Connection Pooling – Part 1

ODBC/JCBC Pooling

By Jeff Ash

Imagine you want to interview three different people, asking each person questions as they occur to you. One approach would be to call Person A (dial the number, establish the connection, greet your interviewee, ask your question, say goodbye, and hang up), then repeat the process for Person B, and finally for Person C. Suddenly, you remember another question you wanted to ask Person B. Each time you want to ask a new question, you must repeat the whole process of re-establishing a connection – even if the question itself only takes a few seconds to ask and receive an answer.

A more efficient approach would be to phone Person A and place the call on hold, and then repeat this process for Person B and Person C. Every time you want to ask a question, just select Line 1, Line 2, or Line 3 on your phone, ask the question, and get an immediate answer. This approach is much faster, especially if you ask several questions of each of the three people, because you only incurred the overhead of calling and establishing a connection once for each person. (We are, of course, assuming that you are a very important person, and that these people have nothing better to do than sit at their phones waiting for you to ask them questions!)

This article discusses the pooling options built into the BBj® ODBC and JDBC drivers, how to use them, and some real world examples.

How Pooling Works

Pooling is similar to that theoretical multiline phone conversation, except with computers that do not mind hanging around all day in case you want to ask a question. In the context of the ODBC and JDBC drivers, a pool is simply a group of open connections that are available for immediate use. While it is possible for an application to manage a connection pool, it can be complicated and, since pooling is useful to many applications, it makes sense to embed it directly in the drivers. The BASIS ODBC/JDBC drivers have supported connection pooling since BBj 4.01. The connection pool is completely transparent to the application. That is to say that the application developer does not need to do anything special to use pooling; the driver manages everything internally. As a result, applications that do not already use connection pooling can benefit from it by simply adjusting their connection strings or data source definitions.

The connection pool always starts out empty, that is, it contains no open connections. When an application requests a connection from the driver, the driver first looks in its pool to see if there are any available connections. If there is a connection available that matches the one requested, the pool returns that connection instead of opening a new one. If there are no connections available that match the request, the driver opens a new connection and hands it off to the application. When the application "closes" the connection, the ODBC/JDBC driver actually keeps the connection open and checks it into the pool, just in case the application needs it again in the future. The connection remains in the pool for some predetermined time (the BASIS ODBC/JDBC default is 10 seconds), or until the application checks it back out or the application terminates.

Example

Suppose an application needs to open a connection, execute a statement, and then immediately close the connection; and the application does this a large number of times in succession. If the application does not enable pooling, it would have to open a completely new connection each time. While this might not be a problem if the application does this only a few times, it can be a problem if it needs to do this several hundred or several thousand times in a row. With pooling enabled, the connection is not closed each time but is simply set aside for later use. Then, when the application needs a new connection, it reuses the earlier one. If the application performs this operation several thousand times, it keeps getting a handle to the existing connection instead of creating an entirely new connection each time. The result is that performance is 2-3 times faster when using pooling.

How to Configure Pooling

By default, the JDBC and ODBC drivers automatically support pooling. The default amount of time that connections remain in the pool is 10 seconds.

ODBC Driver

To configure the pooling settings for an ODBC data source, open the DSN configuration dialogue for the desired data source through the ODBC administrator.

continued...

Jeff Ash
Software Engineer

Locate the setting called "Connections Remain in Pool." Set this value to the number of seconds that connections should remain in the pool. To disable pooling completely, set this value to zero. If the application uses a connection string instead of a DSN name, the name of the property to set is "POOLREMAIN." Here is a sample connection string setting the pooling option to 60 connections:

```
ODBC;Driver={BBj ODBC Driver};Server=LOCALHOST;Port=2001;Database=ChileCompany;POOLREMAIN=60
```

JDBC Driver

To configure the pooling settings for a JDBC connection, ad the "POOLREMAIN" property to the JDBC connection string. For example,

```
jdbc:basis:LOCALHOST?DATABASE=ChileCompany&POOLREMAIN=60
```

Summary

Since the BASIS data access drivers manage the pooling automatically, it is not necessary for application developers to write their own pooling mechanism. This can save the developer a lot of development time as well as maintenance time. For those applications that perform large number of operations involving opening and closing connections, pooling can have an enormously positive impact on the performance of the application.