Search for [\*] for material to be added.

Using JDBC with MySQL

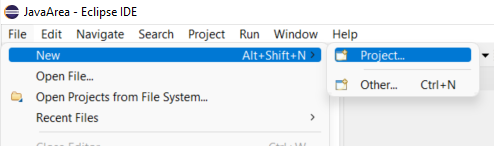
# Requirements

On your computer you will need:

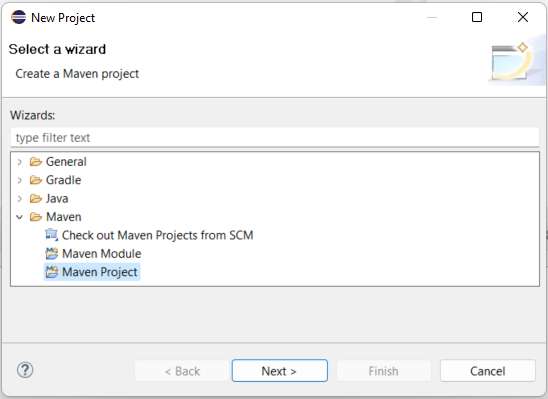
* MySQL
* Eclipse with Maven

# Create Project in Eclipse

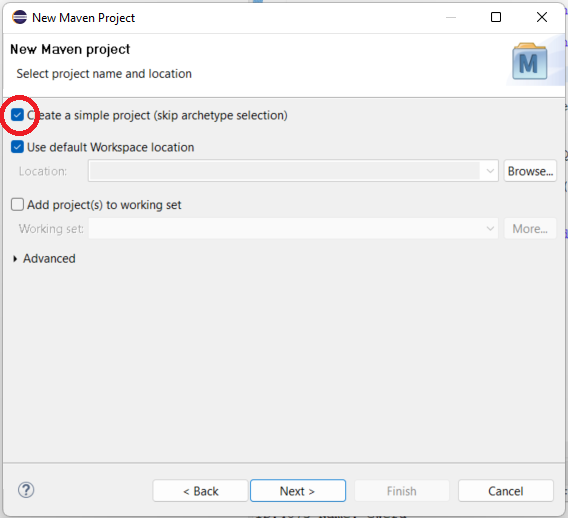
File – New - Project



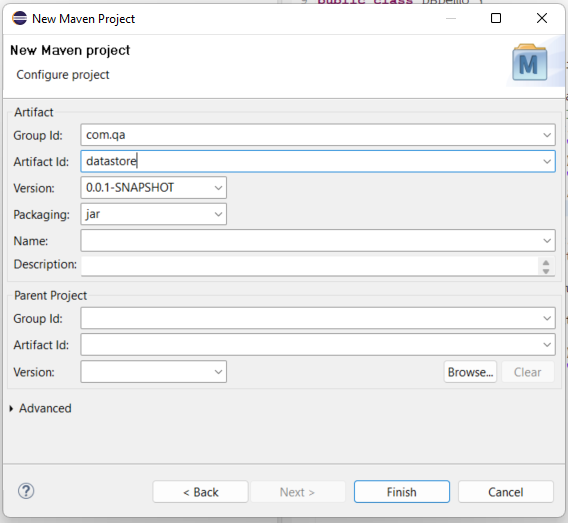
then select Maven Project (if this is not already an option in File – New).



Ensure “Create a simple project” is selected and press Next.

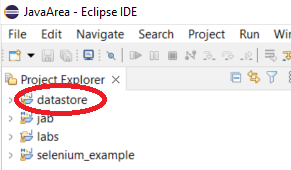


Give suitable values for Group Id and Artifact Id, and press Finish.

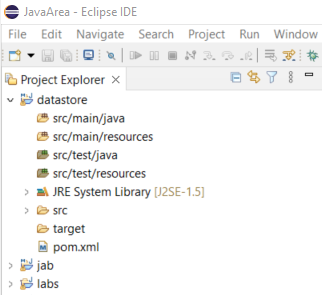


I have chosen a Group Id of com.qa and an Artifact Id of datastore.

Your project should appear in the Project Explorer window, along with any existing projects.

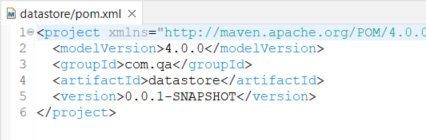


Click on the button to the left of the project to expand it:



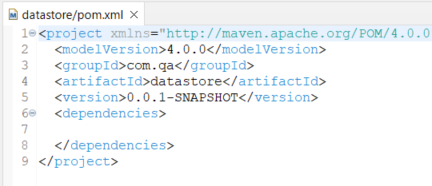
We will be looking more closely at the top item, src/main/java, and the bottom item, pom.xml.

Let’s begin by double clicking on pom.xml. POM stands for Project Object Model, and XML stands for eXtended Markup Language, which consists of data items between paired tags. Here are the contents of pom.xml:



(The first line extends out of view.)

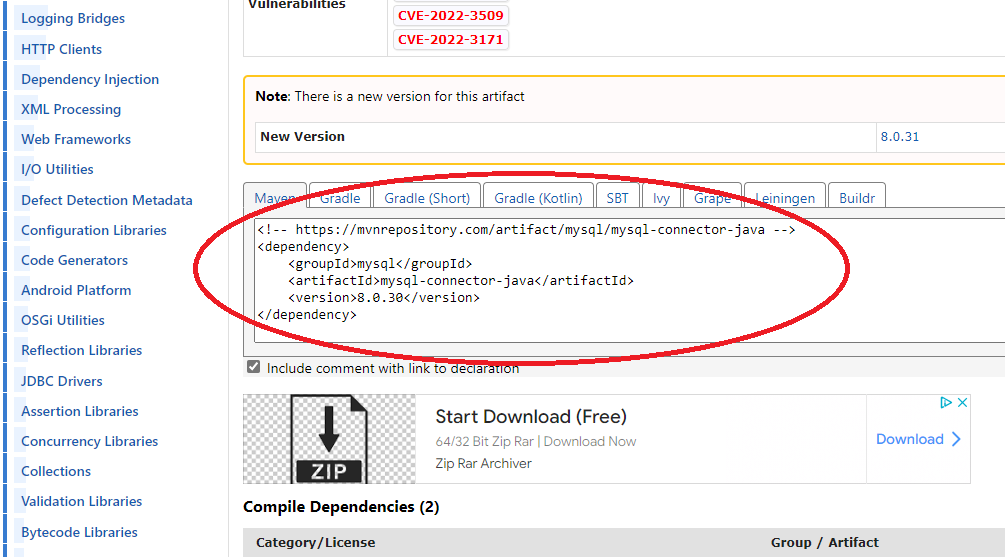
In addition to the existing contents, we need to add a pair of dependencies tags between <project ...> and </project>, after the version tags.



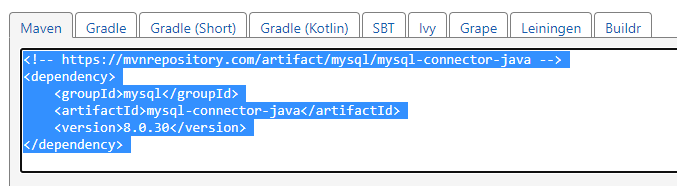
Between <dependencies> and </dependencies> we need to add a pair of dependency (singular) tags and the relevant data for JDBC. This can be found at this link:

[Maven Repository: mysql » mysql-connector-java » 8.0.30 (mvnrepository.com)](https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.30)

Scroll down the page to the XML content:



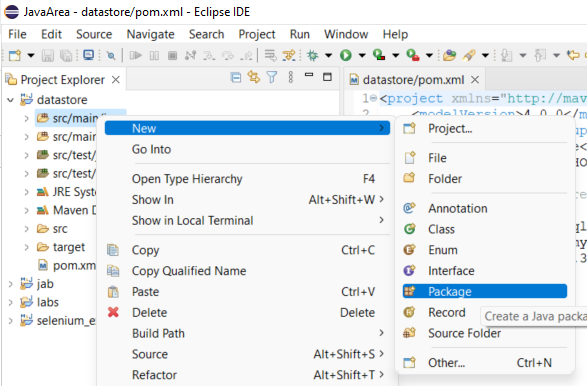
Then highlight the XML:



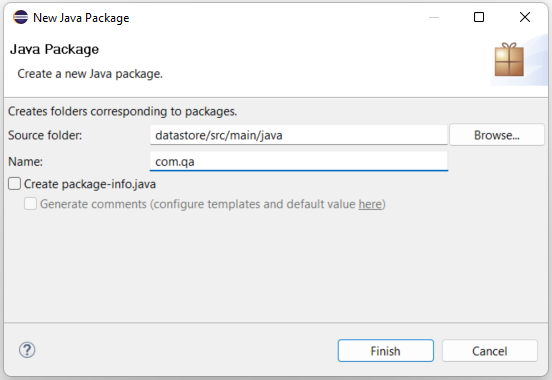
and copy and paste it into the pom.xml file. Press Shift-Ctrl-F to format it neatly:



Return to the Project Explorer window. Under datastore, right click on src/main/java. Select New – Package



Then change the value of Name. (It might be blank, or it might be the same as the project name.) I am using com.qa.

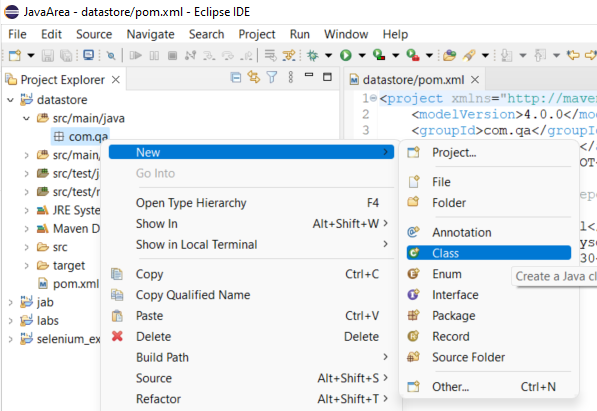


The package appears in the Project Explorer window looking like a parcel – a square with vertical and horizontal string crossing its centre.

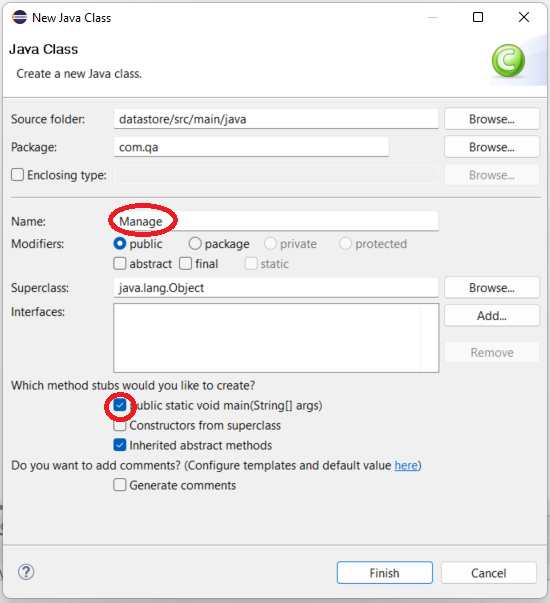
You are now ready to create a JDBC project.

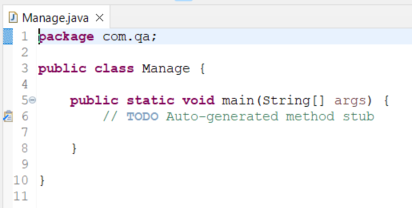
# Create Classes

Right click on the package and select New – Class.

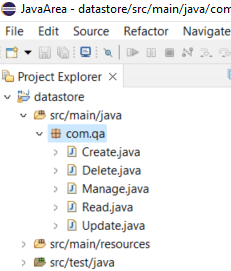


Give it the name Manage. (Class names should begin with a capital letter.) Tick the box next to “public static void main(String[] args)”. Press Finish.





Repeat the process to create further classes in the same package, but do not tick the box next to “public static void main(String[] args)” again. These classes are to be called Create, Read, Update and Delete.



Note that I have put each of these classes in a separate file. This is merely preference; you might choose to keep them all in the same file. However, note that a maximum of one public class is allowed in a .java file, and if one is present, the name of the file (excluding the .java extension) must be the same as the name of the public class. So if you do choose to put all your classes in one file, ensure that none of them have the public access modifier except for Manage.

# Java Code in the Classes

The code you will need can be found in GitHub in the following link:

[\*]

This section describes the code. You may wish to jump to the next section, Check Existing Databases, and come back to this section afterwards.

Note that the package command must be the first line of actual code. (You can have whitespace or comments above it, but nothing else.) If you have called your package something other than com.qa, your code must reflect this.

## Manage.java

From this file, the user chooses which CRUD action to perform – Create, Read, Update or Delete.

The Scanner class is imported on line 3, an instance is created on line 11, and the reference variable userInput is used on line 23 to read the user’s input on each iteration of the while loop. The trim() method is used to remove any leading or trailing whitespace (so that “ read “ looks the same as “read”), the toUpperCase() method makes all letters capital, and charAt(0) returns only the char value at the zero position in the string. So “ update “ returns ‘U’, for example.

The user’s input is stored in the char variable called option.

The while loop (lines 13-48) will repeatedly iterate until the option stores ‘Q’.

Each of the CRUD options calls the proceed() method in the corresponding class. Each proceed() method is static, which means there is no need to create an instance of type Create, Read, Update or Delete. (You may wish to expand the code to allow this.)

When the Create option is selected, it is removed from the menu to prevent the code from attempting to create the same database and table more than once. This is only necessary because the SQL commands in the Create class are currently hard coded. (The SQL commands in the Read, Update and Delete classes are also hard coded, but unlike the CREATE commands they can be executed repeatedly.)

If the user selects one of the other CRUD options before selecting Create, the program will be unable to find the database. (You may prefer to make Read, Update and Delete unavailable until after the database has been created.)

## Create.java

The proceed() method in this class is hard coded to create a database called exams and a single table called results, which it then populates. (You may wish to rewrite this code to create different databases and/or tables, either hard coded or allowing the user to choose.)

Lines 3-6 are the imports required for all activities involving connecting to a database and querying it. The imported classes are used in the code below.

conn, a static variable of type Connection, is declared on line 10.

statement, a static variable of type Statement, is declared on line 11.

Local string variables are declared in the proceed() method on lines 15-17.

On line 20, the static getConnection() method of the imported DriverManager class is called in an attempt to connect to MySQL on the local machine. Note the URL of the local MySQL, and the username and password (both “root” in this case). If successful, the reference is stored in the variable conn.

A number of things can go wrong at this point even if the code has been written correctly. For example, MySQL might not have been installed, or the port number might have been changed so that it is not 3306. This is why exception handling is used for most, if not all, SQL operations in Java.

If line 20 executes successfully, the method createStatement() is called from the instance of DriverManager referred to by conn.

On line 29, Statement’s method executeUpdate() is called with the query stored in newDB on line 27. If successful, this will create a new database called exams.

Line 35 is similar to line 20, but this time, instead of simply connecting to the local MySQL, it is connecting to the newly created exams database. Line 36 ensures that statement refers to this new connection.

Lines 42-47 comprise the definition of a new table which we wish to create. Lines 43-46 consist of column name followed by data type (and other information – the id cannot be null), and line 47 declares that id is the primary key. Line 49 executes this SQL command. We now have an empty table called results.

Lines 55-68 populate the table. For each row of the table, the command consists of “INSERT INTO” followed by the name of the table (results), followed by a pair of parentheses containing the data to be inserted. Note that the order of the data corresponds to the column definitions – a unique integer for the id, varchar (effectively a string) for name, and a (not necessarily) unique integer each for maths and english.

## Read.java

Lines 3-7 are the same imports as for the Create class, with the addition of the ResultSet class which is required for storing the values returned by a SELECT query.

The code in the Read class as far as line 24 is similar to that of the Create class, with the main difference being that we can now immediately link to the exams database. There are also two local variables: a String called query which stores a SQL SELECT statement, and rs, which is of the newly-imported ResultSet class.

Line 26 executes the query (or attempts to); if successful, the results of the query are stored in rs. On lines 27-31, the while loop continues to iterate as long as there is a row remaining in rs. The code in the while loop displays each column value. Note that if you changed the code to allow a different SELECT statement, you would have to adjust this display code accordingly.

You may wish to compare the results with the same query run in MySQL Workbench.

## Update.java

The first 8 lines of this class are the same as those of Read.java, except that ResultSet is not imported as it is not needed. (I have commented it out.) We still have the static members conn and statement.

Inside the proceed() method both the String variable and ResultSet variable have been removed – this time we are using string literals in the statements rather than String values copied to a variable.

The code to link to the exams database is the same as in Read.java.

The update scenario is as follows:

* David does not like his name being shortened to Dave.
* Chris scored 70 in maths, not 60 as the record shows.
* The examiner forgot to include Ursula’s details. She scored 75 in maths and 95 in English.

Lines 23-25 implement the three updates. The first two use the UPDATE command, the third uses INSERT, which is familiar from Create.java.

The id value in the INSERT statement is unique. What would happen if we reused an existing id value instead?

## Delete.java

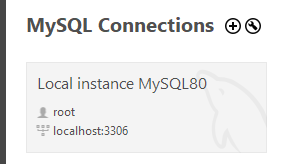
This is essentially very similar to the Update.java file, the only differences being the name of the class and that only a single statement – a DELETE statement – is executed on line 22.

For the sake of variety, we have used the student’s name rather than id for the WHERE conditional. This is not good practice, however, as the name might not be unique.

# Check Existing Databases

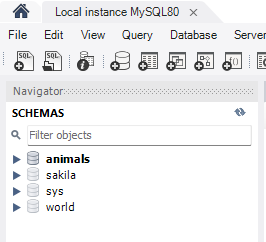
At this point it is assumed that you have copied the code from GitHub into each of the corresponding class files.

Open your MySQL Workbench app and click on File. Under MySQL Connections, about halfway down on the left, you should see something like this:



Click on the grey area, and enter the password when prompted. This should be the same as the User, i.e. “root”.

At the top left, under Navigator, you should see something like this:



Note that when working with MySQL Workbench, you might get the impression that the terms “database” and “schema” are interchangeable. This is not strictly correct, but for the purposes of this course, the distinction is not important.

The databases (or schemas) listed might be different for you, but the important thing is, the database called exams is *not* present. One database will probably be in bold, indicating that it is the most recently used one.

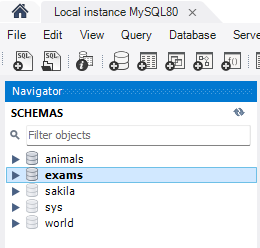
Right click in the white space under the list. A menu appears which includes the option “Refresh All”. Click on it. The most recently used database will probably expand; if it does, either close it by clicking on the down arrow to its left, or ignore it.

# CRUD – Create a Database and a Table

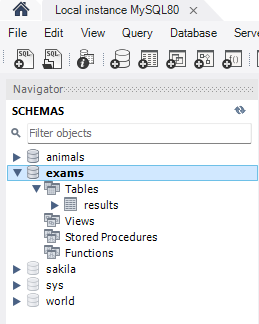
Now run the Java program. Select option C to create a database and table.

Go back to the list of databases in MySQL Workbench, and refresh it again.

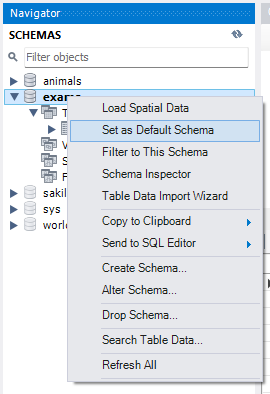
The exams database should now appear:



If it is not already expanded, click on the arrow to the left, then click on the arrow to the left of Tables so that the results table will appear.



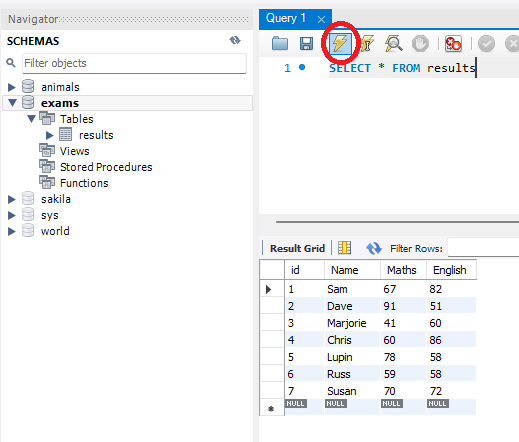
Right click on exams and select “Set as Default Schema” from the dropdown menu.



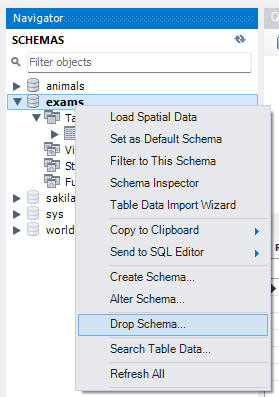
In the query area to the right of this, enter

SELECT \* FROM results

and click on the lighning bolt icon. You should see the results below – a listing of the seven pairs of exam results.



Note, running the Create code again will cause problems, as it will attempt to create a database and table that already exist. If you need to get rid of the exams database – and you probably will when you are experimenting with various SQL commands – right click on exams in the Navigator window and select Drop Schema.



Select Drop Now from the prompt.

To restore the database, run the Java code again and select C.

# CRUD – Read Data from a Database Table

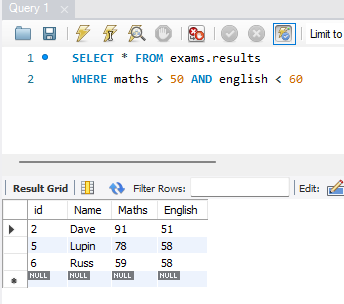
We can conveniently read data from a table using the SELECT keyword in MySQL Workbench, as we saw above.

The hard coded command in the Read option is

SELECT \* FROM exams.results

WHERE maths > 50 and english < 60

Run the code and select R (after creating the database and table). Compare the results with running the same SQL command in MySQL Workbench.



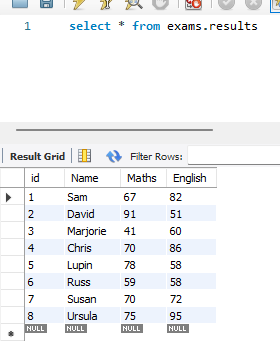
Note, you may get different results if you revisit the Read option after using the Update or Delete options.

# CRUD – Update Table

The update scenario is as follows:

* David does not like his name being shortened to Dave.
* Chris scored 70 in maths, not 60 as the record shows.
* The examiner forgot to include Ursula’s details. She scored 75 in maths and 95 in English.

That is to say, two existing rows have values changed, and one new row is added. Run the code and select U. You may wish to use MySQL Workbench to check the results:



# CRUD - Delete

Lupin has been expelled from the school, so his records have been deleted. Select D, then use MySQL Workbench to check that his details have been removed.

As can be seen below, the row with the id of 5 is gone.

