Subject: Firebird
Posted by Novo on Thu, 30 Jun 2011 04:16:06 GMT
View Forum Message <> Reply to Message

I made an initial version of a firebird driver for UPP. Below is what it can do at this time. This code was tested on Windows with local connection to firebird 2.1 so far.

```
#include "firebird/firebird.h"

using namespace Upp;

void TestInt(FBSession& s)
{
 Sql stmt(s);
 stmt.SetStatement(
     "DROP TABLE test_long_table"
 );
 stmt.Run();

 // Data definition statement.
 stmt.SetStatement(
  " CREATE TABLE test_long_table( \n"
  "   id integer, \n"
  "   first_field integer DEFAULT NULL, \n"
  "   second_field integer DEFAULT NULL \n"
  ")"
 );
 stmt.Run();

 stmt.SetStatement(
  " INSERT INTO test_long_table \n"
  " VALUES(?, ?, ?)"
 );

 for (int i = 0; i < 100; ++i)
 {
  stmt.SetParam(0, i);
  stmt.SetParam(1, i * 10);
  stmt.SetParam(2, i * 100);
  stmt.RunX();
 }

 stmt.SetStatement(
  " SET TRANSACTION WAIT ISOLATION LEVEL READ COMMITTED"

 );
 stmt.Run();
```

```
int _0 = -1;
int _1 = -1;
int _2 = -1;

Ref r0(_0);
Ref r1(_1);
Ref r2(_2);
int i;

// SELECT without parameters.
stmt.SetStatement(
 " SELECT * FROM test_long_table ORDER BY id"
);

stmt.Run();
i = 0;
while (stmt.Fetch())
{
 stmt.GetColumn(0, r0);
 stmt.GetColumn(1, r1);
 stmt.GetColumn(2, r2);

 ASSERT(_0 == i);
 ASSERT(_1 == i * 10);
 ASSERT(_2 == i * 100);
 ++i;
}
ASSERT(i == 100);
ASSERT(!stmt.Fetch());

// Rerun SELECT.
stmt.Run();
i = 0;
while (stmt.Fetch())
{
 ASSERT(stmt[0] == i);
 ASSERT(stmt[1] == i * 10);
 ASSERT(stmt[2] == i * 100);
 ++i;
}
ASSERT(i == 100);
ASSERT(!stmt.Fetch());

// SELECT with parameters.
stmt.SetStatement(
 " SELECT * FROM test_long_table WHERE id < ? ORDER BY id"
);
```

```
 stmt.SetParam(0, 50);

 stmt.Run();
 i = 0;
 while (stmt.Fetch())
 {
  stmt.GetColumn(0, r0);
  stmt.GetColumn(1, r1);
  stmt.GetColumn(2, r2);

  ASSERT(_0 == i);
  ASSERT(_1 == i * 10);
  ASSERT(_2 == i * 100);
  ++i;
 }
 ASSERT(i == 50);
 ASSERT(!stmt.Fetch());

 // Rerun SELECT with parameters.
 stmt.SetParam(0, 25);

 stmt.Run();
 i = 0;
 while (stmt.Fetch())
 {
  stmt.GetColumn(0, r0);
  stmt.GetColumn(1, r1);
  stmt.GetColumn(2, r2);

  ASSERT(_0 == i);
  ASSERT(_1 == i * 10);
  ASSERT(_2 == i * 100);
  ++i;
 }
 ASSERT(i == 25);
 ASSERT(!stmt.Fetch());
}

CONSOLE_APP_MAIN
{
 const Vector<String>& cmd_line = CommandLine();
 if (cmd_line.GetCount() > 0 && FileExists(cmd_line[0]))
 {
  FBSession s;
  s.Connect(
   cmd_line[0],
   NULL,
   "SYSDBA",
```

```
   "masterkey"
   );

  TestInt(s);
  TRANSACTION(s) {
   TestInt(s);
  }
 }
}
```

Actually, it can do more than that. It should support all data types except of blobs and arrays.

It also introduces an interesting macro called TRANSACTION. An example:

```
FBSession s;

TRANSACTION(s) {
 TestInt(s);
}
```

If code inside of a TRANSACTION block successfully reaches end of the block, then transaction will be committed. If for some reason code jumps out of the block (in case of an exception, for example), then transaction will be rolled back.

In this driver I tried to simulate transactional behavior similar to one found in ORACLE. The way firebird itself deals with transactions is very different.

I'd like to know you opinion about this driver, design of code, transactional behavior, e.t.c.

TIA
