

## 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

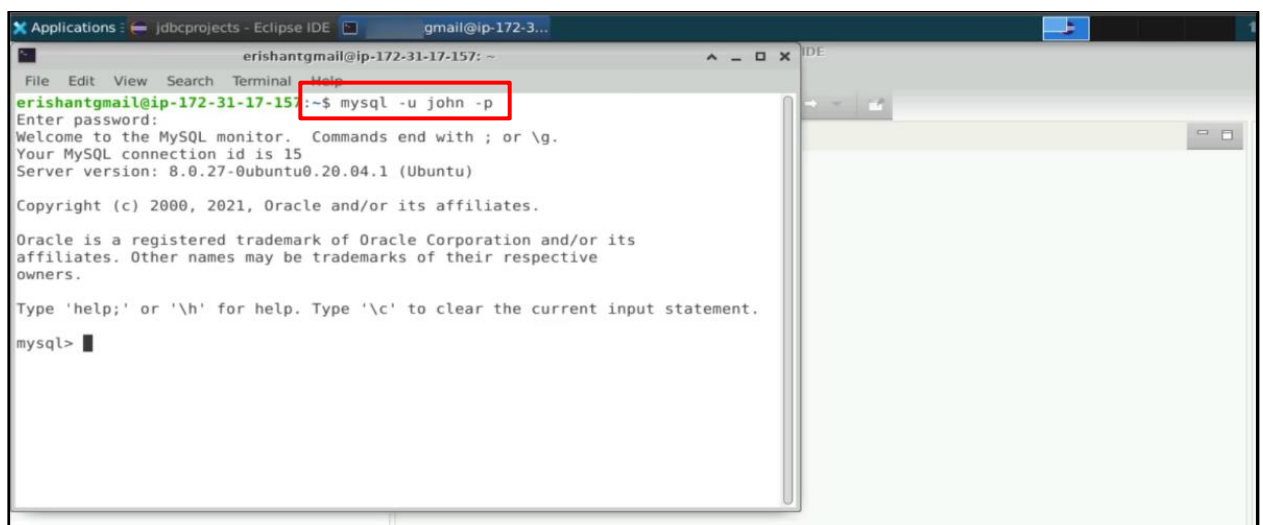
#### Steps to be followed:

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**



```
Applications : jdbcprojects - Eclipse IDE  gmail@ip-172-31-17-157: ~
erishantgmail@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)

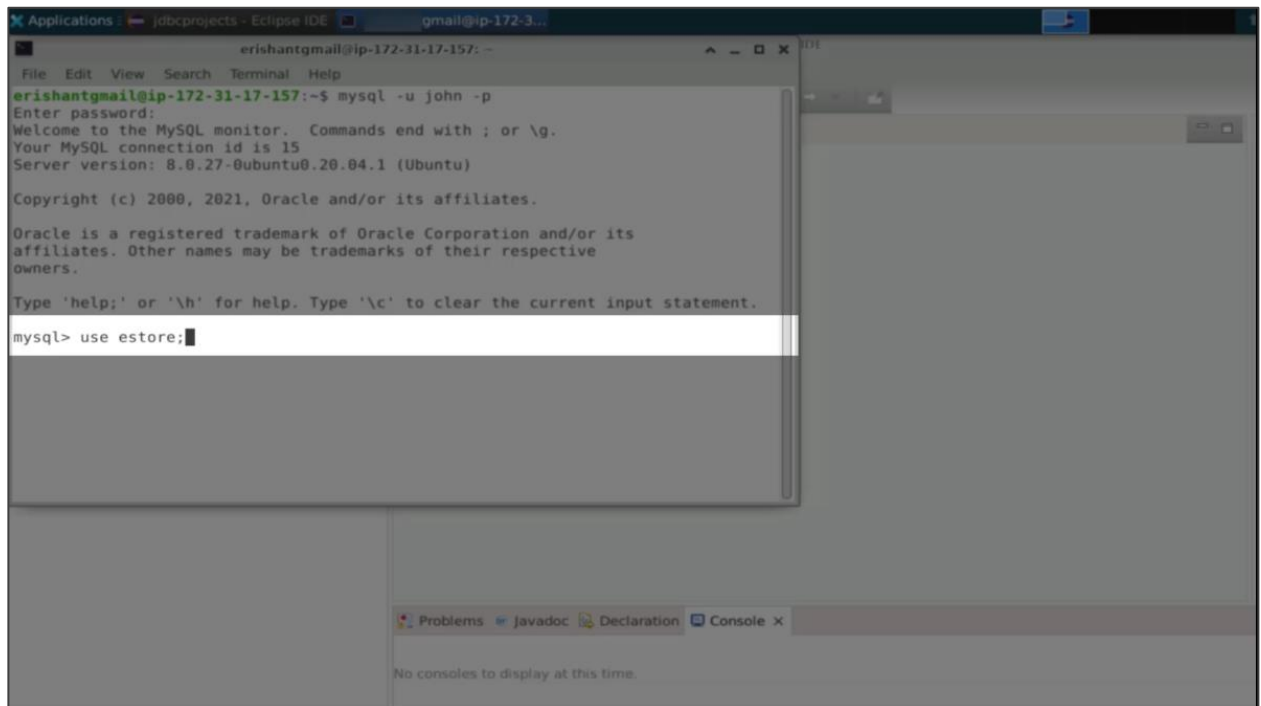
Copyright (c) 2000, 2021, Oracle and/or its affiliates.

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.

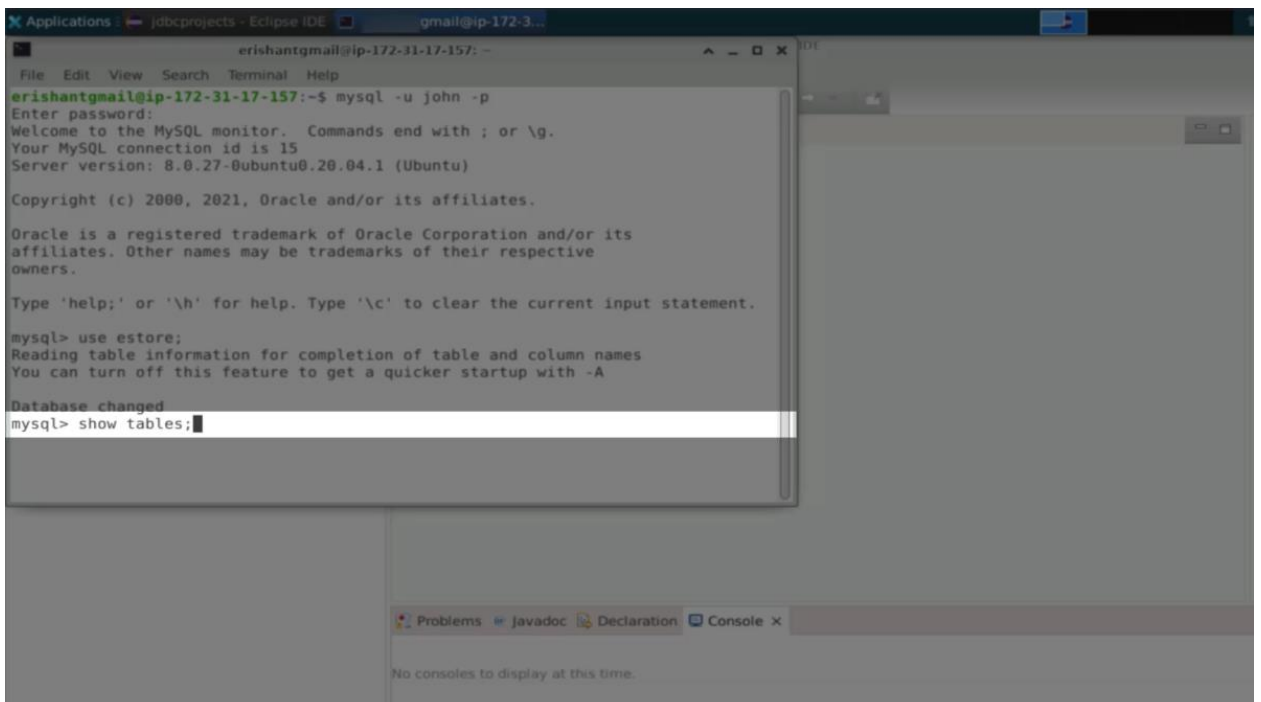
mysql>
```

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



```
erishantgmail@ip-172-31-17-157: ~  
File Edit View Search Terminal Help  
erishantgmail@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;
```

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



```
erishantgmail@ip-172-31-17-157: ~  
File Edit View Search Terminal Help  
erishantgmail@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;
```

```

Applications : jdbcprojects - Eclipse IDE  gmail@ip-172-31-17-157: ~
erishantgmail@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

```

Applications : jdbcprojects - Eclipse IDE  gmail@ip-172-31-17-157: ~
jdbcprojects - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Package Explorer x
  CMS
  JDBCConfiguration
erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
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> 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.

```

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
+-----+
| 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> 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.

```

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
+-----+
| 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> 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. On the left, the Package Explorer displays a project named 'CMS' with a sub-package 'JDBCConfiguration'. The main editor area contains a terminal window titled 'erishantgmail@ip-172-31-17-157: ~'. The terminal shows the following SQL commands and their outputs:

```
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)

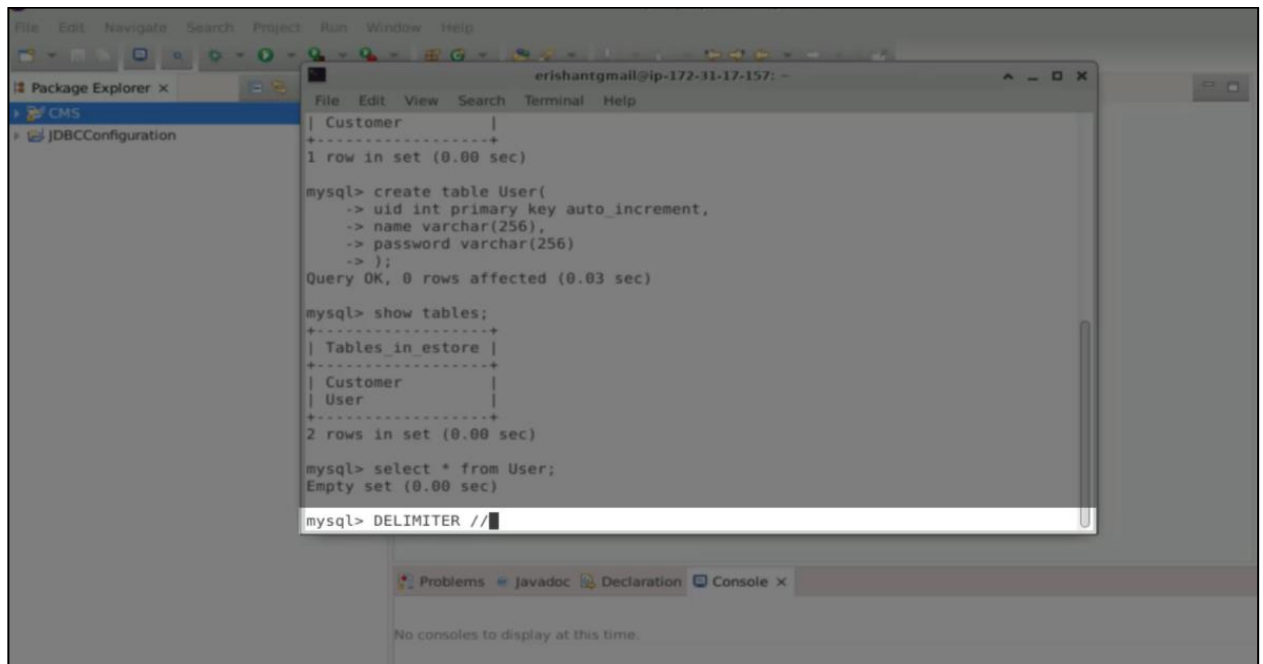
mysql> select * from User;
Empty set (0.00 sec)

mysql>
```

At the bottom of the IDE, there are tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, showing the message '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 an IDE with a terminal window titled "erishantgmail@ip-172-31-17-157: ~". The terminal displays the following MySQL commands and their outputs:

```
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 Package Explorer on the left shows a project named "CMS" with a sub-package "JDBCConfiguration". The bottom of the IDE shows tabs for "Problems", "Javadoc", "Declaration", and "Console".

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

```

Applications: jdbcprojects - Eclipse IDE
jdbcprojects - Eclipse IDE
File Edit Navigate Search Project Run Window Help

Package Explorer x
  CMS
  JDBCConfiguration

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
-> );
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 //
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>
  
```

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

```

Applications: jdbcprojects - Eclipse IDE
jdbcprojects - Eclipse IDE
File Edit Navigate Search Project Run Window Help

Package Explorer x
  CMS
  JDBCConfiguration

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
-> );
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 //
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')
  
```

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

```

Applications : jdbcprojects - Eclipse IDE  gmail@ip-172-31-17-157: ~
jdbcprojects - Eclipse IDE
File Edit Navigate Search Project Run Window Help

Package Explorer X
  CMS
  JDBCConfiguration

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

+-----+
| 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>
  
```

Problems Javadoc Declaration Console x

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

```

Applications : jdbcprojects - Eclipse IDE  gmail@ip-172-31-17-157: ~
jdbcprojects - Eclipse IDE
File Edit Navigate Search Project Run Window Help

Package Explorer X
  CMS
  JDBCConfiguration

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

+-----+
| 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

```

1 package com.example.cms.db;
2
3 import java.sql.Connection;
4
5 /**
6  * JDBC Procedure:
7  * 1. Download the Driver and Link it to the Project. For Maven Project simply configure the d
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
20     final String TAG = "["+getClass().getSimpleName()+"] ";
21
22     public DB() {
23         try {
24             Class.forName("com.mysql.cj.jdbc.Driver");
25             System.out.println(TAG+"Driver Loaded");
26         } catch (Exception e) {
27             System.out.println("Exception Occurred: "+e);
28         }
29     }
30 }
  
```

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     Customer customer = new Customer();
185
186     customer.setCid(set.getInt("cid"));
187     customer.setName(set.getString(2));
188     customer.setPhone(set.getString(3));
189     customer.setEmail(set.getString(4));
190     customer.setBirthDate(set.getString(5));
191     customer.setAge(set.getInt(6));
192     customer.setInDateTime(set.getString(7));
193     customer.setOutDateTime(set.getString(8));
194     customer.setTemperature(set.getFloat(9));
195
196     customers.add(customer);
197 }
198
199 } catch (Exception e) {
200     System.out.println("Exception Occurred: "+e);
201 }
202
203 return customers;
204 }
205
206
207 public 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
7
8
9
10
11
12
13
14
15 /**
16  JDBC Procedure:
17  1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
18  2. Load the Driver from the library i.e. jar file
19  3. Create Connection to the DataBase with url, user and password
20  4. Execute CRUD operations
21  5. Close the Connection
22 */
23
24
25 public class DB implements DAO{
26
27     Connection connection;
28     Statement statement;
29     PreparedStatement preparedStatement;
30     CallableStatement callableStatement;
31
32     final String TAG = "["+getClass().getSimpleName()+"] ";
33
34     public DB() {
35         try {
36             Class.forName("com.mysql.cj.jdbc.Driver");
37             System.out.println(TAG+"Driver Loaded");
38         } catch (Exception e) {
39             System.out.println("Exception Occurred: "+e);
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

```

7 import java.util.ArrayList;
8
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         /*System.out.println( "Welcome to Customer Management System" );
14
15         Customer customer = new Customer();
16
17         customer.setCid(4);
18         customer.setName("George G");
19         customer.setPhone("+91 99009 11111");
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(98.8f);
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         //db.deleteCustomer(3);
34
35         //System.out.println();
36
37         ArrayList<Customer> customers = db.getAllCustomers();
38         customers.forEach( cRef -> System.out.println(cRef));
39
40         db.closeConnection();*/
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 11111");
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(98.8f);
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

```

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

```

188 customer.setCid(set.getInt("cid"));
189 customer.setName(set.getString(2));
190 customer.setPhone(set.getString(3));
191 customer.setEmail(set.getString(4));
192 customer.setBirthDate(set.getString(5));
193 customer.setAge(set.getInt(6));
194 customer.setInDateTime(set.getString(7));
195 customer.setOutDateTime(set.getString(8));
196 customer.setTemperature(set.getFloat(9));
197
198 customers.add(customer);
199
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         String sql = "{ call addUser(?, ?) }";
212         callableStatement = connection.prepareCall(sql);
213         callableStatement.setString(1, "leo");
214         callableStatement.setString(2, "leo@12345");
215
216         callableStatement.execute();
217
218         System.out.println("Stored Procedure is Executed :");
219     } catch (Exception e) {
220         System.out.println("Exception Occurred: "+e);
221     }
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         String sql = "{ call addUser(?, ?) }";
212         callableStatement = connection.prepareCall(sql);
213
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

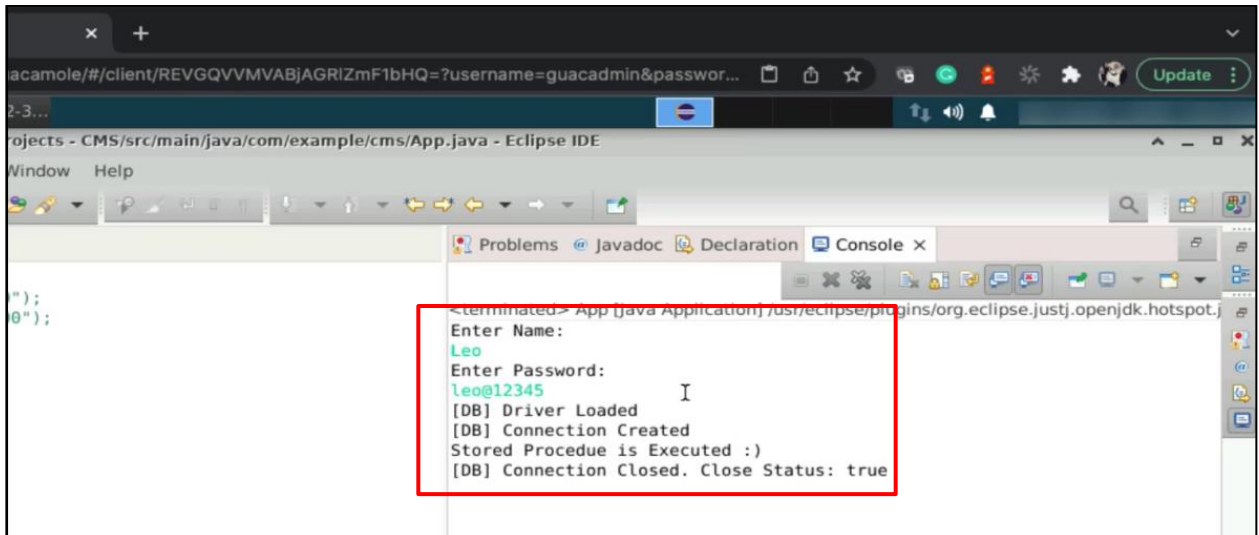
```

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> = 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

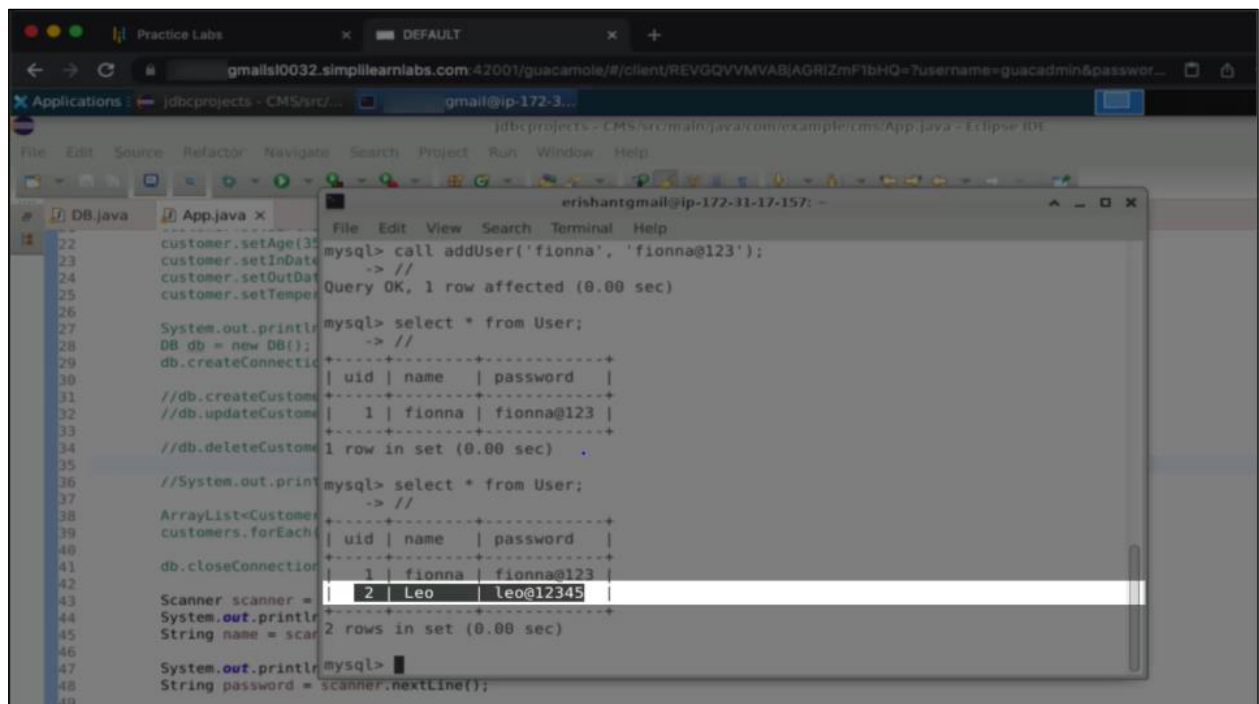
```



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



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.



## Step 4: Create a stored procedure to read data

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

```

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

mysql>
  
```

### 4.2 Create a stored procedure as created earlier

```

mysql> create procedure getCustomerName(IN cid int, OUT customer_name varchar(255))
-> BEGIN
-> select name into customer_name From Customer where cid = cid;
-> END
-> ;
-> //
Query OK, 0 rows affected (0.01 sec)

mysql>
  
```

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

```

mysql> select * from Customer;
+-----+-----+-----+-----+-----+-----+
| cid | name      | phone | outDateTime | email          | temperature | birthDate | a |
+-----+-----+-----+-----+-----+-----+-----+
| 2   | John Watson | +91 98761 22222 | 2022-01-08 11:45:22 | john.watson@example.com | 98.5 | 1990-08-08 | 1 |
| 4   | George G   | +91 99809 11111 | 2022-01-12 10:30:00 | george.g@example.com   | 98.8 | 1986-04-04 | 1 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> call getCustomerName(2);

```

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

```

mysql> call getCustomerName(2);
+-----+
| name      |
+-----+
| John Watson |
+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>

```



## 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 } catch (Exception e) {
202     System.out.println("Exception Occurred: "+e);
203 }
204 }
205
206 return customers;
207 }
208
209 public void executeProcedure(String name, String password) {
210     try {
211         /*String sql = "{ call addUser(?, ?) }";
212         callableStatement = connection.prepareCall(sql);
213         callableStatement.setString(1, name);
214         callableStatement.setString(2, password);
215         callableStatement.execute();*/
216
217         String sql = "{ call getCustomerName(?, ?) }";
218         callableStatement.setInt(1, 0);
219         callableStatement.setInt(2, 0);
220     } catch (Exception e) {
221         System.out.println("Stored Procedure is Executed :");
222     } catch (Exception e) {
223         System.out.println("Exception Occurred: "+e);
224     }
225 }
226 }

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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 //db.createCustomer(customer);
31 //db.updateCustomer(customer);
32 //db.deleteCustomer(3);
33 //System.out.println();
34
35 ArrayList<Customer> customers = db.getAllCustomers();
36 customers.forEach(cRef -> System.out.println(cRef));
37 db.closeConnection();
38
39 Scanner scanner = new Scanner(System.in);
40 //System.out.println("Enter Name: ");
41 //String name = scanner.nextLine();
42 //System.out.println("Enter Password: ");
43 //String password = scanner.nextLine();
44
45 System.out.println("Enter Customer ID:");
46 int cid = scanner.nextInt();
47 scanner.close();
48
49
50
51
52
53
54
55
56 DB db = new DB();
57 db.createConnection();
58 //db.executeProcedure(name, password);
59 db.executeProcedure(cid);
60 db.closeConnection();
  
```

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

```

App.java
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();

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();

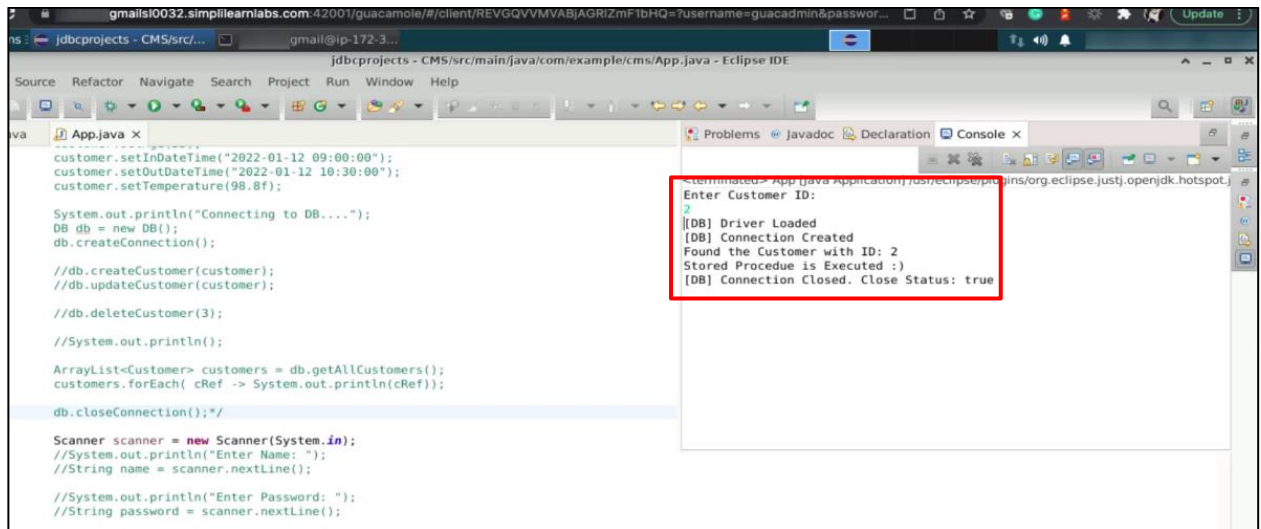
Problems | Javadoc | Declaration | Console
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openjdk.hotspot...
Enter Customer ID:
3
[DB] Driver Loaded
[DB] Connection Created
Exception Occurred: java.lang.NullPointerException: Cannot invoke 'java.sql.CallableStatement' because the variable is null
[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.

```

DB.java
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 public void executeProcedure(int cid) {
226     try {
227
228         String sql = "{ call getCustomerName(?) }";
229         callableStatement = connection.prepareCall(sql);
230         callableStatement.setInt(1, cid);
231
232         ResultSet set = callableStatement.executeQuery();
233         if(set.next()) {
234             System.out.println("Found the Customer with ID: " + cid);
235         } else {
236             System.out.println("Sorry! No Customer Found with ID: " + cid);
237         }
238
239         System.out.println("Stored Procedure is Executed :)");
240     } catch (Exception e) {
241         System.out.println("Exception Occurred: " + e);
242     }
243 }
  
```

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



The screenshot shows the Eclipse IDE with a Java file named `App.java` open. The code includes database connection logic and a scanner for user input. The console output, highlighted with a red box, shows the execution of the application:

```

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.