LS/EECS Building Ecommerce Applications

Lab 6: Student Information System R1.0

Objectives
-understand model view control programming model
-understand JDBC programming model
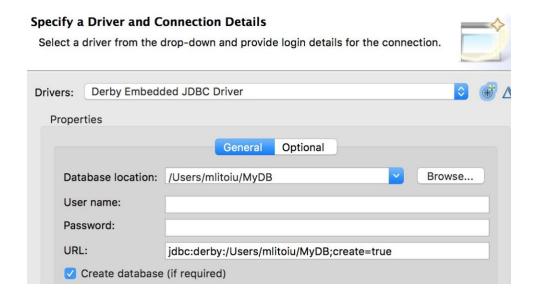-develop an end to end MVC application

The Project

- Create a new project named *SIS-1* (Student Information System, release #1).
- The project enables its client to generate reports that returns information from a student records database.
- The client can filter the students included in the report based on their names and GPAs.
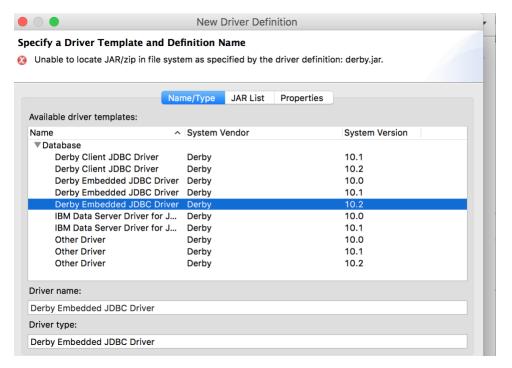
Requirement 1. Installing and Exploring the Database

A. Install Apache Derby Embedded Driver (embedded= the database engine runs within the same VM where JDBC runs). Use these instructions:

1. If you have not done it in Lab 1,
    a. Download and unzip Apache derby version 10.14.2.0 (lib version is enough)
       http://db.apache.org/derby/derby_downloads.html
    b. Unzip it Eclipse directory.
2. Create a Dynamic Web project, SIS-1
3. In Eclipse, switch into Database Development perspective
4. Select Database Connection->New Connection.
5. From the list of connections, select Derby....and then Next
       5.1. you should see something like this....

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: Derby Embedded JDBC Driver

Properties

General | Optional

Database location: /Users/mlitoiu/MyDB | Browse...

User name:

Password:

URL: jdbc:derby:/Users/mlitoiu/MyDB;create=true

☑ Create database (if required)

5.2 if you do not see "Derby Embedded JDBC Driver" at the top of the wizard, then click on the + sign, top right corner of the wizard.

5.3 in the next wizard page you should be able to add the driver: selects its name and add the "derby.jar" file in the "Jar List" tab ("derby.jar" is in the derby folder, see step 2 above).

New Driver Definition

Specify a Driver Template and Definition Name

❌ Unable to locate JAR/zip in file system as specified by the driver definition: derby.jar.

Name/Type | JAR List | Properties

Available driver templates:

Name	System Vendor	System Version
▼ Database		
Derby Client JDBC Driver	Derby	10.1
Derby Client JDBC Driver	Derby	10.2
Derby Embedded JDBC Driver	Derby	10.0
Derby Embedded JDBC Driver	Derby	10.1
Derby Embedded JDBC Driver	Derby	10.2
IBM Data Server Driver for J...	Derby	10.0
IBM Data Server Driver for J...	Derby	10.1
Other Driver	Derby	10.0
Other Driver	Derby	10.1
Other Driver	Derby	10.2

Driver name:

Derby Embedded JDBC Driver

Driver type:

Derby Embedded JDBC Driver

6. Select the location of the database in workspace\SIS-1. Chose the name of the database "SIS_DB" (Derby will create a local database). No user id and pswd are needed for this project.

7. in the wizard showed in step 5.1, there is a button "Test Connection." Press that and if everything is ok then the driver is installed.

B. Use Eclipse "Database Development" to create a connection, then "Open SQL Scrapbook" and create two tables STUDENTS and ENROLLMENT

1. *You can use sql statements available in "projectB.sql" to create and populate the tables.*

2. Explore the tables (if no user id is used, Derby stores the tables in APP.TABLES. You can use "Data->Edit" on the table menu to visually edit the tables.

Now, you can issue any valid SQL statement in the scrapbook and it will be executed. Here are a few examples:
- Select * **from** STUDENTS;
- Select SID, CREDIT_TAKEN **from** STUDENTS **where** CREDIT_TAKEN >= 100;
- **select** SURNAME, CREDIT_GRADUATE **from** STUDENTS **where** CREDIT_TAKEN > 90 **and** SURNAME **like** '%Bradley%';

How can you INSERT a student or an enrolment row?
How can you DELETE a student or an enrolment row?
How can you UPDATE a student or an enrolment row?

Requirement 2. Set the JDBC, pool connection

Configure your SIS-1 project/application to use a JDBC pool.

How to do it:

1. Add "derby.jar" file to WEB-INF/lib directory. This is needed by Tomcat, when you "Run on Server..," Note that this will enable your application to access the database

2. Under META-INF, create "context.xml" file. You can get information about the syntax and semantic of its content from http://tomcat.apache.org/tomcat-8.0-doc/index.html. For now, it is important to know that this file contains information about the data source and the jdb driver to be used at runtime. Below is an example. Note **Resource** element and its attributes **driverClassName** and **url**. These attributes will depend on the jdbc driver you use and the location of your database. You have to change the url so it points to YOUR Database.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Context privileged="true" reloadable="true">
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Manager pathname="" />
    <Resource name="jdbc/EECS"  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
            type="javax.sql.DataSource"
            driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
            url="jdbc:derby:/Users/marin/Documents/workspace/SIS_DB"/>
    <ResourceLink global="jdbc/EECS" name="jdbc/EECS" type="javax.sql.DataSource" />
```

```
</Context>
```

Test Your configuration file and your runtime server settings

Create a *DAO.TestStudentDAO.java* class, with a *main* method that will connect to the database SIS-DB, retrieve all STUDENTS and prints each row to the server console.
To understand each line below, check the course notes and examples…

```java
import java.sql.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class TestStudentDAO {
  public static void main(String[] args) throws SQLException {
    try {
            DataSource ds = (DataSource) (new InitialContext()).lookup("java:/comp/env/jdbc/EECS");
            Connection con = ds.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM STUDENTS");
            while(rs.next())
                    String em= rs.getString("SID");
                    String fname = rs.getString("SURNAME");
                    System.out.println("\t" + em+ ",\t" + fname+ "\t ");
            }//end while loop
        con.close();
  } catch (NamingException e) {
      e.printStackTrace();
  }
}
}
```

To test the above code on the web server (Tomcat), you have to invoke it from a servlet.

- Create a servlet *ctrl.Test.java* and in the *doGet* method, invoke the main method above, such as..

```java
        try {
                TestStudentDAO.main(null);
        }catch (Exception e) {
                e.printStackTrace();
        }
```

- Run the servlet and you should see the student list on printed on the console.

Your environment is set correctly, you can move to the next requirement.

Requirement 3. SIS-1 application

Develop a reporting application that

- enables its client to generate reports that return information from a student records database.
- The application has to follow the MVC architecture

Ho to do it:

3.1 Create a form Form.jspx as shown below

This is the View component of our webapp. It presents a form (with POST submission) containing two input fields (with labels *Surname Prefix* and *Minimum Credit taken*). The form should have an appropriate legend and one submit button captioned *Report*

- Note that when the Report button is clicked then the result should appear below the form. Note also that the name prefix applies to the **SURNAME** only and the search for it must be case sensitive.

Student Information System

Name Prefix:

| Rodriguez |

Minimum Credit Taken:

| 20 |

[Report]

There are 2 entries.

Id	Name	Credits taken	Credits to graduate
cse67895	Philip, Rodriguez	81	90
cse29889	Irene, Rodriguez	84	90

3.2 Create a bean.StudentBean.java class

This class in the `bean` package is just a simple data structure that holds information about one student. It has four private attributes (sid, name, credit_taken, and credit_graduate), a constructor that sets them and four accessor (getter) / mutators (setter) methods. *Hint: You can just declare the attributes and ask Eclipse to generate the rest using its Source menu.*

3.3 Create a bean.EnrollmentBean.java

This class in the `bean` package is just a simple data structure that holds information about one course. It has three private attributes (cid, students, and

credit), a constructor that sets them and four accessor (getter) / mutators (setter) methods. *Hint: You can just declare the attributes and ask Eclipse to generate the rest using its Source menu. Students is the list of student IDs who are currently enrolled in the course.*

3.4   Create DAO.StudentDAO.java

This class in the `model` package is responsible for communicating with the DBMS. It is instantiated from the main model class and it contains one attribute, the data source that can grab connections from the pool. See the course notes to understand how a pooled connection can be used. Check also the test examples above. The essentials are listed below:

a. The constructor has to lookup for the data source. Note the name **jdbc/EECS** (it is from context.xml)

```java
public StudentDAO() throws ClassNotFoundException{
    try {
     ds = (DataSource) (new InitialContext()).lookup("java:/comp/env/jdbc/EECS");
    } catch (NamingException e) {
                e.printStackTrace();
    }
}
```

b. One method in this class must have the following header and sample body:

```java
public Map<String, StudentBean> retrieve(String namePrefix, int credit_taken) throws SQLException{
    String query = "select * from students where surname like '%" + namePrefix +"%'
    and credit_taken >= " + credit_taken;
    Map<String, StudentBean> rv = new HashMap<String, StudentBean>();
    Connection con = this.ds.getConnection();
    PreparedStatement p = con.prepareStatement(query);
    ResultSet r = p.executeQuery();
    while (r.next()){
            String name = r.getString("GIVENNAME") + ", " + r.getString("SURNAME");
            ….
            rv.put(….));
    }
    r.close();
    p.close();
    con.close();
    return rv;
}
```

3.5   DAO.EnrollmentDAO.java

Similar to StudentDAO, this class is responsible to retrieve data from Enrollment table.

3.6   model.SIS.java

This class in the `model` package is the main model class. It has attributes that hold an instance of `StudentDAO and EnrollmentDAO` (initialize it in the class constructor). In addition, it has the methods:

```java
public Map<String, StudentBean> retriveStudent(String namePrefix, String credit_taken) throws Exception
```

```java
public Map<String, EnrollmentBean> retriveEnrollment() throws Exception
```

that perform validation (possible by invoking private methods) and then invokes the appropriate methods in `StudentDAO/EnrollmentDAO`. Note that the validation logic must also sanitize the input to prevent SQL injection.

3.7 ctrl.Start.java

This class in the `ctrl` package is the front controller. Its init method must instantiate the model and store it in the servlet context and handle any exception by proper re-throwing and logging. Its doGet and doPost methods are very similar to (if not simpler than) those of Project-A. Note that no session scope storage is needed.