

Lesson 01 Demo 01

Configuring JDBC and Creating Connection to MySQL DB

Objective: To configure JDBC and create a connection to a MySQL DB by creating a Java project and executing the `createConnection()` method for database operations

Tools required: Eclipse IDE

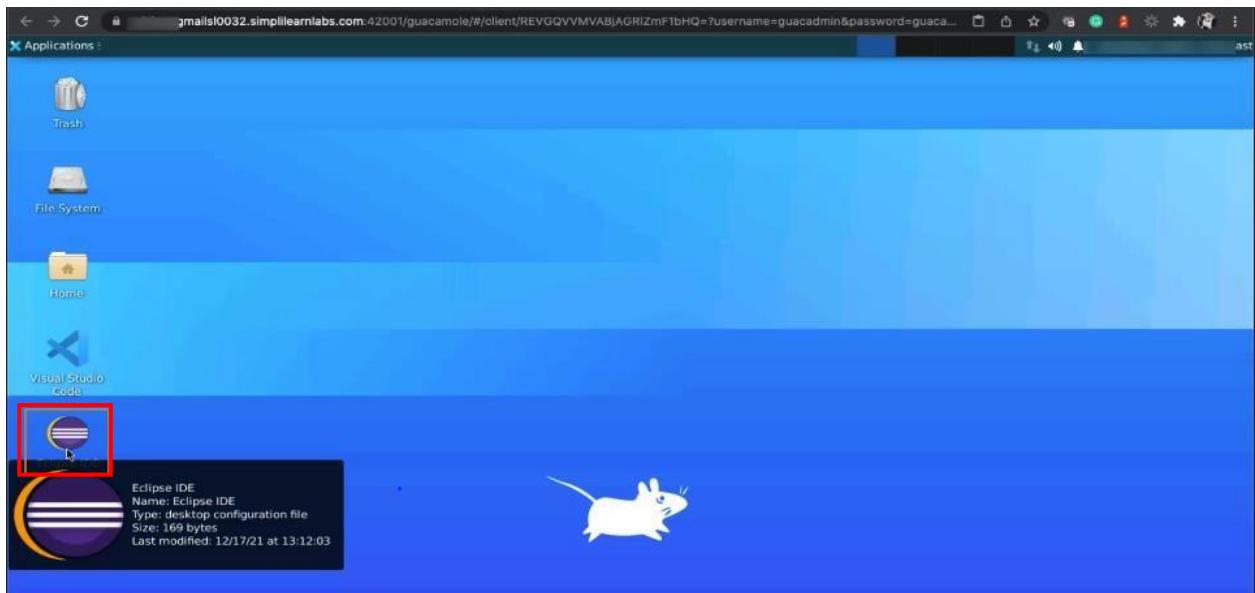
Prerequisites: None

Steps to be followed:

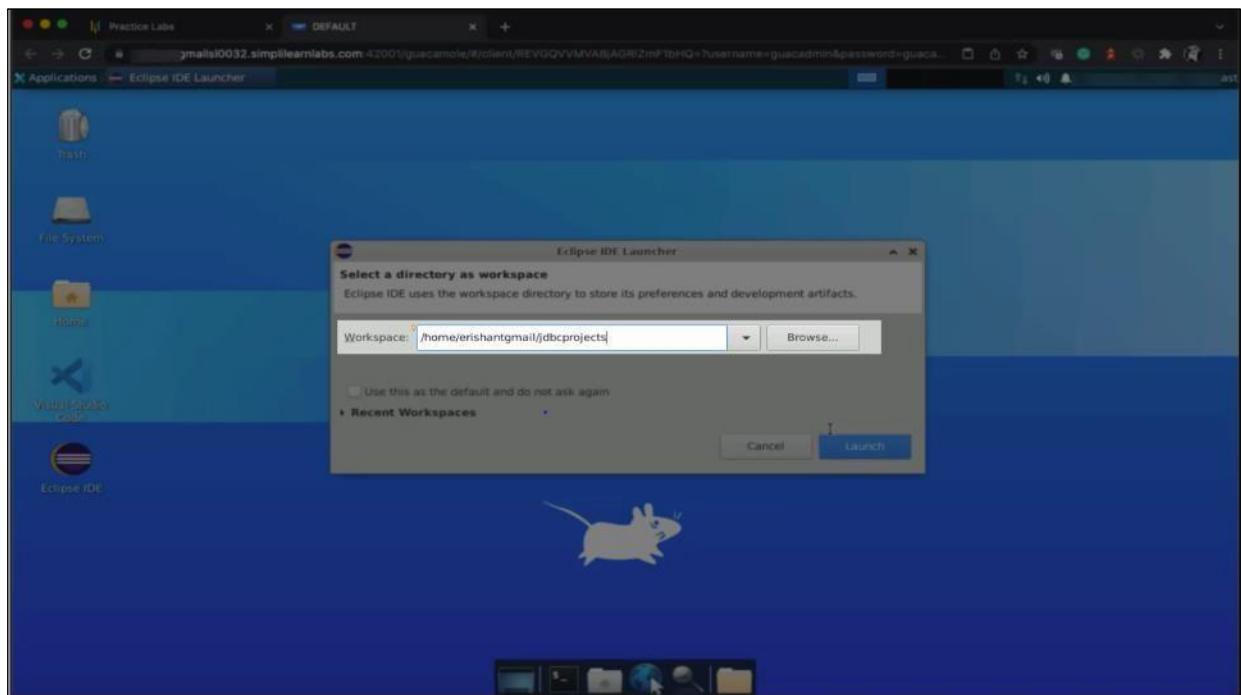
1. Create a Java project
2. Download the MySQL Connector/J library
3. Create a Maven project
4. Create an interface in com.example.cms
5. Create the DB class
6. Create a connection in the DAO.java file
7. Initialize the connection
8. Execute the `createConnection` method

Step 1: Create a Java project

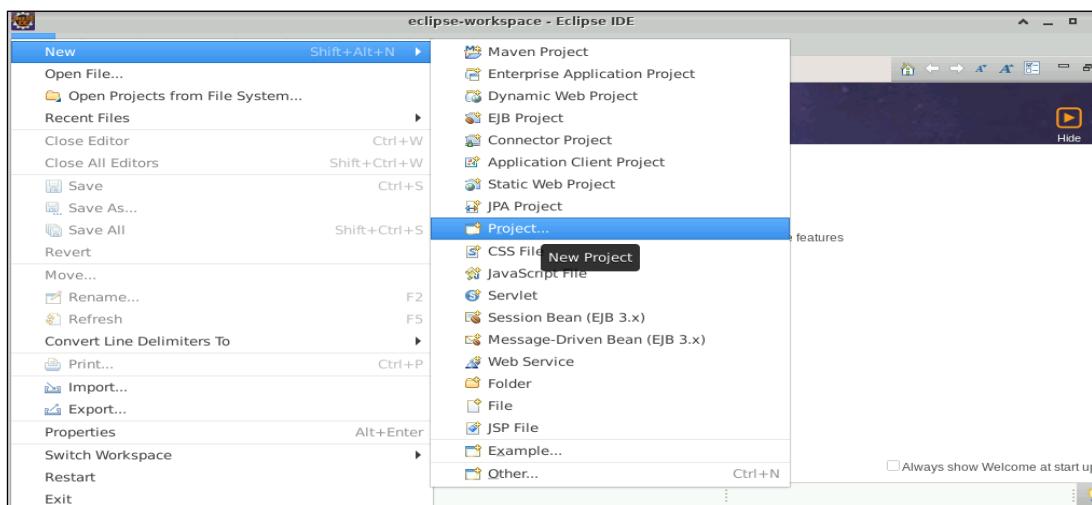
1.1 Open Eclipse IDE to create a Java project



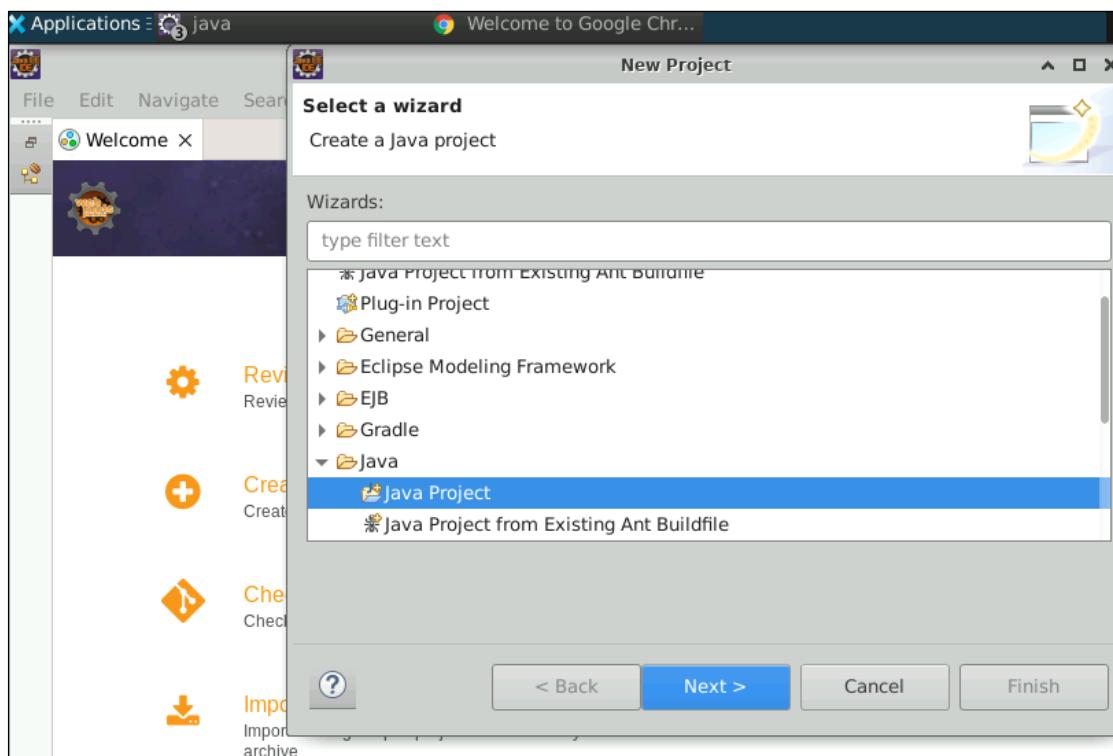
1.2 Select the default workspace and click on **Launch** to open Eclipse IDE



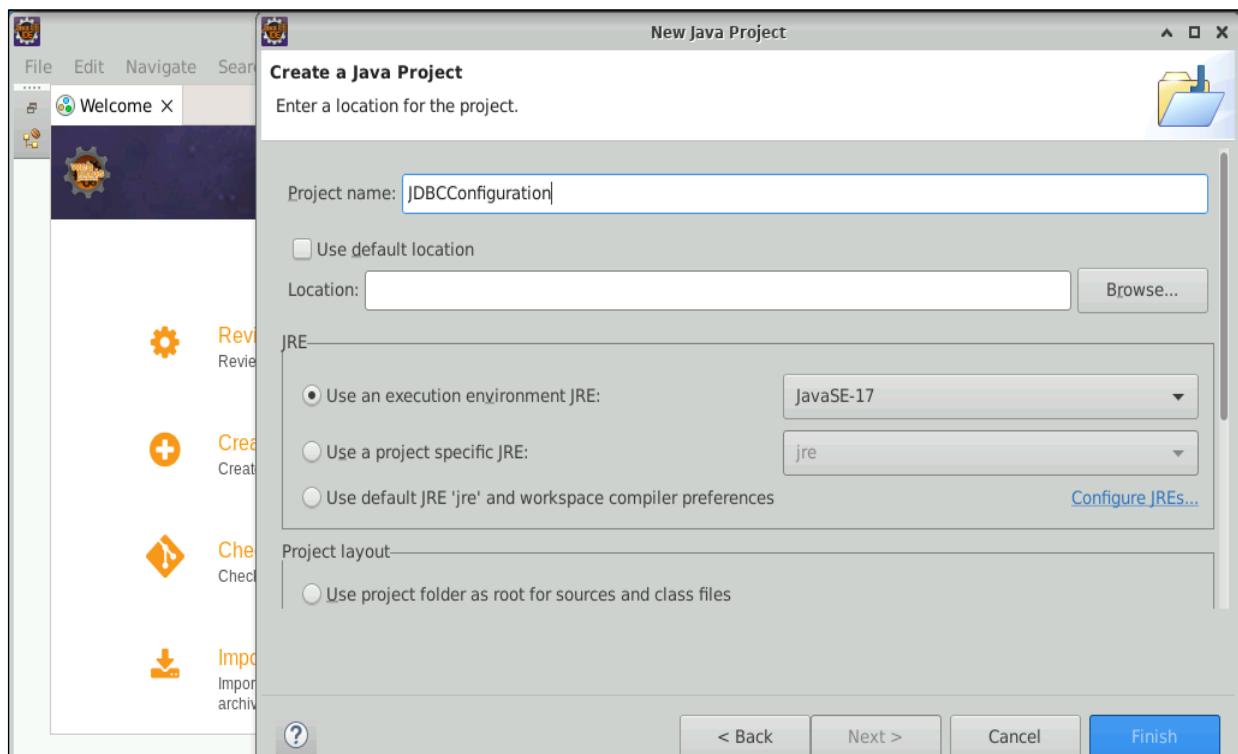
1.3 Create a new Java project, navigate to **File** from the toolbar > **New > Project**



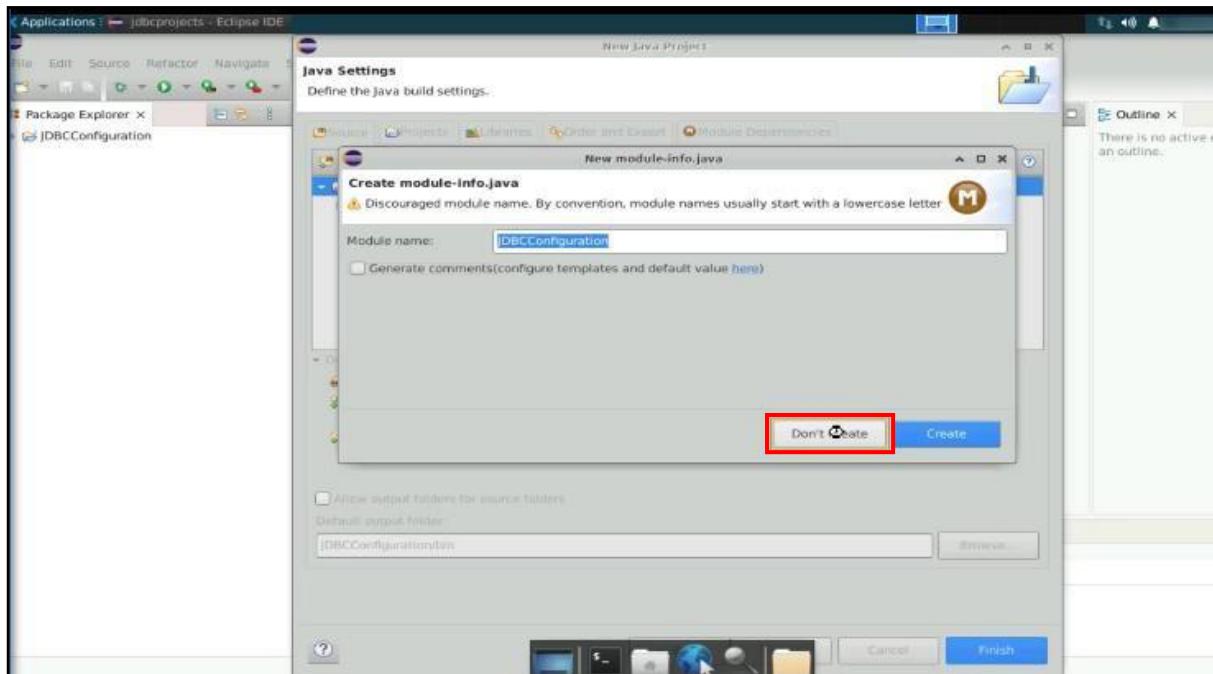
1.4 Select **Java Project** and click on **Next**



1.5 Name the project **JDBCConfiguration** and click on **Finish**

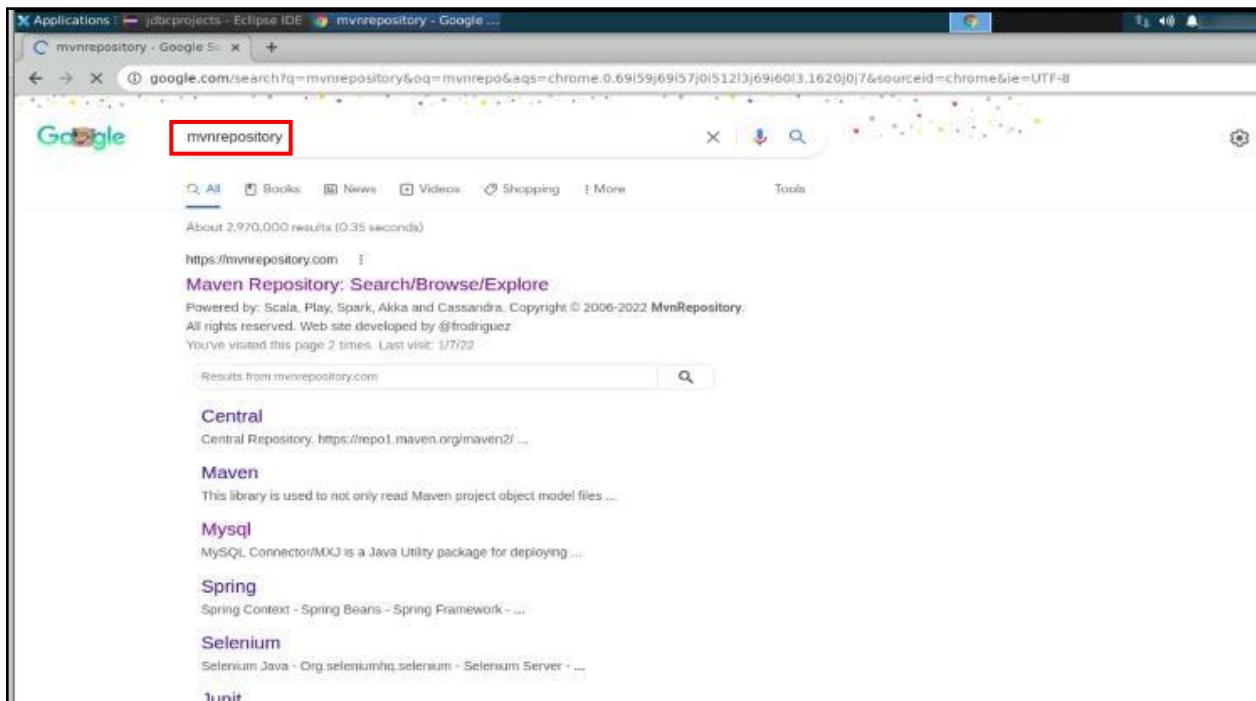


1.6 Click on the **Don't Create** option as there is no need to create a module, as shown in the screenshot below:

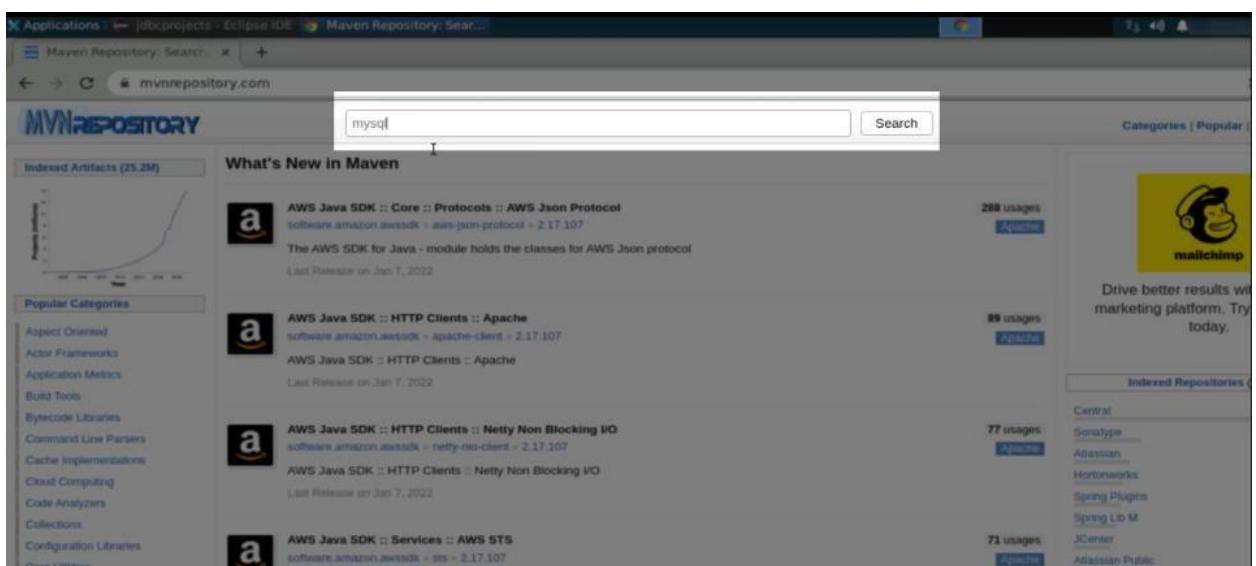


Step 2: Download the MySQL Connector/J library

2.1 Open the preferred browser, search for **mvnrepository**, and click the first link of the Maven Repository



2.2 Search for **mysql** in the search bar



2.3 Select **MySQL Connector/J** as shown in the screenshot below:

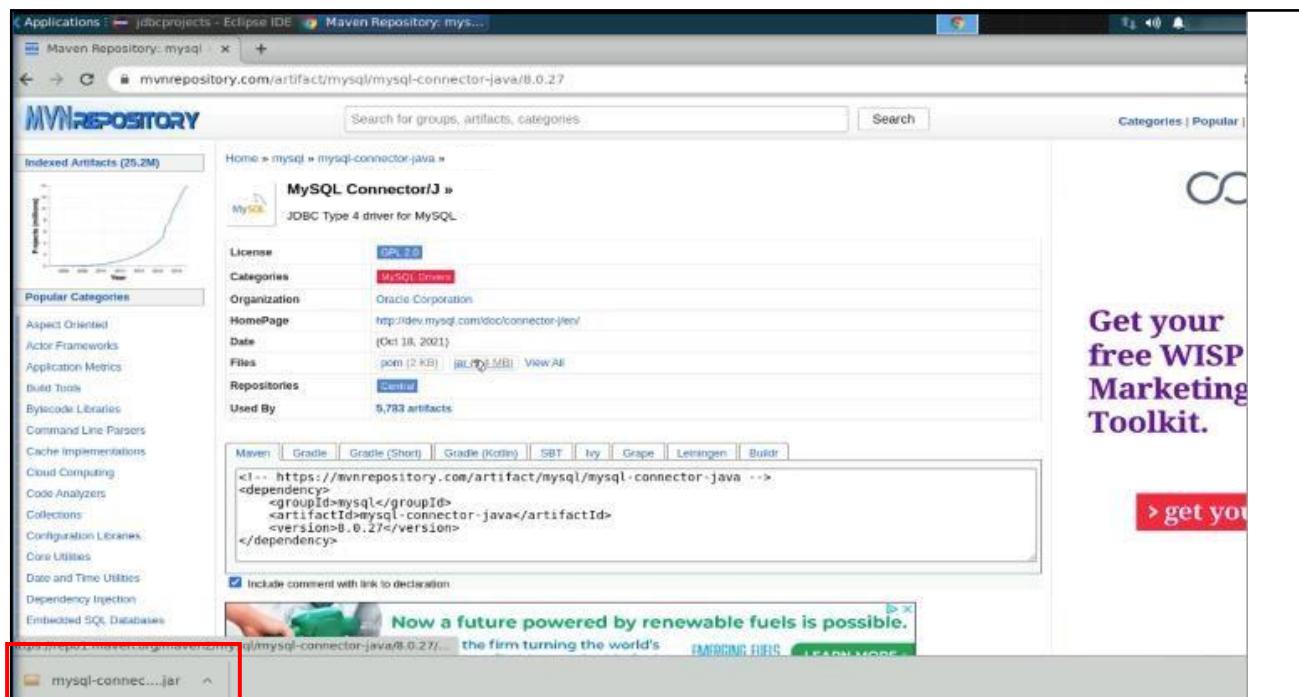
The screenshot shows a web browser displaying the Maven Repository search results for 'mysql'. The URL in the address bar is `mvnrepository.com/search?q=mysql`. On the left, there's a sidebar titled 'Repository' listing various sources: Central (1.0k), Sonatype (96), JCenter (45), Clojars (33), OpenHAB (24), XWiki Releases (22), Spring Plugins (19), and Mulesoft (18). Below that is a 'Group' section with com.github (112), io.github (100), org.apache (70), and org.eclipse (22). The main content area is titled 'Found 1340 results' and shows a list of packages. The first item, '1. MySQL Connector/J com.mysql > mysql-connector-j', is highlighted with a red box. It has a small MySQL logo icon, the version '658 usages', and a brief description: 'MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...'. It also mentions the 'Last Release on Apr 30, 2024'. The second item in the list is '2. MySQL Connector Java 7,552 usages'.

2.4 Check for the latest connector version, for example **8.4.0** as shown in the screenshot below:

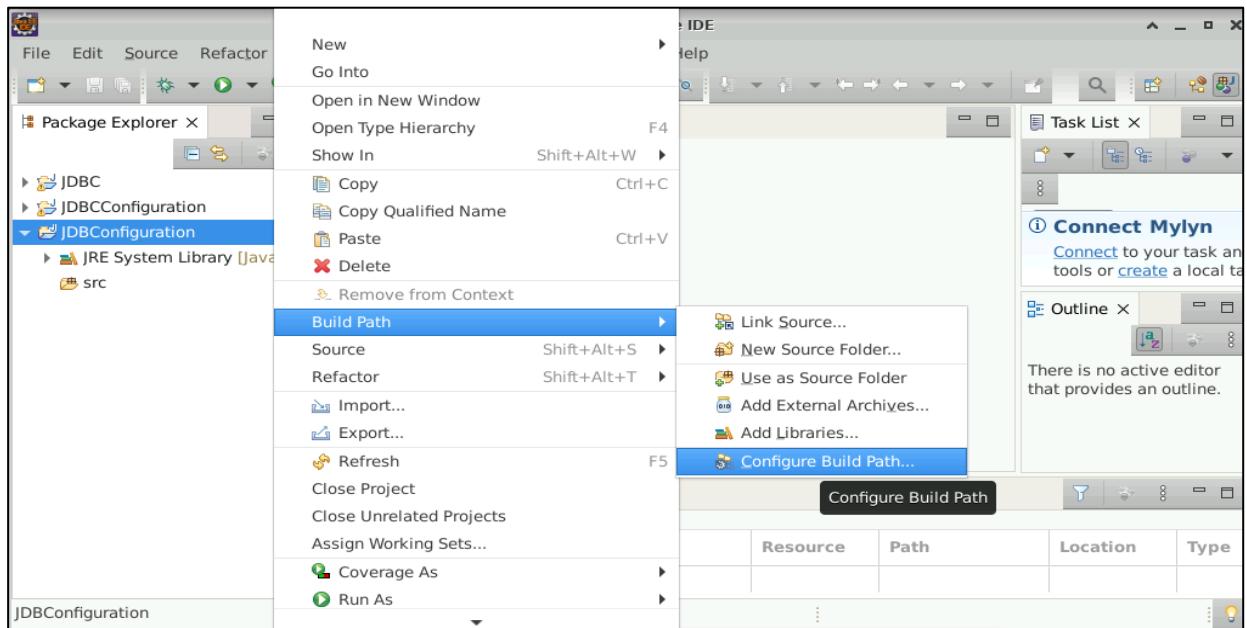
The screenshot shows the Maven Repository page for the MySQL Connector-J artifact. The URL in the address bar is `mvnrepository.com/artifact/com.mysql/mysql-connector-j`. On the left, there's a sidebar with various Java-related categories. The main content area displays the artifact's details: Tags (database, sql, jdbc, driver, connector, rdbms, mysql, connection), Ranking (#744 in MvnRepository, #9 in JDBC Drivers), and Usage (658 artifacts). Below this is a table showing versions available in different repositories. The 'Central' repository section is highlighted with a red border around the row for version 8.4.0.

Central (7) Redhat GA (1) Redhat EA (1)					
Version	Vulnerabilities	Repository	Usages	Date	
8.4.x	8.4.0	Central	41	Apr 30, 2024	
8.3.x	8.3.0	Central	242	Jan 16, 2024	
8.2.x	8.2.0	Central	137	Oct 25, 2023	
8.1.x	8.1.0	Central	165	Jul 18, 2023	
	8.0.33	Central	302	Apr 18, 2023	
8.0.x	8.0.32	Central	140	Jan 18,	

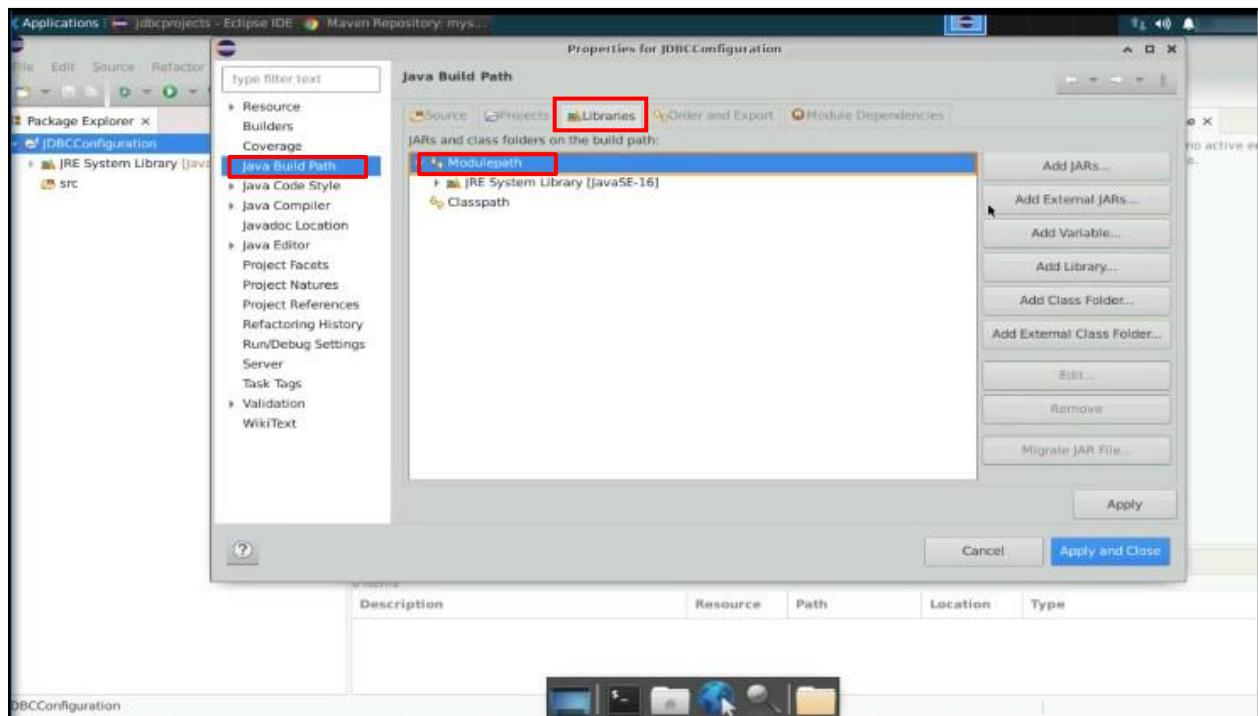
2.5 Click on the latest connector version, then click on JAR file under the files section to download it, as shown in the screenshot below:



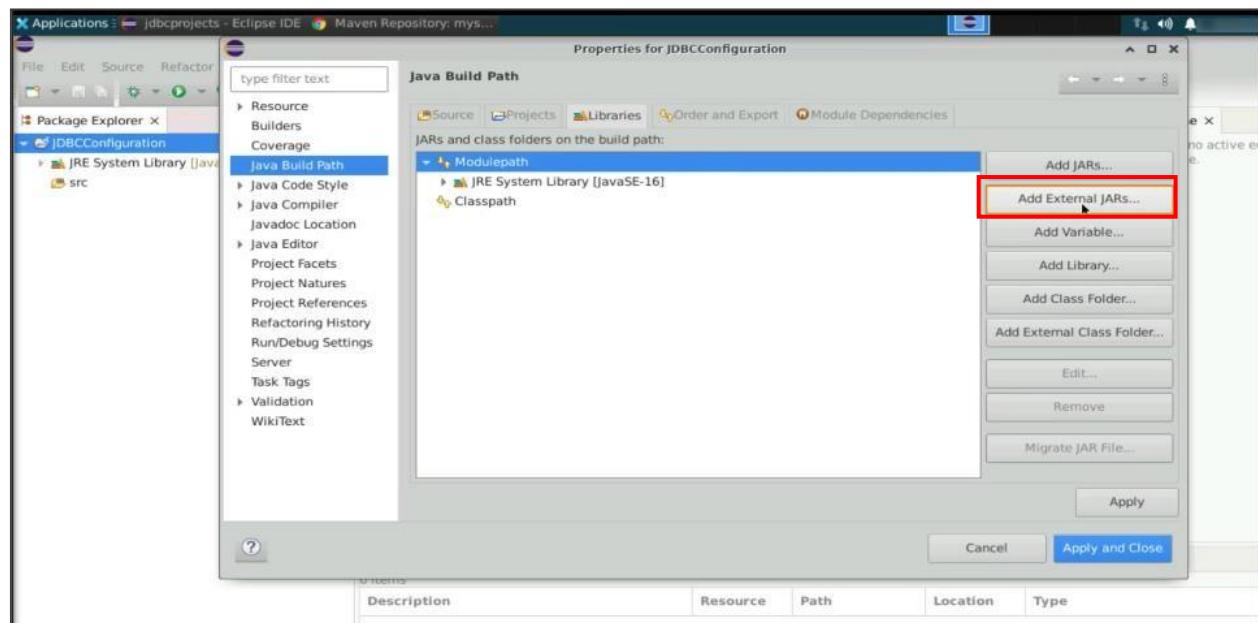
2.6 Right-click on the project file created, select **Build Path**, and choose **Configure Build Path** as shown in the screenshot below:



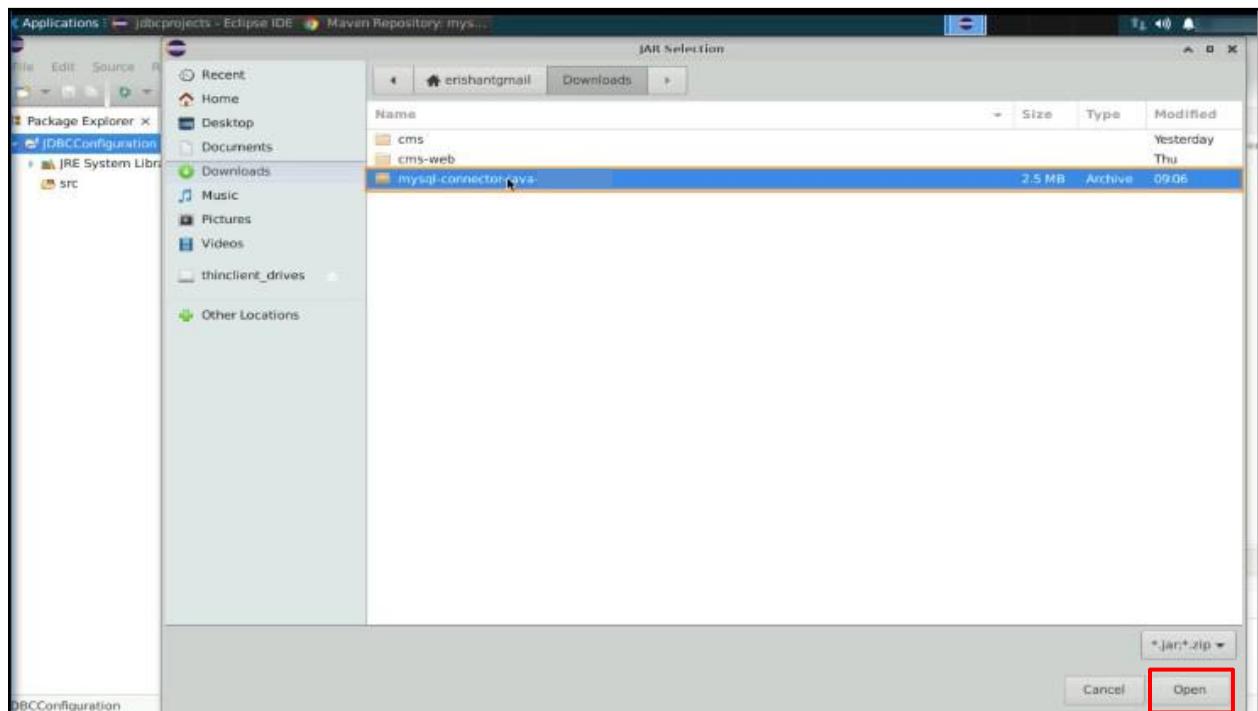
2.7 Go to **Java Build Path > Libraries tab > Modulepath** in the project properties window as shown in the screenshot below:



2.8 Click on the **Add External JARs** button and navigate to the location where the MySQL Connector/J JAR file is saved as shown in the screenshot below:

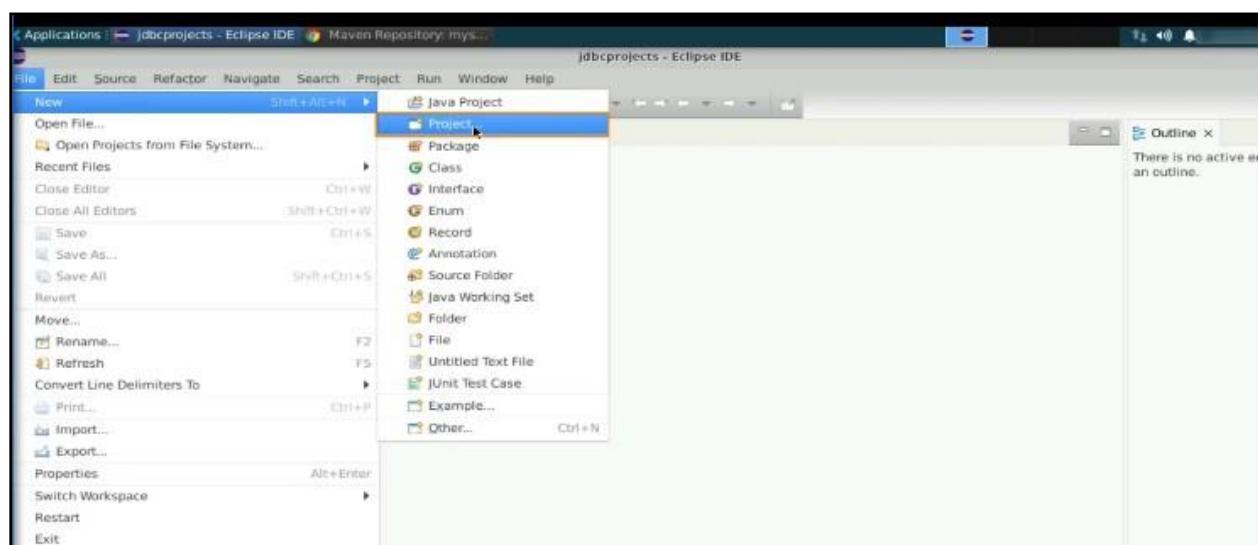


2.9 Select the **mysql-connector-java-8.4.0.jar** file and click **Open** as shown in the screenshot below:

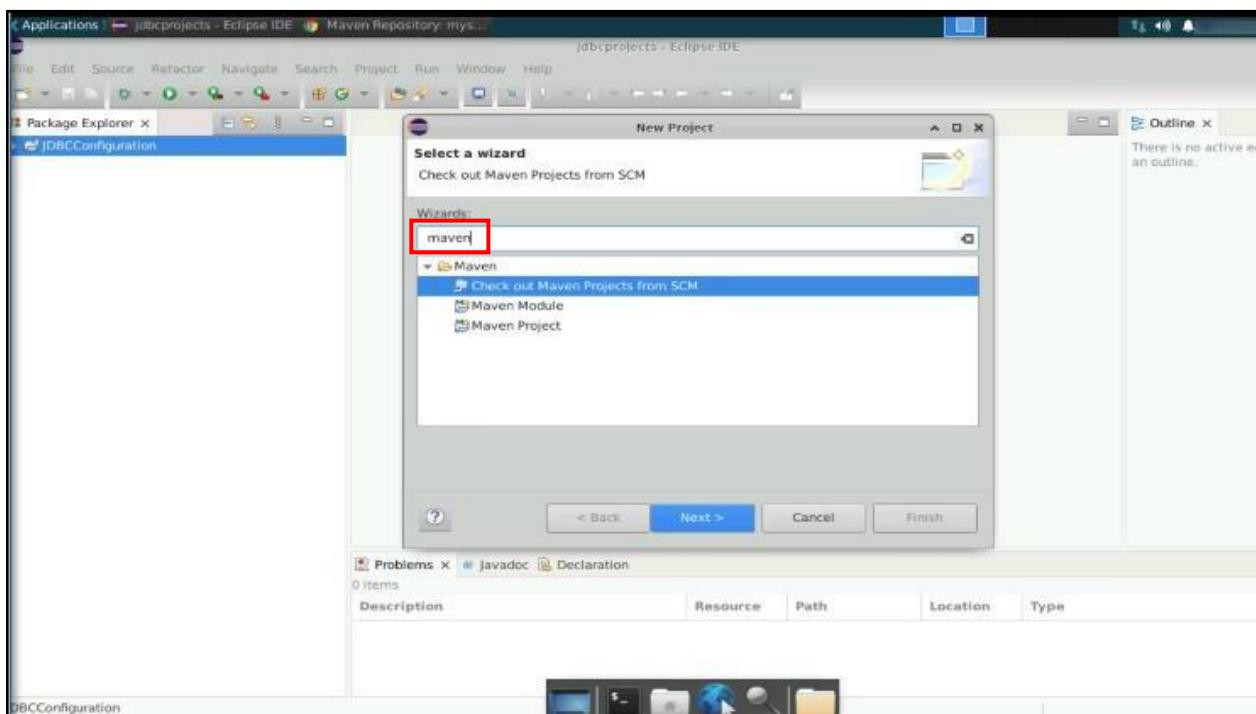


Step 3: Create a Maven project

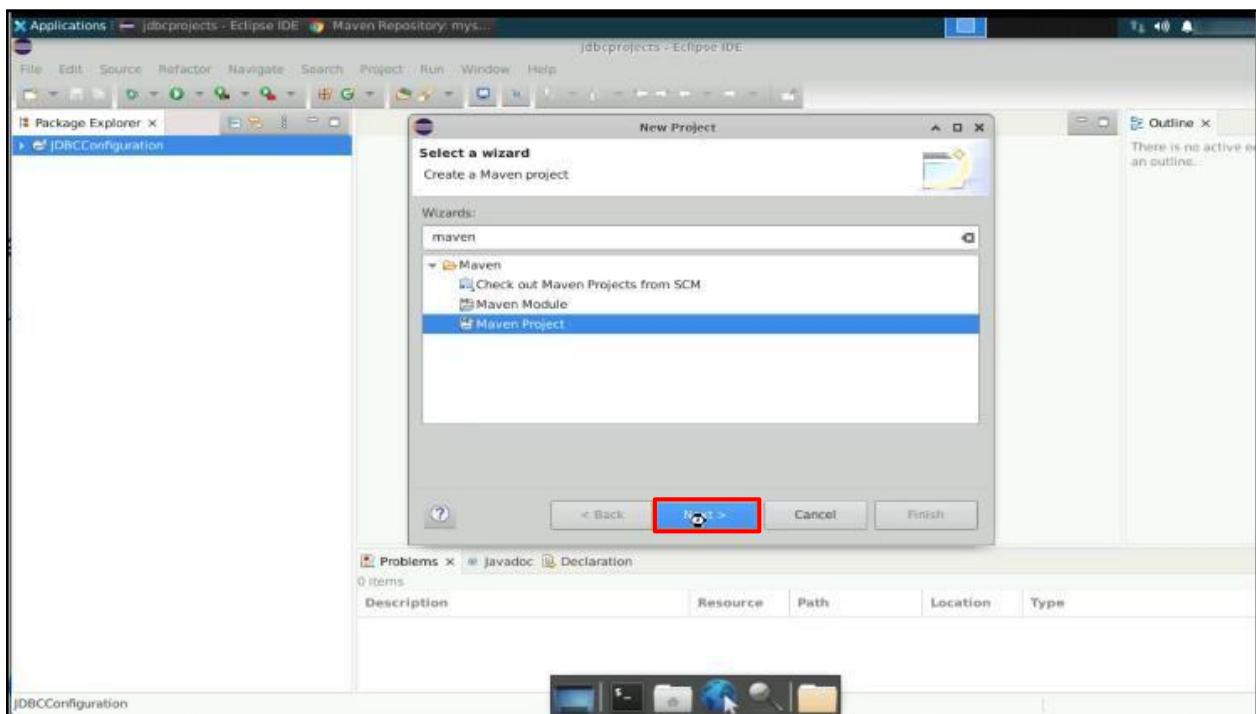
3.1 Create a new project by navigating to **Files > New > Project** as shown in the screenshot below:



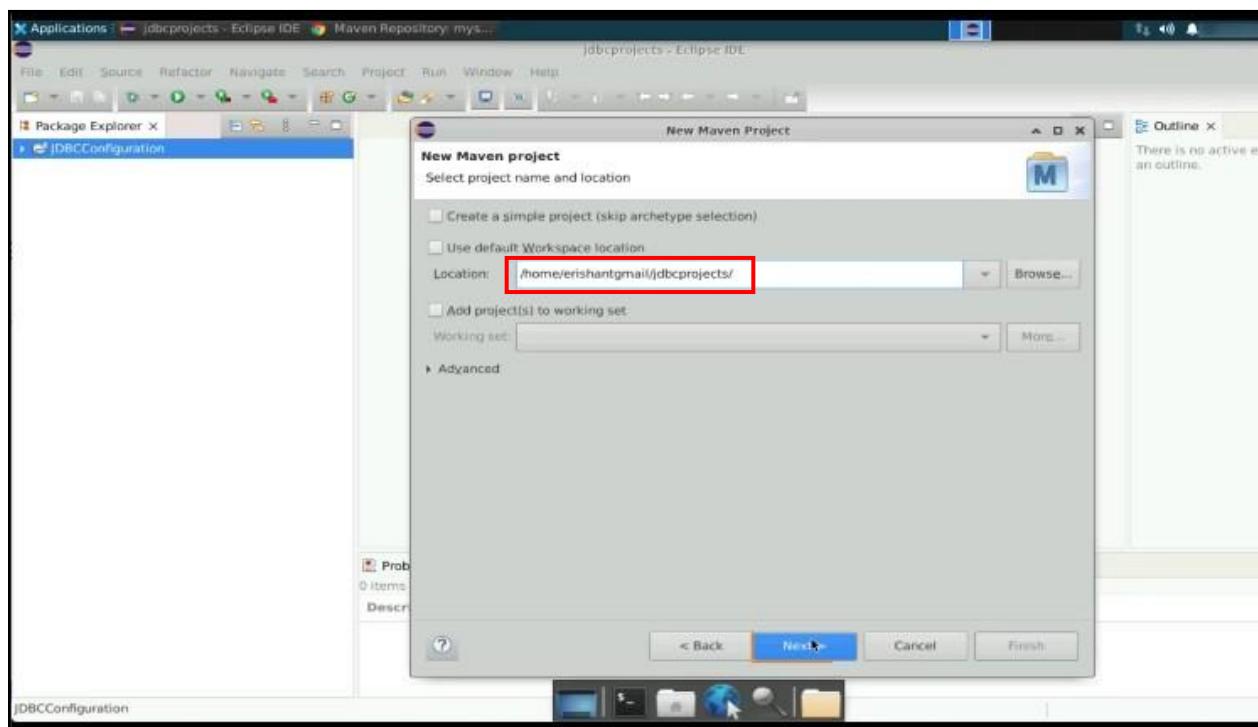
3.2 Search for **Maven** to create a Maven project as shown in the screenshot below:



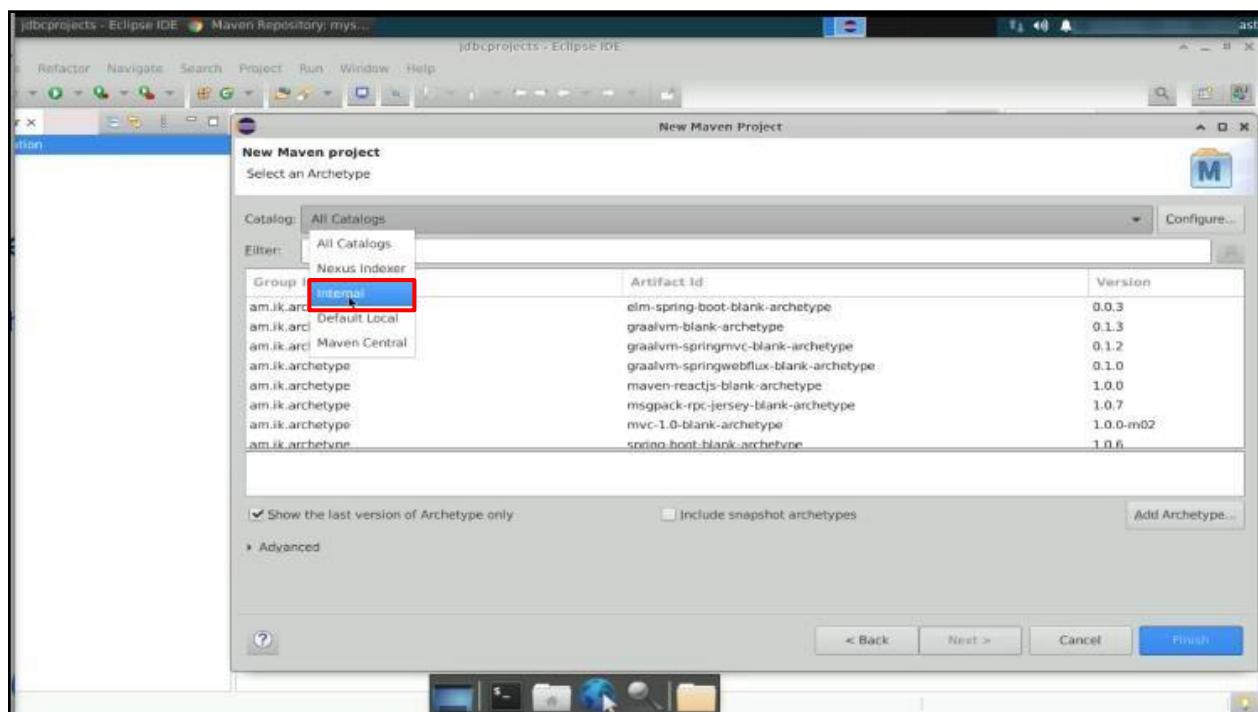
3.3 Click on **Maven Project** and select **Next** as shown in the screenshot below:



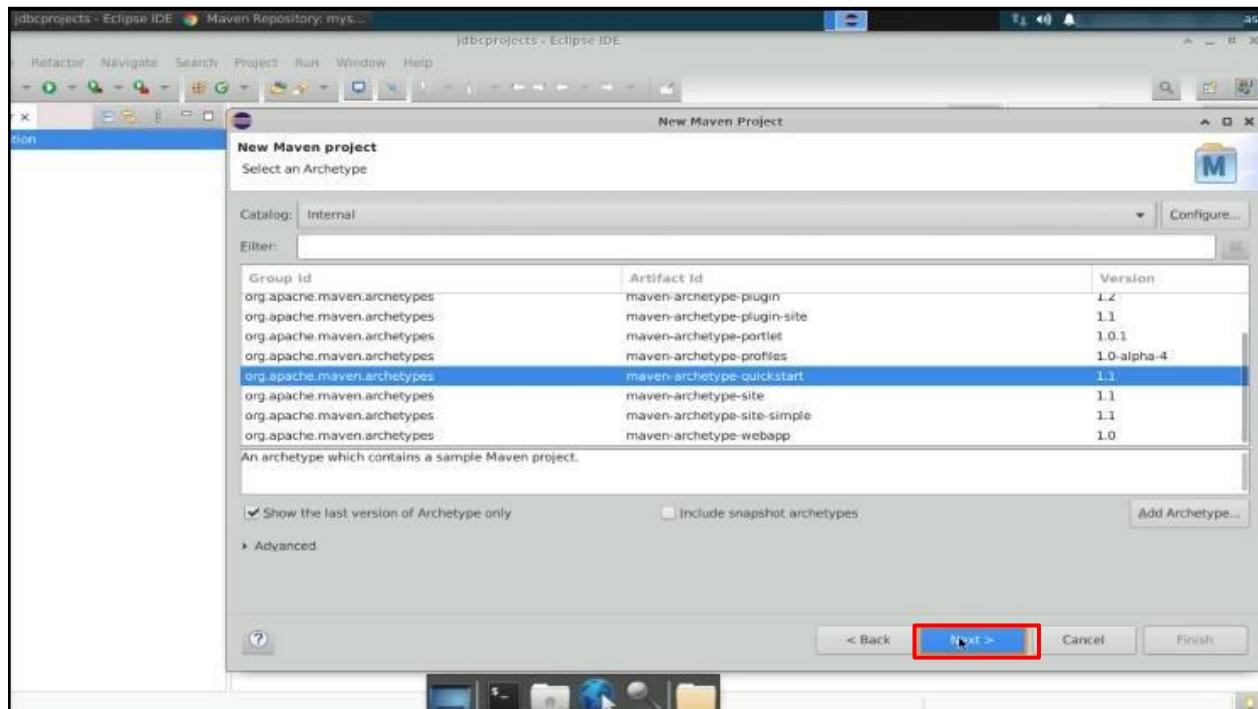
3.4 Set the **workplace location** to match that of the JDBC project, and click **Next** as shown in the screenshot below:



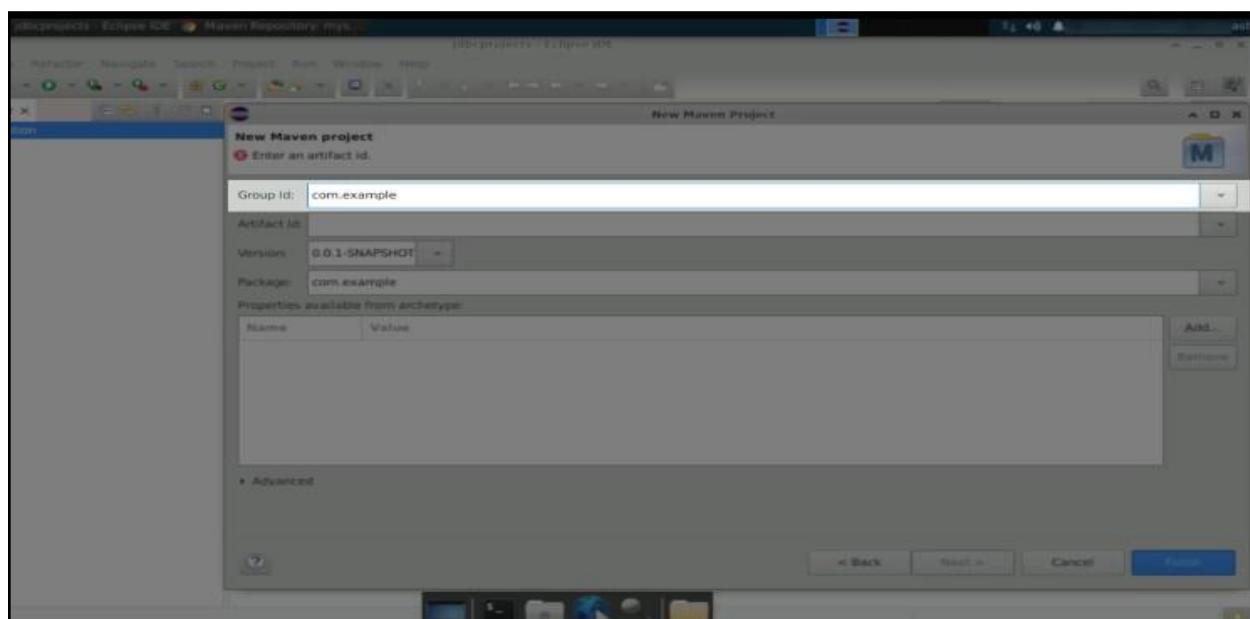
3.5 Select **Internal** from the catalog options as shown in the screenshot below:



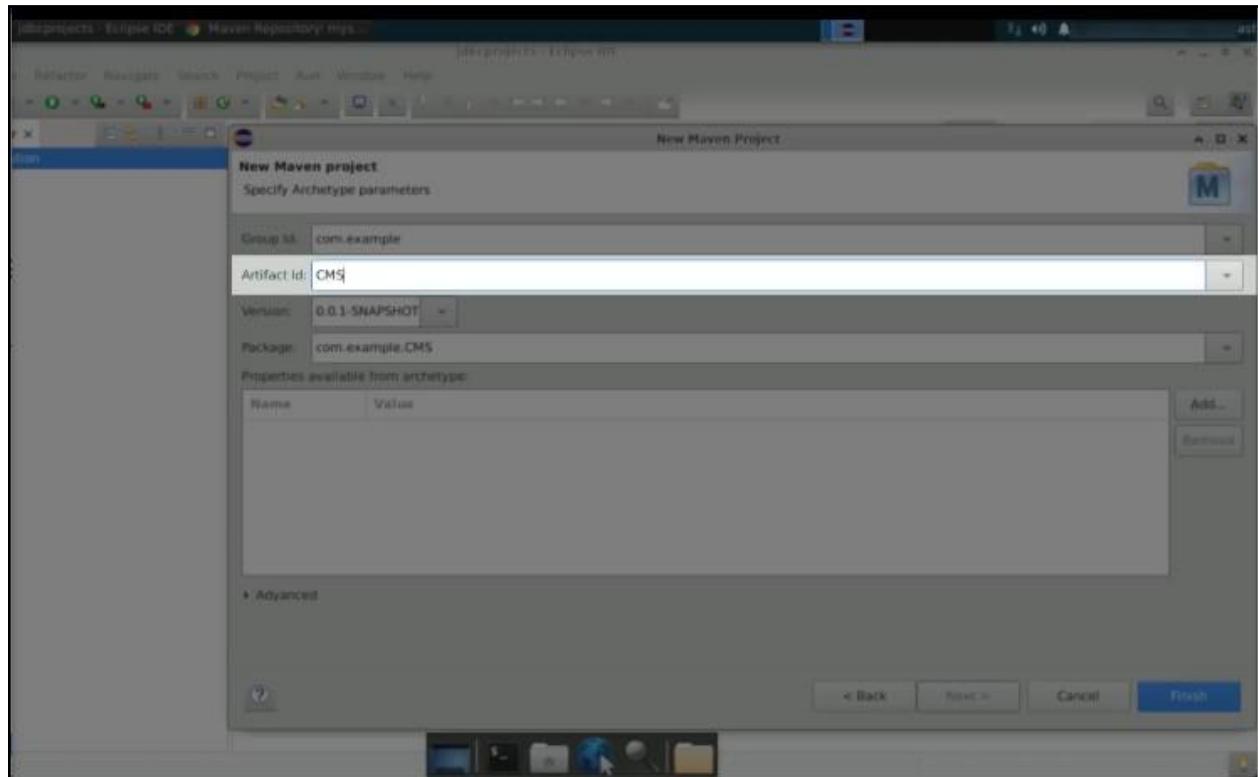
3.6 Select the **maven-archetype-quickstart** option from the options shown, and click on **Next** as shown in the screenshot below:



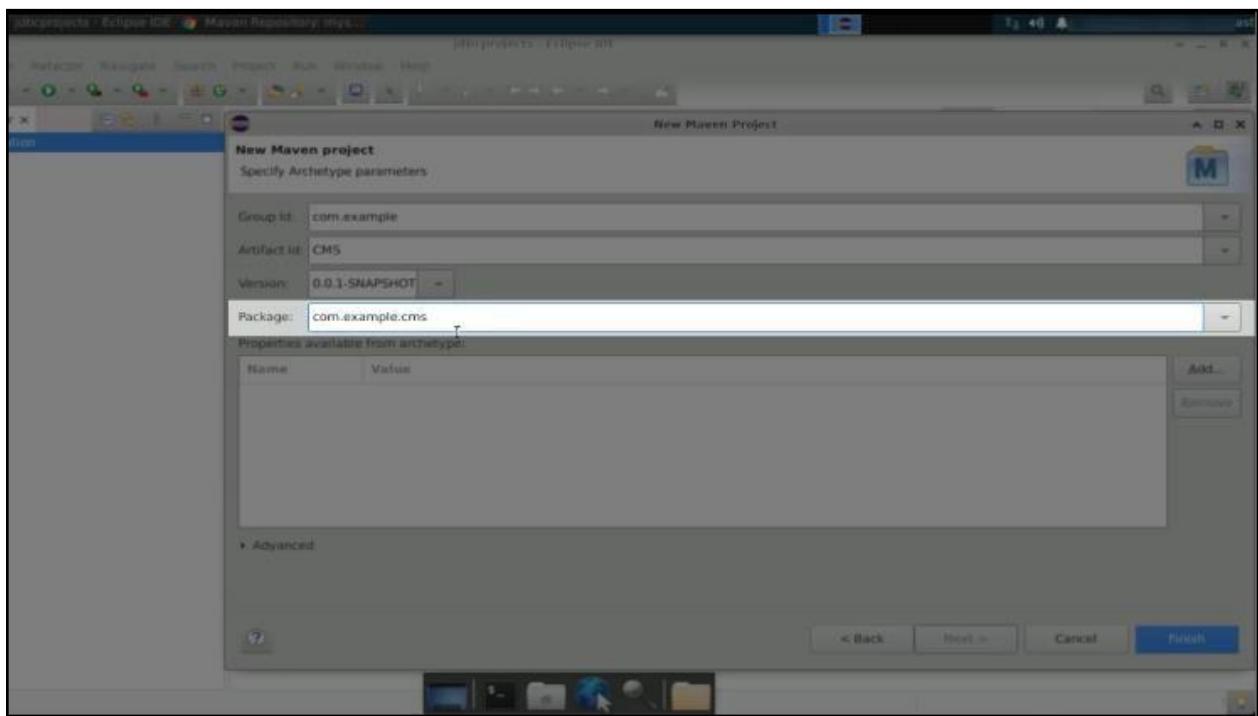
3.7 Provide the **Group Id**, which is your company's domain name in the reverse order:



3.8 Add an **Artifact Id**, which will be your application name, say **CMS** as shown in the screenshot below:

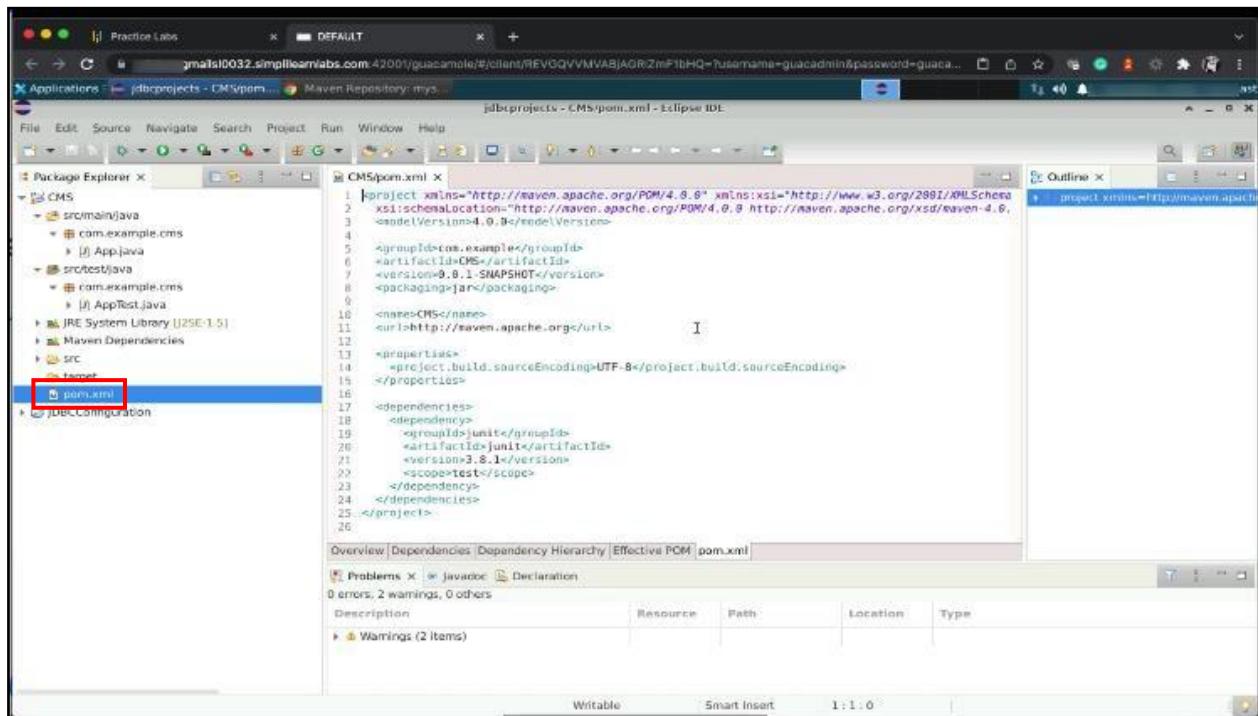


3.9 Add a package name, say **com.example.cms**, and select **Finish** as shown in the screenshot below:



Step 4: Create an interface in com.example.cms

4.1 Go to the **pom.xml** file in the left bar as shown in the screenshot below:



4.2 Return to **mvnrepository** and copy the code for the mysql-connector dependency as shown in the screenshot below:

The screenshot shows the mvnrepository.com website for the MySQL Connector/J artifact. The URL is <https://mvnrepository.com/artifact/mysql/mysql-connector-jar/8.0.27>. The page displays the artifact's details, including its license (Apache 2.0), categories (MySQL, JDBC), organization (Oracle Corporation), homepage, date (Oct 18, 2021), files (jar, 15.4 MB), repositories, and usage statistics (8,783 artifacts). A prominent red box highlights the dependency code in the central text area:

```

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-jar -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-jar</artifactId>
    <version>8.0.27</version>
</dependency>

```

4.3 Paste the dependencies in the **pom.xml** file from line 25 as shown in the screenshot below:

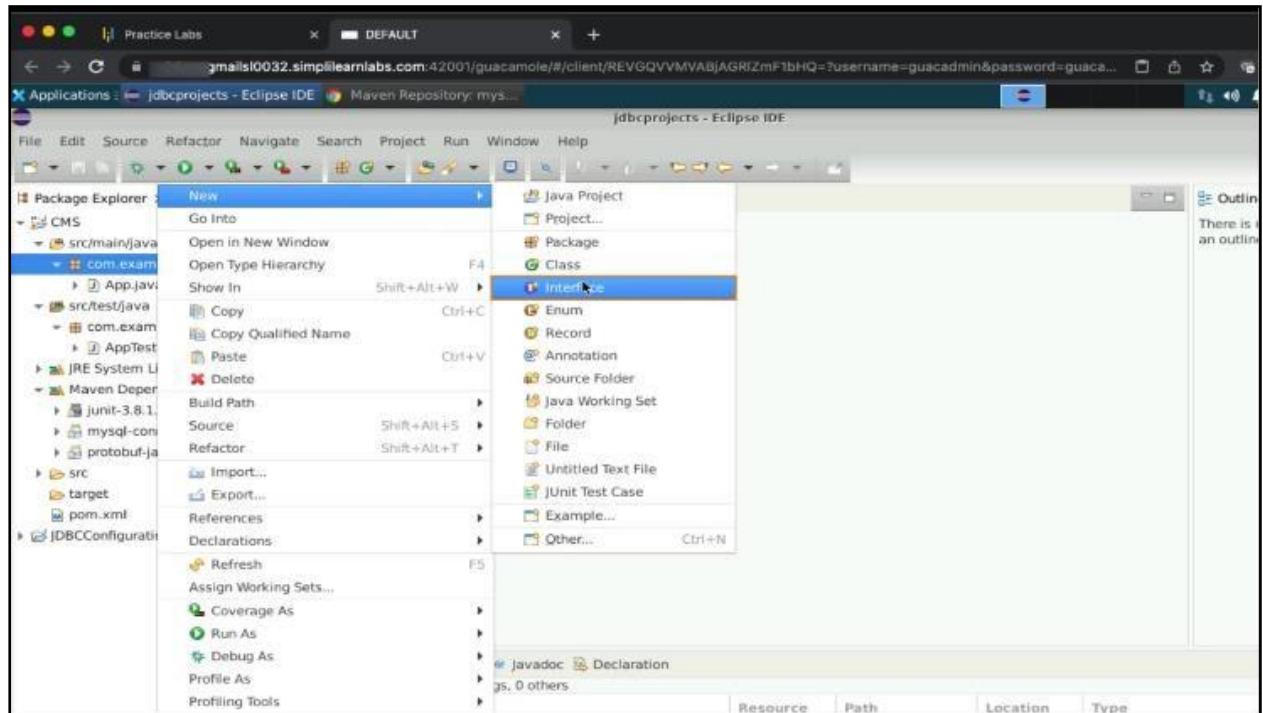
The screenshot shows the Eclipse IDE interface with the Maven Repository plugin. The title bar says "jdbcpatterns - CMS/pom.xml - Eclipse IDE". The left side shows the "Package Explorer" with a project named "CMS" containing "src/main/java", "src/test/java", and "pom.xml". The right side shows the "Outline" view with the "dependencies" section selected. A red box highlights the dependency code in the "pom.xml" editor:

```

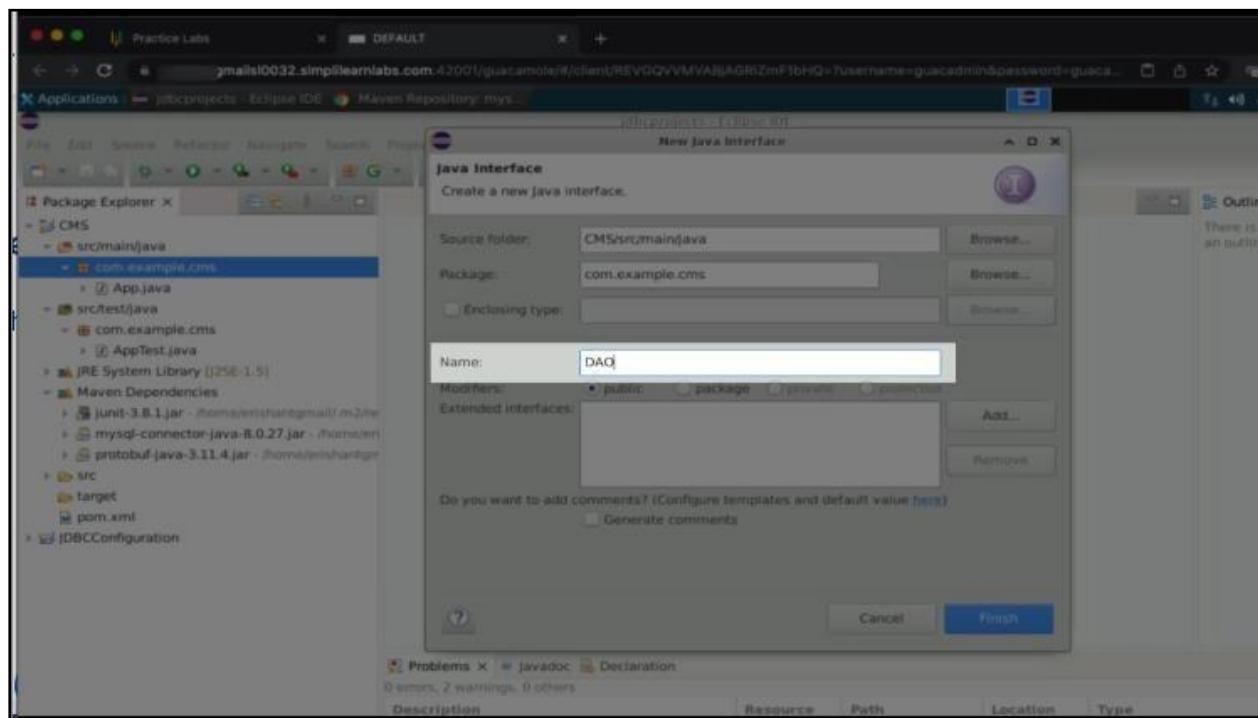
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-jar -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-jar</artifactId>
    <version>8.0.27</version>
</dependency>

```

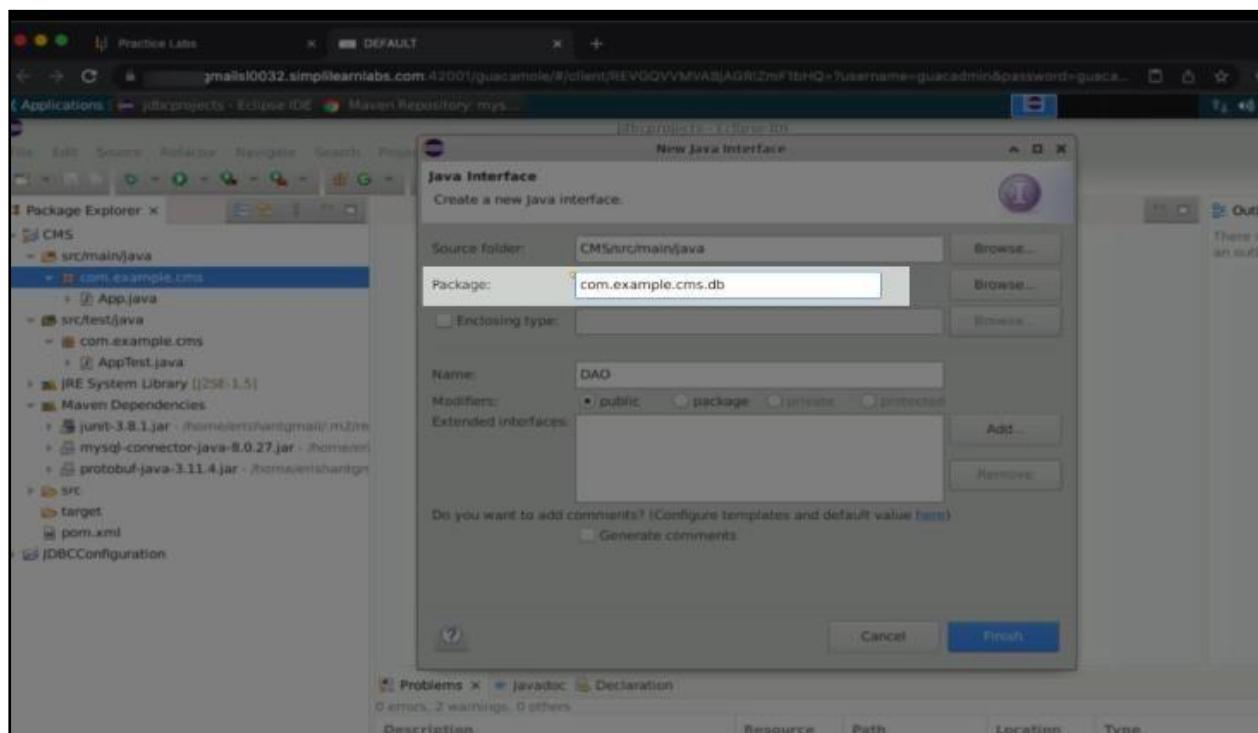
4.4 Right-click on **com.example.cms**, select **New** and create an interface as shown in the screenshot below:



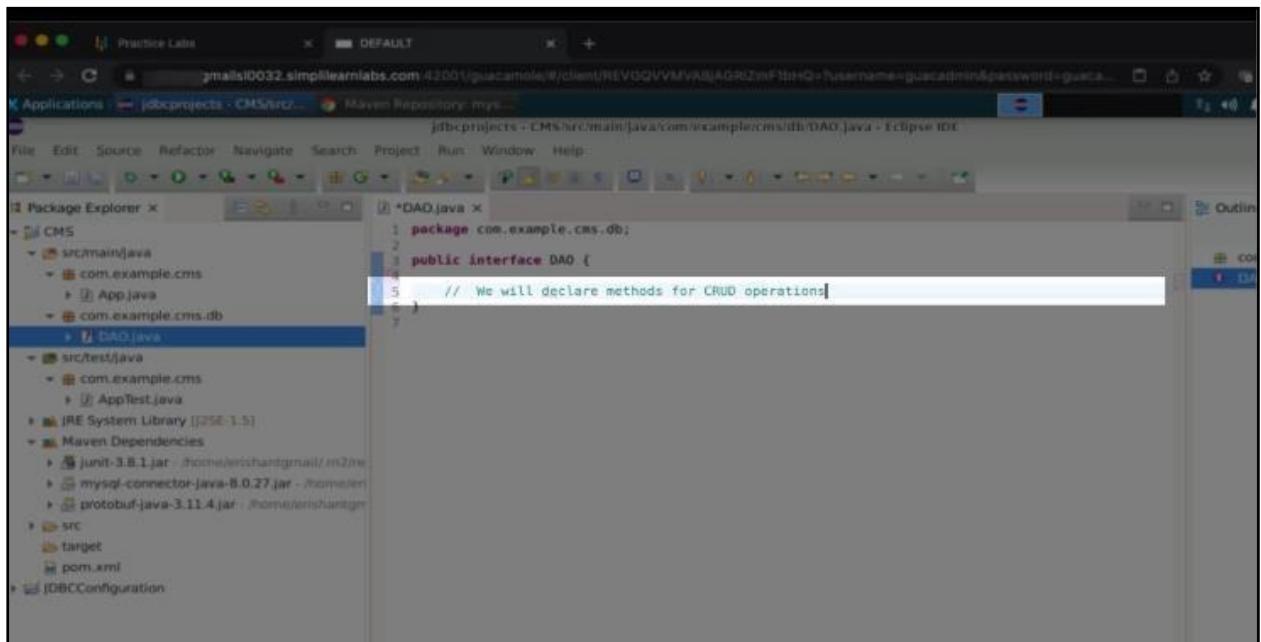
4.5 Provide a name as DAO



4.6 Add a package name, say com.example.cms.db, and click on Finish



4.7 Define the methods for CRUD operations



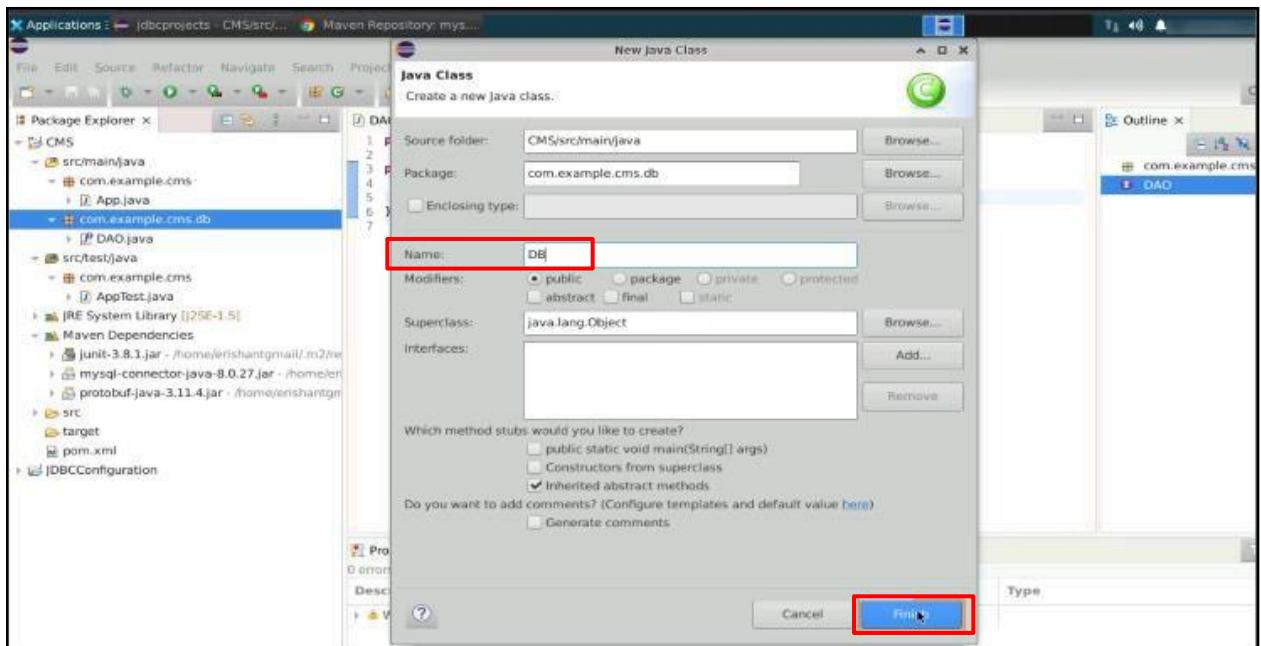
```

1 package com.example.cms.db;
2
3 public interface DAO {
4     // We will declare methods for CRUD operations
5 }

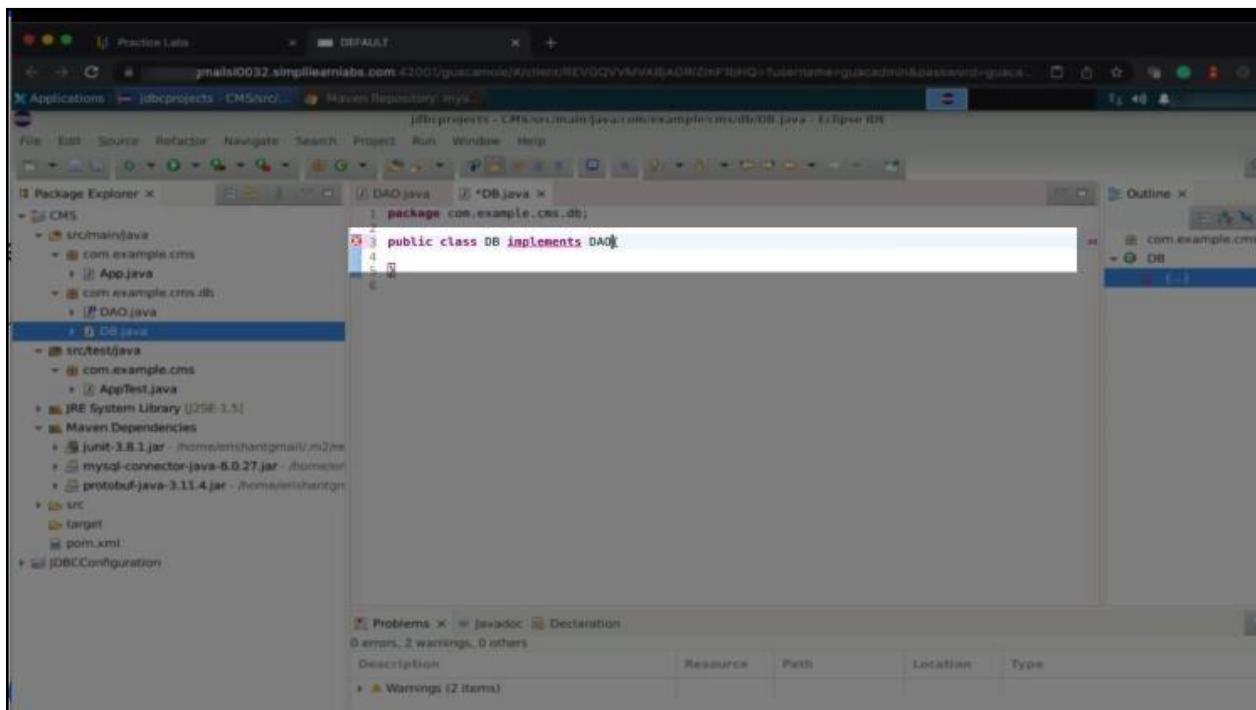
```

Step 5: Create the DB class

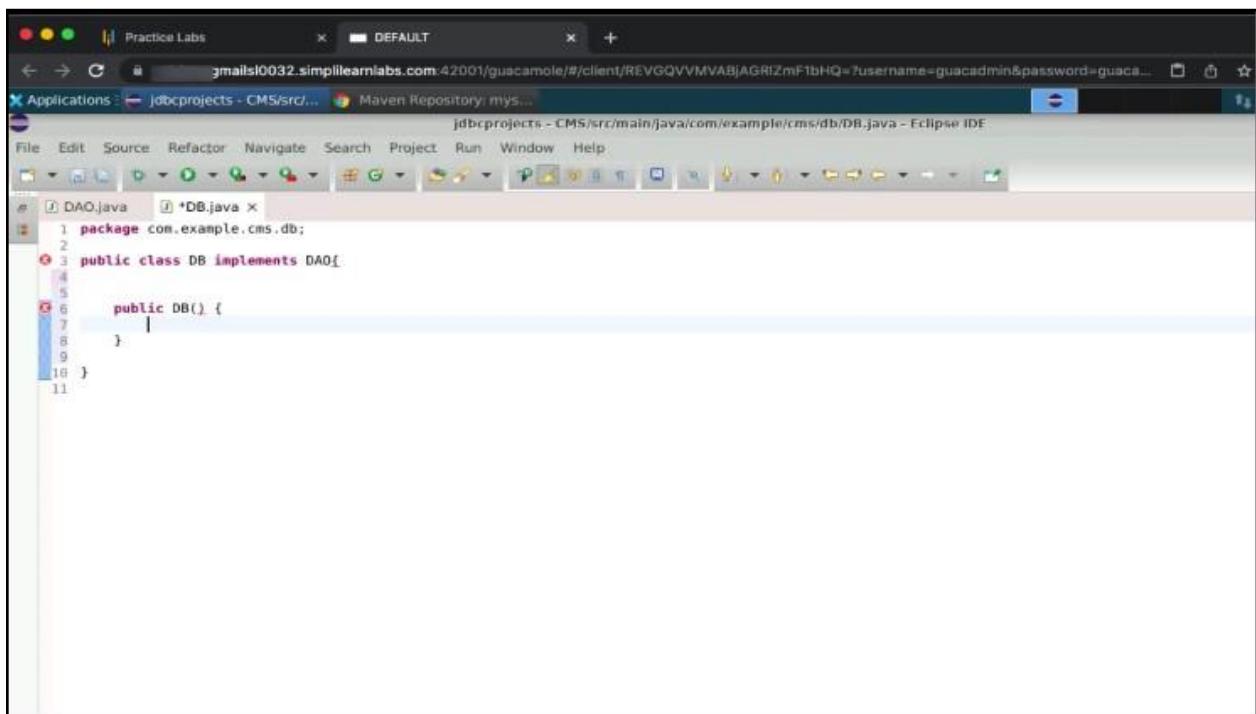
5.1 Create a new class named **DB**, and click on **Finish**



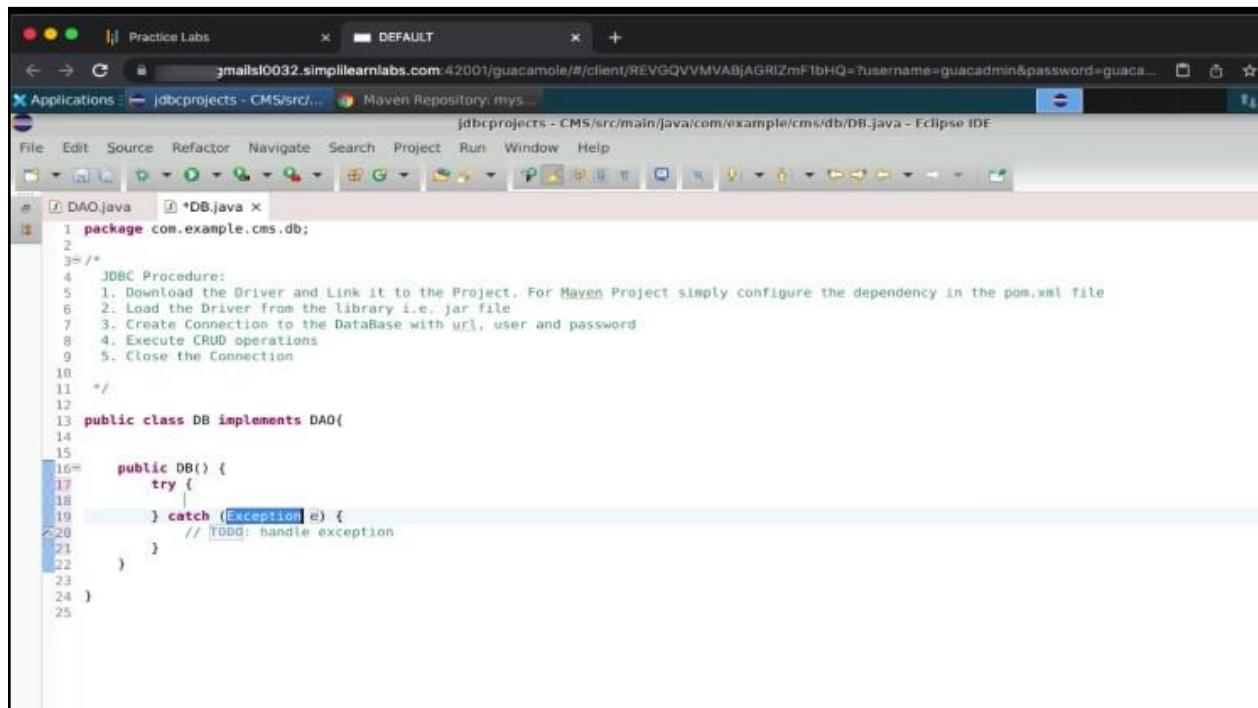
The **DB** class is the implementation file of the **DAO** interface.



5.2 Create a constructor in the **DB** class as shown in the screenshot below:

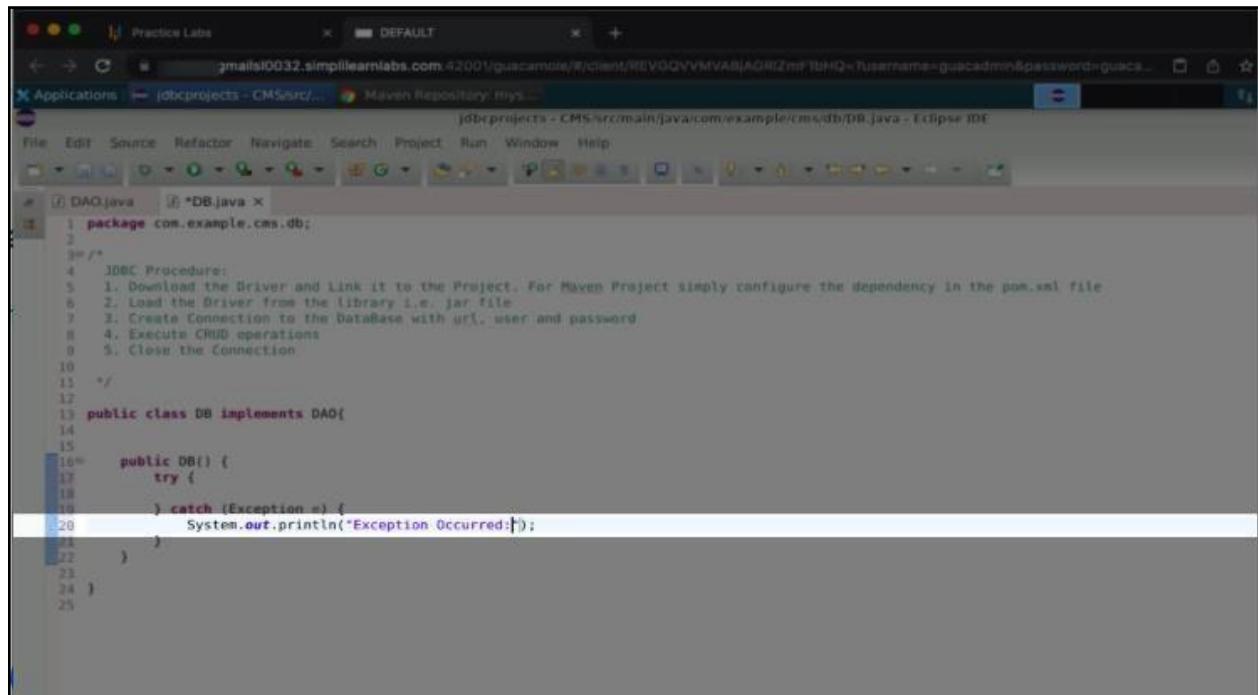


5.3 Use a try-catch block in the constructor to configure the database



```
1 package com.example.cms.db;
2
3/*
4 * JDBC Procedure:
5 * 1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
6 * 2. Load the Driver from the library i.e. jar file
7 * 3. Create Connection to the DataBase with url, user and password
8 * 4. Execute CRUD operations
9 * 5. Close the Connection
10 */
11
12 public class DB implements DAO{
13
14
15    public DB() {
16        try {
17
18            } catch (Exception e) {
19                // TODO: handle exception
20            }
21        }
22    }
23
24 }
```

5.4 Manage the Exception in the catch block



```
1 package com.example.cms.db;
2
3/*
4 * JDBC Procedure:
5 * 1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
6 * 2. Load the Driver from the library i.e. jar file
7 * 3. Create Connection to the DataBase with url, user and password
8 * 4. Execute CRUD operations
9 * 5. Close the Connection
10 */
11
12 public class DB implements DAO{
13
14
15    public DB() {
16        try {
17
18            } catch (Exception e) {
19                System.out.println("Exception Occurred!");
20            }
21        }
22    }
23
24 }
```

5.5 Add an API named Class

```

1 package com.example.cms.db;
2
3/*
4 JDBC Procedure:
5 1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
6 2. Load the Driver from the library i.e. jar file
7 3. Create Connection to the DataBase with url, user and password
8 4. Execute CRUD operations
9 5. Close the Connection
10 */
11
12
13 public class DB implements DAO{
14
15     public DB() {
16         try {
17             Class.forName("");
18         } catch (Exception e) {
19             System.out.println("Exception Occurred: "+e);
20         }
21     }
22
23
24 }
25

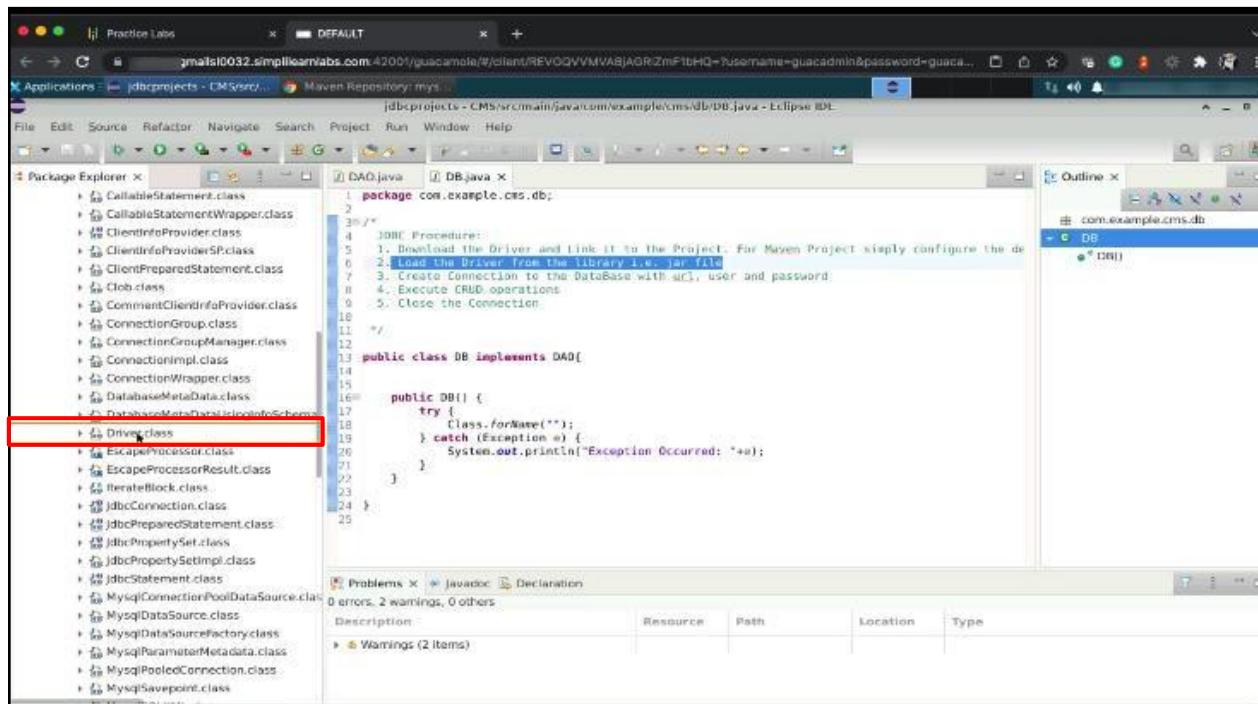
```

5.6 Open mysql-connector-java-8.4.0.jar and find the package containing the driver class

The screenshot shows the Eclipse IDE interface with the following details:

- Project Bar:** Practice Labs, DEFAULT, Applications: jdbcprojects - CMS/src/...
- Maven Repository:** m...
- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Code Editor:** DAO.java, DB.java (selected). Both files contain the same Java code as shown in the previous screenshot.
- Outline View:** Shows the class structure: com.example.cms.db > DB
- Package Explorer:** Shows the project structure and dependencies. Under "Maven Dependencies", there is a dependency on "mysql-connector-java-8.0.22.jar". The package "com.mysql.cj.jdbc" is expanded, showing its contents: JDBCAdmin, JDBCException, JDBCXA, JDBCIntegration, JDBCInterceptors, JDBCJMX, JDBCResult, JDBCUtil, Log, Protocol, ProtocolA, ProtocolAAuthentication, ProtocolResult, and ProtocolX.
- Problems View:** Shows 2 warnings: "Warning (2 items)"
- Bottom Status Bar:** com.mysql.cj.jdbc - /home/erishant@gmail.m2/repository/mysql/mysql-connector-java/8.0.27/mysql-connector-java-8.0.27.jar

5.7 Select the **Driver** class as shown in the screenshot below:



The screenshot shows the Eclipse IDE interface. In the Package Explorer view on the left, a red box highlights the 'Driver' class under the 'com.mysql.cj.jdbc' package. The code editor in the center contains a Java file named 'DB.java' which implements a DAO interface. The 'Outline' view on the right shows the class structure.

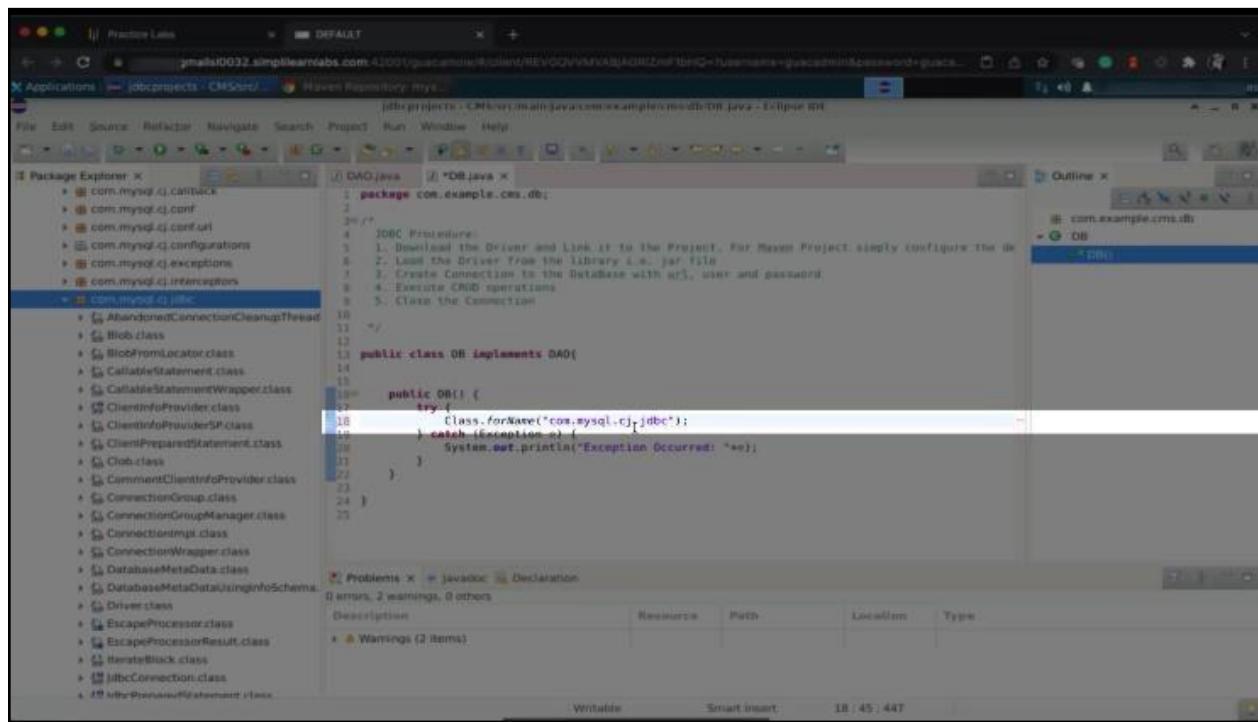
```

package com.example.cms.db;
import java.sql.*;

public class DB implements DAO{
    public DB() {
        try {
            Class.forName("");
        } catch (Exception e) {
            System.out.println("Exception Occurred: " + e);
        }
    }
}

```

5.8 Copy and paste the package name as shown in the screenshot below:



The screenshot shows the Eclipse IDE interface. In the Package Explorer view on the left, a red box highlights the 'Driver' class under the 'com.mysql.cj.jdbc' package. The code editor in the center contains a Java file named 'DB.java' which implements a DAO interface. The 'Outline' view on the right shows the class structure.

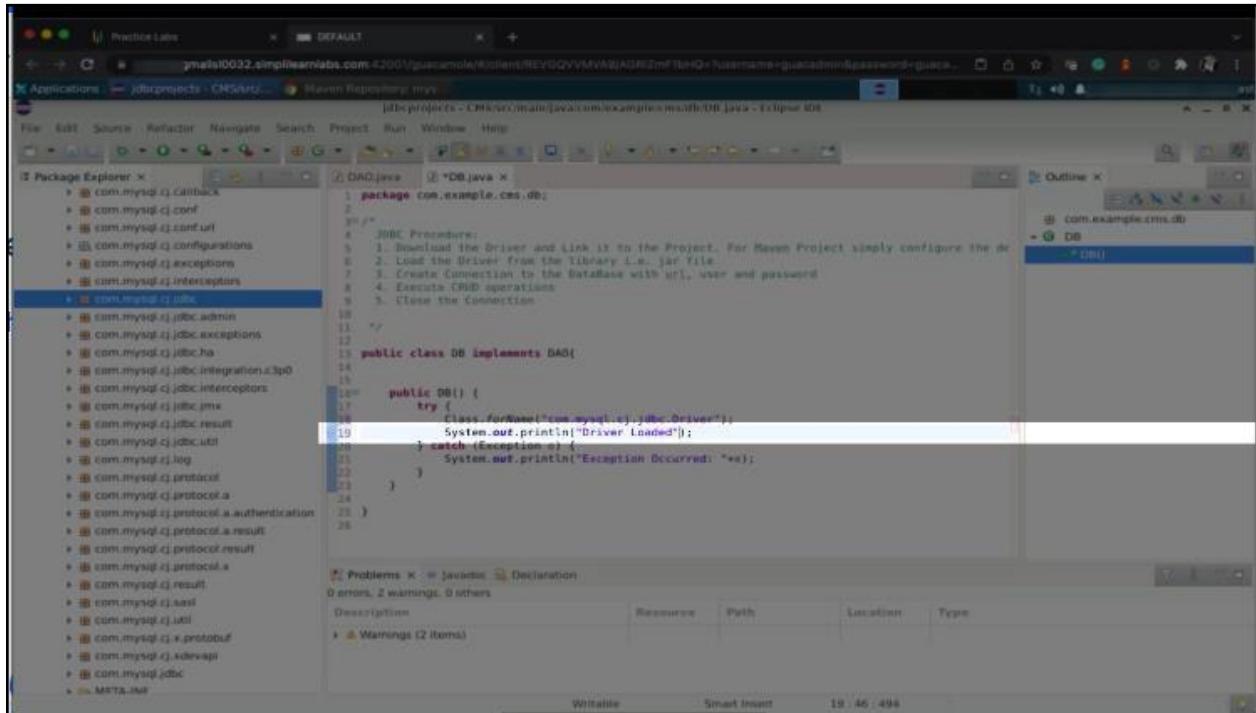
```

package com.example.cms.db;
import java.sql.*;

public class DB implements DAO{
    public DB() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (Exception e) {
            System.out.println("Exception Occurred: " + e);
        }
    }
}

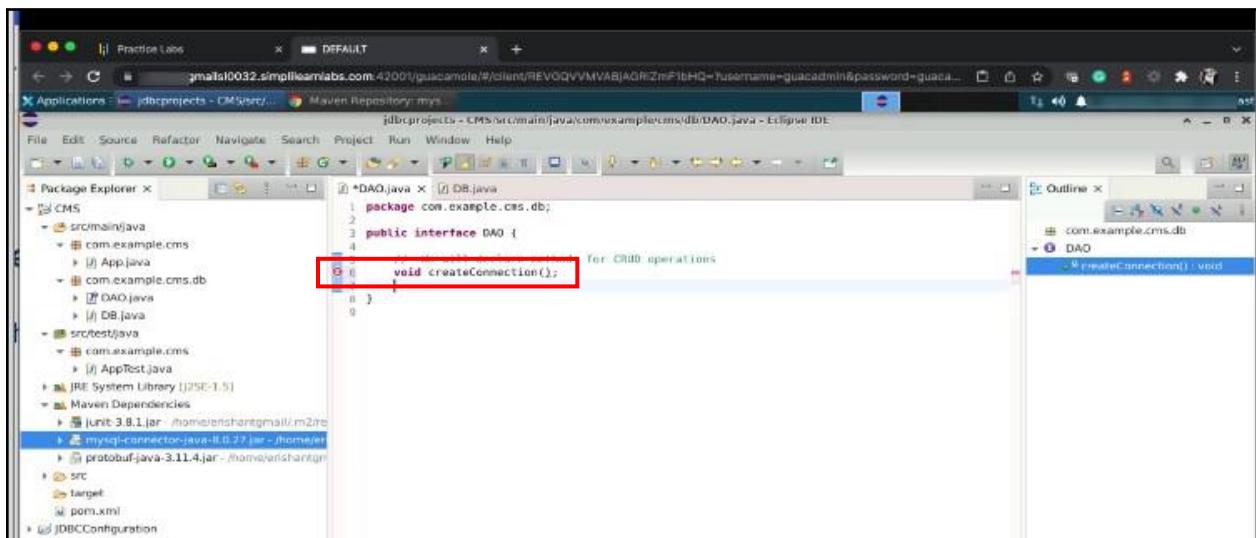
```

5.9 Include a print statement to load the driver as shown in the screenshot below:

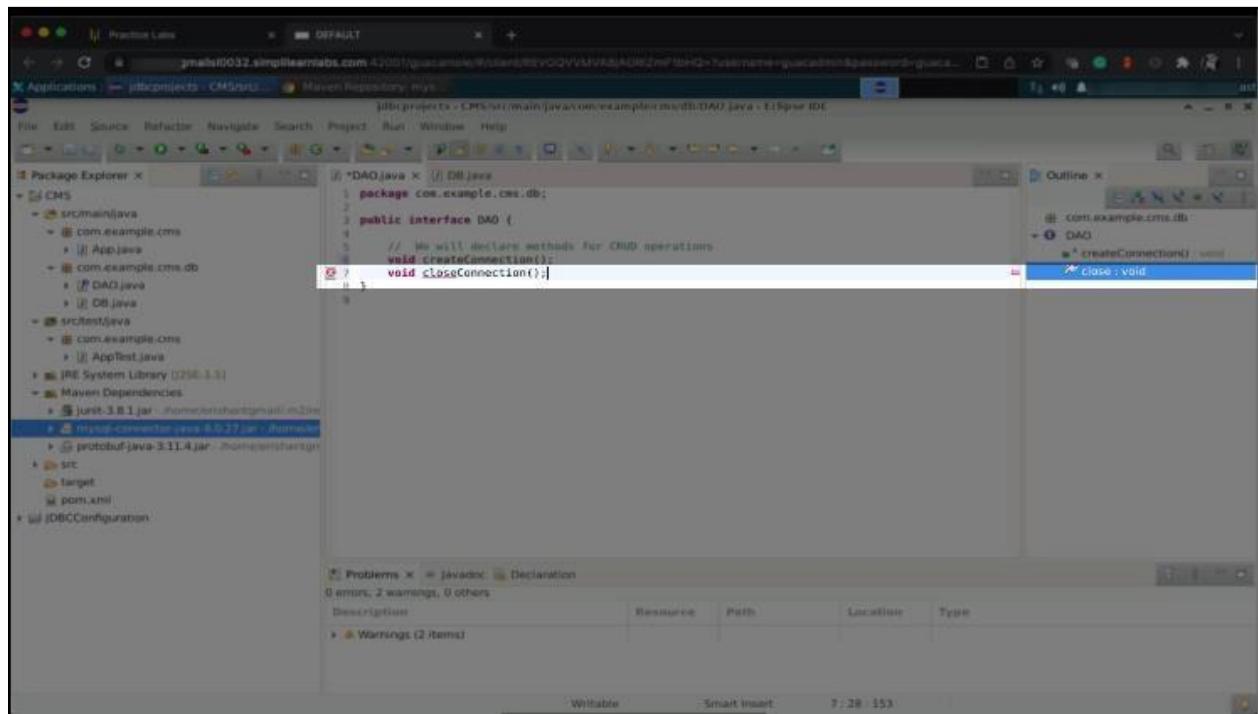


Step 6: Create a connection in the DAO.java file

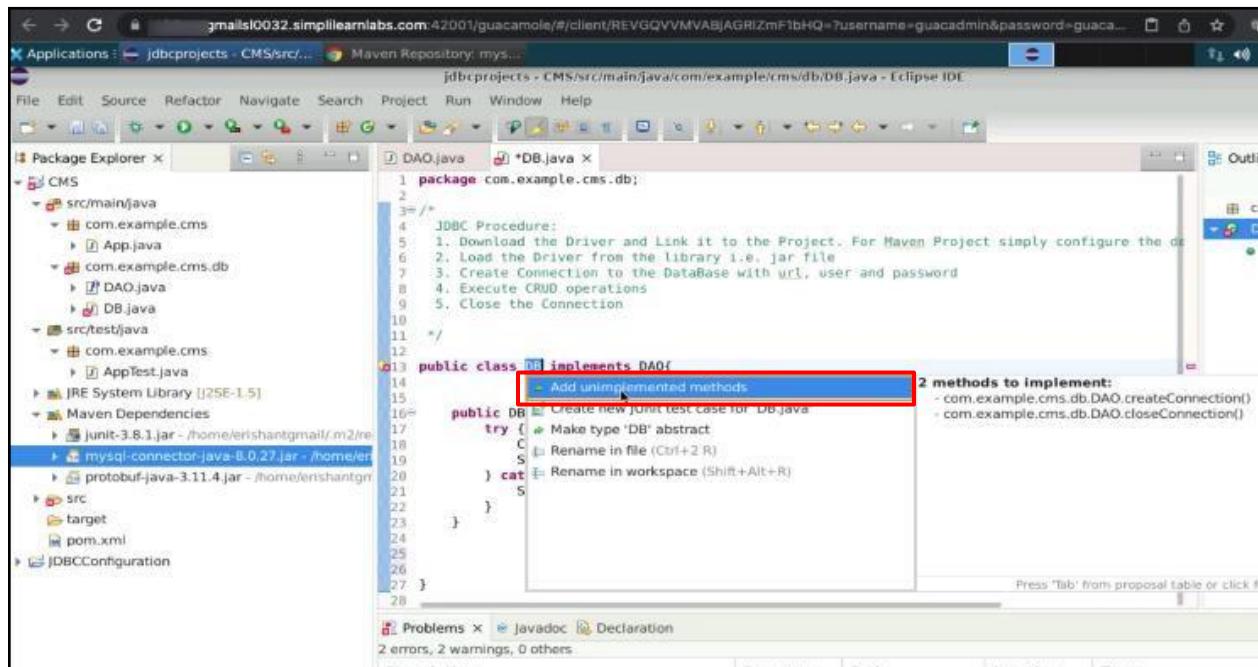
6.1 Create a connection in the **DAO.java** file as shown in the screenshot below:



6.2 Add the `closeConnection()` method in the same file:



6.3 Select Add unimplemented methods in the DB.java file as shown in the screenshot below:



6.4 Add an API from JDBC

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - jdbccprojects - CMSsrc - Maven Repository.mys - jdbccprojects + CMSsrc/main/java/com/example/cms/db/DB.java - Eclipse IDE
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse toolbar icons.
- Package Explorer:** Shows the project structure:
 - CMS
 - src/main/java
 - com.example.cms (App.java)
 - com.example.cms.db (DAO.java, DB.java)
 - src/test/java
 - com.example.cms (AppTest.java)
 - JRE System Library [j2SE-1.5]
 - Maven Dependencies
 - junit-3.8.1.jar
 - mysql-connector-java-8.0.27.jar
 - protobuf-java-3.11.4.jar
 - src
 - target
 - pom.xml
- DB.java Content:** The code implements a DAO interface and handles database connections.

```
1 package com.example.cms.db;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DB implements DAO{
8     Connection connection;
9
10    public DB() {
11        try {
12            Class.forName("com.mysql.cj.jdbc.Driver");
13            System.out.println("Driver Loaded");
14        } catch (Exception e) {
15            System.out.println("Exception Occurred: "+e);
16        }
17    }
18
19    public void createConnection() {
20
21    }
22
23    public void closeConnection() {
24
25    }
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 }
```
- Bottom Status Bar:** Problems x, javadoc, Declaration, 0 errors, 2 warnings, 0 others.

6.5 Use **try-catch** in both the methods as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer View:** Shows the project structure under "src/main/java".
- Editor View:** Displays the code for `DAO.java`. The code implements a `DB` interface with methods for creating and closing database connections.
- Bottom Status Bar:** Shows "0 errors, 2 warnings, 0 others".

```
public DB() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        System.out.println("Driver Loaded");
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

public void createConnection() {
    try {
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

public void closeConnection() {
    try {
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}
```

6.6 Open the terminal window and type the command:

```
mysql -u john -p
```

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays a project structure for 'CMS'. In the center, a terminal window titled 'mail@ip-172-31-17-157:' shows the command 'mysql -u john -p' entered. Below the terminal window, the Java code editor shows a snippet of Java code with a catch block for exceptions.

6.7 Type the **show databases;** command to see the available database names

The screenshot shows the MySQL monitor window within the Eclipse IDE. The command 'mysql -u john -p' has been run, and the user has entered their password. The MySQL monitor displays the following output:

```

mysql> Enter password:
mysql> Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> Your MySQL connection id is 10
mysql> Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)

mysql>

```

6.8 Type the **sudo mysql** command to log in as a **sudo user**

The screenshot shows the Eclipse IDE interface with a MySQL terminal window open. The terminal window displays the MySQL monitor with the following commands and output:

```
mysql> exit
Bye
erishant@gmail:ip-172-31-17-157:~$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.27-Ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

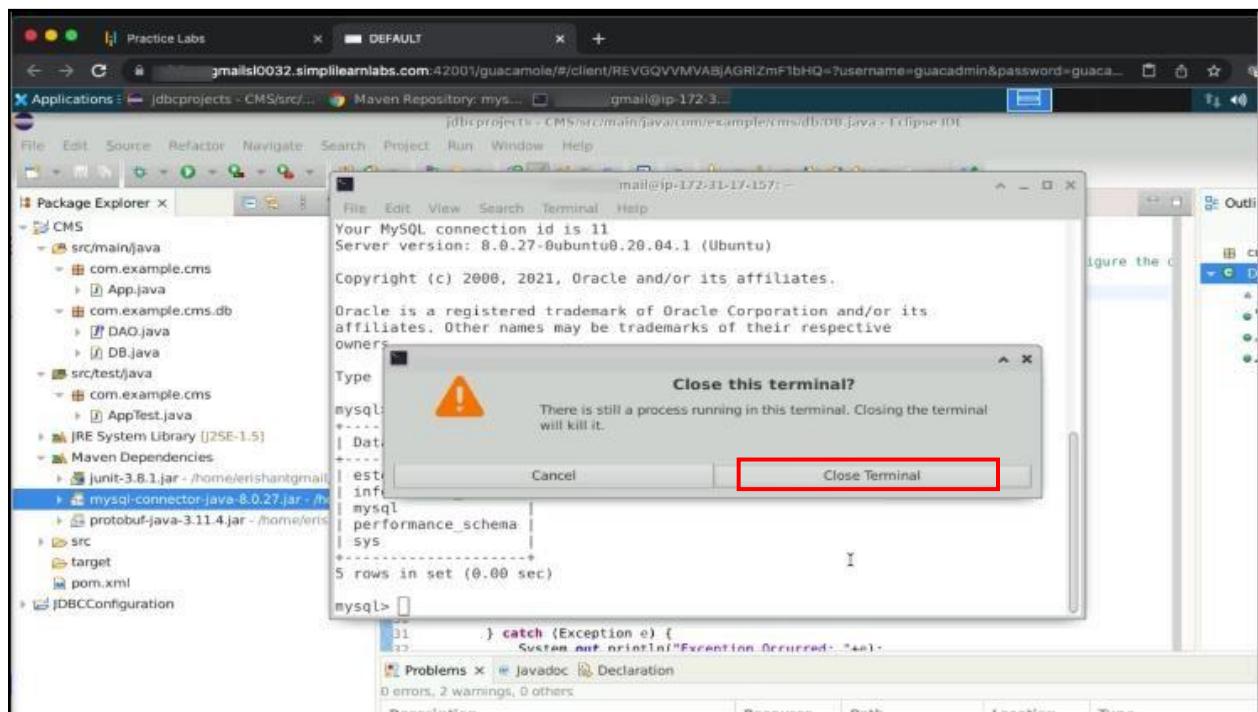
mysql>
```

6.9 Use the **show databases;** command to view some more databases as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with a MySQL terminal window open. The terminal window displays the MySQL monitor with the following commands and output:

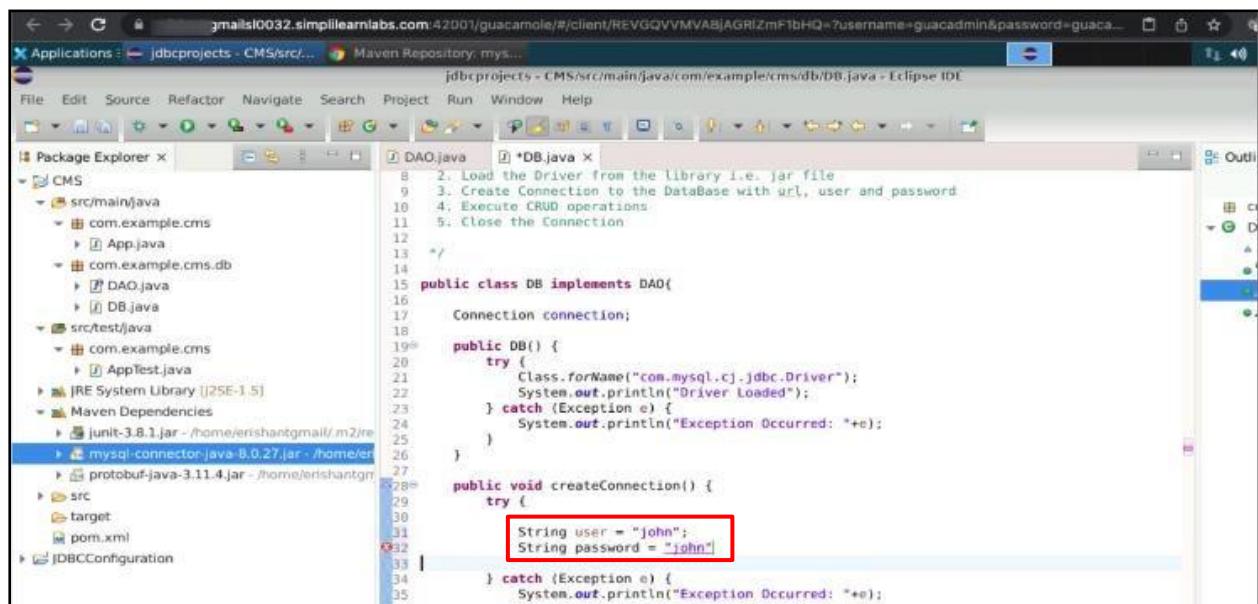
```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

6.10 Close the terminal



Step 7: Initialize the connection

7.1 Add a username and password for the connection as shown in the screenshot below:



7.2 Add a **URL** to connect MySQL as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT
- Toolbar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Left Sidebar (Package Explorer):** CMS package structure:
 - src/main/java:
 - com.example.cms:
 - App.java
 - com.example.cms.db:
 - DAO.java
 - DB.java
 - src/test/java:
 - com.example.cms:
 - AppTest.java
 - Maven Dependencies
 - JRE System Library [J2SE-1.5]
 - mysql-connector-java 8.0.27.jar - /home/erishantgmail/m2/repo
 - protobuf-java-3.11.4.jar - /home/erishantgma...
- Central Area (DAO.java):** Java code for a DAO class.

```
1 public class DB implements DAO {  
2     Connection connection;  
3  
4     public DB() {  
5         try {  
6             Class.forName("com.mysql.cj.jdbc.Driver");  
7             System.out.println("Driver Loaded");  
8         } catch (Exception e) {  
9             System.out.println("Exception Occurred: "+e);  
10        }  
11    }  
12  
13    public void createConnection() {  
14        try {  
15            String user = "john";  
16            String password = "john";  
17            String url = "jdbc:mysql://localhost/estore";  
18  
19            Connection connection = DriverManager.getConnection(url, user, password);  
20            this.connection = connection;  
21        } catch (Exception e) {  
22            System.out.println("Exception Occurred: "+e);  
23        }  
24    }  
25  
26    public void closeConnection() {  
27        try {  
28            if (connection != null)  
29                connection.close();  
30        } catch (Exception e) {  
31            System.out.println("Exception Occurred: "+e);  
32        }  
33    }  
34  
35    public void insertCustomer(String name, String address, String email, String phone) {  
36        String query = "insert into customer values(0, ?, ?, ?, ?);";  
37  
38        try {  
39            PreparedStatement ps = connection.prepareStatement(query);  
40            ps.setString(1, name);  
41            ps.setString(2, address);  
42            ps.setString(3, email);  
43            ps.setString(4, phone);  
44            ps.executeUpdate();  
45        } catch (Exception e) {  
46            System.out.println("Exception Occurred: "+e);  
47        }  
48    }  
49  
50    public void updateCustomer(int id, String name, String address, String email, String phone) {  
51        String query = "update customer set name=?, address=?, email=?, phone=? where id=?";  
52  
53        try {  
54            PreparedStatement ps = connection.prepareStatement(query);  
55            ps.setString(1, name);  
56            ps.setString(2, address);  
57            ps.setString(3, email);  
58            ps.setString(4, phone);  
59            ps.setInt(5, id);  
60            ps.executeUpdate();  
61        } catch (Exception e) {  
62            System.out.println("Exception Occurred: "+e);  
63        }  
64    }  
65  
66    public void deleteCustomer(int id) {  
67        String query = "delete from customer where id=?";  
68  
69        try {  
70            PreparedStatement ps = connection.prepareStatement(query);  
71            ps.setInt(1, id);  
72            ps.executeUpdate();  
73        } catch (Exception e) {  
74            System.out.println("Exception Occurred: "+e);  
75        }  
76    }  
77  
78    public List<Customer> getAllCustomer() {  
79        String query = "select * from customer";  
80  
81        List<Customer> list = new ArrayList<>();  
82  
83        try {  
84            Statement st = connection.createStatement();  
85            ResultSet rs = st.executeQuery(query);  
86  
87            while(rs.next()) {  
88                Customer c = new Customer();  
89                c.setId(rs.getInt("id"));  
90                c.setName(rs.getString("name"));  
91                c.setAddress(rs.getString("address"));  
92                c.setEmail(rs.getString("email"));  
93                c.setPhone(rs.getString("phone"));  
94                list.add(c);  
95            }  
96        } catch (Exception e) {  
97            System.out.println("Exception Occurred: "+e);  
98        }  
99  
100       return list;  
101    }  
102}
```
- Bottom Status Bar:** Problems, Javadoc, Declaration, 0 errors, 2 warnings, 0 others

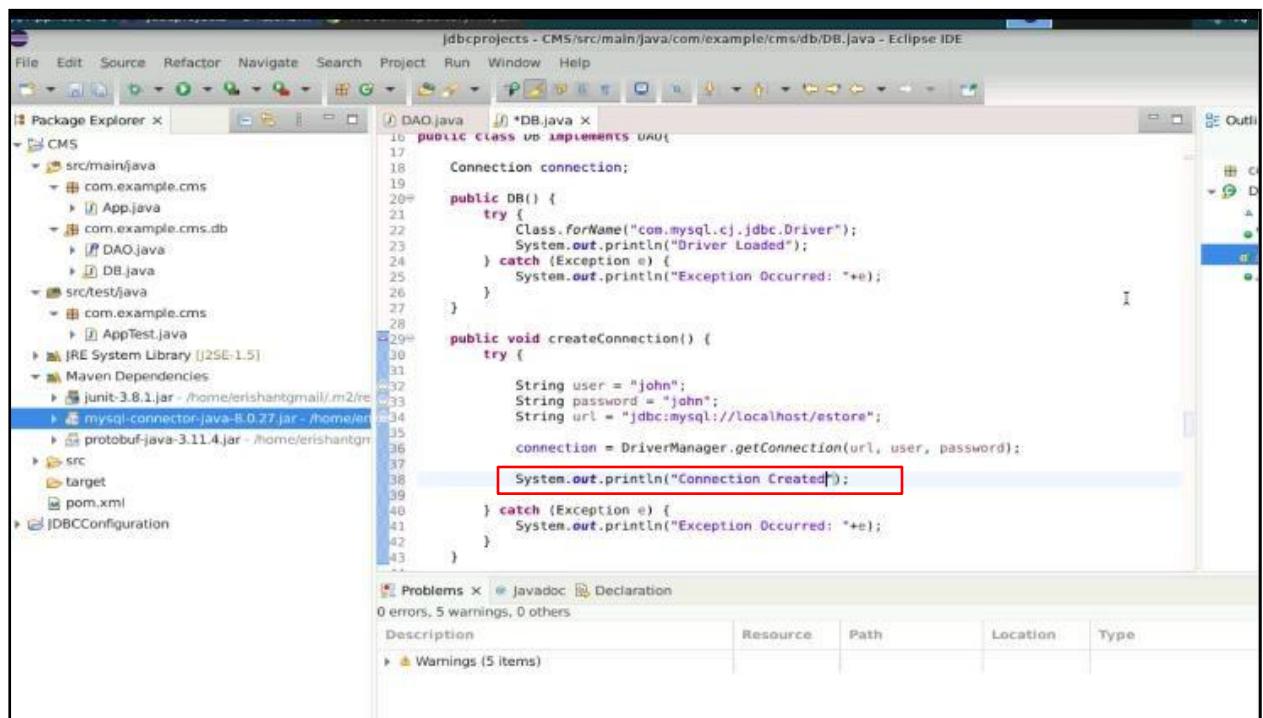
7.3 Initialize the connection as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT
- Toolbar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Package Explorer:** CMS (src/main/java, src/test/java, JRE System Library [J2SE-1.5], Maven Dependencies)
- Maven Repository:** maven central
- Current File:** DAO.java (implements DAO)
- Code Content:** The DAO.java file contains Java code for a DAO class. A specific line of code, `connection = DriverManager.getConnection(url, user, password);`, is highlighted with a red rectangular box.
- Bottom Status Bar:** Problems X Javadoc Declaration, 0 errors, 5 warnings, 0 others

```
1 package com.example.cms;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5
6 public class DAO implements DAO {
7     Connection connection;
8
9     public DAO() {
10         try {
11             Class.forName("com.mysql.cj.jdbc.Driver");
12             System.out.println("Driver Loaded");
13         } catch (Exception e) {
14             System.out.println("Exception Occurred: " + e);
15         }
16     }
17
18     public void createConnection() {
19         try {
20             String user = "john";
21             String password = "john";
22             String url = "jdbc:mysql://localhost:estore";
23
24             connection = DriverManager.getConnection(url, user, password);
25
26         } catch (Exception e) {
27             System.out.println("Exception Occurred: " + e);
28         }
29     }
30
31     public void closeConnection() {
32
33     }
34 }
```

7.4 Add a print statement to show that the connection is created as shown in the screenshot below:



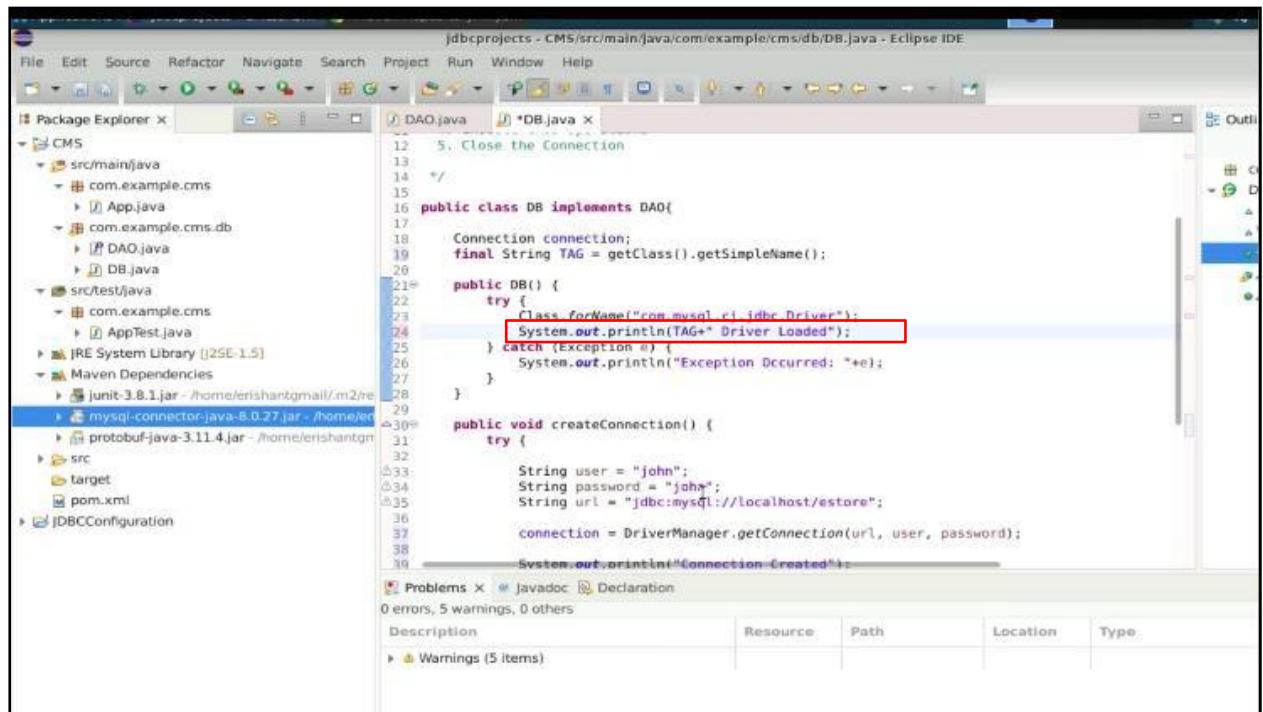
```

    JDBCprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X DAO.java *DB.java X
16 public class DB implements DAO{
17     Connection connection;
18
19     public DB() {
20         try {
21             Class.forName("com.mysql.cj.jdbc.Driver");
22             System.out.println("Driver Loaded");
23         } catch (Exception e) {
24             System.out.println("Exception Occurred: "+e);
25         }
26     }
27
28     public void createConnection() {
29         try {
30             String user = "john";
31             String password = "john";
32             String url = "jdbc:mysql://localhost/estore";
33
34             connection = DriverManager.getConnection(url, user, password);
35
36             System.out.println("Connection Created");
37         } catch (Exception e) {
38             System.out.println("Exception Occurred: "+e);
39         }
40     }
41
42 }

```

The line `System.out.println("Connection Created");` is highlighted with a red box.

7.5 Type the final string tag as shown in the screenshot below:



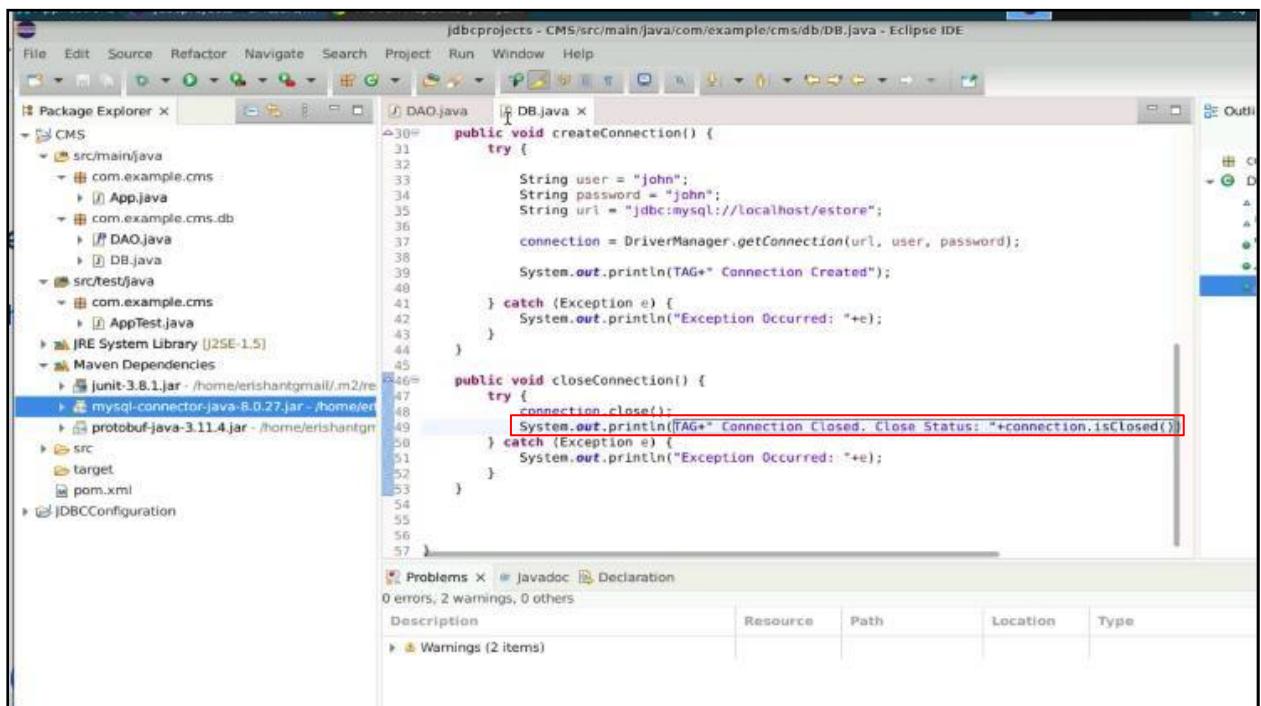
```

    JDBCprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X DAO.java *DB.java X
12     5. Close the Connection
13
14     */
15
16     public class DB implements DAO{
17
18         Connection connection;
19         final String TAG = getClass().getSimpleName();
20
21         public DB() {
22             try {
23                 Class.forName("com.mysql.cj.jdbc.Driver");
24                 System.out.println(TAG+" Driver Loaded");
25             } catch (Exception e) {
26                 System.out.println("Exception Occurred: "+e);
27             }
28         }
29
30         public void createConnection() {
31             try {
32
33                 String user = "john";
34                 String password = "john";
35                 String url = "jdbc:mysql://localhost/estore";
36
37                 connection = DriverManager.getConnection(url, user, password);
38
39                 System.out.println("Connection Created");
40             }
41
42         }

```

The line `System.out.println(TAG+" Driver Loaded");` is highlighted with a red box.

7.6 Execute the **closeConnection** method as shown in the screenshot below:

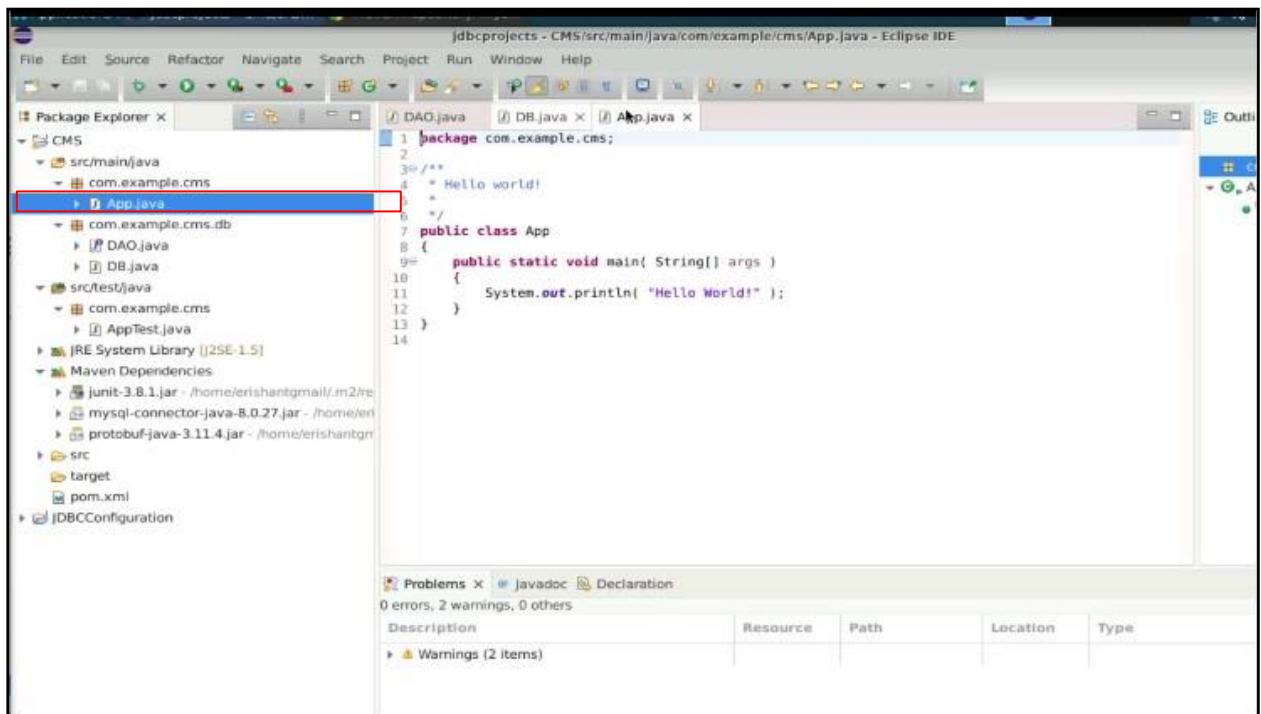


The screenshot shows the Eclipse IDE interface with the title "jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE". The left sidebar shows the "Package Explorer" with the project structure: CMS > src > main > java > com.example.cms > DAO.java, DB.java, App.java, AppTest.java. Below it are JRE System Library [J2SE-1.5] and Maven Dependencies. The right sidebar shows the "Outline" view. The central editor window displays the DB.java code. The closeConnection() method is highlighted with a red rectangular box around the line "System.out.println(TAG+" Connection Closed. Close Status: "+connection.isClosed());". The code is as follows:

```
public void closeConnection() {
    try {
        connection.close();
        System.out.println(TAG+" Connection Closed. Close Status: "+connection.isClosed());
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}
```

The "Problems" view at the bottom shows 0 errors, 2 warnings, and 0 others.

7.7 Open the **App.java** file from the left section as shown in the screenshot below:



The screenshot shows the Eclipse IDE interface with the title "jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE". The left sidebar shows the "Package Explorer" with the project structure: CMS > src > main > java > com.example.cms > App.java. The App.java file is highlighted with a red rectangular box. Below it are JRE System Library [J2SE-1.5] and Maven Dependencies. The right sidebar shows the "Outline" view. The central editor window displays the App.java code:

```
package com.example.cms;
/*
 * Hello world!
 */
public class App {
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

The "Problems" view at the bottom shows 0 errors, 2 warnings, and 0 others.

7.8 Add a print statement in the **App.java** file as shown in the screenshot below:

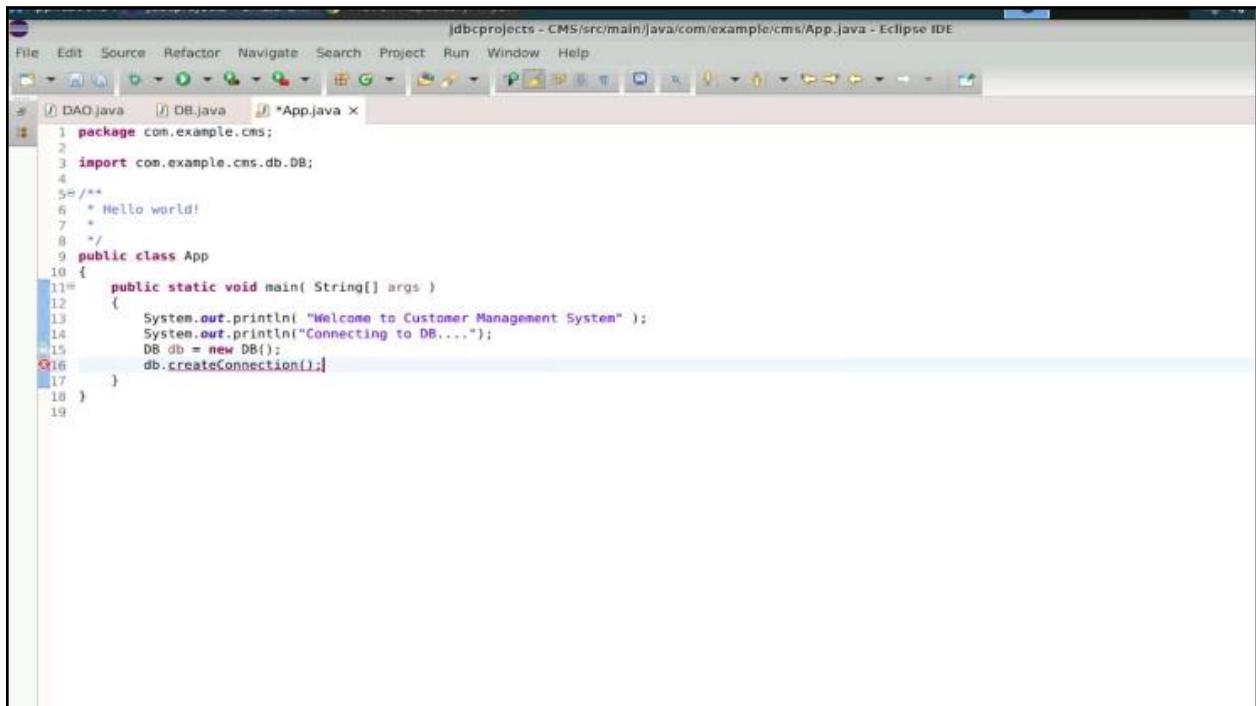
```
jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DAO.java DB.java *App.java
1 package com.example.cms;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Welcome to Customer Management System" );
12        System.out.println("Connecting to DB....");
13    }
14
15 }
16
```

7.9 Use the **DB** database as shown in the screenshot below:

```
jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DAO.java DB.java *App.java
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  *
8  */
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14         System.out.println("Connecting to DB....");
15         DB db = new DB();
16     }
17 }
```

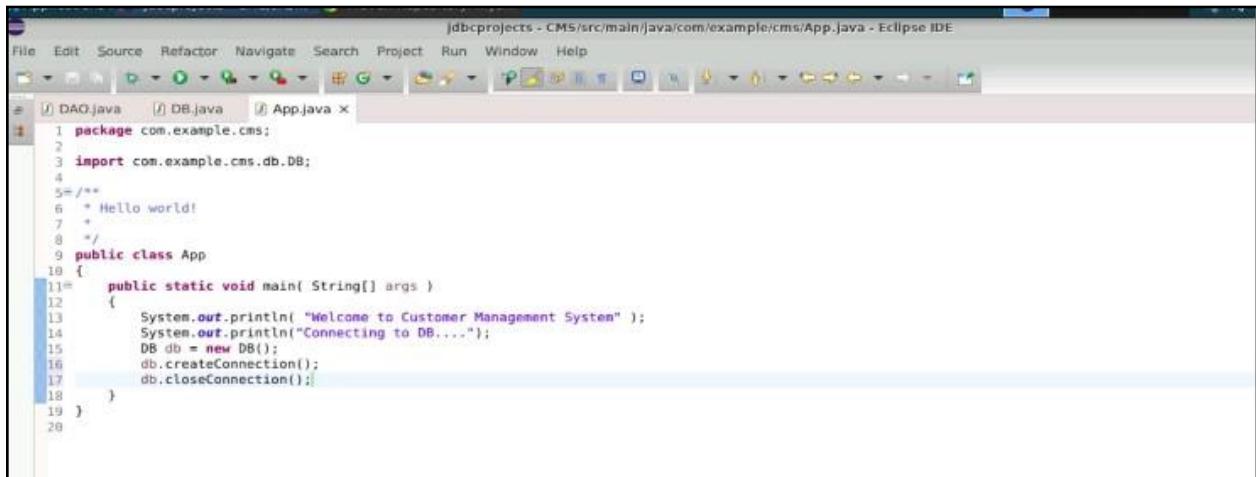
Step 8: Execute the createConnection method

8.1 Execute the `createConnection()` method



```
jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DAO.java DB.java App.java
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  *
8  */
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14         System.out.println("Connecting to DB....");
15         DB db = new DB();
16         db.createConnection();
17     }
18 }
19
```

8.2 Execute `closeConnection()`



```
jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DAO.java DB.java App.java
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  *
8  */
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14         System.out.println("Connecting to DB....");
15         DB db = new DB();
16         db.createConnection();
17         db.closeConnection();
18     }
19 }
20
```

8.3 Run the code, and you should see the output as **Close Status: true** as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Shows the title "ls10032.simplelearnlabs.com:42001/guacamole/#/client/REVGQVVVMVABjAGRIZmF1bHQ=?username=guacadmin&password=guaca..." and "Maven Repository: mys...".
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project management.
- Menu Bar:** "Navigate", "Search", "Project", "Run", "Window", "Help".
- Left Side:** Project Explorer and Package Explorer panes.
- Code Editor:** The "App.java" file is open, containing the provided Java code.
- Console View:** Displays the application's log output:

```
<terminated> App [java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jdk11
Welcome to Customer Management System
Connecting to DB....
DB Driver Loaded
DB Connection Created
DB Connection Closed. Close Status: true
```

A red box highlights the last line of the output: "DB Connection Closed. Close Status: true".

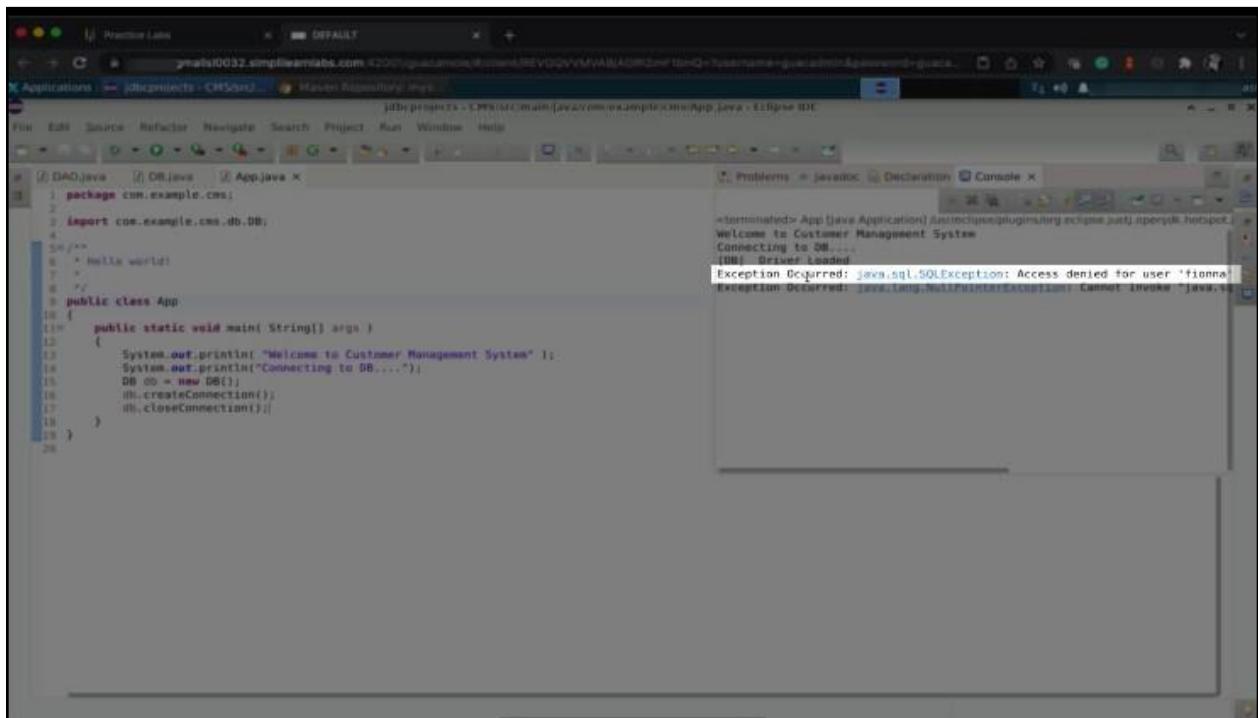
8.4 Change the username, and add a tag as shown in the screenshot below:

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT
- Toolbar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Code Editor:** The code is for a DAO class named DB. The code includes imports for java.sql.Connection and java.sql.DriverManager, and a block of comments explaining JDBC setup. It defines a final variable TAG, a DB() constructor, and a createConnection() method. The code editor has syntax highlighting and a red box highlighting the line `final String TAG = "["+getClass().getSimpleName()+"]";`.
- Status Bar:** Writable, Smart Insert, 19:58:525

```
1 package com.example.cms.db;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5
6 /**
7 * JDBC Procedure:
8 * 1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
9 * 2. Load the Driver from the Library i.e. jar file
10 * 3. Create Connection to the DataBase with url, user and password
11 * 4. Execute CRUD operations
12 * 5. Close the Connection
13 */
14
15
16 public class DB implements DAO{
17
18     final String TAG = "[ "+getClass().getSimpleName()+" ]";
19
20     public DB() {
21         try {
22             Class.forName("com.mysql.cj.jdbc.Driver");
23             System.out.println(TAG+" Driver Loaded");
24         } catch (Exception e) {
25             System.out.println("Exception Occurred: "+e);
26         }
27     }
28
29
30     public void createConnection() {
31         try {
32
33             String user = "fionna";
34             String password = "John";
35             String url = "jdbc:mysql://localhost/estere";
36
37             connection = DriverManager.getConnection(url, user, password);
38
39         } catch (Exception e) {
40             System.out.println("Exception Occurred: "+e);
41         }
42     }
43 }
```

8.5 Re-run the code



The screenshot shows the Eclipse IDE interface. On the left, there are three tabs: DAO.java, DB.java, and App.java. The code for App.java is visible:

```
1 package com.example.cms;
2 import com.example.cms.db.DB;
3
4 public class App {
5     public static void main( String[] args ) {
6         System.out.println( "Welcome to Customer Management System" );
7         System.out.println("Connecting to DB....");
8         DB db = new DB();
9         db.createConnection();
10        db.closeConnection();
11    }
12 }
```

On the right, a terminal window displays the output of the application's execution. It starts with a welcome message, then attempts to connect to the database, and finally shows an exception:

```
>terminated> App (Java Application) [CustomerManagementSystem] [Run]
welcome to Customer Management System
Connecting to DB...
DB Connection
Exception Occurred: java.sql.SQLException: Access denied for user 'fionna'
Exception occurred: java.lang.reflect.InvocationTargetException - Cannot invoke "java.sql.
```

By following these steps, you have successfully connected a Java project to a MySQL database by configuring JDBC and creating a connection using the **createConnection()** method for database operations.