Using JDBC to connect to a MySQL database.

Connecting Java to the MySQL Community Edition database.

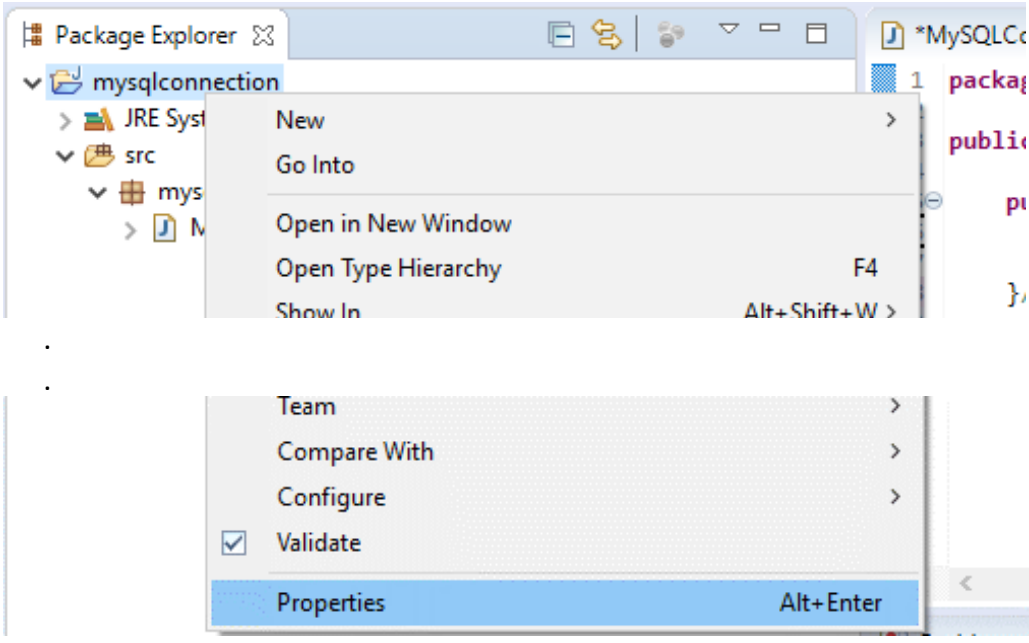
In this tutorial you will learn how to use the MySQL Community Edition database as the data source for a Java application using JDBC in Eclipse.

Creating a Project in Eclipse.

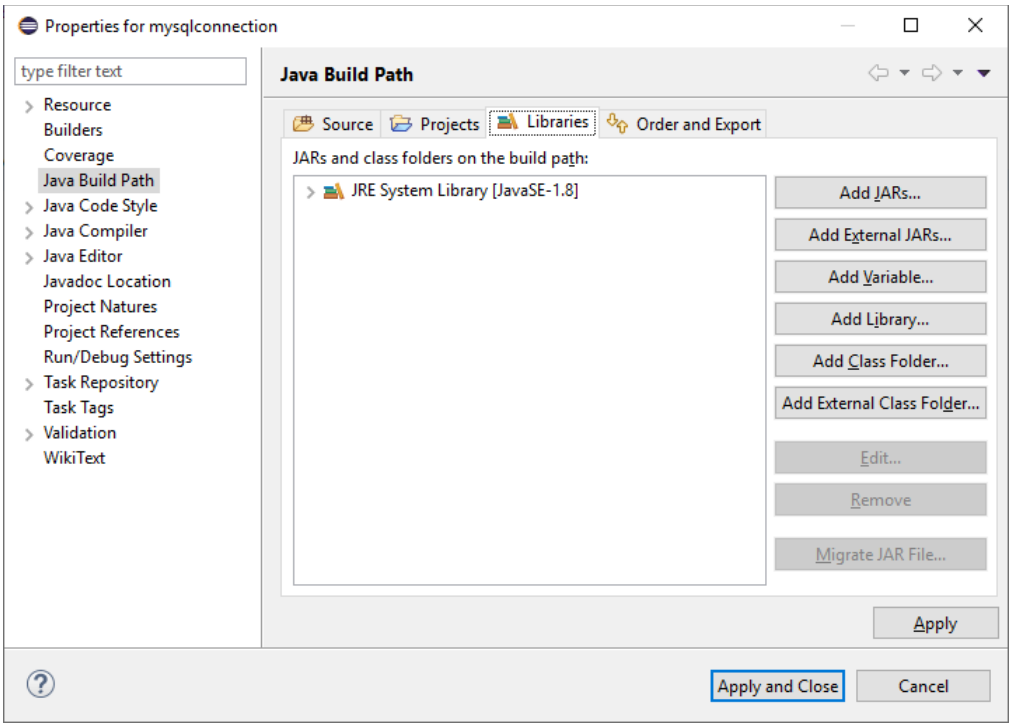
1. Open the Eclipse IDE on your system.
2. Create a project and a package named mysqlconnection that includes a main method.

```
package mysqlconnection;

public class MySQLConnection {

    public static void main(String[] args) {

    }//end method main
}//end class mysqlconnection
```

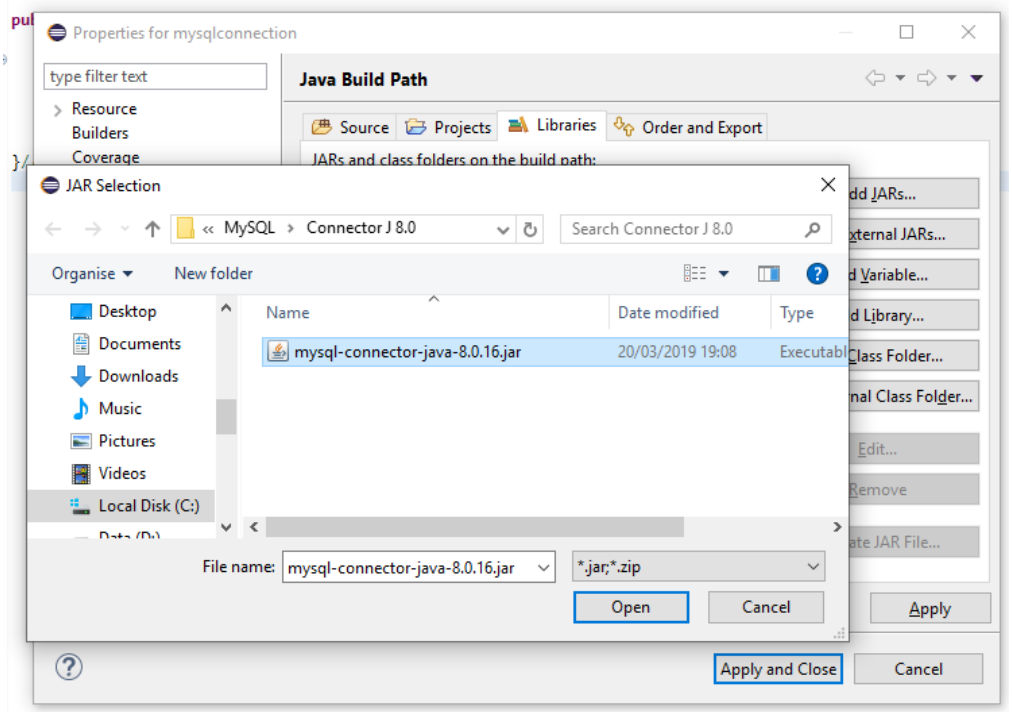
3. You will need to include the jar file that will handle the database connection to your project. Unless you changed it the Java connector was downloaded to the MySQL directory at C:\Program Files (x86)\MySQL\Connector J 8.0.
    a. In Eclipse **right click** the project name (mysqlconnection) and select **properties** from the drop-down menu.



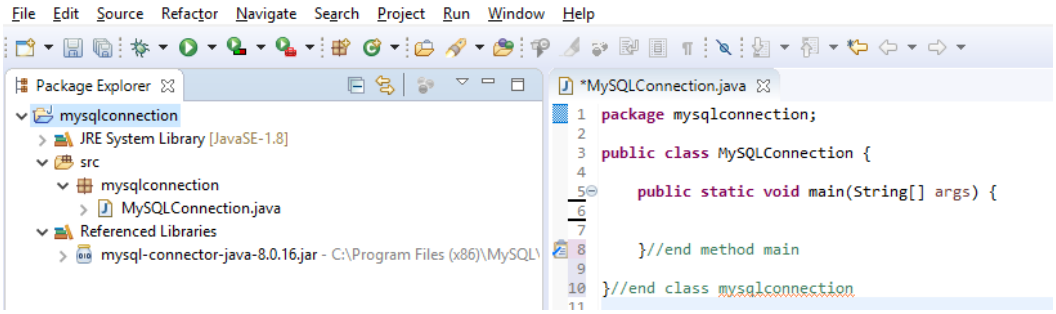
b. From the properties box select **Java Build Path** and then click on the **Libraries** tab.



c. Click on Add External JARs and browse to the location of the JAR file (C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-java-8.0.16.jar) and click Open.



d. Click on **Apply and Close** to complete the addition of the JAR file to the project.



4. Now that the JAR file has been included in the build path you can start to use it in the program as part of setting up the variables.

   a. Create the following variables at the top of the main method:

```
public static void main(String[] args) {
        String driver = "com.mysql.cj.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306/hr?&serverTimezone=UTC";
        String user = "root";
        String password = "root";
}//end method main
```

      i. driver holds the name of the class that implements the java.sql.Driver code that is accessed through the JAR file.
      ii. url holds the connection information to the database. Localhost and 3306 represent the location and port number to access the database, hr is the name of the database and server Timezone resolves any time issues with the database.
      iii. user holds username required to login to the database.
      iv. password holds password required to login to the database.

   b. You now need to register the class so that it can be used within your code. Create the following code under the variables in the main method:

```
        String password = "root";

        try {
                Class.forName(driver);
                String query = "SELECT * FROM departments";
        } catch (ClassNotFoundException e){
                System.out.println(e);
        }//end try catch
}//end method main
```

i. The try catch statement will handle any run time errors that occur when the class can not be found at the location specified by the driver.
ii. Class.forName(driver) gets a reference to a Class and registers it with the DriverManager.
iii. **String query** holds the content of the SQL query in a String variable named query. This is the same query you executed on the database in the command line.

c. Now you need to create some resources to allow for communication with the database. Add the following code in bold to your existing try catch statement:

```
try {
    Class.forName(driver);
    String query = "SELECT * FROM departments";
    try(Connection con = DriverManager.getConnection(url, user, password);
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(query)) {

    } catch (SQLException e) {
        System.out.println(e);
    }//end try catch
} catch (ClassNotFoundException e) {
        System.out.println(e);
}//end try catch
```

i. **Connection con** creates the connection to the database using the url, username and password defined in the variables.
ii. **Statement st** takes the connection and sets up the interface so that an instruction can be passed to the database.
iii. **ResultSet rs** uses the statement to pass the query to the database and returns the result set to the **rs** variable.
iv. The catch statement will handle any run time SQL errors that occur by printing the error to the console.

You will need to import some libraries to enable the code to work. Whenever Eclipse identifies options for you always select the java.sql option. You should have the following import list at this point:

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

d. Create the following method under main in the mysqlconnection class. Import any java.sql libraries required.

```
}//end method main

public static int getColumnNames(ResultSet rs) throws SQLException {
    int numberOfColumns = 0;
    if (rs != null) {
        //create an object based on the Metadata of the result set
        ResultSetMetaData rsMetaData = rs.getMetaData();
        //Use the getColumn method to get the number of columns returned
        numberOfColumns = rsMetaData.getColumnCount();
        //get and print the column names, column indexes start from 1
        for (int i = 1; i < numberOfColumns + 1; i++) {
            String columnName = rsMetaData.getColumnName(i);
            System.out.print(columnName + ", ");
        }//endfor
    }//endif
    //place the cursor on a new line in the console
    System.out.println();
    return numberOfColumns;
}//end method getColumnNames

}//end class mysqlconnection
```

e. In the main method add the method call to the new getColumnNames method as well as the start of the if statement that will be executed if there are more than zero columns in the result set underneath the ResultSet rs line:

```
        ResultSet rs = st.executeQuery(query)){
        int colNum = getColumnNames(rs);
        if(colNum>0)

        //endif
} catch (SQLException e) {
```

f. You will now display the contents of the result set to the console.
   i. You will use a while loop to go through each row in the result set (rs.next()) until it reaches the end.
   ii. You will use a nested for to display the value for each column in the result set. A column is identified by using rs.getString(indexvalue) so for each column in the row the for loop will use the number of columns to be able to display each one.

iii. If the column to be displayed is the last one it will be printed using a println statement to put each row on a new line.

Add the following code in bold inside the if statement in main:

```
            if(colNum>0)
                while(rs.next()){
                    for(int i =0; i<colNum; i++) {
                        if(i+1 == colNum)
                            System.out.println(rs.getString(i+1));
                        else
                            System.out.print(rs.getString(i+1)+ ", ");
                        //endif
                    }//endfor
                }//endwhile
            //endif
        } catch (SQLException e) {
```

g. You can now run the program and view the contents of the departments table in the console.

```
DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID,
10, Administration, 200, 1700
20, Marketing, 201, 1800
50, Shipping, 124, 1500
60, IT, 103, 1400
80, Sales, 149, 2500
90, Executive, 100, 1700
110, Accounting, 205, 1700
190, Contracting, null, 1700
```

5. Substitute the **departments** table for **employees**, **job_history** and **jobs** to view the content of the remaining tables.

6. You can run complex queries through JDBC as easily as you have run these basic select statements. If you wanted to view the department information for each employee then you would have to run a query that includes a join statement for both tables.

   Update the query to do this and then test it by running your code:

```
String query = "SELECT * FROM employees INNER JOIN departments ON
                employees.department_id = departments.department_id;";
```