Extra Lab: JDBC Programming

In this lab, you will write code that uses JDBC to access a relational database. To make the lab as simple as possible, you will use the free, open-source Hypersonic SQL database.

Objectives:

- To use the JDBC 1.x API to execute a query
- To use the JDBC 2.x API and PreparedStatements to execute an insert

Part 1: JDBC Query

In this part, you will write a program that uses the JDBC 1.x to execute a JDBC query and display the results.

Steps:

_1. Start the HypersonicSQL database server running:

   a. Open a command-prompt (click the Windows Start menu, and in the "Search programs" box, enter **cmd**).

   b. Change the current directory:

   ```
   cd \Users\username\javaclass\lib
   ```

   Substitute your actual username.

   c. Start the database server:

   ```
   java -cp hsqldb.jar org.hsqldb.Server -database jdbclab
   ```

   The server should display several lines of startup messages. Leave this window up and the server running for the duration of the lab.

_2. Start the Hypersonic SQL administrative console, then create and populate a database table:

   a. Open another command-prompt window and change to the same **{Lab Installation Directory}/lib** directory as in the last step.

   b. Start the HSQLDB administrative console:

   ```
   java -cp hsqldb.jar org.hsqldb.util.DatabaseManager
   ```

   c. In the admin console, CAREFULLY set the *Type* to **HSQL Database Engine Server** (be sure to choose this type).

   Without changing them, write down the *Driver*, *URL*, *User* and *Password* values, then press OK.

   d. Create a database table to hold data about an insurance policy. In the large text-input area between the Clear and Execute buttons, enter:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
CREATE TABLE lifepolicy (
  policyNum INT NOT NULL IDENTITY,
  holder CHAR(30),
  amount DOUBLE,
  createdate DATE,
  PRIMARY KEY (policyNum)
  );
```

Press Execute to run the command.

From the menu, choose *View* - *Refresh Tree* - you should see the new table in the left-hand tree.

e. Insert a few rows into the table. Press the Clear button, then enter:

```
INSERT INTO lifepolicy (holder, amount, createdate)
  VALUES ('Paul Westerberg', 40000, '2009-04-06');

INSERT INTO lifepolicy (holder, amount, createdate)
  VALUES ('Slim Dunlap', 30000, '2009-03-12');

INSERT INTO lifepolicy (holder, amount, createdate)
  VALUES ('Peter Tosh', 25000, '2009-10-12');
```

Press Execute to run the command.

f. Examine the table. Press Clear, then execute:

```
SELECT * FROM lifepolicy
```

You should see three rows of data.

g. You can leave the admin console running for later parts of the lab. Be sure to leave the server window running.

_3. In Eclipse, create a Java project named **ExtraJDBC**.

_4. Your project will use the JDBC API to communicate with the database. JDBC requires that you use a *JDBC driver* library -- for this program, we need to ensure that the JDBC driver is accessible to our program. Follow these steps:

a. In the *Package Explorer*, right-click on the new project and choose *Build Path* - *Add External Archives*.

b. Navigate to **{Lab Installation Directory/lib**. If your computer is running Windows XP and has the standard lab setup, this directory is:

c. Choose the *hsqldb.jar* file, then press Finish.

_5. Right-click on the project and choose *New* - *Class* to create a Java class named **QueryPolicies** in the *part1* package with a *main* method.

_6. Complete the program so that it:

a. Connects to the database using the JDBC 1.x approach. The connection details are:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
Driver class:  org.hsqldb.jdbcDriver
JDBC URL:      jdbc:hsqldb:hsql://localhost/
User name:     sa
Password:      ------ no password -- i.e. an empty string -------
```

Your *getConnection()* call might look something like:

```
conn = DriverManager.getConnection(
   "jdbc:hsqldb:hsql://localhost/", "sa", "");
```

b. Uses the connection to create a *java.sql.Statement*, then executes a query to retrieve all of the rows, all columns from the *lifepolicy* table into a *java.sql.ResultSet*.

c. Iterates the ResultSet, displaying each of the columns to the console. You can treat the "createdate" column as a Java string.

d. Use a *try-catch-finally* block to handle exceptions. Close the connection in the *finally* block.

_7. Right-clock on QueryPolicies.java and choose *Run As - Java Application*. You should see the insurance policy data from the table.

Part 2: Executing a JDBC Insert

In this part, you will execute a JDBC insert using a prepared statement. You will also use the JDBC 2.x approach to connect, which requires a DataSource.

Steps:

_1. Download **simple-jndi-0.11.4.1.jar** from the class home page into the *javaclass/lib* folder.

_2. Your project will use the JNDI API access a data source. Since your program will not run on a full JEE application server, you will use a simple, standalone JNDI provider. Follow these steps in Eclipse to configure it:

a. Right-click on the project and create a new folder named **jndi-config**.

b. Right-click on the jndi-config directory and create a new file named **PolicyDS.properties**.

Enter the following into the PolicyDS.properties file making sure to get the URL with a trailing slash:

```
type=javax.sql.DataSource
driver=org.hsqldb.jdbcDriver
url=jdbc:hsqldb:hsql://localhost/
user=sa
password=
```

This configures a JNDI resource named *PolicyDS* that refers to the same database you used in the first part of lab.

c. Right-click on the project and choose *Build Path - Add External Achives*, then browse to **{Lab Installation Directory}/simple-jndi** and select the **simple-jndi-0.11.4.1.jar** file.

d. Right-click on the project's *src* folder and create a file named **jndi.properties**. Enter the following into this file:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
java.naming.factory.initial=org.osjava.sj.SimpleContextFactory

org.osjava.sj.root=jndi-config/
```

This configures JNDI to use the Simple JNDI library and configures its configuration directory (where you put the PolicyDS.properties file that defined the data source).

Note: Please ensure that there are no trailing spaces at the end of each of the lines in the properties file.

_3. Create a new package named **part2**, and in it, a Java class named **InsertPolicy** with a *main* method. Then complete *main*:

a. Create an InitialContext.

b. Look up the DataSource with JNDI name **PolicyDS**.

c. Create a Connection from the DataSource.

d. Create a PreparedStatement from the Connection. The SQL should execute an INSERT into the *lifepolicy* table.

Your SQL should have three placeholder question-marks for the *holder*, *amount* and *createdate* columns. Note that you don't need to supply the *policyNum* since it's an auto-generated column.

e. Initialize the PreparedStatement's placeholders and execute it using values of your choice. Note that you can treat the *createdate* field as a string of the form yyyy-mm-dd and that you don't need to provide a value for the *policyNum* field since the database auto-generates values for that column.

You should output the count of rows inserted to the console.

f. Use a *try-catch-finally* block to handle exceptions and close the context and the connection in the *finally* block.

_4. Run and test. You should see your "row count" message or an exception message that will help you correct the problem.

_5. Once you see your "success" message, you can verify that the row was inserted by running your program from the last part of lab or via the HSQLDB admin console.

_6. You can close the HSQLDB admin console, then press Ctrl+C on the HSQLDB server's command-prompt window to kill the server.

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.