OID String

Description

Transform the given OID into the string of OID. The size of oidString should be more than 33 bytes. When passing over oid to the query formula for the queries of SELECT FROM OBJECT, UPDATE OBJECT SET, and DELETE FROM OBJECT, a user transfer it to oid string through this interface.

Return value

eNOERROR	: OID has been successfully transformed into OIDString
< eNOERROR	: Error Code

Example

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

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

2. Utilities

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

OOSQL uses DB to manage the string device. DB is composed of one or more volumes, and each volume is also composed of one or more devices. You should first create DB to construct it. OOSQL_CreateDB is the utility that creates DB.

database_name is the name of DB. Users can name it at random. *database_name* is used as an argument of the interfaces for DB. *volume_name* is the name of a default volume of DB. In case *volume_name* is omitted*,* it has the same name as *database_name*. *database_directory* is the directory where DB will be created, and it is set up as a directory that the environmental variable of $ODYS_OODB when it is ommitted.

A volume is composed of multiple device files, and these files are appointed with –device option. *device_path* is the path of a device, and *numberOfPages* is the value that appoints the number of disk pages composing the device. In case the device is not appointed, a device with the same name as volume_name creates in the database directory. The device is composed of 16,000 pages(when the size of a page is 4Kbyte, the size of a volume becomes 64Mbyte). *extent_size* has a different value according to the size of disk used for DB. *extent_fill_factor* appoints what percent of the extent should be emptied, and it is used to increase the efficiency by emptying a part of the extent in case a large amount of data is stored. OOSQL creates a large object and a small object respectively in areas different one another inside the volume. Each area is allocated in the unit of *segment_size* page, so the clustering effect among objects increases, as *segment_size* value gets higher. However, if *segment_size* value is too high, it can waste the storage. In the most cases, if the volume size is about 2GB, *segment_size* is about 500MB. If *segment_size* is not defined, it is set to be a quarter of the initial device size.

OOSQL_CreateDB creates DB directory, and initializes the given device. Furthermore, it records DB name, directory path, volume name, device path, etc. on the file whose name is $ODYS_OODB/OOSQL_SysDirFile.

Selecting devices that compose a volume

The correctness and performance of database operations depend on the types of devices composing a volume. Thus, we must select proper devices for each type of volume to ensure correctness and to obtain the best performance. For data volumes, temporary volumes, and log volumes, raw devices must be used in UNIX, and O/S file must be used in NT. In NT, raw devices do not have to be used because there is a mechanism to treat an O/S file as a raw device. In UNIX, since such a mechanism is not provided, we must use raw devices. If raw devices were not used, due to unnecessary buffering in the O/S files, main memory is wasted, and consequently, the system can slow down. In particular, the main memory that is used for O/S file buffering can grow as large as the size of the device (say, a few hundred Mbytes ~ a few Gbytes), thrashing can occur due to shortage of remaining main memory.

Example

The following is an example of creating DB whose name is testdb. Since the volume name is omitted, testdb becomes the volume name. The device size is 640,000KB (160,000 * 4KB).

➢ OOSQL_CreateDB testdb –extentSize 32 –segmentSize 6000 –device

$ODYS_OODB/testdb/testdb 160000

2.2. OOSQL_DestroyDB

Usage

OOSQL_DestroyDB *database_name*

Description

Deletes the whole database.

Example

The following is an example of deleting testdb.

➢ OOSQL_DestroyDB testdb

2.3. OOSQL_InitDB

Usage

OOSQL_InitDB *database_name*

Description

Initializes the given DB. OOSQL_InitDB deletes all

objects and classes in DB, and creates a new DB file.

The syntax of OOSQL_InitDB is as follows.

Example

The following is an example of initializing testdb.

➢ 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

Creates a DB. *database_name* is the name of DB that is defined at a user's random, and it is used as an argument of the interface for OOSQL. *volume_name* is the name of default volume of DB. If *volume_name* is omitted, it has the same name as *database_name*. *database_directory* indicates the directory in which DB is created. If it is ommitted, it is set up as a directory that is appointed by $ODYS_OODB.

A volume is composed of several device files. You should appoint these devices with –device option. *device_path* is the path of a device, and *numberOfPages* is the number of disk pages of the device. If a device is not appointed, the device, which is composed of 16, 000 pages, will be created in DB directory. In this case the device has the same name as *volume_name*.The value of *extent_size* varies according to the size of the disk. *extent_fill_factor* appoints what percent of the extent should be emptied. *extent_fill_factor* is used to increase the efficiency of the operation in case of storing a large amount of data at once. OOSQL creates large objects and small objects respectively in different areas. Each area is allocated in the unit of *segment_size* page. Thus, the clustering effect increases, as *segment_size* gets higher. However, if *segment_size* value is too high, it wastes the storage. In the most cases, if the volume size is about 2GB, *segment_size* is set to about 500MB. If *segment_size* is not appointed, it is appointed to be a quarter of the default device size.

OOSQL_AddVolume initializes the given devices, and records volumename, path, etc. on the file whose name is $ODYS_OODB/OOSQL_SysDirFile.

Example

The following is an example of adding the volume of testdb2 to the database of testdb. The size of the device is 640,000KB (160,000 * 4KB).

- ➢ 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

Deletes a volume from DB.

Example

The following is an example of deleting testdb2 volume from testdb database.

- ➢ 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

Initializes the volume in use.

OOSQL_InitVolume can newly appoint the devices that compose a volume while initializing the volume, and it can modify extentSize, extentFillFactor, and segmentSize.

Example

The follwing is an example of initializing testdb2 volume in testdb database.

➢ OOSQL_InitVolume testdb testdb2

2.7. OOSQL_AddDevice

Usage

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

Description

Adds a new device to the volume.

Example

The following is an example of adding testdb2-1 device to testdb2 volume in testdb database.

➢ 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

Initializes the volume for log. Logs are the records of the operations on DB. The contents of DB can be returned to the original state in case DB system is improperly downed by external causes. You have to create the log volume to use the roll back operation of a transaction or the recovery function. When the log volume does not exist or not be appointed, you cannot use the roll back of a transaction and the recovery function caused by an improper down of the system.

Log volume is composed of one device. You do not have to appoint *segment_size*. *volume_id* is the identifier of the volume that is appointed at user's random. You had better appoint the value to be less than 1000 in order to make the identifiers not be overlapped.

For log volumes, raw devices must be used in UNIX, and O/S file must be used in NT. In NT, raw devices do not have to be used because there is a mechanism to treat an O/S file as a raw device. In UNIX, since such a mechanism is not provided, we must use raw devices. If raw devices were not used, due to unnecessary buffering in the O/S files, main memory is wasted, and consequently, the system can slow down.

For enabling OOSQL application program to use the created log, you should set up the environmental variable of $COSMOS_LOG_VOLUME to indicate the position where the log volume exists. Use semicolon(;) to separate devices when you specify multiples devices in $COSMOS_LOG_VOLUME.

Example

The following is an example of initializing the log volume, testdb.log.

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

The following is an example of setting up the environmental variable to make the created log be available.

```
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_FormatCoherencyVolume is a utility to initialize a coherency volume for a multi-server configuration. For a multi-server configuration, if a buffer is updated in a process, the change is recorded in the coherency volume, and other processes make the contents of the buffer consistent by looking up the changes recorded in the coherency volume.

Coherency volumes must use O/S files as devices, but must not use raw devices both in UNIX and in NT. *volume_id* is the volume identifier that the user assigned. It is recommended that *volume_id* of the coherency volume be a number less than 1000 so that it does not conflict with volume identifier of ordinary volumes. An absolute path of the O/S file must be assigned to *device_path*.

For an OOSQL application program to use the coherency volume, the environment variable $COSMOS_COHERENCY_VOLUME must be set the path where the coherency volume resides. The coherency volume must be used where there are updates, insertion or deletion of data in a multi-server configuration.

Example

The following is an example of initializing the coherency volume, coherency.

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

The following is an example of setting up the environmental variable to make the created coherency volume be available.

```
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

Makes an index for the given single text type attribute of the class in batch.

database_name is the name of the DB in which a text index will be made. *volume_name* is the name of the volume in which a text index will be made. *class_name* is the name of the class for which a text index will be made. *attribute_name1 …* is the name of the attribute for which a text index will be made. *data_file_name* is the name of the file in which there are the contents for which a text index will be made. The *data_file_name* file sticks to the input file form of

OOSQL_LoadDB. *loaddb* determines whether OOSQL_LoadDB calls OOSQL_MakeTextIndex or not. When only the keyword extraction and the index construction are needed without loading the data actually, *loaddb* can be omitted.

Example

This is an example of creating a text index. In this example, both database name and the volume name are testdb. The class name is Newspaper, and the attribute name for which a text index will be created is title. The input file name is test.in.

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

OOSQL_MakeTextIndex is composed of several operations. From Section 2.10.1 to 2.10.6, we explain the operations.

2.10.1. OOSQL_ExtractKeyword

Usage

OOSQL_ExtractKeyword *database_name volume_name class_name attribute_name data_file_name*

Description

Reads the contents from the file indicated by data_file_name, extracts the keywords to be used for the index, and stores them in a file.

database_name is the name of the DB in which a text index will be made. *volume_name* is the name of the volume in which a text index will be made. *class_name* is the name of the class for which a text index will be made. *attribute_name1 …* is the name of the attribute for which a text index will be made. *data_file_name* is the name of the file in which there are the contents for which a text index will be made. The *data_file_name* file sticks to the input file form of OOSQL_LoadDB.

The name of the file in which OOSQL_ExtractKeyword stores the extracted keyword is as follows.

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

Example

This is an example of extracting keywords for title attribute in Newspaper class of testdb volume of testdb DB in test.in file.

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

2.10.2. OOSQL_SortPosting

Usage

OOSQL_SortPosting *input_file output_file*

Description

Sorts the contents of the file created by OOSQL_ExtractKeyword.

OOSQL_SortPosting sorts the contents of *input_file* and writes the results on *output_file*. Both *input_file* and *output_file* have the structure of posting file. OOSQL_MakeTextIndex appoints the name of input_file in OOSQL_SortPosting to be $ODYS_TEMP_PATH/TEXT_<*class_name*>_<*attribute_name*>_Posting, and appoints *output_file* as follows.

$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

Stores the contents of a data file in DB and OIDs of newly created objects in TEXT_<class_name>_OID file in $ODYS_TEMP_PATH directory.

Refer to Section 2.11 for a detailes of OOSQL_LoadDB.

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_MapPosting converts the file created by OOSQL_SortPosting using text number/OID table made by OOSQL_LoadDB.

posting_file_name is the name of the posting file to be converted*, and new_posting_file_name* is the name of a new posting file in which the results will be stored. *oid_file_name* is the name of the file in which text number/OID table to be used for conversion is stored.

OOSQL_MapPosting finds the mapping table file in the directory that is appointed in the environmental variable of ODYS_TEMP_PATH, and creates the converted file in the same directory. OOSQL_MakeTextIndex appoints

posting_file_name to be

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

new_posting_file_name to be

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

It changes the name of *new_posting_file_name* file into

$ODYS_TEMP_PATH/TEXT_<class_name>_<attribute_name>_SortedPosting, and continues the operations from the stage of OOSQL_BuildTextIndex.

2.10.5. OOSQL_BuildTextIndex

Usage

OOSQL_BuildTextIndex *database_name* [*volume_name*] *class_name* *attribute_name*

Description

Creates a text index from the files created by OOSQL_MapPosting. *database_name* is the name of the DB containing the data, and *volume_name* is the name of the volume containing the data. *class_name* is the name of the class having a text data. *attribute_name* is the name of a text attribute.

OOSQL_BuildTextIndex finds the file, for which an index is constructed, in the directory appointed in the environmental variable of ODYS_TEMP_PATH.

Example

If you want to construct an index for TEXT_Newspaper_title_SortedPosting that is obtained from the title attribute in Newspaper class in /OOSQL/test.vol volume, you should execute the following command.

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

2.10.6. OOSQL_UpdateTextDescriptor

Usage

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

Description

In an object with text type attributes, there exist text descriptors which are information about each text type attribute. OOSQL_UpdateTextDescriptor updates the field of the text descriptors that indicates whether text index is created or not in all objects that exist in the given class. *database_name* is the name of the DB containing the data, and *volume_name* is the name of the volume containing the data. *class_name* is the name of the class having a text data.

Example

If you want to update all text descriptors of Newspaper class in /OOSQL/test.vol volume, you should execute the following command.

```
> OOSQL_ UpdateTextDescriptor testdb testdb Newspaper
```

2.11. OOSQL_LoadDB

Usage

OOSQL_LoadDB *database_name volume_name input_file_name*

Description

Inserts the data in the input file into the given DB. *database_name* is the name of the DB where the data will be inserted, and *volume_name* is the name of the volume where the data will be inserted. *input_file_name* is the name of the input file having the data. This file is a text file according to the input file format. Refer to the next page for details of the input file format.

OOSQL_LoadDB records text number/OID of the newly created objects on the file of $ODYS_TEMP_PATH/TEXT_<class_name>_OID.

Input File Format used by OOSQL_LoadDB

- Comments

Begins with two hypens.

Eg.: -- This is a comment

- Command Lines

Syntax

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

Description

Specifies the name of the attribute defined in the class. %class is located ahead of all the data lines which are explained later. You should write all data in the order of attributes specified in %class.

Example

%class person (name age)

The example specifies that attributes of name and age are used in the class named person. Apply the information stored in schema for the attribute type.

- Data lines

Syntax

Array the data conforming to each attribute in the order as stated in %class.

Refer to the next chapter for how to specify the type of each data.

Description

Arranges the data to be filled in the attributes of the given class

Example

```
%class person (name age)
‘smith’          31
‘newman’         33
‘jones’          31
‘underwood’      47
```

■ Method of specifying the type of each data

Data type	Example
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 }
MULTISET	{ 1 1 2 2 3 }
LIST (SEQUENCE)	{ ‘one’, ‘two’, ‘three’ }

SET, MULTISET, and LIST will be supported soon.

- Object Reference

Syntax

@class_ref|instance_no

Description

Refers to the given object. class_name or class_id can be adopted for class_ref. class_ref must follow @ mark. | mark devides class_ref and instance_no. Any space is not allowed among them.

Example 1

@person|28

Refers to the object with instance No. 28 in person class.

Example 2

```
%class person (name age)
1: ‘steve’          32
2: ‘joe’            33
```

3: 'mary'	45
'sarah'	23

For appointing instance number, you should allocate a positive number and add : mark just next to it. This number should be unique in a single class.

Example 3

```
%class automobile (make owner)
'Ford'            @person|1
'Mazda'           @person|3
'Jeep'            @person|2
```

The above examples mean that you should refer to the 1st instance of person class with the owner of 'Ford', the 3rd instance of person class with the owner of 'Mazda', and the 2nd instance of person class with the owner of 'Jeep'. Object id of the reference instance is actually stored in DB.

3. OOSQL Statements

3.1. OOSQL Entire Statements

The following are simple definitions of the queries that are available for OOSQL.

```
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 Query

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

Creates a new table or class that has the given attributes. You can define a new table or class inheriting from the definitions of existing tables or classes. In this case, you can list them in parent-table-name-list or parent-class-name-list.

In case you store data in a table temporarily, you can use a temporary table. To use a temporary table, you should specify that the file is temporary when you create the file. A temporary table is destroyed if the transaction is completed. Since accessing a temporary table is faster than accessing an ordinary table, the temporary table is useful for storing the query results temporarily.

Attributes composing a new table or class are represented as a list of <column-identifier, data-type>. column-identifier is the name of the attribute, and data-type is the type of the attribute. The following are the data types available for OOSQL. You can omit the class name in the OID type. See the Path Expressions section, 3.13, for detailed examples.

type Name	Example

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 Query

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

Use the Alter command to change the definition of a table or class. You can change only the leaf table or class in the inheritance hierarchy.

Use the ADD statement to add a new column. To assign the column attributes, you use the same list of <column-identifier, data-type> as it is used in the Create Table command.

Use the DROP statement to delete existing columns. You have two methods of specifying columns: use the DROP COLUMN statement for one column, or use the list of column-identifier for two or more columns.

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 Query

Syntax

drop-table-statement ::=

DROP TABLE base-table-name |

DROP CLASS base-class-name

Description

Drops the definition of the given table.

Example

drop table Employee

3.5. Create Index Query

Syntax

create-index-statement ::=

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

Description

Creates B^+-Tree index or MLGF index in the table. If the key value composing the index is unique in the table, you should use UNIQUE. If you want to order the objects in the table according to the order of the index keys, you should use CLUSTER. If you want to create MLGF index, you should use the keyword MLGF. When the keyword MLGF is not specified, B^+-Tree index is created basically. The position where the index is created is appointed by the names of the table and the attributes.

Example

create cluster index employee_id_index on Employee(id)

3.6. Drop Index Query

Syntax

drop-index-statement ::=

DROP INDEX index-name

Description

Drops the definition of the given index.

Example

drop index employee_id_index

3.7. Create Sequence Query

Syntax

create-sequence-statement ::=

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

Description

Use the Create Sequence command to create the sequence. The sequence is a database object, where you can create a unique value automatically. You can use the sequence object to create values for a primary key. The created sequence has globally unique values, so you can use a sequence for one or more tables.

You can access the sequence value by using <SEQUENCE_NAME>.CURRVAL (it returns the current value of the sequence) or <SEQUENCE_NAME>.NEXTVAL (it returns the next value of the sequence) in the SQL statement.

You can assign the start value of the sequence with the START WITH statement; otherwise it is assigned with 0. If it is 0, the NEXTVAL returns 1.

See the sections 3.10 and 3.11 for examples of using the sequence created in the Insert command and Update command.

Example

create sequence eseq

create sequence eseq start with 100

insert into Employee (id, name) values (eseq.nextval, ‘John’)

```
update Employee set id = eseq.nextval where name = ‘John’
```

3.8. Drop Sequence Query

Syntax

```
drop-sequence-statement ::=
        DROP SEQUENCE sequence-identifier
```

Description

Use the Drop Sequence command to delete the definition of the sequence.

Example

```
drop sequence eseq
```

3.9. Select Query

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

Fetch the objects whose values meet the given condition, from DB. You can make a group of them or fix an order using GOURP BY, HAVING, and ORDER BY. SELECT clause can have the attribute names of the table where the objects will be fetched, or AGGREGATE functions. FROM clause has the name of the table where the objects will be fetched. WHERE clause has the condition that the objects to be fetched should meet. GROUP BY clause has the value of the attribute that makes the results be a group, and HAVING clause has the conditions that each group should meet. ORDER BY clause has the names of the attributes that determine the order of the query results to be output, or AGGREGATE functions. LIMIT clause has the number of tuples to return as query result. Limit clause is not in SQL 99 standard.

For reading the attribute value from the object that has the given OID, you should use SELECT FROM OBJECT. oid-string is the string that has been made from OID of the object to be read through OOSQL_OIDToOIDString. You can obtain the OID of an object from OOSQL_GetOID or the following query.

```
select   e
from     Employee e
```

Using OID obtained from the above query, you can get the value of the attribute as follows.

```
select   *
from     object ‘00000560000A00000000000000000064’
```

In OOSQL, you can use MATCH function for searching the text information search. MATCH is the function used to search the text that qualifies the given ir-expression composed of keywords. The following is the query that uses MATCH function to search for Newspapers, which have the keyword of “Computer” in contents, and outputs the contents.

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

The usage of MATCH is as follows.

```
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
```

The 1st argument of MATCH, column-identifier, has the name of the attribute to be searched for. The 2nd argument, ir-expression, has the keyword to be searched for and the text information formula composed of the operators that describe the relationship among keywords. The following list shows the operators for text information and the meanings.

Operator	Meaning
&	Means that two keywords on both sides of this operator must exist in the text. It returns the minimum value of the ranks of the two keywords. Eg.) Multimedia & database
|	Means that more than one of the two keywords on both sides of this operator must exist in the text. It returns the maximum value of the ranks of the two keywords. Eg.) Multimedia | database
-	Returns the value found by subtracting the rank of the right side keyword from the rank of the left side keyword. Eg.) Multimedia – database
*	Returns the value found by multiplying the rank of the keyword and the given constant value together. Eg.) Multimedia * 3
>	Returns 0, in case the rank of a keyword is below the given constant value. Eg.) Multimedia > 50
:	Returns the texts amounting the given number in the order of ranks. Eg.) Multimedia : 10
^n ~n	Proximity Operator: Checks whether the keywords on either side are within the given distance. ~ checks the distance regardless of the order of the keywords. ^ checks the distance together with the order

of the keywords.

An arbitrary Boolean expression can be used on either side of the operator. For example, an expression (A and B) ~2 C (which means (A ~2 C) and (B ~2 C)) is valid.

However, the * operator cannot be used in the operand keyword. Using the * operator is logically well defined, but incurs a lot of computing overhead. Thus, it is not implemented in the system.

Eg.) multimedia ^2 system

The 3rd argument, lable-id, appoints an unique number for each MATCH function. It is used as an argument of WEIGHT function that returns the value of MATCH function. The following is an example of the query that outputs the result values of two MATCH functions.

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

The 4th argument, scan-direction, appoints the order of retrieval of results. FORWARD is used to retrieve results in the same order as they were inserted into the database. BACKWARD is used to retrieve results in the reversed order. Default is FORWARD. You should not mingle FORWARD and BACKWARD within one query with multiple MATCH functions. The following is an example of the query that retrieves results in the reversed order.

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

Example

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

3.10. Insert Query

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

Inserts the given values or query results into the given table. table-name is the name of the table where you insert the value, and column-identifier is the name of the attribute where you insert the value. insert-value is the value to be actually inserted. You can specify CURRVAL (the current value of the sequence) or NEXTVAL (the next value of the sequence) of sequence as an insert-value.

If the value to be inserted is the text type, you have to write the keyword of 'text' ahead of it. If you want to reflect immediately on the index while inserting the text, you should write 'text' or 'text immediate'. Otherwise, you should write 'text deferred'. If you want to make an index for the text which is deferred, you should use OOSQL_Text_MakeIndex.

When you appoint the value to be inserted using OOSQL_PutData instead of giving it in the query, you can apply '?' for insert-value.

Example

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

3.11. Update Query

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

Modifies the values of the objects, which meet the given condition, using the given expression, or modifies the values of the objects, which have the given OID, using the given expression. table-name is the name of the table that contains the attribute to be modified, and column-identifier is the name of the attribute to be modified. expression is a numerical expression that can calculate the value to be modified, and search-condition is the condition that the object to be modified should meet. You can specify CURRVAL (the current value of the sequence) or NEXTVAL (the next value of the sequence) of sequence as an expression. oid-string is the string that has been made from OID of the object to be modified through OOSQL_OIDToOIDString.

When you appoint the value to be modified using OOSQL_PutData instead of giving it in the query, you should use '?' expression.

Example

```
update Employee set name = 'Steve' where id = 20

update Employee set id = eseq.nextval where name = 'Steve'

update Employee set name = ? where id = 20

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

3.12. Delete Query

Syntax

```
delete-statement ::=
        DELETE FROM table-name [WHERE search-condition] |
        DELETE FROM OBJECT oid-string
```

Description

Deletes the objects, which meet the given condition or have the given OIDs, from the table. table-name is the name of the table that contains the objects to be deleted, and search-condition is the condition that the objects to be deleted should meet. oid-string is the string that has been made from OID of the object to be deleted through OOSQL_OIDToOIDString.

Example

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

3.13. Path Expressions

Description

In OOSQL, the path expression is used for navigating a specific object of complex objects that are linked by OIDs. In a path expression, an attribute having the OID type is followed by DOT(.), and it allows to read the attribute of objects referenced by the OID type attribute. The length of a path expression increases as the number of objects involved in the expression does.

For example, we have the following query: “print city names where employee’s spouses, whose incomes are more than $10,000, live”. In the query, the following path expressions can be specified: “e.spouse.address.city” and “e.spouse.salary”. In “e.spouse.address.city”, “e” is an Employee object; “e.spouse” specifies the spouse of the Employee object. The spouse attribute has the OID type pointing to the Employee object; i.e. the OID(Employee) type. Since the spouse attribute has an OID type, in the path expression, we can read attributes of the object referenced by the spouse attribute. The “e.spouse.address” is a representation where the address attribute of the object referenced by the spouse attribute is accessed. The “address” attribute has an OID type that references an Address object; i.e. the OID(Address) type. Since “address” also has an OID type, it allows reading attributes of the object referenced by the address attribute. The “e.spouse.address.city” reads the city attribute of the object referenced by address. The “e.spouse.salary” path expression also can be interpreted similarly.

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

In a path expression, the kind of class pointed by an OID attribute is determined when the schema is created. However, in some cases, the type of the class is dynamically determined when the query is executed; it is called “domain substitution”. For domain substitution, in path expression, the desired class name is represented in the “[<class name>]” form, and it immediately follows the corresponding OID attribute. For example, if we apply domain substitution to the “e.spouse.address.city” expression, it becomes “e.spouse[Employee].address[Address].city”.

Example

■ Schema Creation

```
// Define Address Class
create class Address (
        city        varchar(100),
        zip         char(10)
);

// Define Employee Class
create class Employee (
        name        varchar(100),
        salary      integer,
        address     OID(Address)
);

// Add the spouse attribute to the Employee class. The OID type referencing
// Employee cannot be used until the Employee class is created.
// Thus, it must be added after the Employee class has been created.
alter class Employee add (spouse OID(Employee));
```

■ Data Insertion

```
// Insert an object for Employee “tom"
insert into Employee (name, salary) values("tom", 100);

// Insert an object for Employee "jane"
insert into Employee (name, salary) values("jane", 10000);

// Insert an object for Address "LA"
insert into Address (city, zip) values ("LA", "111-111");
```

```
// Get the OID of Employee "tom"
// Let the OID of Employee “tom” be assigned to oid_string_of_tom.
select Employee from Employee where name = "tom";

// Get the OID of Employee "jane" -> oid_string_of_jane
select Employee from Employee where name = "jane";

// Get the OID of Address "LA" -> oid_string_of_LA
select Address from Address where city = "LA";

// Make Employee “jane” the spouse of Employee “tom”
// Make Address “LA” the address of Employee “tom”
update object 'oid_string_of_tom' set spouse='oid_string_of_jane',
                                       address = 'oid_string_of_LA';

// Make Employee “tom” the spouse of Employee “jane”
// Make Address “LA” the address of Employee “jane”
update object 'oid_string_of_jane' set spouse='oid_string_of_tom',
                                       address = 'oid_string_of_LA';
```

- **Query String**

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

- **Query Results**

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