# MySQL Week 1 Exercises

Background

The weekly exercises are designed to augment the video lessons. In the exercises, you will develop a menu-driven application in Java. This application will demonstrate how to perform CRUD (Create, Read, Update, and Delete) operations on a MySQL database.

You will be working in a Project schema (database) that contains do-it-yourself (DIY) projects. A DIY project contains project details, materials, steps, and categories. Below is a diagram of the tables and relationships in the Project schema. Don't worry at this point if you don't understand what the diagram is telling you. This will become clear soon. For now, just know that there are five tables in the Project schema: project, material, step, category, and project_category. This is what you will build in the exercises.
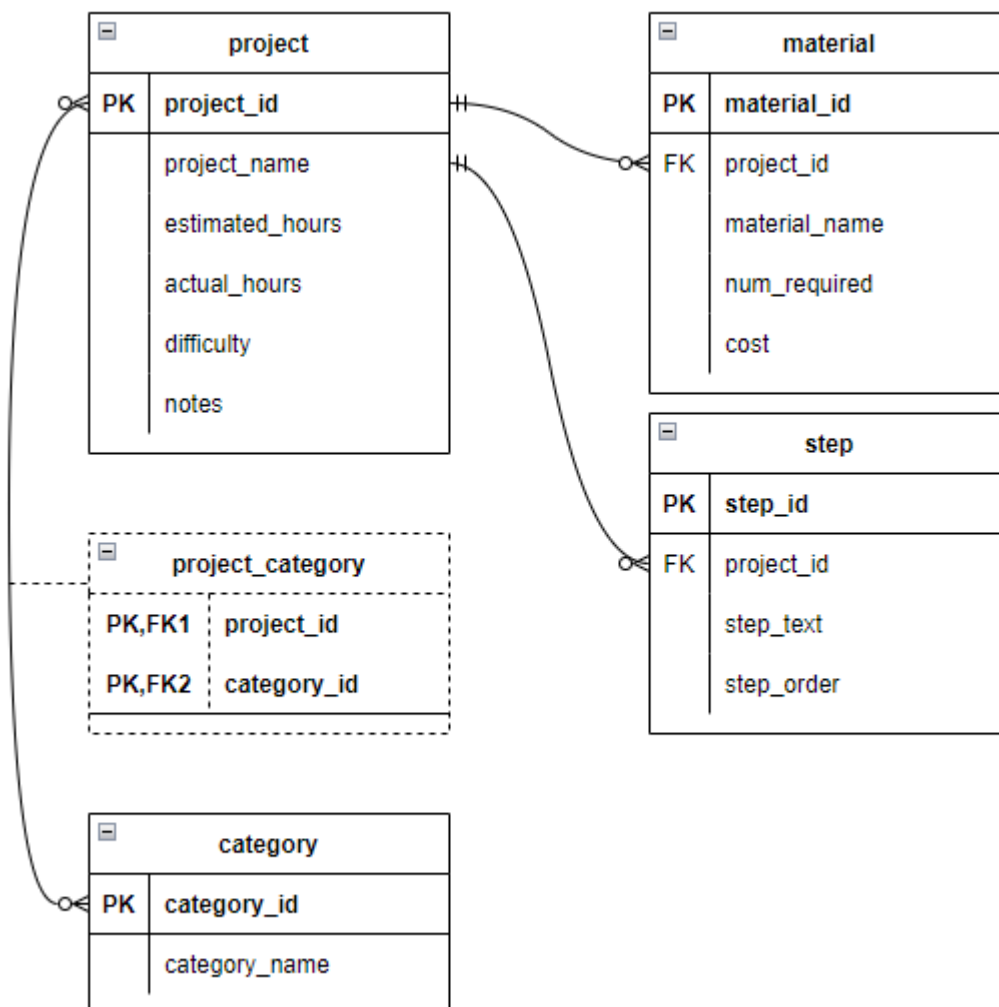
There will be a final project in this (MySQL) part of the back-end course. These exercises will help prepare you for that.

Objectives

In these exercises, you will:

- Use MySQL Workbench to create a schema and user.

- Use MySQL Workbench to assign schema privileges to a user.

- Create a Maven project in Eclipse.

- Add the MySQL driver as a dependency in pom.xml (Maven's Project Object Model – POM).

- Separate project concerns by creating packages.

- Write Java code to connect to a MySQL database and schema.

Important

In the exercises below, you will see this icon: . This means to take a screen shot or snip showing the results of the action or the code in the editor.

Exercises

In these exercises, you will use MySQL Workbench to create the project schema, as well as a user account. Then, you will create a Maven project. In the project you will write code to connect to the database schema that you created using MySQL Workbench.

Create the schema with MySQL Workbench

In this section, you will use MySQL Workbench to create the projects schema, which MySQL traditionally and unfortunately also calls a database. You will also create a user with privileges only for that schema. In other words, the user account that you will create can only access the tables in the projects schema and no others. This is a good practice in which a database account is only able to access the data it needs to perform its job. This step will help you accomplish that goal.

1. Open and log into MySQL Workbench.

2. Create a schema named "projects". Take a screen shot showing the projects schema. Your snip should look something like this:
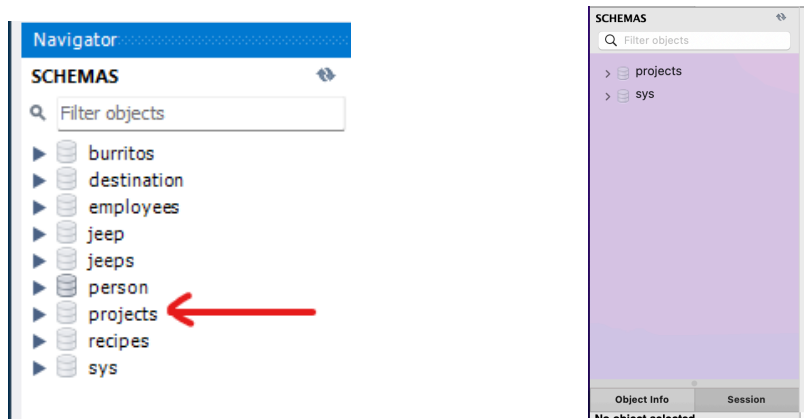
3. Create a user named "projects" and assign a password.

4. Add all privileges except "GRANT OPTION" 📷. Your snip should look pretty much like this:
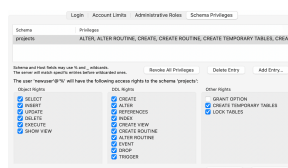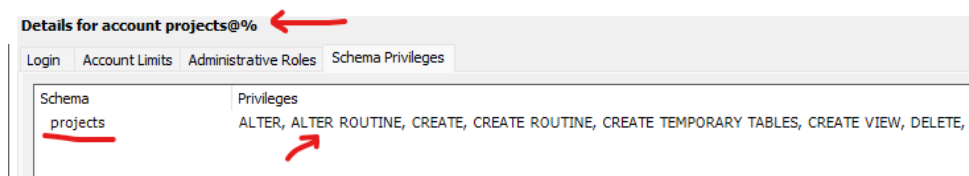
Create a Maven Project

Maven is a tool that is used to control the building of a Java project. Eclipse comes bundled with an internal version of Maven and the two tools have a tight integration. Maven has lots of features, but we will only be using it to add project dependencies. A project dependency is a library of code packaged as a Java Archive (JAR file). For our DIY project application, there is only one dependency required: the MySQL driver.

To create a new Maven project:

1. Click on the File menu in the top menu bar. Then select "New" / "Project". When the New Project wizard appears, expand "Maven". Select "New Maven Project" and click "Next".

2. Check the box, "Create a simple project (skip archetype selection)". Click "Next".

3. Enter the values in the table below into the fields and click "Finish".

Field	Value
Group ID	com.promineotech

Artifact ID	mysql-java
Version	0.0.1-SNAPSHOT

Modify pom.xml

Maven knows how to build projects (and provide project dependencies) based on configuration settings in an XML file named pom.xml. This file defines what Maven calls the Project Object Model (POM). The file is kept in the project root directory.

In this section, you will add a property with the correct Java version. You will then add the MySQL driver as a project dependency. Then, you will add a plugin that will use the Java version property to set the correct Java version.

In this section you will be working in pom.xml.

1. In this step you will add the Java version as a property in the Maven POM. You will set the value to 11 or 17 depending on the Java compiler version you have installed. To add the property:

   a. Create a `<properties>` section below the version element and inside the closing `</project>` tag. Add the closing </properties> tag below the opening tag.

   b. Inside the properties element, add the child element `<java.version></java.version>`. Inside the java.version element, set the version to 11 or 17. It should look like this:
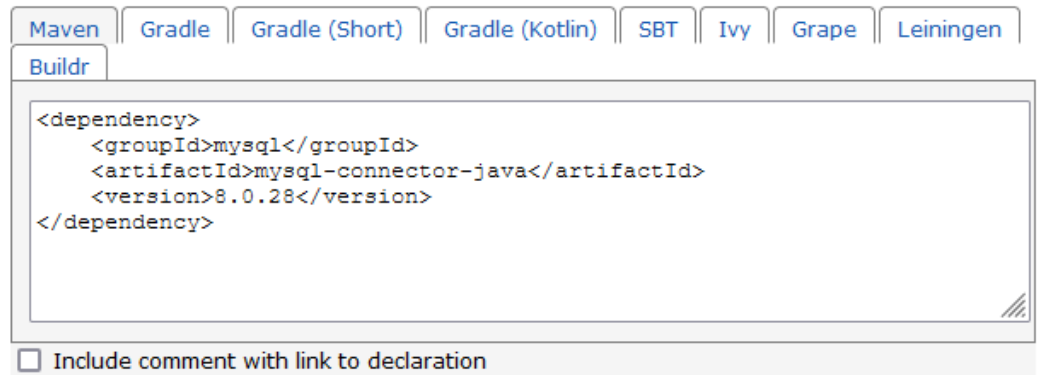
   ```
   <properties>
       <java.version>11</java.version>
   </properties>
   ```

2. In this step, you will add the MySQL driver as a dependency in the dependencies section.

   a. Below the `<properties>` element, create a new element named `<dependencies>` with the closing tag `</dependencies>` below it.

   b. In a browser, navigate to https://mvnrepository.com/. Type "mysql" into the search box and press Enter.

   c. Find "MySQL Connector/J" (most likely the first result) and click the link "mysql-connector-j". Click on the most recent version number link.

   d. You will see a page with information about the dependency in a table followed by several tabs and a box with the dependency. Click in the box and the Maven dependency is copied to the clipboard.

Click in here

```
Maven   Gradle   Gradle (Short)   Gradle (Kotlin)   SBT   Ivy   Grape   Leiningen
Buildr

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.28</version>
</dependency>

☐ Include comment with link to declaration
```

    e. Paste the contents of the clipboard into the <dependencies> section in pom.xml. The dependencies section should look like this:

```
<dependencies>
   <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
   </dependency>
</dependencies>
```

3. In this step, you will need to add a section to the POM that tells Eclipse which compiler version to use when compiling your project. This is done by adding a definition for the Maven compiler plugin.

    a. In a browser, navigate to the Maven compiler usage page: https://maven.apache.org/plugins/maven-compiler-plugin/usage.html.

    b. Find the section "Configuring Your Compiler Plugin". Copy the entire `<build>` section to the clipboard.

    c. Paste the clipboard contents into pom.xml below the `<dependencies>` section. Replace the XML comment inside the configuration element with a reference to the Java version defined in the properties section. To add a property reference, surround it with ${}. The section should look like this:

```xml
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

At this point, if you are hopelessly lost, refer to the solutions section below.

4. Take one or more screen shots to show all of pom.xml.
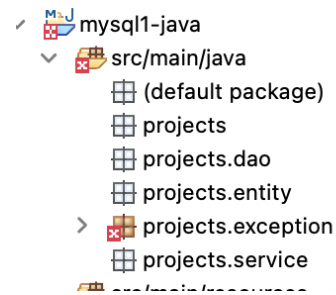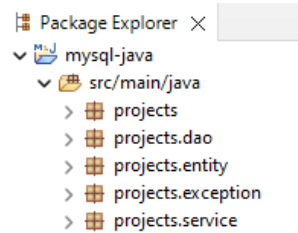

Set up the project packages

In this section you will create the project structure by adding some packages. This will all be done in the Package Explorer panel in Eclipse.

1. Create the following packages and subpackages under src/main/java. Take a screen shot showing the package structure in Package Explorer.

    a. projects

    b. projects.dao

   c. projects.entity

d.　projects.exception

　　　e.　projects.service

Your screen shot should look like this:

```
Package Explorer ×
  mysql-java
    src/main/java
      > projects
      > projects.dao
      > projects.entity
      > projects.exception
      > projects.service
```

```
  mysql1-java
    src/main/java
        (default package)
        projects
        projects.dao
        projects.entity
      > projects.exception
        projects.service
```

Create an exception class

In this section you will create an exception class that will be used in the mysql-java project. This is an unchecked exception that will be used throughout the application. We do this because all of the exceptions thrown by the Java Database Connectivity (JDBC) API classes are checked `SQLException` objects. In coding the application, you will turn the checked exceptions into unchecked exceptions to keep your code clean.

```
*DbConnection.j    DbConnection.ja    DbException.jav    *DbException.ja ×
1  package projects.exception;
2
3  public class DbException extends RuntimeException {
4
5      public DbException() {
6
7      }
8
9      public DbException(String message) {
10         super(message);
11
12     }
13
14     public DbException(Throwable cause) {
15         super(cause);
16
17     }
18
19     public DbException(String message, Throwable cause) {
20         super(message, cause);
21
22     }
23
24 }
25
```

In this section, you will create the class `DbException` in the `projects.exception` package.
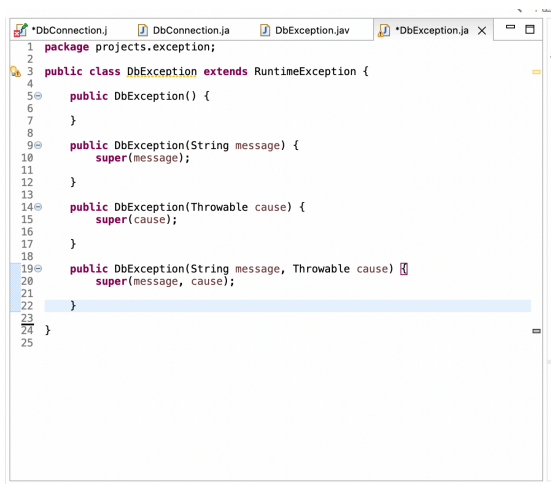
1.In the `projects.exception` package, create a class named "DbException". This class must extend `RuntimeException`. Override the following constructors from the superclass:

```
public DbException(String message) {}
```

```
public DbException(Throwable cause) {}
```

```
public DbException(String message,
Throwable cause) {}
```

Be sure to call the matching constructor in the superclass from each constructor in `DbException`. Take a screen

shot of the `DbException` class. 📷
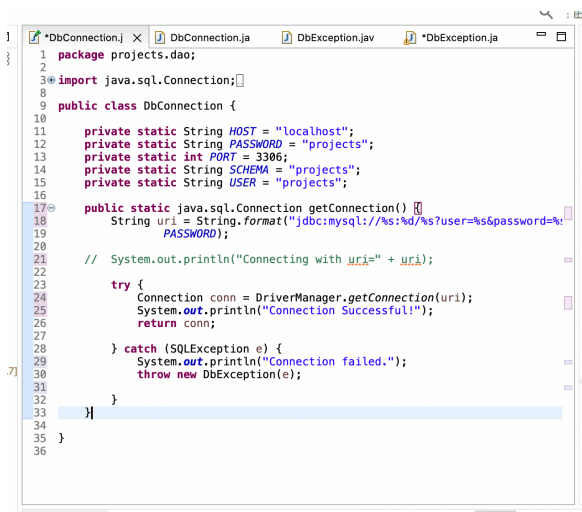
Create the JDBC connection class

In this section, you will create a class that obtains a JDBC `Connection` object from the driver manager. When you call `DriverManager.getConnection()`, the driver manager looks up the MySQL driver and loads it. It then establishes a TCP connection between the application and a MySQL server. If the connection cannot be made for some reason, the driver manager throws a checked `SQLException`. This is converted to an unchecked exception in a `catch` block.

Follow these steps to create the JDBC `Connection` class.

1. Create a class in the `projects.dao` package named `DbConnection`. In the class, create the following constants: `HOST`, `PASSWORD`, `PORT`, `SCHEMA`, and `USER`. Set the constants to the correct values. If you followed the instructions above and created a user in MySQL Workbench, the constants should look like this:

```java
private static String HOST = "localhost";
private static String PASSWORD = "projects";
private static int PORT = 3306;
private static String SCHEMA = "projects";
private static String USER = "projects";
```

2. In the `DbConnection` class, create a method named `getConnection()`. It should be `public` and `static` and should return a `java.sql.Connection` object. In the `getConnection()` method:

a. Create a String variable named uri that contains the MySQL connection URI.

b. Call `DriverManager` to obtain a connection. Pass the connection string (URI) to `DriverManager.getConnection()`.

c. Surround the call to `DriverManager.getConnection()` with a `try/catch` block. The `catch` block should catch `SQLException`.

d. Print a message to the console (System.out.println) if the connection is successful.

e. Print an error message to the console if the connection fails. Throw a DbException if the connection fails.

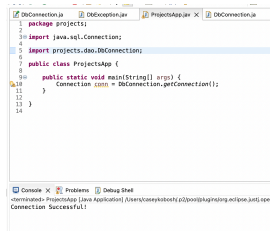f. Create a screen shot of the entire class. 📷

Create the main class

Every Java application must have an entry point. This is a class with a `main()` method. In this section, you will create a class with a `main()` method. From the `main()` method you will temporarily call `DbConnection.getConnection()` to test that you can obtain a connection to the MySQL server.

Follow these steps to create the application entry point.

1. Create a class in the `projects` package named `ProjectsApp`. The class must have a `main()` method.

2. In the `main()` method, call the `DbConnection.getConnection()` method.

Run the Java application. Create a screen shot of the console showing that the connection succeeded.

Final touches

Typically, when you push a project to GitHub, you only want to push source code, not built class files or extra configuration files maintained by a tool such as Eclipse. In a Maven project, all built classes are put in subdirectories off of the `target` directory. So, the `target` directory should not be pushed to GitHub.

Likewise, Eclipse maintains its project configuration in a directory named ".`settings`". This directory should also not be pushed to GitHub.

To exclude files or directories from being pushed to GitHub, put the directory and file names into a file named `.gitignore`. This file must be in the project root directory. Simply put the paths of the files or directories on separate lines in `.gitignore` to tell git to ignore these files or directories.

To ignore files, put the file name relative to the project root. Separate directory names with slashes ("/"). To ignore directories, put the directory name on a separate line in `.gitignore` and add a slash ("/") on the end.

Follow these steps to instruct git to ignore the .settings directory and the target directory.

1. In the root of the Eclipse project, create a file named ".`gitignore`". It should contain the following lines:

   `.settings/`

   `target/`

   https://github.com/CKobosh/Week-7-Coding-Assignment.git

SOLUTIONS:

pom.xml

```xml
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://maven.apache.org/POM/4.0.0
        https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.promineotech</groupId>
    <artifactId>mysql-java</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.27</version>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <source>${java.version}</source>
                        <target>${java.version}</target>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

DbConnection.java

```java
package projects.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import projects.exception.DbException;

public class DbConnection {
  private static String HOST = "localhost";
  private static String PASSWORD = "projects";
  private static int PORT = 3306;
  private static String SCHEMA = "projects";
  private static String USER = "projects";

  public static Connection getConnection() {
    String uri = String.format("jdbc:mysql://%s:%d/%s?user=%s&password=%s", HOST, PORT, SCHEMA,
        USER, PASSWORD);

    try {
      Connection conn = DriverManager.getConnection(uri);
      System.out.println("Connection to schema '" + SCHEMA + "' is successful.");
      return conn;
    } catch (SQLException e) {
      System.out.println("Unable to get connection at " + uri);
      throw new DbException("Unable to get connection at \" + uri");
    }
  }
}
```

ProjectsApp.java

```java
package projects;

/**
 * @author Promineo
 *
 */
public class ProjectsApp {

  /**
   * @param args
   */
  public static void main(String[] args) {
  }

}
```

.gitignore

```
.settings/
target/
```

DbException.java

```java
package projects.exception;

@SuppressWarnings("serial")
public class DbException extends RuntimeException {

    public DbException(String message) {
        super(message);
    }

    public DbException(Throwable cause) {
        super(cause);
    }

    public DbException(String message, Throwable cause) {
        super(message, cause);
    }
}
```