

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

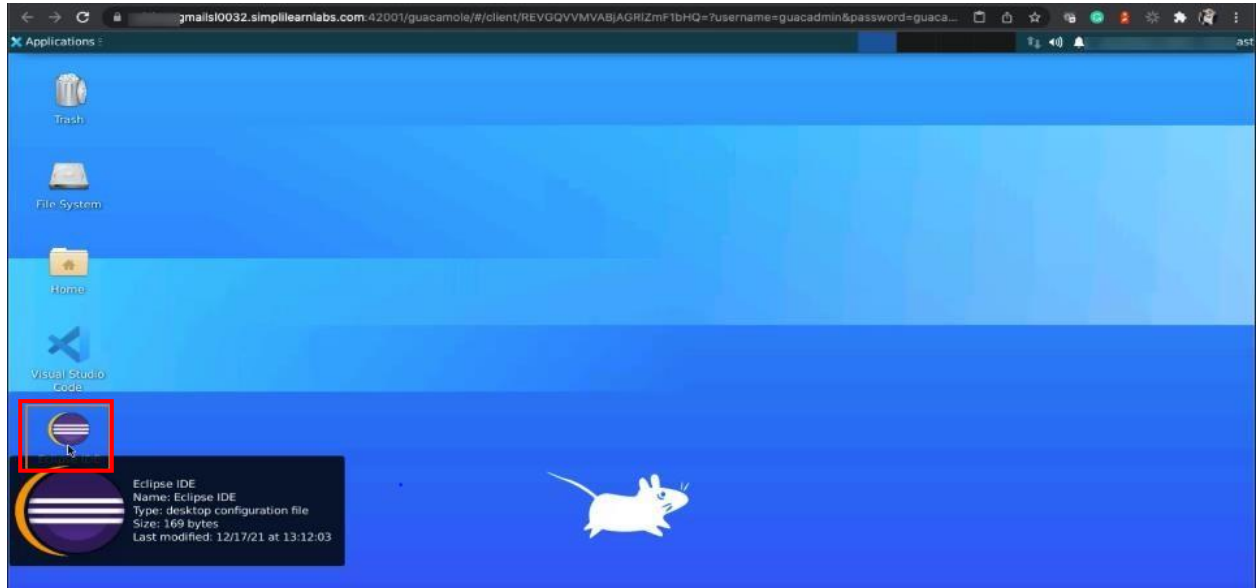
Prerequisites: None

Steps to be followed:

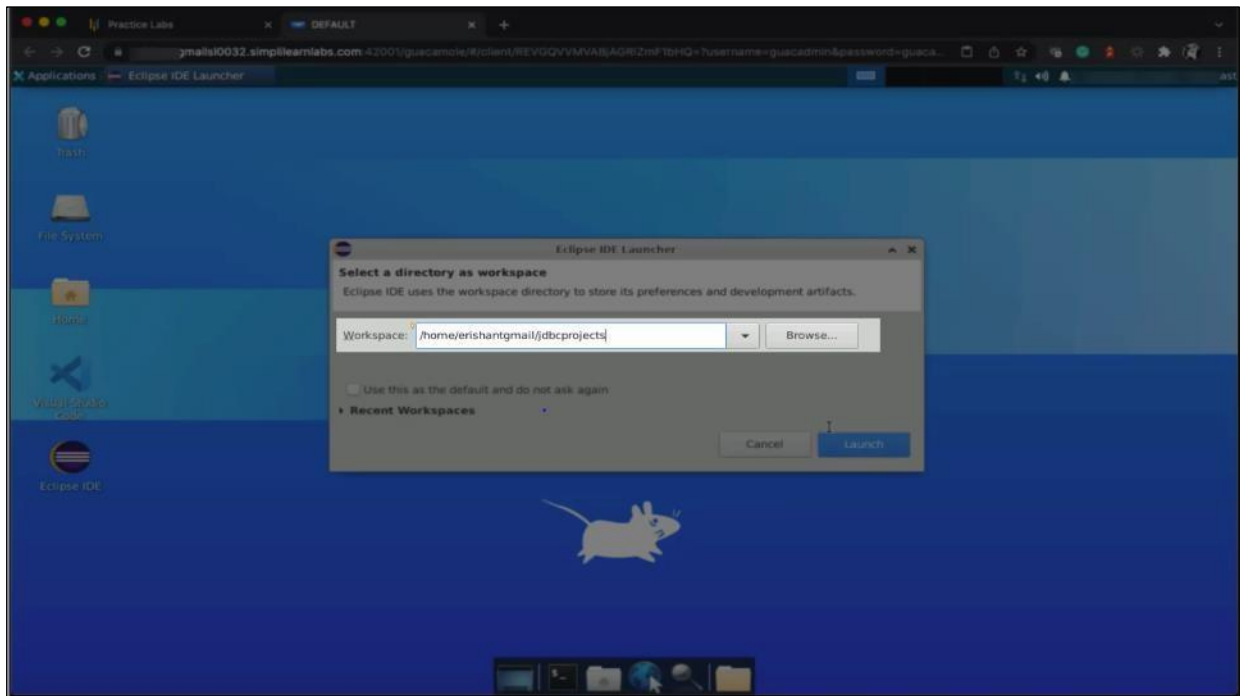
1. Create a Java project
2. Download the MySQL Connector/J library
3. Create a Maven project
4. Create an interface in `com.example.cms`
5. Create the DB class
6. Create a connection in the `DAO.java` file
7. Initialize the connection
8. Execute the `createConnection` method

Step 1: Create a Java project

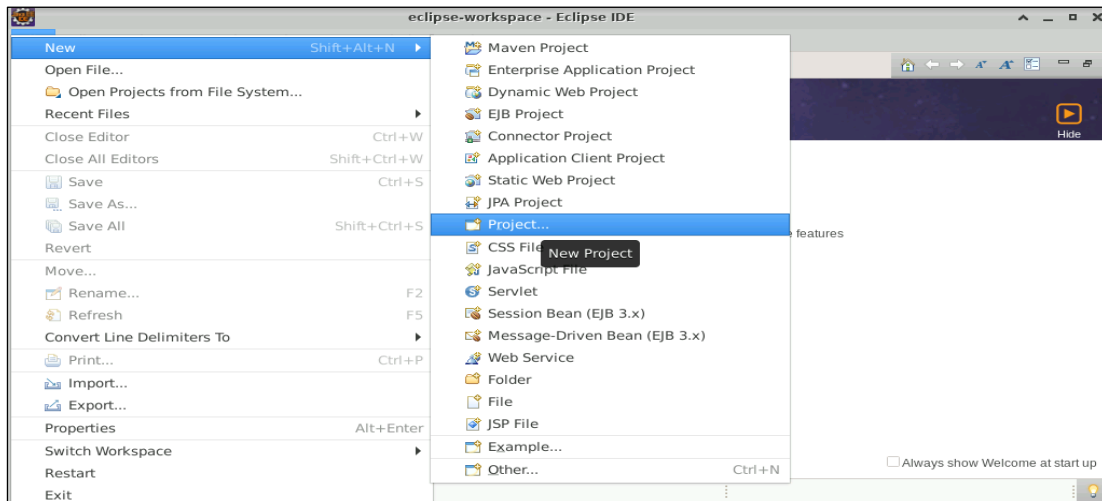
1.1 Open Eclipse IDE to create a Java project



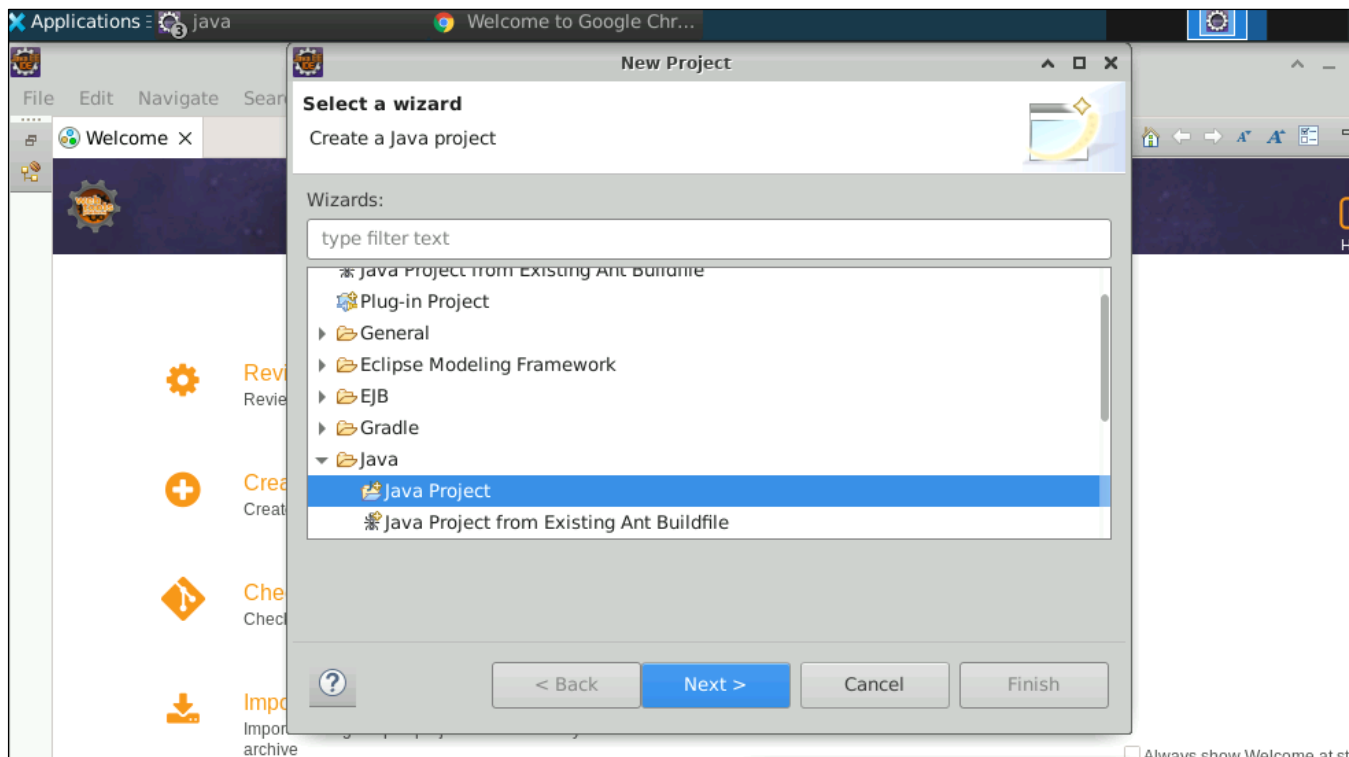
1.2 Select the default workspace and click on **Launch** to open Eclipse IDE



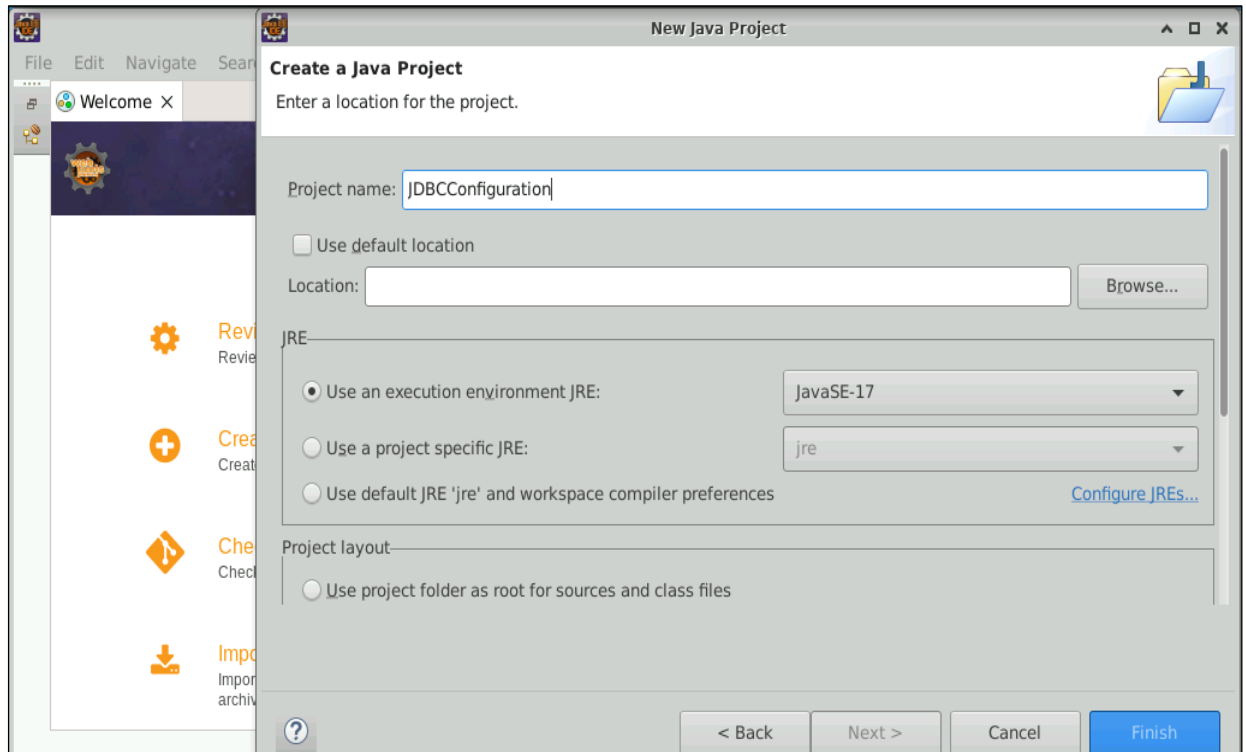
1.3 Create a new Java project, navigate to **File** from the toolbar > **New** > **Project**



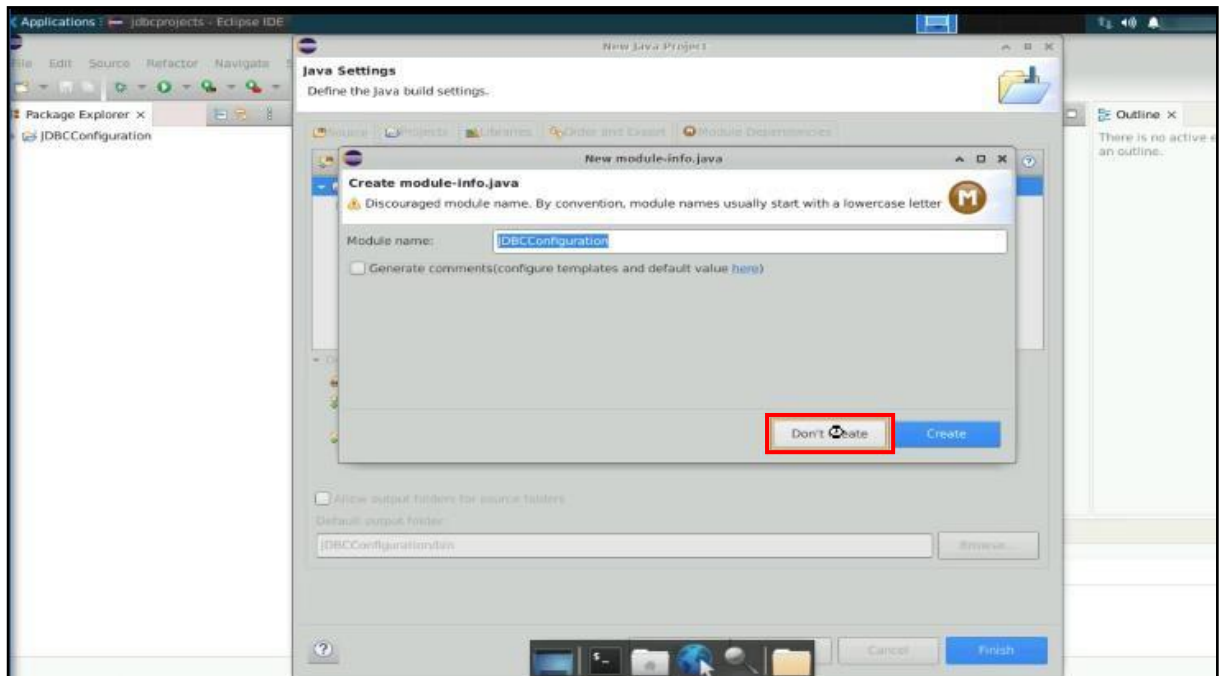
1.4 Select **Java Project** and click on **Next**



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

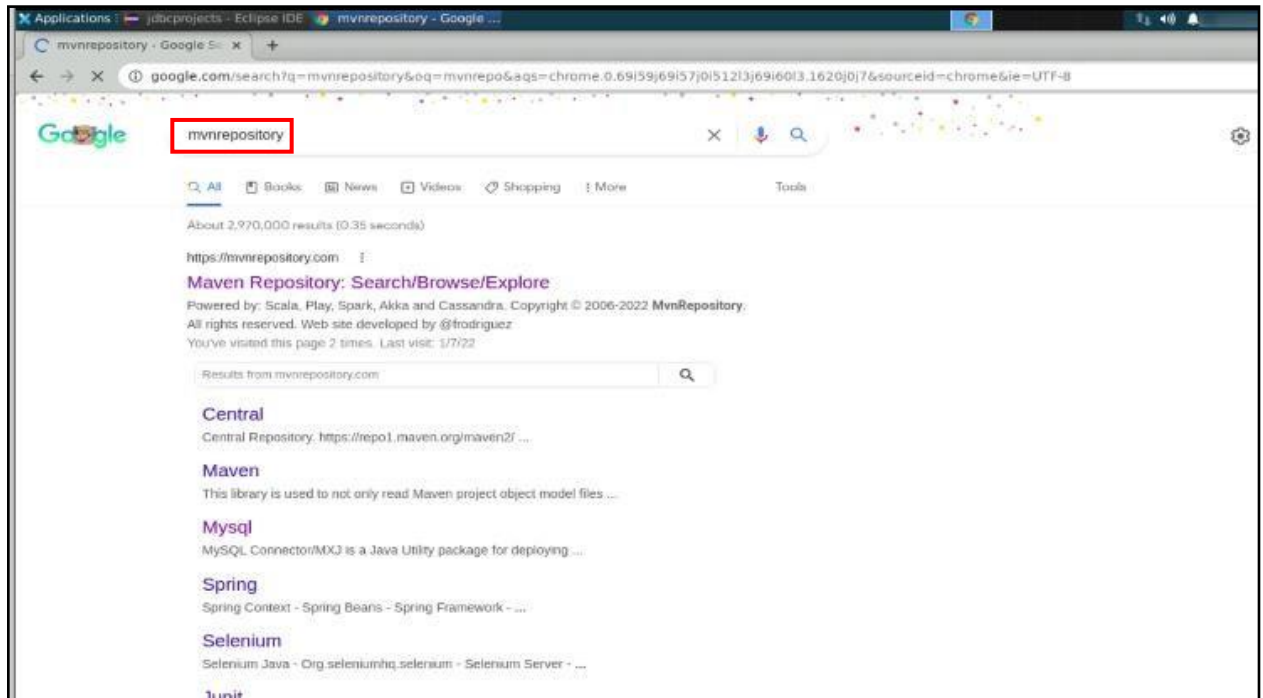


1.6 Click on the **Don't Create** option as there is no need to create a module, as shown in the screenshot below:

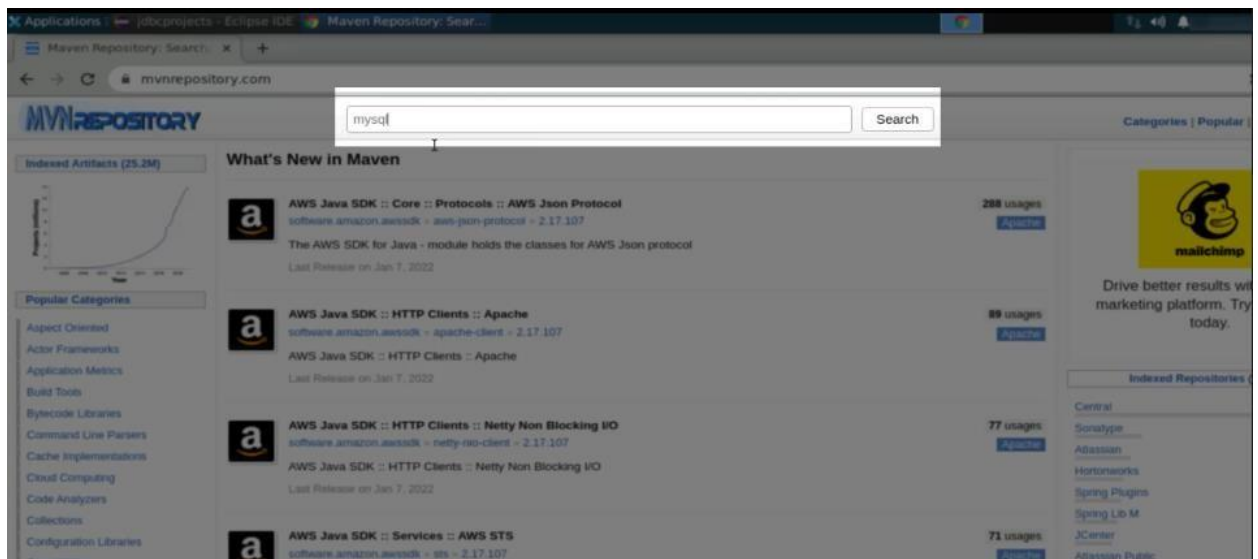


Step 2: Download the MySQL Connector/J library

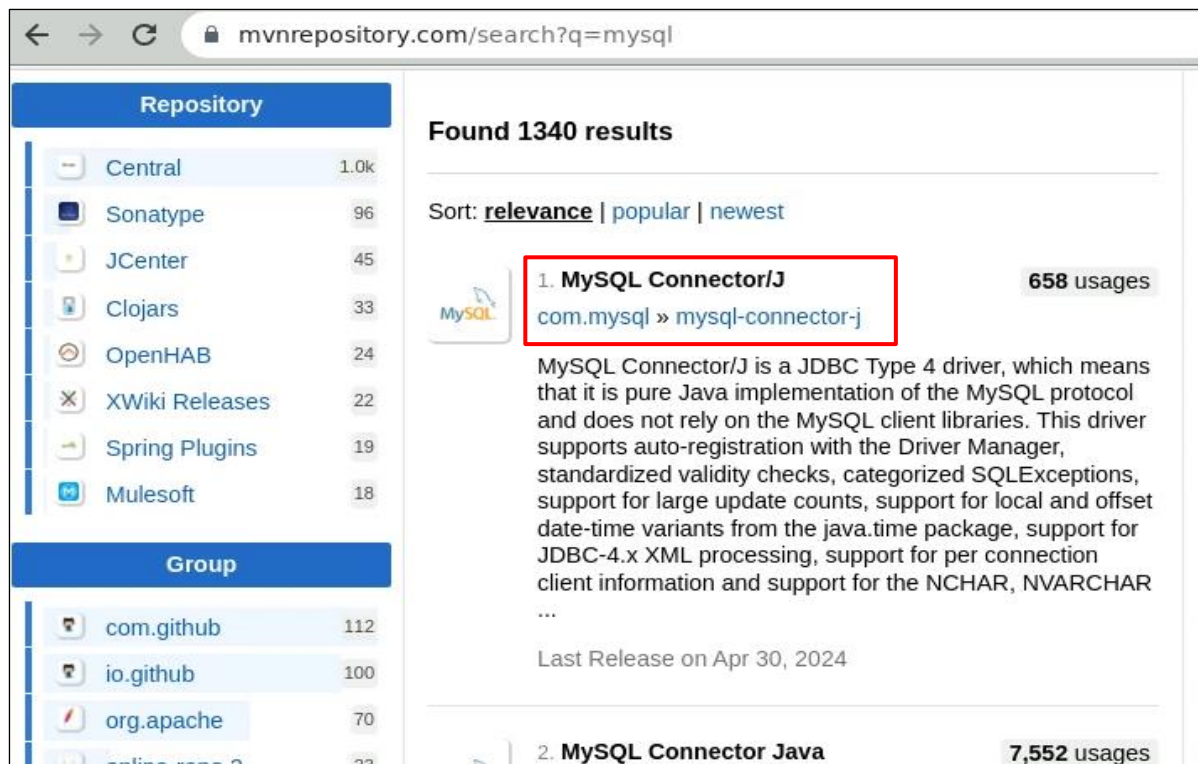
2.1 Open the preferred browser, search for **mvnrepository**, and click the first link of the Maven Repository



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



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



The screenshot shows the mvnrepository.com search results for the query 'mysql'. The left sidebar lists various repositories and groups. The main content area shows 1340 results, sorted by relevance. The first result, 'MySQL Connector/J', is highlighted with a red box. It shows 658 usages and a link to 'com.mysql » mysql-connector-j'. The description states it is a JDBC Type 4 driver, a pure Java implementation of the MySQL protocol, and lists various features like auto-registration, standardized validity checks, and support for large update counts. The last release is dated April 30, 2024. The second result, 'MySQL Connector Java', shows 7,552 usages.

Repository	Count
Central	1.0k
Sonatype	96
JCenter	45
Clojars	33
OpenHAB	24
XWiki Releases	22
Spring Plugins	19
Mulesoft	18

Group	Count
com.github	112
io.github	100
org.apache	70
online.repo.2	22

Found 1340 results

Sort: relevance | popular | newest

1. **MySQL Connector/J** 658 usages

[com.mysql » mysql-connector-j](#)

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 ...

Last Release on Apr 30, 2024

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 artifact `com.mysql:mysql-connector-j`. The page includes a sidebar with navigation links, a top section with tags and statistics, and a table of versions. The version 8.4.0 is highlighted with a red box.

Navigation Links:

- Java Specifications
- JSON Libraries
- JVM Languages
- Language Runtime
- Core Utilities
- Mocking
- Web Assets
- Annotation Libraries
- HTTP Clients
- Logging Bridges
- Dependency Injection
- XML Processing
- Web Frameworks
- I/O Utilities
- Defect Detection Metadata

Tags: database, sql, jdbc, driver, connector, rdbms, mysql, connection

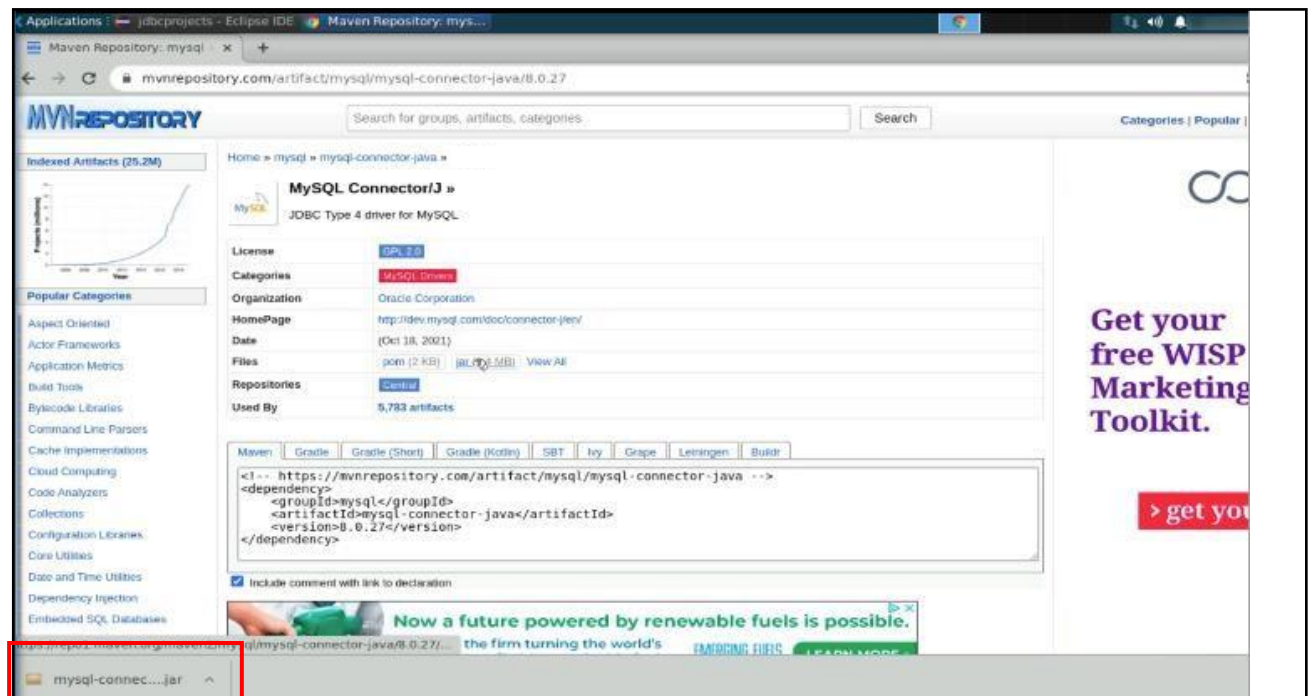
Ranking: #744 in MvnRepository (See Top Artifacts), #9 in JDBC Drivers

Used By: 658 artifacts

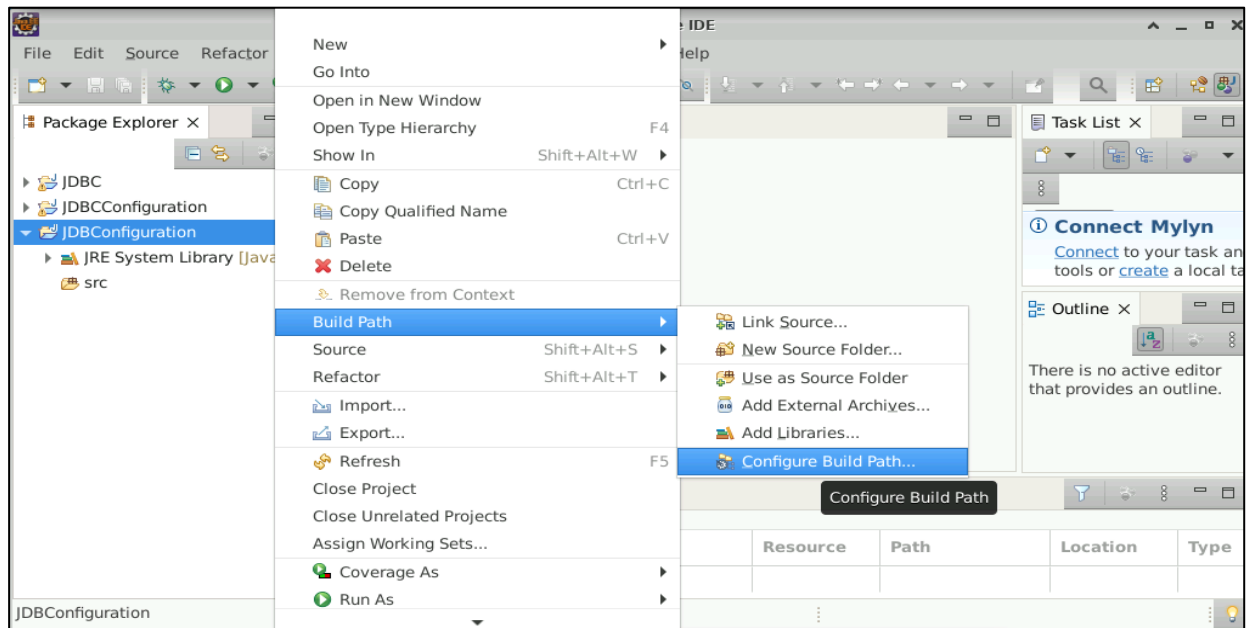
Repository Tabs: Central (7), Redhat GA (1), Redhat EA (1)

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.x 8.0.33		Central	302	Apr 18, 2023
8.0.x 8.0.32		Central	140	Jan 18, 2023

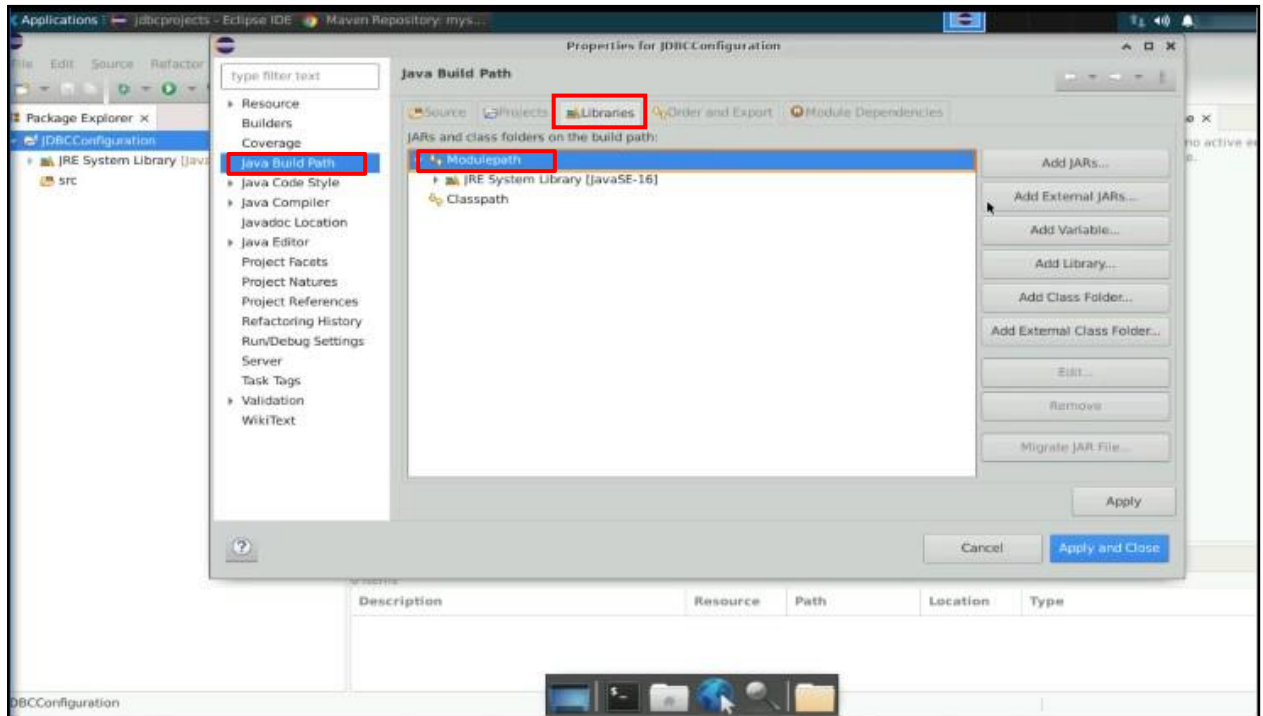
2.5 Click on the latest connector version, then click on JAR file under the files section to download it, as shown in the screenshot below:



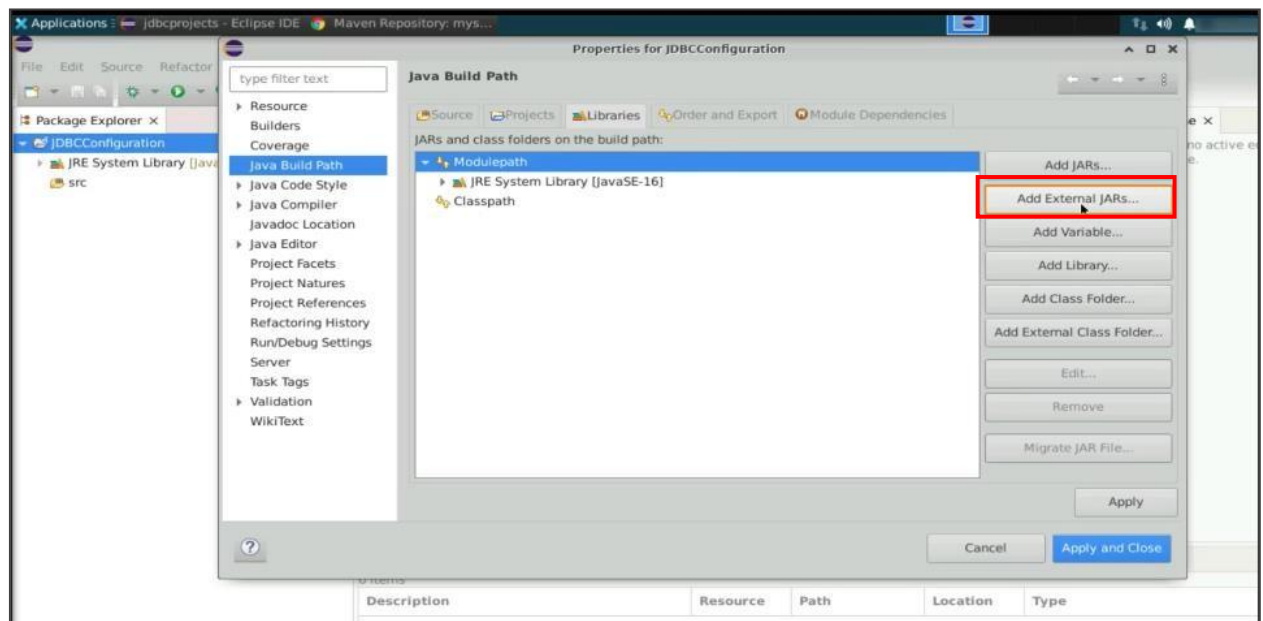
2.6 Right-click on the project file created, select **Build Path**, and choose **Configure Build Path** as shown in the screenshot below:



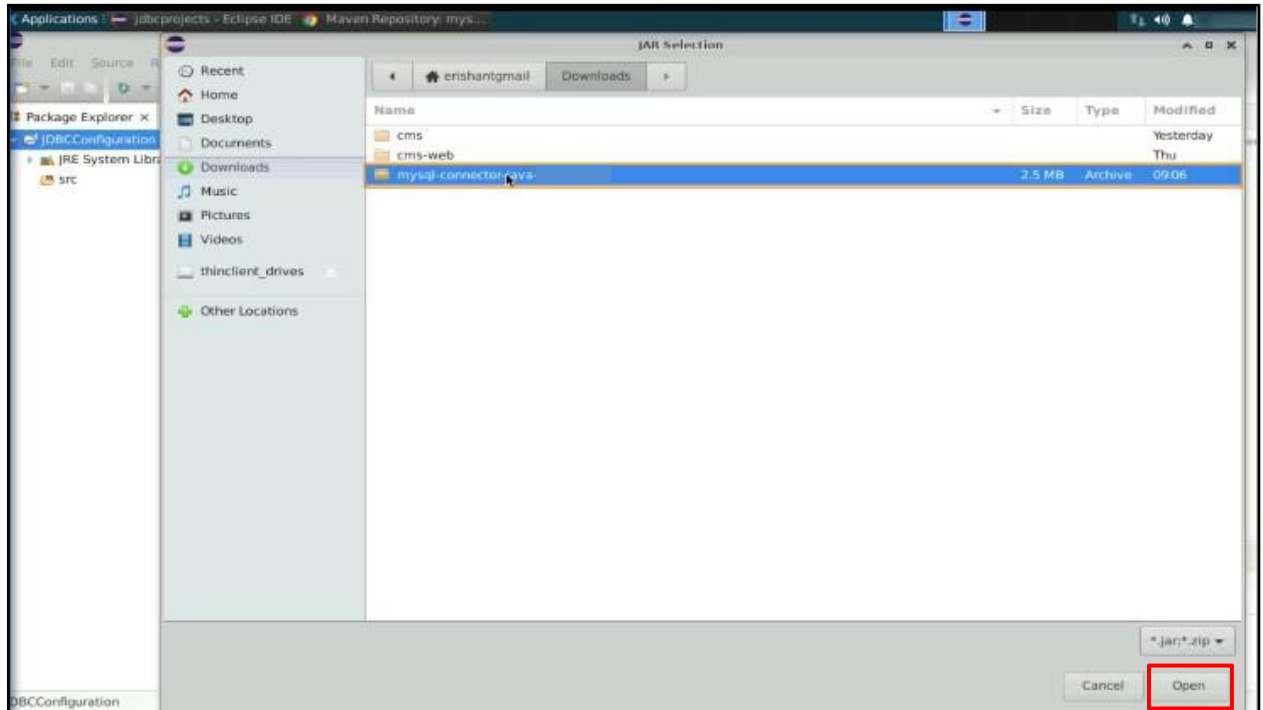
2.7 Go to **Java Build Path > Libraries tab > Modulepath** in the project properties window as shown in the screenshot below:



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:

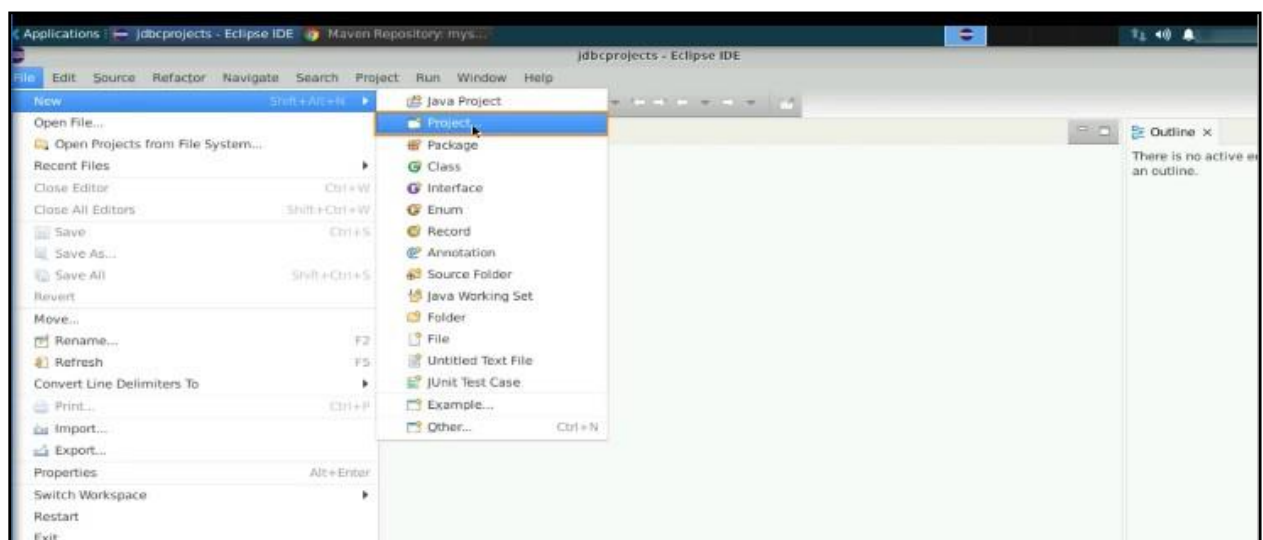


2.9 Select the **mysql-connector-java-8.4.0.jar** file and click **Open** as shown in the screenshot below:

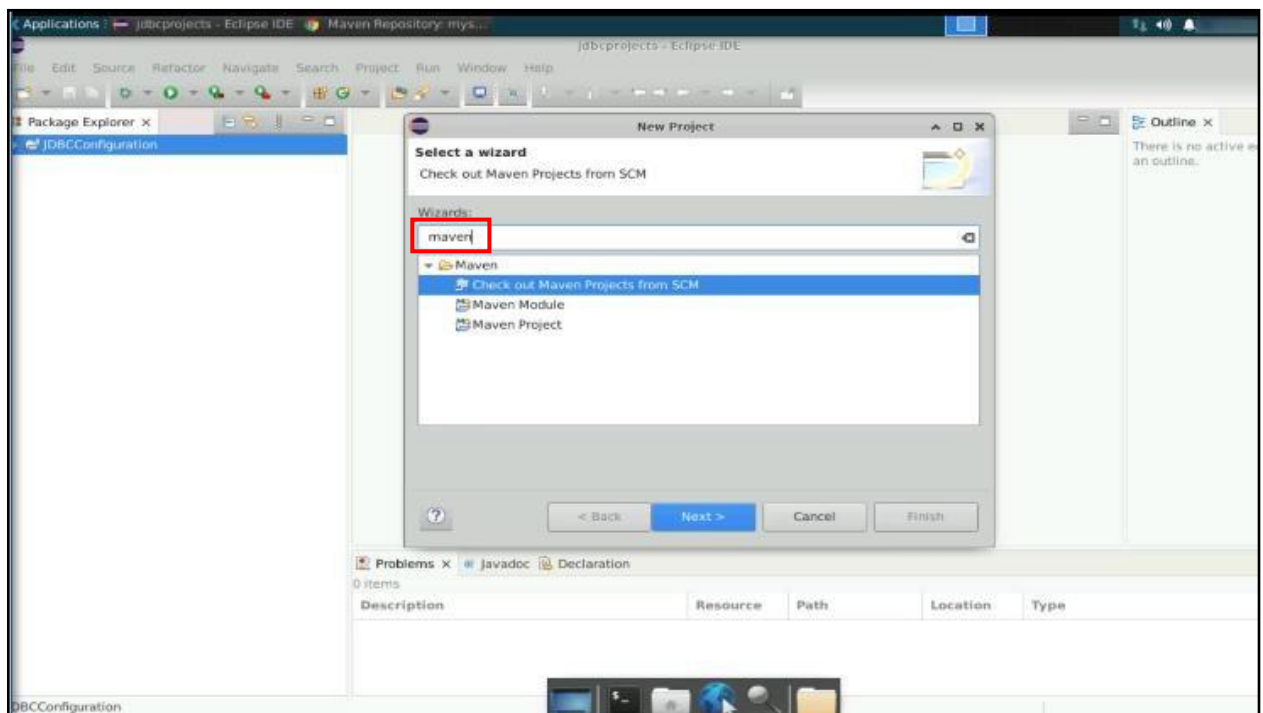


Step 3: Create a Maven project

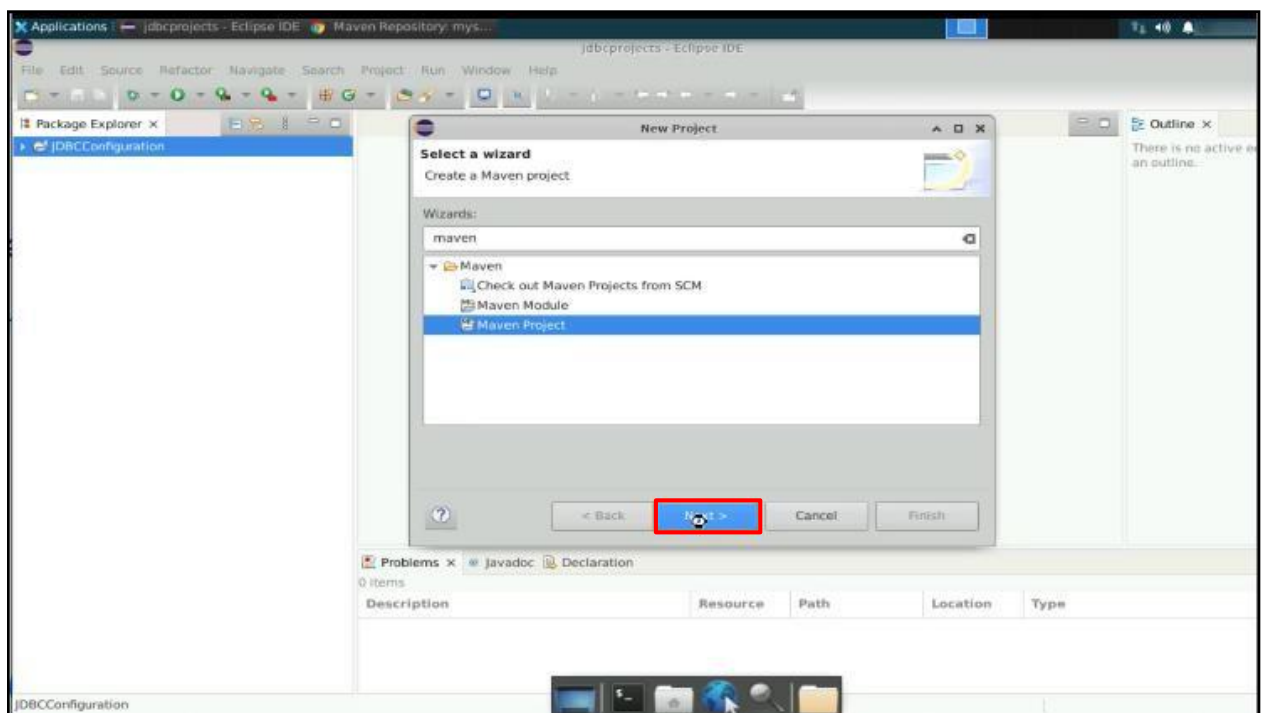
3.1 Create a new project by navigating to **Files > New > Project** as shown in the screenshot below:



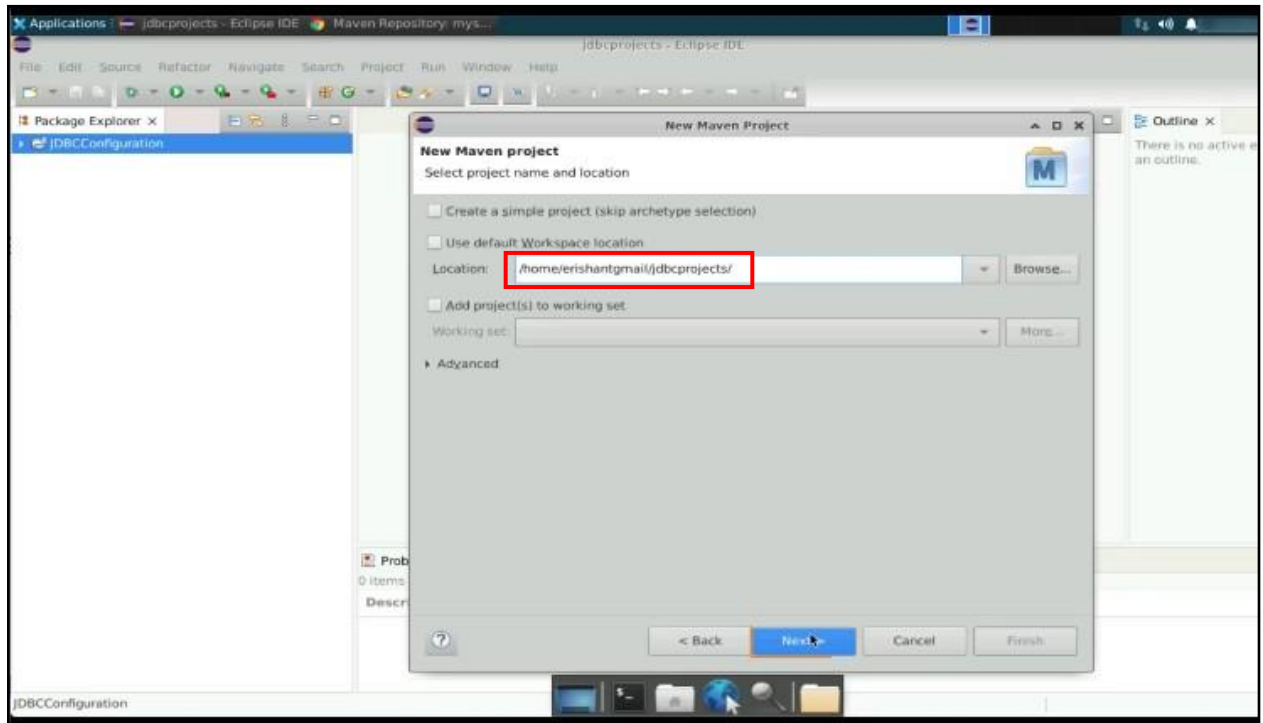
3.2 Search for **Maven** to create a Maven project as shown in the screenshot below:



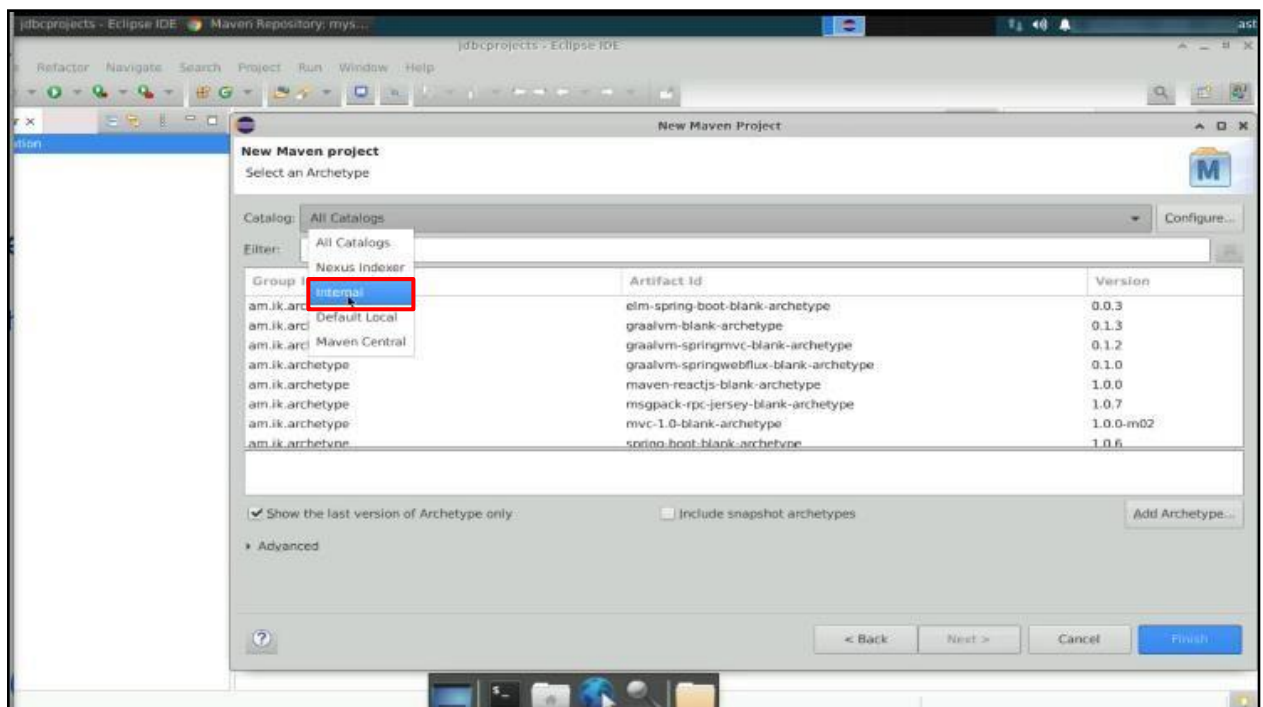
3.3 Click on **Maven Project** and select **Next** as shown in the screenshot below:



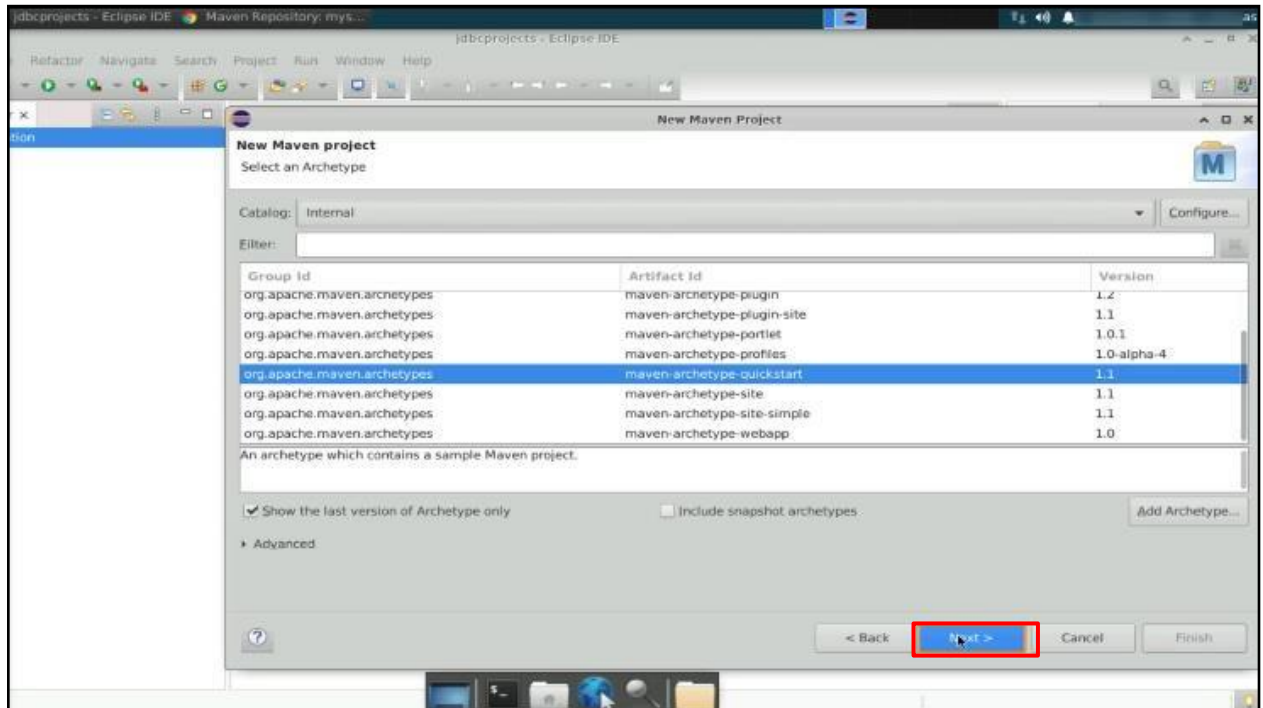
3.4 Set the **workplace location** to match that of the JDBC project, and click **Next** as shown in the screenshot below:



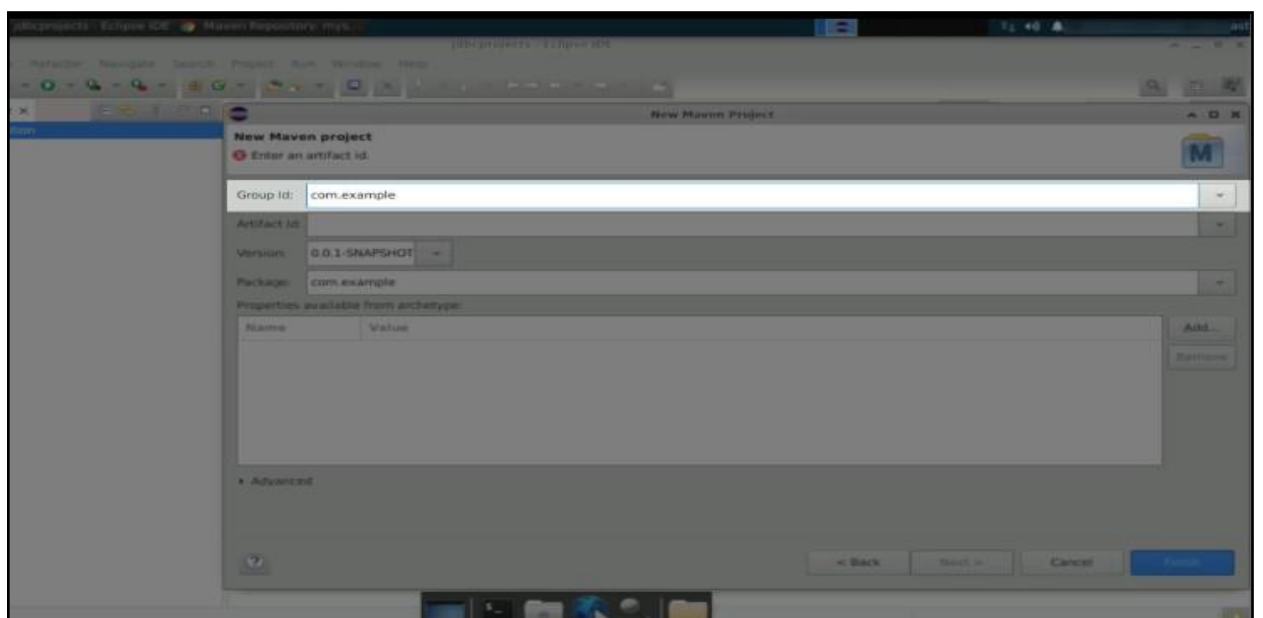
3.5 Select **Internal** from the catalog options as shown in the screenshot below:



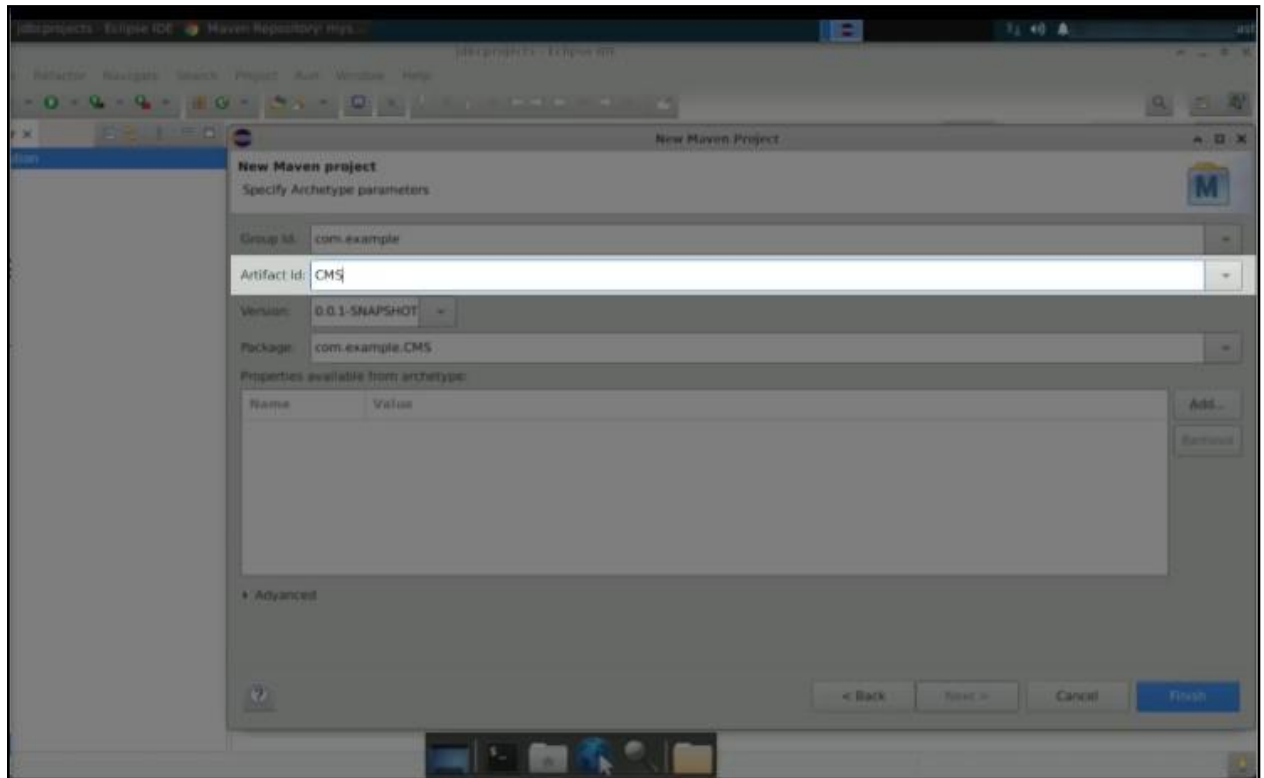
3.6 Select the **maven-archetype-quickstart** option from the options shown, and click on **Next** as shown in the screenshot below:



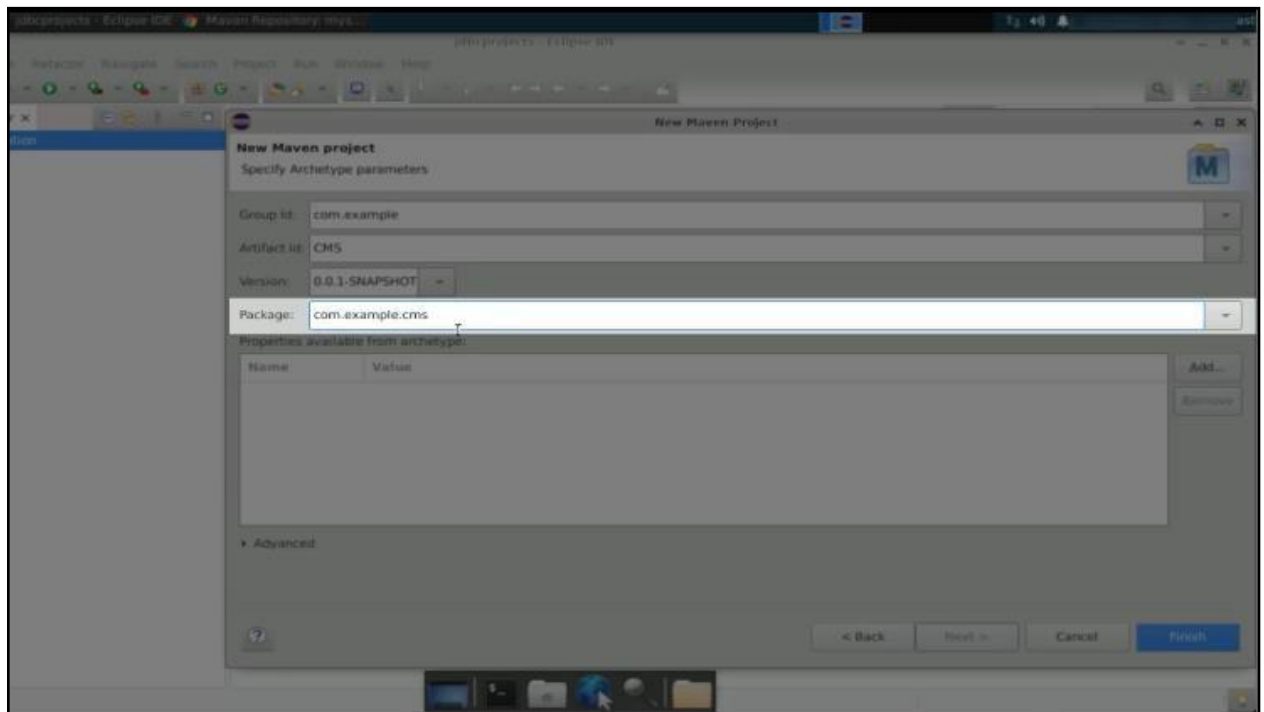
3.7 Provide the **Group Id**, which is your company's domain name in the reverse order:



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

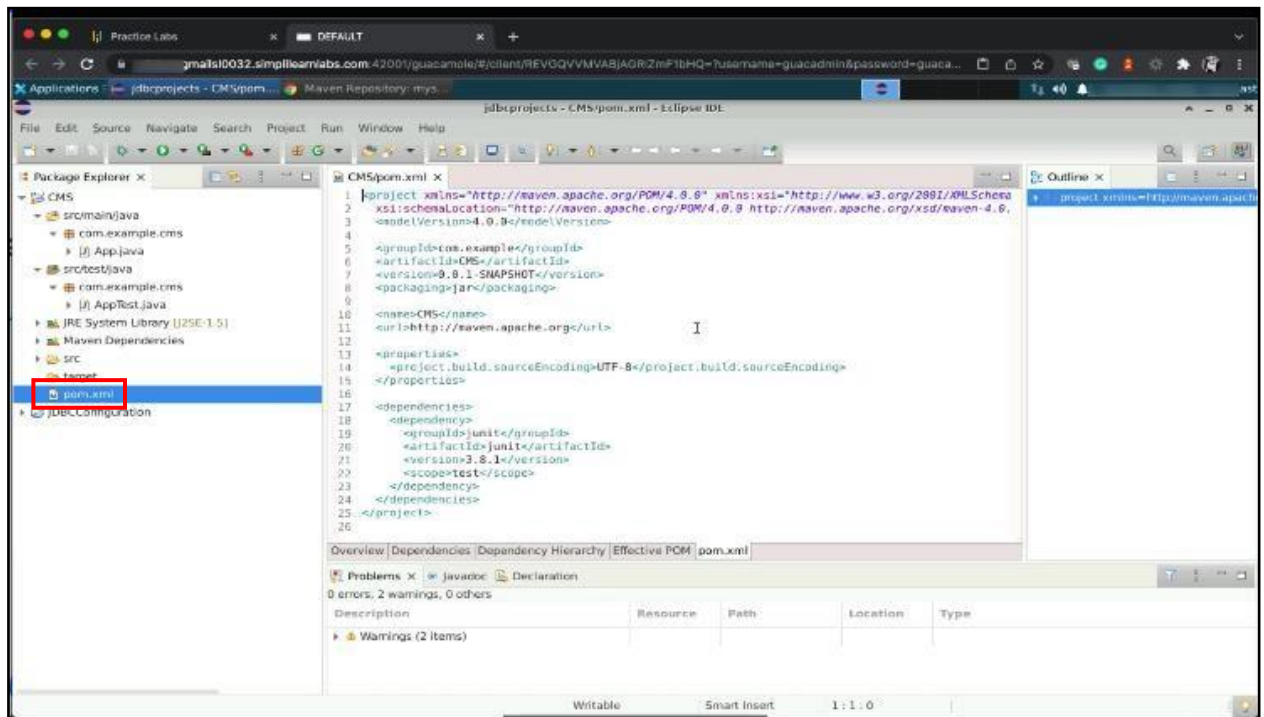


3.9 Add a package name, say **com.example.cms**, and select **Finish** as shown in the screenshot below:

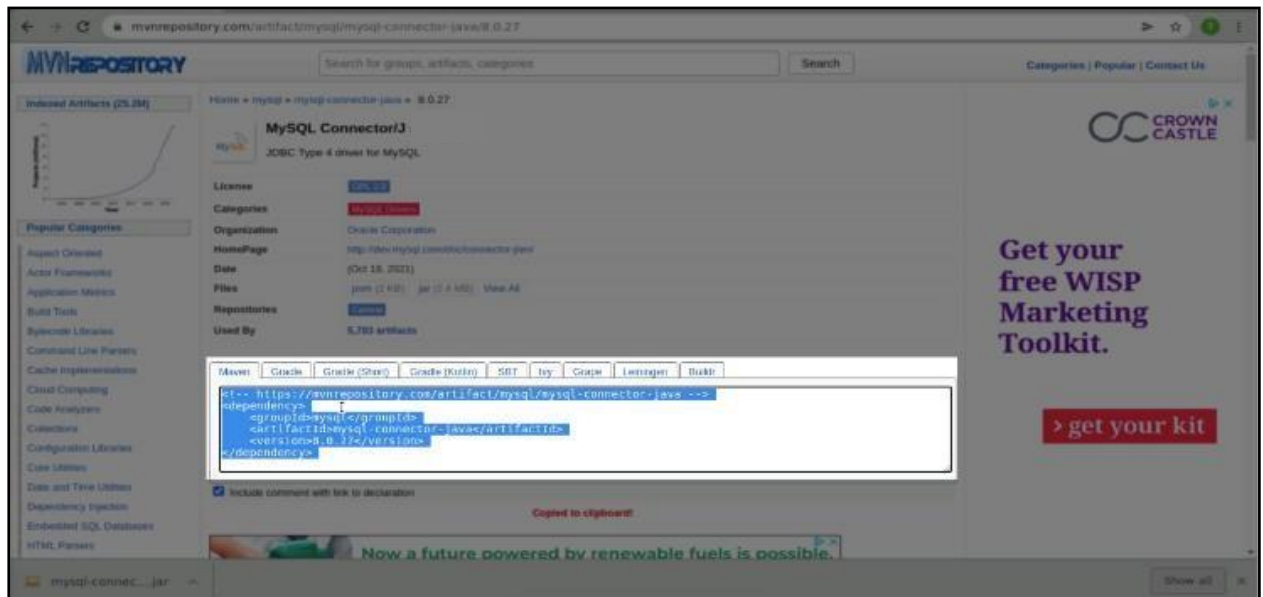


Step 4: Create an interface in com.example.cms

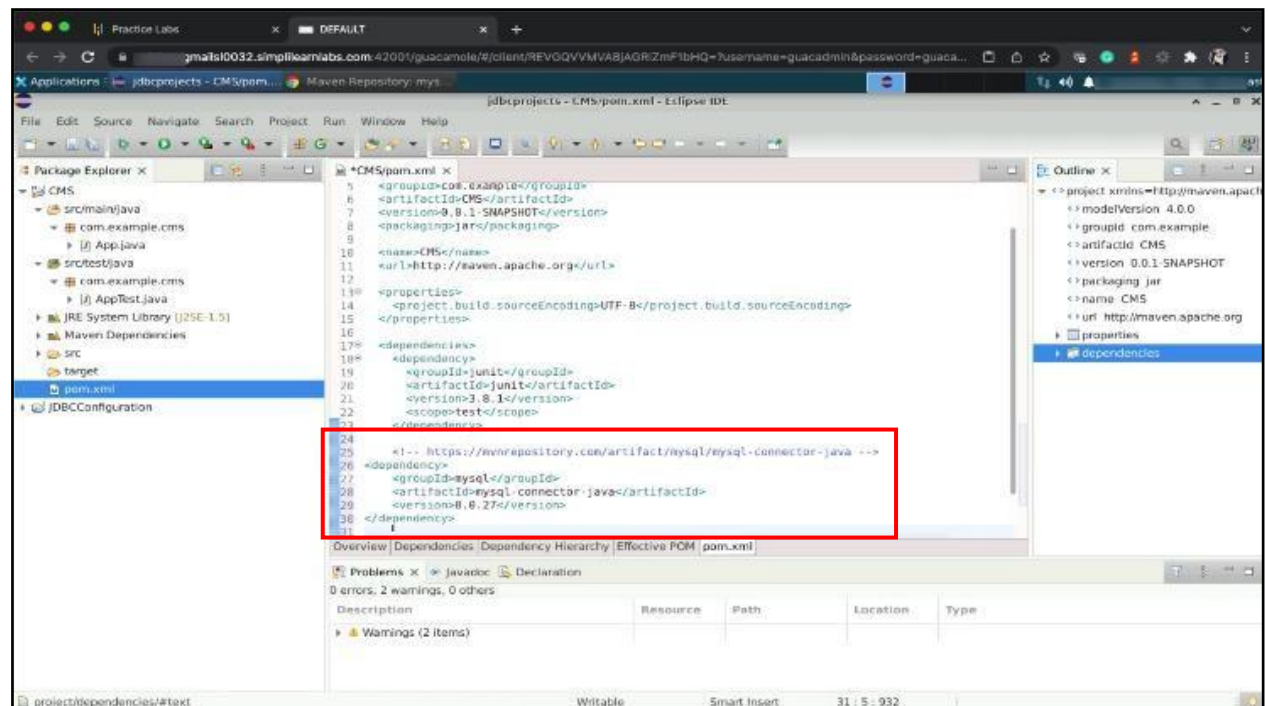
4.1 Go to the **pom.xml** file in the left bar as shown in the screenshot below:



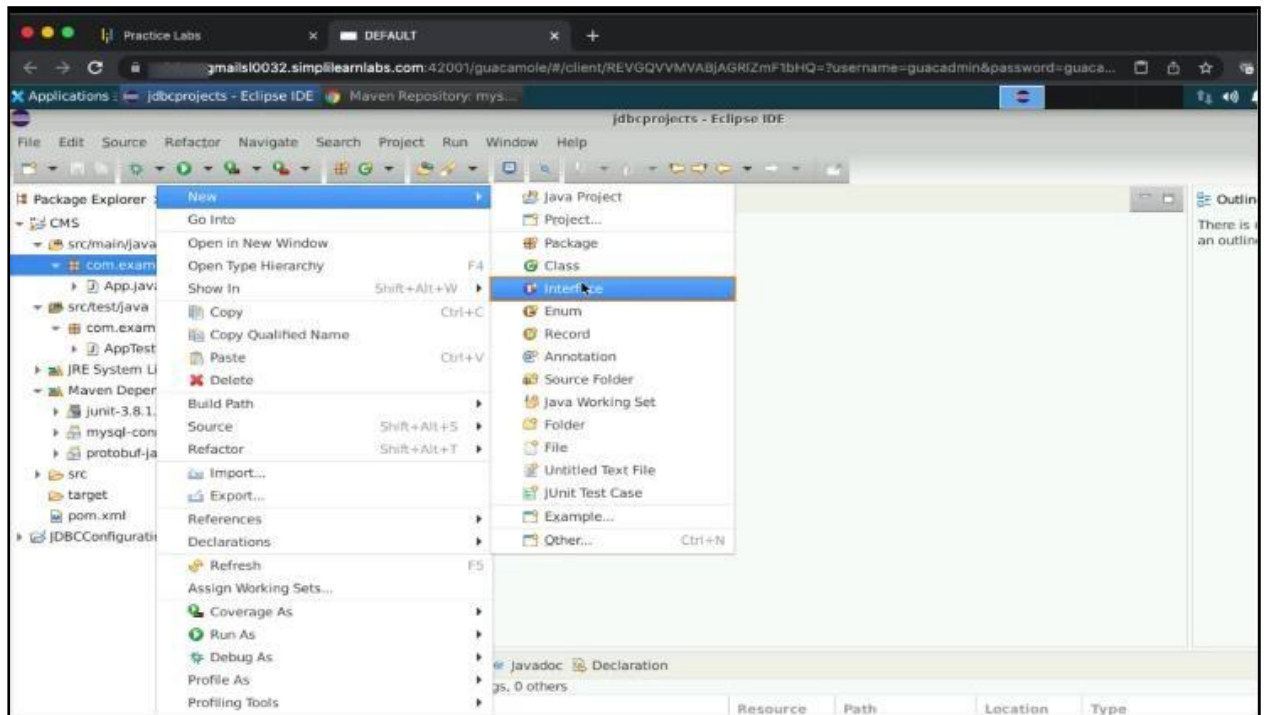
4.2 Return to **mvnrepository** and copy the code for the mysql-connector dependency as shown in the screenshot below:



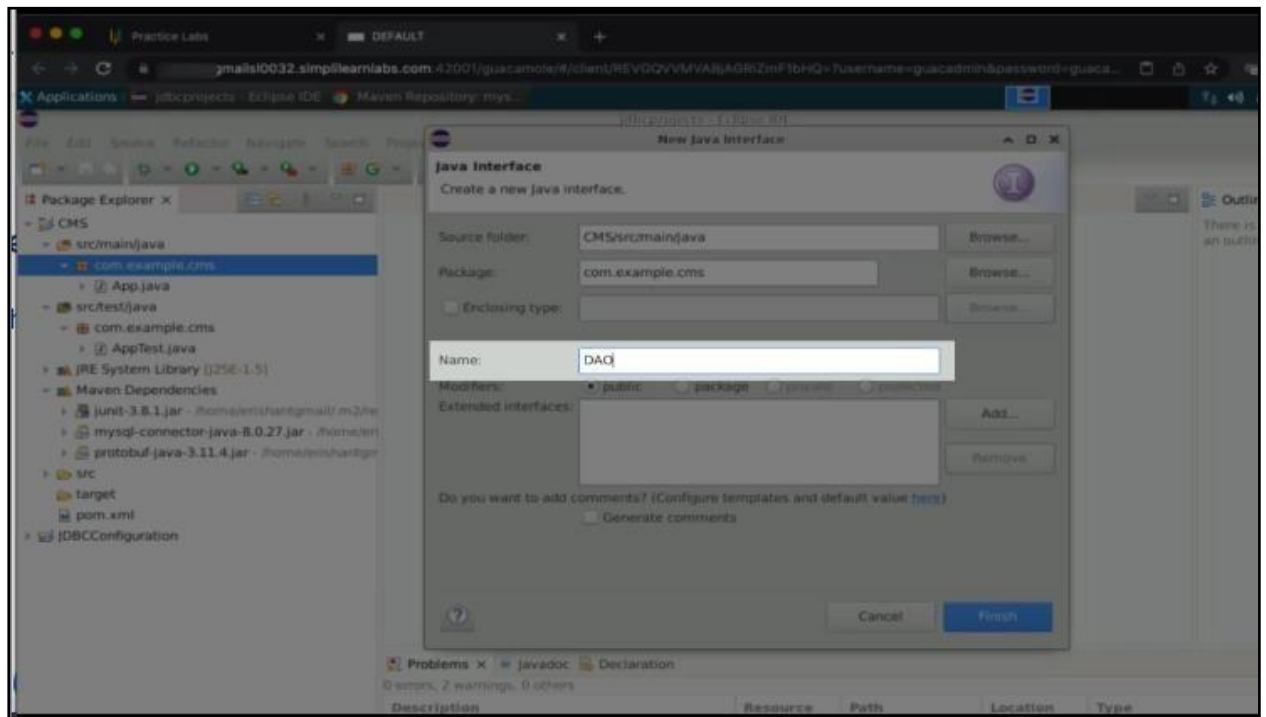
4.3 Paste the dependencies in the **pom.xml** file from line 25 as shown in the screenshot below:



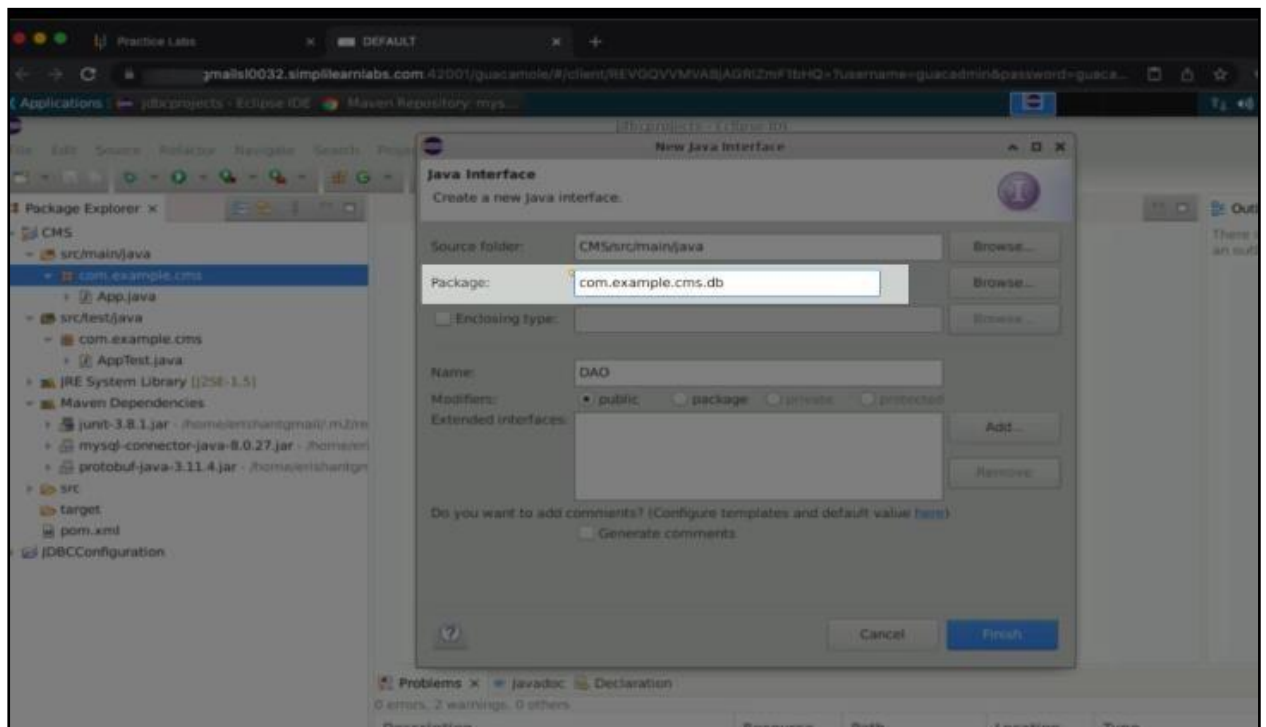
4.4 Right-click on **com.example.cms**, select **New** and create an interface as shown in the screenshot below:



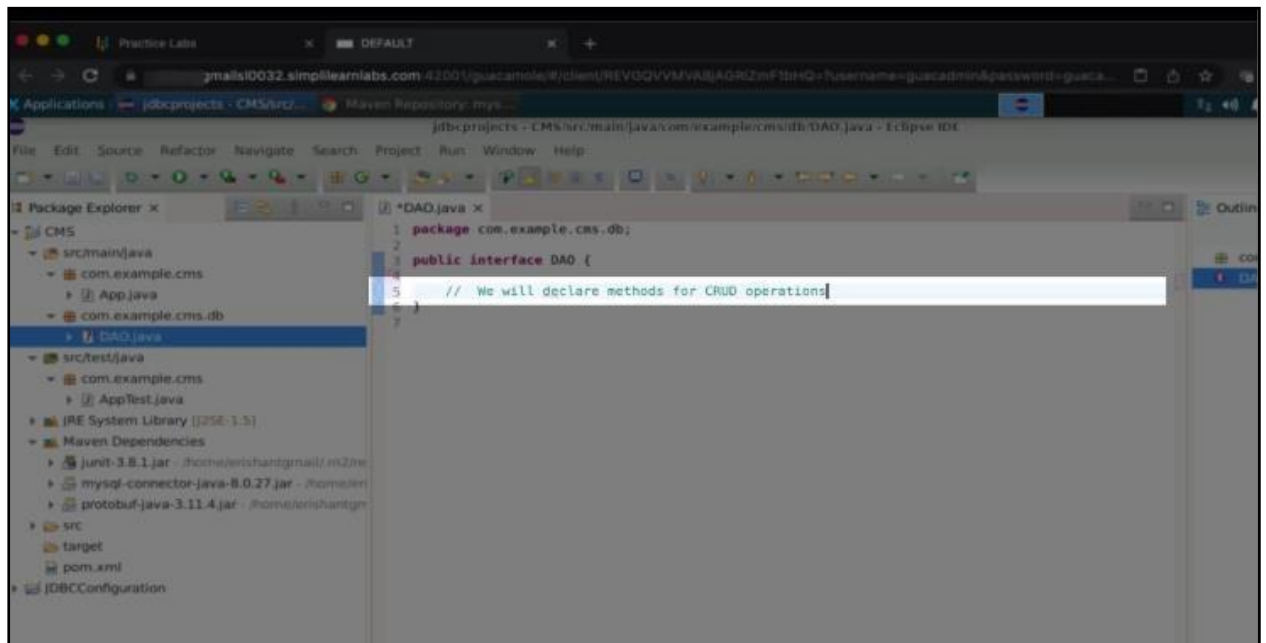
4.5 Provide a name as DAO



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

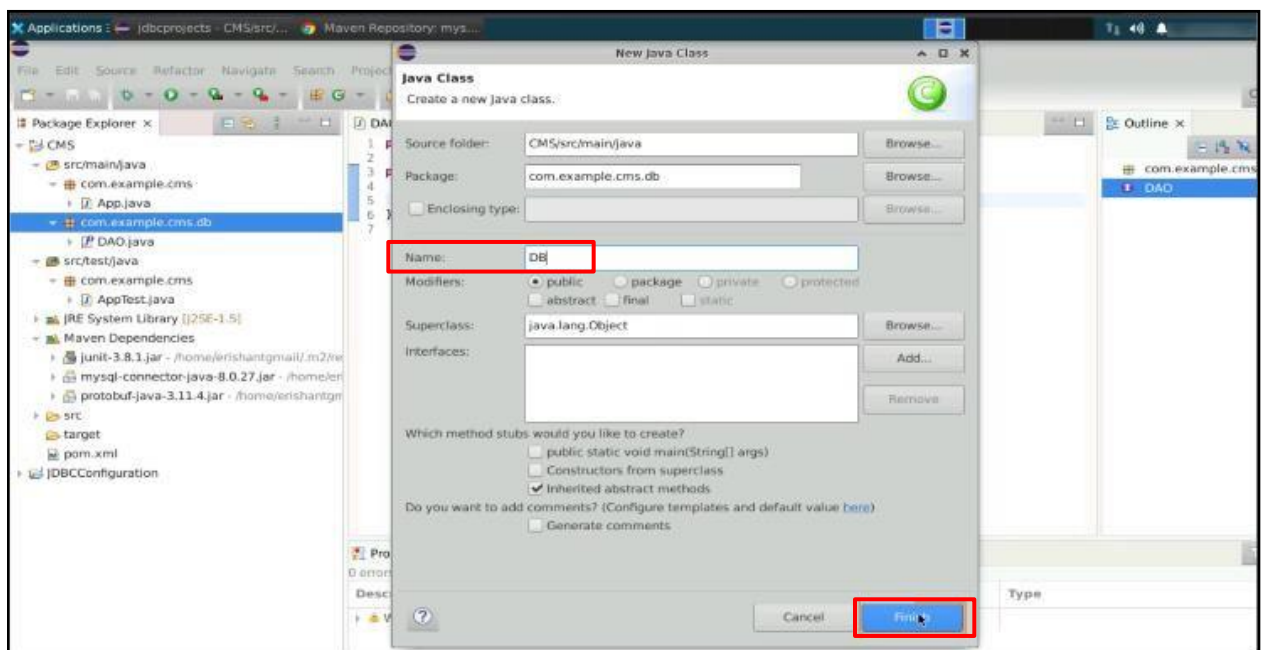


4.7 Define the methods for CRUD operations

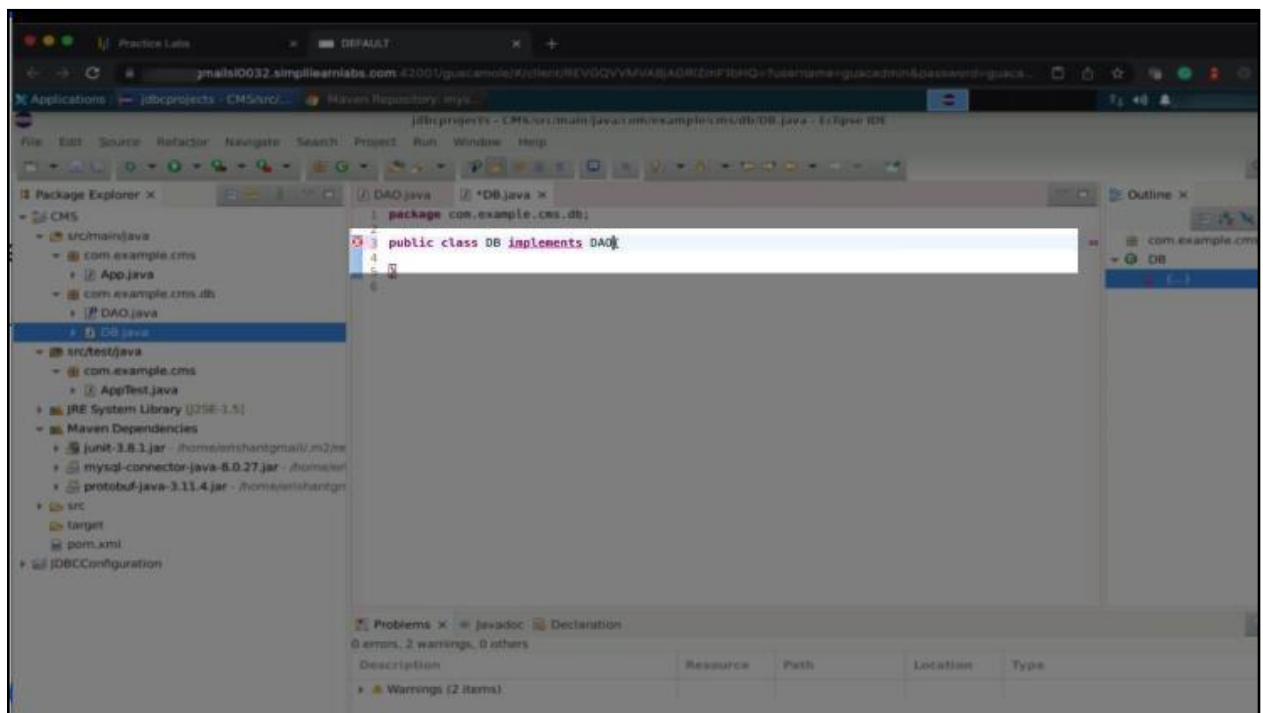


Step 5: Create the DB class

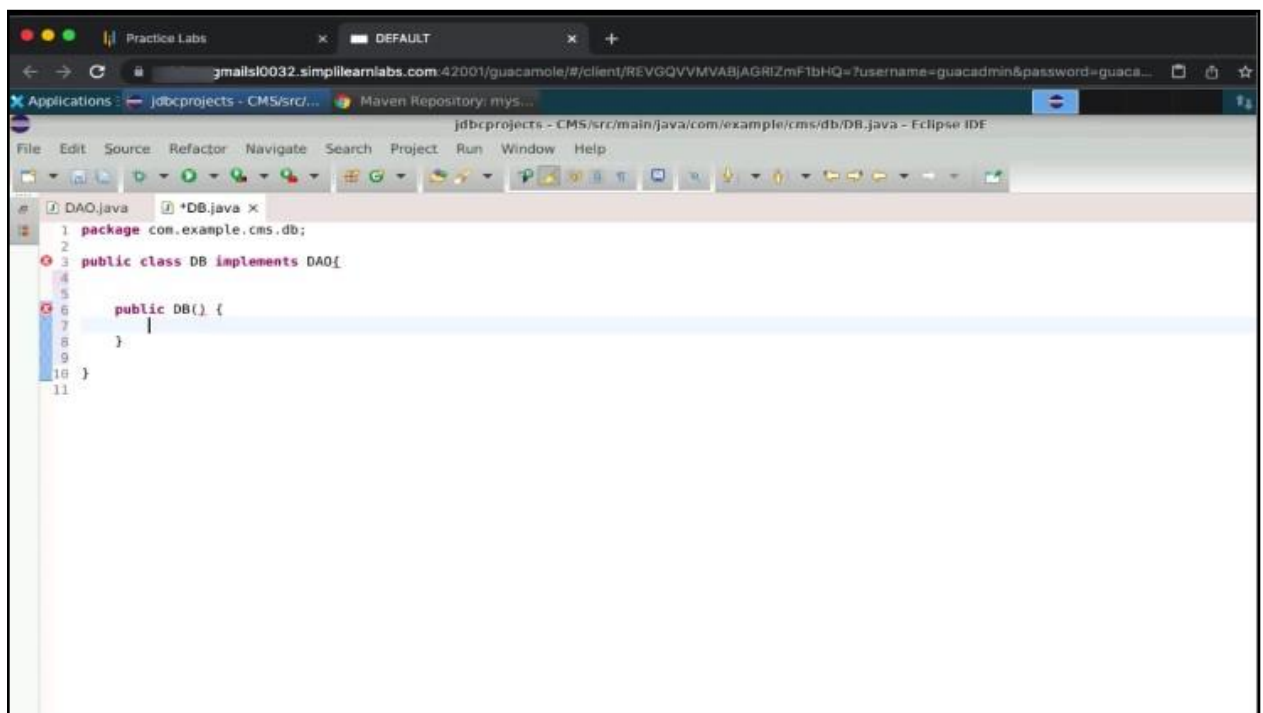
5.1 Create a new class named **DB**, and click on **Finish**



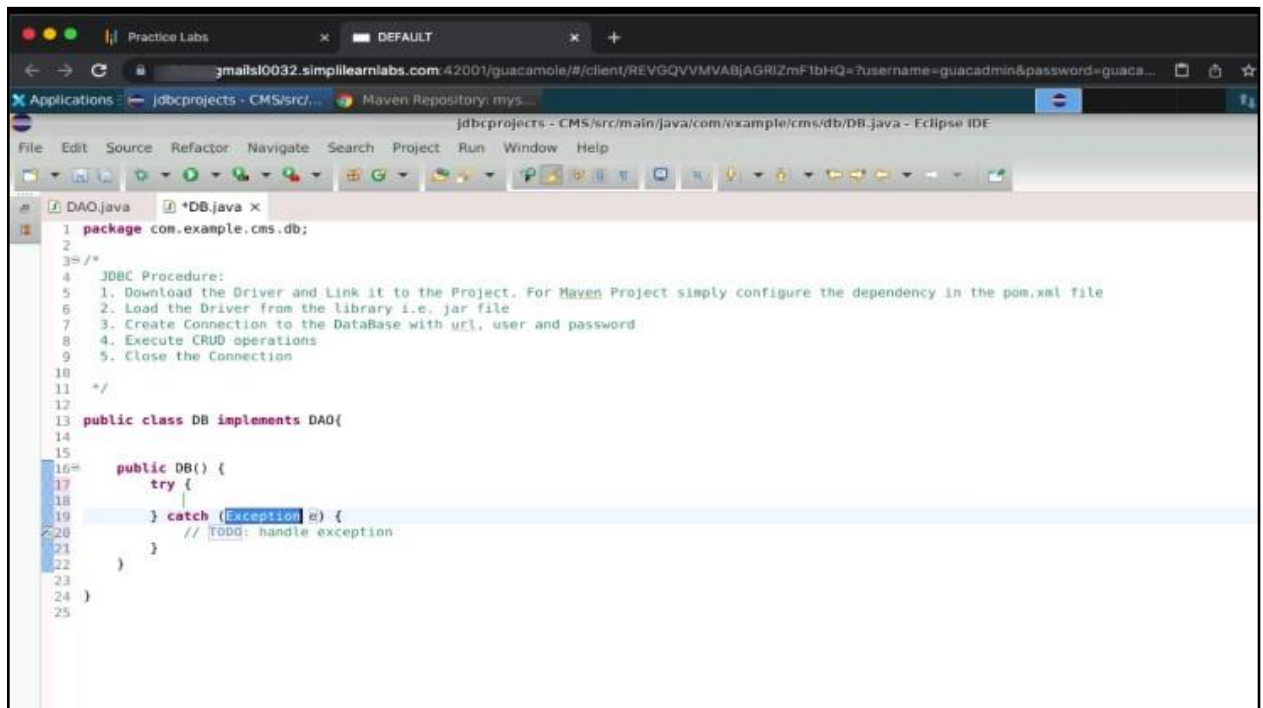
The **DB** class is the implementation file of the **DAO** interface.



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

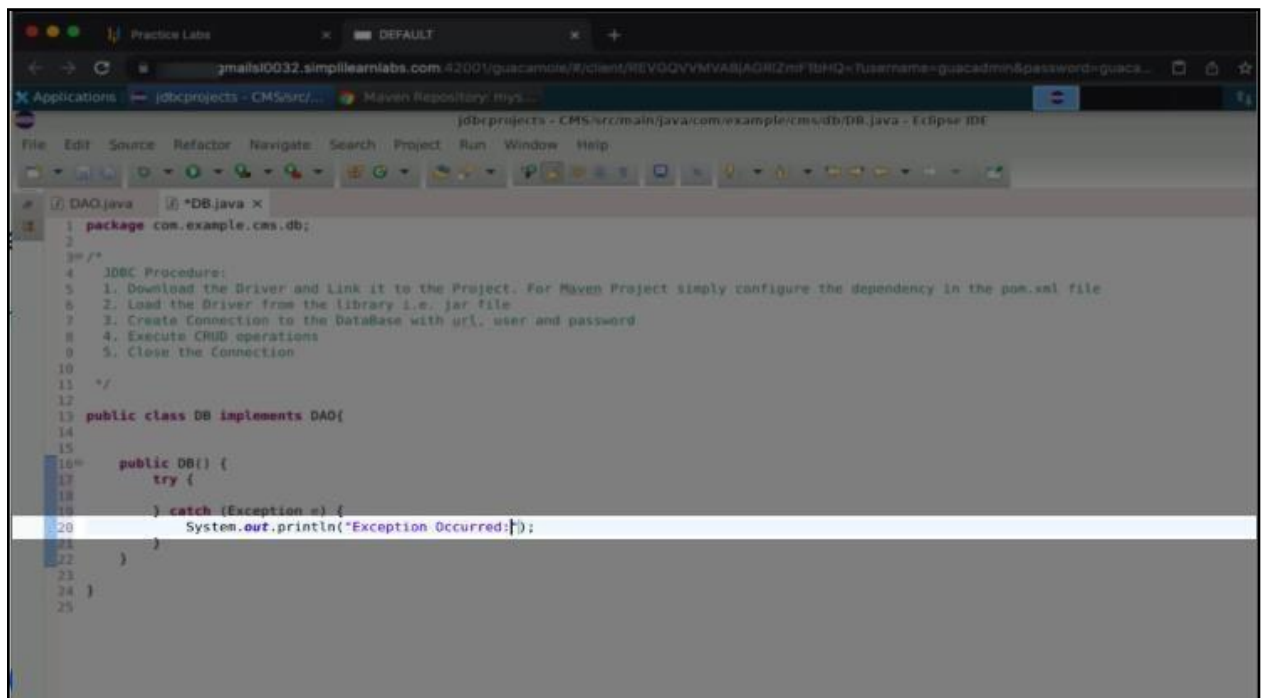


```

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     public DB() {
15         try {
16
17         } catch (Exception e) {
18             // TODO: handle exception
19         }
20     }
21 }
22
23
24
25

```

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     public DB() {
15         try {
16
17         } catch (Exception e) {
18             System.out.println("Exception Occurred:");
19         }
20     }
21 }
22
23
24
25

```

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 public class DB implements DAO{
13
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

```

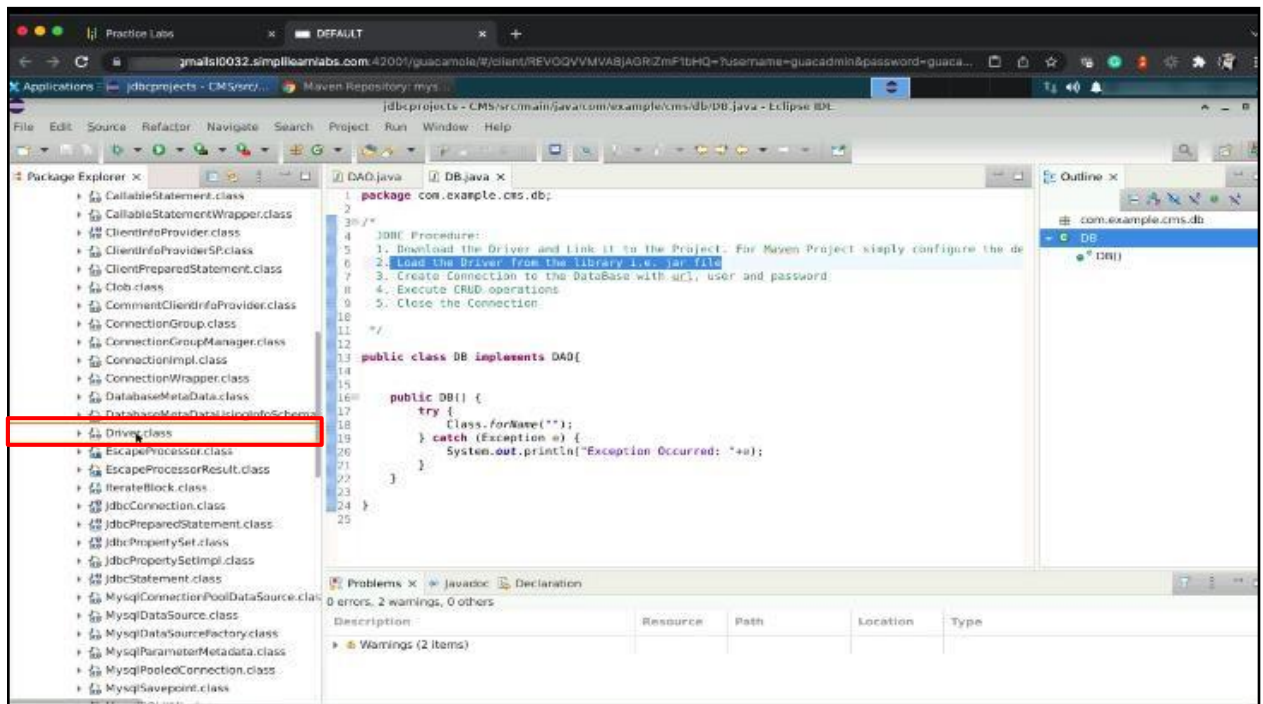
5.6 Open **mysql-connector-java-8.0.27.jar** and find the package containing the driver class

```

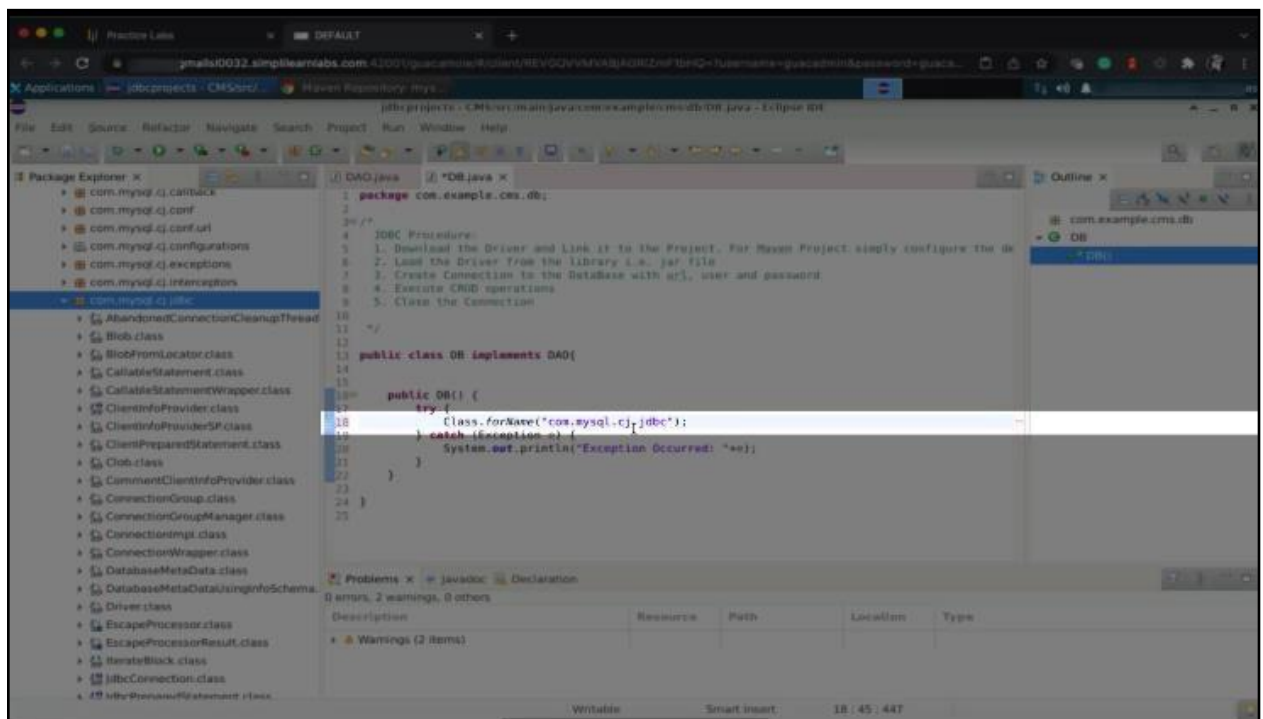
1 package com.example.cms.db;
2
3 /**
4  * JDBC Procedure:
5  * 1. Download the Driver and link it to the Project. For Maven Project simply configure the de
6  * 2. Load the Driver from the library i.e. jar file
7  * 3. Create Connection to the DataBase with url, user and password
8  * 4. Execute CRUD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

```

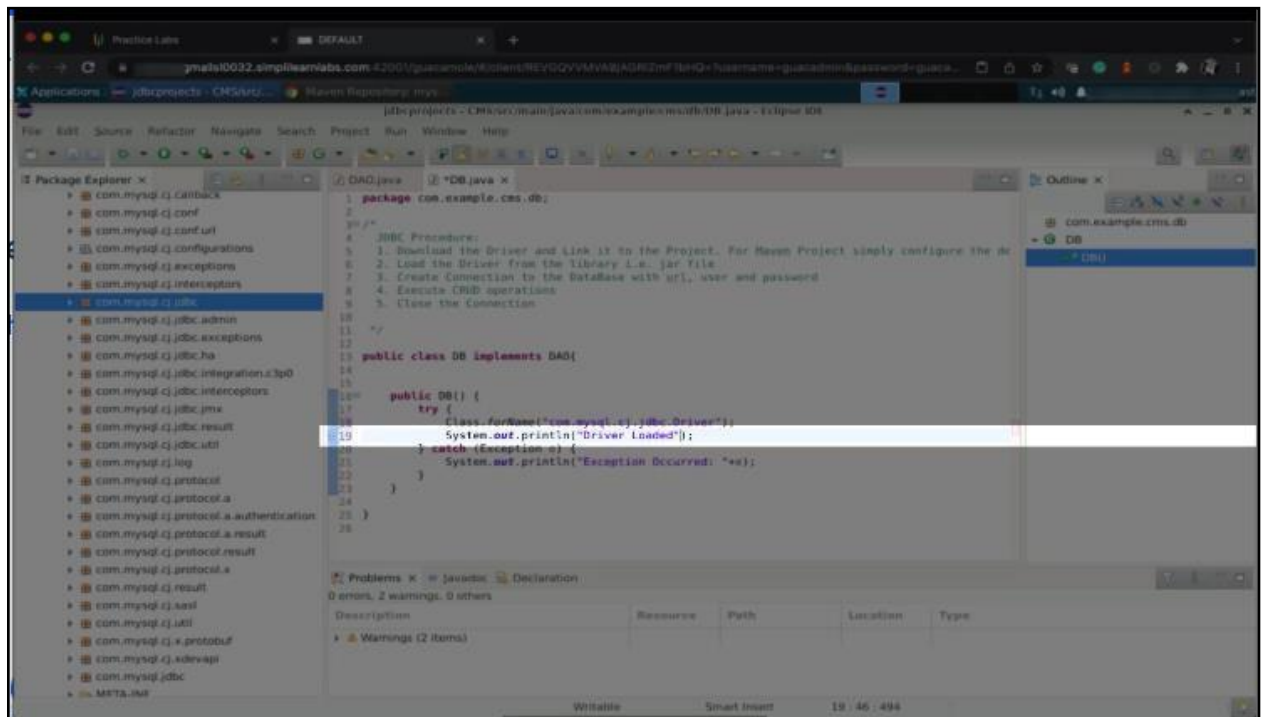
5.7 Select the **Driver** class as shown in the screenshot below:



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

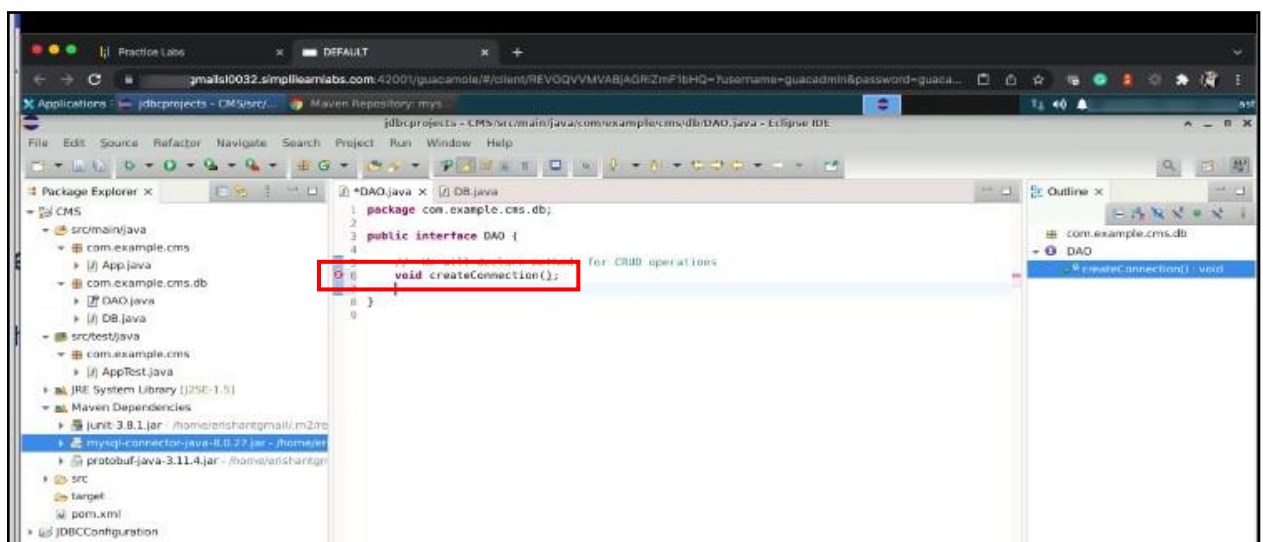


5.9 Include a print statement to load the driver as shown in the screenshot below:

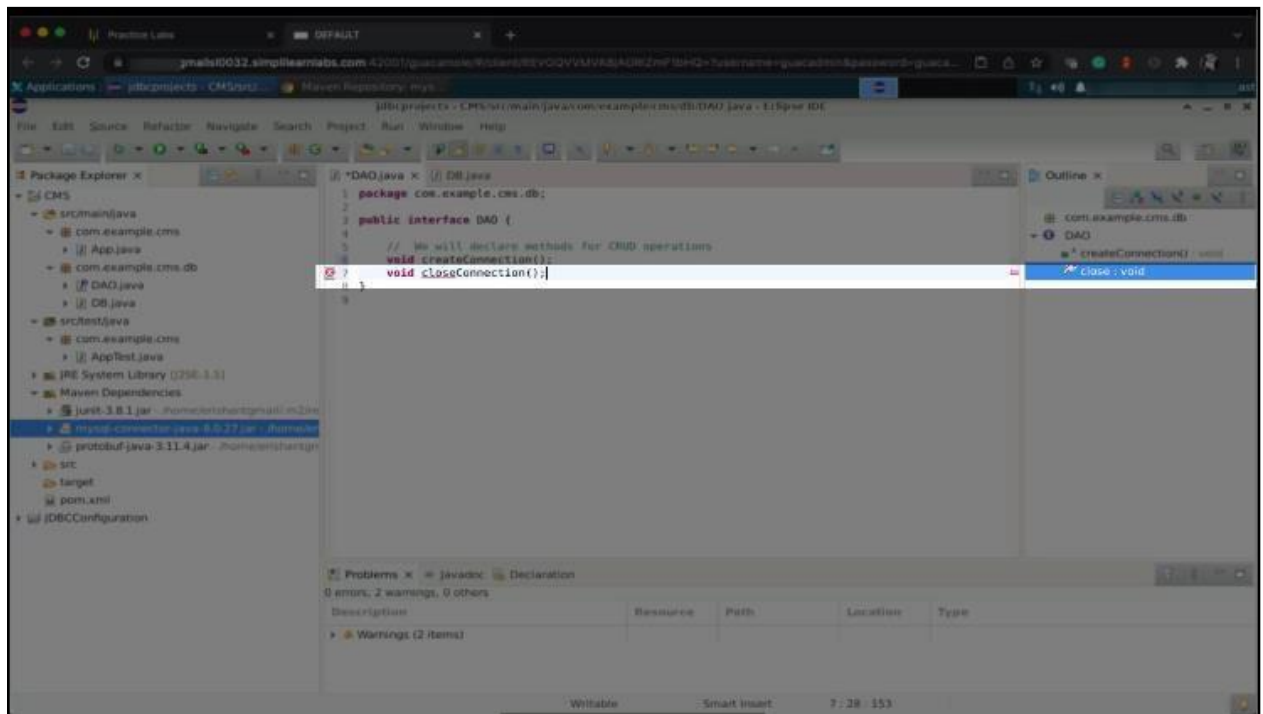


Step 6: Create a connection in the DAO.java file

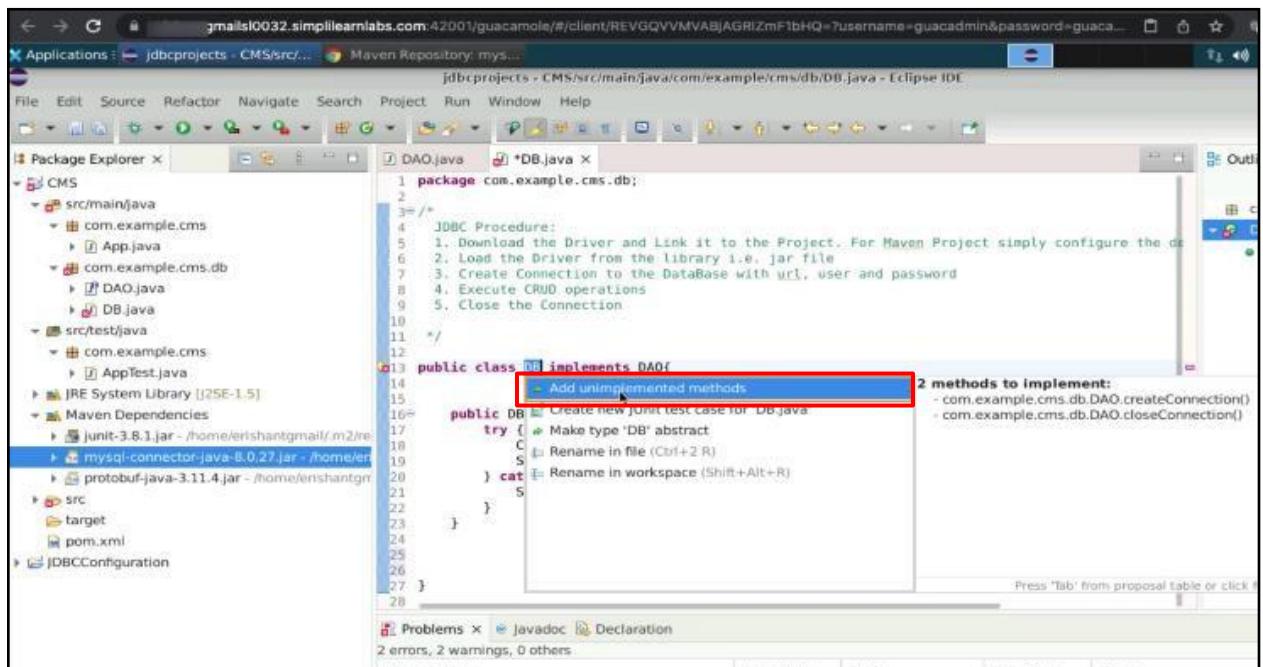
6.1 Create a connection in the **DAO.java** file as shown in the screenshot below:



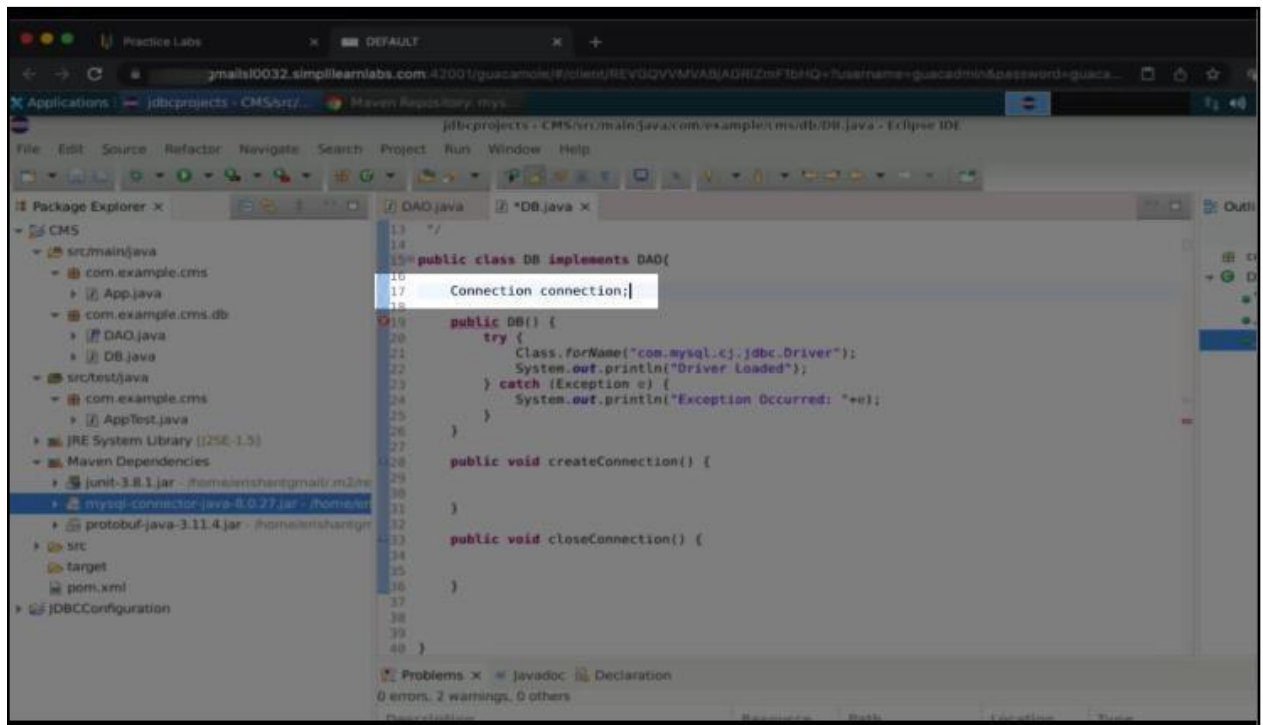
6.2 Add the `closeConnection()` method in the same file:



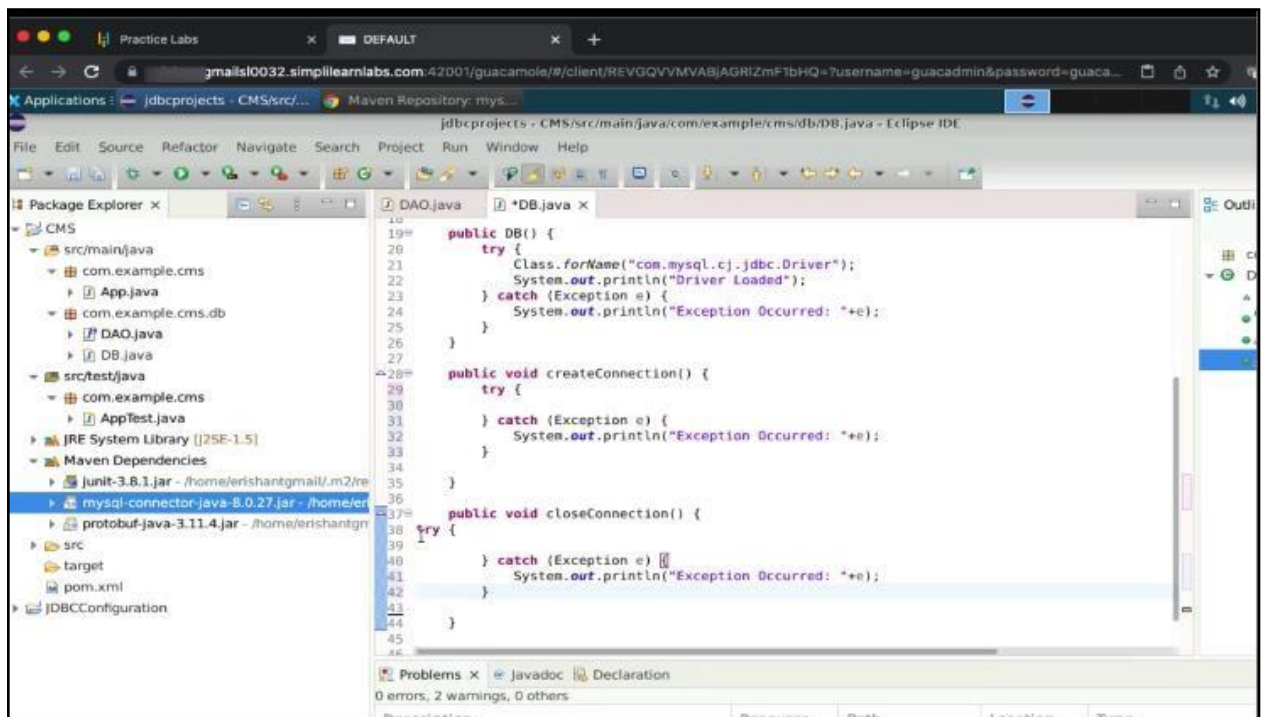
6.3 Select **Add unimplemented methods** in the `DB.java` file as shown in the screenshot below:



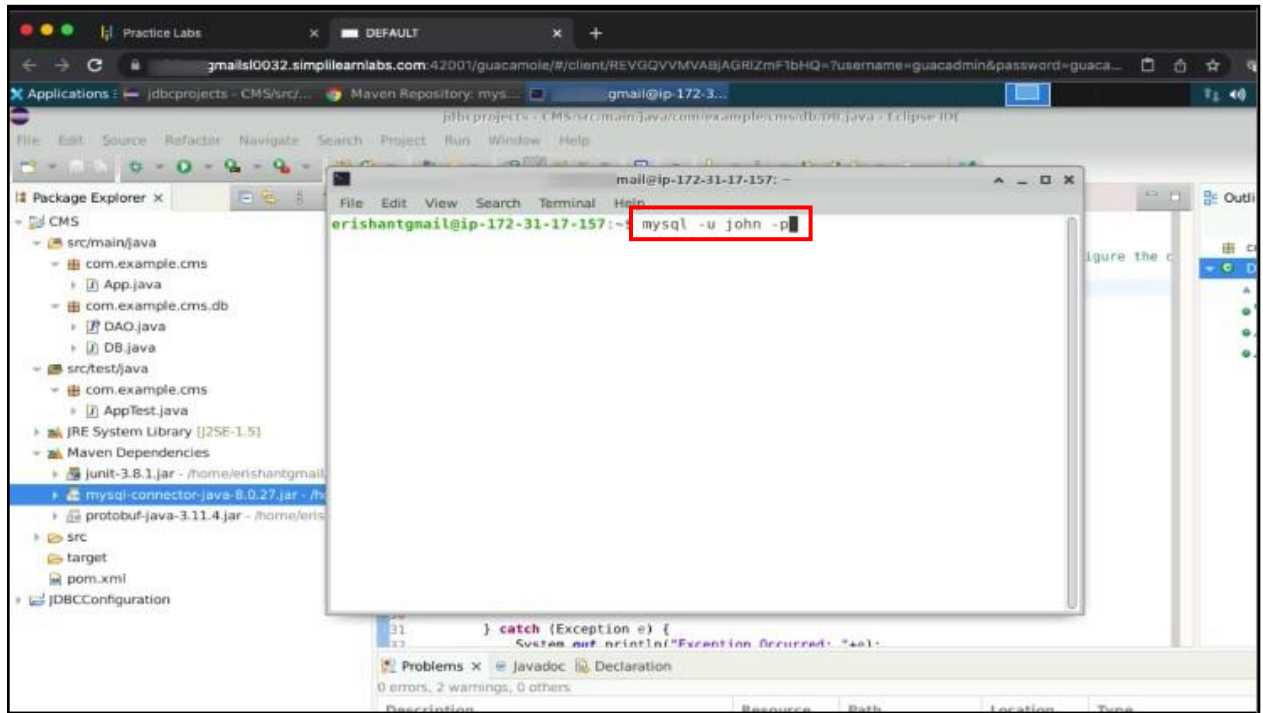
6.4 Add an API from JDBC



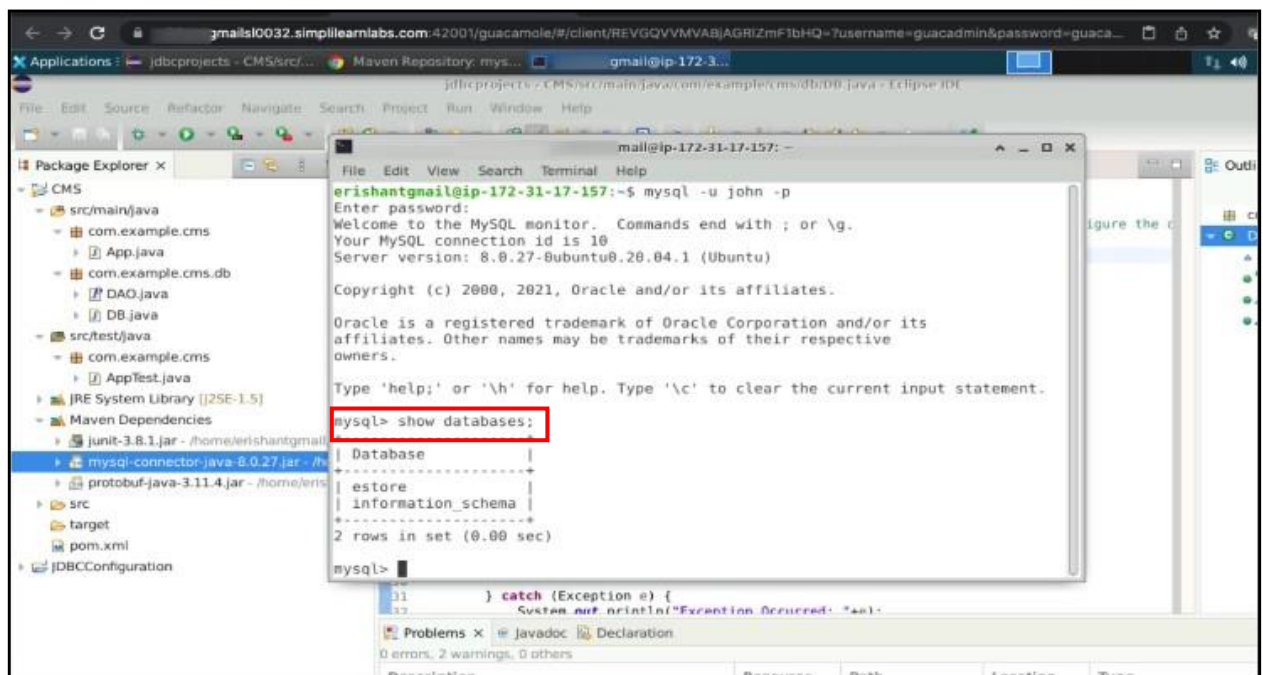
6.5 Use try-catch in both the methods as shown in the screenshot below:



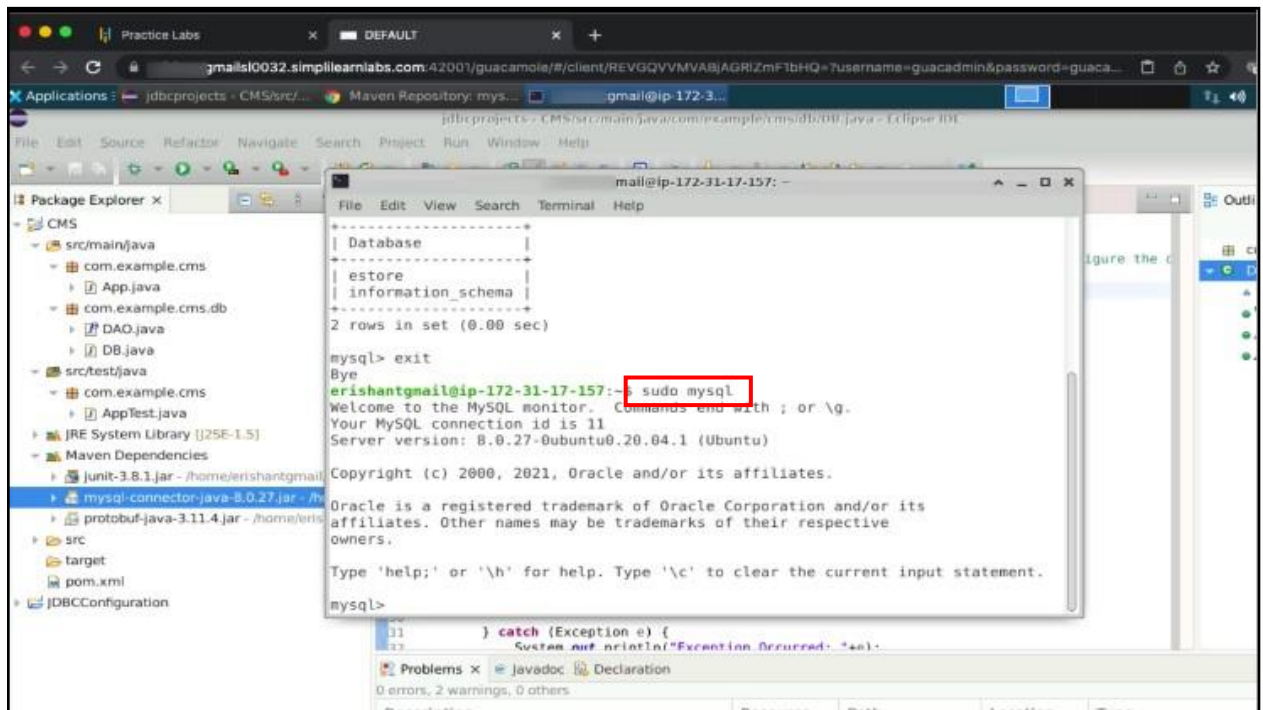
6.6 Open the terminal window and type the command:
mysql -u john -p



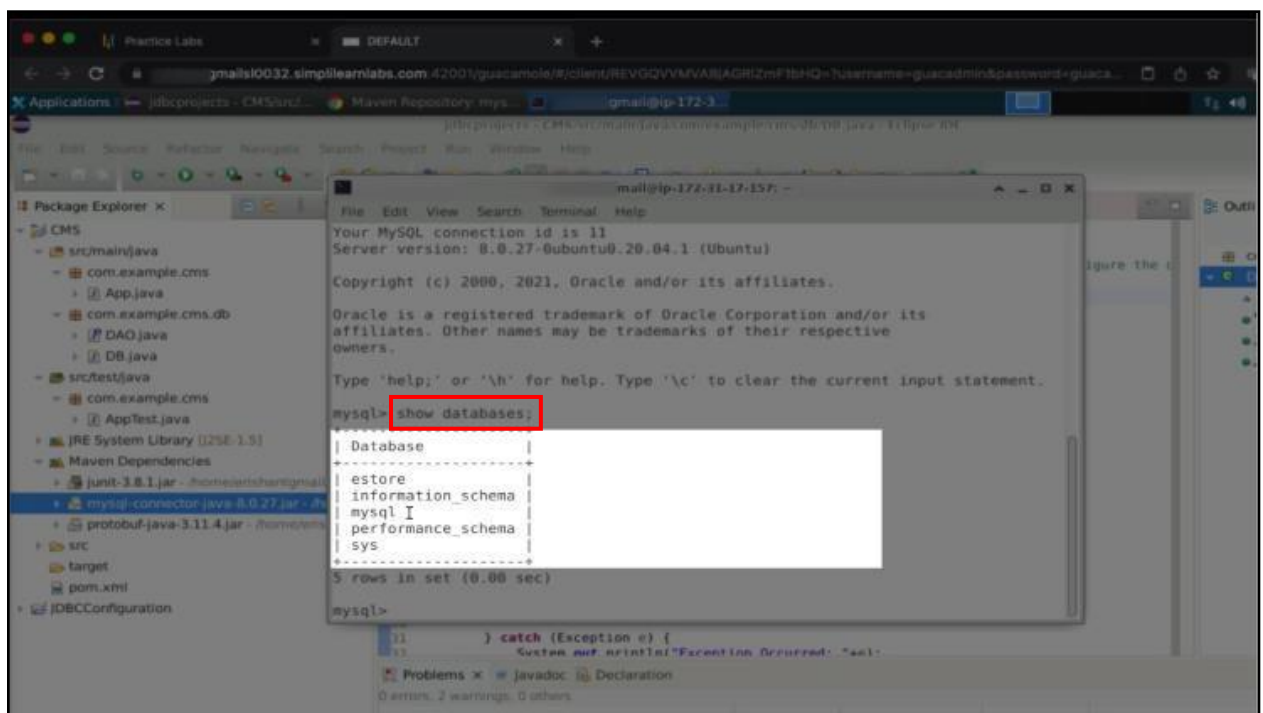
6.7 Type the **show databases;** command to see the available database names



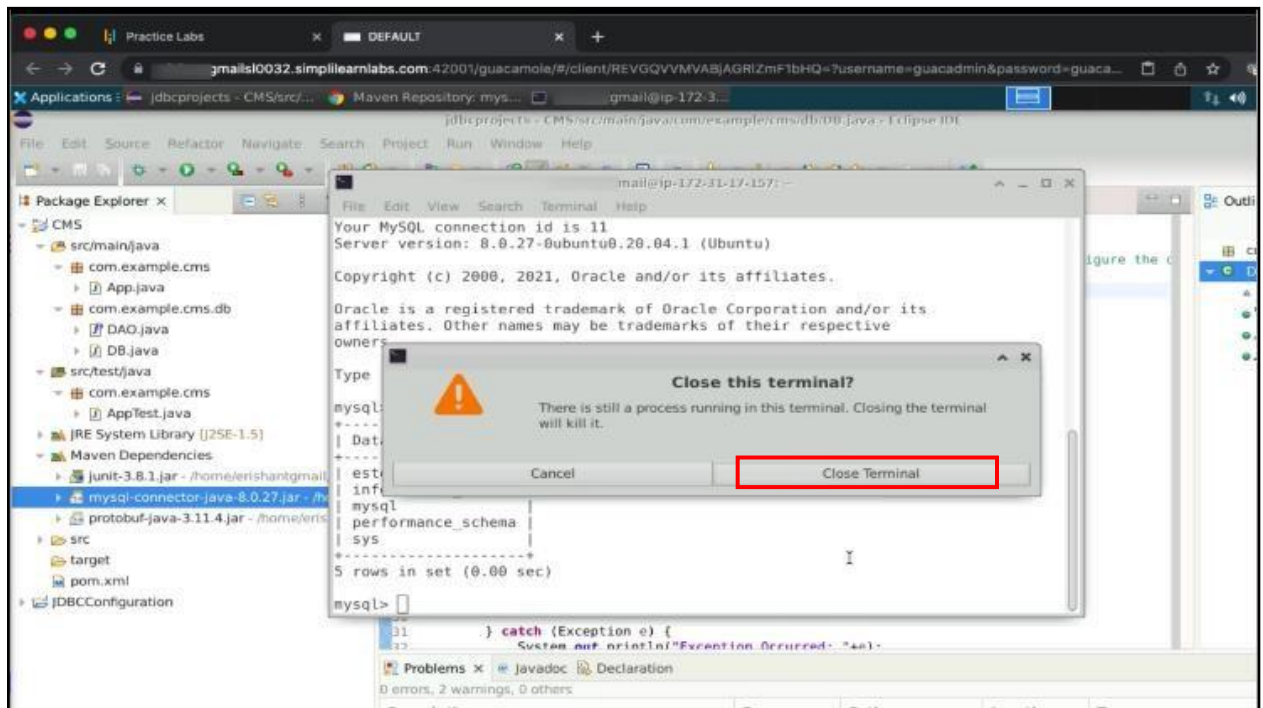
6.8 Type the **sudo mysql** command to log in as a **sudo user**



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

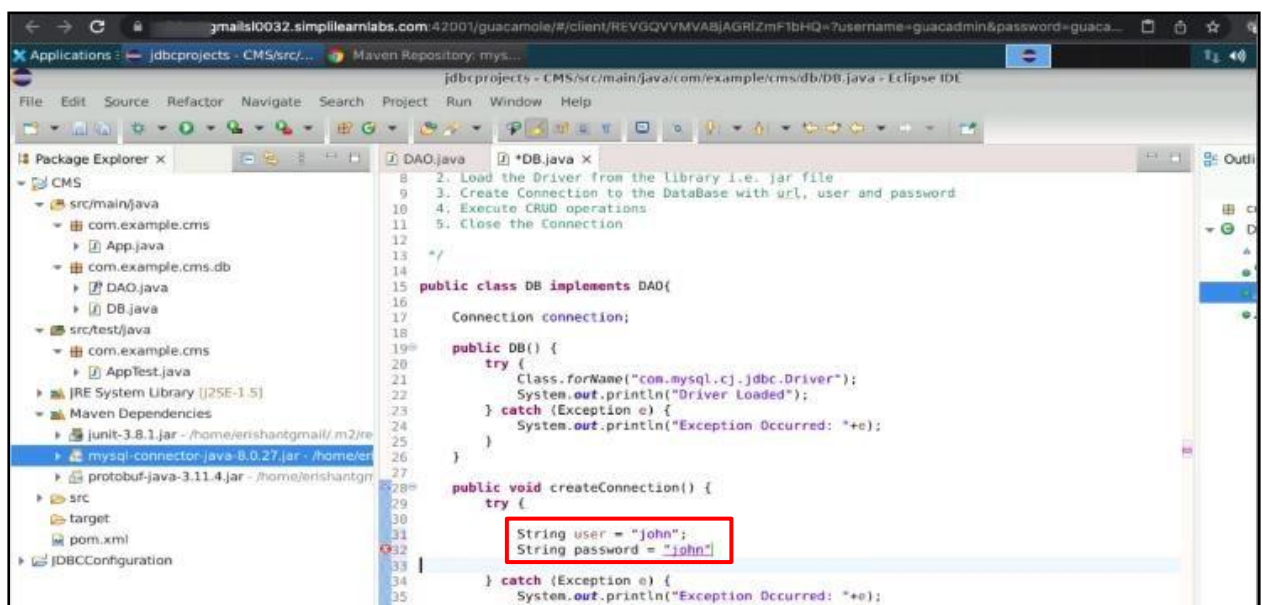


6.10 Close the terminal

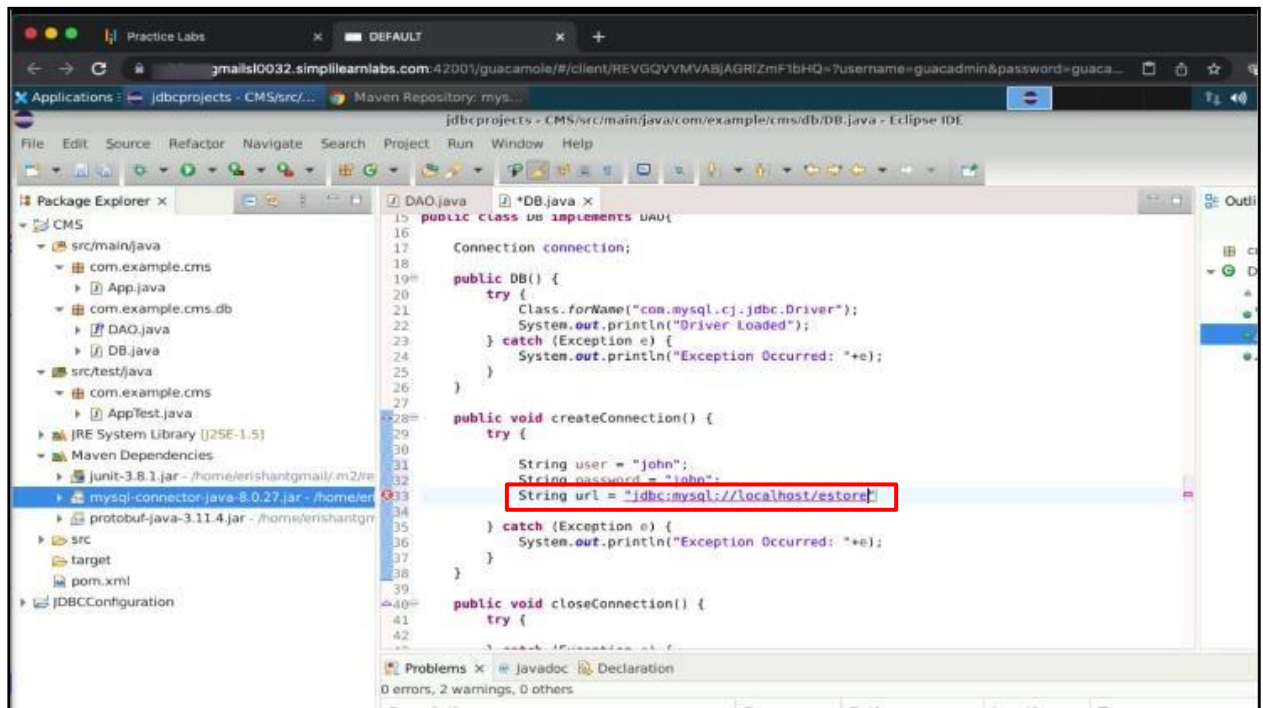


Step 7: Initialize the connection

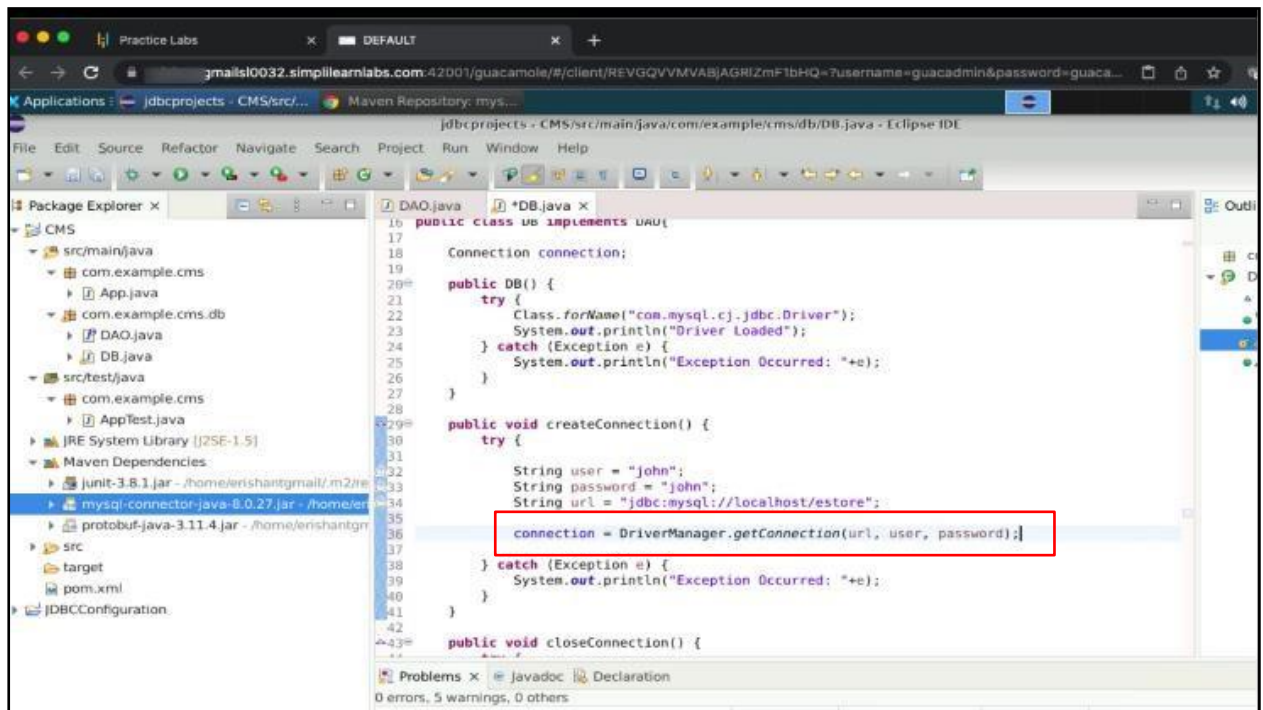
7.1 Add a **username** and **password** for the connection as shown in the screenshot below:



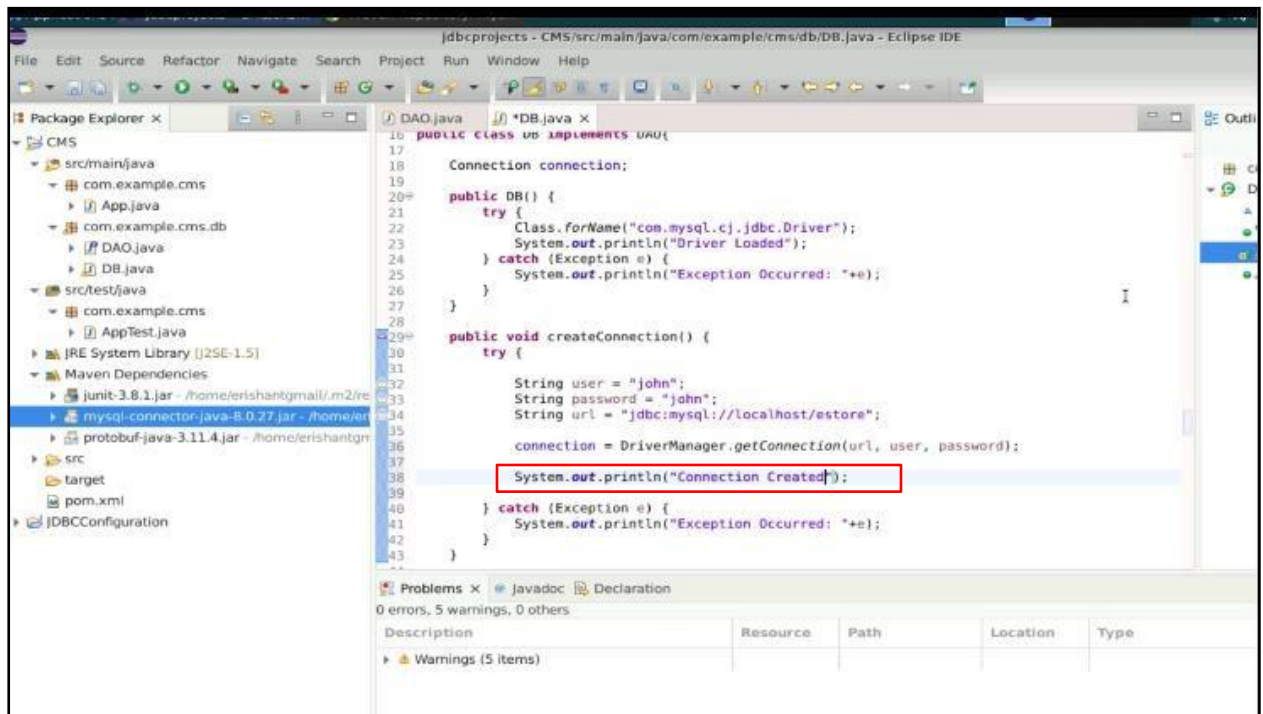
7.2 Add a URL to connect MySQL as shown in the screenshot below:



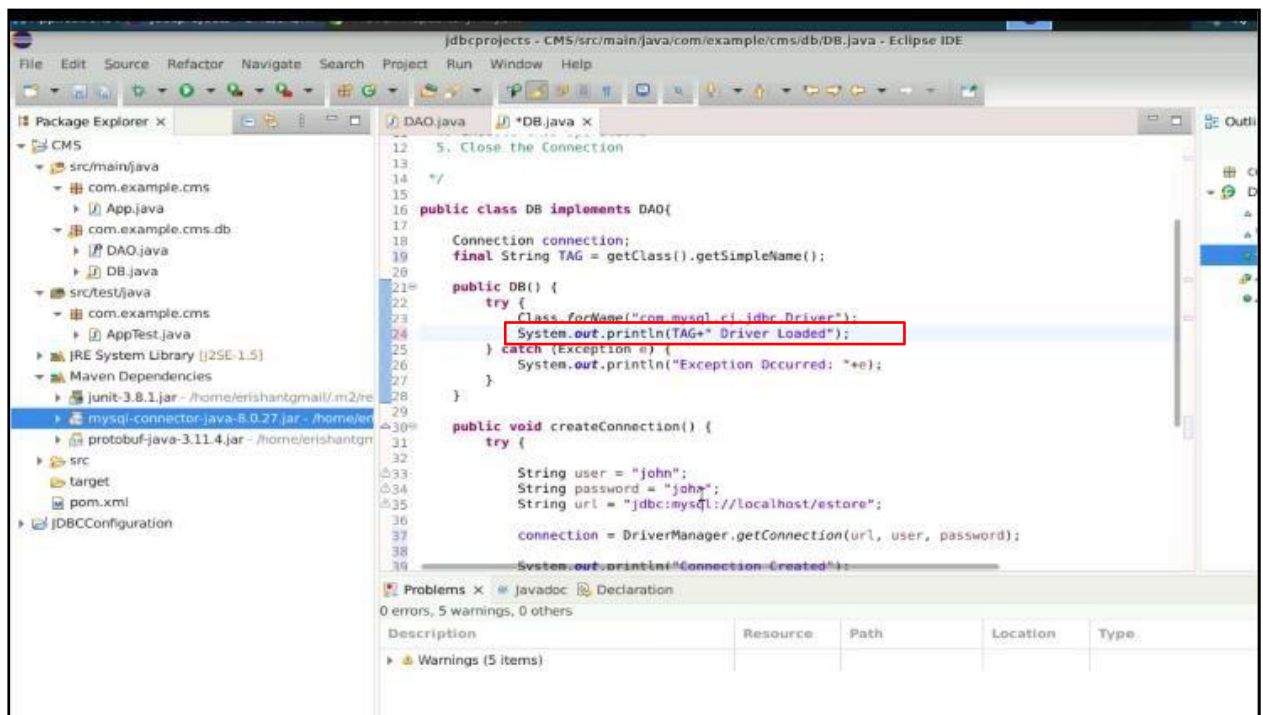
7.3 Initialize the connection as shown in the screenshot below:



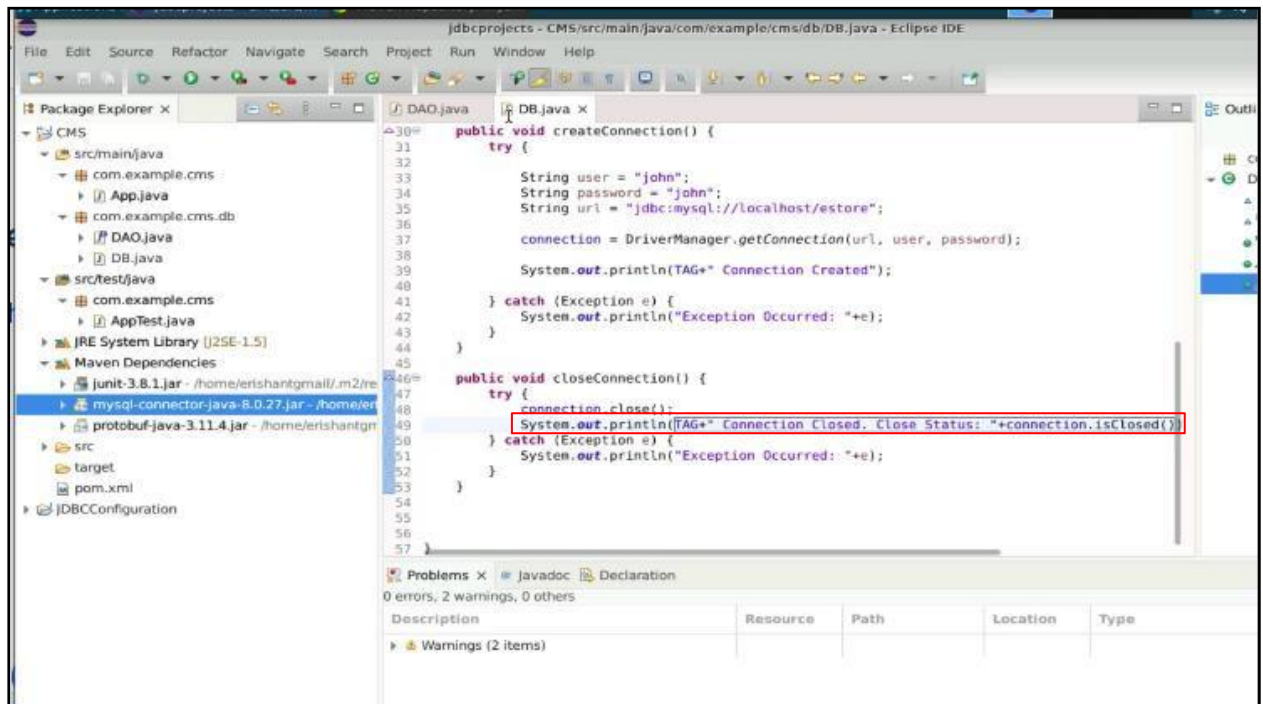
7.4 Add a print statement to show that the connection is created as shown in the screenshot below:



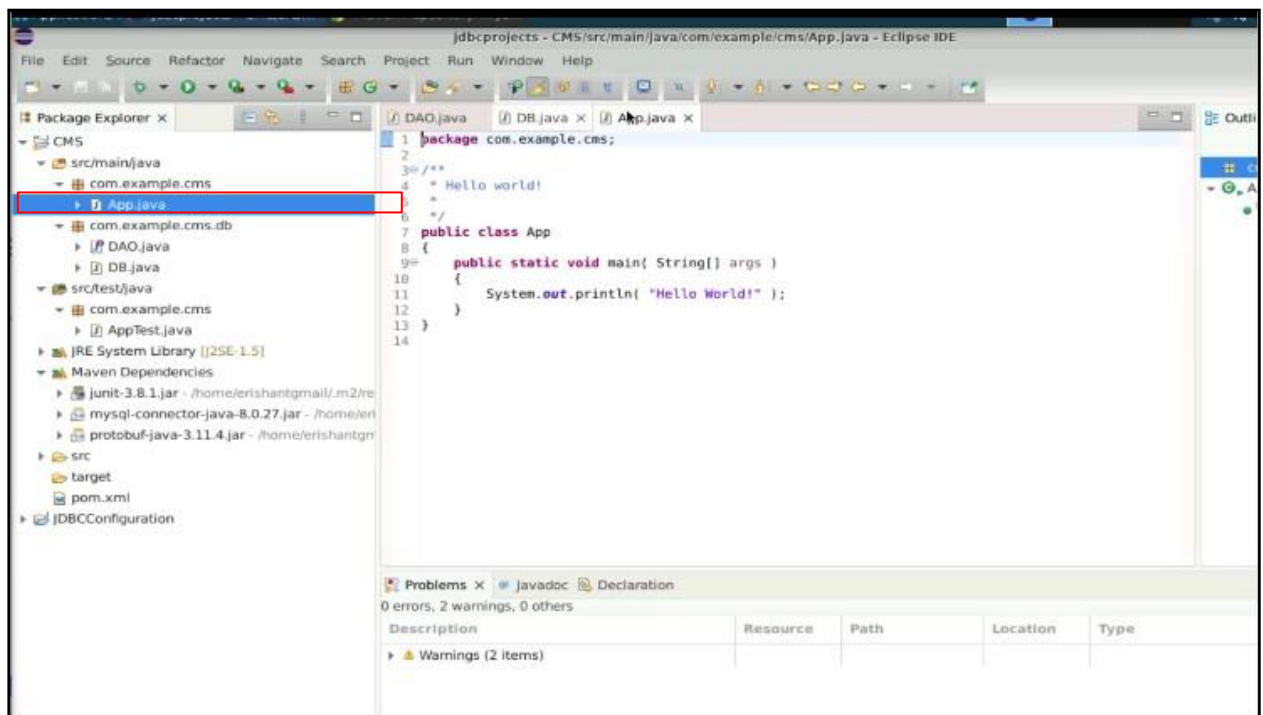
7.5 Type the final string tag as shown in the screenshot below:



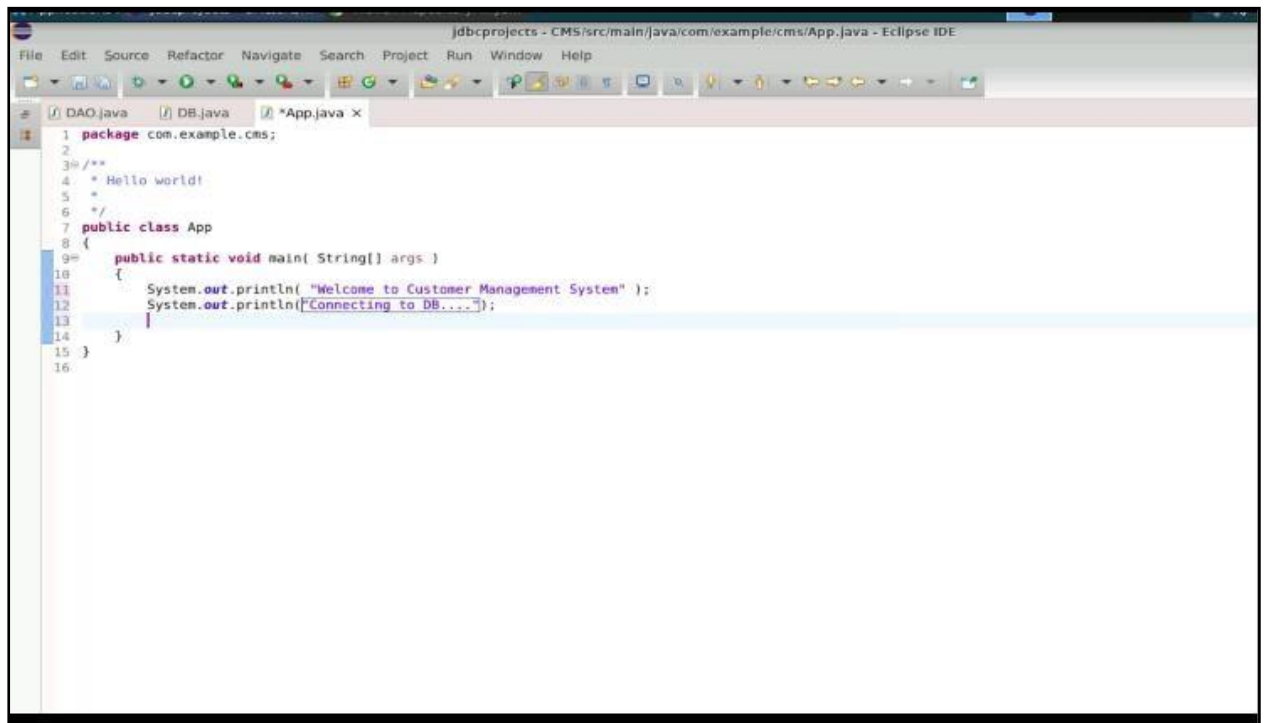
7.6 Execute the **closeConnection** method as shown in the screenshot below:



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



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

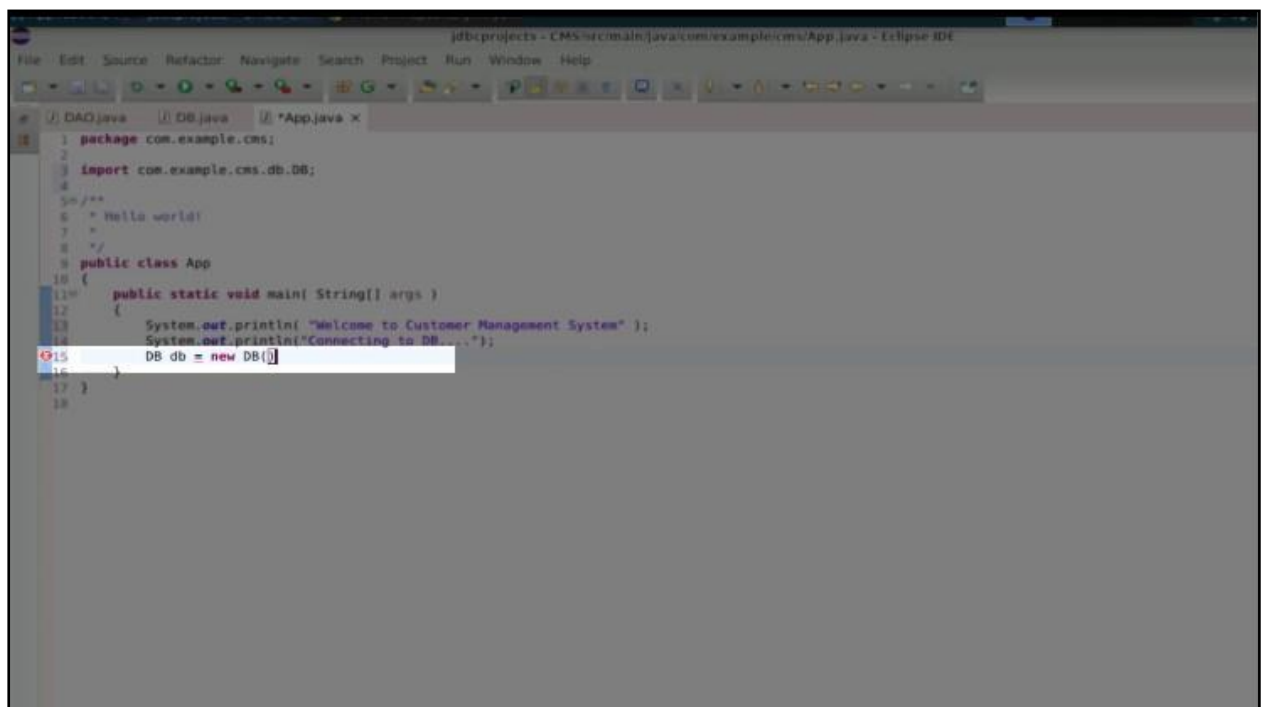


```

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:



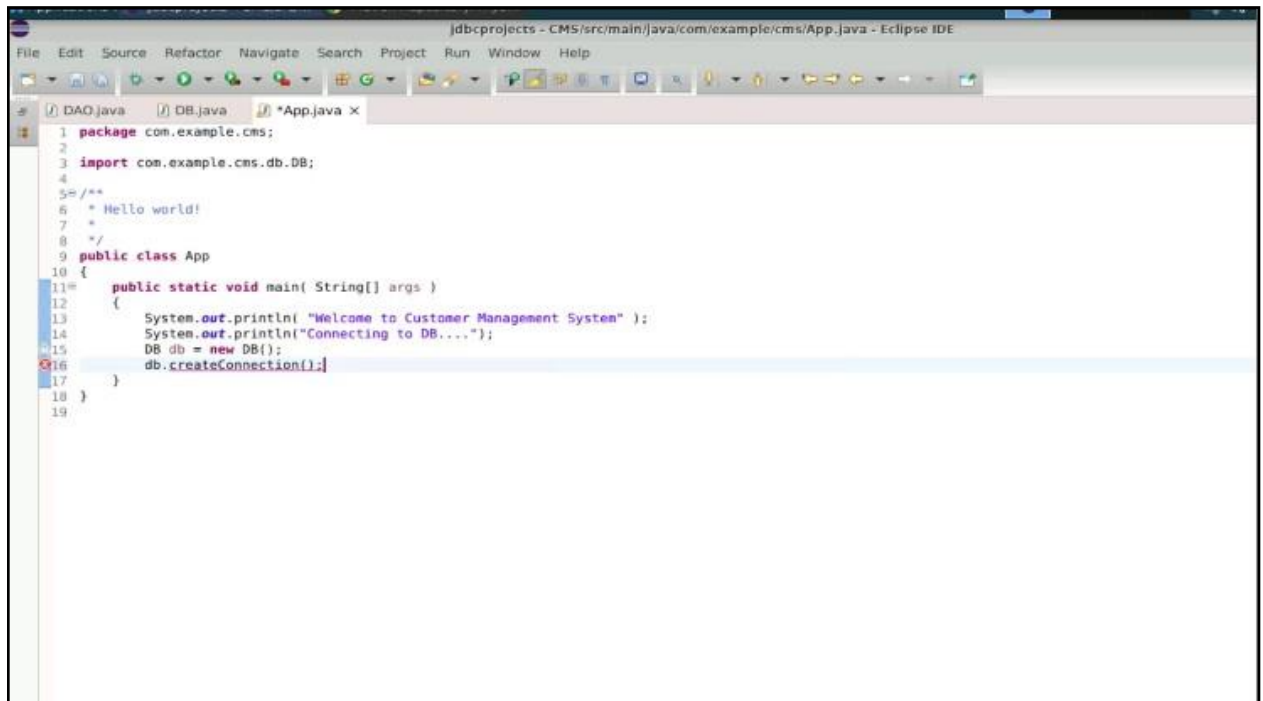
```

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 }
18

```

Step 8: Execute the createConnection method

8.1 Execute the `createConnection()` method

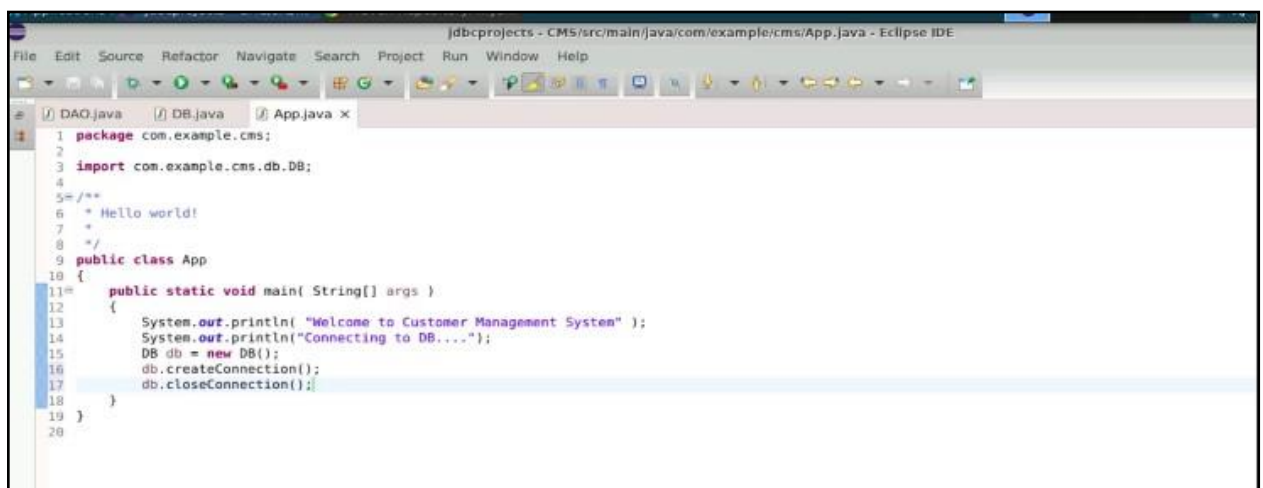


The screenshot shows the Eclipse IDE with the file `App.java` open. The code is as follows:

```
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
```

The line `db.createConnection();` on line 16 is highlighted with a blue selection bar.

8.2 Execute `closeConnection()`

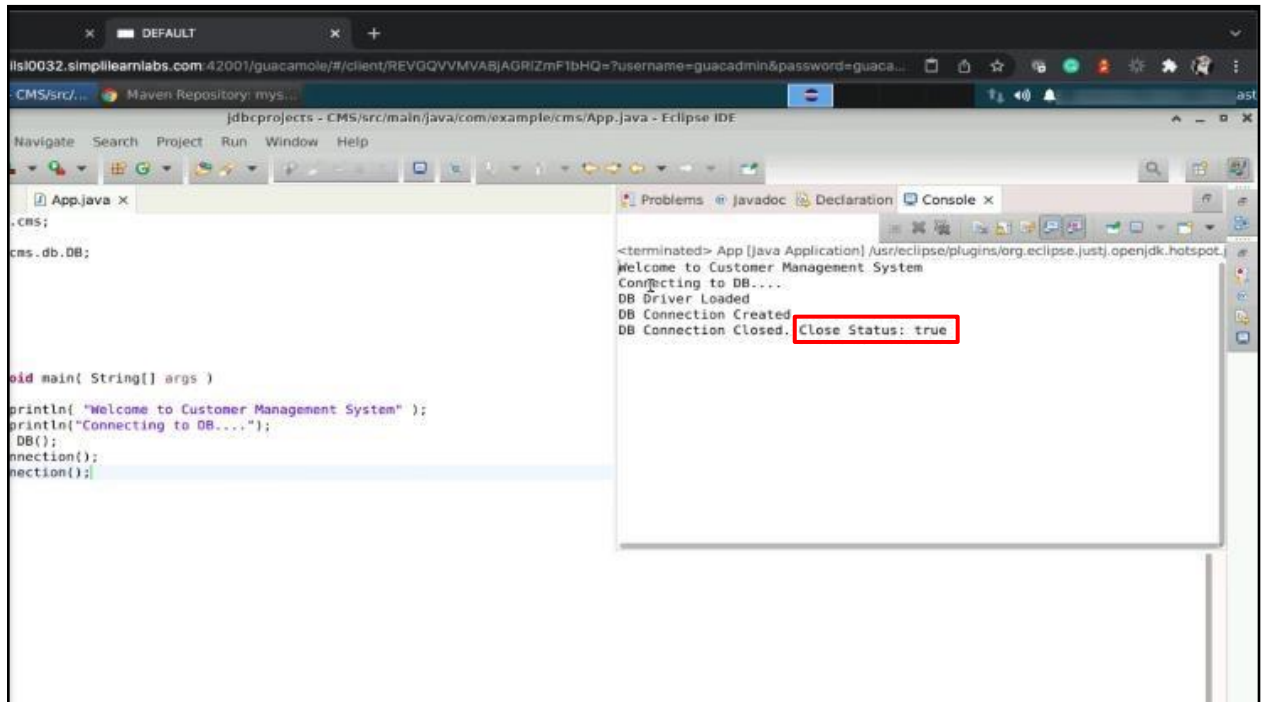


The screenshot shows the Eclipse IDE with the file `App.java` open. The code is as follows:

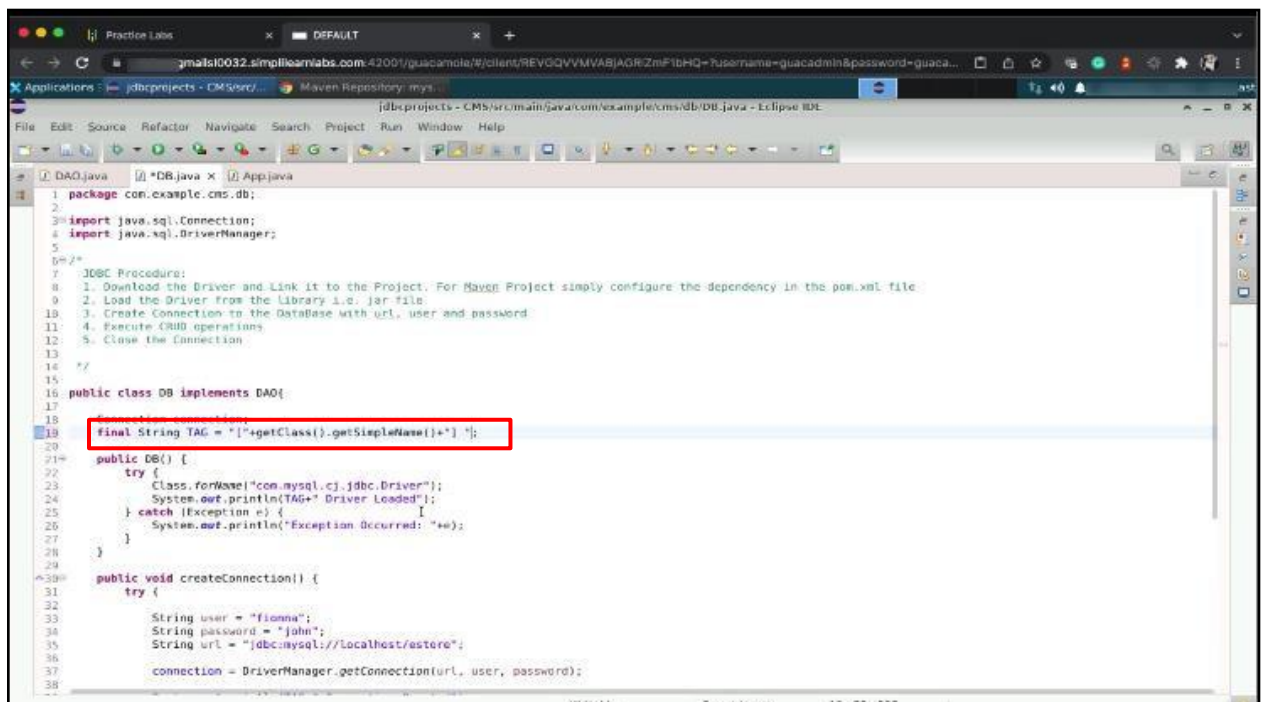
```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  *
8  */
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14         System.out.println("Connecting to DB....");
15         DB db = new DB();
16         db.createConnection();
17         db.closeConnection();
18     }
19 }
20
```

The line `db.closeConnection();` on line 17 is highlighted with a blue selection bar.

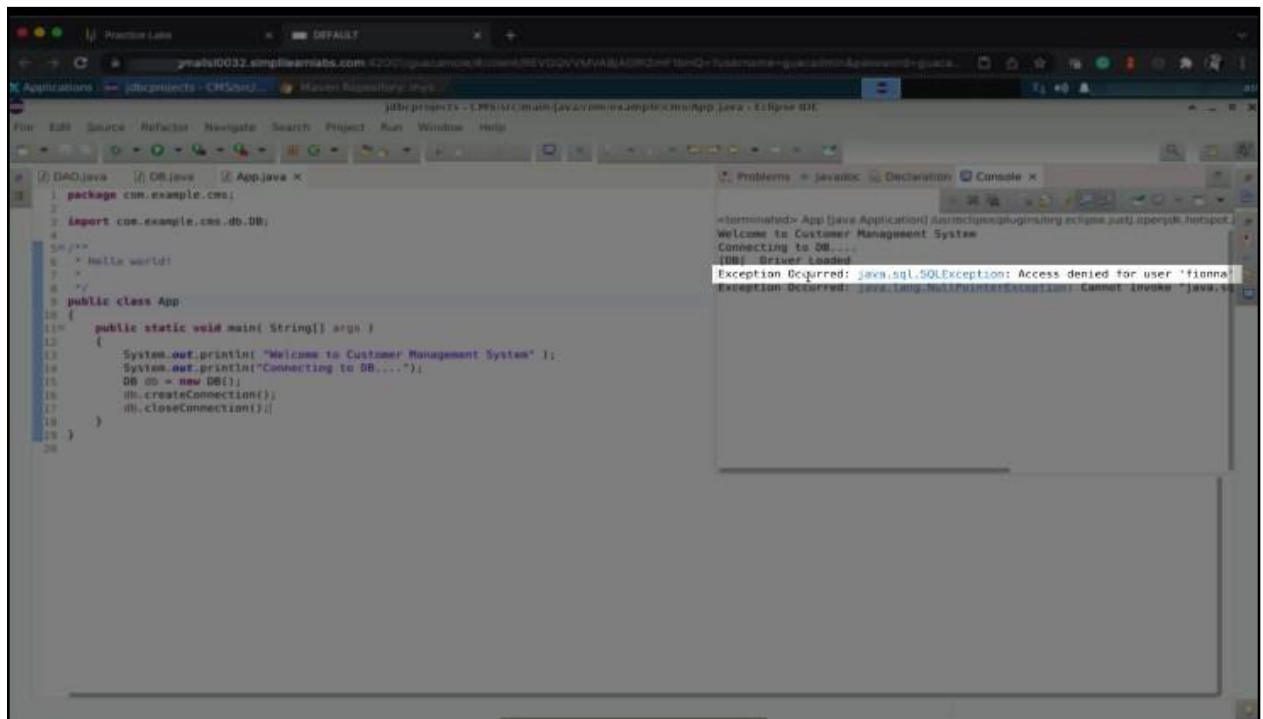
8.3 Run the code, and you should see the output as **Close Status: true** as shown in the screenshot below:



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



8.5 Re-run the code



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.