

Learning Objectives

- Implement a JDBC application that inserts records into a table, queries a table, and presents multi-line results.

Functional Requirements

Please Note: To complete this lab you should first finish lab 04 and Assignment #2.

The goal is write a JDBC application that allows a person to:

1. List all the state codes, longitudes, and latitudes for a particular city
2. Insert information for a new location, including:
zip, state code, city name, longitude, and latitude

To accomplish this, you will need to query the table zip.

Approach

Take the web application that you developed in lab04 and add a new class called **ZipCode**. It is suggested that you add the following methods to this class:

```
boolean addNewLocation(  
    String zip, String stateCode, String cityName, double longitude, double latitude)  
    This function should add a record to the table zip. If an error occurs, you should detect it  
    and return false; otherwise, return true.
```

What should you do if the zip code already exists?

```
int getZipCodeInfo(String cityName);  
    This function runs the query and returns the number of items found.
```

```
String getDataField(int recordID, String fieldName);  
    This function returns the data associated with the record ID and the field name for the  
    previously run query.
```

Getting started

1. Study the following class files on the course website:
 DbBaseConnect.java
 ZipCode.java
2. Provide implementations for the above methods.
3. Create a jsp page to test your methods.

What to submit?

On a lab 05 webpage, please:

1. Post a link to a jsp page that demonstrates the above capabilities.

C:\Documents and Settings\dhendry\My
Documents_Code\java_apps_working\INFO340\web\examples\JDBC\zipcode.jsp

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <@page contentType="text/html"%>
3 <@page pageEncoding="UTF-8"%>
4 <!-- zipcode.jsp -->
5 <!-- Generates an example document representing the results of an SQL query -->
6 <!-- As usual, import the classes in util (e.g., Date, String, etc.) -->
7 <@page import="java.util.*"%>
8
9 <!-- This class creates a db connection -->
10 <@page import="uw.ischool.info340.labs.DbBaseConnect"%>
11
12 <!-- This class does the work -- it extends the class DbBaseConnect -->
13 <@page import="uw.ischool.info340.labs.ZipCode"%>
14
15 <xml comment="An example document generated by jsp">
16 <%
17 /* Initialize the class that manages the database */
18 ZipCode db = new ZipCode();
19 db.init();
20
21 /* Get the results in XML and put it into the output stream */
22 String t = db.getZipCodesForCityAsXML("Boston");
23 %>
24
25 <%=t%>
26 </xml>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xml comment="An example document generated by jsp">

<results num='17' city='Boston'>
  <item>
    <zip>2210</zip>
    <scode>MA</scode>
    <city>BOSTON</city>
    <longitude>71.0465</longitude>
    <latitude>42.3489</latitude>
  </item>
  <item>
    <zip>22713</zip>
    <scode>VA</scode>
    <city>BOSTON</city>
    <longitude>78.1423</longitude>
    <latitude>38.5382</latitude>
  </item>
  <item>
    <zip>14025</zip>
    <scode>NY</scode>
    <city>BOSTON</city>
    <longitude>78.7391</longitude>
    <latitude>42.6314</latitude>
  </item>
  <item>
    <zip>2199</zip>
    <scode>MA</scode>
    <city>BOSTON</city>
    <longitude>71.0825</longitude>
    <latitude>42.3479</latitude>
  </item>
</results>
</xml>
```

C:\Documents and Settings\dhendry\My

Documents_Code\java_apps_working\INFO340\src\uw\ischool\info340\labs\DbBaseConnect.java

```
1 /*
2  * DbBaseConnect.java
3  */
4 package uw.ischool.info340.labs;
5
6 import javax.naming.*;
7 import javax.sql.*;
8 import java.sql.*;
9
10 public class DbBaseConnect
11 {
12     /* CHECK that this connection string is right
13      *   a) dhendry == is the name of the database
14      *   b) 5432 is the port number -- it is optional
15      *   c) linux.ischool.washington.edu -- is the name of the host
16      */
17     private static String cDB_URL_STRING =
18         "jdbc:postgresql://linux.ischool.washington.edu:5432/dhendry";
19
20     /*
21      * NOTE: Check that the user ID and password are correct
22      */
23     private static String cUserID = "dhendry";
24     private static String cUserPassword = "bluemonster";
25
26     /* This is the name of the JDBC driver for using postgresql */
27     private static String cDRIVER_STRING = "org.postgresql.Driver";
28
29     /* Holds the connection to database -- created by init() */
30     /* By default, transactions are auto-committed */
31     /* NOTE: Subclasses will need to use this BUT not clients */
32     protected Connection conn = null;
33
34     /*
35      * Used to hold error messages
36      */
37     protected String debugString;
38
39     /*
40      * Returns of a string containing basic information about the connection
41      */
42     private String _getConnInfo()
43     {
44         String t = "[";
45         t += "Driver: " + cDRIVER_STRING + ";";
46         t += "DB URL: " + cDB_URL_STRING + ";";
47         t += "UserID: " + cUserID + ";";
48         t += "Password: " + cUserPassword + ";";
49         t += "]<p>" + debugString;
50         return t;
51     }
52
53     /**
54      * Returns information about the connection
55      */
56     public String getConnectInfo()
57     {
58         if (conn != null)
59             return "Connection up! " + _getConnInfo();
60         else
61             return "Connection DOWN " + _getConnInfo();
62     }
63 }
```

```
64  /**
65   * Return true if the connection has been initialized
66   */
67  public boolean isConnected() {
68      return (conn == null) ? false : true;
69  }
70
71  /*
72   * Subclasses MUST call this to initialize the connection
73   */
74  public void init() {
75
76      // Already initialized
77      if (conn != null) {
78          return;
79      }
80
81      try {
82          /* This statement implicitly loads the driver */
83          Class.forName(cDRIVER_STRING);
84
85          /* Now, attempt to create a connection */
86          conn = DriverManager.getConnection(cDB_URL_STRING, cUserID, cUserPassword);
87
88          if (conn == null)
89              throw new Exception("Could not connect to " + getConnectInfo());
90      }
91      catch(Exception e) {
92          debugString = e.toString();
93          e.printStackTrace();
94      }
95  }
96 }
97
```

C:\Documents and Settings\dhendry\My
Documents_Code\java_apps_working\INFO340\src\uw\ischool\info340\labs\ZipCode.java

```
1 /*
2  * ZipCode.java
3  */
4
5 package uw.ischool.info340.labs;
6
7 import javax.naming.*;
8 import javax.sql.*;
9 import java.sql.*;
10
11 public class ZipCode extends DbBaseConnect {
12
13     public ZipCode() {
14         /*
15          * Call init() in the superclass to initialize the connection
16          */
17         init();
18     }
19
20     /*
21      * Adds a new location of the zip table
22      */
23     public boolean addNewLocation(
24         String zip,
25         String stateCode,
26         String cityName,
27         double longitude,
28         double latitude)
29     {
30         // To DO
31         return false;
32     }
33
34     /*
35      * Queries the zip table for all rows that match the cityName
36      */
37     public int queryZipCodeInfo(String cityName) {
38         // TO DO
39         return 0;
40     }
41
42     /*
43      * Gets the value of a particular row # and field name
44      */
45     public String getDataField(int recordID, String fieldName) {
46         // TO DO
47         return "";
48     }
49
50     /*
51      * Returns an XML document containing all state codes, lognitudes and
52      * latitudes for a particulare city
53      */
54     public String getZipCodesForCityAsXML(String cityName) {
55         String theQuery = "";
56         try {
57             /* Build up query */
58             String p1 = "select zcode, scode, city, longitude, latitude from zip ";
59             String p2 = "where city=upper('" + cityName + "')";
60             theQuery = p1 + p2;
61
62             /* Execute query -- the results are in r */
63             Statement s = conn.createStatement();
```

```

64         ResultSet r = s.executeQuery(theQuery);
65
66         /*
67          * We will keep track of the number of items processed
68          */
69         int count=0;
70
71         /*
72          * We will put the XML source into this variable
73          */
74         String t="";
75
76         /*
77          * A 'cursor' is used to move through results -- initially,
78          * the cursor is set BEFORE the first row in the result set.
79          * While next() returns TRUE we have a row to process
80          */
81         while(r.next()) {
82             /*
83              * Query generated a result -- now extract data by Name & Index
84              */
85             String zcode    = r.getString("zcode");
86             String scode    = r.getString("scode");        // by column name
87             String city     = r.getString("city");
88             double longn    = r.getDouble(4);              // by index -- 4th column
89             double latit    = r.getDouble("latitude");
90
91             /*
92              * Create XML fragment and concatenate it to XML string
93              */
94             t += "    <item>\n";
95             t += "        <zip>"          + zcode + "</zip>\n";
96             t += "        <scode>"       + scode + "</scode>\n";
97             t += "        <city>"        + city + "</city>\n";
98             t += "        <longitudo>"   + longn + "</longitudo>\n";
99             t += "        <latitudo>"    + latit + "</latitudo>\n";
100            t += "    </item>\n";
101
102            count++;
103        }
104
105        /*
106         * Check for results/no-results and build xml string
107         */
108        if (count > 0) {
109            t = "<results num='" + count + "' city='" + cityName + "'>\n" + t;
110            t += "</results>\n";
111        } else {
112            t = "<noresults city='" + cityName + "'>";
113            t += "    <query>" + theQuery + "</query>";
114            t += "    <reason>Unknown city</reason>";
115            t += "</noresults>";
116        }
117        /*
118         * Return the XML document of results
119         */
120        return t;
121
122    } catch(Exception e) {
123        debugString = e.toString() + "Query: ||" + theQuery + "||";
124        e.printStackTrace();
125    }
126
127    /*
128     * Return an ERROR tag -- if there's an exception send this back
129     */
130    return "<ERROR function='getZipCodesForCityAsXML'>"
131           + debugString

```

```
132         + "</ERROR>";  
133     }  
134 }  
135
```