JTOpen jdbcClient – simplifies Java SQL development

Open source utility makes JDBC testing quick and easy

John W. Eberhard

January 08, 2015
(First published January 08, 2015)

JTOpen *jdbcClient* provides a convenient and quick access to JDBC supported databases for application developers. Allowing easy access to databases using a variety of JDBC drivers, a developer can quickly and easily run SQL statements. Using the prepared statements and Java reflection, this client allows developers to easily use the advanced JDBC features without writing any Java™ code. It is the perfect tool for developers who want to easily use JDBC without the expense of creating and compiling a JDBC program.

Introduction

Starting with version 7.6, JTOpen (version 7.6 and later) includes a simple JDBC client program, *jdbcClient*, to simplify life for JDBC and SQL developers. *jdbcClient* is a command-line client that is implemented using only Java and can be run from any platform that has a Java virtual machine (JVM). The *jdbcClient* program allows you to easily connect to a database and run the SQL statements. For most Java developers, this is easier than starting another program, such as System i Navigator's Run SQL scripts. Using this client, you can quickly view the results of running particular SQL statements and experiment with the intricacies of JDBC methods without the need to write and compile a Java program.

To use *jdbcClient*, you have to simply add the .jar file for the JDBC driver to your class path and specify a JDBC URL for the database server to which you wish to connect. Once connected to a database server, you can execute SQL commands against that server. *jdbcClient* also allows the use of JDBC PreparedStatements to pass a variety of parameter types to the database. As an enhanced developer tool, *jdbcClient* also allows the use of reflection to involve methods on Java and JDBC objects.

Connecting to a database

jdbcClient is included with JTOpen's jt400.jar file and is named com.ibm.as400.access.jdbcClient.Main. To use the client, start the Main program and pass 3

arguments to the program: the JDBC URL, the user ID, and the password. The JDBC URL is a JDBC URL that is accepted by Java's DriverManager.getConnection method. Because the test program accepts a JDBC URL, it can use other JDBC drivers besides the JTOpen JDBC driver, as long as they are included in the class path.

To connect to IBM® DB2® for i using JTOpen driver on a Microsoft® Windows® or Linux® platform, you just start *jdbcClient* with jt400.jar on your class path and add the JDBC URL, user ID, and password.

```
java –cp jt400.jar com.ibm.as400.access.jdbcClient.Main jdbc:as400:SYSTEM USERID PASSWORD
```

You can also connect to DB2 for i using the native JDBC driver by starting the client from the Qshell Interpreter (qsh). In this case, you again need to add jt400.jar to the class path, but in this case, it can be loaded from its shipped location on the system. You do not need to add the .jar file for the native JDBC driver because it is already found on the Java extensions class path. When connecting to the local system using the native JDBC driver, you do not need to specify a user ID and password on the connection request. The native JDBC driver will use the current user ID for the connection when a user ID is not specified.

```
java –cp /QIBM/ProdData/OS400/JT400/lib/jt400.jar com.ibm.as400.access.jdbcClient.Main jdbc:db2:localhost
```

You can also connect to DB2 for i using the IBM DB2 Connect™ JDBC driver from an IBM AIX®, Windows, or Linux system. In this case, your class path must contain jt400.jar as well as the JAR files for the driver: db2jcc4.jar and db2jcc_license_cisuz.jar. In this case, the JDBC URL has the format jdbc:db2://SYSTEM:446/*LOCAL, where SYSTEM is the *system name*, 446 is the *port for the server* on IBM i, and *LOCAL indicates that the *local database* is to be used.

```
java –cp jt400.jar:db2jcc4.jar:db2jcc_license_cisuz.jar
com.ibm.as400.access.jdbcClient.Main jdbc:db2://SYSTEM:446/*LOCAL USERID PASSWORD
```

You can also connect to databases other than DB2 for i. When running on Linux, you can connect to a MySQL database by including mysql.jar on the class path and using the jdbc:mysql JDBC URL.

```
java –cp jt400.jar: mysql.jar com.ibm.as400.access.jdbcClient.Main
jdbc:mysql://localhost/DATABASE USERID PASSWORD
```

jdbcClient is a useful tool that can be used to connect to a wide variety of database systems using a wide variety of JDBC drivers.

Running simple statements

After you are connected, *jdbcClient* provides a command prompt for entering SQL statements and other commands. This prompt assumes that everything is an SQL command, with the exception of *jdbcClient* commands which begin with '!'.

Here is an example of running various SQL statements against a DB2 for i server to create a table, insert data into the table, retrieve the data from the table, and call a stored procedure.

```
>java -cp jt400.jar com.ibm.as400.access.jdbcClient.Main jdbc:as400:SYSTEM UID PWD
>create table sample(c1 int, c2 varchar(80))
>insert into sample values(1,'abc')
>select * from sample
C1,C2
1,abc
>call sysibm.sqltables(null,null,'SAMPLE',null,null)

*** Warning ***

SQLState: 0100C
Message: [SQL0466] 1 result sets are available from procedure SQLTABLES in SYSIBM.
Vendor: 466

TABLE_CAT,TABLE_SCHEM,TABLE_NAME,TABLE_TYPE,REMARKS
SYSTEM,UID,SAMPLE,TABLE,null
```

As can be seen above, the query output of *jdbcClient* is very simple and merely delimits the output columns using commas. The first output row contains the column names and the remaining rows contain the output data. *jdbcClient* also displays the warnings that are encountered, as seen by the **SQL0466** warning above.

Parameters can be passed to a stored procedure call using `–INPARM` after the `call` statement. Here is an example of calling a procedure using this syntax.

```
>call sysibm.sqltables(null,null,?,null,null) -- INPARM SAMPLE

*** Warning ***
SQLState: 0100C
Message: [SQL0466] 1 result sets are available from procedure SQLTABLES in SYSIBM.
Vendor: 466

TABLE_CAT,TABLE_SCHEM,TABLE_NAME,TABLE_TYPE,REMARKS
SYSTEM,UID,SAMPLE,TABLE,null
```

The *jdbcClient* recognizes that a result set was returned by the stored procedure and displays the contents of the result set.

Using prepared statements

Using the *jdbcClient* client commands, you can also run the *prepared* statements which are commonly used in Java applications. The *prepared* statements are often used in conjunction with parameter markers, which allow the parameters to be changed each time the statement is run. Using the *prepared* statements usually consists of three parts: preparing the statement, setting the parameters, and running the statement.

The `!PREPARE` command is used to prepare a statement. The `!SETPARM` command is used to set parameters. The `!EXECUTEUPDATE` or `!EXECUTEQUERY` command is used to run the prepared statement. Here is an example of using the *prepared* statements to insert data into a table.

```
>!PREPARE insert into sample values(?,?)
>!SETPARM 1,10
>!SETPARM 2,insert1
>!EXECUTEUPDATE
>!SETPARM 1,10
>!SETPARM 2,insert2
>!EXECUTEUPDATE
>select * from sample where c1=10
C1,C2
10,insert1
10,insert2
```

In this example, two rows were added to the table and then a query was run to illustrate that the data was actually inserted.

For the `!SETPARM` command, a variety of data types are supported. The various types are shown in the following table.

Table 1. Data types supported by the !SETPARM command

Syntax	Description
UX'....'	Unicode string (in hexadecimal)
X'....'	Byte array (in hexadecimal)
FILEBLOB=<filename>	A binary large object (BLOB) retrieved from the named file
FILECLOB=<filename>	A character large object (CLOB) retrieved from the named file
SAVEDPARM=<number>	A parameter from a previous CALL statement
SQLARRAY[TYPE:e1:e2:...]	A JAVA.SQL.ARRAY type: Supported types are: `String:BigDecimal:Time:Blob:Clob:int:short:long:float:double:byteArray`

Here is an example of using some of the various types.

```
>!PREPARE select cast(? AS VARGRAPHIC(40) CCSID 1200) from sysibm.sysdummy1
>!SETPARM 1,UX'0233'
>!EXECUTEQUERY
00001
U'0233'

>!PREPARE select cast(? AS VARBINARY(10)) from sysibm.sysdummy1
>!SETPARM 1,X'aa'
>!EXECUTEQUERY
00001
aa

>!PREPARE select cast(? AS BLOB),cast(? AS CLOB) from sysibm.sysdummy1
>!PREPARE select cast(? AS BLOB),cast(? AS CLOB) from sysibm.sysdummy1
>!SETPARM 1,FILEBLOB=file.txt
>!SETPARM 2,FILECLOB=file.txt
>!EXECUTEQUERY
00001,00002
5468697320697320612066696c650a, This is a file
```

As shown above, non-printable Unicode characters are shown using the **U**' notation. Any binary data is shown as a hexadecimal value.

Using reflection

A unique feature of *jdbcClient* that appeals to JDBC developers is the ability to use reflection to call Java methods. With the use of reflection, calls to Java methods do not need to come from compiled code, but are dynamically resolved. *jdbcClient* has three main commands that use reflection. The `!CALLMETHOD` command is used to call either static or instance methods. The `!SETVAR` command is used to set a variable with the result of a method call. *jdbcClient* maintains the list of variables that refer to Java objects. The `!SETNEWVAR` command is used to call a constructor to create a new object. Here are examples of how these calls can be used.

```
>!CALLMETHOD java.lang.System.getProperty(java.version)
Call returned 1.6.0
>!CALLMETHOD CON.getCatalog()
Call returned MYSYSTEM
>!SETVAR DMD=CON.getMetaData()
DMD=MYSYSTEM
>!SETVAR DRS=DMD.getTables(null,null,SAMPLE,null)
DRS=CRSR0008
>!CALLMETHOD com.ibm.as400.access.jdbcClient.Main.dispResultSet(DRS)
TABLE_CAT,TABLE_SCHEM,TABLE_NAME,TABLE_TYPE,REMARKS,TYPE_CAT,TYPE_SCHEM,TYPE_NAME,
        SELF_REFERENCING_COL_NAME,REF_GENERATION
MYSYSTEM,EBERHARD,SAMPLE,TABLE,null,null,null,null,null,null Call returned null
>!SETNEWVAR MYINT=java.lang.Integer(10)
MYINT=10
```

In the first call, the `System.getProperty` static method is called to obtain the `java.version` property. In the second call, the `getCatalog()` method on the existing **CON** object is called. The `!SETVAR` command is used to obtain a `DatabaseMetaData` object from the `CON` object and then obtain a `ResultSet` using a call to `getTables()`. To display the result set, the static method `dispResultSet` of the `jdbcClient.Main` class is used. The last call shows the creation of a `java.lang.Integer` object.

To assist with the use of objects assigned to variables, *jdbcClient* has the `!SHOWVARMETHODS` command. When called without any parameters, this command shows the variables that are defined. Here is an example of using this command.

```
>!SHOWVARMETHODS
Could not find variable
Valid variables are the following
DMD
STMT
MAIN
DRS
CON
```

When used in conjunction with a variable, the `!SHOWVARMETHODS` command shows the methods that can be invoked on an object. For example, here is a sample of the methods that can be invoked on the `CON` JDBC connection object.

```
>!SHOWVARMETHODS CON
boolean equals(java.lang.Object)
int hashCode()
java.lang.String toString()
void clearWarnings()
void close()
void commit()
java.sql.Statement createStatement()
java.sql.Statement createStatement(int,int)
java.sql.Statement createStatement(int,int,int)
boolean getAutoCommit()
```

The use of reflection is not limited to JDBC objects. You can use *jdbcClient* to experiment with all kinds of Java objects. When working with a Java class for the first time, this is useful for experimenting with the behavior of the Java class without needing to write and compile a Java program.

Additional features

jdbcClient has a variety of other features available. To see what can be done, just use the *jdbcClient* `!HELP` command. These features include the ability to run a command in a separate thread, the ability to repeat a command for a specified number of times, commands to view the history of the entered commands, and commands to control the formatting of the output data.

Summary

As you have seen, the JTOpen command line *jdbcClient* enables connecting to any database supported by a JDBC driver. More importantly, *jdbcClient* provides an easy way to run SQL statements through JDBC. It also provides support for prepared statements and parameters of various data types. Finally, it exposes Java objects through reflection, allowing Java methods to be dynamically called. *jdbcClient* provides a quick and easy way to work with JDBC drivers and the databases they connect to, thus becoming an irreplaceable tool for Java JDBC developers.

Resources

- JTOpen website
- DB2 for i forum
- Java reflection documentation