Subject: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Fri, 10 Feb 2012 19:02:45 GMT
View Forum Message <> Reply to Message

I haven't been able to find any information on it.

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by dolik.rce on Sat, 11 Feb 2012 09:17:41 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Fri, 10 February 2012 20:02I haven't been able to find any information on it.

Do you mean callbacks? There is a special type of callback called Gate that can return bool. For anything else you can use a function that takes reference to variable as parameter and use that to communicate the result to the caller.

Best regards,
Honza

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by BioBytes on Sat, 11 Feb 2012 15:50:18 GMT
View Forum Message <> Reply to Message

Hello papascalientes,

I guess you mean a prepared statement as follows:

Inserting data with a prepared statement
```
PreparedStatement* pStatement = pDatabase->PrepareStatement(_("INSERT INTO table1
(column1, column2) VALUES (?, ?)"));
if (pStatement)
{
 pStatement->SetParamString(1, _("One"));
 pStatement->SetParamString(2, _("Two"));
 pStatement->RunQuery():
 pDatabase->CloseStatement(pStatement);
}
```

or

```
PreparedStatement* pStatement = pDatabase->PrepareStatement(_("SELECT * FROM table1
WHERE column1 - ?)"));
if (pStatement)
{
```

```
 pStatement->SetParamString(1, _("One"));
 DatabaseResultSet* pResults = pStatement->RunQueryWithResults():
 if (pResults)
 {
  while (pResults->Next())
  {
   wxString strOne = pResults->GetResultString(_("column1"));
   wxString strTwo = pResults->GetResultString(_("column2"));
  }
  pDatabase->CloseResultSet(pResults);
 }
 pDatabase->CloseStatement(pStatement);
}
```

This is possible when using databaselayer library designed for wxWidgets.

For U++ have a look to:

2. Using global main database, executing statements with parameters, getting resultset info

Most applications need to work with just single database backend, therefore repeating SqlSession parameter in all Sql declarations would be tedious.

To this end U++ supports concept of "main database" which is represented by SQL variable. SQL is of Sql type. When any other Sql variable is created with default constructor (no session parameter provided), it uses the same session as the one the SQL is bound to. To assign session to global SQL, use operator=:

```
#include <Core/Core.h>

#include <plugin/sqlite3/Sqlite3.h>
using namespace Upp;

CONSOLE_APP_MAIN

    Sqlite3Session sqlite3;

    if(!sqlite3.Open(ConfigFile("simple.db"))) {

        Cout() << "Can't create or open database file\n";

        return;
    }

#ifdef _DEBUG

    sqlite3.SetTrace();
```

```
#endif
    SQL = sqlite3;
    SQL.Execute("drop table TEST");
    SQL.ClearError();
    SQL.Execute("create table TEST (A INTEGER, B TEXT)");
    for(int i = 0; i < 10; i++)
        SQL.Execute("insert into TEST(A, B) values (?, ?)", i, AsString(3 * i));
    Sql sql;
    sql.Execute("select * from TEST");
    for(int i = 0; i < sql.GetColumns(); i++)
        Cout() << sql.GetColumnInfo(i).name << '\n';
    while(sql.Fetch())
        Cout() << sql[0] << " \"" << sql[1] << "\"\n";
}
```

As global SQL is regular Sql variable too, it can be used to issue SQL statements.

I don't know if this is what you are looking for.

Biobytes

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Sun, 12 Feb 2012 07:56:03 GMT
View Forum Message <> Reply to Message

I'm not quite sure I understand callbacks in the context of a database query.

How I normally call a stored procedure in DTL:

```
Parameters params;
params.A = "something";
params.B = "something";
```

```
// params.C is an output parameter

// BPA is just a parameter mapping structure, similar to .sch file
DBView<Parameters> view (
   "{call stored_procedure_x (?, ?, ?)}",
   stored_procedure_bpa()
   );

// Create an iterator pointing to the view
DBView<Parameters>::sql_iterator sprocIter = view.begin();

// Copy params into the stored procedure iterator
*sprocIter = params;

// In DTL, incrementing an iterator executes stored procedure
++sprocIter;

// Now sprocIter->C will hold results from calling the stored procedure
```

How do you do this in Ultimate++?

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by mirek on Sun, 12 Feb 2012 07:59:24 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Fri, 10 February 2012 14:02I haven't been able to find any information on it.

Note: Previous reply are off-topic.

The answer depends on DB and U++ connector used.

What database do you have in mind?

Mirek

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Sun, 12 Feb 2012 09:37:25 GMT
View Forum Message <> Reply to Message

Mirek,

I'm using SQL Server with MSSQL in U++.

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by dolik.rce on Sun, 12 Feb 2012 11:21:46 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Sun, 12 February 2012 08:56I'm not quite sure I understand callbacks in the context of a database query.I sorry, I completely overlooked that the question is in SQL subforum Callbacks are really not relevant here...

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Mon, 13 Feb 2012 17:06:38 GMT
View Forum Message <> Reply to Message

No problem at all.

I know that stored procedures are being executed in the same form. If I do this:

```
SQL.Execute (
   "{call stored_procedure_x (?,?,?)}",
   item.A,
   item.B,
   item.C
   );
```

The stored procedure IS being executed and runs, but item.C is an output parameter and isn't being populated, obviously. Performing a SQL.Fetch() doesn't show any returned rows either.

It appears there is some other medium by which this output parameter is assigned. Other places suggest an RPC mechanism whereas in the past MSSQL relied on returning a result set.

Irritating.

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by mirek on Mon, 13 Feb 2012 17:58:16 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Mon, 13 February 2012 12:06No problem at all.

I know that stored procedures are being executed in the same form. If I do this:

```
SQL.Execute (
   "{call stored_procedure_x (?,?,?)}",
   item.A,
   item.B,
   item.C
   );
```

The stored procedure IS being executed and runs, but item.C is an output parameter and isn't being populated, obviously. Performing a SQL.Fetch() doesn't show any returned rows either.

It appears there is some other medium by which this output parameter is assigned. Other places suggest an RPC mechanism whereas in the past MSSQL relied on returning a result set.

Irritating.

MSSQL package is based on ODBC, after going through ODBC docs and out code a bit, it looks like we do not support it yet - but it is desirable to add such support. I might look into it in during this or next week, contributions in this area are welcome as well!

Mirek

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Mon, 13 Feb 2012 22:47:47 GMT
View Forum Message <> Reply to Message

Ahh, OK. Yeah I looked through ODBC.cpp and noticed it only binds input parameters under Execute. I'll have to refer to the source more often

Thanks for looking into it, Mirek.

Subject: Re: Is it possible to call a stored procedure with an output parameter?
Posted by jjacksonRIAB on Fri, 17 Feb 2012 20:49:03 GMT
View Forum Message <> Reply to Message

I tried to figure out how I could get this working in U++ but I can't get return values out of MSSQL.

So far under ODBC.cpp I made these changes:

```
struct Param {
```

```
   int    ctype;
   int    sqltype;
   String data;
   SQLLEN li;
   int    direction;
   };
```

Then I modified ODBCConnection::Execute and I'm still trying to get this to work:

```
   bool ODBCConnection::Execute()
   {
 bool isStoredProcedure = false;

 LLOG("Execute " << (void *)this << " " << (void *)session);
 if(session->hstmt == SQL_NULL_HANDLE)
  return false;
 if(IsCurrent())
  session->current = NULL;
 session->FlushConnections();
 last_insert_table.Clear();
 number.Clear();
 text.Clear();
 time.Clear();
 CParser p(statement);

 // parse for evidence of a stored procedure call
 if(p.Char('{'))
 {
  p.Spaces();

  if(p.Id("call") || p.Id("CALL")) {
   procedure_name = p.ReadId();
   isStoredProcedure = true;
   //Cout() << "Proc name: " << procedure_name << "\n";
  }

  SDWORD      cbValue5;
  SDWORD      cbValue4;

  SQLSMALLINT ParameterType = SQL_PARAM_INPUT;

  if(!IsOk(SQLProcedureColumns (
   session->hstmt,
   NULL,
   0,
   NULL,
```

```
 0,
 (SQLCHAR *)~procedure_name,
 procedure_name.GetLength(),
 NULL,
 0
 )))
{
 SQLFreeStmt(session->hstmt, SQL_CLOSE);
 return false;
}

char parameter_name [20];

if(!IsOk(SQLBindCol(
 session->hstmt,
 4, // Column 5 returns column name
 SQL_C_CHAR,
 parameter_name,
 sizeof(parameter_name),
 &cbValue4
 )))
{
}

if(!IsOk(SQLBindCol(
 session->hstmt,
 5, // Column 5 returns whether parameter is input or output
 SQL_C_SHORT,
 &ParameterType,
 0,
 &cbValue5
 )))
{
}

int i = 0;
while(SQLFetch(session->hstmt) == SQL_SUCCESS)
{
 Param& p = param[i];

 switch (ParameterType)
    {
      case SQL_PARAM_INPUT:
      case SQL_PARAM_OUTPUT:
      case SQL_PARAM_INPUT_OUTPUT:
  p.direction = ParameterType;
  i++;
       break;
```

```
        default:
          break;
      }
  }

  if(IsCurrent()) session->current = NULL;
  session->FlushConnections();

  number.Clear();
  text.Clear();
  time.Clear();
 }

 if((p.Id("insert") || p.Id("INSERT")) && (p.Id("into") || p.Id("INTO")) && p.IsId())
  last_insert_table = p.ReadId();
 if(!IsOk(SQLPrepare(session->hstmt, (SQLCHAR *)~statement, statement.GetCount())))
  return false;
 parse = false;
 bparam = param;
 param.Clear();
 for(int i = 0; i < bparam.GetCount(); i++) {
  Param& p = bparam[i];
  SQLSMALLINT     DataType;
  SQLULEN         ParameterSize;
  SQLSMALLINT     DecimalDigits;
  SQLSMALLINT     Nullable;

  if(!IsOk(SQLDescribeParam(session->hstmt, i + 1, &DataType, &ParameterSize,
&DecimalDigits, &Nullable)))
    return false;

  Cout() << "Param direction: " << p.direction << "\n";

  if(!IsOk(SQLBindParameter(session->hstmt, i + 1, p.direction, p.ctype, DataType,
                            ParameterSize, DecimalDigits, (SQLPOINTER)~p.data,
                            p.data.GetLength(),
                            &p.li))) {
    return false;
  }

 }
 SQLSMALLINT ncol;

 SQLExecute(session->hstmt);

 if(isStoredProcedure) {
     while (SQLMoreResults(session->hstmt)!= SQL_NO_DATA );
```

```
}

if(/*!IsOk(SQLExecute(session->hstmt)) ||*/ !IsOk(SQLNumResultCols(session->hstmt, &ncol))) {

 SQLFreeStmt(session->hstmt, SQL_CLOSE);
 return false;
}


session->current = this;
info.Clear();
binary.Clear();
for(int i = 1; i <= ncol; i++) {
 SQLCHAR      ColumnName[256];
 SQLSMALLINT  NameLength;
 SQLSMALLINT  DataType;
 SQLULEN      ColumnSize;
 SQLSMALLINT  DecimalDigits;
 SQLSMALLINT  Nullable;
 if(!IsOk(SQLDescribeCol(session->hstmt, i, ColumnName, 255, &NameLength, &DataType,
                         &ColumnSize, &DecimalDigits, &Nullable)))
  return false;
 binary.Add(false);
 SqlColumnInfo& f = info.Add();
 f.nullable = Nullable != SQL_NO_NULLS;
 f.binary = false;
 f.precision = DecimalDigits;
 f.scale = 0;
 f.width = ColumnSize;
 f.name = (char *)ColumnName;
 switch(DataType) {
 case SQL_DECIMAL:
 case SQL_NUMERIC:
 case SQL_SMALLINT:
 case SQL_INTEGER:
 case SQL_REAL:
 case SQL_FLOAT:
 case SQL_DOUBLE:
 case SQL_BIT:
 case SQL_TINYINT:
  f.type = DOUBLE_V;
  break;
 case SQL_BIGINT:
  f.type = INT64_V;
  break;
 case SQL_TYPE_DATE:
 case SQL_TYPE_TIMESTAMP:
  f.type = TIME_V;
```

```
  break;
 case SQL_BINARY:
 case SQL_VARBINARY:
 case SQL_LONGVARBINARY:
  f.type = STRING_V;
  f.binary = true;
  binary.Top() = true;
  break;
 default:
  f.type = STRING_V;
  break;
 }
}
 SQLLEN rc;
 SQLRowCount(session->hstmt, &rc);
 rowsprocessed = rc;

 return true;
   }
```

I am calling SQLProcedureColumns to find out whether a given bound column is an input column, output column, or bidirectional. They say this is an expensive call so I'm considering wrapping it in a different method that will cache those parameters for later. These changes also do not bind return values (yet), just output parameters.

The problem I'm having is that even after calling SQLMoreResults (after SQLExecute), the documentation says those bound parameters should be modified, yet they aren't being changed - I can look through and confirm that all the direction types are correct.

Where did I go wrong? Am I doing this stupidly?
