Data, I think, is one of the most powerful mechanisms for telling stories. I take a huge pile of data and I try to get it to tell stories. - Steven Levitt

**Introduction**

In this lab, you will add a SQLDriver class in the server package. This will enable the server to update the SQL database with information about the factory. You will then use this ability to add each product name that the server sends to the database.

**Part 1 – Initial set-up**

We will need to use a MySQL connector jar that is provided in order to communicate with the MySQL server. Place the mysql-connector-java-5.0.8-bin.jar into the eclipse project.

Now, add the jar to the build path.

Right-click and select ‘Build Path’ > Add to Build Path.
Now our project can use the packages provided in order to communicate with the MySQL server.
Part 2 – Creating a MySQL Driver

Make a new class in the Server package named MySQLDriver.

```java
public class MySQLDriver {
    public MySQLDriver() {
        try {
            new Driver();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

There are a number of classes named “Driver”. We need to make sure this Driver is a jdbc driver.

Make sure the correct import statement is used.

```java
import com.mysql.jdbc.Driver;
```

Or simply be explicit.

```java
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Also, manually add these import statements before continuing.
Part 3 – Creating a connection

Now that the driver is set-up, we can establish a connection with the Database.

Create a void connect() method in the MySQLDriver class.

```java
private Connection con;

public void connect() {
    try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/factory?user=root&password=root");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

For now, we are hardcoding this string. It is important though, to know what each part of the string means.

`Jdbc:mysql://localhost:3306` gives us the MySQL server we are connecting to. This doesn't have to be our local computer. Since the server is in-charge of the database, we want only the server that is running on the local machine to be able to access it, so this works well.

The ‘factory’ string tells the connection to work with the ‘factory’ schema in our database. Lastly, we send along a username and password to get access.

*Note: It is usually a bad idea to hardcode a username and password. It is better to read that info from a file that is not uploaded to source control. Each programmer on the team can then have their own username and password that corresponds to a user account with various security settings.*

Be sure to have a method to close the connection as well.

```java
public void stop() {
    try {con.close();} catch (SQLException e) {e.printStackTrace();}
}
```
Part 4 – Prepared Statements

When the FactoryServer reads in a file, we potentially have new products that should be added to the database. Let’s create a new method that checks if a name exists in the factoryorders table.

We need to make a MySQL command that will return the results of all products with a given name.

First, create a string that will do this.

```java
private final static String selectName = "SELECT * FROM FACTORYORDERS WHERE NAME=?";
```

This command will select all the entries from the table named factoryorders where the name is some string.

Now, we can create a method that will check if this statement returns any results. If it does, then we know it already exists in the table.

```java
public boolean doesExist(String productName) {
    try {
        PreparedStatement ps = con.prepareStatement(selectName);
        ps.setString(1, productName);
        ResultSet result = ps.executeQuery();
        while (result.next()) {
            FactoryServerGUI.addMessage("exists with count: " + result.getInt(2));
            return true;
        }
    } catch (SQLException e) { e.printStackTrace();
        FactoryServerGUI.addMessage("Unable to find product with name: " + productName);
        return false;
    }
}
```

We do this by creating a PreparedStatement with the string we prepared as a parameter. We can then set the missing parts of the string that were marked as '?' marks. Calling executeQuery() on the PreparedStatement returns a ResultSet. If this ResultSet has anything inside, we know a product with the name ‘productName’ exists somewhere in the table.

*Note: With a result, we can get its information by the index. We know the first index is a string (name), so we can get that by calling the getString(int) method. We also know the second index is an int (count), so we can get that by calling the getInt(int) method.*

We can check if the products are in the database or not right before we send the loaded factory to the connected clients. Hop over to FactoryServer.java.
Right now, we are only going to get unknown statements. We need to create another method that will add the product names to the table. Return to MySQLDriver.java

```java
private final static String addProduct = "INSERT INTO FACTORYORDERS(NAME, CREATED) VALUES(?,?)";
public void add(String productName) {
    try {
        PreparedStatement ps = con.prepareStatement(addProduct);
        ps.setString(1, productName);
        ps.setInt(2, 0);
        ps.executeUpdate();
        FactoryServerGUI.addressMessage("Adding product:"+productName+" to table with count 0");
    } catch (SQLException e) {e.printStackTrace();}
}
```

This time, our command does something a bit different. Instead of selecting from, we insert into the table factoryorders. We pass in the names of the variables to set, name, and created. We then add '?' marks to denote that these will be filled.

We set the first index, (name) to productName. Since it is a String, we use the setString(int,String) method. We then set the second index, (created) to 0.

**Note:** Since it makes sense for a products created count to be defaulted to 0, we could also set this option in MySQL workbench. That way, if the created column isn’t specified, it will default to 0.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>PK</th>
<th>NN</th>
<th>UQ</th>
<th>BIN</th>
<th>UN</th>
<th>ZF</th>
<th>A</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>VARCHAR(25)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>created</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'0'</td>
</tr>
</tbody>
</table>

Now, when a product isn’t found in the table, you can add it using the following code:
public static void sendFactory(Factory factory) {
    MySQLDriver mysql = new MySQLDriver();
    mysql.connect();
    for(Product p : factory.getProducts()) {
        if(!mysql.doesExist(p.getName())) {
            mysql.add(p.getName());
        }
    }
    mysql.stop();

    if (serverlistener != null) {
        serverlistener.sendFactory(factory);
    }
}
Part 5 – Viewing the updated results

Now when the server loads up a factory, it will add products to the database.

Open up MySQL Workbench and view the table.

You can call a ‘SELECT * FROM’ statement by pressing the highlighted button next to the table name.

This will display the entire table.
Expand on This

Currently, the client writes to a file a complete list of all the items produced. Your job is to send that file to the server, let the server parse it, and then write the completed items to a database.

Do not merely use the client to send the completed items list to the SQL database. In real life, this is an unsafe practice because the client could inject bad SQL commands that could corrupt/steal information. That is why we want you to send it to the server, have the server verify and parse the data, then write it to the database.

Here are some hints to get you started:

1. Right now, you can send a string using sendMessage() in the FactoryClientListener, and you should have some experience working with it from Lab 8. Maybe find a way to encode the file (or the file's information) into a string and send it. Or you can make an object output stream but we're just trying to make it easier.

2. Once it arrives, you will need to execute SQL command(s) to write to the table. You might want to look at the syntax for UPDATE, because if you try to re-use the addProduct() again after the table is created you will have a problem.

3. Don't worry about the corner case in case the user closes the client before the SQL database is updated. Assume that the user will (properly) keep the client connected to the server while the database is being updated. This isn't practical in real life but we are testing your ability to have a client send data to a database, not your ability to find and correct corner cases.

4. Don't know how to pass on the info from serverclientlistener to the SQL Driver? Make a new SQL driver in the ServerClientListener.

5. Yes, the FactoryClientListener is created in the FactoryClientGUI, find a way to pass it to FactorySimulation where the checker for ending the factory is. The other way around works too. Tomato tomato.

A great site for learning SQL commands: http://www.w3schools.com/sql/

You don’t have to learn every command, but it is a good thing to have a solid reference.