

Lesson 01 Demo 07

Creating Stored Procedures

Objective: To explore stored procedures and execute them using the JDBC API called **callableStatement** for efficient and reusable database operations. This approach enables the direct invocation of stored procedures defined in the database from a Java application

Tool required: Eclipse IDE

Prerequisites: None

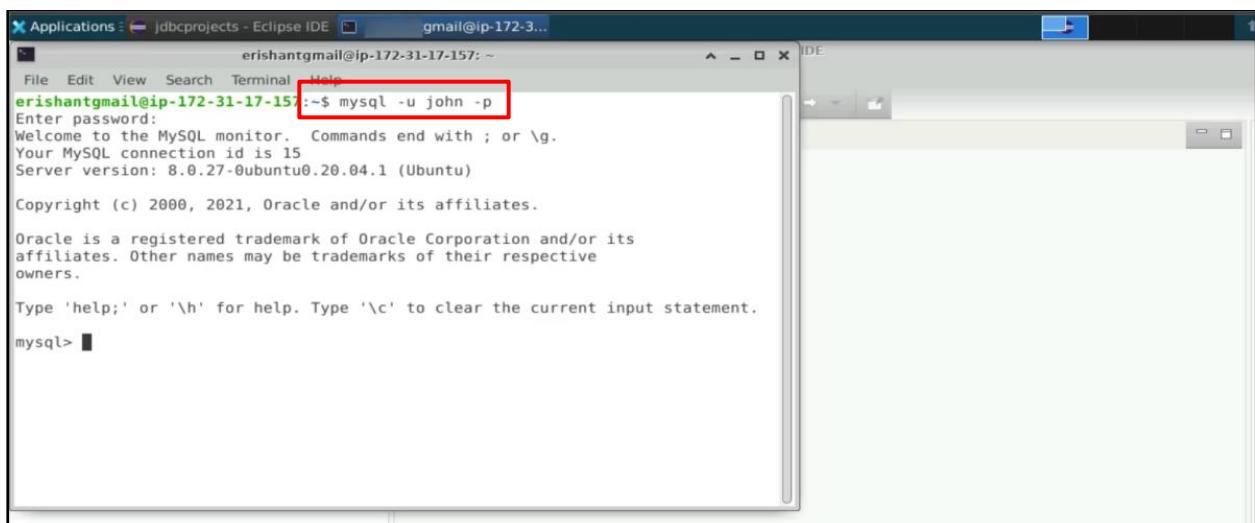
Steps to be followed:

1. Create a new table
2. Create a stored procedure
3. Create a method to execute a stored procedure
4. Create a stored procedure to read data
5. Create a method to call a stored procedure to read data

Step 1: Create a new table

1.1 Open the terminal, and log in to MySQL

`mysql -u john -p`



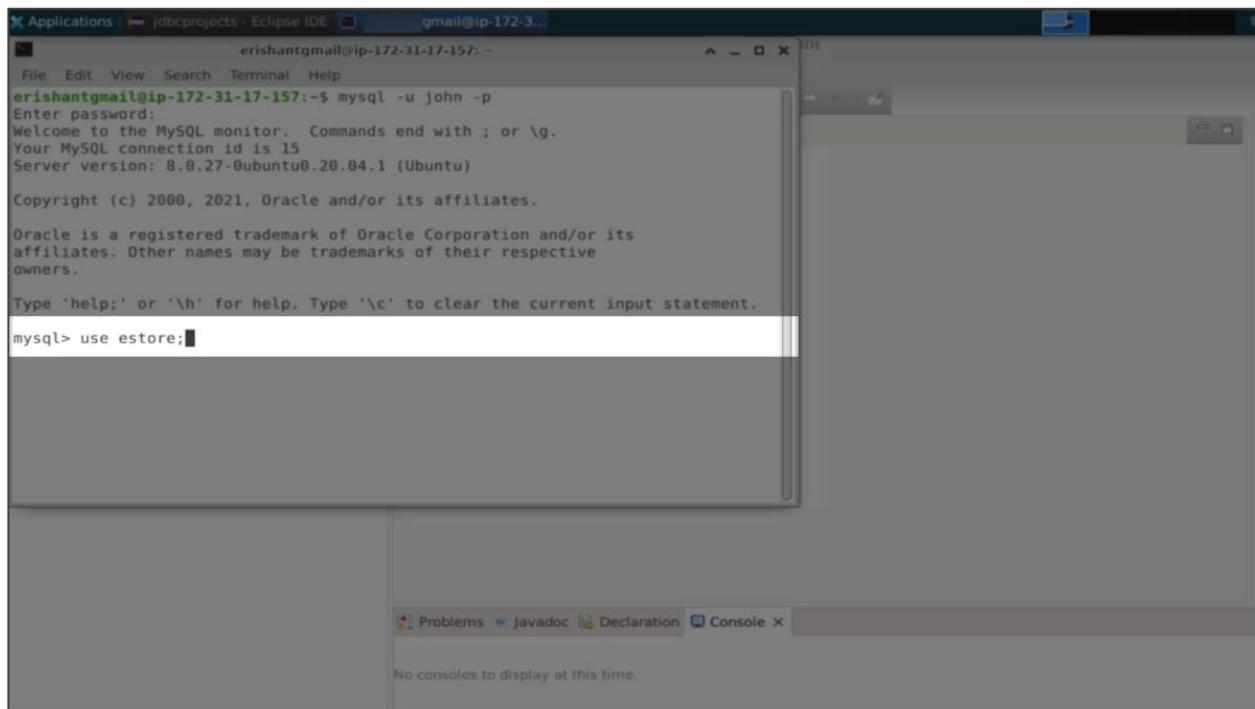
```
erishant@gmail:~/Desktop/jdbcprojects - Eclipse IDE [1] * Applications : jdbcprojects - Eclipse IDE [1] * gmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
erishant@gmail:~/Desktop/jdbcprojects - Eclipse IDE [1] * gmail@ip-172-31-17-157: ~$ mysql -u john -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
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> 
```

1.2 Use the command **use estore;** to change the database

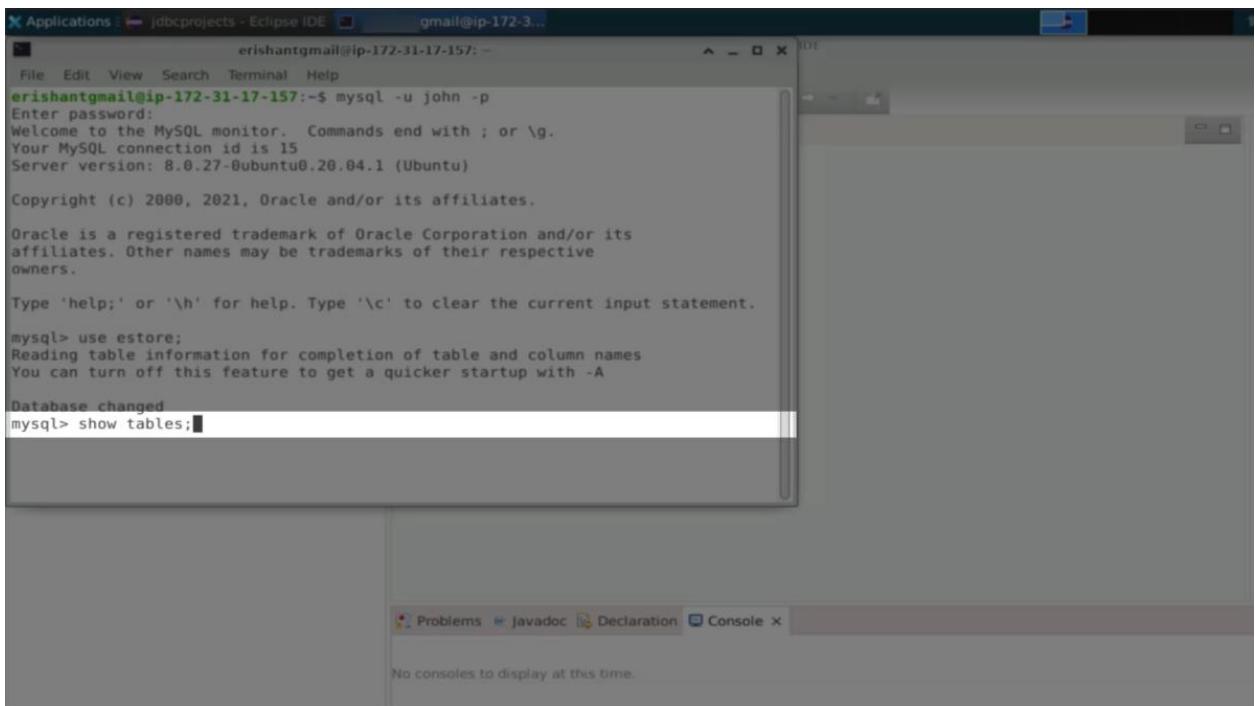


A screenshot of the Eclipse IDE interface. The title bar says "Applications - jdbcprojects - Eclipse IDE" and the tab bar shows "gmail@ip-172-3...". The main window is a terminal window titled "erishant@gmail@ip-172-31-17-157:~\$". It displays the MySQL monitor with the following text:

```
File Edit View Search Terminal Help  
erishant@gmail@ip-172-31-17-157:~$ mysql -u john -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 15  
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> use estore;
```

The bottom right corner of the terminal window shows "No consoles to display at this time."

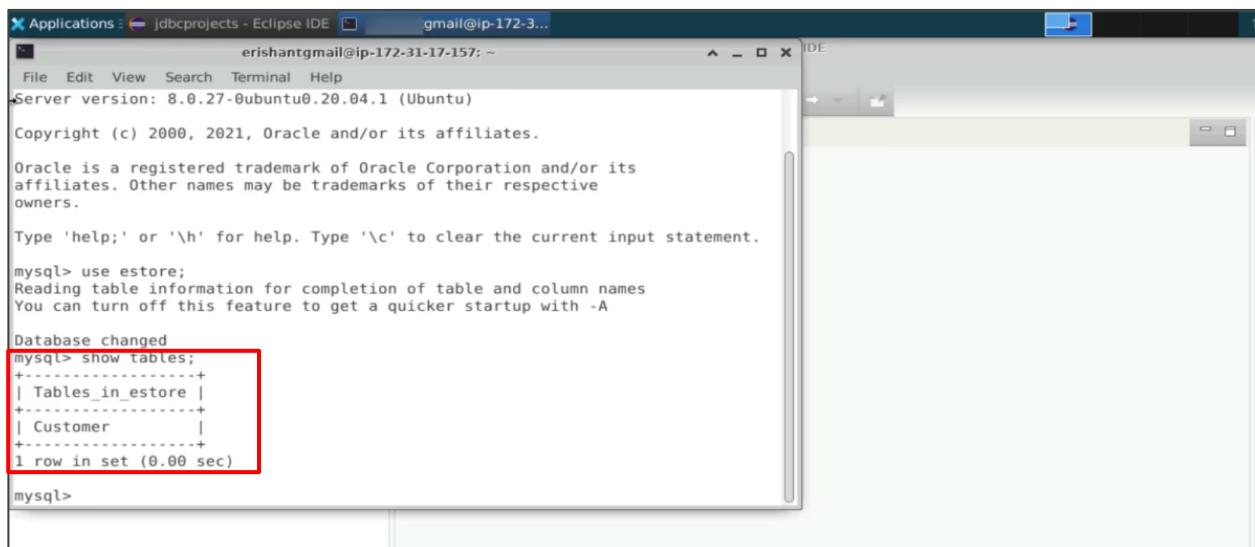
1.3 Use the **show tables;** command to check all tables available in the **estore** database



A screenshot of the Eclipse IDE interface, similar to the previous one. The title bar says "Applications - jdbcprojects - Eclipse IDE" and the tab bar shows "gmail@ip-172-3...". The main window is a terminal window titled "erishant@gmail@ip-172-31-17-157:~\$". It displays the MySQL monitor with the following text:

```
File Edit View Search Terminal Help  
erishant@gmail@ip-172-31-17-157:~$ mysql -u john -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 15  
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> use estore;  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed  
mysql> show tables;
```

The bottom right corner of the terminal window shows "No consoles to display at this time."



```
erishant@gmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
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.

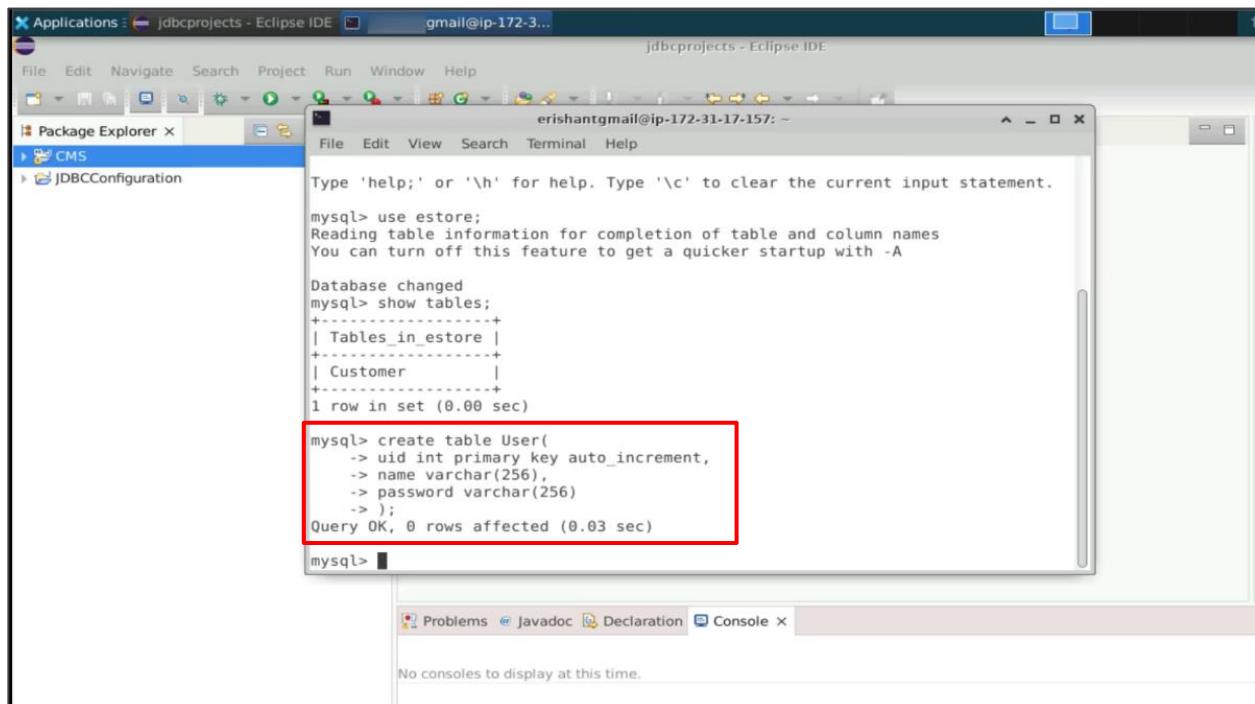
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
+-----+
1 row in set (0.00 sec)

mysql>
```

1.4 Use the `create table User();` command to create a new user table with attributes



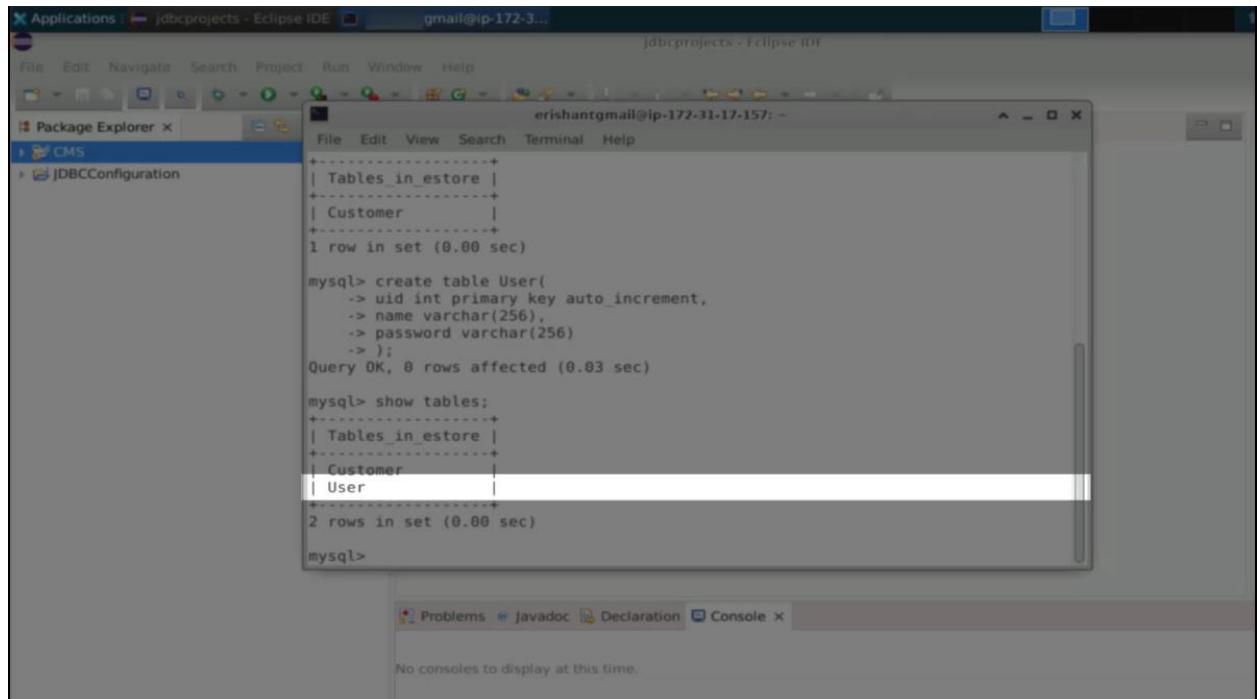
```
File Edit View Search Terminal Help
erishant@gmail@ip-172-31-17-157: ~
mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
+-----+
1 row in set (0.00 sec)

mysql> create table User(
-> uid int primary key auto_increment,
-> name varchar(256),
-> password varchar(256)
-> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

1.5 Write the command **show tables;** . The User table is created successfully and listed in the tables list.



The screenshot shows the Eclipse IDE interface with a MySQL terminal window. The terminal window displays the following MySQL session:

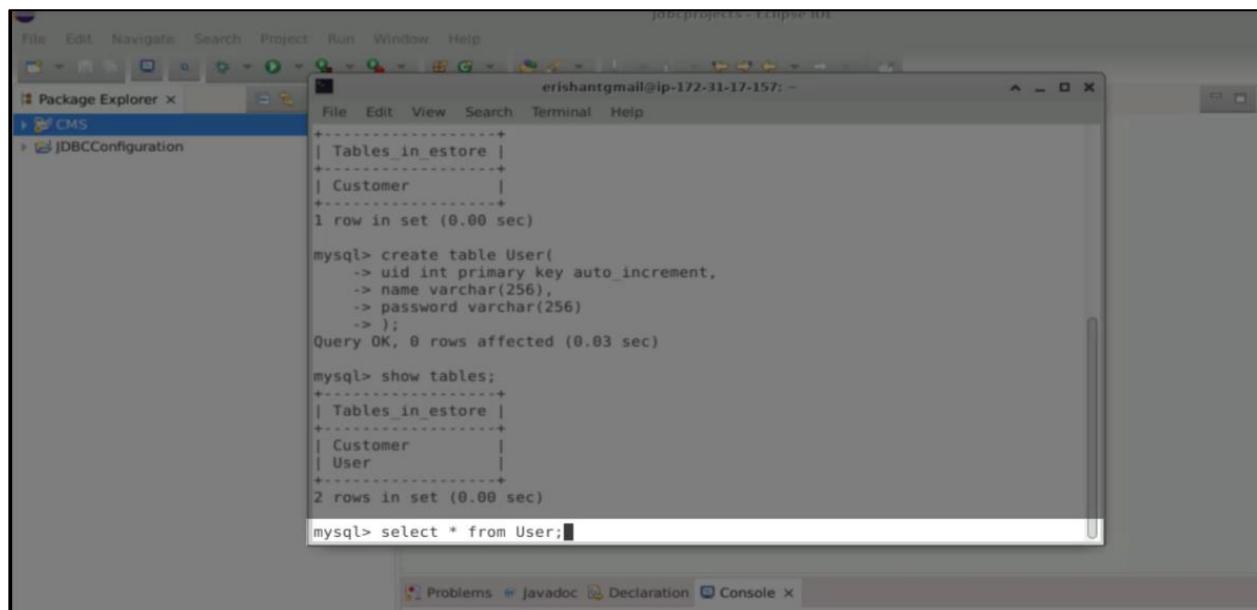
```
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer      |
| User          |
+-----+
2 rows in set (0.00 sec)

mysql> create table User(
    -> uid int primary key auto_increment,
    -> name varchar(256),
    -> password varchar(256)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer      |
| User          |
+-----+
2 rows in set (0.00 sec)

mysql>
```

1.6 Use the **select * from User** command to select data from the User table. You can see that the table is empty for now.



The screenshot shows the Eclipse IDE interface with a MySQL terminal window. The terminal window displays the following MySQL session, including a query:

```
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer      |
| User          |
+-----+
2 rows in set (0.00 sec)

mysql> create table User(
    -> uid int primary key auto_increment,
    -> name varchar(256),
    -> password varchar(256)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer      |
| User          |
+-----+
2 rows in set (0.00 sec)

mysql> select * from User;
```

The screenshot shows the Eclipse IDE interface with a MySQL terminal window open. The terminal window title is "erishant@gmail@ip-172-31-17-157: ~". The MySQL command-line interface is running, displaying the creation of a "User" table, a "show tables" command output, and a "select * from User" command result.

```
File Edit View Search Terminal Help
| Customer      |
+-----+
1 row in set (0.00 sec)

mysql> create table User(
    -> uid int primary key auto_increment,
    -> name varchar(256),
    -> password varchar(256)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer      |
| User          |
+-----+
2 rows in set (0.00 sec)

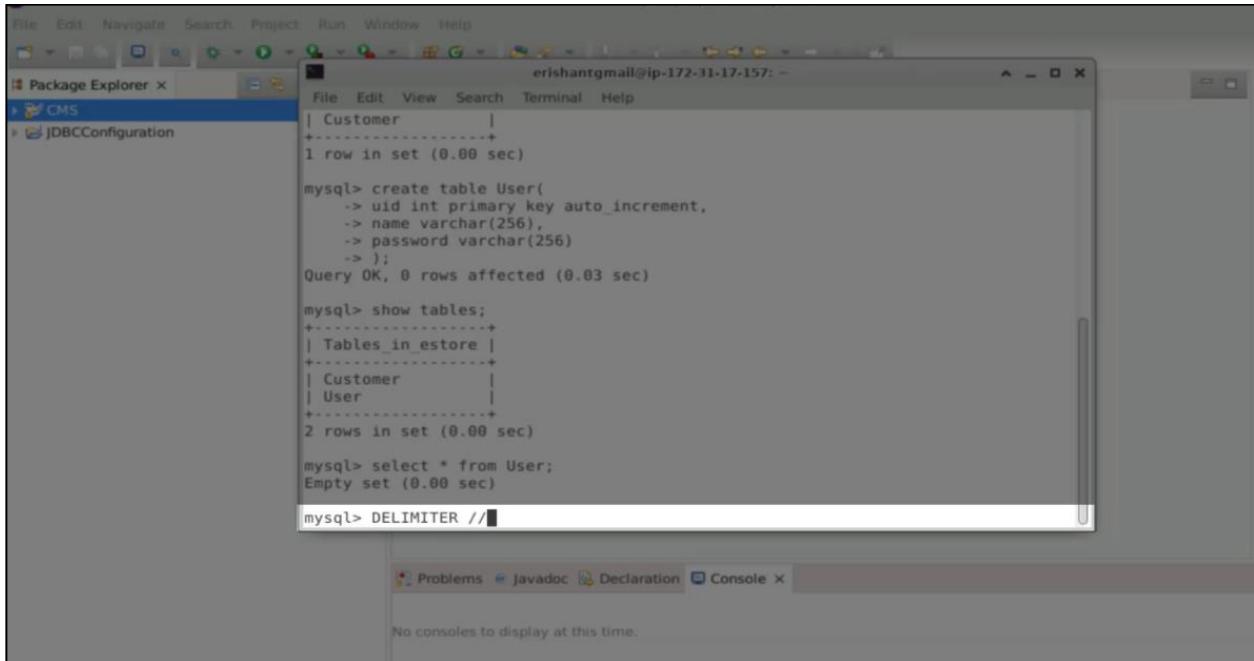
mysql> select * from User;
Empty set (0.00 sec)

mysql>
```

The Eclipse interface includes a "Package Explorer" view showing projects like "CMS" and "JDBCConfiguration". Below the terminal window, there are tabs for "Problems", "Javadoc", "Declaration", and "Console". A message in the "Console" tab states "No consoles to display at this time."

Step 2: Create a stored procedure

2.1 To enter records in this empty set, we must create a Stored Procedure. Change the **Delimiter** to // so it does not get confused with the ; used while creating the procedure.



The screenshot shows the Eclipse IDE interface with the MySQL terminal window open. The terminal window displays the following MySQL session:

```
erishant@gmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
| Customer |
+-----+
1 row in set (0.00 sec)

mysql> create table User(
    -> uid int primary key auto_increment,
    -> name varchar(256),
    -> password varchar(256)
    -> );
Query OK, 0 rows affected (0.03 sec)

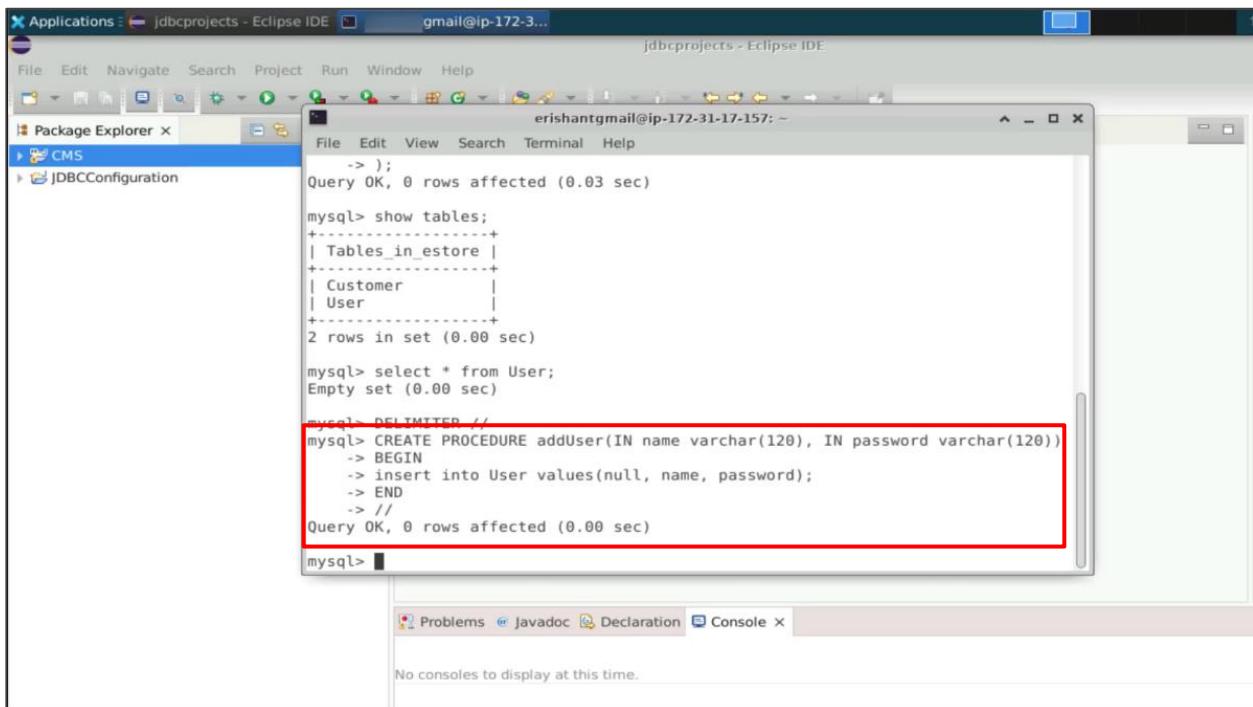
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer |
| User |
+-----+
2 rows in set (0.00 sec)

mysql> select * from User;
Empty set (0.00 sec)

mysql> DELIMITER //
```

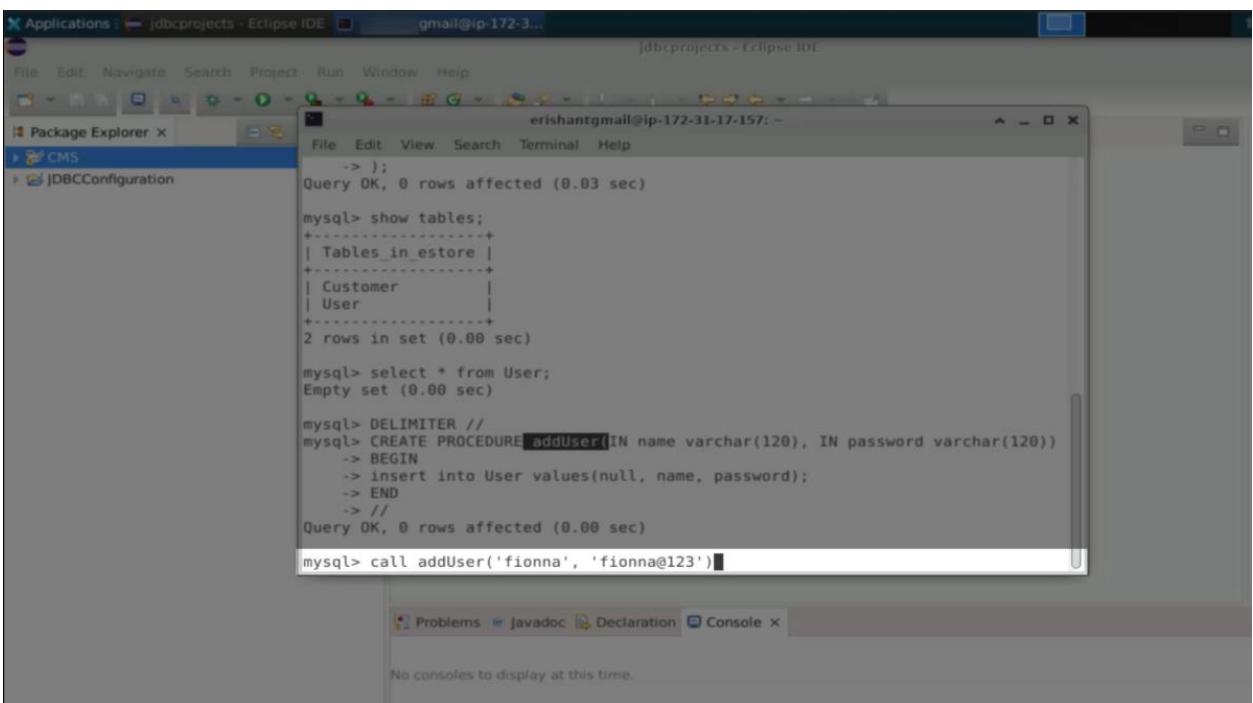
The MySQL prompt is currently at `mysql> DELIMITER //`. The bottom status bar indicates "No consoles to display at this time."

2.2 Run the **CREATE PROCEDURE addUser()** command to create a procedure with attributes



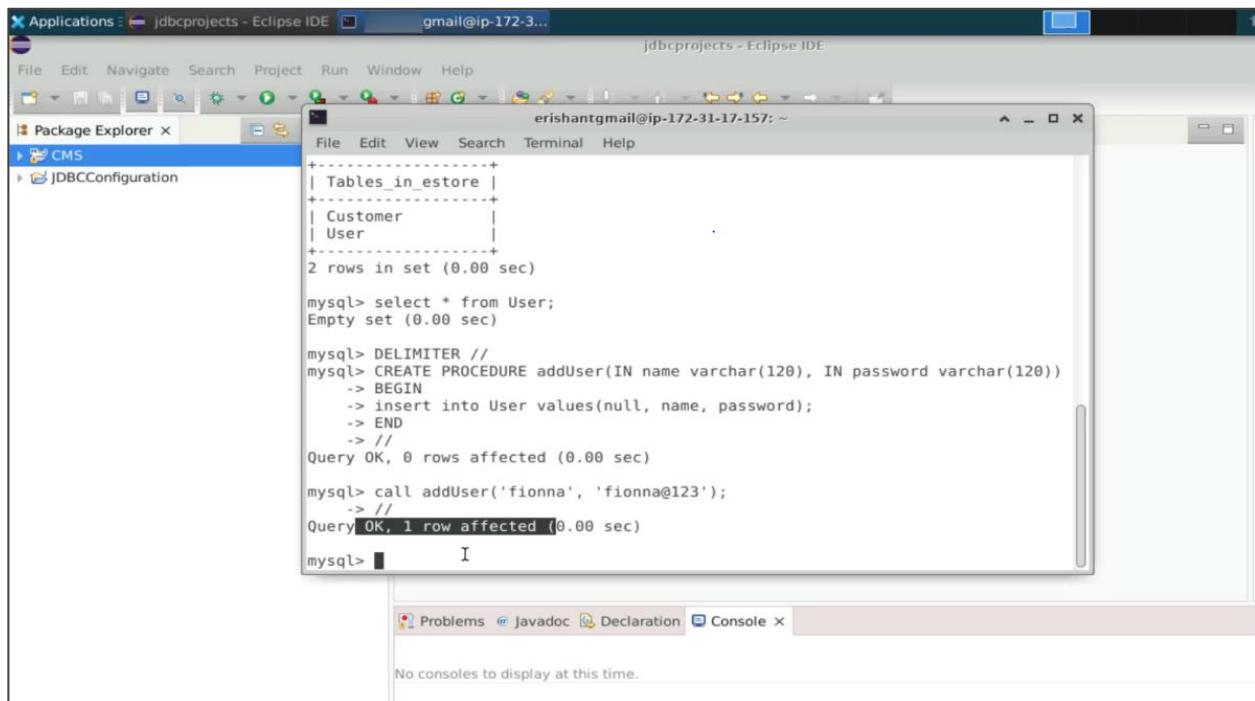
```
mysql> DELIMITER //
mysql> CREATE PROCEDURE addUser(IN name varchar(120), IN password varchar(120))
-> BEGIN
-> insert into User values(null, name, password);
-> END
-> //
Query OK, 0 rows affected (0.00 sec)
```

2.3 Execute the procedure we created by writing the **call addUser();** command with some data



```
mysql> call addUser('fionna', 'fionna@123')
```

The command is executed, and you can see **1 row affected** as output.

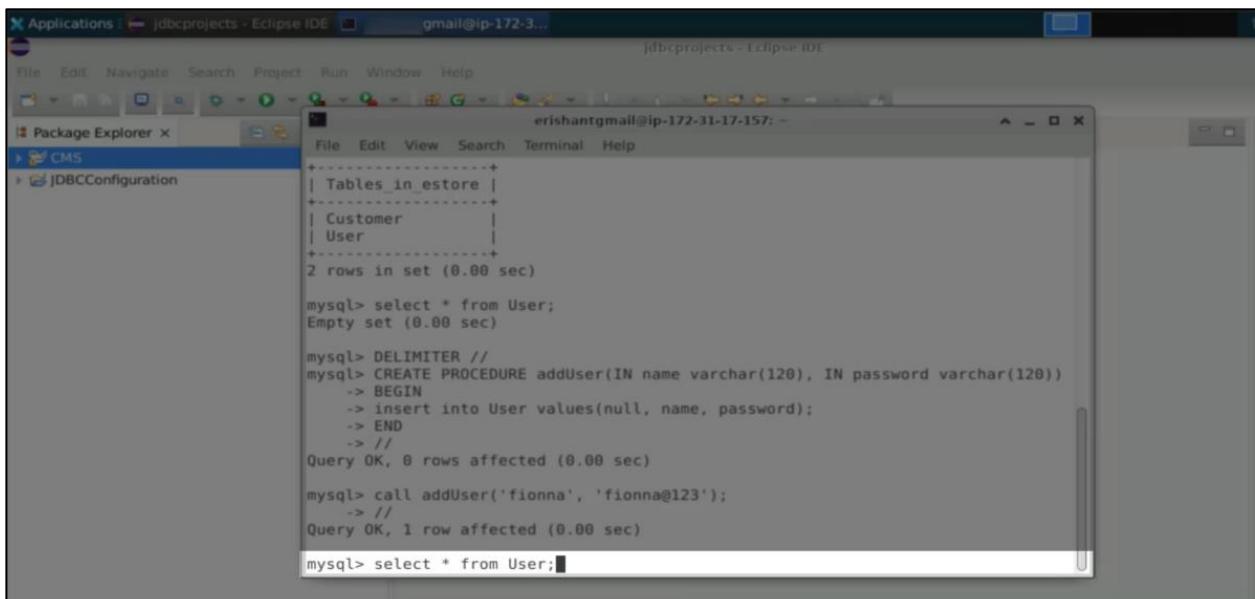


A screenshot of the Eclipse IDE interface. The title bar says "Applications : jdbcprojects - Eclipse IDE" and "gmail@ip-172-3...". The main window shows a MySQL terminal window titled "erishant@gmail@ip-172-31-17-157: ~". The terminal output is as follows:

```
+-----+  
| Tables_in_estore |  
+-----+  
| Customer |  
| User |  
+-----+  
2 rows in set (0.00 sec)  
  
mysql> select * from User;  
Empty set (0.00 sec)  
  
mysql> DELIMITER //  
mysql> CREATE PROCEDURE addUser(IN name varchar(120), IN password varchar(120))  
    -> BEGIN  
        -> insert into User values(null, name, password);  
        -> END  
    -> //  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> call addUser('fionna', 'fionna@123');  
-> //  
Query OK, 1 row affected (0.00 sec)  
  
mysql> I
```

The bottom status bar says "No consoles to display at this time."

2.4 Run the **select * from User** command to check if the data inserted in the table is successful or not



A screenshot of the Eclipse IDE interface, similar to the previous one. The title bar says "Applications : jdbcprojects - Eclipse IDE" and "gmail@ip-172-3...". The main window shows a MySQL terminal window titled "erishant@gmail@ip-172-31-17-157: ~". The terminal output is as follows:

```
+-----+  
| Tables_in_estore |  
+-----+  
| Customer |  
| User |  
+-----+  
2 rows in set (0.00 sec)  
  
mysql> select * from User;  
Empty set (0.00 sec)  
  
mysql> DELIMITER //  
mysql> CREATE PROCEDURE addUser(IN name varchar(120), IN password varchar(120))  
    -> BEGIN  
        -> insert into User values(null, name, password);  
        -> END  
    -> //  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> call addUser('fionna', 'fionna@123');  
-> //  
Query OK, 1 row affected (0.00 sec)  
  
mysql> select * from User;
```

The record has been inserted successfully, as seen in the table.

```

mysql> DELIMITER //
mysql> CREATE PROCEDURE addUser(IN name varchar(120), IN password varchar(120))
-> BEGIN
->     insert into User values(null, name, password);
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> call addUser('fionna', 'fionna@123');
-> //
Query OK, 1 row affected (0.00 sec)

mysql> select * from User;
-> //
+----+-----+-----+
| uid | name  | password |
+----+-----+-----+
| 1   | fionna | fionna@123 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Step 3: Create a method to execute the stored procedure

3.1 Open Eclipse IDE and go to the DB.java file

```

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

/*
* JDBC Procedure:
* 1. Download the Driver and Link it to the Project. For Maven Project simply configure the d
* 2. Load the Driver from the library i.e. jar file
* 3. Create Connection to the DataBase with url, user and password
* 4. Execute CRUD operations
* 5. Close the Connection
*/
public class DB implements DAO{
    Connection connection;
    Statement statement;
    PreparedStatement preparedStatement;

    final String TAG = "["+getClass().getSimpleName()+"] ";
    public DB() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println(TAG+"Driver Loaded");
        } catch (Exception e) {
            System.out.println("Exception Occurred: "+e);
        }
    }
}

```

3.2 Execute the stored procedure, create a method `executeProcedure()` and put it inside the `try-catch` block to handle exceptions

```

179     //ResultSet set = statement.executeQuery(sql);
180
181     ResultSet set = preparedStatement.executeQuery();
182
183     while(set.next()) {
184
185         Customer customer = new Customer();
186
187         customer.setCustomerId(set.getInt("cid"));
188         customer.setName(set.getString(2));
189         customer.setPhone(set.getString(3));
190         customer.setEmail(set.getString(4));
191         customer.setBirthDate(set.getString(5));
192         customer.setAge(set.getInt(6));
193         customer.setInDate(set.getString(7));
194         customer.setOutDate(set.getString(8));
195         customer.setTemperature(set.getFloat(9));
196
197         customers.add(customer);
198     }
199
200     } catch (Exception e) {
201         System.out.println("Exception Occurred: "+e);
202     }
203
204     return customers;
205 }
206
207 public void executeProcedure() {
208     try {
209
210         } catch (Exception e) {
211             System.out.println("Exception Occurred: "+e);
212         }
213     }
214 }
215
216

```

3.3 Call the API known as `callableStatement` from the SQL package for the stored procedure. Use the `callableStatement()` method

```

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

```

3.4 Initialize the reference variable, create an SQL query, and give substitution to the callableStatement

```

189     customer.setCid(set.getInt("cid"));
190     customer.setName(set.getString(2));
191     customer.setPhone(set.getString(3));
192     customer.setEmail(set.getString(4));
193     customer.setBirthDate(set.getString(5));
194     customer.setAge(set.getInt(6));
195     customer.setInDateTime(set.getString(7));
196     customer.setOutDateTime(set.getString(8));
197     customer.setTemperature(set.getFloat(9));
198
199     customers.add(customer);
200   }
201
202   } catch (Exception e) {
203     System.out.println("Exception Occurred: "+e);
204   }
205
206   return customers;
207 }
208
209 public void executeProcedure() {
210   try {
211
212     String sql = "{ call addUser(?, ?) }";
213     callableStatement = connection.prepareCall(sql);
214     callableStatement.setString(1, "leo");
215     callableStatement.setString(2, "leo@12345");
216
217     callableStatement.execute();
218
219     System.out.println("Stored Procedure is Executed :)");
220   } catch (Exception e) {
221     System.out.println("Exception Occurred: "+e);
222   }
223 }
224
225 }
226

```

3.5 Go to the App.java file and comment on all the existing code available in the main method

```

3* import java.util.ArrayList;
4
5 public class App
6 {
7   public static void main(String[] args)
8   {
9     /*System.out.println( "Welcome to Customer Management System" );
10
11     Customer customer = new Customer();
12
13     customer.setCid(4);
14     customer.setName("George G");
15     customer.setPhone("+91 99889 11111");
16     customer.setEmail("george.g@example.com");
17     customer.setBirthDate("1986-04-04");
18     customer.setAge(35);
19     customer.setInDateTime("2022-01-12 09:00:00");
20     customer.setOutDateTime("2022-01-12 10:30:00");
21     customer.setTemperature(98.8f);
22
23     System.out.println("Connecting to DB....");
24     DB db = new DB();
25     db.createConnection();
26
27     //db.createCustomer(customer);
28     //db.updateCustomer(customer);
29
30     //db.deleteCustomer(3);
31
32     //System.out.println();
33
34     ArrayList<Customer> customers = db.getAllCustomers();
35     customers.forEach(cRef -> System.out.println(cRef));
36
37     db.closeConnection();*/
38
39
40
41
42

```

3.6 Create a scanner to process the data from the command prompt and create a db connection to execute the procedure

```

19     customer.setPhone("+91 99009 1111");
20     customer.setEmail("george.g@example.com");
21     customer.setBirthDate("1986-04-04");
22     customer.setAge(35);
23     customer.setInDateTime("2022-01-12 09:00:00");
24     customer.setOutDateTime("2022-01-12 10:30:00");
25     customer.setTemperature(98.6f);
26
27     System.out.println("Connecting to DB....");
28     DB db = new DB();
29     db.createConnection();
30
31     //db.createCustomer(customer);
32     //db.updateCustomer(customer);
33
34     //db.deleteCustomer(3);
35
36     //System.out.println();
37
38     ArrayList<Customer> customers = db.getAllCustomers();
39     customers.forEach(cRef -> System.out.println(cRef));
40
41     db.closeConnection();*/
42
43     Scanner scanner = new Scanner(System.in);
44     System.out.println("Enter Name: ");
45     String name = scanner.nextLine();
46
47     System.out.println("Enter Password: ");
48     String password = scanner.nextLine();
49
50     scanner.close();
51
52     DB db = new DB();
53     db.createConnection();
54     db.executeProcedure(name, password);
55     db.closeConnection();

```

3.7 Go to the DB.java file and modify the code to get the name and password as input from the user

```

189     customer.setCid(set.getInt("cid"));
190     customer.setName(set.getString(2));
191     customer.setPhone(set.getString(3));
192     customer.setEmail(set.getString(4));
193     customer.setBirthDate(set.getString(5));
194     customer.setAge(set.getInt(6));
195     customer.setInDateTime(set.getString(7));
196     customer.setOutDateTime(set.getString(8));
197     customer.setTemperature(set.getFloat(9));
198
199     customers.add(customer);
200 }
201
202 } catch (Exception e) {
203     System.out.println("Exception Occurred: "+e);
204 }
205
206 return customers;
207
208
209+ public void executeProcedure(String name, String password) {
210     try {
211         I
212         String sql = "{ call addUser(?, ?) }";
213         callableStatement = connection.prepareCall(sql);
214         callableStatement.setString(1, "leo");
215         callableStatement.setString(2, "leo@12345");
216
217         callableStatement.execute();
218
219         System.out.println("Stored Procedure is Executed :)");
220     } catch (Exception e) {
221         System.out.println("Exception Occurred: "+e);
222     }

```

3.8 Modify the code to make the name and password more dynamic

```

189         customer.setCid(set.getInt("cid"));
190         customer.setName(set.getString(2));
191         customer.setPhone(set.getString(3));
192         customer.setEmail(set.getString(4));
193         customer.setBirthDate(set.getString(5));
194         customer.setAge(set.getInt(6));
195         customer.setInDateTime(set.getString(7));
196         customer.setOutDateTime(set.getString(8));
197         customer.setTemperature(set.getFloat(9));
198
199         customers.add(customer);
200     }
201
202     } catch (Exception e) {
203         System.out.println("Exception Occurred: "+e);
204     }
205
206     return customers;
207 }
208
209 public void executeProcedure(String name, String password) {
210     try {
211
212         String sql = "{ call addUser(?, ?) }";
213         callableStatement = connection.prepareCall(sql);
214         callableStatement.setString(1, name);
215         callableStatement.setString(2, password);
216
217         callableStatement.execute();
218
219         System.out.println("Stored Procedure is Executed :");
220     } catch (Exception e) {
221         System.out.println("Exception Occurred: "+e);
222     }
223 }
224
225 }
226

```

3.9 Go to the App.java file and run the program

```

22         customer.setAge(35);
23         customer.setInDateTime("2022-01-12 09:00:00");
24         customer.setOutDateTime("2022-01-12 10:30:00");
25         customer.setTemperature(98.8f);
26
27         System.out.println("Connecting to DB....");
28         DB db = new DB();
29         db.createConnection();
30
31         //db.createCustomer(customer);
32         //db.updateCustomer(customer);
33
34         //db.deleteCustomer(3);
35
36         //System.out.println();
37
38         ArrayList<Customer> customers = db.getAllCustomers();
39         customers.forEach(cRef -> System.out.println(cRef));
40
41         db.closeConnection();
42
43         Scanner scanner = new Scanner(System.in);
44         System.out.println("Enter Name: ");
45         String name = scanner.nextLine();
46
47         System.out.println("Enter Password: ");
48         String password = scanner.nextLine();
49
50         scanner.close();
51
52         DB db = new DB();
53         db.createConnection();
54         db.executeProcedure(name, password);
55         db.closeConnection();
56
57     }
58 }
59

```

3.10 Fill in the information and press the **Enter** key in the console. In the output, you can see
Stored Procedure is Executed :)

```

acamole#/client/REVGQVVMVABjAGRIZmF1bHQ=?username=guacadmin&password=
2-3...
objects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console X
"");
0");
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.jst.j.openjdk.hotspot.jvm.v1_6_0.v20110915-1100/jdk1.6.0_31/bin/java -Dfile.encoding=UTF-8 -Xms128m -Xmx512m -XX:MaxPermSize=256m -jar /home/erishant/Downloads/MySQL-Workbench-5.2.4-Linux-x64/MySQLWorkbench.jar
Enter Name:
Leo
Enter Password:
leo@12345
[DB] Driver Loaded
[DB] Connection Created
Stored Procedue is Executed :)
[DB] Connection Closed. Close Status: true

```

3.11 Go back to the terminal and run the select command again. You can see the user record inserted in the above step in the data table.

```

File Edit View Terminal Help
erishant@gmail@ip-172-31-17-157: ~
mysql> call addUser('fionna', 'fionna@123');
-> //
Query OK, 1 row affected (0.00 sec)

mysql> select * from User;
-> //
+-----+-----+-----+
| uid | name | password |
+-----+-----+-----+
| 1 | fionna | fionna@123 |
+-----+-----+-----+
1 row in set (0.00 sec) .

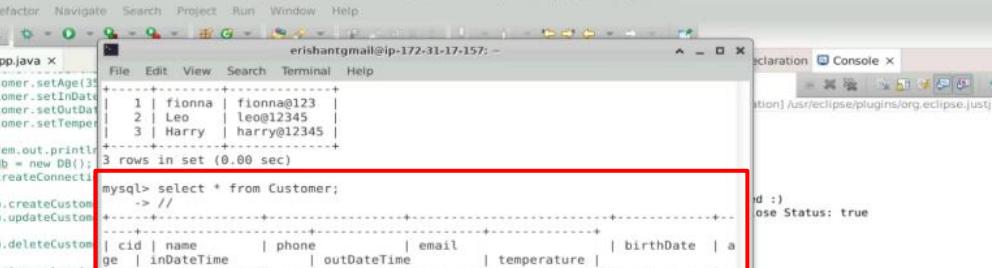
mysql> select * from User;
-> //
+-----+-----+-----+
| uid | name | password |
+-----+-----+-----+
| 1 | fionna | fionna@123 |
| 2 | Leo | leo@12345 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> 

```

Step 4: Create a stored procedure to read data

4.1 Run **select * from Customer** to get all customer data



The screenshot shows the Eclipse IDE interface with multiple windows open. The central window displays a MySQL terminal session with the following output:

```
erishant@gmail:ip-172-31-17-157: ~
File Edit View Search Terminal Help
+-----+
| 1 | fionna | fionna@123 |
| 2 | Leo     | leo@12345 |
| 3 | Harry   | harry@12345 |
+-----+
3 rows in set (0.00 sec)

mysql> select * from Customer;
-> //
+-----+
| cid | name      | phone        | email       | birthDate | a
ge |
+-----+
| 2  | John Watson | +91 98761 22222 | john.watson@example.com | 1990-08-08 |
| 32 | 2022-01-08 10:39:52 | 2022-01-08 11:45:22 | 98.5 |
| 4  | George G    | +91 99009 11111 | george.g@example.com | 1986-04-04 |
| 35 | 2022-01-12 09:00:00 | 2022-01-12 10:30:00 | 98.8 |
+-----+
2 rows in set (0.00 sec)

mysql>
```

The MySQL session shows two sets of results. The first set contains three rows of customer data with columns: id, name, phone, email, birthDate, and age. The second set shows two rows of customer data with columns: cid, name, phone, email, birthDate, and age.

4.2 Create a **stored procedure** as created earlier

The screenshot shows the Eclipse IDE interface with multiple windows open. The main window displays Java code for a Customer class and a MySQL procedure. The code includes database connection logic and a MySQL command to create a stored procedure named 'getCustomerName'. The MySQL command is highlighted with a red box.

```
customer.setAge(35);
customer.setInDate("2022-01-08 10:39:52");
customer.setOutDate("2022-01-12 09:00:00");
customer.setTemp(98.5);
System.out.println("Data inserted successfully");

Db db = new DB();
db.createConnection();
//db.createCustomer();
//db.updateCustomer();
//db.deleteCustomer();
//System.out.println("Data deleted successfully");

ArrayList<Customer> customers;
customers.forEach(cust->{
    db.closeConnection();
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter customer ID: ");
    String name = scanner.nextLine();
    System.out.print("Enter customer name: ");
    String password = scanner.nextLine();
    scanner.close();
    db = new DB();
    db.createConnection();
    db.executeProcedure(name, password);
    db.closeConnection();
});
```

```
mysql> create procedure getCustomerName(IN cid int, OUT customer_name varchar(256))
    -> BEGIN
    ->     select name into customer_name From Customer where cid = cid;
    -> END
    -> ;
    -> //
```

4.3 Execute the created stored procedure, run the command `call getCustomerName(2);` where **2** is the customer ID we are calling

You can see the result with the name of the customer with **cid 2**.

Step 5: Create a method to call a stored procedure to read data

5.1 Open the **DB.java** file and comment out the **try block** in Eclipse. Create the SQL statement by passing a single input and using the **prepareCall ()** method

```

189     customer.setCID(set.getInt("cid"));
190     customer.setName(set.getString(2));
191     customer.setPhone(set.getString(3));
192     customer.setEmail(set.getString(4));
193     customer.setBirthdate(set.getString(5));
194     customer.setAge(set.getInt(6));
195     customer.setInDateTime(set.getString(7));
196     customer.setOutDateTime(set.getString(8));
197     customer.setTemperature(set.getFloat(9));
198
199     customers.add(customer);
200 }
201
202 } catch (Exception e) {
203     System.out.println("Exception Occurred: "+e);
204 }
205
206 return customers;
207 }
208
209 public void executeProcedure(String name, String password) {
210     try {
211
212         /*String sql = "{ call addUser(?, ?) }";
213         callableStatement = connection.prepareCall(sql);
214         callableStatement.setString(1, name);
215         callableStatement.setString(2, password);
216
217         callableStatement.execute();*/
218
219         String sql = "{ call getCustomerName(?) }";
220         callableStatement.setInt(1, 0);
221
222         System.out.println("Stored Procedure is Executed :)");
223     } catch (Exception e) {
224         System.out.println("Exception Occurred: "+e);
225     }
226 }
227
228

```

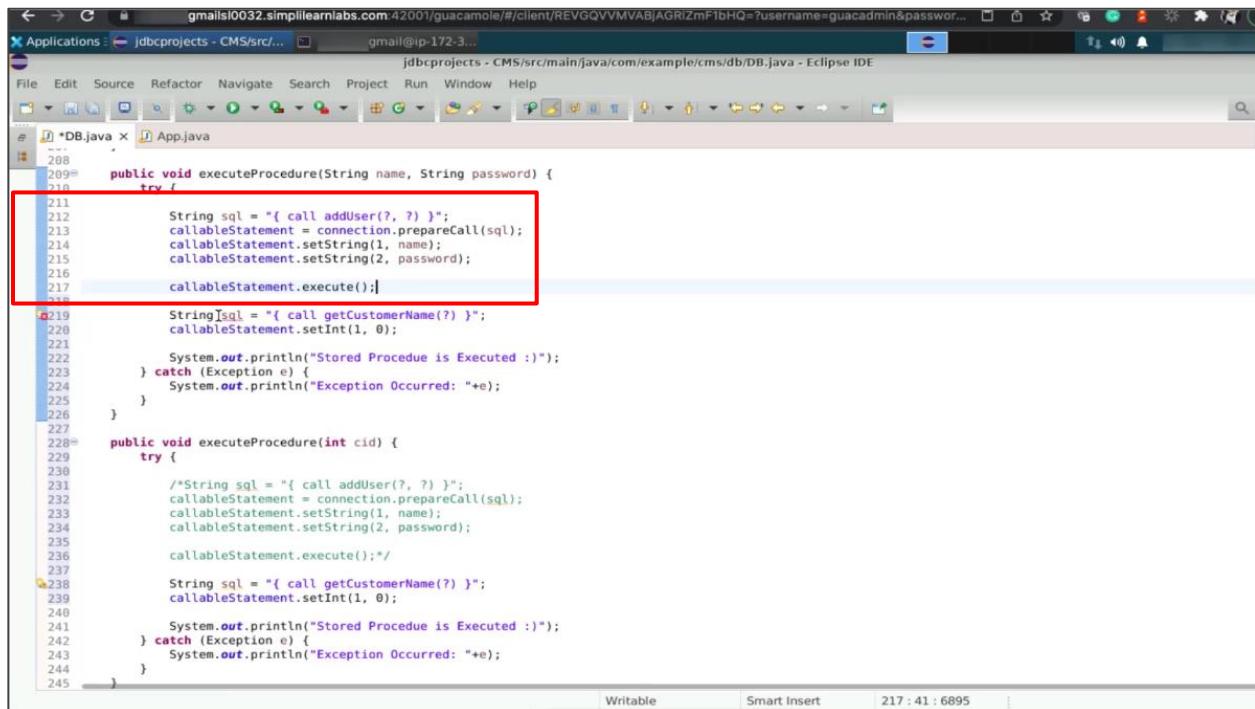
5.2 Create an overloaded version of the method by adding **cid** as an input

```

208
209 public void executeProcedure(String name, String password) {
210     try {
211
212         /*String sql = "{ call addUser(?, ?) }";
213         callableStatement = connection.prepareCall(sql);
214         callableStatement.setString(1, name);
215         callableStatement.setString(2, password);
216
217         callableStatement.execute();*/
218
219         String sql = "{ call getCustomerName(?) }";
220         callableStatement.setInt(1, 0);
221
222         System.out.println("Stored Procedure is Executed :)");
223     } catch (Exception e) {
224         System.out.println("Exception Occurred: "+e);
225     }
226 }
227
228 public void executeProcedure(int cid) {
229     try {
230
231         /*String sql = "{ call addUser(?, ?) }";
232         callableStatement = connection.prepareCall(sql);
233         callableStatement.setString(1, name);
234         callableStatement.setString(2, password);
235
236         callableStatement.execute();*/
237
238         String sql = "{ call getCustomerName(?) }";
239
240     } catch (Exception e) {
241         System.out.println("Exception Occurred: "+e);
242     }
243 }
244
245

```

5.3 Uncomment the **try block** of the previous procedure



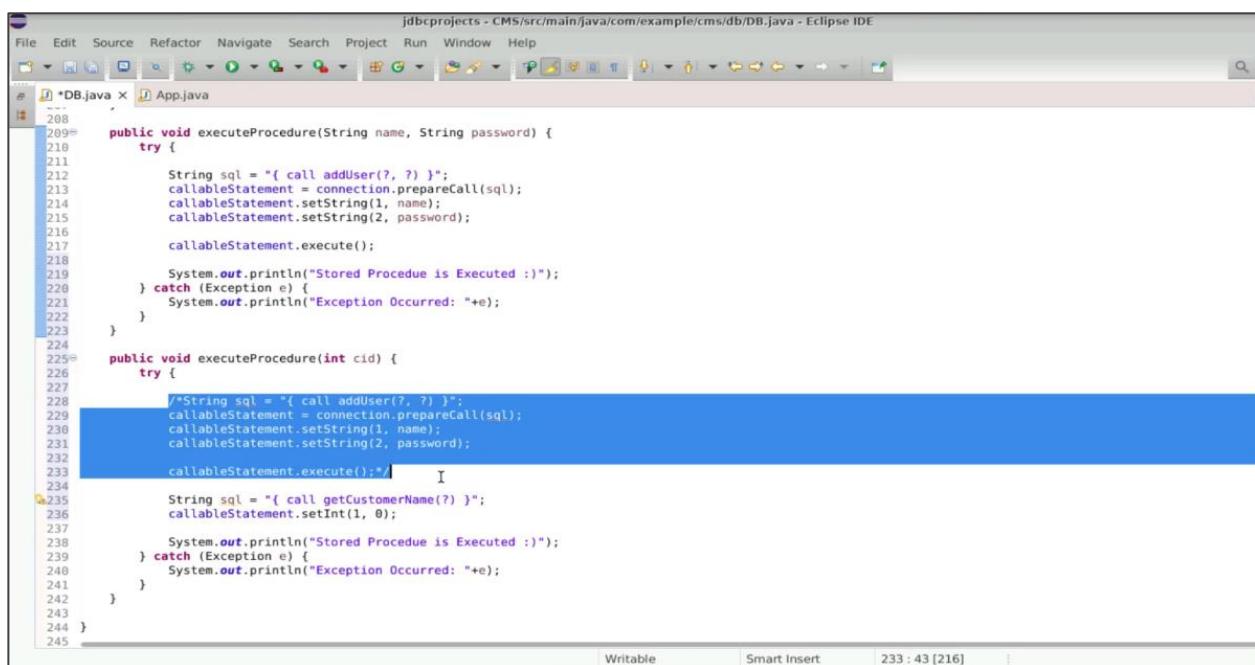
```

public void executeProcedure(String name, String password) {
    try {
        String sql = "{ call addUser(?, ?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setString(1, name);
        callableStatement.setString(2, password);
        callableStatement.execute();
    }
    catch (Exception e) {
        System.out.println("Stored Procedure is Executed :)");
    }
}

public void executeProcedure(int cid) {
    try {
        String sql = "{ call addUser(?, ?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setString(1, name);
        callableStatement.setString(2, password);
        callableStatement.execute();
    }
    catch (Exception e) {
        System.out.println("Stored Procedure is Executed :)");
    }
}

```

5.4 Delete the **String SQL** and **callableStatement** variables and remove the commented lines from the try block



```

public void executeProcedure(String name, String password) {
    try {
        String sql = "{ call addUser(?, ?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setString(1, name);
        callableStatement.setString(2, password);
        callableStatement.execute();
        System.out.println("Stored Procedure is Executed :)");
    }
    catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

public void executeProcedure(int cid) {
    try {
        /*String sql = "{ call addUser(?, ?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setString(1, name);
        callableStatement.setString(2, password);
        callableStatement.execute();*/
        String sql = "{ call getCustomerName(? ) }";
        callableStatement.setInt(1, 0);
        System.out.println("Stored Procedure is Executed :)");
    }
    catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

```

5.5 Create an if-else statement to print the results in all conditions

```

 208
 209 public void executeProcedure(String name, String password) {
 210     try {
 211
 212         String sql = "{ call addUser(?, ?) }";
 213         callableStatement = connection.prepareCall(sql);
 214         callableStatement.setString(1, name);
 215         callableStatement.setString(2, password);
 216
 217         callableStatement.execute();
 218
 219         System.out.println("StoredProcedure is Executed :)");
 220     } catch (Exception e) {
 221         System.out.println("Exception Occurred: "+e);
 222     }
 223
 224
 225 public void executeProcedure(int cid) {
 226     try {
 227
 228         String sql = "{ call getCustomerName(?) }";
 229         callableStatement.setInt(1, cid);
 230
 231         ResultSet set = callableStatement.executeQuery();
 232         if(set.next()) {
 233             System.out.println("Found the Customer with ID: "+cid);
 234         } else {
 235             System.out.println("Sorry! No Customer Found with ID: "+cid);
 236         }
 237
 238         System.out.println("StoredProcedure is Executed :)");
 239     } catch (Exception e) {
 240         System.out.println("Exception Occurred: "+e);
 241     }
 242
 243
 244 }
 245

```

5.6 Add a comment line asking the user to enter the customer ID and run the code

```

 23
 24 customer.setInDate("2022-01-12 09:00:00");
 25 customer.setOutDate("2022-01-12 10:30:00");
 26 customer.setTemperature(98.8f);
 27
 28 System.out.println("Connecting to DB....");
 29 DB db = new DB();
 30 db.createConnection();
 31
 32 //db.createCustomer(customer);
 33 //db.updateCustomer(customer);
 34
 35 //db.deleteCustomer(3);
 36
 37 //System.out.println();
 38
 39 ArrayList<Customer> customers = db.getAllCustomers();
 40 customers.forEach(cRef -> System.out.println(cRef));
 41
 42 db.closeConnection();*/
 43
 44 Scanner scanner = new Scanner(System.in);
 45 //System.out.println("Enter Name: ");
 46 //String name = scanner.nextLine();
 47
 48 //System.out.println("Enter Password: ");
 49 //String password = scanner.nextLine();
 50
 51 System.out.println("Enter Customer ID:");
 52 int cid = scanner.nextInt();
 53
 54 scanner.close();
 55
 56
 57 DB db = new DB();
 58 db.createConnection();
 59 //db.executeProcedure(name, password);
 60 db.executeProcedure(cid);
 61 db.closeConnection();

```

The output with the error **Exception Occurred** can be seen.

```

customer.setInDate("2022-01-12 09:00:00");
customer.setOutDate("2022-01-12 10:30:00");
customer.setTemperature(98.0f);

System.out.println("Connecting to DB....");
DB db = new DB();
db.createConnection();

//db.createCustomer(customer);
//db.updateCustomer(customer);

//db.deleteCustomer(3);

//System.out.println();

ArrayList<Customer> customers = db.getAllCustomers();
customers.forEach(cRef -> System.out.println(cRef));

db.closeConnection();*/

Scanner scanner = new Scanner(System.in);
//System.out.println("Enter Name: ");
//String name = scanner.nextLine();

//System.out.println("Enter Password: ");
//String password = scanner.nextLine();

System.out.println("Enter Customer ID:");
int cid = scanner.nextInt();

scanner.close();

DB db = new DB();
db.createConnection();
//db.executeProcedure(name, password);
db.executeProcedure(cid);
db.closeConnection();

```

<terminated> App [Java Application] Aus/eclipse/plugins/org.eclipse.jdt.open/dk.hotspot
Enter Customer ID:
[DB] Driver Loaded
[DB] Connection Created
Exception Occurred: java.lang.NullPointerException: Cannot invoke "java.sql.Connection.close()"
[DB] Connection Closed. Close Status: true

5.7 Go to the **DB.java** file and initialize the **callableStatement**. Open the **App.java** file and run the program.

```

public void executeProcedure(String name, String password) {
    try {
        String sql = "{ call addUser(?, ?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setString(1, name);
        callableStatement.setString(2, password);

        callableStatement.execute();
        System.out.println("Stored Procedure is Executed :)");
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

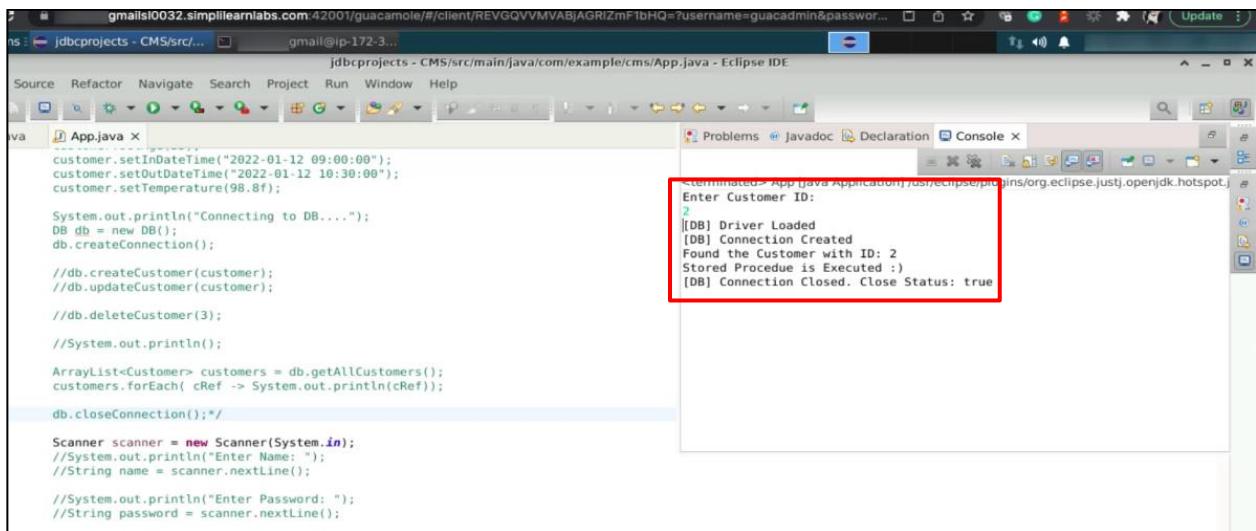
public void executeProcedure(int cid) {
    try {
        String sql = "{ call getCustomerName(?) }";
        callableStatement = connection.prepareCall(sql);
        callableStatement.setInt(1, cid);

        ResultSet set = callableStatement.executeQuery();
        if(set.next()) {
            System.out.println("Found the Customer with ID: "+cid);
        } else {
            System.out.println("Sorry! No Customer Found with ID: "+cid);
        }

        System.out.println("Stored Procedure is Executed :)");
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

```

The output **Stored Procedure is Executed :)** is seen.



A screenshot of the Eclipse IDE interface. The top bar shows the URL "gmails1032.simplilearnlabs.com:42001/guacamole/#/client/BEVGQVVVMVABjAGRIZmF1bHQ=?username=guacadmin&password=guacadmin" and the title "jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE". The menu bar includes "Source", "Refactor", "Navigate", "Search", "Project", "Run", "Window", and "Help". Below the menu is a toolbar with various icons. The main workspace contains a Java file named "App.java" with code related to JDBC operations. To the right of the workspace is a "Console" tab showing the execution output. A red box highlights the output text: "terminated> App [java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jvm.v64 [DB] Driver Loaded [DB] Connection Created Found the Customer with ID: 2 Stored Procedure is Executed :) [DB] Connection Closed. Close Status: true".

```
customer.setInDateTime("2022-01-12 09:00:00");
customer.setOutDateTime("2022-01-12 10:30:00");
customer.setTemperature(98.8f);

System.out.println("Connecting to DB....");
DB db = new DB();
db.createConnection();

//db.createCustomer(customer);
//db.updateCustomer(customer);

//db.deleteCustomer(3);

//System.out.println();

ArrayList<Customer> customers = db.getAllCustomers();
customers.forEach(cRef -> System.out.println(cRef));

db.closeConnection();*/
```

```
<terminated> App [java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jvm.v64
Enter Customer ID:
2
[DB] Driver Loaded
[DB] Connection Created
Found the Customer with ID: 2
Stored Procedure is Executed : )
[DB] Connection Closed. Close Status: true
```

By following these steps, you have successfully created stored procedures and executed them using the JDBC API called **callableStatement** for efficient and reusable database operations. This approach enables the direct invocation of stored procedures defined in the database from a Java application.