Occiwrapper User Guide

Occiwrapper is an open source and cross platform Oracle Driver that delivers efficient access to Oracle database. It was written by C++, and offers a rich, full featured API. It helps C++ developers operate Oracle database easily.

With Occiwrapper, you can execute SQL with following format:

```
occiwrapper::Session s = SessionInstance( connection );
s << "truncate table TBL_TEST ", now;
```

The Oracle session is maintained by the Session object. The SQL is imported into the session and executed; this operation is analogous to the stream.

For parameter binding, with the help of Occiwrapper, you can directly bind the C++ variable to the Oracle parameter. For more information, please see details in the documentation afterwards.

```
occiwrapper::Session s = SessionInstance( info );
struct tm tm_value;
s << "insert into TBL_TEST( date_value ) values ( :1 )", use( tm_value ), now;
```

Meanwhile, you can directly bind a vector to the Oracle binding parameter and easily retrieve the result which is returned by select operation into a vector, and you do not need to care about the correspondence of the type. In order to improve the access performance, the batch mode is taken for writing and reading. The smart pointer is used to manage memory buffer pool automatically, which can set the Oracle developers free.

We will revise the bugs you submit ASAP, welcome to join us!

Contact author: CUCmehp@foxmail.com
QQ Group:348648166

```
occiwrapper::Session s = SessionInstance( info );
vector< int > vec1;
s << "select A from tbl_test2 t", into( vec1 ), now;
```

Most technics used in the component are from my experiences and summaries in daily work. Owing to the limitation of my Oracle and C++ knowledge, I hope we can discuss together, and I will revise the bugs you submit ASAP; your participation in revising the library is also highly appreciated.

1. Download and Installation

1.1 Third Party Component Dependency

Occiwrapper depends on the support of the Oracle OCCI library (Oracle C++ Calling Interface). Please choose the corresponding OCCI version for the different version of Oracle database.

For more information about OCCI, please refer to Oracle official documentation.

1.2 Developing Environment

In Microsoft Windows platforms, Occiwrapper can be compiled with Visual Studio 2008 SP1 or Visual Studio 2010.

In Linux platforms, it can be compiled with g++ compiler.

1.3 Operating Environment

The component works cross-platform, supporting both Windows and Linux.

1.4 Database Environment

In the ***db*** folder, you can find all the tables, procedures and functions, which the test cases use. You can easily create them by executing the scripts.

(1) Run the script create_user.sql by user "system".

(2) Run the script occiwrapper.sql by user "occiwrapper" (password is "occiwrapper").

Note: all test cases use occiwrapper as the user name of oracle.

1.5 Testing

All test cases are stored in test folder of the source code. They were successfully tested in both Windows and Linux platforms (SUSE 10, SUSE 11, OpenSuse12, Redhat Enterprise 6, and Centos 6). The Oracle connecting configurations is stored in the file db_config.ini.

All the test runs in Oracle 11gR2 and Oracle 12cR1(oracle 12.1).

2. Compiling Instruction

2.1 Windows

- Use Visual Studio 2008 SP1

Open occi_wrapper_vs2008.sln directly by Visual Studio 2008, choose Debug and Release to compile.

Notes: Requiring vs2008 SP1 or above is due to Occiwrapper using tr1 library, which is supported since vs2008 SP1.

- Use Visual Studio 2010

Open occi_wrapper_vs2010.sln directly by Visual Studio 2010, choose Debug and Release to compile.

Notes: The header files for Oracle 11.2.0.2 are stored in directory "include/occi_11g". The library files are stored in directory lib_vs2008 and lib_vs2010. The Dll file is stored in directory bin_vs2008 and bin_vs2010.

If you need other version of OCCI library, please replace them with corresponding files.

2.2 Linux

As occiwrapper depends on occi header files and library files, which always be found in the directory OCI/lib of oracle install dir. If oracle is not installed, You can find them here:

http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html

head files:： instantclient-sdk-linux.x64-*.*.*.*.*.zip

library files: instantclient-basic-linux.x64-*.*.*.*.*.zip

You Just need to choose the right version of oracle, then unzip it. Remember to put the so files, in the directories which will be found by linux system, such as /usr/local/lib, /usr/lib act. You can also add the directory to /etc/ld.so.conf.

Download occiwrapper.tar.gz，then unzip it with following command.

```
tar -zxvf occiwrapper.tar.gz
```

Grave execute permission to the file genConfigure.sh.

```
chmod +x genConfigure.sh
```

Execute file genConfigure.sh, with the following format.

```
./genConfigure.sh --occi-include=occi_include_path_value --occi-lib=occi_lib_path_value
```

occi_include_path_value refers to the directory for occi header files and occi_lib_path_value refers to occi library files. For example, header files are in the directory /u01/install/oracle_client/instantclient_12_1/sdk/include/ and the library files are in directory /u01/install/oracle_client/instantclient_12_1/, the execute command like this:

```
./genConfigure.sh
--occi-include=/u01/install/oracle_client/instantclient_12_1/sdk/include/
--occi-lib=/u01/install/oracle_client/instantclient_12_1/
```

After finish that, executing ./configure. The default destination is /usr/local. You can set it yourself.

```
./configure --prefix=/usr/local/occiwrapper
```

Then, execute command make.

```
make
```

At last, make install.

```
make install
```

Finally, Your will found include files and library files in the destiny directory. And you can find test files in the directory test. After configure the database info in the file db_config.ini, then execute it for enjoy.

3. User Guide

This part mainly introduces how to use Occiwrapper, including:

- Connecting to Oracle Database;
- Creating Session;

- Executing DDL;
- Simple Inserting;
- Binding variables when inserting;
- Updating;
- Commit and Rollback;
- Executing Procedure;
- Executing Function;
- Retrieving results when Selecting;
- Processing Date type;
- Processing NULL values.

3.1 Oracle Connection

Occiwrapper stores Oracle connection information by data structure *occiwrapper::ConnectionInfo*.

```
occiwrapper::ConnectionInfo info;
info.ip = "127.0.0.1";
info.port = 1521
info.username = "occiwrapper";
info.password = "occiwrapper";
info.sid = "orcl";
```

To open an Oracle connection, firstly you need an Oracle Environment variable. Occiwrapper calls the OCCI function createEnvironment to do that.

```
oracle::occi::Environment::createEnvironment( oracle::occi::Environment
    ::THREADED_MUTEXED );
```

The default parameter value for the function is THREADED_MUTEXED; the parameter used to create Environment by OCCI defined as below:

- ***DEFAULT:*** *not thread safe, not in object mode;*
- ***THREADED_MUTEXED:*** *thread safe, mutexed internally by OCCI;*
- ***THREADED_UN-MUTEXED:*** *thread safe, client responsible for mutexing;*

- ***OBJECT:*** *uses object features*

With Occiwrapper, the developer does not have to know the details about creation and destruction of the real oracle connection.

The connection for Oracle database was managed by Occiwrapper. You can get a database connection by the static method GetConnection. The following example shows how to create an Oracle connection:

```
shared_ptr< occiwrapper::Environment > pEnv =
        occiwrapper::Environment::CreateEnvironment();
assert( pEnv->CreateEnvironment() != NULL );
occiwrapper::ConnectionInfo info;
info.ip = "127.0.0.1";
info.username = "occiwrapper";
info.password = "occiwrapper";
info.sid = "orcl";
assert( occiwrapper::Connection::GetConnection(
            shared_ptr< occiwrapper::Environment >( pEnv ), info ) != NULL );
```

In above example, the data structure shared_ptr was introduced by tr1 library in C++0x. The details about C++ 0x and tr1 introduction please refer to Wikipedia.

- EN：http://en.wikipedia.org/wiki/C%2B%2B11
- CN：http://zh.wikipedia.org/wiki/C%2B%2B11

Occiwrapper adopts connection pool to manage Oracle connection. Operation for the same oracle schema will share the same connection. The following codes show how to use Occiwrapper connection pool to obtain Oracle connection.

```
occiwrapper::ConnectionInfo info;
occiwrapper::ConnectionPool connPool;
info.ip = "127.0.0.1";
info.username = "occiwrapper";
info.password = "occiwrapper";
info.sid = "orcl";
shared_ptr< occiwrapper::Connection > p = connPool.GetConnection( info );
assert( p != NULL);
assert( p->GetValidity() == occiwrapper::VALID );
shared_ptr< occiwrapper::Connection > other = connPool.GetConnection( info );
assert( connPool.GetConnMapSize() == 1 );
```

3.2 Session Creation

The class Session manages the sessions between client and Oracle server. You can flexibly create certain Statement to execute SQL command via client.

The Occiwrapper creates session by SessionFactory, which is a singleton pattern factory class and can creates Session dynamically. Moreover, a built-in connection pool in SessionFactory manages Oracle connection. For more information about connection pool, see Chapter 3.2.

The following codes show how to use SessionFactory to create a Session.

```
occiwrapper::ConnectionInfo info;
occiwrapper::Session s =
        occiwrapper::SessionFactory::Instance().Create( info, false );
```

You can also use macro to create Session more easily.

```
occiwrapper::Session s = SessionInstance( info );
```

3.3 Execute DDL

Use Session to execute DDL command conveniently. The following example shows how to create, clear and delete a table.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet = false;
string strErrMsg = "";
s << "create table tbl_test( x int )", now, bRet, strErrMsg;
s << "truncate table tbl_test", now, bRet, strErrMsg;
s << "drop table tbl_test", now, bRet, strErrMsg;
```

3.4 Execute Simple Insert

The simple insert is an insert without binding variable. Here we only introduce the simplest SQL, the next two chapter will show you how to use Occiwrapper to insert binding variable.

For example, Table *TBL_TEST1* only contains a round figure X. Insert the following code 10004 into it.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet = false;
string strErrMsg = "";
s << "insert into tbl_test1( x ) values ( 10004 )", now, bRet, strErrMsg;
```

The result of Session will be stored in bRet. If it runs failed, the error cause will be output in strErrMsg.

3.5 Insert binding variable

The binding variable is an important skill in Oracle programming. By using it, the performance of a repeated SQL execution can be improved greatly. We do not explain the issue further here.

To introduce the binding variables, we divide them into 2 groups, including binding simple variable and binding container.

- Binding simple variable

Bind the simple data variable to Oracle binding variable, such as *int*, *float*, *double*, *string*, *struct tm*.

The following form shows the binding types supported by the occiwrapper.

Form1 Binding types supported by occiwrapper in *Windows*

occiwrapper Type Definition	Operating System Types
Int8	signed char
UInt8	unsigned char
Int16	signed short
UInt16	unsigned short
Int32	int
UInt32	unsigned int
Int64	signed __int64
UInt64	unsigned __int64
float	float
double	double
struct tm	struct tm
std::string	std::string

Form2 Binding types supported by occiwrapper in Linux

occiwrapper Type Definition	Operating System Types
Int8	signed char
UInt8	unsigned char
Int16	signed short

UInt16	unsigned short
Int32	int
UInt32	unsigned int
Int64	signed long
UInt64	signed long long
float	float
double	double
struct tm	struct tm
std::string	std::string

The Occiwrapper binds the simple variable via ***use***. For example, execute *use(1)* to bind the number 1 with corresponding binding variable, execute *use("hello world")* to bind the character string hello world with the binding variable.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet = false;
string strErrMsg = "";
s << "insert into tbl_test1( x ) values ( :1 )", use( 2 ), now, result, err_msg;
cout << "result: " << result << endl << "error message: " << err_msg << endl;
assert( result );

s << "insert into test_string( id, string_val ) values( :1, :2 )", use( 2 ),
use( "CUCMehp" ), now, result, err_msg;
cout << "result: " << result << endl << "error message: " << err_msg << endl;
assert( result );

s << "insert into test_number( id, number_value ) values( :1, :2 )", use( 1 ),
use( 3.5 ), now, result, err_msg;
cout << "result: " << result << endl << "error message: " << err_msg << endl;
assert( result );
```

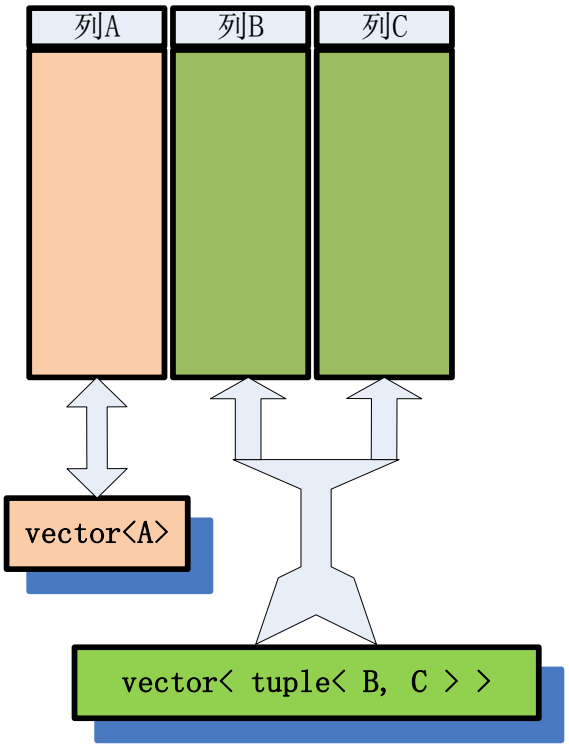
A Session can be bind repeatedly; the following codes insert 0-9 into table *TBL_TEST1*.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg;
s << "truncate table tbl_test1", now;
occiwrapper::Statement stmt = s << "insert into tbl_test1( x ) values( :1 )";
for( int i = 0; i < 10; ++ i )
{
    stmt, use( i ), now, bRet, strErrMsg;
    assert( bRet );
}
```

- Binding Container

The binding container binds certain columns in table and one or more containers (*1.0.0* version only support vector) together.

As shown in the diagram below, a table contains three columns, A、B、C, you can bind data in column A with variable vector<A>,bind data in column B and column C with variable vector< tuple< B, C> >,the tuple is a tuple type and for its definition ,you can refer to library tr1,C++ 0x standard.



You can flexibly realize above binding via ***batched_use*** in Occiwrapper. The following example shows how to insert the batch vectors created by array {20, 21, 22, 23, 24} into table *TBL_TEST1*.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg;
int a[5] = { 20, 21, 22, 23, 24 };
vector< int > vec( a, a + 5 );
s << "insert into tbl_test1( x ) values ( :1 )", batched_use( vec ), now, bRet, strErrMsg;
assert(bRet);
```

Below example shows how to bind result into two *vectors* or one *vector<tuple>*. The table *TEST_STRING* contains two fields, *ID*(integer) and *STRING_VAL*(varchar).

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg;
int a[5] = { 20, 21, 22, 23, 24 };
vector< int > vec( a, a + 5 );
string strArray[5] = { "message1", "message2", "message3", "message4",
"message5" };
vector< string > vecStr( strArray, strArray + 5 );
s << "insert into test_string( id, string_val ) values ( :1, :2 )", batched_use( vec ),
batched_use( vecStr ), now, bRet, strErrMsg;
assert( bRet );
s << "truncate table test_string", now, bRet;
assert( bRet );
vector< tuple< int, string > > vecTuple;
for( size_t i = 0; i < 5; ++ i )
{
    vecTuple.push_back( make_tuple( vec[ i ], vecStr[ i ] ) );
}
s << "insert into test_string( id, string_val ) values ( :1, :2 )", batched_use( vecTuple ),
now, bRet, strErrMsg;
assert( bRet );
```

3.6 Execute Update

The execution of update is basically similar to DDL and select,you can also use binding variable. Here's a simple example.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet = false;
string strErrMsg = "";
s << "truncate table tbl_test1", now, bRet, strErrMsg;
s << "insert into tbl_test1( x ) values ( 10004 )", now, bRet, strErrMsg;
s << "update tbl_test1 set x = :1 where x = 10004", use( 10005 ), now, bRet, strErrMsg;
```

In above example, firstly insert value 10004 into X field of table *TBL_TEST1*, then modify the value from 10004 to 10005.

3.7 Commit and *Rollback*

The Occiwrapper supports two Commit executions. You can designate the type of Commit when you create Session, and the commit will run automatically if you set the parameter of isAutoCommit true, otherwise you need manually run commit or rollback to let the execution become effective.

```
occiwrapper::Session occiwrapper::SessionFactory::Create( const ConnectionInfo&
        connInfo, bool bAutoCommit )
```

The following codes is an example for you.

```
occiwrapper::Session s = occiwrapper::SessionFactory::Instance().Create( info,
        false );
s << "truncate table tbl_test1", now;
s.Commit();
bool bRet = false;
string strErrMsg = "";
s << "insert into tbl_test1( x ) values ( 10005 )", now, bRet, strErrMsg;
s.Rollback();
occiwrapper::Statement stmt2 = ( s << "insert into tbl_test1( x ) values ( 10005 )" );
for ( int i = 0; i < 2; ++i)
{
    stmt2.execute();
}
s.Commit();
```

Creating a non-Automatic submit Session firstly, then inserting value 10005 without submission and on the contrary cancel it. Now you will find the table is blank if you check it by PL/SQL tool. Execute Insert twice and submit, then you will get two records in the table.

3.8 Execute Memory Process

The Oracle memory process includes input parameter and output parameter, the input parameter is pass to Oracle, while the output

parameter is retrieve out from Oracle. When you adopt *Use*, you attach *PAR_IN* as input parameter, *PAR_OUT* as output parameter.

In the following example, the memory process p_test_procedure includes two parameters, one's type is int and the other's is varchar2. The function changes the int to string and output via second parameter.

Following codes shows above process.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg;
int nParIn = 1000;
string strParOut = "";
s << "begin p_test_procedure( :1,:2 ); end;", use( nParIn,
        occiwrapper::PAR_IN ), use( strParOut, occiwrapper::PAR_OUT ),
        now, bRet, strErrMsg;
assert( bRet );
assert( strParOut == "1000" );
```

3.9 Call Function

Function Calling is similar to invoking process, but the function returns value, which is output type.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper",
        "occiwrapper", "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg = "";
int a = 0;
s << "begin :1 := f_ins_tbl_test1( :2 ); end;", use( a,
        occiwrapper::PAR_OUT ), use( 2, occiwrapper::PAR_IN ) , now,
        bRet, strErrMsg;
assert( bRet );
int b;
s << "begin :1 := f_test2( :2, :3 ); end;", use( a, occiwrapper::PAR_OUT ),
        use( 2 ), use( b, occiwrapper::PAR_OUT ), now, bRet, strErrMsg;
assert( bRet );
```

3.10 Execute Select and Save Results

The select operation is the most common operation in SQL, it gets data out of the database. The Occiwrapper realizes the binding of field and container via keyword *into*.

```
vector< string > vStr;
vector< struct tm > vDate;
vector< occiwrapper::Int32 > vInt;
vector< float > vFloat;
vector< double > vDouble;
s << "select string_value, date_value, int_value, float_value, number_value
        from test_batched_table", into( vStr ), into( vDate ), into( vInt ),
        into( vFloat ), into( vDouble ), now, bRet, strErrMsg;
assert( bRet );
```

Above codes read data in table *tbl_batched_table*, and store them in 5 vector groups.

The Occiwrapper inner adopts batch reading mode, which can improve the reading performance and automatically manage the memory space required in batch binding by itself.

When the table contains a large amount of data but you only need certain items, you can use Limit.

The following codes give an example to you.

```
vector< int > vec1;
vector< int > vec2;
// test limit select
s << "select * from tbl_test2 t", into( vec1 ), into( vec2 ),
        occiwrapper::Limit( 3 ), now;
```

3.11 Process Oracle date types

In order to process Oracle date types, the type of external processing in Occiwrapper is *struct tm*, which means the type of insert and read are all *struct tm* in occiwrapper.

The following example shows how to insert present time and read it from Oracle to a vector array.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
bool bRet;
string strErrMsg;
struct tm tmValue;
time_t nowTime = time( NULL );
localtime_r( &nowTime, &tmValue);
s << "truncate table test_date", now, bRet, strErrMsg;
s << "insert into test_date( date_val ) values ( :1 )", use( tmValue ), now, bRet;
vector< struct tm > vTmDb;
s << "select date_val from test_date", into( vTmDb ), now, bRet, strErrMsg;
```

3.12 Process NULL value

The ociwrapper set default value into vector, when retrieve NULL value from database.

occiwrapper Type Definition	Default
Int8	*0*
UInt8	*0*
Int16	*0*
UInt16	*0*
Int32	*0*
UInt32	*0*
Int64	*0*
UInt64	*0*
float	*0*
double	*0*
struct tm	*1900-1-1 0:0:0*
std::string	“”

If you don’t want to use above default value, using nvl() function is recommended. For example, select nvl(a, -1) from tbl_test.

3.13 Batch Insert

In Chapter 3.5, the data batch import is realized by binding container variable.

Here’s another example for multi-type data batch inserting.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
struct tm tm_value;
time_t now_time = time( NULL );
localtime_s( &tm_value, &now_time );
vector< string > vec0;
vec0.push_back( "123456" );
vec0.push_back( "222222" );
vector< struct tm >   vec1;
vec1.push_back( tm_value );
vec1.push_back( tm_value );
vector< occiwrapper::Int8 > vec2;
vec2.push_back( 1 );
vec2.push_back( 2 );
vector< float > vec3;
vec3.push_back( 0.1 );
vec3.push_back( 0.2 );
vector< double > vec4;
vec4.push_back( 10000 );
vec4.push_back( 100 );
s << "insert into test_batched_table( string_value, date_value, int_value, float_value,
        number_value ) values ( :1, :2, :3, :4, :5 )", batched_use( vec0 ),
        batched_use( vec1 ), batched_use( vec2 ), batched_use( vec3 ),
        batched_use( vec4 ), now, bRet, strErrMsg;
assert( bRet );
```

3.14 Batch Read

The batch read is required in case a table contains a large number of records. The Occiwrapper provides a very convenient batch reading mode, which can read the results to the container directly and shield complex binding types and memory processing.

In Chapter 3.10, we provided a SQL, which can read all data in the table at a time.

In fact, a table often contains millions of data, even hundreds of millions data, therefore it will be out of memory if you read them only

once. In that case, you need batch read repeatedly. Please refer to the following codes.

```
occiwrapper::ConnectionInfo info( "127.0.0.1", 1521, "occiwrapper", "occiwrapper",
        "orcl" );
occiwrapper::Session s = SessionInstance( info );
struct tm tm_value;
time_t now_time = time( NULL );

vector< int > vec1;
vector< int > vec2;
// test limit select, getting all of data to the vectors
occiwrapper::Statement stmt = ( s << "select * from tbl_test2 t", into( vec1 ),
        into( vec2 ), limit( 10000 ) );
do
{
    vec1.clear();
    vec2.clear();
    stmt.Execute();
}
while( stmt.HasNext() );
```

By executing above codes, every time you call Execute, you can get 10000 records to the container vec1 and vec2 from the table. When you call Execute again, another 10000 records will be added.