- 기타 변경사항

Altibase® Application Development

CLI User's Manaul

Altibase Application Development Altibase CLI User's Manaul

Release 7.1

㈜알티베이스

08378 서울시 구로구 디지털로 306 대륭포스트타워II 10층

전화: 02-2082-1114 팩스: 02-2082-1099

고객서비스포털: http://support.altibase.com

homepage: http://www.altibase.com

2.Altibase CLI 함수

SQLGetInfo

애플리케이션에 접속한 DBMS의 일반적인 정보를 반환한다.

Unicode SQLGetInfoW() 동작은 SQLGetInfo()와 동일하다.

구 문

```
SQLRETURN SQLGetInfo(
        SQLHANDLE    ConnectionHandle,
        SQLUSMALLINT InfoType,
        SQLPOINTER   InfoValuePtr,
        SQLSMALLINT  BufferLength,
        SQLSMALLINT  *StringLengthPtr );
```

인 자

자료유형	인자	사용	설명
SQLHANDLE	ConnectionHandle	입력	데이터베이스 연결 핸들
SQLUSMALLINT	InfoType	입력	검색하려는 정보의 유형. ODBC 표준에서 제공하는 정보 유형 외에 Altibase 전용인 다음의 값도 사용할 수 있다. ALTIBASE_PROTO_VER: Altibase CLI 드라이버와 접속한 Altibase 간의 통신 프로토콜 버전을 문자열로 반환받기 위한 정보 유형.
SQLPOINTER	InfoValuePtr	출력	원하는 정보를 돌려줄 버퍼의 포인터 정보 유형에 따라 다음 5가지 중 하나로 반환된다. 16 bit integer value 32 bit integer value 32 bit binary value 32 bit mask Null 종료 문자 스트링
SQLSMALLINT	BufferLength	입력	InfoValuePtr이 가리키는 버퍼의 바이트 크기
SQLSMALLINT *	StringLengthPtr	출력	InfoValuePtr이 가리키는 결과값의 바이트 길이

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_ERROR
SQL_INVALID_HANDLE
```

설 명

SQLGetInfo()는 DBMS의 일반적인 정보를 얻기 위해 사용한다. 이 함수는 InfoType에 따라 각각의 유형별로 관련 정보를 획득할 수 있으며, Altibase는 ODBC 표준을 준수한다.

진 단

SQLSTATE	설명	부연설명
01004	데이터 잘림	반환 될 값의 크기가 주어진 버퍼의 크기보다 클 경우
08003	연결이 끊김	Connection이 연결되지 않은 상태
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY090	유효하지 않은 인자 사용	이름 길이 인자들 중 하나의 값이 0보다 작거나 SQL_NTS와 같지 않음.
HY096	InfoType의 범위를 벗어남	InfoType에서 지정한 값이 ODBC에서 제공하는 버전에 유효하지 않음

SQLGetPlan

실행 정보를 가져오는 함수이나, 비표준 함수이다.

구 문

```
SQLRETURN SQLGetPlan (
    SQLHSTMT    stmt,
    SQLCHAR**   aPlan);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	입력 명령문 핸들
SQLCHAR**	aPlan	출력	출력 실행 정보를 저장할 포인터

결과값

```
SQL_SUCCESS
SQL_ERROR
```

설 명

비표준 함수이지만, 실행 계획 정보를 가져올 때 사용한다.

이 때 인자 aPlan으로 받은 정보를 수정해서는 안된다.

관련함수

```
SQLSetConnectAttr
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_plan.cpp 참고 >

```
if( SQLSetConnectAttr( dbc, ALTIBASE_EXPLAIN_PLAN,
                          (SQLPOINTER) 1, 0) != SQL_SUCCESS)
.
.
.
    if ( SQLGetPlan(stmt, (SQLCHAR**)&plan) != SQL_SUCCESS )
```

SQLGetStmtAttr

현재 설정되어 있는 명령문 핸들과 관련된 속성 값을 가져온다.

Unicode SQLGetStmtAttrW() 동작은 SQLGetStmtAttr()와 동일하다.

구 문

```
SQLRETURN  SQLGetStmtAttr (
        SQLHSTMT               stmt,
        SQLINTEGER             Attribute,
        SQLPOINTER             param,
        SQLINTEGER      StringLength
        SQLINTEGER *    StringLengthPtr );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLINTEGER	Attribute	입력	설정할 속성. SQLSetStmtAttr() 참조할 것.
SQLPOINTER	param	출력	*Attribute*와 연관된 값의 포인터 *Attribute* 값에 따라 *param*은 32 bit 부호 없는 정수 값이거나 null-terminated 문자열의 포인터, 바이너리 버퍼, 또는 ODBC에서 정의된 값 *Attribute*가 ODBC 고유의 값이면 *param*은 부호 표시 정수임
SQLINTEGER	StringLength	입력	*Attribute*가 ODBC에서 정의된 속성이고 *param*이 문자열 또는 바이너리 버퍼를 가리키면 이 인자는 **param*의 바이트 길이여야만 한다. *Attribute*가 ODBC에서 정의된 속성이고 *param*이 정수면 이 인자는 무시된다.
SQLINTEGER *	StringLengthPtr	출력	**ValuePtr*에 반환되는 값의 바이트 길이(null-termination character 제외)를 반환

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_ERROR
SQL_INVALID_HANDLE
```

설 명

SQLGetStmtAttr()는 명령문 핸들과 관련된 속성을 반환한다. 명령문 속성이나 자세한 내용은SQLSetStmtAttr()를 참조하기 바란다.

진 단

SQLSTATE	설명	부연설명
HY090	유효하지 않은 문자열 또는 버퍼 길이	StringLength가 0보다 작다.
HY092	유효하지 않은 속성 또는 옵션	*Attribute* 인자에 명시된 값이 이 드라이버에서 지원하는 유효한 값이 아니다.
HYC00	구현되지 않은 옵션	*Attribute* 인자에 명시된 값이 이 드라이버에서 지원하는 ODBC에서 유효한 값이나 현재 지원되지 않는다.

관련함수

```
SQLGetConnectAttr
SQLSetConnectAttr
SQLSetStmtAttr
```

SQLGetTypeInfo

DB에서 지원되는 데이터 타입에 관한 정보를 반환한다. Altibase CLI 드라이버는 SQL 실행 결과에 대한 정보를 반환한다. 데이터 타입은 DDL 문에서 사용된다.

구 문

```
SQLRETURN  SQLGetTypeInfo (
        SQLHSTMT               stmt,
        SQLSMALLINT            type );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLSMALLINT	type	입력	SQL 데이터 타입

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLGetTypeInfo()는 Altibase에서 제공하는 데이터 타입에 대한 정보를 표준 결과 집합 형태로 반환한다. 현재 결과 집합은 TYPE_NAME에 대한 오름 차순으로 정렬되어서 반환된다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY001	메모리 할당 오류	Altibase CLI 드라이버가 함수를 실행하고 완료하기 위해 요구된 메모리를 할당할 수 없음

관련함수

```
SQLBindCol
SQLColAttribute
SQLFetch
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_info1.cpp 참고 >

```
if (SQLGetTypeInfo(stmt, SQL_ALL_TYPES) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLGetTypeInfo");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
    if ( rc == SQL_ERROR )
    {
        execute_err(dbc, stmt, "SQLGetTypeInfo:SQLFetch");
        break;
    }
    SQLGetData(stmt, 1, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
    SQLGetData(stmt, 2, SQL_C_SSHORT, &DataType, 0, &cbDataType);
      SQLGetData(stmt, 3, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
      SQLGetData(stmt, 9, SQL_C_SSHORT, &Searchable, 0, NULL);
      SQLGetData(stmt, 14, SQL_C_SSHORT, &MinScale, 0, &cbMinScale);
      SQLGetData(stmt, 15, SQL_C_SSHORT, &MaxScale, 0, &cbMaxScale);
      SQLGetData(stmt, 16, SQL_C_SSHORT, &SQLDataType, 0, &cbSQLDataType);
      SQLGetData(stmt, 18, SQL_C_SLONG, &NumPrecRadix, 0, &cbNumPrecRadix);

      printf("%-20s%10d%10d%10d\t",
szTypeName, DataType, ColumnSize, SQLDataType);
      if ( Searchable == SQL_PRED_NONE )
      {
          printf("SQL_PRED_NONE\n");
      }
      else if ( Searchable == SQL_PRED_CHAR )
      {
          printf("SQL_PRED_CHAR\n");
      }
      else if ( Searchable == SQL_PRED_BASIC )
      {
          printf("SQL_PRED_BASIC\n");
      }
      else if ( Searchable == SQL_SEARCHABLE )
      {
          printf("SQL_SEARCHABLE\n");
      }
}
```

SQLMoreResults

SQLMoreResults는 결과가 더 남아있는가에 대한 결과를 리턴한다.

구문

```
SQLRETURN SQLMoreResults (
        SQLHSTMT               stmt);
```

인자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들

결과값

```
SQL_SUCCESS
SQL_NO_DATA
```

설명

Statement를 Fetch해서 결과 집합을 가져 올 경우 해당 커서의 위치는 결과 집합의 첫 번째 레코드에 있게 된다. 애플리케이션에 따라서는 그 결과 집합을 사용할 때, 그 결과가 더 남아 있는지에 대한 검사를 하게 된다. 결과 값이 남아 있는 경우에는

SQL_SUCCESS가 반환이 되며, 더 이상 결과가 남아 있지 않은 경우에는 SQL_NO_DATA가 반환된다.

결과 집합을 Fetch하면 Cursor의 위치는 first result set에 오게된다. Application에 따라 그 결과 집합을 쓰고나서 결과가 더 남아 있는지 대한 검사를 하게된다.
결과값이 남아 있다면 SQL_SUCCESS가 리턴되며, 더 이상 결과가 남아 있지 않은 경우 SQL_NO_DATA가 리턴된다.

관련함수

```
SQLFetch
```

SQLNativeSql

SQL 구문을 Altibase CLI 드라이버가 지원하는 문장으로 변환한다.

Unicode SQLNativeSqlW() 동작은 SQLNativeSql()와 동일하다.

구 문

```
SQLRETURN SQLNativeSql (
                SQLHDBC dbc,
                SQLCHAR *InstatementText,
                SQLINTEGER textLength,
                SQLCHAR *OutStstementText,
                SQLINTEGER bufferLength,
                SQLINTEGER textLength);
```

인 자

자료유형	인자	사용	설명
SQLHDBC	dbc	입력	접속 핸들
SQLCHAR *	InstatementText	입력	변환하고자 하는 SQL 구문
SQLINTEGER	textLength	입력	InstatementText의 길이
SQLCHAR *	OutStstementText	출력	변환된 SQL 구문을 받을 버퍼의 포인터
SQLINTEGER	bufferLength	입력	OutStstementText의 크기
SQLINTEGER	textLength	출력	OutStstementText에 저장된 크기

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQL 구문을 Altibase CLI 드라이버가 지원하는 문장으로 변환시켜준다.
SQLNativeSql() 실행은 하지 않는다.

ALTIBASE는 다음과 같은 구문들을 지원한다.

- {d, '1981-09-27'} → TO_DATE('1981-09-27', 'YYYY-MM-DD')
- {t, '07:23:56'} → TO_DATE('07:23:56', 'HH:MI:SS')
- {ts, '1981-09-27 07:23:56'} → TO_DATE('1981-09-27 07:23:56', 'YYYY-MM-DD HH:MI:SS')
- {? = call PROC1(a, ?, c)} → EXECUTE ? := PROC1(a, ? ,c)

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	명시적인 오류 발생 없음
HY001	메모리 할당 오류	명시된 핸들을 위한 메모리 할당에 실패
01004	자료가 잘림	OutStstementText 버퍼의 크기가 반환될 데이터의 크기보다 작음

예 제

```
SQLNativeSql(  dbc,
               (SQLCHAR*)"INSERT INTO T1 VALUES( {d '1981-09-27'} )",
               SQL_NTS,
               outText,
               100,
               &outTextLen);
outText에 INSERT INTO T1 VALUES( TO_DATE('1981-09-27', 'YYYY-MM-DD'))가 저장된다.
```

SQLNumParams

SQL 문에서의 매개변수의 개수를 반환한다.

구 문

```
SQLRETURN  SQLNumParams (
        SQLHSTMT                  stmt,
        SQLSMALLINT *          parameterCounterPtr );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLSMALLINT *	parameterCountPtr	출력	매개변수의 개수의 포인터

결과값

```
SQL_SUCCESS
```

설 명

이 함수는 SQLPrepare()가 호출된 후에만 호출할 수 있다.

만약 stmt와 관련된 명령문이 매개변수들을 포함하지 않으면, SQLNumParams()는
**parameterCountPtr*에 0을 설정한다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	

관련함수

```
SQLBindParameter
SQLDescribeParam
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_info2.cpp 참고 >

```
SQLPrepare(hstmt, Statement, SQL_NTS);

// Check to see if there are any parameters. If so, process them.
SQLNumParams(hstmt, &NumParams);
if (NumParams) {
   // Allocate memory for three arrays. The first holds pointers to buffers in which
   // each parameter value will be stored in character form. The second contains the
   // length of each buffer. The third contains the length/indicator value for each
   // parameter.
   PtrArray = (SQLPOINTER *) malloc(NumParams * sizeof(SQLPOINTER));
   BufferLenArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));
   LenOrIndArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));

   for (i = 0; i < NumParams; i++) {
   // Describe the parameter.
   SQLDescribeParam(hstmt, i + 1, &DataType, &ParamSize, &DecimalDigits, &Nullable);

   // Call a helper function to allocate a buffer in which to store the parameter
   // value in character form. The function determines the size of the buffer from
   // the SQL data type and parameter size returned by SQLDescribeParam and returns
   // a pointer to the buffer and the length of the buffer.
   PtrArray[i] = (char*)malloc(ParamSize);
   BufferLenArray[i] = SQL_NTS;


   // Bind the memory to the parameter. Assume that we only have input parameters.
   SQLBindParameter(hstmt, i + 1, SQL_PARAM_INPUT, SQL_C_CHAR, DataType, ParamSize,
         DecimalDigits, PtrArray[i], BufferLenArray[i],
         &LenOrIndArray[i]);

   // Prompt the user for the value of the parameter and store it in the memory
   // allocated earlier. For simplicity, this function does not check the value
   // against the information returned by SQLDescribeParam. Instead, the driver does
   // this when the statement is executed.
   strcpy((char*)PtrArray[i], "AAAAAAA");
   BufferLenArray[i] = 7;
   }
}
```

SQLNumResultCols

결과 집합의 열의 개수를 반환한다.

구 문

```
SQLRETURN  SQLNumResultCols (
        SQLHSTMT                    stmt,
        SQLSMALLINT *           col );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLSMALLINT *	col	출력	결과 집합의 열의 개수를 저장할 포인터

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

만약 stmt와 관련된 명령문이 열들을 반환하지 않으면, SQLNumResultCols()는 *col에 0을 설정한다.이 함수를 호출하기 전에 SQLPrepare() 또는 SQLExecDirect()를 호출해야 한다.

이 함수를 호출한 후에 SQLDescribeCol(), SQLColAttribute(), SQLBindCol() 또는 SQLGetData()를 호출할 수 있다.
SQLNumResultCols()는 단지 명령문이 준비, 실행 또는 지정된 상태일 때 성공적으로 호출될 수 있다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

SQLNumResultCols()가 SQLPrepare() 후, 그리고 SQLExecute() 전에 호출될 때 두 함수에 의해서 반환된 어떠한 SQLSTATE도 반환할 수 있다.

관련함수

```
SQLBindCol
SQLColAttribute
SQLDescribeCol
SQLExecDirect
SQLExecute
SQLFetch
SQLGetData
SQLPrepare
SQLSetStmtAttr
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp 참고 >

```
sprintf(query,"SELECT * FROM DEMO_EX1");
if (SQLExecDirect(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLNumResultCols(stmt, &columnCount);
```

SQLParamData

수행중인 명령문에 데이터를 넣을 때 사용한다.

구 문

```
SQLRETURN SQLParamData (
                SQLHSTMT   stmt,
                SQLPOINTER *value);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령어 핸들
SQLPOINTER *	value	출력	SQLBindParameter에서 지정한 주소를 저장할 포인터

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_NEED_DATA
SQL_NO_DATA
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

명령문이 수행중에 데이터를 넣을 때 사용한다.

SQLPutData와 함께 사용한다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	명시적인 오류 발생 없음
HY001	메모리 할당 오류	명시된 핸들을 위한 메모리 할당에 실패
HY010	함수 연속 오류	

관련함수

```
SQLBindParameter
SQLExecDirect
SQLExecute
SQLPutData
```

SQLPrepare

실행을 위한 SQL 문자열을 준비한다.

Unicode SQLPrepareW() 동작은 SQLPrepare()와 동일하다.

구 문

```
SQLRETURN  SQLPrepare (
        SQLHSTMT               stmt,
        SQLCHAR *              sql,
        SQLSMALLINT            sqlLength );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLCHAR *	sql	입력	SQL 텍스트 문자열
SQLSMALLINT	sqlLength	입력	**sql*의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

애플리케이션은 사전 준비를 위해서 SQL문을 DB로 보내기 위해 SQLPrepare()를 호출한다. 애플리케이션은 SQL문에 하나 이상의 매개변수 마커(?)를 포함할 수 있다. 매개변수 마커를 포함하기 위해, 애플리케이션은 적절한 위치에서 SQL 문자열에 매개변수 마커를 삽입한다.

일단 명령문이 준비되면, 애플리케이션은 나중에 함수를 호출하는데 명령문을 참고하기 위해 명령문 핸들을 사용한다. 명령문 핸들과 관련된 준비된 명령문은 애플리케이션이 SQL_DROP 옵션으로 SQLFreeStmt()를 호출함으로서 명령문을 해제할 때까지 또는 명령문 핸들이 SQLPrepare(), SQLExecDirect(), 또는 카탈로그 함수들 (SQLColumns(), SQLTables(), 등등) 중 하나가 호출로 인해 사용될 때까지 SQLExecute()를 호출함으로서 다시 실행될 수 있다. 일단 애플리케이션이 명령문을

준비하면, 애플리케이션은 결과 집합의 형식에 대한 정보를 요구할 수 있다. 몇 몇 구현에 있어서는 SQLPrepare() 후에 SQLDescribeCol()을 호출하는 것은 SQLExecute() 또는 SQLExecDirect() 후에 호출하는것 만큼 효과적이지 않을 수 있다.

ODBC 데이터 타입은 명령문이 실행(모든 매개변수가 바인드 되지 않았다면) 될 때 검사된다. 최대한의 상호운용을 위해, 애플리케이션은 같은 명령문에서 새로운 SQL 문을 준비하기 전에 이전 SQL 문에 적용된 모든 매개변수들을 언바운드 해야만 한다. 이것은 새로운 정보에 적용되고 있는 이전 매개변수 정보 때문에 발생하는 오류들을 예방한다.

진 단

SQLSTATE	설명	부연설명
08003		*stmt*가 연결 되지 않은 상태거나 연결이 끊어진 상태
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY001	메모리 할당 오류	Altibase CLI 드라이버가 함수를 실행하고 완료하기 위해 요구된 메모리를 할당할 수 없음
HY009	유효하지 않은 인자 사용 (null pointer)	*sql*이 null pointer 임

관련함수

```
SQLAllocHandle
SQLBindCol
SQLBindParameter
SQLEndTran
SQLExecDirect
SQLExecute
SQLRowCount
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp 참고 >

```
sprintf(query,"INSERT INTO DEMO_EX2 VALUES( ?, ?, ?, ?, ?, ? )");

/* Statement를 준비하고 변수를 바인드한다. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLPrimaryKeys

단일 테이블의 프라이머리 키를 구성하는 열 이름들을 결과 집합 형태로 반환한다. 이 함수는 단일 호출에서 다중 테이블들의 프라이머리 키들을 반환하는 것은 지원하지 않는다.

Unicode SQLPrimarykeysW() 동작은 SQLPrimarykeys()와 동일하다.

구 문

```
SQLRETURN SQLPrimaryKeys (
                SQLHSTMT        stmt,
                SQLCHAR *       cName,
                SQLSMALLINT     cNameLength,
                SQLCHAR *       sName,
                SQLSMALLINT     sNameLength,
                SQLCHAR *       tName,
                SQLSMALLINT     tNameLength );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	*stmt*	입력	명령문 핸들
SQLCHAR *	*cName*	입력	카탈로그 이름
SQLSMALLINT	*cNameLength*	입력	**cName*의 문자 개수
SQLCHAR *	*sName*	입력	스키마 이름
SQLSMALLINT	*sNameLength*	입력	**sName*의 문자 개수
SQLCHAR *	*tName*	입력	테이블 이름, null pointer일 수 없다. *tName*은 문자열 탐색 유형을 포함할 수 없다.
SQLSMALLINT	*tNameLength*	입력	**tName*의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLPrimaryKeys()는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, 그리고 KEY_SEQ에 의해 순서화된 표준 결과 집합 형태로 결과들을 반환한다.

탐색 유형은 스키마 규정자 또는 테이블 이름을 지정하기 위해 사용될 수 없다.

많은 경우에 있어서, SQLPrimaryKeys() 호출이 시스템 카탈로그에 대해 복잡하여 비용이 드는 조회에 맵핑되므로 가끔 사용되어야 하며, 호출을 반복하는 대신 저장된 결과를 사용해야 한다.

다음 테이블은 결과 집합의 열들을 나열한 것이다.

열번호	열 이름	자료유형	설명
1	TABLE_CAT	VARCHAR	항상 NULL이 반환
2	TABLE_SCHEM	VARCHAR	프라이머리 키 테이블 스키마 이름
3	TABLE_NAME	VARCHAR (NOT NULL)	프라이머리 키 테이블 이름
4	COLUMN_NAME	VARCHAR (NOT NULL)	1차 프라이머리 키 열 이름
5	KEY_SEQ	SMALLINT (NOT NULL)	1부터 시작하는 1차 프라이머리 키의 열 순서 번호
6	PK_NAME	VARCHAR	1차 프라이머리 키 이름

[표 2-2] SQLPrimaryKeys()에 의해 반환되는 열

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY009	유효하지 않은 인자 (null pointer) 사용	*tName*이 null pointer 임

관련함수

```
SQLBindCol
SQLFetch
SQLStatistics
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta3.cpp 참고 >

```
if (SQLPrimaryKeys(stmt,
                   NULL, 0,
                   NULL, 0,
                   (char*)"DEMO_META3", SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrimaryKeys");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

SQLBindCol(stmt, 1, SQL_C_CHAR, szCatalog, STR_LEN,&cbCatalog);
SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
SQLBindCol(stmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
SQLBindCol(stmt, 5, SQL_C_SSHORT, &KeySeq, 0, &cbKeySeq);
SQLBindCol(stmt, 6, SQL_C_CHAR, szPkName, STR_LEN, &cbPkName);
while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
    if ( rc == SQL_ERROR )
    {
        execute_err(dbc, stmt, "SQLPrimaryKeys:SQLFetch");
        break;
    }
    printf("%-20s%-20s%-3d%-20s\n", szTableName,
                szColumnName, KeySeq, szPkName);
}
```

SQLProcedureColumns

명시된 프로시저에 대해 결과 집합을 구성하고 있는 열들 뿐만 아니라 입력과 출력 매개변수들의 목록을 결과 집합의 형태로 반환한다.

Unicode SQLProcedureColumnsW() 동작은 SQLProcedureColumns()와 동일하다.

구 문

```
SQLRETURN  SQLProcedureColumns (
        SQLHSTMT                stmt,
        SQLCHAR *               cName,
        SQLSMALLINT     cNameLength,
        SQLCHAR *               sName,
        SQLSMALLINT     sNameLength,
        SQLCHAR *               pName,
        SQLSMALLINT     pNameLength,
        SQLCHAR *               colName,
        SQLSMALLINT         colNameLength );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLCHAR *	cName	입력	프로시저 카탈로그 이름
SQLSMALLINT	cNameLength	입력	**cName*의 문자 개수
SQLCHAR *	sName	입력	프로시저 스키마 이름
SQLSMALLINT	sNameLength	입력	**sName*의 문자 개수
SQLCHAR *	pName	입력	프로시저 이름, null pointer일 수 없다. *pName*은 문자열 탐색 유형을 포함할 수 없다.
SQLSMALLINT	pNameLength	입력	**pName*의 문자 개수
SQLCHAR *	colName	입력	열 이름
SQLSMALLINT	colNameLength	입력	**colName*의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

이 함수는 프로시저 매개변수들과 프로시저에 의해 반환된 결과 집합(들)을 구성하는
열들을 검색하기 위하여 명령문 실행 전에 사용된다.

SQLProcedureColumns()은 PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, 그리고
COLUMN_TYPE에 의해 순서화된 표준 결과 집합 형태로 결과들을 반환한다.

열 이름들은 각 프로시저에 대해 다음 순서로 반환된다.

- 반환 값의 이름
- 프로시저를 호출하는데 있어서 각 매개변수의 이름들(호출 순서)
- 프로시저에서 반환된 결과 집합의 각 열의 이름들 (열 순서)

SQLProcedureColumns()에 의해 반환되는 열은 [표 2-3]과 같다.

이름	번호	자료 유형	설명
PROCEDURE_CAT	1	VARCHAR	항상 NULL 반환
PROCEDURE _SCHEM	2	VARCHAR	프로시저 스키마 이름; DB에 적절하지 않은 경우 NULL
PROCEDURE _NAME	3	VARCHAR (NOT NULL)	프로시저 이름
COLUMN_NAME	4	VARCHAR (NOT NULL)	프로시저 열 이름. Altibase CLI 드라이버는 이름을 갖지 않은 열에 대해 빈 문자열을 반환한다.
COLUMN_TYPE	5	SMALLINT (NOT NULL)	매개변수 또는 결과 집합 열로서 프로시저 열을 정의: SQL_PARAM_INPUT: 프로시저 열이 입력 매개변수임 SQL_PARAM_INPUT_OUTPUT: 프로시저 열이 입/출력 매개변수임 SQL_PARAM_OUTPUT: 프로시저 열이 출력 매개변수임
DATA_TYPE	6	SMALLINT (NOT NULL)	SQL 데이터 타입
TYPE_NAME	7	VARCHAR (NOT NULL)	DB에 대응하는 데이터 타입의 이름
COLUMN_SIZE	8	INTEGER	열의 크기. 열의 크기가 적합치 않으면 NULL 이 반환됨
BUFFER_LENGTH	9	INTEGER	자료를 저장하는 최대 바이트
DECIMAL_DIGITS	10	SMALLINT	열의 소수자리 수, 소수자리 수를 적용할 수 없는 데이터 타입은 널이 반환
NUM_PREC_RADIX	11	SMALLINT	Numeric 데이터 타입인 경우 10: COLUMN_SIZE와 DECIMAL_DIGITS의 값은 이 열에 허용되는 십진 자릿수가 주어진다. 예를 들어 DECIMAL(12,5) 열은 NUM_PREC_RADIX가 10, COLUMN_SIZE가 12 그리고 DECIMAL_DIGITS의 값 5를 반환할 수 있다.
NULLABLE	12	SMALLINT (NOT NULL)	프로시저 열이 널 값을 허용하지 않는 경우 SQL_NO_NULLS 허용 할 경우 SQL_NULLABLE
REMARKS	13	VARCHAR	프로시저 열에 대한 설명 정보

이름	번호	자료 유형	설명
COLUMN_DEF	14	VARCHAR	열의 디폴트 값. 이 열이 작은 따옴표로 묶여 있다면 이 값은 스트링, 널이 디폴트로 지정되어 있으면 이 열은 작은 따옴표로 묶인 단어가 아니라 단어 NULL을 반환, 디폴트 값이 잘림이 없이 나타낼 수 없다면 이 열은 작은 따옴표가 없는 TRUNCATED를 포함, 디폴트 값이 지정 되어있지 않으면 이 열은 NULL COLUMN_DEF 값은 새로운 열 정의를 생성하는데 사용될 수 있다. (TRUNCATED 값을 가지고 있을 때를 제외)
SQL_DATA_TYPE	15	SMALLINT (NOT NULL)	SQL 데이터 타입
SQL_DATETIME_SUB	16	SMALLINT	Date 데이터 타입을 위한 subtype 코드. 이외의 데이터 타입인 경우 ø이 반환
CHAR_OCTET_LENGTH	17	INTEGER	문자나 바이너리 데이터 타입 열의 최대 자릿수
ORDINAL_POSITION	18	INTEGER (NOT NULL)	프로시저 정의에 있어서 매개변수의 순서 위치 (1부터 증가)
IS_NULLABLE	19	VARCHAR	NO : 열이 널들을 포함하지 않을 때. YES : 열이 널들을 포함할 때

[표 2-3] SQLProcedureColumns()에 의해 반환되는 열

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	
HY009	유효하지 않은 인자 (null pointer) 사용	*cName*이 null pointer 임

관련함수

```
SQLBindCol
SQLFetch
SQLProcedures
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta6.cpp 참고 >

```
if (SQLProcedureColumns(stmt,
                        NULL, 0,
                        NULL, 0,
                        "DEMO_META6_PROC", SQL_NTS,
                        NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLProcedureColumns");
      SQLFreeStmt(stmt, SQL_DROP);
      return SQL_ERROR;
}

SQLBindCol(stmt, 1, SQL_C_CHAR, szCatalog, STR_LEN,&cbCatalog);
  SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
  SQLBindCol(stmt, 3, SQL_C_CHAR, szProcName, STR_LEN,&cbProcName);
  SQLBindCol(stmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
  SQLBindCol(stmt, 5, SQL_C_SSHORT, &ColumnType, 0, &cbColumnType);
  SQLBindCol(stmt, 7, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
  SQLBindCol(stmt, 8, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
  SQLBindCol(stmt, 10, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
  SQLBindCol(stmt, 11, SQL_C_SSHORT, &NumPrecRadix, 0, &cbNumPrecRadix);
  SQLBindCol(stmt, 18, SQL_C_SLONG, &OrdinalPosition, 0, &cbOrdinalPosition);
```

SQLProcedures

특정 테이터 소스에 저장된 프로시저 이름들의 목록을 반환한다.

Unicode SQLProceduresW()의 동작은 SQLProcedures()와 동일하다.

구 문

```
SQLRETURN  SQLProcedures (
        SQLHSTMT                stmt,
        SQLCHAR *               cName,
        SQLSMALLINT     cNameLength,
        SQLCHAR *               sName ,
        SQLSMALLINT     sNameLength,
        SQLCHAR *               pName,
        SQLSMALLINT     pNameLength );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLCHAR *	cName	입력	프로시저 카탈로그 이름
SQLSMALLINT	cNameLength	입력	**cName*의 문자 개수
SQLCHAR *	sName	입력	프로시저 스키마 이름
SQLSMALLINT	sNameLength	입력	**sName*의 문자 개수
SQLCHAR *	pName	입력	프로시저 이름, null pointer일 수 없다. *pName*은 문자열 탐색 유형을 포함할 수 없다.
SQLSMALLINT	pNameLength	입력	**pName*의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLProcedures()는 요구된 범위 내에서 모든 프로시저의 목록을 나열한다. 사용자는
이러한 프로시저들을 실행할 수 있는 사용 권한을 가질 수도 있고 없을 수도 있다.

SQLProcedures()는 PROCEDURE_CAT, PROCEDURE_SCHEM, 그리고 PROCEDURE_NAME에 의해
순서화된 표준 결과 집합 형태로 결과들을 반환한다.

> Note: SQLProcedures()는 모든 프로시저들을 반환하지 않을 수도 있다.
> 애플리케이션은 프로시저가 SQLProcedures()에 의해 반환 여부와 상관없이 유효한
> 프로시저를 사용할 수 있다. SQLProcedures()에 의해 반환되는 열은 [표 2-4]와 같다.

열 이름	열번호	자료 유형	설명
PROCEDURE_CAT	1	VARCHAR	항상 NULL이 반환
PROCEDURE _SCHEM	2	VARCHAR	프로시저 스키마 식별자; DB에 적절하지 않은 경우 NULL
PROCEDURE _NAME	3	VARCHAR (NOT NULL)	프로시저 식별자
NUM_INPUT_PARAMS	4	N/A	나중 용도로 예약됨. 애플리케이션은 이 결과 열에 반환된 데이터를 적용하면 않된다.
NUM_OUTPUT_PARAMS	5	N/A	나중 용도로 예약됨. 애플리케이션은 이 결과 열에 반환된 데이터를 적용하면 않된다.
NUM_RESULT_SETS	6	N/A	나중 용도로 예약됨. 애플리케이션은 이 결과 열에 반환된 데이터를 적용하면 않된다.
REMARKS	7	VARCHAR	프로시저에 대한 설명 정보

열 이름	열번호	자료 유형	설명
PROCEDURE_TYPE	8	SMALLINT	프로시저 타입 정의 SQL_PT_UNKNOWN: 프로시저가 임의의 값을 반환했는지를 결정할 수 없다. SQL_PT_PROCEDURE: 반환된 객체가 프로시저; 반환 값을 가지고 있지 않다. SQL_PT_FUNCTION: 반환된 객체가 함수; 반환 값을 가지고 있다.

[표 2-4] SQLProcedures()에 의해 반환되는 열

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY009	유효하지 않은 인자 (null pointer) 사용	*cName*이 null pointer 임. *sNme*, *pName*이 null pointer 임

관련함수

```
SQLBindCol
SQLFetch
SQLProcedureColumns
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta5.cpp 참고 >

```
if (SQLProcedures(stmt,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLProcedures");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLPutData

수행중인 명령문에 데이타를 넣을 때 사용한다.

구 문

```
SQLRETURN SQLPutData (
                SQLHSTMT        stmt,
                SQLPOINTER      data,
                SQLLEN      strLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	*stmt*	입력	명령어 핸들
SQLPOINTER	*data*	입력	데이터 버퍼의 포인터
SQLLEN	*strLength*	입력	데이터의 크기

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

명령문을 수행중에 데이터를 넣을 때 사용한다. SQLParamData와 같이 사용한다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	명시적인 오류 발생 없음
HY001	메모리 할당 오류	명시된 핸들을 위한 메모리 할당에 실패
HY010	함수 연속 오류	

관련함수

```
SQLBindParameter
SQLExecDirect
SQLExecute
SQLParamData
```

SQLRowCount

UPDATE, INSERT 또는 DELETE 문에 의해 영향을 받은 행들의 수를 반환한다.

구 문

```
SQLRETURN  SQLRowCount (
        SQLHSTMT               stmt,
        SQLLEN *               row );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLLEN *	row	출력	행의 개수를 저장할 포인터

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

UPDATE, INSERT 또는 DELETE 문에 의해 영향을 받은 행들의 수를 반환한다. 그 외에 MOVE, ENQUEUE, MERGE 처럼 행에 변경시키는 SQL문의 경우에도 영향을 받은 행들의 개수가 **row*에 반환된다. 만약 영향을 받은 행이 없다면 **row*에 0이 반환된다.

입력 명령문 핸들로 마지막 실행한 SQL문이 위에 열거한 SQL 문이 아니거나 성공적으로 실행되지 않은 경우 이 함수는 **row*에 -1이 반환된다.

Cascading delete operation (부모 테이블에 임의의 행이 삭제되면 그것과 관련된 자식 테이블의 모든 행도 삭제됨)과 같은 SQL문에 의해 영향을 받는 임의의 테이블의 행들은 개수에 포함되지 않는다.

이 함수를 호출하기전에 SQLExecute() 또는 SQLExecDirect()를 호출해야만 한다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	

관련함수

```
SQLExecDirect
SQLExecute
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_exa5.cpp 참고 >

```
if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLRowCount(stmt, &row_count);
```

SQLSetConnectAttr

특정 연결에 대한 연결 속성값을 설정한다.

Unicode SQLSetConnectAttrW() 동작은 SQLSetConnectAttr()와 동일하다.

구 문

```
SQLRETURN  SQLSetConnectAttr (
        SQLHDBC         dbc,
        SQLINTEGER      Attribute,
        SQLPOINTER      ValuePtr,
        SQLINTEGER      StringLength );
```

인 자

자료유형	인자	사용	설명
SQLHDBC	dbc	입력	연결 핸들
SQLINTEGER	Attribute	입력	설정할 속성
SQLPOINTER	ValuePtr	입력	속성에 해당하는 값의 포인터 Attribute 값에 따라, ValuePtr은 32-bit unsigned integer 값이거나, null-terminated character string을 가리키는 포인터. SQL_ATTR_AUTOCOMMIT SQL_ATTR_CONNECTION_TIMEOUT SQL_ATTR_PORT SQL_ATTR_TXN_ISOLATION ALTIBASE_APP_INFO ALTIBASE_DATE_FORMAT ALTIBASE_CONN_ATTR_IPC_FILEPATH
SQLINTEGER	StringLength	입력	*ValuePtr*이 character string 일 때 *StringLength*는 문자열의 바이트 길이 또는 SQL_NTS. 32-bit Integer 일 때는 무시한다.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

애플리케이션은 연결이 할당되거나 해제되는 어느 시점에서나 SQLSetConnectAttr()을 호출할 수 있다. 애플리케이션에서 성공적으로 설정된 모든 연결과 명령문 속성들은 현재 연결에서 SQLFreeHandle()이 호출될 때까지 보존된다.

속성	내용
SQL_ATTR_AUTOCOMMIT	연결에 대한 반영 작동을 설정하는 32비트 값. SQL_AUTOCOMMIT_ON: 각각의 SQL문은 실행 시 자동으로 반영. SQL_AUTOCOMMIT_OFF: 각 SQL문을 실행 시 자동으로 반영하지 않는다.

속성	내용
SQL_ATTR_CONNECTION_TIMEOUT	네트워크 불안정 시 select() 또는 poll() 에서 발생할 수 있는 blocking을 방지하기 위한 timeout 값 설정
SQL_ATTR_PORT	Server port 설정 (32-bit Integer)
SQL_ATTR_TXN_ISOLATION	*dbc*에 의해 참조되는 현재 연결에 대한 트랜잭션 분리 레벨을 설정하는 32 비트값.
SQL_ATTR_ENLIST_IN_DTC	DTC(Distributed Transaction Coordinator)로 초기화된 분산 트랜잭션에 참여시킨다. 분산 트랜잭션을 참여하기 위해 속성 값으로 분산 트랜잭션의 객체(ITransaction *)를 설정하거나 SQL_DTC_DONE을 설정한다. SQL_DTC_DONE : 값을 설정하면 분산 트랜잭션에 참여 완료 후 로컬 트랜잭션으로 수행된다.
ALTIBASE_APP_INFO	Altibase CLI 애플리케이션이 가질수 있는 임의의 식별자를 지정하여, 사용자 임의의 세션에 대한 보다 정확한 정보를 알 수 있다.
ALTIBASE_DATE_FORMAT	날짜 형식 설정. 기본값으로 YYYY/MM/DD HH:MI:SS을 지원한다.
ALTIBASE_CONN_ATTR_IPC_FILEPATH	유닉스 환경에서 서버와 클라이언트가 IPC로 접속할 때 ALTIBASE_HOME이 서로 다른 경우, 유닉스 도메인의 소켓 경로가 일치하지 않아 접속할 수 없다. 이 때 ALTIBASE_HOME/trc/cm-ipc 파일을 이용하면, 유닉스 도메인 통신이 가능해져 공유 메모리의 정보를 가져올 수 있다.
ALTIBASE_SOCK_RCVBUF_BLOCK_RATIO	소켓 수신 버퍼의 크기를 32K 단위로 설정한다. 만약 이 속성의 값이 2로 설정되었다면 소켓 수신 버퍼의 크기는 64K가 된다. 기본값은 0이다. TCP kernel parameter 중 최대 소켓 수신 버퍼 크기가 이 속성값에 의해 설정된 소켓 수신 버퍼 크기 미만으로 설정되어 있을 경우, 이 속성 은 OS에 따라 무시되거나 에러를 발생시킬 수 있다. (Linux OS 인 경우, 'net.core.rmem_max' TCP kernel parameter에 해당된다)
ALTIBASE_MESSAGE_CALLBACK	서버로부터 전달되는 메시지를 수신하기 위해 콜백 함수를 등록한다. 사용자는 수신된 메시지를 콜백 함수로 핸들할 수 있으며, 자세한 사용법은 아래 샘플을 참고한다. <$ALTIBASE_HOME/sample/SQLCLI/demo_message.cpp>
ALTIBASE_GLOBAL_TRANSACTION_LEVEL	Sharding 사용시 동작할 트랜잭을 레벨을 지정할수 있다. 트랜잭션 레벨에 대한 자세한 내용은 Sharding 매뉴얼의 샤드 트랜잭션 항목을 참조한다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
25000	로컬 트랜잭션 수행중이므로 연산 불가	NON-AUTOCOMMIT 모드에서 SQL_ATTR_ENLIST_IN_DTC 속성으로 분산 트랜잭션에 참여 시도시 실패
HY000	일반 오류	

*Attribute*가 명령문 속성일 때, SQLSetConnectAttr()은 SQLSetStmtAttr()에 의해
반환된 어떠한 SQLSTATE도 반환할 수 있다.

관련함수

```
SQLAllocHandle
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_tran1.cpp 참고 >

```
if (SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
                      (void*)SQL_AUTOCOMMIT_OFF, 0) != SQL_SUCCESS)
{
    execute_err(dbc, SQL_NULL_HSTMT, "Autocommit OFF");
    return SQL_ERROR;
}
```

SQLSetDescField

descriptor의 속성을 한 개 지정한다.

Unicode SQLSetDescFieldW() 동작은 SQLSetDescField()와 동일하다.

구 문

```
SQLRETURN SQLSetDescField (
                SQLHDESC        desc,
                SQLSMALLINT     recNumber,
                SQLSMALLINT     fieldIdentifier,
                SQLPOINTER      value,
                SQLINTEGER      bufferLength);
```

인 자

자료유형	인자	사용	설명
SQLHDESC	*desc*	입력	Descriptor 핸들
SQLSMALLINT	*renNumber*	입력	칼럼 번호 1부터 시작한다.
SQLSMALLINT	*fieldIdentifier*	입력	가져올 칼럼의 속성을 지정
SQLPOINTER	*value*	입력	속성을 지정할 버퍼 포인터
SQLINTEGER	*bufferLength*	입력	value의 크기

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

descriptor의 속성을 한 개 지정한다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	명시적인 오류 발생 없음
HY001	메모리 할당 오류	명시된 핸들을 위한 메모리 할당에 실패함.
HY010	함수 호출 순서 오류	
07009	유효하지 않은 설명자 인덱스	recNumber의 값이 잘못됨

관련함수

```
SQLBindCol
SQLBindParameter
SQLGetDescField
SQLGetDescRec
```

SQLSetEnvAttr

현재 환경에 대한 환경 속성을 설정한다.

구 문

```
SQLRETURN  SQLSetEnvAttr (
        SQLHENV         env,
        SQLINTEGER      Attribute,
        SQLPOINTER      Value,
        SQLINTEGER      StringLength );
```

인 자

자료유형	인자	사용	설명
SQLHENV	env	입력	환경 핸들
SQLINTEGER	Attribute	입력	설정 할 환경 속성
SQLPOINTER	Value	입력	*Attribute*와 관련된 값의 포인터. *Attribute* 값에 따라, *Value*는 32-bit unsigned integer 값이거나, null-terminated character string을 가리키는 포인터.
SQLINTEGER	StringLength	입력	속성 값이 문자일 경우 **Value*의 길이

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

애플리케이션은 현재 환경에서 어떠한 연결 핸들도 할당되지 않은 경우에만
SQLSetEnvAttr()을 호출할 수 있다. 애플리케이션에서 성공적으로 설정된 모든 환경
속성들은 현재 환경에서 SQLFreeHandle()이 호출될 때까지 보존된다.

환경 속성

Attribute	내용
SQL_ATTR_ODBC_VERSION	(Win32에서만 적용 됨) SQL_OV_ODBC3 or SQL_OV_ODBC2

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	

관련함수

```
SQLAllocHandle
```

SQLSetPos

행집합 내에서 커서 위치를 지정한다.

구 문

```
SQLRETURN  SQLSetPos (
        SQLHSTMT        stmt,
        SQLSETPOSIROW   rowNumber,
        SQLUSMALLINT    operation,
        SQLUSMALLINT    lockType);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들

자료유형	인자	사용	설명
SQLSETPOSIROW	rowNumber	입력	행집합에서 *operation* 인자로 지정한 작업을 수행할 행의 위치. 1부터 시작한다. 0이면 행집합의 모든 행에 작업을 적용한다.
SQLUSMALLINT	operation	입력	수행할 작업. 다음 중 하나가 올 수 있다: SQL_POSITION SQL_REFRESH SQL_UPDATE SQL_DELETE
SQLUSMALLINT	lockType	입력	잠금 유형. 현재 미지원.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_NEED_DATA
SQL_STILL_EXECUTING
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLSetPos는 행집합 내에서 커서의 위치를 지정하고, 그 위치의 데이터를 새로 가져오거나 갱신 또는 삭제하도록 해준다. *rowNumber*는 행집합 내에서 작업을 수행할 행의 번호를 지정한다. *rowNumber*가 1이면 행집합에서 첫 번째 행을 가리킨다. *rowNumber*가 0이면 행집합 내의 모든 행에 작업이 적용된다. 하지만 *operation*이 SQL_POSITION일 때, *rowNumber*는 0일 수 없다.

커서 위치는 아래의 작업에 요구된다:

- SQLGetData 호출 시
- SQL_DELETE, SQL_REFRESH, 또는 SQL_UPDATE 옵션으로 SQLSetPos 호출 시

SELECT문 실행 후 처음 fetch를 하면 행집합 내에서 커서의 위치는 1이다.

SQLSetPos 함수의 *operation* 인자에는 아래의 옵션이 지원된다:

옵션	설명
SQL_POSITION	드라이버는 *rowNumber*로 지정한 행에 커서를 둔다.
SQL_REFRESH	드라이버는 *rowNumber*로 지정한 행에 커서를 두고, 행집합 버퍼 내의 그 행의 데이터를 새로 가져온다. 만약 *rowNumber*에 0을 지정하면, 행집합 내의 모든 행의 데이터를 새로 가져온다. 만약 드라이버가 HOLE을 감지하면 행 상태 값으로 SQL_ROW_DELETED를 반환한다. 커서가 Sensitive이면 드라이버는 데이터베이스에서 최신 데이터를 가져오고, 아니면 캐시에서 기존 데이터를 가져온다.
SQL_UPDATE	드라이버는 *rowNumber*로 지정한 행에 커서를 두고, 행집합 버퍼 내의 그 행의 데이터로 데이터베이스의 데이터를 갱신한다. 만약 *rowNumber*에 0을 지정하면, 행집합 내의 모든 데이터로 데이터베이스를 갱신한다. SQLBindCol의 length/indicator 인자에 SQL_COLUMN_IGNORE를 지정하면 그 칼럼은 갱신되지 않는다. SQL_ATTR_ROW_OPERATION_PTR 명령문 속성에 의해 설정된 행 작업 배열에 SQL_ROW_IGNORE로 설정한 행이 있다면, 그 행은 무시하고 다음 행을 갱신한다.
SQL_DELETE	드라이버는 *rowNumber*로 지정한 행에 커서를 두고, 그 행을 삭제한다. 만약 *rowNumber*에 0을 지정하면, 행집합 내의 모든 행을 삭제한다. SQL_ATTR_ROW_OPERATION_PTR 명령문 속성에 의해 설정된 행 작업 배열에 SQL_ROW_IGNORE로 설정한 행이 있다면, 그 행은 무시하고 다음 행을 삭제한다.

SQLSetPos 사용하기

애플리케이션에서 SQLSetPos을 호출하려면, 아래의 단계대로 프로그램을 작성하라:

1. SQLBindCol을 호출해서 버퍼 바인딩
2. SQLExecDirect등을 이용해서 SELECT 쿼리문 실행
3. SQL_UPDATE 옵션으로 SQLSetPos를 호출하려면,
   A.한 버퍼의 데이터를 변경하고자 하는 값으로 변경
   B. 칼럼을 무시하려면 length/indicator를 SQL_COLUMN_IGNORE로 지정해서 SQLBindCol을 재호출
   C. 행을 작업 대상에서 제외하려면, SQL_ATTR_ROW_OPERATION_PTR 명령문 속성을 설정하고 그 행에 대응하는 배열 요소에 SQL_ROW_IGNORE를 설정

4. SQLSetPos 호출
   A.L_ATTR_ROW_STATUS_PTR 명령문 속성에 상태 배열을 설정했다면, 이 배열의
   값을 통해 각 행의 수행 결과를 확인할 수 있다.

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	
HY001	메모리 할당 오류	Altibase CLI 드라이버가 함수를 실행하고 완료하기 위해 요구된 메모리를 할당할 수 없음
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
08003		stmt가 연결되지 않거나 connection이 끊어진 상태
HY010	함수 연속 오류	주어진 *stmt*는 이 함수를 수행할 수 없음. 비동기 방식은 미지원.
24519	홀이 발견되었으나 지시자 변수가 없음	
01S01	행에서 오류 발생	*rowNumber*가 0이었고, 작업 수행 중에 한 개 이상의 행에서 오류 발생
HY107	범위를 벗어난 행 값	*rowNumber*에 지정한 값이 행집합 내의 행의 개수보다 크다.
HY109	유효하지 않은 커서 위치	
01001	커서 작업 충돌	
HY024	유효하지 않은 배열 크기	
21S02	테이블의 타입과 칼럼 개수가 칼럼 리스트와 일치하지 않음	
HY092	유효하지 않은 속성/옵션	
24000	올바르지 않은 커서 상태	

관련함수

```
SQLBindCol
SQLBulkOperations
SQLCancel
SQLFetchScroll
SQLGetDescField
SQLGetDescRec
SQLSetDescField
SQLSetDescRec
SQLSetStmtAttr
```

SQLSetStmtAttr

명령문 핸들과 관련된 속성을 설정한다.

Unicode SQLSetStmtAttrW() 동작은 SQLSetStmtAttr()와 동일하다.

구 문

```
SQLRETURN  SQLSetStmtAttr (
        SQLHSTMT        stmt,
        SQLINTEGER      Attribute,
        SQLPOINTER      param,
        SQLINTEGER      StringLength );
```

인 자

자료유형	인자	사용	설명
SQLHENV	stmt	입력	명령문 핸들

자료유형	인자	사용	설명
SQLINTEGER	Attribute	입력	설정할 속성, 지원되는 속성값은 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_PARAM_BIND_TYPE, SQL_ATTR_PARAM_STATUS_PTR, SQL_ATTR_PARAMS_PROCESSED_PTR, SQL_ATTR_PARAMS_ROW_COUNTS_PTR, SQL_ATTR_PARAMS_SET_ROW_COUNTS, SQL_ATTR_PARAMSET_SIZE, SQL_ATTR_PREFETCH_ROWS, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_ROW_BIND_TYPE, SQL_ATTR_ROW_STATUS_PTR, SQL_ATTR_ROWS_FETCHED_PTR, ALTIBASE_STMT_ATTR_ATOMIC_ARRAY SQL_ATTR_FETCH_BOOKMARK_PTR, SQL_ATTR_ROW_OPERATION_PTR, SQL_ATTR_USE_BOOKMARKS, SQL_ATTR_CURSOR_HOLD ALTIBASE_PREFETCH_AUTO_TUNING ALTIBASE_PREPARE_WITH_DESCRIBEPARAM 현재 지원되지 않는 속성값은 SQL_ATTR_APP_PARAM_DESC, SQL_ATTR_APP_ROW_DESC, SQL_ATTR_ASYNC_ENABLE, SQL_ATTR_ENABLE_AUTO_IPD, SQL_ATTR_IMP_PARAM_DESC, SQL_ATTR_IMP_ROW_DESC, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_METADATA_ID, SQL_ATTR_NOSCAN, SQL_ATTR_PARAM_BIND_OFFSET_PTR, SQL_ATTR_PARAM_OPERATION_PTR, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_RETRIEVE_DATA, SQL_ATTR_ROW_BIND_OFFSET_PTR, SQL_ATTR_ROW_NUMBER, SQL_ATTR_SIMULATE_CURSOR
SQLPOINTER	param	입력	*Attribute*와 연관된 값의 포인터 *Attribute* 값에 따라 *param*은 32 bit 부호 없는 정수 값이거나 null-terminated 문자열의 포인터, 바이너리 버퍼, 또는 ODBC에서 정의된 값 *Attribute*가 ODBC 고유의 값이면 *param*은 부호 표시 정수임
SQLINTEGER	StringLength	입력	*Attribute*가 ODBC에서 정의된 속성이고 *param*이 문자열 또는 바이너리 버퍼를 가리키면 이 인자는 **param*의 바이트 길이 여야만 한다. Attribute가 ODBC에서 정의된 속성이고 *param*이 정수면 이 인자는 무시된다.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

명령문 속성은 다른 SQLSetStmtAttr() 호출에 의해 옵션이 변경되거나 명령문이 SQLFreeHandle()을 호출하여 *stmt*가 제거될 때까지 *stmt*에 대한 명령문 옵션은 효력이 있다. 핸들 해제 방식 SQL_CLOSE, SQL_UNBIND, 또는 SQL_RESET_PARAMS과 함께 SQLFreeStmt()를 호출하는 것은 명령문 속성을 재설정하지 않는다.

열 양식 바인딩을 사용하기 위해 애플리케이션은 함수 SQLSetStmtAttr()의 인자 Attribute에 SQL_ATTR_PARAM_BIND_TYPE을 설정하고 인자 param에 SQL_PARAM_BIND_BY_COLUMN을 설정한다. 만약 실행 도중 매번 ARRAY_SIZE가 변경된다면, PARAM_SIZE만 변경을 하면 된다.

PARAM_STATUS_PTR은 각 열들이 수행되었는지에 대한 여부를 돌려주기 위한 값으로 SQLSMALLINT의 배열로 지정한다. 성공하면 SQL_SUCCESS, SQL_SUCCESS_WITH_INFO가, 실패하면 SQL_PARAM_ERROR, 수행되지 않은 경우 SQL_PARAM_UNUSED가 반환된다. 매크로 값은 SQL_PARAM_SUCCESS은 0, SQL_PARAM_ERROR는 5, SQL_PARAM_SUCCESS_WITH_INFO는 6, SQL_PARAM_UNUSED는 7이다.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
```

PARAMS_PROCESSED_PTR에 처리된 열의 개수를 저장하기 위한 변수의 포인터를 지정한다. SQLINTEGER의 포인터형이다. 그 다음 기존처럼 SQLBindParameter()를 수행하면 된다.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
```

SQLBindParameter()를 수행

행 양식 바인딩을 사용할 때, 열 양식 바인딩과의 차이는 PARAM_BIND_TYPE에
structure의 크기를 지정하는 것이다.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, sizeof(struct…));
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
```

SQLBindParameter()를 수행

명령문 속성

Attribute	내용
SQL_ATTR_CONCURRENCY	커서의 동시성 처리를 명시하는 SQLUINTEGER 값: SQL_CONCUR_READ_ONLY: 커서는 read-only임. update는 허용되지 않음 SQL_CONCUR_ROWVER: 행 버전으로 동시성 제어 SQL_CONCUR_LOCK 및 SQL_CONCUR_VALUES는 미지원.
SQL_ATTR_CURSOR_SCROLLABLE	이 명령문 핸들에 대해 열린 커서를 스크롤할 수 있는지를 지정하는 32비트 정수 값.
SQL_ATTR_CURSOR_SENSITIVITY	명령문 핸들에 대해 열린 커서가 다른 커서가 변경한 내용을 볼 수 있는지 여부를 지정하는 32비트 정수 값. SQL_INSENSITIVE 또는 SQL_SENSITIVE로 지정 가능. SQL_UNSPECIFIED는 미지원.
SQL_ATTR_CURSOR_TYPE	커서 타입을 명시하는 SQLUINTEGER 값: SQL_CURSOR_FORWARD_ONLY: 커서는 순방향으로만 진행 SQL_CURSOR_STATIC: 결과 집합 내의 데이터는 STATIC 임. SQL_CURSOR_KEYSET_DRIVEN: 커서에 포함되는 행의 멤버와 순서가 고정적이다. Altibase CLI 드라이버는 순수한 키집합 커서를 지원한다. SQL_KEYSET_SIZE 명령문 속성은 무시된다.
SQL_ATTR_PARAM_BIND_TYPE	(array binding 시 필요한 설정값) 이 필드는 열 양식 바인딩으로 검색하기 위해 SQL_PARAM_BIND_BY_COLUMN (기본값)이 설정된다. 행 양식 바인딩으로 검색하기 위해서는 구조의 길이나 동적 매개변수들이 바인드될 버퍼의 길이가 설정된다. 이 길이는 바인드된 매개변수 스페이스를 포함해야 한다.
SQL_ATTR_PARAM_STATUS_PTR	(array binding 시 필요한 설정값) 이 필드는 PARAMSET_SIZE가 1 보다 클 때만 요구된다. 상태 값들은 다음 값들을 가질 수 있다: SQL_PARAM_SUCCESS: SQL 문이 성공적으로 수행됨. 매크로 값은 0. SQL_PARAM_SUCCESS_WITH_INFO: SQL 문이 성공적으로 수행됨; 그러나 경고 정보를 진단 데이터 구조에서 볼 수 있다. 매크로 값은 5. SQL_PARAM_ERROR: 매개변수들을 수행하는데 있어서 오류가 있음. 추가적인 오류 정보를 진단 데이터 구조에서 볼 수 있다. 매크로 값은 6. SQL_PARAM_UNUSED: 이 매개변수 집합은 사용되지 않음, 아마도 이전 매개변수 집합이 추가 진행을 중단하는 오류를 야기했기 때문임. 매크로 값은 7.
SQL_ATTR_PARAMS_PROCESSED_PTR	(array binding 시 필요한 설정값) 오류를 포함한 매개변수 개수를 반환하는 버퍼를 가리킴. null pointer이면 아무 값도 반환되지 않을 것이다. SQLExecDirect() or SQLExecute()의 호출에 대해 SQL_SUCCESS 또는 SQL_SUCCESS_WITH_INFO이 반환되지 않는다면 버퍼의 내용은 정의되지 않는다.
SQL_ATTR_PARAMS_ROW_COUNTS_PTR	(array binding 시 필요한 설정값) array binding에 대한 SQLExecute의 결과를 아래의 값으로 반환하는 버퍼를 설정한다. SQL_SUCCESS: 모든 array elements에 대해 SQL 문이 성공적으로 수행된 경우 SQL_ERROR: 하나의 array element 라도 SQL 문 수행이 실패한 경우 SQL_NO_DATA: 하나의 array element 라도 변경(입력, 삭제)된 것이 없는 경우
SQL_ATTR_PARAMS_SET_ROW_COUNTS	(array binding 시 필요한 설정값) SQL_ROW_COUNTS_ON: 각 array element에 대해 변경된 레코드의 개수를 반환한다. 즉, 변경된 레코드가 있으면 그 개수가 반환되고, 변경된 레코드가 없으면 0, 오류가 발생하면 SQL_USHRT_MAX(65534)를 반환한다. SQL_ROW_COUNTS_OFF: 속성 SQL_ATTR_PARAM_STATUS_PTR의 기존 동작을 그대로 유지한다.
SQL_ATTR_PARAMSET_SIZE	(array binding 시 필요한 설정값) 각 매개변수에 대한 값들의 개수를 명시하는 SQLUINTEGER 값
SQL_ATTR_PREFETCH_ROWS	한 번의 Fetch 연산으로 서버에서 한 번에 가져올 행의 개수를 지정한다. 이 속성을 0으로 설정하면, CLI 드라이버는 한 네트워크 패킷에 담을 수 있는 최대 크기만큼 서버로부터 데이터를 가지고 온다. 설정 가능한 값의 범위는 0에서 2147483647이다.

Attribute	내용
SQL_ATTR_ROW_ARRAY_SIZE	(SELECT 문에서 array fetch 할 때 필요한 설정값) 각 SQLFetch()을 호출 함으로서 반환되는 행들의 개수를 명시하는 SQLUINTEGER 값
SQL_ATTR_ROW_BIND_TYPE	(SELECT 문에서 array fetch 할 때 필요한 설정값) SQLFetch()가 호출될 때 사용될 바인딩 방향을 설정하는 SQLUINTEGER 값. 열 양식 바인딩은 SQL_BIND_BY_COLUMN 값을 설정 함으로서 검색된다. 행 양식 바인딩은 구조의 길이나 결과 열들이 바인드될 버퍼의 길이를 설정 함으로서 검색된다. 길이가 명시된다면, 바인드될 열에 대해 스페이스를 포함해야만 한다.
SQL_ATTR_ROW_STATUS_PTR	(SELECT 문에서 array fetch 할 때 필요한 설정값)배열의 크기는 행집합의 행의 수만큼 이다.이 속성은 null pointer가 설정될 수 있고, 그런 경우 Altibase CLI 드라이버는 열 상태 값을 반환하지 않는다.
SQL_ATTR_ROWS_FETCHED_PTR	(SELECT 문에서 array fetch 할 때 필요한 설정값) SQLFetch()를 호출 후에 페치된 행들의 수를 반환하는 버퍼를 가리키는 SQLUINTEGER * 값
ALTIBASE_STMT_ATTR_ATOMIC_ARRAY	Altibase의 전용 속성으로, Atomic Array Insert를 수행한다. 일반적인 Array Insert가 구문들을 각각 처리하는 것과 비교해 Atomic Array Insert는 1개의 구문으로 여러 개의 구문을 처리한다. Attribute에 ALTIBASE_STMT_ATTR_ATOMIC_ARRAY를 설정하고 인자 param에 SQL_TRUE 또는 SQL_FALSE를 설정한다. SQL_TRUE를 설정하면 Atomic Array Insert를 실행한다. 그러나 Atomic Array Insert가 기존의 Array Insert보다 빠른 성능을 발휘하기 위해선 Array Size(SQL_ATTR_PARAMSET_SIZE) 를 지정한다. 또한 다음의 속성들과 함께 사용해 결과값을 저장할 수 있다. SQL_ATTR_PARAM_STATUS_PRT : 1번째 로우 자리에만 실제 결과값이 저장되고 나머지는 모두 성공으로 처리된다. SQL_ATTR_PARAMS_PROCESSED_PTR : 1번째 로우 자리에만 실제 결과값 저장되고 나머지는 정의되지 않는다.
SQL_ATTR_FETCH_BOOKMARK_PTR	바이너리 북마크 값을 담고 있는 버퍼를 가리키는 포인터. SQL_FETCH_BOOKMARK로 SQLFetchScroll을 호출하면, 드라이버는 이 속성으로 설정한 북마크 값을 사용해서 행을 스크롤한다. 이 속성의 기본값은 null 포인터이다. 이 속성으로 설정한 북마크 값은 SQLBulkOperations가 북마크로 삭제, 갱신, 또는 fetch할 때 사용되지는 않는다. SQLBulkOperations은 행집합 버퍼에 캐시된 북마크를 사용한다.
SQL_ATTR_ROW_OPERATION_PTR	SQLSetPos를 이용한 벌크 작업 중에 어떤 행을 작업할 것인지 무시할 것인지를 지정하기 위한 SQLUSMALLINT 값들의 배열을 가리키는 포인터. SQL_ROW_IGNORE: 벌크 작업에서 제외할 행 SQL_ROW_PROCEED: 벌크 작업에 포함할 행 SQLBulkOperations 수행 중에는 이 배열의 값이 적용되지 않는다.
SQL_ATTR_USE_BOOKMARKS	북마크 사용 여부를 설정하는 SQLULEN 값. SQL_UB_OFF: 기본값 SQL_UB_ON 또는 SQL_UB_FIXED: 고정 길이(4bytes) 북마크 사용 SQL_UB_VARIABLE: 가변 길이 북마크 사용
SQL_ATTR_CURSOR_HOLD	열린 커서에서 트랜잭션 완료 효과를 제어하는 SQLULEN 값. SQL_CURSOR_HOLD_ON: 트랜잭션 커밋 시 커서가 닫히지 않는다. SQL_CURSOR_HOLD_OFF: 트랜잭션 커밋 시 커서가 닫힌다.(기본값)
ALTIBASE_PREFETCH_ASYNC	비동기적으로 prefetch를 수행하여 fetch 성능을 향상시킨다. TCP, SSL 접속에서만 사용할 수 있다. ALTIBASE_PREFETCH_ASYNC_OFF: 동기적으로 prefetch를 수행한다.(기본값) ALTIBASE_PREFETCH_ASYNC_PREFERRED: 비동기적으로 prefetch를 수행한다. 명령문마다 비동기 prefetch 하도록 설정할 수 있지만 접속 당 하나의 명령문만 prefetch가 비동기적으로 수행된다.
ALTIBASE_PREFETCH_AUTO_TUNING	비동기적으로 prefetch를 수행할 경우, 네트워크 상태에 따라 auto-tuning 할 것인지 여부를 지정한다. Auto-tuning이란 네트워크 상태에 따라 prefetch row 개수를 자동으로 조절해주는 기능이다. 이 기능은 리눅스에서만 사용할 수 있다. ALTIBASE_PREFETCH_AUTO_TUNING_OFF: auto-tuning 기능을 사용하지 않는다 (리눅스 이외의 OS에서 기본값) ALTIBASE_PREFETCH_AUTO_TUNING_ON: auto-tuning 기능을 사용한다. (리눅스에서 기본값)
ALTIBASE_PREPARE_WITH_DESCRIBEPARAM	Prepare시 매개변수(parameter) 정보를 같이 요청해서 네트워크 I/O 비용을 줄이기 위한 속성이다. 매개변수 정보가 필요한 경우 사용하면 성능을 향상시킬 수 있다. ALTIBASE_PREPARE_WITH_DESCRIBEPARAM_OFF: Prepare시 매개변수 정보를 요청하지 않는다.(기본값) ALTIBASE_PREPARE_WITH_DESCRIBEPARAM_ON: Prepare시 매개변수 정보를 같이 요청한다

진 단

SQLSTATE	설명	부연설명
HY000	일반 오류	
24000	올바르지 않은 커서 상태	
HY010	함수 순서 오류	
HY011	현재는 속성 설정 불가능	
HY013	메모리 관리 오류	
HYC00	지원하지 않는 속성 사용	인자 *Attribute*에 명시된 값이 드라이버가 지원 안 하는 값 임.
HY009	null 포인터의 유효하지 않은 사용	
HY017	자동으로 할당되는 descriptor 핸들의 유효하지 않은 사용	
01S02	미지원 옵션 값 사용	
HY024	유효하지 않은 속성 값	명시한 *Attribute*에 대해 유효하지 않은 값이 지정되었음.
HY024	유효하지 않은 배열 크기	

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_ex4.cpp 참고 >

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, (void*)sizeof(demo_ex4_data), 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (void*)10, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR,(void*) &processed_ptr, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, (void*)status, 0);
Automic Array Insert를 사용한다.
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER) array_size, 0); // Array Size를 지정한다.
SQLSetStmtAttr(stmt, ALTIBASE_STMT_ATTR_ATOMIC_ARRAY, (SQLPOINTER) SQL_TRUE, 0); // Atomic 속성을 지정한다.
아래는 커서 타입을 지정하는 예제 코드이다.
...
SQLAllocStmt(sDbc, &sStmt);
SQLSetStmtAttr(sStmt, SQL_ATTR_CURSOR_TYPE, (SQLPOINTER) SQL_CURSOR_KEYSET_DRIVEN, NULL);
SQLSetStmtAttr(sStmt, SQL_ATTR_ROW_STATUS_PTR, (SQLPOINTER) sRowStatus, NULL);

/* ... */

SQLExecDirect(sStmt, (SQLCHAR *)"SELECT t1.* FROM t1", SQL_NTS);
while (1)
{
        sRC = SQLFetch(sStmt);
        if (! SQL_SUCCEEDED(sRC))
        {
        break;
        }
        if (sRowStatus == SQL_ROW_DELETED)
        {
        /* hole */
        }
        else
        {
        /* todo */
        }
}
...
```

SQLSpecialColumns

명시된 테이블 내의 열들에 대해 다음 정보를 검색한다.

- 테이블의 행을 유일하게 식별하는 (예: 프라이머리 키) 최적의 열 집합
- 행의 어떤 값이 트랜잭션에 의해 갱신될 때 자동으로 갱신되는 열들

Unicode SQLSpecialColumnsW() 동작은 SQLSpecialColumns()와 동일하다.

구 문

```
SQLRETURN  SQLSpecialColumns (
        SQLHSTMT                stmt,
        SQLSMALLINT             fColType,
        SQLCHAR *               szTableQual,
        SQLSMALLINT             cbTableQual,
        SQLCHAR *               szTableOwner,
        SQLSMALLINT     cbTableOwner,
        SQLCHAR *               szTableName,
        SQLSMALLINT             cbTableName,
        SQLSMALLINT             fScope,
        SQLSMALLINT     fNullable );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLSMALLINT	fColType	입력	반환할 열의 타입. SQL_BEST_ROWID: 열(들)의 값들을 검색 함으로서 테이블의 행을 유일하게 식별하는 최적의 열들을 반환
SQLCHAR *	szTableQual	입력	항상 NULL이 반환
SQLSMALLINT	cbTableQual	입력	*szTableQual의 문자 개수
SQLCHAR *	szTableOwner	입력	스키마 이름
SQLSMALLINT	cbTableOwner	입력	*szTableOwner의 문자 개수
SQLCHAR *	szTableName	입력	테이블 이름, null pointer일 수 없다.
SQLSMALLINT	cbTableName	입력	*szTableName의 문자 개수
SQLSMALLINT	fScope	입력	사용 안함
SQLSMALLINT	fNullable	입력	사용 안함. (프라이머리 키에 해당하는 열을 반환하므로 NULL을 허용하지 않음)

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

*fColType*이 SQL_BEST_ROWID 일 때, SQLSpecialColumns()는 테이블의 각 행을 유일하게 식별하는 열(들)을 반환한다. 이러한 열들은 select 목록이나 WHERE 절에서 사용될 수 있다.

테이블의 열들에 대한 다양한 정보를 반환하는데 사용되는 SQLColumns()는 행을 유일하게 식별하는 열들이나 또는 행의 어떤 값이 트랜잭션에 의해 갱신될 때 자동으로 갱신되는 열들을 반드시 반환하지는 않기 때문에 SQLSpecialColumns()는 이러한 열들을 반환하는데 사용한다.

테이블의 각 행을 유일하게 식별하는 열들이 없다면, SQLSpecialColumns()는 행들이 없는 행집합을 반환한다. 명령문상에서 SQLFetch()에 대한 후속 호출은 SQL_NO_DATA를 반환한다.

fColType, *fScope*, *fNullable* 인자들이 DB에서 지원되지 않는 특징들을 명시하면 SQLSpecialColumns()는 빈 결과 집합을 반환한다.

이름	번호	자료유형	설명
SCOPE	1	SMALLINT	SQL_SCOPE_SESSION 값이 2로 고정
COLUMN_NAME	2	VARCHAR (NOT NULL)	열 이름. Altibase CLI 드라이버는 이름을 갖지 않은 열에 대해 빈 문자열을 반환한다.

이름	번호	자료유형	설명
DATA_TYPE	3	SMALLINT (NOT NULL)	SQL 데이터 타입
TYPE_NAME	4	VARCHAR (NOT NULL)	DATA_TYPE에 대응하는 데이터 타입의 이름을 나타내는 문자 스트링
COLUMN_SIZE	5	INTEGER	열의 크기. 열의 크기가 적합치 않으면 NULL 이 반환됨
BUFFER_LENGTH	6	INTEGER	자료를 저장하는 최대 바이트
DECIMAL_DIGITS	7	SMALLINT	열의 소수자리 수, 소수자리 수를 적용할 수 없는 데이터 타입은 널이 반환
PSEUDO_COLUMN	8	SMALLINT	문자나 바이너리 데이터 타입 열의 최대 자릿수. 이외의 데이터 타입일 경우 널이 반환

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY009	유효하지 않은 인자 사용 (null pointer)	*szTableName*이 null pointer 임

관련함수

```
SQLBindCol
SQLColumns
SQLFetch
SQLPrimaryKeys
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta7.cpp 참고 >

```
if (SQLSpecialColumns(stmt, 0,
                          NULL, 0,
                          NULL, 0,
                          "DEMO_META7", SQL_NTS,
                          NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLColumns");
      SQLFreeStmt(stmt, SQL_DROP);
      return SQL_ERROR;
}

  SQLBindCol(stmt, 2, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
  SQLBindCol(stmt, 3, SQL_C_SSHORT, &DataType, 0, &cbDataType);
  SQLBindCol(stmt, 4, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
  SQLBindCol(stmt, 5, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
  SQLBindCol(stmt, 6, SQL_C_SLONG, &BufferLength, 0, &cbBufferLength);
  SQLBindCol(stmt, 7, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
```

SQLStatistics

단일 테이블에 대한 통계 목록과 그 테이블과 관련된 색인들을 검색한다. Altibase CLI 드라이버는 정보들을 결과 집합 형태로 반환한다.

Unicode SQLStatisticsW() 동작은 SQLStatistics()와 동일하다.

구 문

```
SQLRETURN  SQLStatistics (
        SQLHSTMT                stmt,
        SQLCHAR *               cName,
        SQLSMALLINT             cNameLength,
        SQLCHAR *               sName,
        SQLSMALLINT     sNameLength,
        SQLCHAR *               tName,
        SQLSMALLINT             tNameLength,
        SQLSMALLINT             unique,
        SQLSMALLINT     reserved );
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLCHAR *	cName	입력	카탈로그 이름
SQLSMALLINT	cNameLength	입력	**cName* 문자 개수
SQLCHAR *	sName	입력	스키마 이름
SQLSMALLINT	sNameLength	입력	**sName*의 문자 개수
SQLCHAR *	tName	입력	테이블 이름, null pointer일 수 없다.
SQLSMALLINT	tNameLength	입력	**tName*의 문자 개수
SQLSMALLINT	unique	입력	index type:SQL_INDEX_UNIQUE or SQL_INDEX_ALL
SQLSMALLINT	reserved	입력	사용 안함

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLStatistcs()는 단일 테이블에 대한 정보를 NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, 그리고 ORDINAM_POSITION 순서로 결과 집합에 반환한다. 결과 집합은 테이블의 통계 정보와 (CARDINALITY와 PAGES) 각 색인에 대한 정보를 결합시킨다.

이름	번호	자료 유형	설명
TABLE_CAT	1	VARCHAR	항상 NULL이 반환
TABLE_SCHEM	2	VARCHAR	TABLE_NAME을 포함하는 스키마 이름
TABLE_NAME	3	VARCHAR (NOT NULL)	테이블 이름
NON_UNIQUE	4	SMALLINT	색인이 중복 값을 금지하는지의 여부 SQL_TRUE: 색인이 중복 값을 허용 SQL_FALSE: 색인이 고유한 값 NULL: *TYPE*이 SQL_TABLE_STAT 일 때 NULL을 반환
INDEX_QUALIFIER	5	VARCHAR	사용 안함 (빈 문자열이 반환 됨)
INDEX_NAME	6	VARCHAR	색인 이름; *TYPE*이 SQL_TABLE_STAT 이면 NULL 반환
TYPE	7	SMALLINT (NOT NULL)	반환된 정보의 type SQL_TABLE_STAT: 테이블에 대한 통계정보를 포함하는지 표시 (CARDINALITY 또는 PAGES 열에) SQL_INDEX_BTREE: B-Tree 인덱스임을 표시 SQL_INDEX_HASHED: 해시된 색인 인지를 표시 SQL_INDEX_OTHER : 색인 유형이 그 밖의 다른 색인 인지를 표시. (T-Tree)
ORDINAL_POSITION	8	SMALLINT	색인 내의 열의 순서적 위치. (1 부터 시작) *TYPE*이 SQL_TABLE_STAT이면 NULL이 반환.

이름	번호	자료 유형	설명
COLUMN_NAME	9	VARCHAR	열의 이름. 만약 열이 표현식이면 (e.g. SALARY + BENEFITS) 표현식이 반환; 표현식이 결정되어질 수 없다면 빈 문자열이 반환. *TYPE*이 SQL_TABLE_STAT이면 NULL이 반환
ASC_OR_DESC	10	CHAR(1)	열에 대한 정렬 순서. A: 오름차순, D: 내림차순 이 열 정력 순서가 DB에 의해 지원되지 않고 *TYPE*이 SQL_TABLE_STAT이면 NULL 반환
CARDINALITY	11	INTEGER	테이블이나 색인의 순서; *TYPE*이 SQL_TABLE_STAT이면 행 번호; *TYPE*이 SQL_TABLE_STAT이 아니면 색인의 고유한 값 번호; 값이 DB에서 이용 가능하지 않으면 NULL 반환
PAGES	12	INTEGER	색인이나 테이블에 저장하기 위해 사용된 페이지 번호; *TYPE*이 SQL_TABLE_STAT이면 이 열은 테이블을 지정하기 위해 사용된 페이지 수를 포함; *TYPE*이 SQL_TABLE_STAT이 아니면 이 열은 색인을 저장하기 위해 사용된 페이지 수를 포함; 값이 DB에서 이용 가능하지 않으면 NULL 반환
FILTER_CONDITION	13	VARCHAR	색인이 필터된 색인이면 이 열은 필터 조건이다. 즉, SALARY > 30000; 필터 조건이 결정 되어질 수 없다면 이 열은 빈 문자열이다. NULL: 색인이 필터된 색인이 아닌 경우, 색인이 필터된 색인인지 또는 *TYPE*이 SQL_TABLE_STAT 인지 결정 되어질 수 없는 경우

[표 2-5] SQLStatistics()에 의해 반환 되는 열

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	
HY009	유효하지 않은 인자 사용 (null pointer)	*tName*이 null pointer 임 *cName*이 null pointer 임 *sName*이 null pointer 임

관련함수

```
SQLBindCol
SQLFetch
SQLPrimaryKeys
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta4.cpp 참고 >

```
if (SQLStatistics(stmt,NULL, 0,
                      NULL, 0,
                      "DEMO_META4", SQL_NTS,
                      SQL_INDEX_ALL, 0) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLStatistics");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
    SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
    SQLBindCol(stmt, 4, SQL_C_SSHORT, &NonUnique, 0, &cbNonUnique);
    SQLBindCol(stmt, 6, SQL_C_CHAR, szIndexName, STR_LEN, &cbIndexName);
    SQLBindCol(stmt, 8, SQL_C_SSHORT, &OrdinalPosition, 0, &cbOrdinalPosition);
    SQLBindCol(stmt, 9, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
    SQLBindCol(stmt, 10, SQL_C_CHAR, szAscDesc, 2, &cbAscDesc);
```

SQLTablePrivileges

테이블들의 목록과 각 테이블과 관련된 권한들을 반환한다. Altibase CLI 드라이버는
명시된 명령문 상에 결과 집합으로써 정보를 반환한다.

Unicode SQLTablePrivilegesW() 동작은 SQLTablePrivileges()와 동일하다.

구 문

```
SQLRETURN  SQLTablePrivileges(
        SQLHSTMT               stmt,
        SQLCHAR *              cName,
        SQLSMALLINT            cNaneLength,
        SQLCHAR *              sName,
        SQLSMALLINT            sNameLength,
        SQLCHAR *              tName,
        SQLSMALLINT            tNameLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	명령문 핸들
SQLCHAR*	cName	입력	카탈로그 이름
SQLSMALLINT	cNameLength	입력	*cName의 문자 개수
SQLCHAR *	sName	입력	검색할 스키마 이름
SQLSMALLINT	sNameLength	입력	**sName*의 문자 개수
SQLCHAR *	tName	입력	검색할 테이블 이름
SQLSMALLINT	tNameLength	입력	**tName*의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLTablePrivileges()는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, PRIVILEGE, 그리고 GRANTEE에 의해 순서화된 표준 결과 집합 형태로 결과들을 반환한다.

다음 테이블은 결과 집합의 열들을 나열한다.

이름	번호	자료 유형	설명
TABLE_CAT	1	VARCHAR	항상 NULL 반환
TABLE_SCHEM	2	VARCHAR	스키마 이름; DB에 적절하지 않은 경우 NULL
TABLE_NAME	3	VARCHAR (NOT NULL)	테이블 이름
GRANTOR	4	VARCHAR	권한을 부여한 사용자 이름; DB에 적절하지 않은 경우 NULL
GRANTEE	5	VARCHAR (NOT NULL)	권한을 부여 받은 사용자 이름
PRIVILEGE	6	VARCHAR (NOT NULL)	테이블 권한. 다음 권한 중에 하나이다. ALTER: 피 수여자는 테이블의 정의를 변경할 수 있다. DELETE: 피 수여자는 테이블의 행들을 삭제하는 것이 허용된다. INDEX: 피 수여자는 테이블에 대하여 인덱스 연산 (create, alter 등)이 허용된다. INSERT: 피 수여자는 새로운 행들을 테이블에 삽입할 수 있다. REFERENCES: 피 수여자는 제한조건을 가진 테이블의 열을 참조하는 것이 허용된다. SELECT: 피 수여자가 테이블에서 한 개 또는 여러 열들을 검색하는 것이 허용된다. UPDATE: 피 수여자는 테이블에 대하여 하나 이상의 데이터를 수정할 수 있다.
IS_GRANTABLE	7	VARCHAR	피 수여자가 다른 사용자들에게 권한을 부여할 수 있는지를 나타낸다: YES, NO, 또는 DB에 적절치 않거나 알려지지 않은 경우 NULL

[표 2-6] SQLTablePrivileges()에 의해 반환 되는 열

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY009	유효하지 않은 인자 (null pointer) 사용	인자 *cName*이 null 포인터임.
HY090	유효하지 않은 문자열 또는 버퍼 길이	이름 길이 인자들 중 하나의 값이 0보다 작거나 SQL_NTS와 같지 않음.

관련함수

```
SQLBindCol
SQLCancel
SQLColumns
SQLFetch
SQLPrimaryKeys
SQLStatistics
SQLTables
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta10.cpp 참고 >

```
if (SQLTablePrivileges(stmt,
                       NULL, 0,
                       "SYS", SQL_NTS,
                       "DEMO_META10", SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLTablePrivileges");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, NAME_LEN, &cbSchema);
    SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, NAME_LEN,&cbTableName);
    SQLBindCol(stmt, 4, SQL_C_CHAR, szGrantor, NAME_LEN, &cbGrantor);
    SQLBindCol(stmt, 5, SQL_C_CHAR, szGrantee, NAME_LEN, &cbGrantee);
    SQLBindCol(stmt, 6, SQL_C_CHAR, szPrivilege, NAME_LEN,&cbPrivilege);
    SQLBindCol(stmt, 7, SQL_C_CHAR, szGrantable, 5, &cbGrantable);
```

SQLTables

특정 DB에 저장된 테이블, 카탈로그 또는 스키마 이름들, 그리고 테이블 타입의 목록이 결과 집합 형태로 반환된다.

Unicode SQLTablesW() 동작은 SQLTables()와 동일하다.

구 문

```
SQLRETURN  SQLTables (
        SQLHSTMT                stmt,
        SQLCHAR *               cName,
        SQLSMALLINT     cNameLength,
        SQLCHAR *               sName,
        SQLSMALLINT     sNameLength,
        SQLCHAR *               tName,
        SQLSMALLINT     tNameLength,
        SQLCHAR *               tableType,
        SQLSMALLINT     tableTypeLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLCHAR *	cName	입력	카탈로그 이름

자료유형	인자	사용	설명
SQLSMALLINT	cNameLength	입력	**cName*의 문자 개수
SQLCHAR *	sName	입력	스키마 이름
SQLSMALLINT	sNameLength	입력	**sName*의 문자 개수
SQLCHAR *	tName	입력	테이블 이름
SQLSMALLINT	tNameLength	입력	**tName*의 문자 개수
SQLCHAR *	tableType	입력	대조할 테이블 타입의 목록
SQLSMALLINT	tableTypeLength	입력	*tableType의 문자 개수

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLTables()는 요구된 범위 내에서 모든 테이블 정보를 나열한다. 어떤 사용자는 임의의 테이블에 대해서 SELECT 특권을 가질 수도 있고 아닐 수도 있다.

애플리케이션은 SELECT 특권이 주어지지 않은 사용자가 임의의 테이블을 검색하는 상황을 조정할 수 있어야만 한다.

카탈로그, 스키마, 테이블 타입의 목록을 지원하기 위하여, 다음 특별한 구문들은 SQLTables()의 *cName*, *sName*, *tName*, 그리고 *tableType*을 위해 정의된다:

만약 *sName*이 SQL_ALL_SCHEMAS이고 그리고 *cName*과 *tName*이 빈 문자열이면 결과 집합은 DB에 대해 유효한 스키마 목록들을 포함한다. (모든 열들은 TABLE_SCHEM 열이 NULL 들을 포함하기를 기대한다.)

만약 *tableType*이 SQL_ALL_TABLE_TYPES이고 *cName*, *sName*, 그리고 *tName*이 빈 문자열이면, 결과 집합은 DB에 대해 유효한 테이블 타입 목록들을 포함한다. (TABLE_TYPE 열을 제외한 모든 열은 NULL들을 포함한다.)

애플리케이션은 *tableType*을 대문자로 항상 명시해야 하며, 다음의 값 중 하나를 지정할 수 있다: SQL_ALL_TABLE_TYPES, "SYSTEM TABLE", "TABLE", "VIEW", "QUEUE", "SYNONYM", "MATERIALIZED VIEW".

SQLTables()는 TABLE_TYPE, TABLE_CAT, TABLE_SCHEM, 그리고 TABLE_NAME 순서에 의해 결과들을 표준 결과 집합 형태로 반환한다.

다음 테이블은 결과 집합의 열들을 나열한 것이다.

열 이름	열 번호	자료 유형	설명
TABLE_CAT	1	VARCHAR	항상 NULL이 반환
TABLE_SCHEM	2	VARCHAR	TABLE_NAME을 포함하는 스키마 이름; DB에 적합치 않으면 NULL
TABLE_NAME	3	VARCHAR (NOT NULL)	테이블 이름
TABLE_TYPE	4	VARCHAR	테이블 타입 이름 아래 타입 중 하나가 반환된다: SQL_ALL_TABLE_TYPES 'SYSTEM TABLE' 'TABLE' 'VIEW' 'QUEUE' 'SYNONYM' 'MATERIALIZED VIEW'
REMARKS	5	VARCHAR	사용되지 않음

[표 2-7] SQLTables()에 의해 반환 되는 열

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLBindCol
SQLColumns
SQLFetch
SQLStatistics
```

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_meta1.cpp 참고 >

```
if (SQLTables(stmt,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLTables");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                   schem, sizeof(schem), &schem_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 3, SQL_C_CHAR,
                   name, sizeof(name), &name_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 4, SQL_C_CHAR,
                   type, sizeof(type), &type_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
    {
        if ( rc == SQL_ERROR )
        {
            execute_err(dbc, stmt, "SQLFetch");
            break;
        }
        printf("%-40s%-40s%s\n", schem, name, type);
    }
```

SQLTransact

연결 상태에서 현재 트랜잭션을 정상 종료하거나 철회한다.

연결 시간 이후나 이전 SQLTransact() 호출 이후의 연결에서 수행 된 데이터베이스에
대한 모든 변경을 정상 종료하거나 철회한다.

트랜잭션이 연결 상태에서 사용 중이면 애플리케이션은 데이터베이스를 단절하기 전에 SQLTransact()를 호출해야 한다.

SQLTransact()는 SQLEndTran()으로 대체될 수 있다.

구 문

```
SQLRETURN  SQLTransact (
        SQLHENV                 env,
        SQLHDBC                 dbc,
        SQLSMALLINT     type );
```

인 자

자료유형	인자	사용	설명
SQLHENV	env	입력	환경 핸들
SQLHDBC	dbc	입력	연결 핸들
SQLSMALLINT	type	입력	다음 두 값 중 하나 SQL_COMMIT, SQL_ROLLBACK

결과값

```
SQL_SUCCESS
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

트랜잭션을 SQL_COMMIT 또는 SQL_ROLLBACK으로 완료하면 다음과 같은 효과가 있다.

SQLTransact()를 호출한 후에도 명령문 핸들이 유효

바인드된 매개변수, 열 바인딩이 트랜잭션보다 오래 살아 있다.

미결 검색인 결과 집합이 삭제된다.

현재 사용 중인 트랜잭션이 없으면 SQLTransact()를 호출하는 것은 데이터베이스 서버에 아무런 영향을 끼치지 않고 SQL_SUCCESS를 반환한다.

연결 손실 때문에 COMMIT 또는 ROLLBACK을 실행하는 동안 SQLTransact()가 실패할 수 있다. 이 경우에는 애플리케이션이 반영 또는 롤백이 처리되었는지 판별할 수 없으므로 데이터베이스 관리자에게 도움을 요청

예 제

< $ALTIBASE_HOME/sample/SQLCLI/demo_tran1.cpp 참고 >

```
SQLTransact(SQL_NULL_HENV, dbc, SQL_COMMIT);
```

3.LOB 인터페이스

이 장에서는 LOB 데이터를 사용하는데 필요한 함수 및 데이터 타입을 설명한다.

LOB data types

다음 [표 3-1]은 LOB을 지원하는 SQL 데이터 타입의 식별자들이다.

SQL 식별자	데이터 타입	설명
SQL_BLOB	BLOB	BLOB은 가변 길이를 가지는 이진 데이터 타입.
SQL_CLOB	CLOB	CLOB은 가변 길이를 가지는 데이터 타입.

[표 3-1] SQL 데이터 타입의 식별자

다음 표는 LOB을 지원하는 C 데이터 타입의 식별자이다. 각 식별자에 해당하는 ODBC의 C 데이터 타입과 이 데이터 타입의 정의를 나열한다.

C 타입 식별자	ODBC C 타입	C 타입 정의
SQL_C_BLOB_LOCATOR	SQLUBIGINT	unsigned _int64
SQL_C_CLOB_LOCATOR	SQLUBIGINT	unsigned _int64

[표 3-2] LOB 지원 C 데이터 타입의 식별자

64비트 정수형의 이름은 플랫폼에 따라 다르다. 위 표에서 사용한 _int64는 일부 플랫폼에서 사용되는 64비트 정수형의 이름이다.

CLOB 데이터는 SQL_C_CHAR를, BLOB 데이터는 SQL_C_BINARY를 사용하여 사용자 변수를 바인딩하도록 한다.

LOB Locator를 얻고자 한다면, 해당 LOB 칼럼의 타입에 따라서 SQL_C_CLOB_LOCATOR, 혹은 SQL_C_BLOB_LOCATOR를 적절하게 바인드하면 된다. 여기서 말하는 LOB Locator 즉, LOB 위치 입력기는 운영체제의 파일 포인터처럼 LOB 데이터를 연산할 때 사용되는 핸들이다.

읽기용 LOB 위치 입력기는 SELECT LOB칼럼이름 FROM 테이블 where… 와 select를 수행한 후에 획득된다. 쓰기용 LOB 위치입력기는 SELECT LOB칼럼이름 FROM 테이블 where… FOR UPDATE를 수행한 후에 획득된다.

LOB 위치입력기는 MVCC와 관련하여 특정 시점의 LOB 데이터를 가리키기 때문에 위치입력기를 발생시킨 트랜잭션과 수명주기(life-cycle)를 같이 한다. 따라서 LOB 위치입력기를 이용하여 LOB에 대한 연산을 하기 위해서는 connection을 항상 NON-AUTOCOMMIT 모드로 설정하여 사용해야 한다.

사용자 변수의 LOB 타입으로 지정된 타입, 이를테면 SQL_C_BLOB, SQL_C_CLOB 등은 따로 존재하지 않음에 유의한다.

LOB Function Overview

LOB 데이터를 다루기 위해서 사용되는 함수들은 아래와 같다.

1. SQLBindFileToCol() (비표준)
   Full Retrieve
2. SQLBindFileToParam() (비표준)
   Full Insert, Full Update
3. SQLGetLobLength() (비표준)
   LOB 데이터의 길이를 구한다.
4. SQLGetLob() (비표준)
   Partial Retrieve
5. SQLPutLob() (비표준)
   Partial Insert, Partial Update, Partial Delete
6. SQLFreeLob() (비표준)
   사용중이던 LOB locator를 해제
7. SQLGetData(), SQLPutData() (표준)
   Full Retrieve/Update
8. 여타 ODBC 의 모든 표준 함수들

위 함수들 중 1 ~ 6번은 Altibase가 LOB 을 다루기 위해 제공하는 특수 함수들로써, ODBC 표준에는 없는 함수들이다.

7, 8 번과 같이 ODBC 스펙에서 정의하는 함수들을 이용해서 데이터베이스의 칼럼 타입이 LOB 인지의 여부에 무관하게 표준 함수만으로도 LOB 데이터에 접근할 수 있다. 단, ODBC 표준 함수만을 사용했을 경우, 부분 갱신(partial update), 부분 검색(partial retrieve) 등의 기능은 사용할 수 없다.

만약 사용자가 ODBC driver manager를 이용해서 프로그래밍을 하고자 할 경우에는,
odbc.ini 파일에 다음과 같은 항목을 추가해야 한다.

```
LongDataCompat = yes또는
LongDataCompat = on
```

위의 항목을 odbc.ini 파일에 추가했을 경우, SQL_BLOB, SQL_CLOB 과 같은 타입은
각각 SQL_LONGVARBINARY, SQL_LONGVARCHAR 와 같은 타입으로 변환되어서 사용자에게
전달된다. 따라서 ODBC driver manager를 사용하더라도 투명하게 LOB 데이터를 다룰
수 있다.

SQLBindFileToCol

BLOB 또는 CLOB 데이터 타입에 대해 파일 또는 파일들을 결과 집합의 열에
바인드한다.

구 문

```
SQLRETURN SQLBindFileToCol(
    SQLHSTMT              stmt,
    SQLSMALLINT           col,
    SQLCHAR *             fileName,
    SQLLEN *              fileNameLength,
    SQLUINTEGER *         fileOptions,
    SQLLEN                fileNameBufferSize,
    SQLLEN *              valueLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLSMALLINT	col	입력	바인드할 결과 집합에서의 칼럼의 순서로 1부터 시작.
SQLCHAR *	filename	입력 (유예중)	파일 이름 또는 파일 이름의 배열을 저장할 버퍼를 가리키는 포인터로 NULL일 수 없다. SQLFetch() 시 이 버퍼에 파일 이름이 저장되어 있어야 하며, SQLFetch()는 이 파일 또는 파일들에 데이터를 반환한다. 절대 경로(권장)와 상대 경로 모두 가능하다.
SQLLEN *	fileNameLength	입력 (유예중)	파일 이름의 길이 또는 길이의 배열을 저장할 버퍼를 가리키는 포인터. SQLFetch() 시 이 버퍼에 파일 이름의 길이가 저장되어 있어야 한다. 본 인자가 NULL일 경우 파일 이름을 NULL 종결 문자열로 간주한다. 즉, SQL_NTS를 본 인자가 가리키는 메모리에 저장시켜 두고 사용하는 것과 같다. 파일 이름의 최대 길이는 255이다.
SQLUINTEGER *	fileOptions	입력 (유예중)	파일 옵션 또는 옵션의 배열을 저장할 버퍼를 가리키는 포인터. SQLFetch() 시 이 버퍼에 파일 옵션이 저장되어 있어야 한다. 다음의 옵션이 가능하다. SQL_FILE_CREATE는 파일이 없을 경우 생성하고, 파일이 있을 경우 SQL_ERROR를 리턴한다. SQL_FILE_OVERWRITE는 파일이 없을 경우 생성하고, 파일이 있을 경우 파일을 덮어쓴다. SQL_FILE_APPEND는 파일이 없을 경우 생성하고, 파일이 있을 경우 파일 뒤에 덧붙여 쓴다. 위 옵션 중 한 개만 선택 가능하며, 디폴트 옵션은 정의되어있지 않다. 본 인자는 NULL일 수 없다.
SQLLEN	fileNameBufferSize	입력	fileName 버퍼의 길이를 설정한다.
SQLLEN *	valueLength	출력 (유예중)	지시자 변수 또는 지시자 변수의 배열을 저장할 버퍼를 가리키는 포인터로 NULL일 수 없다. 파일에 저장된 데이터의 길이를 반환하거나, LOB이 NULL임을 반환하기 위해 사용된다. SQLFetch()는 이 포인터가 가리키는 버퍼에 다음 값을 반환할 수 있다. 1. 데이터 길이, 2. SQL_NULL_DATA.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLBindFileToCol()은 결과 집합의 LOB 데이터를 파일로 바인드하며, SQLBindCol()은 애플리케이션 변수(메모리 버퍼)에 바인드한다.

SQLBindFileToCol() 호출 후 SQLFetch()가 호출되면, DBMS의 LOB 데이터가 파일로 저장되며, valueLength 포인터가 가리키는 버퍼에는 파일에 저장된 데이터의 길이 (바이트 단위) 가 저장된다. 만약, LOB이 NULL일 경우 valueLength 포인터가 가리키는 버퍼에는 SQL_NULL_DATA가 저장된다. fileName, fileNameLength, fileOptions 인자의 값은 SQLFetch() 시 참조되며, 인자의 오류 여부도 FETCH시 보고된다.

FETCH 시 한 번에 여러 개의 LOB을 파일로 가져오는 경우, fileName, fileNameLength, fileOptions, valueLength 인자는 모두 배열이어야 한다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLBindCol
SQLBindFileToParam
SQLDescribeCol
SQLFetch
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

한 개의 LOB을 파일에 쓰기

```
SQLCHAR fileName[16];
SQLLEN fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_CREATE;
SQLLEN valueLength;
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Select의 결과값을 가져올 파일 지정 */
strcpy(fileName, "Terminator2.avi");
if (SQLBindFileToCol(stmt, 1, fileName, &fileNameLength, &fileOptions, 16, &valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToCol : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) == SQL_SUCCESS)
{
    printf("SQLFetch success!!!\n");
}
else
{
    execute_err(dbc, stmt, “SQLFetch : “);
}
```

세 개의 LOB을 파일에 쓰기

```
SQLCHAR fileName[3][10];
SQLLEN fileNameLength[3];
SQLUINTEGER fileOptions[3];
SQLLEN valueLength[3];
.
.
.
if (SQLSetStmtAttr(stmt, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER) 3, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLSetStmtAttr(stmt, SQL_ATTR_ROW_BIND_TYPE, SQL_BIND_BY_COLUMN, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(query, "SELECT i2 FROM T1 WHERE i1 >= 1 AND i1 <= 3");

if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Select의 결과값을 가져올 파일 지정 */
strcpy(fileName[0], "Cube.avi");
strcpy(fileName[1], "Movie.avi");
strcpy(fileName[2], "Term.avi");

for (i = 0; i < 3; i++)
{
    fileNameLength[i] = strlen(fileName[i]);
    fileOptions[i] = SQL_FILE_CREATE;
}

if (SQLBindFileToCol(stmt, 1, (SQLCHAR *) fileName, fileNameLength, fileOptions, 10, valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToCol : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) == SQL_SUCCESS)
{
    printf("SQLFetch success!!!\n");
}
else
{
    execute_err(dbc, stmt, “SQLFetch : “);
```

n개의 LOB을 파일에 쓰기

```
SQLCHAR fileName[11];
SQLLEN fileNameLength = 10;
SQLUINTEGER fileOptions = SQL_FILE_OVERWRITE;
SQLLEN valueLength;
.
strcpy(query, "SELECT i2 FROM T1");

if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindFileToCol(stmt, 1, fileName, &fileNameLength, &fileOptions, 11, &valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToCol : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

for (i = 0; ; i++)
{
    sprintf(fileName, "Term%02d.avi", i + 1);

    rc = SQLFetch(stmt);
    if (rc == SQL_SUCCESS)
    {
        printf("SQLFetch of file[%02] success!!!\n", i + 1);
    }
    else if (rc == SQL_NO_DATA)
    {
        break;
    }
    else
    {
        execute_err(dbc, stmt, "SQLFetch : ");
        break;
    }
}
```

SQLindFileToParam

SQL 문에서 LOB 데이터 타입을 위해 사용된 매개변수 마커 ‘?’를 파일 또는 파일들에 바인드시킨다. SQLExecute() 또는 SQLExecDirect()가 호출될 때 자료가 파일에서 데이터베이스 관리 시스템으로 전송된다.

구 문

```
SQLRETURN SQLBindFileToParam(
    SQLHSTMT         stmt,
    SQLSMALLINT      par,
    SQLSMALLINT      sqlType,
    SQLCHAR *        fileName,
    SQLLEN *         fileNameLength,
    SQLUINTEGER *    fileOptions,
    SQLLEN           maxFileNameLength,
    SQLLEN *         ind);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLSMALLINT	Par	입력	매개변수의 순서. 1부터 시작.
SQLSMALLINT	sqlType	입력	매개변수의 SQL 데이터 타입. 다음과 같은 값을 설정할 수 있다. SQL_BLOB SQL_CLOB

자료유형	인자	사용	설명
SQLCHAR *	fileName	입력 (유예중)	파일 이름 또는 파일 이름의 배열을 저장할 버퍼를 가리키는 포인터. SQLExecute() 또는 SQLExecDirect() 시 이 버퍼에 파일 이름이 저장되어 있어야 한다. 절대 경로(권장)와 상대 경로 모두 가능하다. 본 인자는 NULL일 수 없다.
SQLLEN *	fileNameLength	입력 (유예중)	파일 이름의 길이 또는 길이의 배열을 저장할 버퍼를 가리키는 포인터. SQLExecute() 또는 SQLExecDirect() 시 이 버퍼에 파일 이름의 길이가 저장되어 있어야 한다. 본 인자가 NULL일 경우 파일 이름을 NULL 종결 문자열로 간주한다. 즉, SQL_NTS를 본 인자가 가리키는 메모리에 저장시켜 두고 사용하는 것과 같다. 파일 이름의 최대 길이는 255이다.
SQLUINTEGER *	fileOptions	입력 (유예중)	파일 옵션 또는 옵션의 배열을 저장할 버퍼를 가리키는 포인터로 NULL일 수 없다 SQLExecute() 또는 SQLExecDirect() 시 이 버퍼에 파일 옵션이 저장되어 있어야 한다. 다음과 같은 옵션이 가능하다: SQL_FILE_READ.
SQLLEN	fileNameBufferSize	입력	filename의 버퍼 길이를 설정한다.
SQLLEN *	Ind	입력 (유예중)	지시자 변수 또는 지시자 변수의 배열을 저장할 버퍼를 가리키는 포인터로 NULL일 수 없다 LOB의 NULL 여부를 지정하기 위해 사용된다. 이 포인터가 가리키는 버퍼에는 다음과 같은 값을 설정할 수 있다. 0, SQL_NULL_DATA.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

SQLBindFileToParam()은 LOB 매개변수 마커를 파일에 바인드시킨다. 매개변수 마커를 애플리케이션 변수(메모리 버퍼)에 바인드시키고 싶을 경우, SQLBindParameter()를 사용하면 된다. SQLBindFileToParam(), SQLBindParameter() 중 어느 것이든 가장 최근에 호출한 바인드 함수에 의한 바인딩만 유효하게 된다.

fileName, fileNameLength, fileOptions, ind 인자의 값은 SQLExecute() 또는 SQLExecDirect() 시 참조되므로 SQLExecute() 또는 SQLExecDirect() 호출 전에 값을 설정해주어야 한다. SQLExecute() 또는 SQLExecDirect()가 호출되면, 데이터가 바인드된 파일로부터 읽혀 DBMS로 전송된다.

LOB이 NULL일 경우 ind 포인터가 가리키는 버퍼에 SQL_NULL_DATA를 설정한 후 SQLExecute() 또는 SQLExecDirect()를 호출한다. 만약, LOB이 NULL이 아닐 경우 ind 포인터가 가리키는 버퍼에는 0을 설정해야 한다. ind 인자는 NULL 포인터여서는 안된다.

파일의 배열을 한 매개변수 마커에 바인드하는 경우, fileName, fileNameLength, fileOptions, ind 인자는 모두 배열이어야 한다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLBindCol
SQLBindFileToCol
SQLExecute
SQLExecDirect
SQLDescribeParam
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

한 개의 LOB을 테이블에 입력

```
SQLCHAR fileName[16];
SQLLEN fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_READ;
SQLLEN ind = 0;
.
strcpy(query, "INSERT INTO T1 VALUES (1, ?)");

/* Statement를 준비하고 파일을 바인드한다. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(fileName, "Terminator2.avi");
if (SQLBindFileToParam(stmt, 1, SQL_BLOB, fileName, &fileNameLength, &fileOptions, 16, &ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToParam : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

세 개의 LOB을 테이블에 입력

```
SQLINTEGER i1[3];
SQLCHAR fileName[3][10];
SQLLEN fileNameLength[3];
SQLUINTEGER fileOptions[3];
SQLLEN ind[3];
.
if (SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER) 3, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(query, "INSERT INTO T1 VALUES (?, ?)");

/* Statement를 준비하고 파일을 바인드한다. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER, SQL_INTEGER, 0, 0, (SQLPOINTER) i1, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindParameter : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindFileToParam(stmt, 2, SQL_BLOB, (SQLCHAR *) fileName, fileNameLength, fileOptions, 10, ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToParam : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Insert할 데이터를 가져올 파일 지정 */
strcpy(fileName[0], "Cube.avi");
strcpy(fileName[1], "Movie.avi");
strcpy(fileName[2], "Term.avi");

for (i = 0; i < 3; i++)
{
    i1[i] = i + 1;
    fileNameLength[i] = strlen(fileName[i]);
    fileOptions[i] = SQL_FILE_READ;
    ind[i] = 0;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

테이블의 한 개의 LOB을 업데이트

```
SQLCHAR fileName[16];
SQLLEN fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_READ;
SQLLEN ind = 0;
.
strcpy(query, "UPDATE T1 SET i2=? WHERE i1=1");

/* Statement를 준비하고 파일을 바인드한다. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(fileName, "Terminator2_fix.avi");
if (SQLBindFileToParam(stmt, 1, SQL_BLOB, fileName, &fileNameLength, &fileOptions, 16, &ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToParam : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLGetLobLength

현재의 트랜잭션 도중에 얻어진 LOB Locator가 가리키는 LOB 의 길이를 얻어온다.

구 문

```
SQLRETURN SQLGetLobLength(
    SQLHSTMT         stmt,
    SQLUBIGINT       locator,
    SQLSMALLINT      locatorCType,
    SQLUINTEGER *    valueLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLUBIGINT	locator	입력	LOB Locator
SQLSMALLINT	locatorCType	입력	LOB Locator 의 C 데이터타입 식별자로 다음의 값을 가질 수 있다. SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUINTEGER *	valueLength	출력	LOB 의 길이를 저장하기 위해 사용한다. 포인터가 가리키는 버퍼는 데이터 길이를 반환한다.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

LOB locator가 가리키는 LOB의 길이를 얻기 위해 사용되는 함수이다.

LOB locator는 데이터베이스 내의 LOB을 직접 가리키는(LOB 내에서의 오프셋 아님)
값이다. LOB locator를 얻는 방법은 두 가지가 존재한다 :

SQLBindCol()이나 SQLGetData() 함수를 통해 SELECT SQL 문의 결과 집합의 LOB 열로부터 얻을 수 있다.

이 경우, 사용자가 바인드하는 application buffer type 은 SQL_C_CLOB_LOCATOR, 혹은, SQL_C_BLOB_LOCATOR 여야 한다.

SQLBindParameter() 의 output parameter 를 통해 얻을 수 있다.

이 경우, 사용자가 바인드하는 application buffer type 은 SQL_C_CLOB_LOCATOR, 혹은, SQL_C_BLOB_LOCATOR 여야 한다.

현재의 트랜잭션 도중 얻어진 LOB locator가 아닌 경우 본 함수의 인자로 사용할 수 없다. 트랜잭션이 종료하면 LOB locator가 무효하게 되기 때문이다. 만약 유효하지 않은 LOB locator를 인자로 사용할 경우, 본 함수는 SQL_ERROR 를 리턴하며, valueLength 인자가 가리키는 버퍼는 변경되지 않는다.

valueLength 인자를 통해 LOB의 길이가 리턴된다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLBindCol
SQLBindParameter
SQLFetch
SQLExecute
SQLGetLob
SQLPutLob
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

LOB 데이터를 검색하여 길이 조회

```
SQLINTEGER valueLength;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecDirect : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_BLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindCol : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFetch : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLGetLobLength(stmt, lobLoc, SQL_C_BLOB_LOCATOR, &valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLGetLobLength : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

printf("SQLGetLobLength success!!!\n");

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFreeLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLGetLob

트랜잭션이 진행 중에 얻어진 LOB Locator가 가리키는 LOB에서 데이터의 일부분을 애플리케이션 데이터 버퍼(application data buffer)로 가져온다.

구 문

```
SQLRETURN SQLGetLob(
    SQLHSTMT        stmt,
    SQLSMALLINT     locatorCType,
    SQLUBIGINT      sourceLocator,
    SQLUINTEGER     fromPosition,
    SQLUINTEGER     forLength,
    SQLSMALLINT     targetCType,
    SQLPOINTER      value,
    SQLUINTEGER     bufferSize,
    SQLUINTEGER *   valueLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLSMALLINT	locatorCType	입력	LOB Locator 의 C 데이터타입 식별자. 다음과 같은 값을 가질 수 있다. SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUBIGINT	sourceLocator	입력	Source LOB Locator

자료유형	인자	사용	설명
SQLUINTEGER	fromPosition	입력	LOB 데이터에서 가져올 데이터의 시작 위치 (바이트 단위). 0부터 시작된다.
SQLUINTEGER	forLength	입력	LOB에서 가져올 데이터의 길이 (바이트 단위).
SQLSMALLINT	targetCType	입력	Value 버퍼의 C 데이터타입 식별자. 다음과 같은 값을 설정할 수 있다. SQL_C_BINARY SQL_C_CHAR 사용자가 BLOB 데이터를 SQL_C_CHAR 버퍼에 읽어들였을 경우 BINARY CHAR 컨버젼이 일어나며, 그 결과값이 애플리케이션 버퍼에 저장된다.
SQLPOINTER	value	출력	데이터가 저장될 버퍼의 포인터
SQLUINTEGER	bufferSize	입력	value 버퍼의 크기 (바이트 단위)
SQLUINTEGER*	valueLength	출력	value 버퍼에 저장된 데이터의 길이를 돌려받기 위한 버퍼를 가리키는 포인터. 본 인자는 NULL 일 수 없다.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

sourceLocator가 가리키는 LOB 데이터의 일부분을 서버에서 애플리케이션(application data buffer)으로 가져온다. LOB 데이터를 나누어 가져오기 위해 사용된다. LOB의 전체 길이는 SQLGetLobLength()를 호출하여 얻을 수 있다.

sourceLocator가 현재의 트랜잭션 내에서 열린 LOB locator가 아닌 경우 본 함수의 인자로 사용할 수 없다. 트랜잭션을 종료하면 LOB locator가 무효화되기 때문이다. 소스 LOB locator가 유효하지 않은 경우, SQLGetLob() 함수는 SQL_ERROR 를 리턴하며, value 및 valueLength 인자가 가리키는 버퍼는 변경되지 않는다.

sourceLocator가 NULL인 LOB을 가리킬 경우, SQLGetLob() 함수는 LOB locator의 길이가 0인 LOB을 가리키고 있는 경우와 동일하게 동작한다.

SQLGetLob() 호출 결과 반환될 데이터의 크기가 bufferSize의 버퍼 크기보다 클 경우, SQLGetLob() 함수는 SQL_SUCCESS_WITH_INFO (SQLSTATE=01004)를 리턴하며, 버퍼에 반환되는 데이터는 버퍼의 크기에 맞춰 끝부분이 잘린다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLGetLobLength
SQLPutLob
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 CLOB);
```

SQLGetLob() 함수를 이용하여, LOB 데이터를 애플리케이션 버퍼로 가져온다.

```
SQLCHAR buf[1024];
SQLINTEGER valueLength, accumLength, forLength, procLength;
SQLUBIGINT lobLoc;
.
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecDirect : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindCol : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFetch : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLGetLobLength(stmt, lobLoc, SQL_C_CLOB_LOCATOR, &valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLGetLobLength : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

for (accumLength = 0; accumLength < valueLength; accumLength += procLength)
{
    if (valueLength - accumLength > 256)
    {
        forLength = 256;
    }
    else
    {
        forLength = valueLength - accumLength;
    }

    if (SQLGetLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, accumLength, forLength, SQL_C_CHAR, buf, 256, &procLength) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, “SQLGetLob : ”);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    printf("%s", buf);
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFreeLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLPutLob

이 함수의 동작들은 모두 내부에서 갱신 작업으로 동작한다. 삽입은 길이가 0인 기존 LOB 데이터를 다른 값으로 갱신하는 동작이다. 갱신은 인자의 값에 따라 기존 데이터의 지정한 위치부터 다른 값으로 덮어쓰거나 기존 데이터 뒤에 덧붙이는 동작을 할 수 있다.

구 문

```
SQLRETURN SQLPutLob(
    SQLHSTMT        stmt,
    SQLSMALLINT     locatorCType,
    SQLUBIGINT      targetLocator,
    SQLUINTEGER     fromPosition,
    SQLUINTEGER     forLength,
    SQLSMALLINT     sourceCType,
    SQLPOINTER      value,
    SQLUINTEGER     valueLength);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLSMALLINT	locatorCType	입력	Target LOB Locator의 C 데이터 타입 식별자. SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUBIGINT	targetLocator	입력	Target LOB Locator
SQLUINTEGER	fromPosition	입력	LOB 데이터에서 갱신될 시작 위치(바이트 단위)로 0부터 시작된다.
SQLUINTEGER	forLength	입력	사용되지 않음
SQLSMALLINT	sourceCType	입력	value 버퍼의 C 데이터 타입 식별자. SQL_C_BINARY (BLOB인 경우), SQL_C_CHAR (CLOB인 경우)
SQLPOINTER	value	입력	입력 데이터를 갖고 있는 버퍼를 가리키는 포인터
SQLUINTEGER	valueLength	입력	value 버퍼에 입력한 데이터의 길이 (단위: 바이트). 0 이상의 값을 설정해야 하며, SQL_NULL_DATA는 설정할 수 없다.

결과값

```
SQL_SUCCESS
SQL_SUCCESS_WITH_INFO
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

이 함수는 애플리케이션 버퍼(application data buffer, value 인자)가 담고 있는 데이터를 타겟 LOB locator(targetLocator 인자)가 가리키는 LOB에 삽입 또는 갱신한다.

이 함수를 실행하면, 서버는 타겟 LOB의 fromPosition 위치부터의 데이터를 value 버퍼의 valueLength 길이만큼의 데이터로 덮어쓴다. valueLength가 (LOBSize – fromPosition)보다 큰 경우, 데이터베이스에서 타겟 LOB의 길이가 늘어난다. fromPosition이 타겟 LOB 값의 끝 위치를 가리키면, value 버퍼의 valueLength 길이만큼의 데이터가 기존 값 뒤에 덧붙여진다.

타겟 LOB locator가 현재의 트랜잭션에서 열린 LOB locator가 아닌 경우 본 함수의 인자로 사용할 수 없다. 트랜잭션이 종료하면 LOB locator가 무효하게 되기 때문이다. 타겟 LOB locator가 유효하지 않을 때에는, SQLPutLob() 함수는 SQL_ERROR를 리턴한다.

타겟 LOB locator가 NULL인 LOB을 가리킬 경우, SQLPutLob() 함수는 LOB locator가 길이가 0인 LOB을 가리키고 있는 경우와 동일하게 동작한다.

fromPosition 인자는 호출 시점의 타겟 LOB 길이보다 크면 안 된다. fromPosition 값이 타겟 LOB 길이보다 크면, SQLPutLob() 함수는 SQL_ERROR를 리턴한다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패

SQLSTATE	설명	부연설명
HY000	일반 오류	

관련함수

```
SQLGetLobLength
SQLGetLob
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 CLOB);
```

CLOB 칼럼 값이 'Ver.Beta'인 레코드 삽입 후 'Beta' 부분을 'Gamma'로 치환

```
SQLCHAR buf[5];
SQLUBIGINT lobLoc;
.
strcpy(query, "INSERT INTO T1 VALUES (1, 'Ver.Beta')");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1 FOR UPDATE");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFetch : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "Gamma", 5);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 4, 4, SQL_C_CHAR, buf, 5) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPutLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFreeLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

CLOB 칼럼 값이 'Ver.0.9a'인 레코드 한 개 삽입

```
SQLCHAR buf[8];
SQLINTEGER lobInd;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "INSERT INTO T1 VALUES (5, ?)");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_CLOB_LOCATOR, SQL_CLOB_LOCATOR, 0, 0, &lobLoc, 0, &lobInd) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindParameter : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecute : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "Ver.0.9a", 8);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 0, 0, SQL_C_CHAR, buf, 7) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* ‘Ver.0.9a’에서 ‘0.9’를 ‘1’로 치환 */
memcpy(buf, "1", 1);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 4, 3, SQL_C_CHAR, buf, 1) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFreeLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

여러 레코드의 CLOB 칼럼을 일괄적으로 ‘Retail’로 변경

```
SQLCHAR buf[6];
SQLINTEGER lobInd;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "UPDATE T1 SET i2=? WHERE i1>=1 AND i1<=100");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* LOB locator 파라미터를 아웃바인드하고 UPDATE 쿼리를 수행하면, 갱신 대상인 LOB 칼럼들이 자동적으로 null로 truncate  */
if (SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_CLOB_LOCATOR, SQL_CLOB_LOCATOR, 0, 0, &lobLoc, 0, &lobInd) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindParameter : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecute : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, “Retail”, 6);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 0, 0, SQL_C_CHAR, buf, 6) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFreeLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

SQLTrimLob

LOB Locator가 가리키는 LOB 값의 지정한 위치 뒤쪽 부분을 삭제한다.

구 문

```
SQLRETURN SQLTrimLob(
    SQLHSTMT         stmt,
    SQLSMALLINT      locatorCType,
    SQLUBIGINT       targetLocator,
    SQLLEN           fromPosition);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLSMALLINT	locatorCType	입력	Target LOB Locator의 C 데이터 타입 식별자. SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUBIGINT	targetLocator	입력	Target LOB Locator
SQLLEN	fromPosition	입력	삭제할 LOB 데이터의 시작 위치 (단위: 바이트). 0부터 시작된다.

결과값

```
SQL_SUCCESS
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

이 함수는 타겟 LOB locator가 가리키는 LOB 값에서 지정한 위치 뒤쪽의 데이터를
삭제하며, 삭제 후에는 타겟 LOB의 길이가 줄어든다.

타겟 LOB locator가 현재의 트랜잭션에서 열린 LOB locator가 아닌 경우 본 함수의
인자로 사용할 수 없다. 트랜잭션이 종료하면 LOB locator가 무효가 되기 때문이다.
타겟 LOB locator가 유효하지 않은 경우, SQLTrimLob() 함수는 SQL_ERROR을 리턴한다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB 서버간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLGetLobLength
SQLGetLob
```

예 제

테이블은 다음 DDL에 의해 생성되었다고 가정한다.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 CLOB);
```

CLOB 칼럼 값이 'Ver.Beta'인 레코드 삽입 후 'Beta' 부분 삭제

```
SQLCHAR buf[5];
SQLUBIGINT lobLoc;

strcpy(query, "INSERT INTO T1 VALUES (1, 'Ver.Beta')");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLExecDirect : ”);
     SQLFreeStmt(stmt, SQL_DROP);
     return SQL_ERROR;
}

strcpy(query, "SELECT i2 FROM T1 WHERE i1=1 FOR UPDATE");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLExecDirect : ”);
     SQLFreeStmt(stmt, SQL_DROP);
     return SQL_ERROR;
}
if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLBindCol : ”);
     SQLFreeStmt(stmt, SQL_DROP);

     return SQL_ERROR;
}
if (SQLFetch(stmt) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLFetch : ”);
     SQLFreeStmt(stmt, SQL_DROP);
     return SQL_ERROR;
}

if (SQLTrimLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 4) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLTrimLob : ”);
     SQLFreeStmt(stmt, SQL_DROP);
     return SQL_ERROR;
}
if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
     execute_err(dbc, stmt, “SQLFreeLob : ”);
     SQLFreeStmt(stmt, SQL_DROP);
     return SQL_ERROR;
}
```

SQLFreeLob

현재의 트랜잭션에서 열린 LOB Locator와 관련된 자원들을 해제한다.

구 문

```
SQLRETURN SQLFreeLob (
    SQLHSTMT        stmt,
    SQLUBIGINT      locator);
```

인 자

자료유형	인자	사용	설명
SQLHSTMT	stmt	입력	검색된 결과들에 대한 명령문 핸들
SQLUBIGINT	locator	입력	LOB Locator

결과값

```
SQL_SUCCESS
SQL_INVALID_HANDLE
SQL_ERROR
```

설 명

LOB locator로 대표되는 LOB에 대한 조작이 종료되었음을 서버에게 알려준다. 이로 인해 서버에서 할당된 LOB locator는 해제되며, 그와 관련된 서버의 자원들도 해제된다.

본 함수는 LOB locator가 가리키는 LOB에 대한 변경 사항을 commit 또는 rollback하지 않는다.

SQLEndTran()으로 트랜잭션을 종료한 경우, LOB locator는 자동으로 해제되므로 본 함수를 호출할 필요가 없다.

진 단

SQLSTATE	설명	부연설명
08S01	통신 회선 장애 (데이터 송수신 실패)	Altibase CLI 드라이버와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패
HY000	일반 오류	

관련함수

```
SQLGetLobLength
SQLGetLob
SQLPutLob
```

예 제

SQLGetLobLength(), SQLGetLob(), SQLPutLob() 의 예제를 참고한다.

4.커서 사용

이 장은 Altibase CLI 드라이버가 제공하는 커서의 사용법을 설명한다.

커서 특성

Altibase CLI 드라이버의 커서는 다음을 지원한다:

- 여러 유형의 커서
- 커서 내에서 스크롤 및 위치 지정
- 여러 가지 동시성 옵션
- 위치 지정 업데이트

커서의 특성은 SQL문을 실행하기 전에 SQLSetStmtAttr로 설정한 명령문 속성에 의해 제어된다. 결과 집합을 처리하는 Altibase CLI API 함수들은 fetch, 스크롤, 및 위치 지정 업데이트 같은 커서의 기능을 지원한다.

커서 특성 설정

애플리케이션에서 SQL문을 실행하기 전에 하나 이상의 커서 속성을 설정하여 커서의 동작을 제어할 수 있다. ODBC 표준에는 커서의 특성을 지정하는 아래의 두 가지 방법이 있다.

- 커서 유형
  커서 유형은 SQLSetStmtAttr의 SQL_ATTR_CURSOR_TYPE 속성을 사용하여 설정할 수 있다. Altibase CLI 드라이버는 정방향 전용(forward-only), 정적(static), 및 키집합(keyset-driven) 커서 유형을 지원한다.
- 커서 동작
  커서 동작은 SQLSetStmtAttr의 SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, 및 SQL_ATTR_CONCURRENCY 속성을 사용하여 설정할 수 있다.

커서 유형은 ODBC 스타일이며, 커서 동작은 SQL-92/ISO 스타일이다.

커서 유형

Altibase CLI 드라이버는 아래 세 가지 유형의 커서를 지원한다.

- 정방향 전용 커서: 스크롤을 지원하지 않으며, 커서의 처음부터 끝까지 순차적으로 행을 fetch하는 것만 지원한다.
- 정적 커서: 커서가 열릴 당시의 결과 집합을 표시하며, 읽기 전용이다.
- 키 집합 커서: 행 순서는 커서가 열릴 때 고정된다. 키가 아닌 칼럼의 데이터 수정 내용은 커서를 통해 볼 수 있다.

커서 동작

SQLSetStmtAttr의 SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, 및 SQL_ATTR_CONCURRENCY 속성을 사용해서 커서의 스크롤 가능 여부, 민감도, 및 동시성을 지정할 수 있다.

- 스크롤 가능 여부
  SQL_ATTR_CURSOR_SCROLLABLE이 SQL_SCROLLABLE로 설정되면, 커서는 여러 방향으로 이동 가능하다. SQL_ATTR_CURSOR_SCROLLABLE이 SQL_NONSCROLLABLE로 설정되면, 커서는 SQL_FETCH_NEXT로만 움직일 수 있다.
- 민감도
  SQL_ATTR_CURSOR_SENSITIVITY가 SQL_SENSITIVE로 설정되면, 커서는 데이터베이스의 데이터 수정 내용을 반영한다. 즉, 행 집합을 다시 가져올 필요가 경우, 데이터베이스에서 최신 데이터를 가져온다. SQL_ATTR_CURSOR_SENSITIVITY가 SQL_INSENSITIVE로 설정되면, 커서는 데이터 수정 내용을 반영하지 않는다. 즉, 행 집합을 다시 가져올 필요가 경우, 캐시에서 데이터를 가져온다.
- 동시성
  커서의 동시성은 SQLSetStmtAttr에 SQL_ATTR_CONCURRENCY 속성을 사용하여 설정된다. Altibase CLI 드라이버는 아래 두 가지 유형의 커서 동시성을 지원한다.
  - SQL_CONCUR_READONLY: 읽기 전용
  - SQL_CONCUR_ROWVER: 행 버전을 사용하여 동시성 제어

암시적 커서 변환

애플리케이션은 커서 유형을 지정하는 대신에 커서 동작 관련 속성들을 일일이 지정할 수도 있다. 그 방법은 명령문 핸들에 대해 커서를 열기 전에 SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, 및 SQL_ATTR_CONCURRENCY 명령문 속성을 설정하는 것이다. 그러면 Altibase CLI 드라이버가 애플리케이션에서 요청한 특성을 가장 효율적으로 제공하는 커서 유형을 선택한다.

애플리케이션이 SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, SQL_ATTR_CONCURRENCY, 또는 SQL_ATTR_CURSOR_TYPE 명령문 속성 중 어떤 것을 설정할 때마다, 드라이버는 네 가지 속성 값들이 양립하도록 다른 속성 값을 변경한다. 즉, 애플리케이션이 커서 동작 관련 속성을 지정하면, 드라이버는 암시적 선택에 기반하여 커서 유형을 나타내는 속성을 변경할 수 있다. 그리고, 애플리케이션이 커서 유형을 지정하면, 드라이버는 선택된 커서 유형의 특성에 부합하도록 다른 속성들을 변경할 수 있다.

애플리케이션이 커서 유형과 커서 동작 속성을 모두 지정하면 의도하지 않은 특성의 커서를 얻을 수 있으므로 주의가 필요하다.

아래의 표는 커서 유형별 기본 커서 동작을 보여준다.

커서 유형	민감도	스크롤 가능 여부	동시성
정방향 전용 (forward-only)	insensitive	non-scrollable	read-only
정적 (static)	insensitive	scrollable	read-only
키집합 (keyset-driven)	sensitive	scrollable	updatable
동적 (dynamic)	-	-	-

Altibase CLI 드라이버는 동적 커서는 지원하지 않는다. 또한, 정방향 전용과 정적 커서의 경우, 위 표의 조합 외에 다른 동작의 조합은 지원하지 않는다. 키 집합 커서의 경우, 위 표의 조합 외에 (SENSITIVE, SCROLLABLE, READ-ONLY) 조합을 지원한다.

커서 변환 규칙은 아래와 같다.

- 사용자가 지정한 커서 속성이 커서 유형과 부합하지 않으면, 드라이버는 커서 유형을 아래의 순서로 변환한다.
  - dynamic → keyset-driven → static → foward-only
- 애플리케이션이 커서 유형을 지정하면, 드라이버는 선택된 유형의 특성에 부합할 때까지 다른 속성들을 아래의 순서로 변환한다.
  - sensitive → insensitive
  - scrollable → non-scrollable
  - updatable → read-only

행 스크롤 및 Fetch

스크롤 가능한 커서를 사용하려면 Altibase CLI 애플리케이션에서 다음을 수행해야 한다.

- SQLSetStmtAttr을 사용하여 커서 속성을 설정한다.
- SQLExecute 또는 SQLExecDirect를 사용하여 커서를 연다.
- SQLFetch 또는 SQLFetchScroll을 이용하여 행을 스크롤하고 가져온다.

ODBC 커서는 한 번에 한 행씩만 가져가도록 제한하지는 않는다. SQLFetch 또는 SQLFetchScroll을 호출할 때마다 여러 행을 가져올 수 있다. ODBC 응용 프로그램 같은 클라이언트/서버 구조의 데이터베이스 작업 시에는 한 번에 여러 행을 가져오는 것이 효율적이다. fetch 시 반환되는 행의 수를 행 집합 크기라고 하며 SQLSetStmtAttr의 SQL_ATTR_ROW_ARRAY_SIZE를 사용하여 지정할 수 있다.

행 집합 크기가 1보다 큰 커서를 블록 커서라고 하며, 배열 바인딩을 사용해서 블록 커서에서 데이터를 검색할 수 있다. 이에 대한 자세한 내용은 2장의 SQLFetch 함수를 참고한다.

SQLFetch는 정방향 전용 커서만 사용 가능하며, SQLFetchScroll는 커서를 여러 방향으로 이동할 수 있다.

행 집합에 북마크 설정

북마크는 데이터의 행을 식별하는 데 사용되는 값이다. Altibase CLI 애플리케이션에서는 특정 행에 대해 북마크를 요청하고 이를 저장한 다음 다시 이것을 커서에 전달하여 그 행으로 돌아간다.

SQLFetchScroll을 사용하여 행을 가져올 때, 애플리케이션에서는 시작 행을 선택하기 위한 기준으로 북마크를 사용할 수 있다. 이 북마크는 현재 커서 위치를 사용하지 않기 때문에 절대 주소 형태로 되어 있다. 애플리케이션에서는 SQL_FETCH_BOOKMARK로 SQLFetchScroll을 호출하여 북마크가 표시된 행으로 스크롤할 수 있다. 이 작업은 SQL_ATTR_FETCH_BOOKMARK_PTR 명령문 속성으로 설정한 북마크를 사용한다. 즉, 북마크로 식별된 행부터 행집합을 반환한다. 애플리케이션에서 SQLFetchScroll 호출 시 인자에 오프셋을 지정할 수 있다. 오프셋이 지정되면, 반환되는 행집합의 첫 번째 행은 북마크로 식별된 행에서 오프셋 수만큼 더하여 결정된다. Altibase CLI 드라이버는 정적 및 키집합 커서에 대해서만 북마크를 지원한다. 북마크가 설정되어 있을 때, 동적 커서가 요청되면 키집합 커서가 대신 열린다.

제약 사항

Altibase CLI 드라이버의 커서를 사용하려면, 결과 집합을 가져오는 SELECT 문에 아래와 같은 제약 사항이 있다.

- FROM 절에 한 개의 테이블만 지정할 수 있다.
- ORDER BY 절에 반드시 칼럼 이름을 명시해야 한다.

커서 지정 업데이트 수행을 하기 위해서는 아래의 제약 사항이 추가된다.

- FROM 절에 일반 테이블만 지정할 수 있다.
- SELECT 리스트에 순수 칼럼만 지정할 수 있다. 수식 또는 함수가 포함될 수 없다. 그리고, NOT NULL 제약조건이 있으면서 DEFAULT값이 없는 칼럼은 SELECT 리스트에 반드시 포함되어야 한다.

A.부록: Sample Code

이 부록은 본 매뉴얼에서 전반적으로 사용된 예제에 대하여 전체 코드 리스트를 제공한다.

프로그래밍 시 각 단계에서 주의할 점

Altibase CLI 애플리케이션 작성시 주의할 점과 범하기 쉬운 오류에 대해서 설명한다.

파라미터 바인딩

이 절은 SQLBindCol()과 SQLBindParameter() 함수의 마지막 인자인 *valueLength* (지시자, indicator) 사용 시의 주의 사항에 대해 설명한다.

SQLBindCol() 함수의 *valueLength*는 출력 인자이다. 조회된 데이터가 NULL 이면, 이 인자 변수에 SQL_NULL_DATA 가 반환된다.

SQLBindParameter() 함수의 바인딩 입출력 타입을 SQL_PARAM_INPUT 로 지정하면, *valueLength*는 입력 인자로 사용된다. 바인딩되는 인자 변수의 데이터 타입이 가변 길이 타입일 경우 실제 바인딩되는 데이터의 길이를 표시하는 데 사용된다.

예를 들어, *valueLength*에 아래의 값을 지정할 수 있다.

- SQL_NTS: 버퍼가 null ('\0') 로 끝나는 스트링
- SQL_NULL_DATA: 널 값을 바인딩
- SQL_DATA_AT_EXEC 혹은 SQL_LEN_DATA_AT_EXEC() 매크로의 결과값: 사용자가 SQLPutData(), SQLParamData() 의 조합으로 데이터를 삽입하겠다는 의미로써, SQLExecute() 실행시 SQL_NEED_DATA 를 리턴한다.

SQLBindParameter()에서 입출력 타입을 SQL_PARAM_OUTPUT 로 지정할 경우의 *valueLength*는 SELECT 문 실행시 사용되는 SQLBindCol() 함수의 *valueLength*와 동일한 역할을 한다.

트랜잭션 Commit

autocommit off의 세션에서 commit을 하지 않고 응용프로그램을 종료할 경우 commit 하지 않은 트랜잭션은 모두 Altibase 서버에 의해 롤백(rollback) 된다. 그러나, SQLDisconnect() 를 호출하고 응용프로그램을 종료할 경우에는 Altibase 서버가 커밋되지 않은 트랜잭션들을 commit 한다.

SQL 함수 사용

SQLFreeStmt()에서 두 번째 인자를 SQL_DROP으로 했을 경우에는 핸들의 상태가 핸들을 할당하기 이전의 상태가 된다. 즉, SQLFreeHandle()과 완전히 동일한 동작을 수행한다. 그러나, SQL_CLOSE 인자를 사용하면 SQLCloseCursor()와 완전히 동일한 동작을 수행한다. SQLPrepare()를 수행한 명령문의 경우에는, SQLCloseCursor() 호출 후에 준비(prepare)를 한 상태로 돌아가게 된다.

SQLPrepare()를 사용하여 SELECT 문을 수행할 경우, SQLExecute() -> SQLFetch() 시 SELECT 문의 수행 결과, 결과 집합의 마지막 레코드까지 페치를 하지 않고, 다른 호스트 변수를 바인딩해서 SQLExecute()를 다시 호출할 경우에 “invalid cursor state”가 발생한다. 이를 방지하기 위해서는 SQLCloseCursor()를 호출하고 나서 SQLExecute()를 호출하면 된다.

Altibase CLI 프로그램 기본 예제

```
/*************************************************
**  파일명 = demo_ex1.cpp
*************************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>


#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN execute_select();
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    free_handle();
}


static void print_diagnostic(SQLSMALLINT aHandleType, SQLHANDLE aHandle)
{
    SQLRETURN   rc;
    SQLSMALLINT sRecordNo;
    SQLCHAR     sSQLSTATE[6];
    SQLCHAR     sMessage[2048];
    SQLSMALLINT sMessageLength;
    SQLINTEGER  sNativeError;

    sRecordNo = 1;

    while ((rc = SQLGetDiagRec(aHandleType,
                               aHandle,
                               sRecordNo,
                               sSQLSTATE,
                               &sNativeError,
                               sMessage,
                               sizeof(sMessage),
                               &sMessageLength)) != SQL_NO_DATA)
```

```
    {
        printf("Diagnostic Record %d\n", sRecordNo);
        printf("     SQLSTATE     : %s\n", sSQLSTATE);
        printf("     Message text : %s\n", sMessage);
        printf("     Message len  : %d\n", sMessageLength);
        printf("     Native error : 0x%X\n", sNativeError);

        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        {
            break;
        }

        sRecordNo++;
    }
}

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    printf("Error : %s\n",q);

    if (aStmt == SQL_NULL_HSTMT)
    {
        if (aCon != SQL_NULL_HDBC)
        {
            print_diagnostic(SQL_HANDLE_DBC, aCon);
        }
    }
    else
    {
        print_diagnostic(SQL_HANDLE_STMT, aStmt);
    }
}

SQLRETURN alloc_handle()
{
    /* allocate Environment handle */
    if (SQLAllocEnv(&env) != SQL_SUCCESS)
    {
        printf("SQLAllocEnv error!!\n");
        return SQL_ERROR;
    }

    /* allocate Connection handle */
    if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
    {
        printf("SQLAllocConnect error!!\n");
        return SQL_ERROR;
    }
    return SQL_SUCCESS;
}

void free_handle()
{
    if ( conn_flag == 1 )
    {
        /* close connection */
        SQLDisconnect( dbc );
    }
    /* free connection handle */
    if ( dbc != NULL )
    {
        SQLFreeConnect( dbc );
    }
    if ( env != NULL )
    {
        SQLFreeEnv( env );
    }
}

SQLRETURN db_connect()
{
    char    *USERNAME = "SYS";        // user name
    char    *PASSWD   = "MANAGER";    // user password
    char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
    char     connStr[1024];

    sprintf(connStr,
```

```
            "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /* ;PORT_NO=20300", */
            USERNAME, PASSWD, 1, NLS);

    /* establish connection */
    if (SQLDriverConnect( dbc, NULL, (SQLCHAR *)connStr, SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
    {
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}

SQLRETURN execute_select()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;
    int          i;
    char         query[SQL_LEN];

    SQLSMALLINT  columnCount;
    char         columnName[50];
    SQLSMALLINT  columnNameLength;
    SQLSMALLINT  dataType;
    SQLSMALLINT  scale;
    SQLSMALLINT  nullable;
    SQLULEN      columnSize;

    void       **columnPtr;
    SQLLEN      *columnInd;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"SELECT * FROM DEMO_EX1");
    if (SQLExecDirect(stmt, (SQLCHAR *)query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    SQLNumResultCols(stmt, &columnCount);
    columnPtr = (void**) malloc( sizeof(void*) * columnCount );
    columnInd = (SQLLEN*) malloc( sizeof(SQLLEN) * columnCount );
    if ( columnPtr == NULL )
    {
        return SQL_ERROR;
    }

    for ( i=0; i<columnCount; i++ )
    {
        SQLDescribeCol(stmt, i+1,
                       (SQLCHAR *)columnName, sizeof(columnName), &columnNameLength,
                       &dataType, &columnSize, &scale, &nullable);
        printf("columnName = %s, nullable = %d\n", columnName, nullable);
        switch (dataType)
        {
        case SQL_CHAR:
            printf("%s : CHAR(%d)\n", columnName, columnSize);
            columnPtr[i] = (char*) malloc( columnSize + 1 );
            SQLBindCol(stmt, i+1, SQL_C_CHAR, columnPtr[i], columnSize+1, &columnInd[i]);
            break;
        case SQL_VARCHAR:
            printf("%s : VARCHAR(%d)\n", columnName, columnSize);
            columnPtr[i] = (char*) malloc( columnSize + 1 );
            SQLBindCol(stmt, i+1, SQL_C_CHAR, columnPtr[i], columnSize+1, &columnInd[i]);
            break;
        case SQL_INTEGER:
```

```
        printf("%s : INTEGER\n", columnName);
        columnPtr[i] = (int*) malloc( sizeof(int) );
        SQLBindCol(stmt, i+1, SQL_C_SLONG, columnPtr[i], 0, &columnInd[i]);
        break;
    case SQL_SMALLINT:
        printf("%s : SMALLINT\n", columnName);
        columnPtr[i] = (short*) malloc( sizeof(short) );
        SQLBindCol(stmt, i+1, SQL_C_SSHORT, columnPtr[i], 0, &columnInd[i]);
        break;
    case SQL_NUMERIC:
        printf("%s : NUMERIC(%d,%d)\n", columnName, columnSize, scale);
        columnPtr[i] = (double*) malloc( sizeof(double) );
        SQLBindCol(stmt, i+1, SQL_C_DOUBLE, columnPtr[i], 0, &columnInd[i]);
        break;
    case SQL_TYPE_TIMESTAMP:
        printf("%s : DATE\n", columnName);
        columnPtr[i] = (SQL_TIMESTAMP_STRUCT*) malloc( sizeof(SQL_TIMESTAMP_STRUCT) );
        SQLBindCol(stmt, i+1, SQL_C_TYPE_TIMESTAMP, columnPtr[i], 0, &columnInd[i]);
        break;
    }
}

/* fetches next rowset of data from the result set and print to stdout */
printf("================================================================================\n");
while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
    if ( rc != SQL_SUCCESS )
    {
        execute_err(dbc, stmt, query);
        break;
    }
    for ( i=0; i<columnCount; i++ )
    {
        SQLDescribeCol(stmt, i+1,
                       NULL, 0, NULL,
                       &dataType, NULL, NULL, NULL);
        if ( columnInd[i] == SQL_NULL_DATA )
        {
            printf("NULL\t");
            continue;
        }
        switch (dataType)
        {
        case SQL_CHAR:
        case SQL_VARCHAR:
            printf("%s\t", columnPtr[i]);
            break;
        case SQL_INTEGER:
            printf("%d\t", *(int*)columnPtr[i]);
            break;
        case SQL_SMALLINT:
            printf("%d\t", *(short*)columnPtr[i]);
            break;
        case SQL_NUMERIC:
            printf("%10.3f\t", *(double*)columnPtr[i]);
            break;
        case SQL_TYPE_TIMESTAMP:
            printf("%4d/%02d/%02d %02d:%02d:%02d\t",
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->year,
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->month,
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->day,
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->hour,
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->minute,
                    ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->second);
            break;
        }
    }
    printf("\n");
}

SQLFreeStmt(stmt, SQL_DROP);

for ( i=0; i<columnCount; i++ )
{
    free( columnPtr[i] );
}
free( columnPtr );
```

```
    free( columnInd );

    return SQL_SUCCESS;
}
```

메타 정보 검색 프로그램 예제

```
/***********************************************
**  파일명 = demo_meta1.cpp
**  메타 정보 검색 프로그램 예제
***********************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>

#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN get_tables();
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    rc = get_tables();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    free_handle();
}


static void print_diagnostic(SQLSMALLINT aHandleType, SQLHANDLE aHandle)
{
    SQLRETURN   rc;
    SQLSMALLINT sRecordNo;
    SQLCHAR     sSQLSTATE[6];
    SQLCHAR     sMessage[2048];
    SQLSMALLINT sMessageLength;
    SQLINTEGER  sNativeError;

    sRecordNo = 1;

    while ((rc = SQLGetDiagRec(aHandleType,
                               aHandle,
                               sRecordNo,
                               sSQLSTATE,
                               &sNativeError,
                               sMessage,
                               sizeof(sMessage),
                               &sMessageLength)) != SQL_NO_DATA)
```

```
    {
        printf("Diagnostic Record %d\n", sRecordNo);
        printf("     SQLSTATE     : %s\n", sSQLSTATE);
        printf("     Message text : %s\n", sMessage);
        printf("     Message len  : %d\n", sMessageLength);
        printf("     Native error : 0x%X\n", sNativeError);

        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        {
            break;
        }

        sRecordNo++;
    }
}

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    printf("Error : %s\n",q);

    if (aStmt == SQL_NULL_HSTMT)
    {
        if (aCon != SQL_NULL_HDBC)
        {
            print_diagnostic(SQL_HANDLE_DBC, aCon);
        }
    }
    else
    {
        print_diagnostic(SQL_HANDLE_STMT, aStmt);
    }
}


SQLRETURN alloc_handle()
{
    /* allocate Environment handle */
    if (SQLAllocEnv(&env) != SQL_SUCCESS)
    {
        printf("SQLAllocEnv error!!\n");
        return SQL_ERROR;
    }

    /* allocate Connection handle */
    if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
    {
        printf("SQLAllocConnect error!!\n");
        return SQL_ERROR;
    }
    return SQL_SUCCESS;
}

void free_handle()
{
    if ( conn_flag == 1 )
    {
        /* close connection */
        SQLDisconnect( dbc );
    }
    /* free connection handle */
    if ( dbc != NULL )
    {
        SQLFreeConnect( dbc );
    }
    if ( env != NULL )
    {
        SQLFreeEnv( env );
    }
}

SQLRETURN db_connect()
{
    char    *USERNAME = "SYS";        // user name
    char    *PASSWD   = "MANAGER";    // user password
    char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
    char     connStr[1024];
```

```
    sprintf(connStr,
            "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /* ;PORT_NO=20300", */
            USERNAME, PASSWD, 1, NLS);

    /* establish connection */
    if (SQLDriverConnect( dbc, NULL, (SQLCHAR *) connStr, SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
    {
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}

SQLRETURN get_tables()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;

    char         schem[50+1] = {0};
    char         name[50+1] = {0};
    char         type[50+1] = {0};
    SQLLEN       schem_ind;
    SQLLEN       name_ind;
    SQLLEN       type_ind;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt))
    {
        printf("SQLAllocHandle error!!\n");
        return SQL_ERROR;
    }

    if (SQLTables(stmt,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLTables");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                   schem, sizeof(schem), &schem_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 3, SQL_C_CHAR,
                   name, sizeof(name), &name_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 4, SQL_C_CHAR,
                   type, sizeof(type), &type_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* fetches the next rowset of data from the result set and print to stdout */
    printf("TABLE_SCHEM\t\tTABLE_NAME\t\tTABLE_TYPE\n");
    printf("============================================================\n");
    while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
    {
```

```
        if ( rc == SQL_ERROR )
        {
            execute_err(dbc, stmt, "SQLFetch");
            break;
        }
        printf("%-40s%-40s%s\n", schem, name, type);
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);/* == SQLFreeStmt(stmt, SQL_DROP); */

    return SQL_SUCCESS;
}
```

프로시저 테스트 프로그램 예제

```
/************************************************
**  파일명 = proctest.cpp
**  procedure 테스트 프로그램 예제
************************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>


#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN execute_proc();
SQLRETURN execute_select();

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* select data */
    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* procedure 실행 */
    rc = execute_proc();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* select data */
    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* disconnect, free handles */
    free_handle();
}
```

```
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[MSG_LEN];

    printf("Error : %s\n",q);

    if (SQLError ( SQL_NULL_HENV, aCon, aStmt,
                   NULL, &errNo,
                   errMsg, MSG_LEN, &msgLength ) == SQL_SUCCESS)
    {
        printf(" Error : # %ld, %s\n", errNo, errMsg);
    }
}

SQLRETURN alloc_handle()
{
    /* allocate Environment handle */
    if (SQLAllocEnv(&env) != SQL_SUCCESS)
    {
        printf("SQLAllocEnv error!!\n");
        return SQL_ERROR;
    }

    /* allocate Connection handle */
    if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
    {
        printf("SQLAllocConnect error!!\n");
        return SQL_ERROR;
    }
    return SQL_SUCCESS;
}

void free_handle()
{
    if ( conn_flag == 1 )
    {
        /* close connection */
        SQLDisconnect( dbc );
    }
    /* free connection handle */
    if ( dbc != NULL )
    {
        SQLFreeConnect( dbc );
    }
    if ( env != NULL )
    {
        SQLFreeEnv( env );
    }
}

SQLRETURN db_connect()
{
    char    *USERNAME = "SYS";        // user name
    char    *PASSWD   = "MANAGER";    // user password
    char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
    char     connStr[1024];

    sprintf(connStr,
            "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /* ;PORT_NO=20300", */
            USERNAME, PASSWD, 1, NLS);

    /* establish connection */
    if (SQLDriverConnect( dbc, NULL, (SQLCHAR *) connStr, SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
    {
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}
```

```
SQLRETURN execute_select()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;
    char         query[SQL_LEN];

    SQLINTEGER           id;
    char                 name[20+1];
    SQL_TIMESTAMP_STRUCT birth;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"SELECT * FROM DEMO_EX6");
    if (SQLPrepare(stmt, (SQLCHAR *)query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* binds application data buffers to columns in the result set */
    if (SQLBindCol(stmt, 1, SQL_C_SLONG,
                   &id, 0, NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                   name, sizeof(name), NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindCol(stmt, 3, SQL_C_TYPE_TIMESTAMP,
                   &birth, 0, NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* fetches the next rowset of data from the result set and print to stdout */
    printf("id\t        Name\tbirth\n");
    printf("=============================================================================\n");
    if ( SQLExecute(stmt) != SQL_SUCCESS )
    {
        execute_err(dbc, stmt, "SQLExecute : ");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA )
    {
        if ( rc != SQL_SUCCESS )
        {
            execute_err(dbc, stmt, query);
            break;
        }
        printf("%d%20s\t%4d/%02d/%02d %02d:%02d:%02d\n",
                id, name, birth.year, birth.month, birth.day,
                birth.hour, birth.minute, birth.second);
    }

    SQLFreeStmt(stmt, SQL_DROP);

    return SQL_SUCCESS;
```

```
}

SQLRETURN execute_proc()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    char         query[SQL_LEN];

    SQLINTEGER           id;
    char                 name[20+1];
    SQL_TIMESTAMP_STRUCT birth;
    SQLINTEGER           ret = 0;

    SQLLEN               name_ind = SQL_NTS;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"EXEC DEMO_PROC6( ?, ?, ?, ? )");

    /* prepares an SQL string for execution */
    if (SQLPrepare(stmt, (SQLCHAR *) query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindParameter(stmt,
                         1, /* Parameter number, starting at 1 */
                         SQL_PARAM_INPUT, /* in, out, inout */
                         SQL_C_SLONG, /* C data type of the parameter */
                         SQL_INTEGER, /* SQL data type of the parameter : char(8)*/
                         0,          /* size of the column or expression, precision */
                         0,          /* The decimal digits, scale */
                         &id,        /* A pointer to a buffer for the parameter's data */
                         0,          /* Length of the ParameterValuePtr buffer in bytes */
                         NULL        /* indicator */
                         ) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
                         SQL_C_CHAR, SQL_VARCHAR,
                         20,  /* varchar(20) */
                         0,
                         name, sizeof(name), &name_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindParameter(stmt, 3, SQL_PARAM_INPUT,
                         SQL_C_TYPE_TIMESTAMP, SQL_DATE,
                         0, 0, &birth, 0, NULL) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindParameter(stmt, 4, SQL_PARAM_OUTPUT,
                         SQL_C_SLONG, SQL_INTEGER,
                         0, 0, &ret,
                         0,/* For all fixed size C data type, this argument is ignored */
                         NULL) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* executes a prepared statement */
```

```
    id = 5;
    sprintf(name, "name5");
    birth.year=2004;birth.month=5;birth.day=14;
    birth.hour=15;birth.minute=17;birth.second=20;
    birth.fraction=0;
    name_ind = 5;            /* name => length=5 */
    if (SQLExecute(stmt) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    else
    {
        printf("\n======= Result of exec procedure ======\n");
        printf("ret => %d\n\n", ret);
    }

    SQLFreeStmt(stmt, SQL_DROP);

    return SQL_SUCCESS;
}
```

B.부록: 데이터형

이 부록은 Altibase의 데이터 타입과 SQL 데이터 타입, C 데이터 타입의 종류를 설명하고, 각각의 데이터 타입간에 변경이 가능한지 여부를 나타낸다.

SQL 데이터형

ODBC 드라이버와 Altibase CLI가 Altibase SQL 데이터 타입을 ODBC 스펙에 정의된 SQL type 식별자들에 어떻해 대응하는가에 대한 정보와 Altibase SQL 데이터 타입을 ODBC 스펙의 SQL type 식별자들에 어떻게 대응하는가에 대한 정보는 SQLGetTypeInfo()를 호출함으로서 반환된다.

ODBC 드라이버와 Altibase CLI가 어떤 데이터 타입을 지원하는가는 SQLGetTypeInfo()를 호출함으로서 알 수 있다.

다음 테이블은 Altibase CLI가 지원하는 SQL 데이터 타입들에 대한 유효한 SQL type 식별자들의 목록이다.

SQL type 식별자	ALTIBASE 데이터 타입	부연 설명
SQL_CHAR	CHAR(n)	고정된 길이(n)의 데이터 타입
SQL_VARCHAR	VARCHAR(n)	가변 길이의 데이터 타입: FIXED로 선언했을 경우는 명시된 크기 만큼 저장소의 크기가 설정, VARIABLE로 선언했을 경우는 명시된 크기 내에서 가변 길이를 가짐
SQL_WCHAR	NCHAR(n)	고정된 길이(n)의 유니코드 문자형 데이터 타입. N은 문자 개수임
SQL_WVARCHAR	NVARCHAR(n)	가변 길이의 유니코드 문자형 데이터 타입: FIXED로 선언했을 경우는 명시된 크기 만큼 저장소의 크기가 설정, VARIABLE로 선언했을 경우는 명시된 크기 내에서 가변 길이를 가짐. N은 문자 개수임
SQL_DECIMAL	DECIMAL(p, s)	NUMERIC 데이터 타입과 동일한 데이터 타입
SQL_NUMERIC	NUMERIC(p, s)	Precision과 scale을 가지는 숫자형 데이터 타입으로 precision 만큼의 유효 숫자와 scale 만큼의 소수점이하 정밀도를 가지는 고정 소수점형 (1<=p<=38, -84<=s<=126)
SQL_SMALLINT	SMALLINT	2 bytes 크기의 데이터 타입 (-2^15+1 ~ 2^15-1)
SQL_INTEGER	INTEGER	4 bytes 크기의 데이터 타입 (-2^31+1 ~ 2^31-1)
SQL_BIGINT	BIGINT	8 bytes 크기의 데이터 타입 (-2^63+1 ~2^63-1)
SQL_REAL	REAL	C의 FLOAT과 동일한 데이터 타입
SQL_FLOAT	FLOAT(p)	-1E+120에서 1E+120까지의 부동 소수점 숫자 데이터 (1<=p<=38)

SQL type 식별자	ALTIBASE 데이터 타입	부연 설명
SQL_DOUBLE	DOUBLE	C의 DOUBLE과 동일한 데이터 타입
SQL_BLOB	BLOB	최대 4GB-1Byte 길이를 가지는 가변길이 이진 데이터 타입
SQL_CLOB	CLOB	최대 4GB-1Byte 길이를 가지는 가변길이 문자 데이터 타입
SQL_TYPE_DATE	DATE	날짜를 표현하는 데이터 타입
SQL_TYPE_TIME	DATE	날짜를 표현하는 데이터 타입
SQL_TYPE_TIMESTAMP	DATE	날짜를 표현하는 데이터 타입
SQL_INTERVAL	-	DATE – DATE의 결과 타입
SQL_BYTES	BYTE(n)	명시된 크기(n)만큼 고정된 길이를 가지는 이진 데이터 타입 (1 byte<=n<=32000 bytes)
SQL_NIBBLE	NIBBLE(n)	명시된 크기(n)만큼 가변 길이를 가지는 이진 데이터 타입 (1 character<=n<=254 characters)
SQL_GEOMETRY	GEOMETRY	내부적으로 7개의 데이터 타입(Point, LineString, Polygon, GeometryCollection, MultiLineString, MultiPolygon, MultiPoint)의 속성을 함축하고 있으며 이 함축된 속성은 각 타입에 따라 지원되는 함수로서 구분 가능: FIXED로 선언했을 경우는 고정길이의 저장 공간을 사용하고, VARIABLE로 선언했을 경우는 가변 길이의 저장 공간을 사용

C 데이터형

C 데이터 타입들은 애플리케이션에서 데이터를 저장하기 위해 사용되는 C 버퍼의 데이터 타입을 나타낸다.

C 데이터 타입은 *type* 인자와 함께 SQLBindCol()과 SQLGetData()에 그리고 cType과 함께 SQLBindParameter()에 명시된다.

다음 테이블은 C 데이터 타입에 대해 유효한 타입 식별자들의 목록이다. 또한, 테이블은 각 식별자에 해당하는 ODBC 스펙의 C 데이터 타입과 이 데이터 타입의 정의를 나열한다.

C Type 식별자	ODBC C typedef	C type
SQL_C_CHAR	SQLSCHAR *	char *
SQL_C_WCHAR	SQLWCHAR *	short *
SQL_C_STINYINT	SQLSCHAR	signed char
SQL_C_UTINYINT	SQLCHAR	unsigned char
SQL_C_SBIGINT	SQLBIGINT	_int64
SQL_C_UBIGINT	SQLUBIGINT	unsigned _int64
SQL_C_SSHORT	SQLSMALLINT	short int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_SLONG	SQLINTEGER	int
SQL_C_ULONG	SQLUINTEGER	unsigned int
SQL_C_FLOAT	SQLREAL	float
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_BINARY	SQLCHAR *	unsigned char *
C Type 식별자	ODBC C typedef	C type
SQL_C_TYPE_DATE	SQL_DATE_STRUCT	struct tagDATE_STRUCT { SQLSMALLINT year; SQLSMALLINT month; SQLSMALLINT day; } DATE_STRUCT

C Type 식별자	ODBC C typedef	C type
SQL_C_TYPE_TIME	SQL_TIME_STRUCT	struct tagTIME_STRUCT { SQLSMALLINT hour; SQLSMALLINT minute; SQLSMALLINT second; } TIME_STRUCT
SQL_C_TYPE_TIMESTAMP	SQL_TIMESTAMP_STRUCT	struct tagTIMESTAMP_STRUCT {SQLSMALLINT year; SQLSMALLINT month; SQLSMALLINT day; SQLSMALLINT hour; SQLSMALLINT minute; SQLSMALLINT second; SQLINTEGER fraction; **}** TIMESTAMP_STRUCT;
SQL_C_NIBBLE	SQL_NIBBLE_STRUCT	struct tagNIBBLE_STRUCT { SQLCHAR length; SQLCHAR value[1]; } NIBBLE_STRUCT

SQL 데이터형을 C 데이터형으로 변환하기

Converting Data from SQL to C Data types

	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_BIT	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_SBIGINT
SQL_CHAR	#		○	○	○	
SQL_VARCHAR	#		○	○	○	
SQL_WCHAR		#	○	○	○	
SQL_WVARCHAR		#	○	○	○	
SQL_DECIMAL	#		○	○	○	○
SQL_NUMERIC	#		○	○	○	○
SQL_SMALLINT	○		○	○	○	○
(signed)						
SQL_INTEGER	○		○	○	○	○
(signed)						
SQL_BIGINT	○		○	○	○	#
(signed)						
SQL_REAL	○		○	○	○	○
SQL_FLOAT	#		○	○	○	○
SQL_DOUBLE	○		○	○	○	○
SQL_BINARY	○					
SQL_TYPE_DATE	○					
SQL_TYPE_TIME	○					
SQL_TYPE_TIMESTAMP	○					
SQL_INTERVAL	○					
SQL_BYTES	○					
SQL_NIBBLE	○					
SQL_GEOMETRY						

: Default conversion

○ : Supported conversion

C 데이터형을 SQL 데이터형으로 변환하기

Converting Data from C to SQL Data types

	SQL_CHAR	SQL_VARCHAR	SQL_WCHAR	SQL_WVARCHAR	SQL_DECIMAL	SQL_NUMERIC	SQ
SQL_C_CHAR	#	#			#	#	○
SQL_C_WCHAR			#	#			
SQL_C_BIT	○	○	○	○	○	○	○
SQL_C_STINYINT	○	○	○	○	○	○	○
SQL_C_UTINYINT	○	○	○	○	○	○	○
SQL_C_SBIGINT	○	○	○	○	○	○	○
SQL_C_UBIGINT	○	○	○	○	○	○	○
SQL_C_SSHORT	○	○	○	○	○	○	#
SQL_C_USHORT	○	○	○	○	○	○	○
SQL_C_SLONG	○	○	○	○	○	○	○
SQL_C_ULONG	○	○	○	○	○	○	○
SQL_C_FLOAT	○	○	○	○	○	○	○
SQL_C_DOUBLE	○	○	○	○	○	○	○
SQL_C_BINARY	○	○	○	○			
SQL_C_TYPE_DATE							
SQL_C_TYPE_TIME							
SQL_C_TYPE_TIMESTAMP							
SQL_C_BYTES							
SQL_C_NIBBLE							

: Default conversion

○ : Supported conversion

C.부록: 오류 코드

SQLError는 X/Open 및 SQL Access Group SQL CAE 규격(1992)과 ODBC 스펙에서 정의한 대로 SQLSTATE 값을 반환한다. SQLSTATE 값은 다섯 문자를 포함하는 문자열이다. 이 부록에서는 Altibase CLI와 ODBC를 위한 SQLSTATE 값을 설명한다.

SQLSTATE

다음 테이블은 Altibase CLI 드라이버의 SQLError가 반환할 수 있는 SQLSTATE 값을 표시한다.

SQLSTATE	Error	Can be returned from
01004	자료가 잘림 (반환 될 값의 크기가 주어진 버퍼의 크기보다 클 경우)	SQLDescribeCol SQLFetch SQLGetData
07006	제한된 데이터 타입 속성 위반	SQLBindParameter SQLExecute SQLFetch
07009	유효하지 않은 설명자 인덱스	SQLBindCol SQLBindParameter SQLColAttribute SQLDescribeCol SQLDescribeParam SQLGetData

> * SQLSTATE 08001, 08002, 08003, 08S01은 아래 참고

SQLSTATE	Error	Can be returned from
HY000	일반 오류	SQLAllocStmt SQLAllocConnect SQLBindCol SQLBindParameter SQLColAttribute SQLColumns SQLConnect SQLDescribeCol SQLDisconnect SQLDriverConnect SQLEndTran SQLExecDirect SQLExecute SQLFetch SQLFreeHandle SQLFreeStmt SQLGetData SQLNumParams SQLNumResultCols SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLRowCount SQLSetAttribute SQLSetConnectAttr SQLSetEnvAttr SQLStatistics SQLTables
HY001	메모리 할당 오류 (ODBC가 함수를 실행하고 완료하기 위해 요구된 메모리를 할당할 수 없음)	SQLAllocConnect SQLAllocStmt SQLBindCol SQLBindParameter SQLConnect SQLDriverConnect SQLExecDirect SQLGetTypeInfo SQLPrepare
HY003	애플리케이션 버퍼 타입이 유효하지 않음 (*cType* 인자의 값이 유효한 C 데이터 타입이 아님)	SQLBindCol SQLBindParameter
HY009	유효하지 않은 인자 사용 (null pointer)	SQLAllocConnect SQLAllocStmt SQLBindParameter SQLExecDirect SQLForeignKeys SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLSpecialColumns SQLStatistics SQLTablePrivileges
HY010	함수 연속 오류	SQLAllocStmt SQLDescribeParam SQLGetData
HY090	유효하지 않은 문자열 또는 버퍼 길이	SQLBindParameter SQLDescribeCol SQLExecute SQLForeignKeys SQLGetData SQLGetStmtAttr SQLTablePrivileges
HY092	유효하지 않은 속성 또는 옵션	SQLGetStmtAttr
HY105	유효하지 않은 매개변수 타입	SQLBindParameter
HYC00	지원하지 않는 속성 사용	SQLGetConnectAttr SQLGetStmtAttr

데이터베이스 연결 관련 오류

SQLSTATE	Code	Error	Can be returned from
HY000	0x51001	Character set가 존재하지 않음	SQLConnect SQLDriverConnect
	0x5003b	The communication buffer is insufficient. (통신 버퍼의 길이를 초과했음)	SQLExecute
HY001	0x5104A	메모리 할당 오류(드라이버가 함수를 실행하고 완료하기 위해 요구된 메모리를 할당할 수 없음)	SQLConnect SQLDriverConnect
08001	0x50032	드라이버가 DB에 연결을 설정할 수 없음	SQLConnect SQLDriverConnect

네트워크 관련 오류

SQLSTATE	Code	Error	Can be returned from
08002	0x51035	해당 *dbc*는 이미 DB에 연결 되 있음	SQLConnect SQLDriverConnect
08003	0x51036	*stmt*가 연결 되지 않은 상태거나 연결이 끊어진 상태	SQLExecDirect SQLExecute SQLPrepare

SQLSTATE	Code	Error	Can be returned from
08S01	0x51043	통신 회선 장애 (ODBC와 DB간에 함수 처리가 완료되기 전에 통신 회선 실패)	SQLColumns SQLConnect SQLDriverConnect SQLExecDirect SQLExecute SQLFetch SQLForeignKeys SQLGetConnectAttr SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLSetConnectAttr SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables

명령문 상태 전이

다음 테이블들은 명령문 상태에서 해당 핸들 타입(environment, connection, 또는 statement)을 사용하고 있는 Altibase CLI 함수들이 호출됐을 때 각 상태들의 전환이 어떻게 되는지를 보여준다.

Altibase CLI 명령문들은 다음과 같은 상태를 갖는다.

State	Description
S0	Unallocated statement. (The connection state must be connected connection.)
S1	Allocated statement.
S2	Prepared statement. (A (possibly empty) result set will be created.)
S6	Cursor positioned with SQLFetch.

전환 테이블내에서의 각 entry 값은 다음 중 하나이다.

- -- : 함수를 실행한 후 상태의 변환이 없는 경우
- Sn : 명령문 상태가 특정 상태로 전환된 경우
- (IH) : Invalid Handle
- (HY010) : Function sequence error
- (24000) :

> Note)
>
> - S : Success. 이 경우 함수는 다음 값들 중 하나를 반환한다: SQL_SUCCESS_WITH_INFO 또는 SQL_SUCCESS.
> - E : Error. 이 경우 함수는 SQL_ERROR를 반환한다:
> - R : Results. 명령문을 수행했을 때 결과 집합이 있음. (결과 집합은 empty set 일 가능성이 있다.)
> - NR : No Results. 명령문을 수행했을 때 결과 집합이 없음.
> - NF : No Data Found. 함수는 SQL_NO_DATA를 반환한다.
> - RD : Receive Done
> - P : Prepared. The statement was prepared.
> - NP : Not Prepared. The statement was not prepared.

예를 들어 다음은 SQLPrepare 함수에 대한 statement state transition table을 보는 방법이다.

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	S => S1	S => S2	(24000)
		E => S1	

핸들 타입이 SQL_HANDLE_STMT 이며 명령문 상태가 S0 일 때 SQLPrepare 함수가 불려진 경우, -SQL_INVALID_HANDLE (IH)이 반환된다. 핸들 타입이 SQL_HANDLE_STMT 이고 상태가 S1일 때는 불려진 함수 SQLPrepare가 성공적으로 수행되면 S1 상태를 그대로 유지한다. 핸들 타입이 SQL_HANDLE_STMT 이고 S2인 상태에서 SQLPrepare 함수가 불렸을 때 그 함수가 성공적으로 수행되면 명령문의 상태는 S2로 전환되고 수행에 실패를 했을 경우 명령문의 상태는 S1 상태 그대로 남아있다. 핸들 타입이 SQL_HANDLE_STMT인 S6 상태에서 함수가 불리우면 항상 SQL_ERROR와 SQLSTATE 24000 (Invalid Cursor State)이 반환된다.

SQLAllocHandle

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch

--	--	--	--
S1*			

> * *HandleType*이 SQL_HANDLE_STMT인 경우

SQLBindCol

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(IH)	--	--	--

SQLBindParameter

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(IH)	--	--	--

SQLColumns, SQLGetTypeInfo, SQLPrimaryKeys, SQLProcedureColumns, SQLProcedures, SQLStatistics, SQLTables

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	S => S6	E => S1	(24000)
		S => S6	

SQLConnect

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(Error)	(Error)	(Error)	(Error)

SQLDisconnect

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
--	* => S0	* => S0	* => S0

> * 함수 SQLDisconnect를 부르면 연결 핸들과 연관된 모든 명령문들을 종료한다.
> 그리고, 이 함수는 연결 상태를 allocated connection 상태로 놓는다; 명령문 상태가
> S0가 되기 전에 연결 상태는 C4 (connected connection)이다.

SQLDriverConnect

SQLConnect 참고

SQLExecDirect

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	S, NR => --	S, NR => S1	(24000)
	S, R => S6	S, R => S6	
		E => S1	

SQLExecute

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	(HY010)	S, NR => --	(24000)
		S, R => S6	
		E => --	

SQLFetch

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	(HY010)	(HY010)	S => -- RD || NF || E => (if NP => S1, if P => S2)

SQLFreeHandle

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
--	(HY010)	(HY010)	(HY010)
(IH)	S0	S0	S0

(1) 첫 번째 행은 *handleType*이 SQL_HANDLE_ENV 또는 SQL_HANDLE_DBC인 경우

(2) 두 번째 행은 *handleType*이 SQL_HANDLE_STMT인 경우

SQLFreeStmt

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	--	--	NP = > S1 P => S2
(IH)	S0	S0	S0

(1) 첫 번째 행은 *fOption*이 SQL_CLOSE인 경우
(2) 두 번째 행은 *fOption*이 SQL_DROP인 경우

SQLGetData

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(IH)	(HY010)	(HY010)	S || NF => --

SQLGetTypeInfo

SQLColumns 참고

SQLNumResultCols

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(IH)	(HY010)	S => --	S => --

SQLPrepare

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	S => --	S => -- E => S1	(24000)

SQLPrimaryKeys

SQLColumns 참고

SQLProcedureColumns

SQLColumns 참고

SQLProcedures

SQLColumns 참고

SQLSetConnectAttr

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
--*	--	--	(24000)

> * 설정한 속성 (*Attribute*)이 연결 속성인 경우. 설정한 속성이 명령문 속성인 경우는 SQLSetStmtAttr 참고.

SQLSetEnvAttr

S0 Unallocated	S1 Allocated	S2 Prepared	S6 Infetch
(Error)	(Error)	(Error)	(Error)

SQLSetStmtAttr

S0 Unallocated	S1Allocated	S2 Prepared	S6 Infetch
(IH)	--	(1) => --	(1) => --
		(2) => (Error)	(2) => (24000)

(1) *Attribute* 인자가 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, SQL_ATTR_USE_BOOKMARKS, SQL_ATTR_CURSOR_SCROLLABLE, 또는 SQL_ATTR_CURSOR_SENSITIVITY이 아닌 경우

(2) *Attribute* 인자가 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, SQL_ATTR_USE_BOOKMARKS, SQL_ATTR_CURSOR_SCROLLABLE, 또는 SQL_ATTR_CURSOR_SENSITIVITY인 경우

SQLStatistics

SQLColumns 참고

SQLTables

SQLColumns 참고

상태 전이 테이블

다음은 state transition에 영향을 미치는 주요 함수들을 요약한 것이다.

Current State / Request	S0 UNALLOCATED	S1 ALLOCATED	S2 PREPARED	S6 INFETCH
Prepare	(IH)	S => S1	S => S2	(24000)
			E => S1	
ExecDirect	(IH)	S,NR => S1	S => S2	(24000)
		S,R => S6	S,R => S6	
			E => S1	
Execute	(IH)	(HY010)	S,NR => S2	(24000)
			S,R => S6	
			E => S2	
Fetch	(IH)	(HY010)	(HY010)	S => S6
				RD || NF || E => (if NP => S1, if P => S2)
FreeStmt (CLOSE)	(IH)	S1	S2	NP => S1

				P => S2
FreeStmt (DROP)	(IH)	S0	S0	S0

Cf)

- (IH) : Invalid Handle (HY010) : Function Sequence Error

- (24000) : Invalid Cursor State

- S : Success E : Error except Network Error

- R : Results NR : No Results NF : No data Found RD: Receive Done

- P : Prepared NP : Not Prepared

D.부록: 업그레이드

이 부록은 Altibase 4를 위한 ODBC 또는 Altibase CLI 응용 프로그램들을 Altibase 5에서도 사용할 수 있도록 필요한 사항들을 정리했다.

Altibase 5로 업그레이드 되면서 지원하는 CLI 인터페이스의 수준이 확장됐다. 특히 X/Open CLI 표준과 ODBC 스펙을 최대한 준수하여 사용자의 응용 프로그램뿐 아니라 범용 애플리케이션과의 호환성을 높였다.

이 부록에서는 Altibase 5로 업그레이드되면서 추가되거나 재정의된 데이터 타입과 기타 변경 사항들을 설명한다.

- 데이터 타입
- 기타 변경사항

데이터 타입

이 절에서는 Altibase 5에서 새롭게 추가된 데이터 타입들을 설명한다.

이전 버전보다 표준을 더욱 준수하여 기존의 애플리케이션을 컴파일할 때 발생할 수 있는 문제들을 해결하도록 한다.

SQLCHAR, SQLSCHAR

이전의 CLI 애플리케이션들은 SQLCHAR와 char를 혼용해서 사용했다. 그러나 표준 지향의 SQLCHAR는 아래와 같이 새롭게 정의된다.

```
typedef unsigned char SQLCHAR;
typedef signed char SQLSCHAR;
```

따라서 기존 응용 프로그램에서 아래와 같은 구문으로 컴파일할 때 오류가 발생한다.

```
char *query = “....”;
SQLPrepare(stmt, query, SQL_NTS);
```

이런 문제를 해결하기 위해 아래와 같이 타입 캐스팅을 수정하면 된다.

```
char *query = “....”;
SQLPrepare(stmt, (SQLCHAR *)query, SQL_NTS);
```

SQL_BIT, SQL_VARBIT

Altibase는 버전 5부터 SQL 92 표준의 BIT 타입뿐 아니라 사용자 편의를 위한 VARBIT 타입을 지원한다. BIT와 VARBIT에 대한 자세한 내용은 *SQL Reference*을 참조하기 바란다.

BIT to C type

다음은 BIT 관련 변환 테이블을 나타낸다.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_CHAR	BufferLength > 1	Data (‘0’ 혹은 ‘1’)	1	n/a
	BufferLength <= 1	Undefined	Undefined	22003
SQL_C_STINYINT SQL_C_UTINYINT SQL_C_SBIGINT SQL_C_UBIGINT SQL_C_SSHORT SQL_C_USHORT SQL_C_SLONG SQL_C_ULONG SQL_C_FLOAT SQL_C_DOUBLE SQL_C_NUMERIC	None (BufferLength 의 값은 이와 같은 고정크기 타입으로의 컨버젼시에는 무시된다)	Data (0 혹은 1)	C type 의 size	n/a
SQL_C_BIT	None	Data (0 혹은 1)	1	n/a
SQL_C_BINARY	None	Data (포맷은 아래 참조)	사용자가 바인드한 메모리에 write 할 데이터의 길이	n/a

VARBIT to C type

다음은 VARBIT 관련 변환 테이블을 나타낸다.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_CHAR	BufferLength > 1	Data	varbit 의 precision	n/a
	BufferLength <= 1	Undefined	Undefined	22003
SQL_C_BIT	None	Data (0 혹은 1)	1	n/a
SQL_C_BINARY		Data (포맷은 BIT와 동일)	사용자가 바인드한 메모리에 write 할 데이터의 길이	n/a

C type to BIT/VARBIT

현재 BIT/VARBIT로 변환될 수 있는 타입이 존재하지 않는다.

바이너리 형식

0 7	8 15	16 23	24 31	32 39	...	8n 8n+7
Precision				Data		

where n= (Precision+7)/8 + 3

Precision : Length of BIT data

Data : BIT Data

데이터 타입 변환 예제

BIT/VARBIT SQL_C_BINARY

BIT를 SQL_C_BINARY로 바인드하여 페치할 때에는 서버에서 전달되는 데이터를 사용자에게 그대로 전달해준다. 사용자 버퍼에 저장되는 데이터의 형식은 앞에서 설명한 것과 같다.

다음의 예제처럼 구조체 struct bit_t를 정의해서 사용하면 보다 편리하게 서버에 접근할 수 있다.

```
CREATE TABLE T1(I1 BIT(17), I2 VARBIT(37));
INSERT INTO T1 VALUES(BIT'11111011010011011',
VARBIT'0010010010101110001010100010010011011');
INSERT INTO T1 VALUES(BIT'110011011',
VARBIT'0011100010101000100100110111');

----------------------

void dump(unsigned char *Buffer, SQLINTEGER Length)
{
for (SQLINTEGER i = 0; i < Length; i++)
{
printf("%02X ", *(Buffer + i));
}
}

typedef struct bit_t
{
SQLUINTEGER mPrecision;
unsigned char mData[1];
} bit_t;

bit_t *Bit;
bit_t *Varbit;
SQLLEN Length;
SQLRETURN rc;

Bit = (bit_t *)malloc(BUFFER_SIZE);
Varbit = (bit_t *)malloc(BUFFER_SIZE);

SQLBindCol( stmt, 1,
SQL_C_BINARY,
(SQLPOINTER)Bit,
BUFFER_SIZE,
&LengthBit);

SQLBindCol( stmt, 2,
SQL_C_BINARY,
(SQLPOINTER)Varbit,
BUFFER_SIZE,
&LengthVarbit);
do
{
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);

printf("-----\n");

printf(">> Bit\n");
printf("Length : %d\n", LengthBit);
printf("Precision : %d\n", Bit->mPrecision);
dump(Bit->mData, LengthBit – sizeof(SQLUINTEGER));

printf(">> Varbit\n");
printf("Length : %d\n", LengthVarbit);
printf("Precision : %d\n", Varbit->mPrecision);
dump(Varbit->mData, LengthVarbit – sizeof(SQLUINTEGER));
} while (rc != SQL_NO_DATA);
```

위의 프로그램을 실행시켰을 때 결과는 다음과 같다.

```
------
>> Bit
Length : 7
Precision : 17
FB 4D 80 (1111 1011 0100 1101 1)
>> Varbit
Length : 9
Precision : 37
24 AE 2A 24 D8 (0010 0100 1010 1110 0010 1010 0010 0100 1101 1)
------
>> Bit
Length : 7
Precision : 17 -> BIT 는 0으로 패딩하므로 17.
CD 80 00 (1100 1101 1000 0000)
>> Varbit
Length : 8
```

```
Precision : 27 -> VARBIT 는 패딩하지 않으므로 27.
38 A8 93 60 (0011 1000 1010 1000 1001 0011 011)
```

만약, BUFFER_SIZE가 필요한 메모리 크기보다 작을 경우 SQLFetch()는 SQL_SUCCESS_WITH_INFO를 리턴하고, BUFFER_SIZE 만큼만 바인드 된 메모리에 쓴다.

BIT/VARBIT to SQL_C_BIT

ODBC의 SQL_C_BIT는 0 또는 1의 값을 갖는 8비트의 부호 없는 정수이기 때문에 주의가 필요하다. 즉, SQL_C_BIT로 바인드하여 페치할 경우 서버의 테이블에 BIT '011001'이 저장되었어도 바인드 된 변수에는 0x64가 아니라 0x01이 들어간다

BIT to SQL_C_CHAR

BIT 칼럼을 페치할 때 SQL_C_CHAR 로 바인드했다면, 결과는 ODBC 타입 변환 규칙에 따라 항상 0 또는 1을 갖는다.

```
CREATE TABLE T1 (I1 BIT(12));
INSERT INTO T1 VALUES(BIT'110011000010');
INSERT INTO T1 VALUES(BIT'010011000010');

SQLCHAR sData[128];
SQLLEN sLength;

sQuery = (SQLCHAR *)"SELECT I1 FROM T1";

SQLBindCol(stmt, 1, SQL_C_CHAR, sData, sizeof(sData), sLength);

SQLExecDirect(stmt, sQuery, SQL_NTS);

while (SQLFetch(stmt) != SQL_NO_DATA)
{
printf("bit value = %s, ", sData);
printf("sLength = %d\n", sLength);
}
위의 프로그램을 실행시키면, 다음과 같이 출력된다.
1, sLength = 1
0, sLength = 1
```

VARBIT to SQL_C_CHAR

ODBC 표준에 VARBIT 타입이 없어 변환 툴을 자체적으로 만들었기 때문에, VARBIT 칼럼을 페치할 때에는 해당 칼럼의 데이터가 모두 페치된다.

```
CREATE TABLE T1 (I1 VARBIT(12));
INSERT INTO T1 VALUES(VARBIT'110011000010');
INSERT INTO T1 VALUES(VARBIT'01011010');

SQLCHAR sData[128];
SQLLEN sLength;
sQuery = (SQLCHAR *)"SELECT I1 FROM T1";
SQLBindCol(stmt, 1, SQL_C_CHAR, sData, sizeof(sData), &sLength);
SQLExecDirect(stmt, sQuery, SQL_NTS);
while (SQLFetch(stmt) != SQL_NO_DATA)
{
printf("bit value = %s, ", sData);
printf("sLength = %d\n", sLength);
}
위의 프로그램을 실행시키면, 다음과 같이 출력된다.
110011000010, sLength = 12
01011010, sLength = 8
```

SQL_NIBBLE

Altibase 4에서 지원하던 SQL_C_NIBBLE 타입은 Altibase 5에서 사용하지 않는다. 그러나 SQL_C_BINARY를 이용하여 원하는 데이터를 페치할 수 있다.

NIBBLE to C type

SQL_C_CHAR 및 SQL_C_BINARY로의 변환만 가능하다.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_CHAR	BufferLength > 1	Data (‘0’ 또는 ‘1’)	사용자가 바인드 한 메모리에 write 할 데이터의 길이 (null 종료문자 제외)	n/a
	BufferLength <= 1	Undefined	Undefined	22003
SQL_C_BINARY	None	Data (포맷은 아래 참조)	사용자가 바인드 한 메모리에 write 할 데이터의 길이	n/a

방법은 BIT를 BINARY로 페치할 때와 동일하지만, NIBBLE의 BINARY 형식은 BIT와 달리 precision 필드가 1바이트 정수라는 것을 주의해야 한다.

바이너리 형식

0 7	8 15	...	8n 8n+7
Precision		Data	

Where n = (Precision+1)/2
Precision : Length of NIBBLE data

Data : Nibble Data

데이터 타입 변환 예제

NIBBLE to SQL_C_BINARY

NIBBLE을 SQL_C_BINARY로 바인드하여 페치할 때에는 서버에서 전달되는 데이터를 사용자에게 그대로 전달해준다. 사용자 버퍼에 저장되는 데이터의 형식은 앞에서 설명한 것과 같다.

다음의 예제에서 사용하는 것처럼 nibble_t 구조체를 정의해서 사용하면 보다 편리하게 접근할 수 있다.

```
CREATE TABLE T1(I1 NIBBLE, I2 NIBBLE(10), I3 NIBBLE(21) NOT NULL);
INSERT INTO T1 VALUES(NIBBLE'A', NIBBLE'0123456789', NIBBLE'0123456789ABCDEF00121');
INSERT INTO T1 VALUES(NIBBLE'B', NIBBLE'789', NIBBLE'ABCD1234');

-------------------

void dump(unsigned char *Buffer, int Length)
{
for (int i = 0; i < Length; i++) printf("%02X ", *(Buffer + i));
}

typedef struct nibble_t
{
unsigned char mPrecision;
unsigned char mData[1];
} nibble_t;

nibble_t *Buffer;
SQLLEN Length;
SQLRETURN rc;

Buffer = (nibble_t *)malloc(BUFFER_SIZE);

SQLBindCol(stmt, 2, SQL_C_BINARY, (SQLPOINTER)Buffer, BUFFER_SIZE, &Length);
do
{
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);

printf("----\n");
printf("Length : %d\n", Length);
printf("Precision : %d\n", Buffer->mPrecision);
dump(Buffer->mData, Length – sizeof(SQLUINTEGER));
} while (rc != SQL_NO_DATA);
위의 프로그램을 실행시키면, 다음과 같은 결과가 나온다.
Length : 6
Precision : 10
01 23 45 67 89
----
Length : 3
Precision : 3
78 90
```

NIBBLE to SQL_C_CHAR

이 경우 특이사항이 없으므로, 예제와 결과는 생략한다.

SQL_BYTE

Altibase 4에 있던 SQL_C_BYTE 대신 SQL_C_BINARY로 바인드하여 동일한 방식으로 동작한다.

BYTE to C types

SQL_C_CHAR, SQL_C_BINARY 외의 다른 타입으로의 변환은 불가능하다. 그러나 바이너리 데이터를 SQL_C_CHAR로 변환할 때 원천 데이터의 1 바이트가 ASCII 2 글자로 표현되는 것을 주의해야 한다.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_CHAR	(Byte length of data) * 2 < BufferLength	Data	Length of data in bytes	n/a
	(Byte length of data) * 2 >= BufferLength	Truncated data	Length of data in bytes	01004
SQL_C_BINARY	Byte length of data <= BufferLength	Data	Length of data in bytes	n/a
	Byte length of data > BufferLength	Truncated data	Length of data in bytes	01004

Altibase CLI는 바이너리 데이터를 SQL_C_CHAR로 변환할 때 각각의 바이트가 항상 16진수의 쌍(pair)으로 변환되고, 널 터미네이트(NULL terminate) 시킨다. 따라서 바인드 한 SQL_C_CHAR 버퍼의 크기가 짝수일 때에는 바인드 한 버퍼의 마지막 바이트에 NULL termination 문자가 찍히는 것이 아니라, 바인드 한 버퍼의 마지막에서 한 바이트 앞의 위치에 NULL termination 문자가 찍힌다.

Source binary data	Size of bound buffer	Contents of the buffer bound as	*StrLen_or_IndPtr	SQLSTATE

(16진수)	(bytes)	SQL_C_CHAR		
AA BB 11 12	8	hex : 41 41 42 42 31 31 00 ??[1] string : “AABB11”	8	01004
	9	hex : 41 41 42 42 31 31 31 32 00 string : “AABB1112”	8	n/a

[1] ??로 표기한 부분은 정의하지 않는다.

바이너리 형식

바이트 타입은 NIBBLE이나 BIT처럼 형식이 따로 존재하는 것이 아니라, 이진 데이터의 칼럼이다.

데이터 타입 변환 예제

BYTE to SQL_C_CHAR

```
CREATE TABLE T1(I1 BYTE(30));
INSERT INTO T1 VALUES(BYTE’56789ABC’);

-------------------
SQLLEN Length;
SQLRETURN rc;

// 실행 쿼리 : SELECT * FROM T1;

Buffer = (nibble_t *)malloc(BUFFER_SIZE);

SQLBindCol(stmt, 1, SQL_C_CHAR, (SQLPOINTER)Buffer, BUFFER_SIZE, &Length);
do
{
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);
printf(“Length : %d\n”, Length);
printf(“Data : %s\n”, Length);
} while (rc != SQL_NO_DATA);
위의 프로그램을 실행시키면, 다음과 같은 결과가 나온다.
실행 결과 1 : BUFFER_SIZE >= 9
Length : 8
Data : 56789ABC

실행 결과 2 : BUFFER_SIZE == 8
Length : 8
Data : 56789A -> 8 바이트 버퍼에 바인딩 했는데 6글자만 나왔음.

실행 결과 3 : BUFFER_SIZE == 7
Length : 8
Data : 56789A -> BUFFER_SIZE == 8 일 때와 결과가 동일함.

실행 결과 4 : BUFFER_SIZE == 6
Length : 8
Data : 5678

실행 결과들 중에서 1번을 제외한 나머지에 대해서는 SQLFetch()가 SQL_SUCCESS_WITH_INFO를 반환한다. 해당 SQLSTATE는 01004이다.
```

BYTE to SQL_C_BINARY

바이너리로 바인딩 하는 것은 특이사항이 없다.

DATE : SQL_TYPE_TIMESTAMP

Altibase 5에서는 DATE 칼럼에 SQLDescribeCol() 또는SQLColAttribute()를 이용하여 SQL type으로 가져올 경우, SQL_TYPE_TIMESTAMP를 반환한다. SQL_TYPE_TIMESTAMP는 ODBC 표준 타입 중에서 Altibase의 DATE 타입과 유사한 타입으로 년, 월, 일, 시, 분, 초로 구성됐다.

그러나 Altibase 4에서는 DATE 타입이 날짜, 시간 등의 기본요소와 이들을 구분하는 특수문자 등 다량의 데이터로 구성되어 있어, SQLColAttribute() 또는 SQLDescribeCol()을 호출하면 SQL type으로 SQL_DATE가 반환됐다.

따라서 Altibase 5의 Altibase CLI를 사용할 때에는 ODBC 3.0의 타입 상수인 SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP를 사용하기를 권장한다.

LOB

데이터 타입

Altibase 4에서 LOB 타입은 길이가 페이지 크기로 제한되었으나, Altibase 5에서는 최대 4GB-1byte 까지 지원하는 BLOB, CLOB으로 구성된다.

Altibase 4 DDL

```
CREATE TABLE T1 (I1 BLOB(3));
```

Altibase 5 DDL

```
CREATE TABLE T1 (I1 BLOB); ---> 괄호 안의 precision 이 사라짐.
```

LOB 함수 사용

Altibase CLI 애플리케이션에서 LOB을 사용하기 위해 제공되는 전용 함수가 제공된다. LOB을 위한 특수 함수에 대한 자세한 내용은 3장 LOB 인터페이스를 참조한다.

LOB을 위한 함수 외에도 일반적인 바이너리, 문자형 타입에서 사용할 수 있는 함수도 사용이 가능하다. 이러한 특징 때문에 ODBC 애플리케이션에서 표준 ODBC를 사용하여 LOB 데이터의 저장 및 검색 등이 가능하다.

그러나 부분 수정(partial update), 부분 검색(partial retrieve)과 같은 기능은 ODBC 응용 프로그램에서 사용할 수 없다. 단 CLI 응용 프로그램에서는 SQLGetLob, SQLPutLob 등의 함수를 이용하여 사용이 가능하다.

```
CREATE TABLE T1 (I1 BLOB, I2 CLOB);  // CONNECTION의 AUTOCOMMIT 모드를 OFF 시킨다.

SQLCHAR sBlobData[128];
SQLCHAR sClobData[128];
SQLLEN sBlobLength;
SQLLEN sClobLength;

SQLCHAR *sQuery = (SQLCHAR *)”INSERT INTO T1 VALUES(?, ?)”;
SQLPrepare(stmt, sQuery, SQL_NTS);

SQLBindParameter(stmt, 1, SQL_C_BINARY, SQL_BLOB,0, 0, sBlobData, sizeof(sBlobData), sBlobLength);
SQLBindParameter(stmt, 2, SQL_C_CHAR, SQL_CLOB,0, 0, sClobData, sizeof(sClobData), &sClobLength);
sBlobLength = create_blob_data(sBlobData);
sprintf((char *)sClobData, “this is clob data”);
sClobLength = SQL_NTS;

SQLExecute(stmt);
```

ODBC 응용프로그램에서 LOB 사용하기

ODBC 응용 프로그램에서 LOB 칼럼을 페치(fetch)하거나 LOB 칼럼에 데이터를 저장할 때에는 SQLDescribeCol, SQLColAttribute 또는 SQLDescribeParam 함수를 호출한다.

LOB 칼럼에서 이들 함수를 수행하면 SQL_BLOB, SQL_CLOB의 데이터 타입으로 반환된다. 그러나 ODBC 응용 프로그램은 SQL_BLOB이나 SQL_CLOB과 같은 데이터 타입을 인지할 수 없다.

따라서 ODBC 응용 프로그램이 인식할 수 있는 데이터 타입으로 반환해야 한다. 이러한 문제는 odbc.ini에 LongDataCompat = on 으로 옵션을 주어서 해결할 수 있다. 이 옵션을 실행할 경우 LOB 칼럼에 SQLColAttribute() 등의 함수를 호출하면, SQL_BLOB 대신 SQL_LONGVARBINARY를, SQL_CLOB 대신 SQL_LONGVARCHAR를 해당 칼럼의 타입으로 반환한다.

PHP 프로그램에서 LOB 사용 예제

다음은 PHP 응용 프로그램에서 LOB을 사용하는 예제이다.

프로그램을 실행하기 전에 php.ini에서 다음의 두가지 속성을 확인해야 한다.

odbc.defaultlrl = 4096 (반드시 1 이상의 값으로 설정한다)
odbc.defaultbinmode = 0 (LOB을 쓰기 위해서 0으로 설정해야 추가메모리를 할당하지 않고 동작한다)
~/.odbc.ini는 아래와 같다.

```
[Altibase]
Driver = AltibaseODBCDriver
Description = Altibase DSN
ServerType = Altibase
UserName = SYS
Password = MANAGER
Server = 127.0.0.1
Port = 20073
LongDataCompat = on
NLS_USE = US7ASCII
```

php 프로그램

```
<?
/*
* ================================================
* 연결 시도
* ================================================
*/

$Connection = @odbc_connect("Altibase", "SYS", "MANAGER");
if (!$Connection)
{
echo "ConnectFail!!!\n";
exit;
}

/*
* ================================================
* 테이블 생성
* ================================================
*/

@odbc_exec($Connection, "DROP TABLE T2 ");
if (!@odbc_exec($Connection,
"CREATE TABLE T2 (I1 INTEGER, B2 BLOB, C3 CLOB) TABLESPACE SYS_TBS_DATA"))
{
echo "create test table Fail!!!\n";
exit;
}

/*
* ================================================
* LOB 사용을 위한 autocommit off
* ================================================
*/

odbc_autocommit($Connection, FALSE);

/*
* ================================================
* 데이터의 삽입
* ================================================
*/

$query = "INSERT INTO T2 VALUES (?, ?, ?)";
$Result1 = @odbc_prepare($Connection, $query);
if (!$Result1)
{
$msg = odbc_errormsg($Connection);
echo "prepare insert: $msg\n";
exit;
}

for ($i = 0; $i < 10; $i++)
{

/*
* ----------------------
* 파일에서 읽기
* ----------------------
*/

$fileno2 = $i + 1;
$filename2 = "a$fileno2.txt";
```

```
print("filename = $filename2\n");
$fp = fopen($filename2, "r");
$blob = fread($fp, 1000000);
fclose($fp);

$fileno3 = 10 - $i;
$filename3 = "a$fileno3.txt";
print("filename = $filename3\n");
$fp = fopen($filename3, "r");
$clob = fread($fp, 1000000);
fclose($fp);

/*
* ----------------------
* INSERT
* ----------------------
*/

$Result2 = @odbc_execute($Result1, array($i, $blob, $clob));

print("inserting $i ,$filename2 and $filename3 into T2 ......... ");

if (!$Result2)
{
print("FAIL\n");
$msg = odbc_errormsg($Connection);
echo "execute insert: $msg \n";
exit;
}

print("OK\n");
}

/*
* ================================================
* COMMIT
* ================================================
*/

odbc_commit($Connection);

/*
* ================================================
* INSERT된 데이터 확인
* ================================================
*/
print "\n\n";
print "========================================\n";
print "Selecting from table\n";
print "========================================\n";

$query = "select * from t2";
$Result1 = @odbc_exec($Connection, $query);
if (!$Result1)
{
$msg = odbc_errormsg($Connection);
echo "ERROR select: $msg\n";
exit;
}

$rownumber = 0;
while (odbc_fetch_row($Result1))
{
$data1 = odbc_result($Result1, 1);
$data2 = odbc_result($Result1, 2);
$data3 = odbc_result($Result1, 3);
$len2 = strlen($data2);
$len3 = strlen($data3);

print "\n========================================\n";
print "Row $rownumber....\n";
$rownumber++;
print "data1 = ".$data1."\n";
print "-------\n";
print "data2 = \n";
// print $data2; // binary data 이므로 출력 생략
print "\n";
```

```
print "dataLen2 = [$len2]\n";
print "-------\n";
print "data3 = \n";
print $data3;
print "\n";
print "dataLen3 = [$len3]\n";
}

odbc_commit($Connection);

@odbc_close($Connection);
?>
```

기타 변경사항

이 절에서는 데이터 타입 이외의 변경 사항들에 대해 설명한다.

SQLCloseCursor

Altibase 4의 Altibase CLI 라이브러리에는 ODBC state machine이 있지 않기 때문에 아래의 순서로 함수를 호출한다.

```
SQLHSTMT stmt;
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
SQLPrepare(stmt, (SQLCHAR *)"SELECT I1 FROM T1", SQL_NTS);

SQLExecute(stmt);
SQLFetch(stmt);
SQLExecute(stmt);
```

그러나 Altibase 5의 Altibase CLI 드라이버로 위의 코드를 실행하면, SQLExecute(stmt)에서 Function sequence error가 발생한다.

이러한 오류가 발생하는 이유는 처음 SQLExecute()를 수행한 stmt가 result set을 생성시키는 문장이기 때문이다. 따라서 ODBC cursor가 open 되며, stmt의 상태는 S5로 된다 (MSDN ODBC specification 참조). 그러나 S5 상태에서는 SQLExecute()를 수행할 수 없어 오류가 발생한다.

이 상태에서 다시 SQLExecute()를 수행하려면, 아래와 같이 명시적으로 SQLCloseCursor()를 호출하여 stmt를 S1 이나 S3 상태로 만들어야 한다.

```
SQLExecute(stmt);
SQLFetch(stmt);
SQLCloseCursor(stmt);

SQLExecute(stmt);
```

SQLBindParameter – ColumnSize 인자

Altibase 5에서는 SQLBindParameter의 인자들 중 6 번째 인자인 ColumnSize의 사용이 이전 버전과 달라졌다.

이전 버전에서는 이 인자에 0을 입력해도 문제가 없었다. 그러나 Altibase 5에서는 서버로 전송하는 데이터의 최대 길이를 입력하지 않으면 수행할 때 마다 해당 정보를 확인하므로 성능에 문제가 발생한다.

SQLBindParameter – StrLen_or_IndPtr 인자

Altibase 4의 CLI 라이브러리에서는 StrLen_or_IndPtr 인자가 가리키는 데이터가 가변길이일 때에만 참조를 했다.

그러나 Altibase 5에서 SQLPutData()와 SQLParamData() 함수를 구현하게 되면서, StrLen_or_IndPtr 인자가 가리키는 곳의 메모리에 있는 값은 SQLExecute() 또는 SQLExecDirect()를 수행할 때마다 참조한다.

따라서 SQLBindParameter()의 마지막 인자인 StrLen_or_Ind를 널(NULL) 포인터가 아닌 유효한 포인터 변수로 전달하였다면, 포인터가 가리키는 곳의 메모리가 제대로

초기화되었는지 주의해야 한다.

만약 해당 메모리가 제대로 초기화되지 않고, SQL_DATA_AT_EXEC 상수 (-2) 이거나 SQL_LEN_DATA_AT_EXEC() 매크로의 결과 값인 -100 이하의 값을 가졌을 때, CLI 라이브러리는 사용자가 해당 인자를 SQLPutData()를 이용해서 전달하겠다는 의도로 해석한다. 그리고 SQLExecute() 함수를 호출했을 때 SQL_NEED_DATA를 반환한다.

위와 같이 초기화되지 않은 값 때문에 SQLExecDirect() 등의 함수가 의도하지 않은 값(SQL_NEED_DATA)을 반환한다면, 그 다음 호출하는 함수들에도 영향을 준다. 그리고 다음부터 호출하는 함수에서 모두 Function Sequence Error가 발생하여, SQL_ERROR가 반환되는 것을 주의해야 한다.

SQLPutData(), SQLParamData()

Altibase 5의 Altibase CLI는 이전 버전에서 지원하지 않았던 SQLPutData()와 SQLParamData()를 지원한다. 각각의 함수의 자세한 사용법은 MSDN을 참조하기 바란다.

다음 예제는 상기 함수 및 앞서 설명한 StrLen_or_IndPtr 인자를 사용한 예제 프로그램이다.

```
Table Schema :
CREATE TABLE T2_CHAR (I1 INTEGER, I2 CHAR(50));

void putdata_test(void)
{
SQLRETURN sRetCode;
SQLHANDLE sStmt;

SQLINTEGER i1;
SQLLEN i1ind;

SQLCHAR *i2[10] =
{
(unsigned char *)"0000000000000.",
(unsigned char *)"1111111111111. test has been done.",
(unsigned char *)"2222222222222. Abra ca dabra",
(unsigned char *)"3333333333333. Short accounts make long friends.",
(unsigned char *)"4444444444444. Whar the hell are you doing man!",
(unsigned char *)"5555555555555. Oops! I missed this row. What an idiot!",
(unsigned char *)"6666666666666. SQLPutData test is well under way.",
(unsigned char *)"7777777777777. The length of this line is well over 50 characters.",
(unsigned char *)"8888888888888. Hehehe",
(unsigned char *)"9999999999999. Can you see this?",
};

SQLLEN i2ind;

SQLINTEGER i;

SQLINTEGER sMarker = 0;

i1ind = SQL_DATA_AT_EXEC;
i2ind = SQL_DATA_AT_EXEC;

sRetCode = SQLAllocHandle(SQL_HANDLE_STMT, gHdbc, &sStmt);
check_error(SQL_HANDLE_DBC, gHdbc, "STMT handle allocation", sRetCode);

sRetCode = SQLBindParameter(sStmt, 1, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER,
0, 0, (SQLPOINTER *)1, 0, &i1ind);

sRetCode = SQLBindParameter(sStmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CHAR,
60, 0, (SQLPOINTER *)2, 0, &i2ind);

sRetCode = SQLPrepare(sStmt,
(SQLCHAR *)"insert into t2_char values (?, ?)", SQL_NTS);

for(i = 0; i < 10; i++)
{
i1 = i + 1000;

printf("\n");
printf(">>>>>>>> row %d : inserting %d, \"%s\"\n", i, i1, i2[i]);
sRetCode = SQLExecute(sStmt);

if(sRetCode == SQL_NEED_DATA)
{
sRetCode = SQLParamData(sStmt, (void **)&sMarker);

while(sRetCode == SQL_NEED_DATA)
{
printf("SQLParamData gave : %d\n", sMarker);

if(sMarker == 1)
{
sRetCode = SQLPutData(sStmt, &i1, 0);
}
else if(sMarker == 2)
{
int unitsize = 20;
int size;
int pos;
int len;

len = strlen((char *)(i2[i]));
```

```
for(pos = 0; pos < len;)
{
size = len - pos;
if(unitsize < size)
{
size = unitsize;
}
sRetCode = SQLPutData(sStmt, i2[i] + pos, size);

pos += size;
}
}
else
{
printf("bbbbbbbbbbbbbbbbbbbbbbbbbbbbb!!! unknown marker value\n");
exit(1);
}

sRetCode = SQLParamData(sStmt, (void **)&sMarker);
}
}
}

sRetCode = SQLFreeHandle(SQL_HANDLE_STMT, sStmt);
check_error(SQL_HANDLE_DBC, gHdbc, "STMT free", sRetCode);
}
```

ALTER SESSION 구문의 사용 제한

Altibase 4에서는 세션 속성(property)인 AUTOCOMMIT MODE, DEFAULT_DATE_FORMAT를 다음과 같이 지정했다.

```
SQLExecDirect(stmt,
“ALTER SESSION SET AUTOCOMMIT=FALSE”,
SQL_NTS);

SQLExecDirect(stmt,
“ALTER SESSION SET DEFAULT_DATE_FORMAT='YYYY/MM/DD'",
SQL_NTS);
```

위의 두 세션 속성은 Altibase CLI의 변환(conversion) 및 트랜잭션 관련 함수들의 동작에 결정적인 영향을 주기 때문에 반드시 Altibase CLI 드라이버가 이에 대한 정보를 가지고 있어야 한다.

그러나 SQLExecDirect() 함수를 이용하여 SQL 구문을 서버에 직접 전송할 경우 Altibase CLI 드라이버가 해당 속성의 변경사항을 알 수 없다. 물론 해당 속성의 값을 알기 위해 Altibase CLI 드라이버가 서버에서 정보를 얻어올 수도 있지만 이는 성능을 저하시킬 수 있다.

이러한 문제 해결을 위해 Altibase 5에서는 SQLSetConnectAttr()으로 해당 속성을 수정하여, 별도의 절차 없이 Altibase CLI 드라이버가 유지하는 속성과 서버에 세팅된 속성이 항상 일치할 수 있도록 했다. Altibase CLI를 이용할 때 Altibase 5에서는 다음과 같이 프로그래밍하라.

```
SQLSetConnectAttr(conn,
SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_OFF,
0);
SQLSetConnectAttr(conn,
ALTIBASE_DATE_FORMAT,
(SQLPOINTER)"YYYY/MM/DD",
SQL_NTS);
```

SQLRowCount(), SQLMoreResults() functions

Altibase CLI의 결과 값(Result)은 다음 두 가지로 나뉜다.

- Number of affected rows
- Result set

Altibase CLI는 Altibase 5에서 다중 결과(multiple results)를 고려하고 있다. 즉, 한 번의 실행으로 복수의 결과들을 얻을 수 있다는 의미이다. 따라서 배열(Array) 바인딩을 사용할 때, SQLRowCount() 함수의 반환 값은 Altibase 4의 값과는 차이가 있다.

- SQLRowCount() : “current” result 에서 affected row count를 얻는 함수
- SQLMoreResults() : “next” result 로 이동하는 함수. current result 가 마지막 result 이면 SQL_NO_DATA를 리턴한다.

예제

```
CREATE TABLE T1 (I1 INTEGER);
INSERT INTO T1 VALUES(1);
INSERT INTO T1 VALUES(2); ........ 1000 번 반복.

SELECT * FROM T1;
T1
-----
1
2
3
.
.
.
1000
-----

SQLINTEGER p1[3];
SQLINTEGER p2[3];
SQLLEN rowcount = 0L;
SQLLEN totalRowcount = 0L;

p1[0] = 10; p2[0] = 20;
p1[1] = 100; p2[1] = 200;
p1[2] = 11; p2[2] = 14;

SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, 3); // <--- array binding

SQLBindParameter(stmt, 1, p1 ..);
SQLBindParameter(stmt, 2, p2 ..);

SQLExecDirect(stmt,
(SQLCHAR *)“DELETE FROM T1 WHERE I1>? AND I1<?”,
SQL_NTS);

do {
SQLRowCount(stmt, &rowcount);
printf("%d\n", rowcount);
totalRowcount += rowcount;
rc = SQLMoreResults(stmt);
} while (rc != SQL_NO_DATA);

printf(“totalRowcount = %d\n”, totalRowCount);
실행결과
9   -> DELETE FROM T1 WHERE I1>10 AND I1<20 에 대한 affected row count
199 -> DELETE FROM T1 WHERE I1>100 AND I1<200 에 대한 affected row count
0   -> DELETE FROM T1 WHERE I1>11 AND I1<14 에 대한 affected row count
(첫번째 execution에서 지워졌으므로 레코드 존재하지 않음)
208 -> affected row count 의 합계.
```

각각의 인자를 받은 구문들의 실행 결과는 각각 생성되어 Altibase CLI 드라이버로 전달된다. 이렇게 여러 개의 결과가 생성될 경우 각각의 결과들은 SQLMoreResults() 함수를 통해 다음 결과로 이동하고, SQLRowCount()로 결과를 가져온다.

Altibase 4에서는 위의 세 개의 결과를 모두 합치면 SQLRowCount()가 208 이라는 결과를 반환한다.

만약 Altibase 5에서 Altibase 4와 동일한 결과를 얻기 위해서는 SQLMoreResults() 함수를 SQL_NO_DATA를 반환할 때까지 반복하고, SQLRowCount()로 가져온 결과를 더해줘야 한다.

크기 제한 없는 Array Execute, Array Fetch

Altibase 5에서는 통신 버퍼 크기 만큼 Array Execute, Array Fetch를 할 수 있는 제약이 사라졌다.

따라서 메모리가 허용하는 만큼 Array binding을 할 수 있게 되었으며, CM_BUFF_SIZE의 속성도 더 이상 지원하지 않는다.

사라진 속성들

배치 프로세싱 모드

연결 문자열(connection string)의 배치 키워드가 더 이상 지원되지 않는다.

또한 연결 속성인 SQL_ATTR_BATCH도 더 이상 지원되지 않는다.

SQL_ATTR_MAX_ROWS

Altibase 4에서는 성능 향상과 관련하여 prefetch 하는 행의 숫자를 지정하는 의미로 사용했다. 그러나 ODBC 문서에 따르면 이 속성은 SELECT 구문의 LIMIT 절과 비슷한 의미를 가진다. 이와 같은 기능은 Altibase CLI에서는 지원하지 않는다.

따라서 위 속성을 SQLSetStmtAttr() 함수로 세팅할 경우, ‘Optional feature not implemented’ 오류가 발생하므로 주의해야 한다.