DB2: JDBC on PRISM at CSE York

JDBC Application Programs

This will walk you through building a database application program in Java via JDBC on CSE's PRISM environment for accessing DB2.

Priming the Shell

First, you need to have a number of environment variables set in the shell so when you compile your java application program (APP), `javac` will know where to locate the `JDBC` library (that is, it is on the CLASSPATH). These environment variables are also necessary at runtime so the APP can resolve *which* database server it is to talk with.

Once you have a window (and, thus, shell) open on a PRISM machine (e.g., *red.cs.yorku.ca*), you can do this by *sourcing* the following script:

```
% source ~db2leduc/cshrc.runtime
```

That is just an ASCII file, so feel free to read it if you want to see the set up.

The Connection

In your APP, you will need to first establish the appropriate driver with the `DriverManager`. This is specific for the database system (or other data source) your APP will be talking to via the JDBC API. E.g.,

```
import java.net.*;
import java.sql.*;
...
// Register the driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
```

Next, you want to establish a connection with a particular database (served by the particular database server "instance"). This is really the same thing as when you say `db2 connect to c3421m` in the shell.

```
private Connection conDB;   // Connection to the database system.
private String url;         // URL: Which database?
    ...
url = "jdbc:db2:c3421m";    // URL: This database.
conDB = DriverManager.getConnection(url);
```

This connection convention can be used to establish a connection to a database server and database that is on a remote system. In this case, we are connecting to a local database system. Every machine on PRISM is a client to PRISM's DB2 system.

The `getConnection` method comes in a second flavour with three arguments, so one could specify an account name and password:

```
    conDB = DriverManager.getConnection(url, userid, passwd);
```

You do not need that here though. You are connecting locally, and PRISM's DB2 server is set up to accept PRISM's account authenication. So DB2 assumes the account to be the same as the PRISM account excuting the APP.

When the APP is done, it is good programming practise to close the connection.

```
    conDB.close();
```

Of course, the connection does automatically get closed when the APP process shuts down.

For each of these calls, one needs to encase them in a `try...catch` block (or further throw the exceptions yourself for your caller to worry about, which is a bit more rude). Each can throw a `COM.ibm.db2.jdbc.DB2Exception`. Best for your APP writing would be to catch any such exception, print out some meaningful message about where in the process the failure occured, and then exit. There is not much really you can do (at this point) if the database system is refusing to cooperate.

Registering the driver can result in a number of exception types. Either just catch the generic `Exception` for it, or see the example for more.

Queries: "Talking" to the Database

In a JDBC application program, you get the information you need from the database, and set the information you need to in the database, via SQL. The SQL statements are composed in Java strings, *prepared*, and *executed*. A *cursor* is used to walk through the results (if needed) of an executed SQL query. E.g.,

```
        String            queryText = "";     // The SQL text.
        PreparedStatement querySt   = null;   // The query handle.
        ResultSet         answers   = null;   // A cursor.
```

Design a query:

```
        queryText =
            "SELECT COUNT(*) as #custs"
          + "    FROM yrb_customer";
```

Prepare the query:

```
        querySt = conDB.prepareStatement(queryText);
```

At this point the database system parses the query, and builds an executable query plan for it. The query has not been executed yet, though. A query *handle* object is returned (`querySt` here) which we use to execute the query.

To execute, we can say

```
        answers = querySt.executeQuery();
```

The `executeQuery` method returns a cursor object (`answers` here), which is of type `ResultSet` in JDBC speak.

If there are no tuples being returned (e.g., this is an update "query"), we could execute the SQL statement instead with the method `updateQuery()`. This returns an `int` that reports how many rows in the database were affected.

In the above case, we expect just a single answer tuple to be found.

```
answers.next();
int num_of_customers = answers.getInt("#custs");
System.out.print("There are ");
System.out.print(num_of_customers);
System.out.println(" number of customers.");
```

Of course, as good programmers, we should check whether there is an answer tuple! If there are no customers, the query would have resulted in the empty table.

```
if (answers.next()) {
    int num_of_customers = answers.getInt("#custs");
    System.out.print("There are ");
    System.out.print(num_of_customers);
    System.out.println(" number of customers.");
} else {
    System.out.println("There are no customers.");
}
```

For each of these calls, we might get an `SQLException` thrown. (The `COM.ibm.db2.jdbc.DB2Exception` is effectively a superclass of this, so it is caught when we catch a `SQLException` in these cases.) Usually this means that something went wrong with respect to the SQL statement in question. Each of the calls above should be encased in a `try`...`catch` block to catch such exceptions (or properly thrown along).

When we are done with the cursor, we should close it.

```
// Close the cursor.
answers.close();
```

Likewise, when we are done with a query handle, we ought to close it too.

```
// We're done with the handle.
querySt.close();
```

An Example

The following is an example. It reports the total sales for a specified customer from the YRB database.

CustTotal.java

```
/*============================================================================
CustTotal: A JDBC APP to list total sales for a customer from the YRB DB.

Parke Godfrey
2013 March 26 [revised]
2004 March    [original]
============================================================================*/

import java.util.*;
import java.net.*;
import java.text.*;
import java.lang.*;
import java.io.*;
import java.sql.*;

```

```
/*============================================================================
CLASS CustTotal
============================================================================*/

public class CustTotal {
    private Connection conDB;   // Connection to the database system.
    private String url;         // URL: Which database?

    private Integer custID;     // Who are we tallying?
    private String  custName;   // Name of that customer.

    // Constructor
    public CustTotal (String[] args) {
        // Set up the DB connection.
        try {
            // Register the driver with DriverManager.
            Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(0);
        } catch (InstantiationException e) {
            e.printStackTrace();
            System.exit(0);
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            System.exit(0);
        }

        // URL: Which database?
        url = "jdbc:db2:c3421m";

        // Initialize the connection.
        try {
            // Connect with a fall-thru id & password
            conDB = DriverManager.getConnection(url);
        } catch(SQLException e) {
            System.out.print("\nSQL: database connection error.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Let's have autocommit turned off.  No particular reason here.
        try {
            conDB.setAutoCommit(false);
        } catch(SQLException e) {
            System.out.print("\nFailed trying to turn autocommit off.\n");
            e.printStackTrace();
            System.exit(0);
        }

        // Who are we tallying?
        if (args.length != 1) {
            // Don't know what's wanted.  Bail.
            System.out.println("\nUsage: java CustTotal cust#");
            System.exit(0);
        } else {
            try {
                custID = new Integer(args[0]);
            } catch (NumberFormatException e) {
                System.out.println("\nUsage: java CustTotal cust#");
                System.out.println("Provide an INT for the cust#.");
                System.exit(0);
            }
        }

        // Is this custID for real?
        if (!customerCheck()) {
            System.out.print("There is no customer #");
            System.out.print(custID);
            System.out.println(" in the database.");
            System.out.println("Bye.");
            System.exit(0);
        }

        // Report total sales for this customer.
        reportSalesForCustomer();

        // Commit.  Okay, here nothing to commit really, but why not...
        try {
            conDB.commit();
        } catch(SQLException e) {
            System.out.print("\nFailed trying to commit.\n");
            e.printStackTrace();
            System.exit(0);
        }
        // Close the connection.
        try {
            conDB.close();
        } catch(SQLException e) {
            System.out.print("\nFailed trying to close the connection.\n");
            e.printStackTrace();
            System.exit(0);
        }

    }

    public boolean customerCheck() {
        String            queryText = "";     // The SQL text.
        PreparedStatement querySt   = null;   // The query handle.
        ResultSet         answers   = null;   // A cursor.

        boolean           inDB      = false;  // Return.

        queryText =
            "SELECT name       "
          + "FROM yrb_customer "
          + "WHERE cid = ?     ";

        // Prepare the query.
        try {
            querySt = conDB.prepareStatement(queryText);
        } catch(SQLException e) {
            System.out.println("SQL#1 failed in prepare");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Execute the query.
        try {
            querySt.setInt(1, custID.intValue());
            answers = querySt.executeQuery();
        } catch(SQLException e) {
            System.out.println("SQL#1 failed in execute");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Any answer?
        try {
            if (answers.next()) {
                inDB = true;
                custName = answers.getString("name");
            } else {
                inDB = false;
                custName = null;
            }
        } catch(SQLException e) {
            System.out.println("SQL#1 failed in cursor.");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Close the cursor.
        try {
            answers.close();
        } catch(SQLException e) {
            System.out.print("SQL#1 failed closing cursor.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

        // We're done with the handle.
        try {
            querySt.close();
        } catch(SQLException e) {
            System.out.print("SQL#1 failed closing the handle.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

        return inDB;
    }

    public void reportSalesForCustomer() {
        String            queryText = "";     // The SQL text.
        PreparedStatement querySt   = null;   // The query handle.
        ResultSet         answers   = null;   // A cursor.

        queryText =
            "SELECT SUM(P.qnty * O.price) as total          "
          + "    FROM yrb_purchase P, yrb_offer O           "
          + "    WHERE P.cid = ?                            "
          + "      AND P.title = O.title AND P.year = O.year";

        // Prepare the query.
        try {
            querySt = conDB.prepareStatement(queryText);
        } catch(SQLException e) {
            System.out.println("SQL#2 failed in prepare");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Execute the query.
        try {
            querySt.setInt(1, custID.intValue());
            answers = querySt.executeQuery();
        } catch(SQLException e) {
            System.out.println("SQL#2 failed in execute");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Variables to hold the column value(s).
        float  sales;

        DecimalFormat df = new DecimalFormat("####0.00");

        // Walk through the results and present them.
        try {
            System.out.print("#");
            System.out.print(custID);
            System.out.print(" (" + custName + ") has spent $");
            if (answers.next()) {
                sales = answers.getFloat("total");
                System.out.print(df.format(sales));
            } else {
                System.out.print(df.format(0));
            }
            System.out.println(".");
        } catch(SQLException e) {
            System.out.println("SQL#2 failed in cursor.");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Close the cursor.
        try {
            answers.close();
        } catch(SQLException e) {
            System.out.print("SQL#2 failed closing cursor.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

        // We're done with the handle.
        try {
            querySt.close();
        } catch(SQLException e) {
            System.out.print("SQL#2 failed closing the handle.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

    }

    public static void main(String[] args) {
        CustTotal ct = new CustTotal(args);
    }
}
```

parke godfrey