

## 1 Introduction

In this lab you are going to learn how to use a database directly and integrated with applications written in Java and PHP. You are going to connect and manipulate data in a PostgreSQL database that we have set up for you.

This lab consists of two distinct parts. First, you are going to connect directly to the database and execute SQL queries in your shell. Afterwards, you are going to write a simple application with front end that allows the user to enter search criteria, and back end that connects to the database to execute search queries and passes the results to the front end. For the second part we provide you with two versions of the front end - a Swing-based GUI and an HTML web page. You can choose whether to use Java or PHP to implement the back end of a desktop or web-based application. We recommend that if you already have experience interacting with a SQL database using either Java or PHP, that you try out whichever one you know less.

## 2 Part I: Direct Interaction with a Database

To connect to the database type in your shell:

```
psql -h bridget cs32lab
```

-h stands for the host name;  
cs32lab is the database name.

Create table:

```
CREATE TABLE $USER$ (student_id integer,  
                        name          text,  
                        box_n         integer,  
                        year          integer);
```

Here (and in all that follows) \$USER\$ should be your user-id (or someother unique name).

You have to ensure that the data entered in the table is valid. For example, the student id column should contain unique id number for each student. To set this constraint on the student id column:

```
ALTER TABLE $USER$ ADD CONSTRAINT constr1 UNIQUE (student_id);
```

You could also define the constraint when you created the table by adding the word 'unique' after the type of the column:

```
CREATE TABLE $USER$ (student_id integer unique,  
                        name          text,  
                        box_n         integer,  
                        year          integer);
```

To make sure you created the table correctly and see its definition type in:

```
\d $USER$
```

To insert data in the table:

```
INSERT INTO $USER$ VALUES (1, 'jsmith', 1000, 2008);
```

To view the data:

```
SELECT * from $USER$;
```

Repeat the above INSERT statement to insert information for more students. For example:

```
INSERT INTO $USER$ VALUES (2, 'aronald', 1020, 2010);  
INSERT INTO $USER$ VALUES (3, 'dfred', 1030, 2009);  
INSERT INTO $USER$ VALUES (2, 'fmillur', 1010, 2008);  
INSERT INTO $USER$ VALUES (4, 'jsmith', 1015, 2008);
```

### CHECKPOINT 1

Notice that it didn't allow you to insert two entries with the same student ID number.

Say for example you did a mistake when you inserted dfred's box number and you want to correct it. To update his information execute the following statement:

```
UPDATE $USER$ SET box_n = 1010 WHERE student_id = 3;
```

To view the change:

```
SELECT box_n from $USER$ WHERE student_id = 3;
```

To search for all \$USER\$ with name jsmith, graduating in 2008:

```
SELECT * from $USER$ WHERE year = 2008 AND name='jsmith';
```

### CHECKPOINT 2: Show to a TA that you can execute simple SELECT and UPDATE SQL queries.

To delete the table:

```
DROP TABLE $USER$;
```

## 3 Part II: Database Interaction in Java

For this part of the lab we provide you with a Swing-based GUI and stencil code in Java that you will have to complete to set up the back end of the application that communicates with the database. For this purpose Java provides the java.sql library. In particular you are going to use java.sql.Connection, java.sql.Statement, java.sql.DriverManager and java.sql.ResultSet. You can look at the javadocs for detailed description of these classes. You are also going to write simple SQL queries that you are going to embed and execute in the Java code. For further help with the SQL queries you can look at the PostgreSQL tutorial at <http://www.postgresql.org/docs/8.3/static/queries.html>.

You can copy the support code from /course/cs032/pub/labs/06\_db. To set up the project in Eclipse you have to import two external jar files: /course/cs032/lib/swing-layout.jar and /course/cs032/lib/postgresql.jar.

The methods that you will have to fill in are in the DBLabFormHandler.java class. It contains methods that handle opening and closing connection to the database, creating empty table, inserting data into the table and searching. These methods are called within the actionPerformed() methods associated with each button of the GUI. To give you a better idea how the GUI interacts with the back end, here's a small part of the DBLabGUI implementation:

```

private DBLabFormHandler my_handler;
// ...
jButton2.setText("Add");

jButton2.addActionListener(new java.awt.event.ActionListener() {
// This method is called when the Add button, called jButton2, is pressed.
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        String name, student_id, box_n;
        Object year;

        // Get the entered values from the GUI text field and combo box.
        name = jTextField2.getText();
        student_id = jTextField1.getText();
        box_n = jTextField3.getText();
        year = jComboBox1.getSelectedItem();

        // Call the insert method of the handler, passing it the data.
        my_handler.handleAddStudentInfo(name, student_id, box_n, year);
    }
});

```

## 4 How to connect to PostgreSQL Database

To connect to a database within your java code you need an URL, username and password. These information has already been set up for you in the DBLabFormHandler class. All you need to do to establish the connection is to write the following lines of code in the initDB() method within a try/catch block:

```

try {
    Class.forName("org.postgresql.Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
// The url, username and pass are given to you.
Connection connection = DriverManager.getConnection(url, username, pass);
Statement stmt = connection.createStatement();

```

Go ahead and complete the initDB() method. **CHECKPOINT 3**

## 5 How to execute SQL queries in Java

To execute an update to the database, i.e. SQL statement that does not return result set, simply call executeQuery() to the instance of class Statement, passing it a String for the query. For example:

```

private Statement stmt;
// ...
String sql = "INSERT INTO $USER$ VALUES (...);";
stmt.executeUpdate(sql);

```

To execute SELECT query and get the result data:

```

ResultSet rs;
String sql = "SELECT * from $USER$";
rs = stmt.executeQuery(sql);

```

```

// To print the results:
while ( rs.next() ) {
    int cur_id = rs.getInt("student_id");
    String cur_name = rs.getString("name");
    int cur_boxn = rs.getInt("box_n");
    int cur_year = rs.getInt("year");

    System.out.println("cur_name = " + cur_name);
    System.out.println("cur_student id = " + cur_id);
    System.out.println("cur_box number = " + cur_boxn);
    System.out.println("cur_year = " + cur_year);
}
rs.close();

```

Don't forget to call the close() method of the ResultSet once you are done.

To display the results in the GUI call the setModel() method of the JTable (created in the DBLabGUI class), which is going to hold the results. You have to pass the method a Vector that holds the data and a Vector with the column names. Here's how you can do that:

```

Vector<Vector> data = new Vector<Vector>(); // A 2D vector that holds vectors, representing the
rows.
Vector<String> columnNames = new Vector<String>(); // A vector with the column names
// ...
jtable.setModel(new javax.swing.table.DefaultTableModel(data, columnNames));

```

Go ahead and complete the handleAddStudentInfo() method. **CHECKPOINT 4**

Go ahead and complete the handleSearchStudentInfo() method. **CHECKPOINT 5**

## 6 How to Close the Database Connection

Simply call the close() method to the instances of Connection and Statement within a try/catch block:

```

Connection conn;
Statement stmt;
// ...
stmt.close();
conn.close();

```

Go ahead and complete the closeDB() method. **CHECKPOINT 6**