The Oracle SQL Class Library

Version 1.0

Short Introduction

Sergei Kuchin

Mosakin Corporation
1070-R Highway 34
Matawan, NJ 07747-3459

Email: kuchin@empros.com
kuchin@mosakin.com

1995

TABLE OF CONTENTS

1. Introduction 4

2. SQL stream classes 4

3. SQL pipe 5

4. Macros for hard-coding SQL statements 5

5. Exception handling 7

6. Low level interface classes 8

1. Introduction

The Oracle SQL class library provides a powerful and convenient programming interface with Oracle SQL and PL/SQL. The library comprises of a collection of C++ classes and macros:

- SQL stream classes;
- SQL pipe and SQL pipe requestors for C and C++
- Macros for hard-coding SQL and PL/SQL statements into database application
- Exception handling
- Low level interface classes

The library was tested with the following compilers:

- Visual C++ 1.5 and higher, MS-Windows
- Borland C++ 4.0 and higher, MS-Windows
- GNU C++ 2.4.5 and higher, SCO Unix, IBM AIX
- IBM C++ (xlC) 1.2 and higher, IBM AIX

The author welcomes feedback from database developers who will be interested in the library.

2. SQL stream classes

Any SQL or PL/SQL statement, which has input/output bind variables, can be opened as a buffered stream. The stream classes have << and >> operators for reading and writing objects of different data types.

Example

```
SQLStream s("select state_name, state_code           \
             from state                              \
             where state_name like :v1<char[33]>",
            50 // fetch 50 rows per one fetch operation
           );
char name[33];
int  code;

s<<"M%";
while(!s.eof()){
   s>>name>>code;
   cout<<"State="<<name<<", Code="<<code<<endl;
}
```

3. SQL pipe

The SQL pipe is a stand alone program which encapsulates all Oracle runtime libraries in one Unix application. There are SQL pipe requestors for both C and C++. Client programs must be linked with the requestors. The requestors have programming interface with the SQL pipe.

The requestor for C++ is the same as the SQL stream interface. The requestor for C has scanf/printf-like functions which are similar to ordinary file operations (see example below).

The main advantage of using the SQL pipe is that there is no need to link database applications with Oracle libraries.

Example

```
int s=sql_open_stream("select state_name,state_code          \
                       from state                            \
                       where state_name like :v1<char[33]>",
                      50
                     );
char name[33];
int code;

sql_printf(s,"%s","M%");
while(!sql_eof(s)){
  sql_scanf(s,"%s %d",name,&code);
  printf("State=%s, Code=%d\n",name,code);
}

sql_stream_close(s);
```

4. Macros for hard-coding SQL statements

The OSCL library provides a collection of C++ macros for hard-coding SQL (or PL/SQL) statements into source code. This style of embedding SQL statements is akin to Embedded SQL but has the added advantage of requiring no special precompiler.

Example

```
class SelectState{
public:

 STRING_ITEM(state_name,33); // Define select list item “state_name”
 INT_ITEM(state_code);       // Define select list item “state_code”

 STRING_VAR(v1,33); // Define bind variable “:v1”

 BEGIN_SELECT_LIST(stm) // Define select list for “stm”
  ITEM(state_name),
  ITEM(state_code),
 END_SELECT_LIST;

 BEGIN_VAR_LIST(stm) // Define bind variable list for “stm”
  VAR(v1),
 END_VAR_LIST;

 DEFINE_SELECT_STM(stm, // Declare SELECT statement “stm”
                   “select state_name, state_code     \
                     from state                          \
                    where state_name like :v1”,
                   50
                  );

 SelectState(PConnect db){stmOpen(db);} // Open “stm”
 ~SelectState(){stmClose();} // Close “stm”

};

void Select(PConnect db)
{
 SelectState in(db);

 strcpy(in.v1,”M%”); // Assign “M%” to v1

 int rc;
 do{
  rc=in.stm->GetNext(); // Fetch next row
  if(rc==FetchOk)
  cout<<“State=”
      <<in.state_name
      <<“, Code=”
     <<in.state_code
     <<endl; // Print the row
 }while(rc==FetchOk);

} /* Select */
```

5. Exception handling

The library has an optional mechanism of raising and handling exceptions from Oracle operations. For C++ compilers which support exceptions, the exception handling mechanism was implemented on top of regular C++ exceptions. For C++ compilers which do not support exceptions, the mechanism was implemented by means of long jumps.

All implementation details are hidden in the exception handling macros which should be used to make source code portable. Macro calls remain the same regardless of their macro expansions.

Example

```
BEGIN_TRY

 SQLStream s("select state_name, state_code          \
              from state                             \
              where state_name like :v1<char[33]>",
             50 // fetch 50 rows per one fetch operation
           );
  char name[33];
 int  code;

 s<<"M%";
 while(!s.eof()){
   s>>name>>code;
   cout<<"State="<<name<<", Code="<<code<<endl;
 }

END_TRY
BEGIN_CATCH
  cerr<<SQL::err_msg()<<endl;
END_CATCH
```

6. Low level interface classes

The low level interface classes constitute the base of the library. Everything is derived from the base.Here is a fragment of the hierarchy of the basic classes: