

## Lesson 01 Demo 03

### Performing CRUD Operations

**Objective:** To perform create, read, update, and delete operations on the created database for managing customer records effectively

**Tool required:** Eclipse IDE

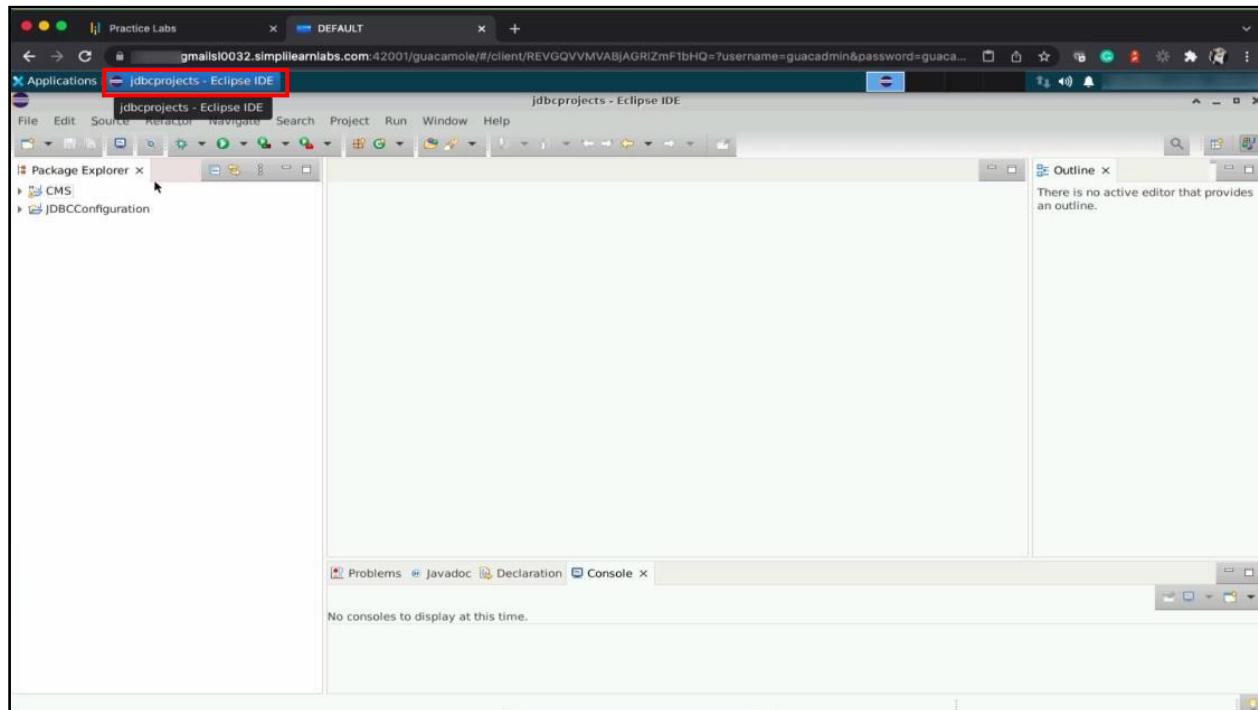
**Prerequisites:** Lesson 01 Demo 02

#### Steps to be followed:

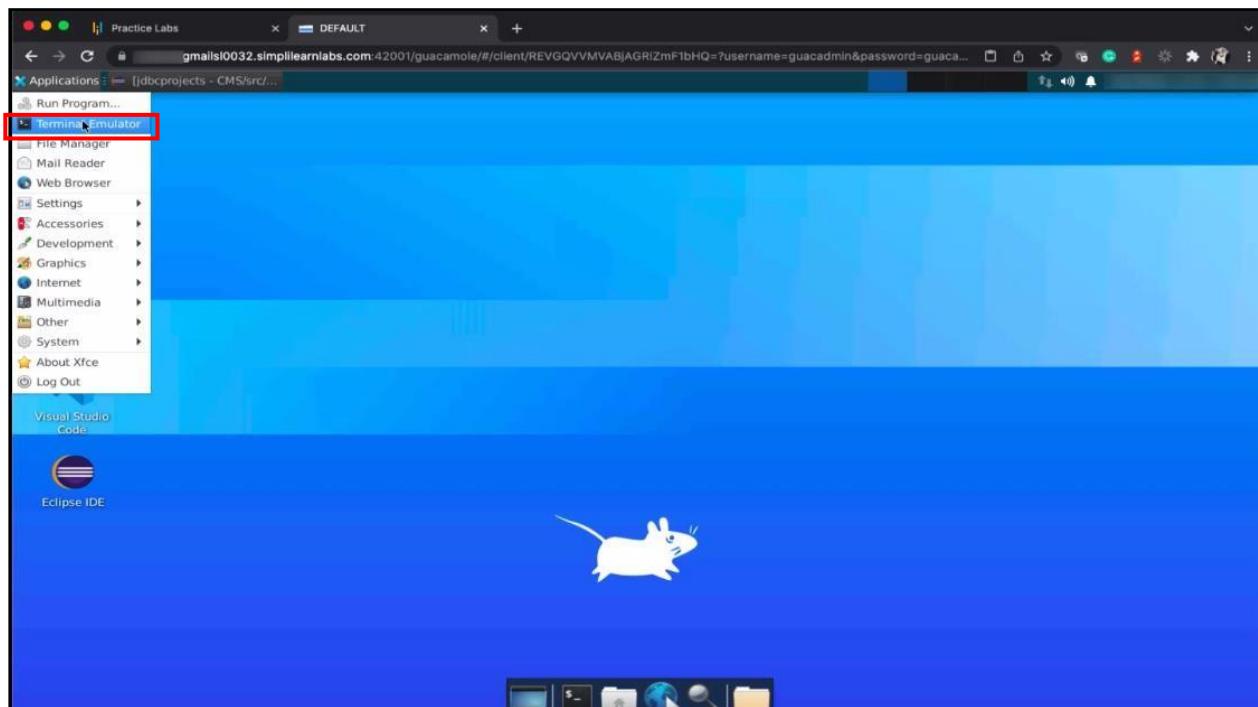
1. Open the Terminal Emulator
2. Write an insert operation in Eclipse IDE
3. Create an SQL query for the insert operation
4. Write an update operation
5. Perform the getallcustomer operations
6. Perform the delete operation

## Step 1: Open the Terminal Emulator

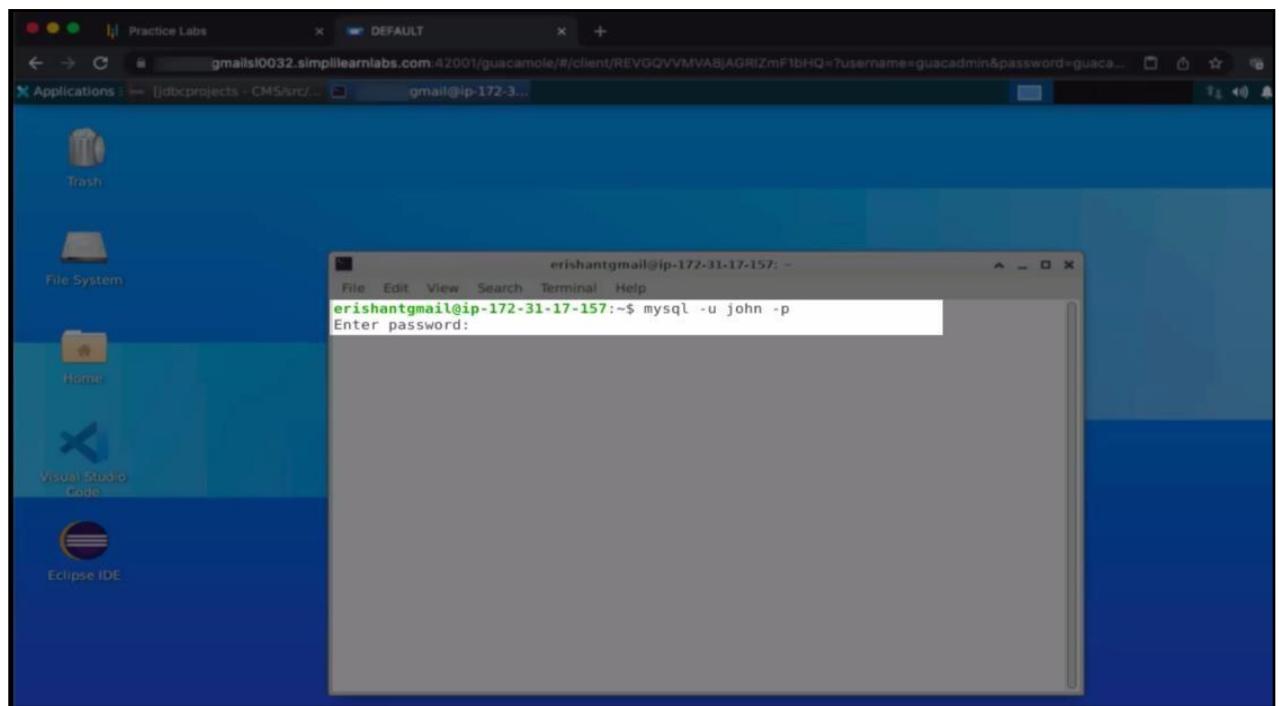
### 1.1 Open Eclipse IDE



## 1.2 Open the Terminal Emulator

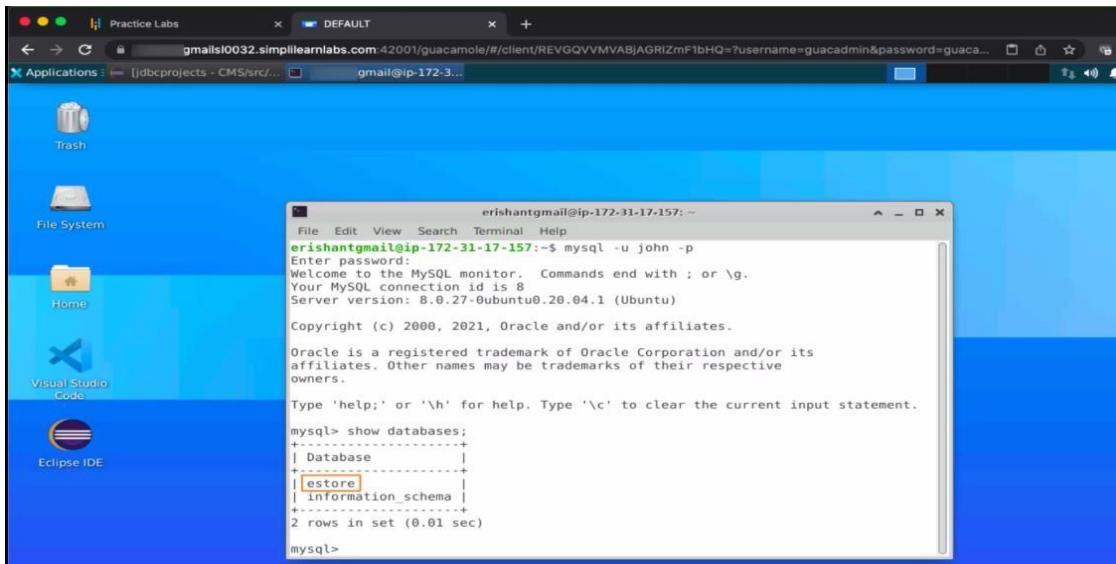


### 1.3 Log in to MySQL using the command **mysql -u john -p**



**Note:** A user named **John** has already created for the database.

#### 1.4 Type `show databases;` and list all the databases available



The screenshot shows a Linux desktop environment with a blue-themed desktop. On the left, there is a dock with icons for Practice Labs, Applications, and several application windows. One application window titled "Terminal" is open, showing a MySQL session. The terminal output is as follows:

```
erishant@gmail@ip-172-31-17-157: ~
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 8
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

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

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

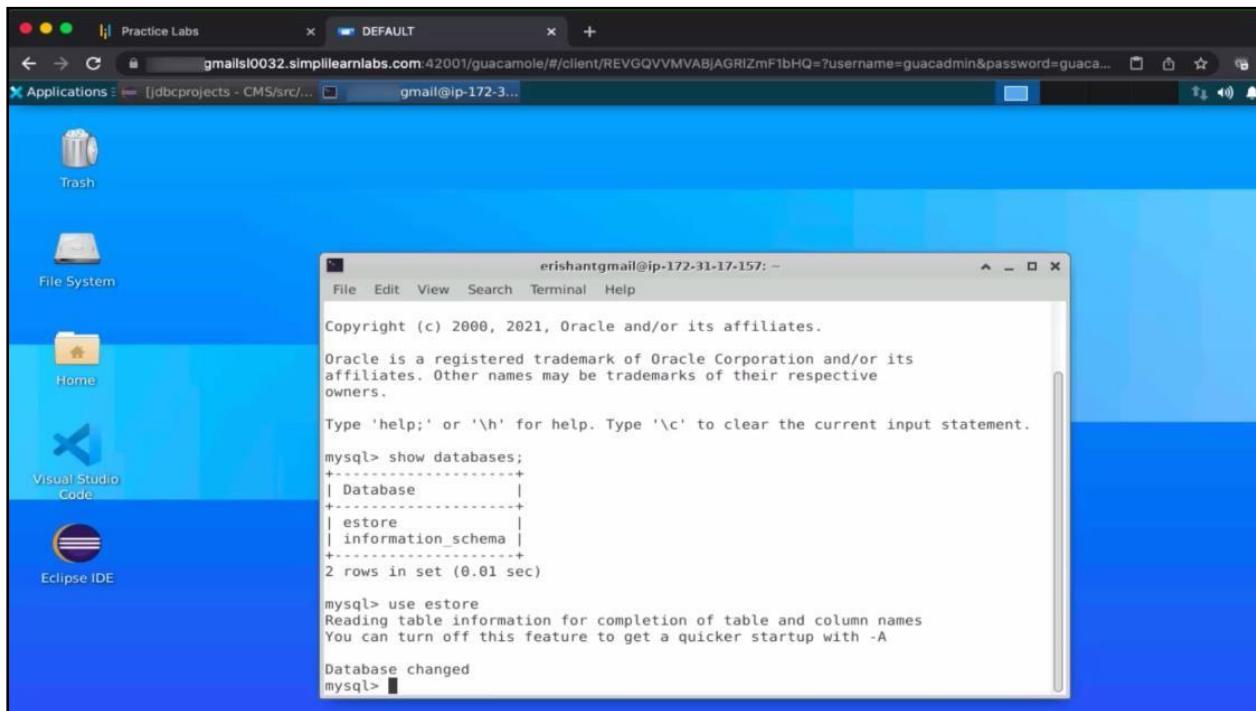
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| testore |
| information_schema |
+-----+
2 rows in set (0.01 sec)

mysql>
```

The database "testore" is highlighted with a red box.

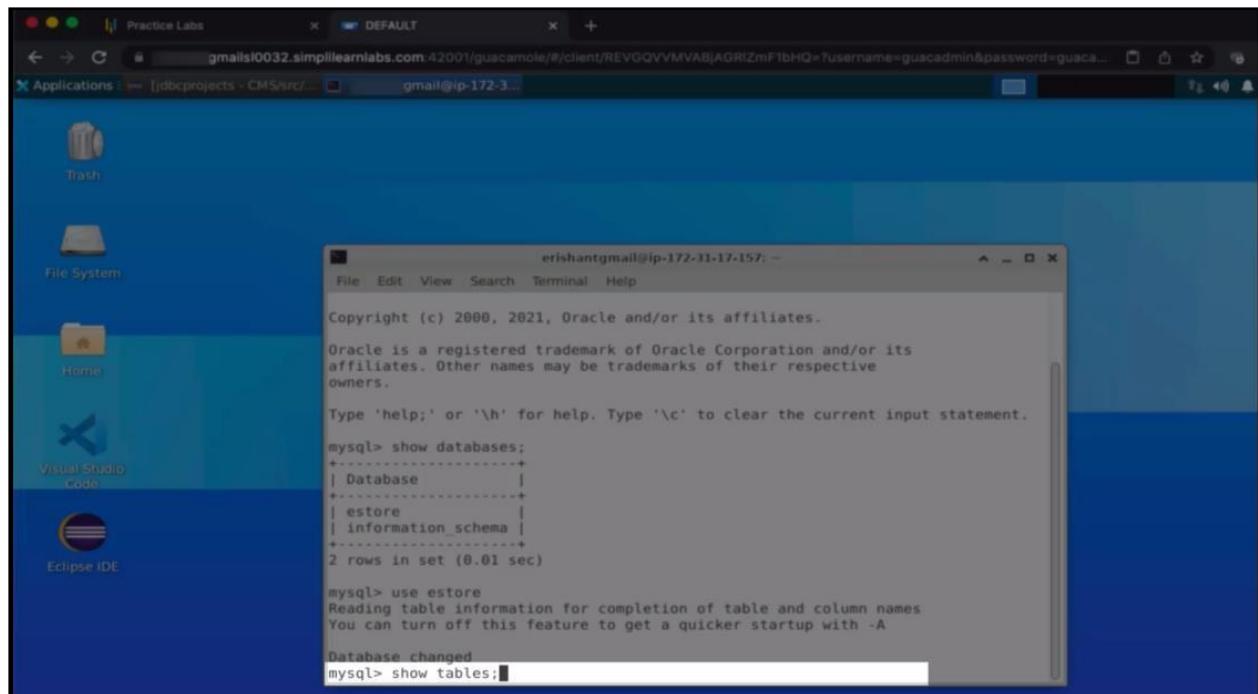
### 1.5 Type **use estore;** and change the database to **estore** database



The screenshot shows a Linux desktop environment with a blue gradient background. On the left, there is a vertical dock containing icons for 'Trash', 'File System', 'Home', 'Visual Studio Code', and 'Eclipse IDE'. A terminal window titled 'erishant@gmail@ip-172-31-17-157: ~' is open, displaying the following MySQL session:

```
Copyright (c) 2000, 2021, Oracle and/or its affiliates.  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| estore |  
| information_schema |  
+-----+  
2 rows in set (0.01 sec)  
  
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> [REDACTED]
```

### 1.6 Type `show tables;` to list the tables available in the `estore` database



A screenshot of a Linux desktop environment. On the left, there's a dock with icons for Trash, File System, Home, Visual Studio Code, and Eclipse IDE. In the center, a terminal window titled "erishant@gmail@ip-172-31-17-157: ~" is open. The terminal shows the following MySQL session:

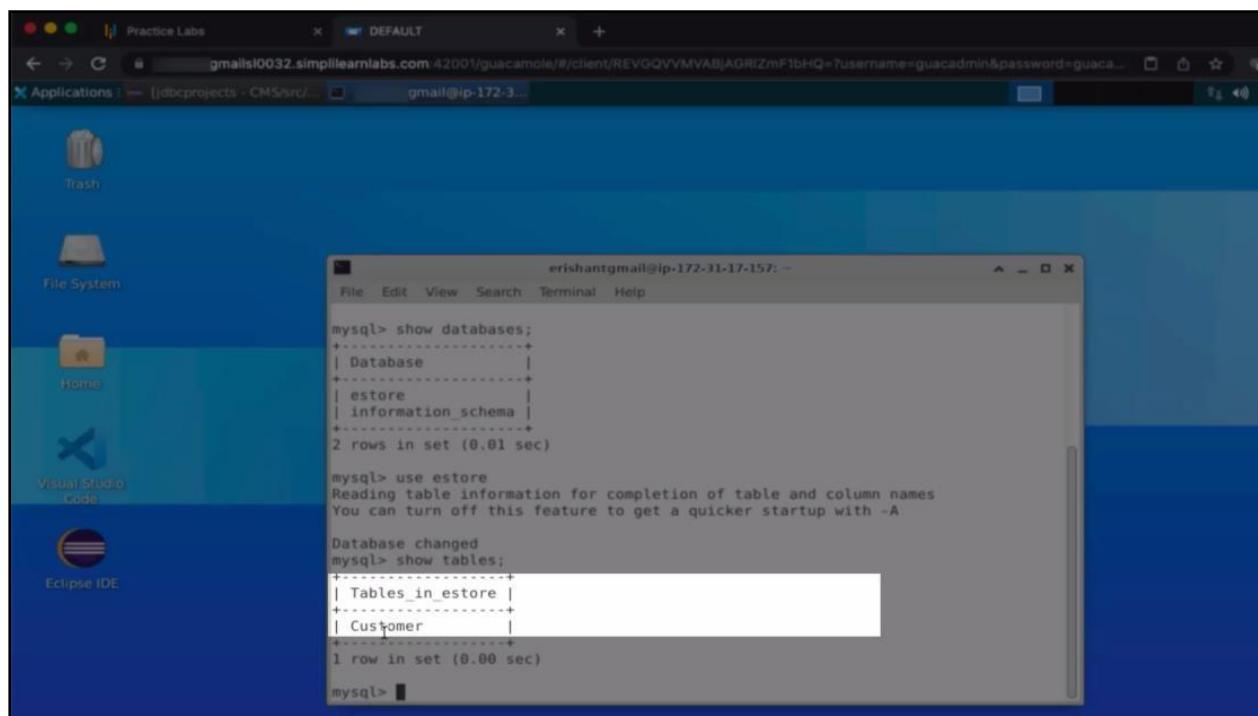
```
Copyright (c) 2000, 2021, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+-----+
| Database |
+-----+
| estore   |
| information_schema |
+-----+
2 rows in set (0.01 sec)

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;
```



A screenshot of a Linux desktop environment, identical to the one above. The terminal window shows the following MySQL session:

```
mysql> show databases;
+-----+
| Database |
+-----+
| estore   |
| information_schema |
+-----+
2 rows in set (0.01 sec)

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>
```

### 1.7 Type **describe Customer;** to get detailed information on the table

```

Practice Labs x DEFAULT
gmails10032.simplilearnlabs.com:42001/guacamole/#/client/REVGQVVMVABJAGRIZmF1bHQ=?username=guacadmin&password=guaca...
Applications : [jdbcprojects - CMS/src/... gmail@ip-172-31-17-157: ~
Trash
File System
Home
Visual Studio Code
Eclipse IDE

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

mysql> describe Customer;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cid | int | NO | PRI | NULL | auto_increment |
| name | varchar(256) | YES | | NULL |
| phone | varchar(20) | YES | | NULL |
| email | varchar(256) | YES | | NULL |
| birthDate | date | YES | | NULL |
| age | int | YES | | NULL |
| inDateTime | datetime | YES | | NULL |
| outDateTime | datetime | YES | | NULL |
| temperature | float | YES | | NULL |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

mysql> 
```

### 1.8 Type **select \* from Customer;** to select data from the table

```

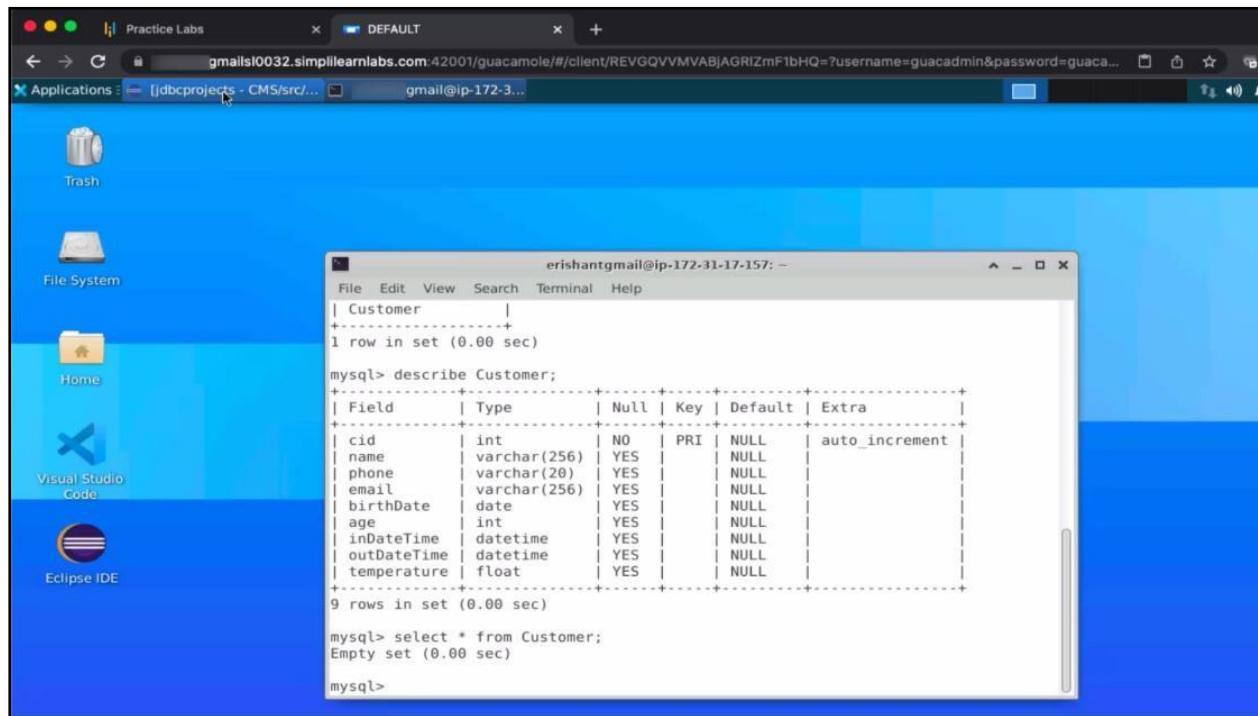
Practice Labs x DEFAULT
gmails10032.simplilearnlabs.com:42001/guacamole/#/client/REVGQVVMVABJAGRIZmF1bHQ=?username=guacadmin&password=guaca...
Applications : [jdbcprojects - CMS/src/... gmail@ip-172-31-17-157: ~
Trash
File System
Home
Visual Studio Code
Eclipse IDE

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

mysql> describe Customer;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cid | int | NO | PRI | NULL | auto_increment |
| name | varchar(256) | YES | | NULL |
| phone | varchar(20) | YES | | NULL |
| email | varchar(256) | YES | | NULL |
| birthDate | date | YES | | NULL |
| age | int | YES | | NULL |
| inDateTime | datetime | YES | | NULL |
| outDateTime | datetime | YES | | NULL |
| temperature | float | YES | | NULL |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

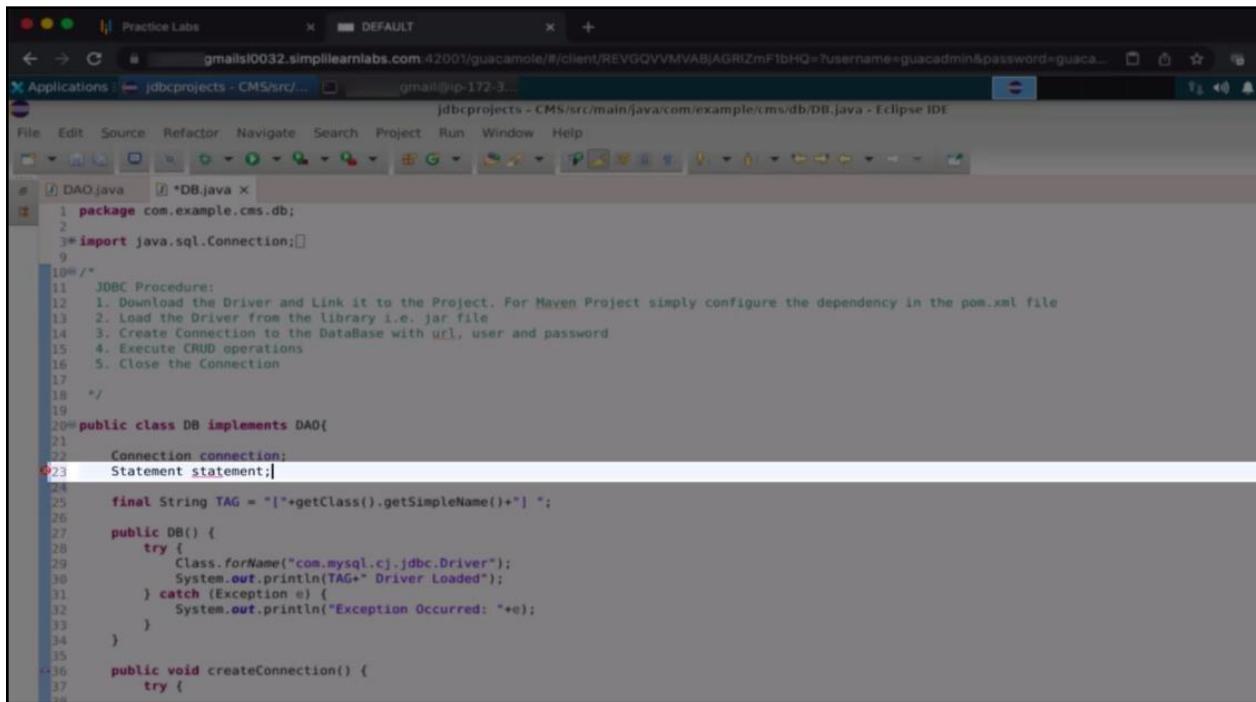
mysql> select * from Customer;

```



## Step 2: Write an insert operation in Eclipse IDE

### 2.1 Handle exception errors by typing the API call statement using the **Statement** object

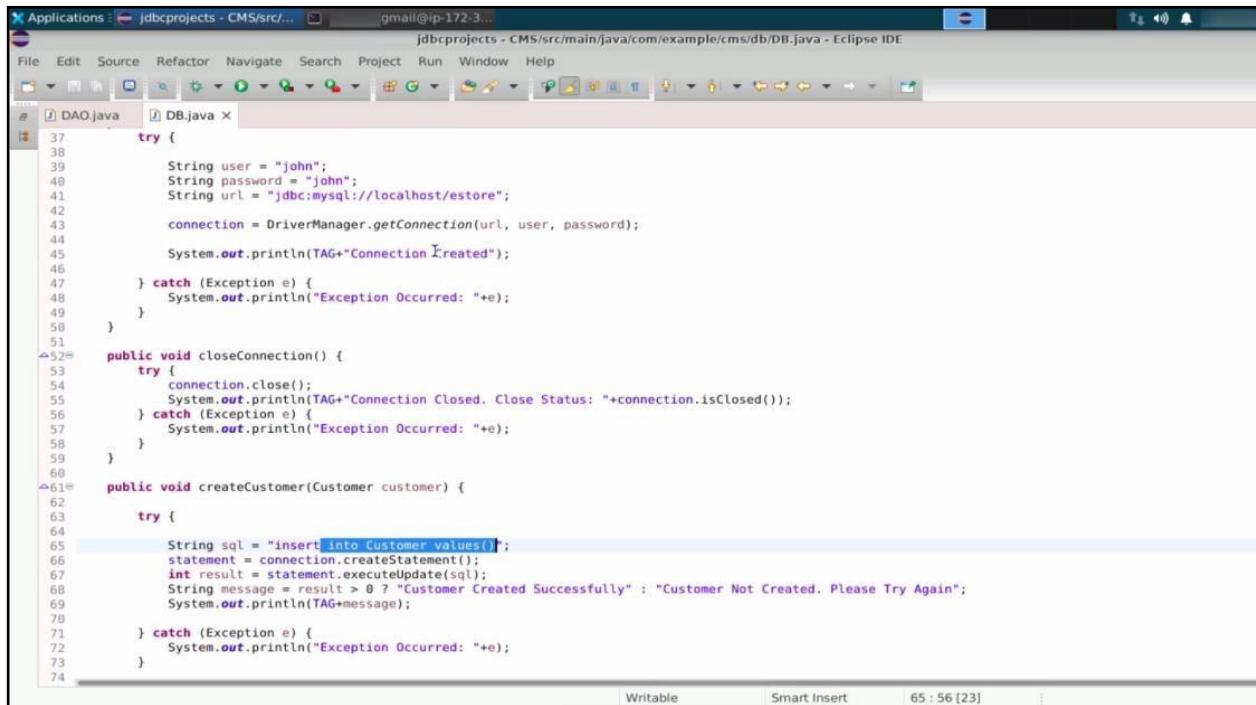


The screenshot shows the Eclipse IDE interface with a Java file named 'DB.java' open. The code implements a JDBC connection and statement handling. A red dot marker is placed on the line 'Statement statement;' in the code.

```
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 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
19     final String TAG = "["+getClass().getSimpleName()+"] ";
20
21     public DB() {
22         try {
23             Class.forName("com.mysql.cj.jdbc.Driver");
24             System.out.println(TAG+" Driver Loaded");
25         } catch (Exception e) {
26             System.out.println("Exception Occurred: "+e);
27         }
28     }
29
30     public void createConnection() {
31         try {
```

2.2 Type the insert operation inside the **try-catch** block:

**String sql = "insert into Customer values()"**

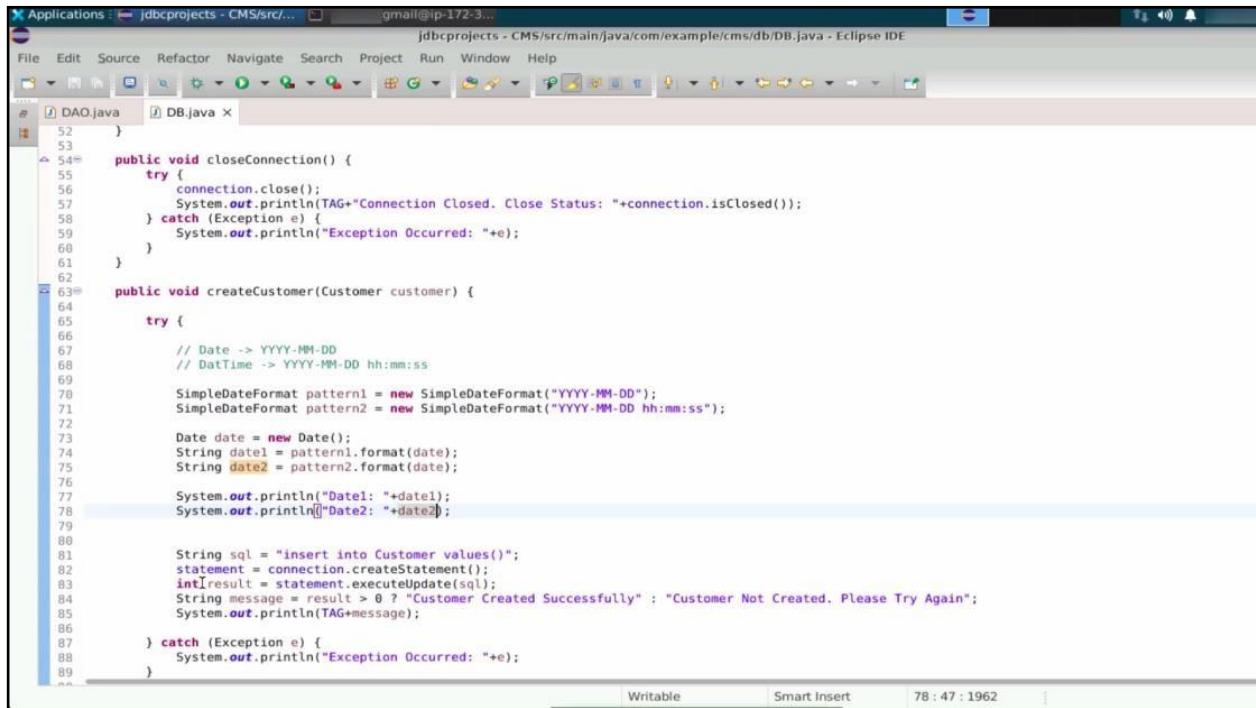


The screenshot shows the Eclipse IDE interface with the title bar "Applications : jdbcprojects - CMS/src/... gmail@ip-172-3..." and "jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The code editor window contains Java code for a DAO class:

```
37  try {
38
39      String user = "john";
40      String password = "john";
41      String url = "jdbc:mysql://localhost/estore";
42
43      connection = DriverManager.getConnection(url, user, password);
44
45      System.out.println(TAG+"Connection Created");
46
47  } catch (Exception e) {
48      System.out.println("Exception Occurred: "+e);
49  }
50
51
52  public void closeConnection() {
53
54      try {
55          connection.close();
56      } catch (Exception e) {
57          System.out.println("Exception Occurred: "+e);
58      }
59
60
61  public void createCustomer(Customer customer) {
62
63      try {
64
65          String sql = "insert into Customer values();";
66          statement = connection.createStatement();
67          int result = statement.executeUpdate(sql);
68          String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
69          System.out.println(TAG+message);
70
71      } catch (Exception e) {
72          System.out.println("Exception Occurred: "+e);
73      }
74 }
```

The code uses JDBC to connect to a MySQL database, create a customer record, and handle exceptions. The SQL statement "insert into Customer values();" is intended to be placed inside the try block of the createCustomer method.

## 2.3 Create a simple date format API from line 65

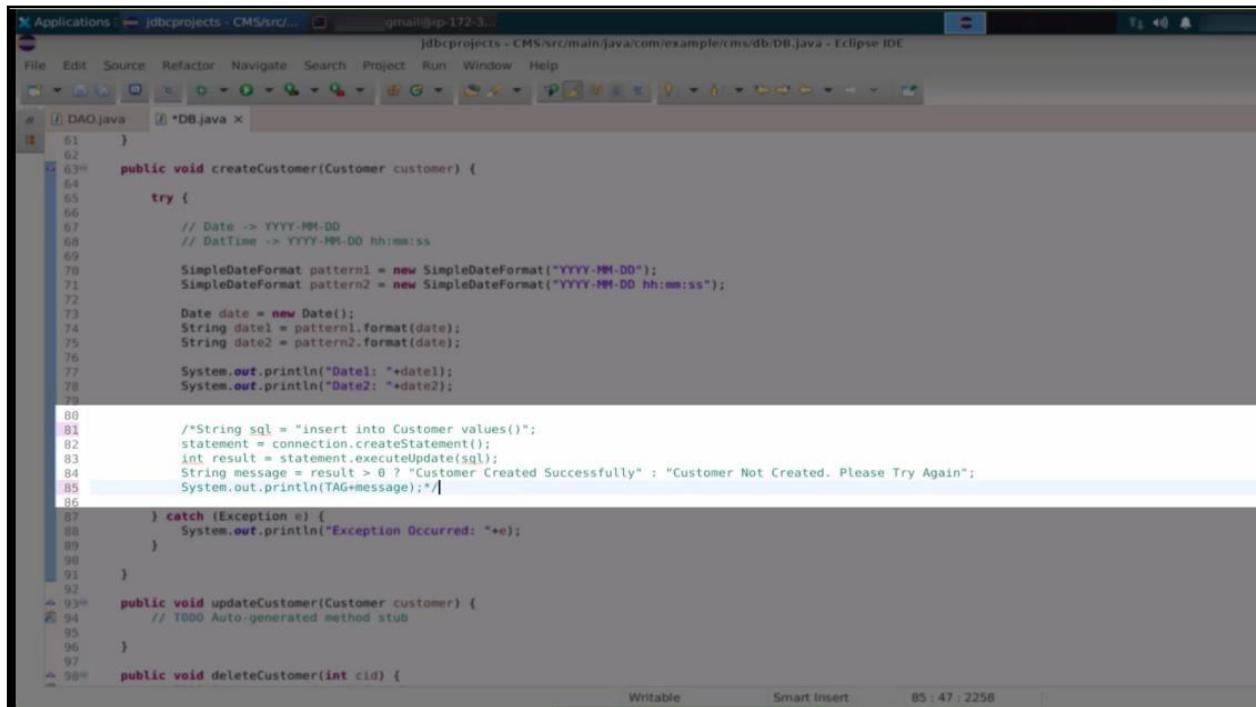


```

52
53
54    public void closeConnection() {
55        try {
56            connection.close();
57            System.out.println(TAG+"Connection Closed. Close Status: "+connection.isClosed());
58        } catch (Exception e) {
59            System.out.println("Exception Occurred: "+e);
60        }
61    }
62
63    public void createCustomer(Customer customer) {
64        try {
65            // Date -> YYYY-MM-DD
66            // DateTime -> YYYY-MM-DD hh:mm:ss
67            SimpleDateFormat pattern1 = new SimpleDateFormat("YYYY-MM-DD");
68            SimpleDateFormat pattern2 = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");
69
70            Date date = new Date();
71            String date1 = pattern1.format(date);
72            String date2 = pattern2.format(date);
73
74            System.out.println("Date1: "+date1);
75            System.out.println("Date2: "+date2);
76
77            String sql = "insert into Customer values();";
78            statement = connection.createStatement();
79            int result = statement.executeUpdate(sql);
80            String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
81            System.out.println(TAG+message);
82
83        } catch (Exception e) {
84            System.out.println("Exception Occurred: "+e);
85        }
86    }
87
88
89

```

## 2.4 Comment out the following code:



```

61
62
63    public void createCustomer(Customer customer) {
64        try {
65            // Date -> YYYY-MM-DD
66            // DateTime -> YYYY-MM-DD hh:mm:ss
67            SimpleDateFormat pattern1 = new SimpleDateFormat("YYYY-MM-DD");
68            SimpleDateFormat pattern2 = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");
69
70            Date date = new Date();
71            String date1 = pattern1.format(date);
72            String date2 = pattern2.format(date);
73
74            System.out.println("Date1: "+date1);
75            System.out.println("Date2: "+date2);
76
77
78
79
80            /*String sql = "insert into Customer values();";
81            statement = connection.createStatement();
82            int result = statement.executeUpdate(sql);
83            String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
84            System.out.println(TAG+message);*/
85
86        } catch (Exception e) {
87            System.out.println("Exception Occurred: "+e);
88        }
89
90
91
92    public void updateCustomer(Customer customer) {
93        // TODO Auto-generated method stub
94
95    }
96
97    public void deleteCustomer(int cid) {
98

```

## 2.5 Navigate to the App.java file

```

1 package com.example.cms;
2
3 import com.example.cms.DB;
4
5 /**
6 * Hello world!
7 */
8
9
10 public class App {
11     public static void main( String[] args ) {
12         System.out.println( "Welcome to Customer Management System" );
13
14         Customer customer = new Customer();
15         customer.setName("Fionna");
16         customer.setPhone("+91 99999 11111");
17
18         System.out.println("Connecting to DB....");
19         DB db = new DB();
20         db.createConnection();
21         db.createCustomer(customer);
22
23         db.closeConnection();
24     }
25
26 }
27
28
29

```

## 2.6 Call the db.createCustomer() method

```

1 package com.example.cms;
2
3 import com.example.cms.DB;
4
5 /**
6 * Hello world!
7 */
8
9
10 public class App {
11     public static void main( String[] args ) {
12         System.out.println( "Welcome to Customer Management System" );
13
14         Customer customer = new Customer();
15         customer.setName("Fionna");
16         customer.setPhone("+91 99999 11111");
17
18         System.out.println("Connecting to DB....");
19         DB db = new DB();
20         db.createConnection();
21         db.createCustomer(customer);
22
23         db.closeConnection();
24     }
25
26 }
27
28
29

```

## 2.7 Run the code to get the output as **Connection Created** and the dates

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

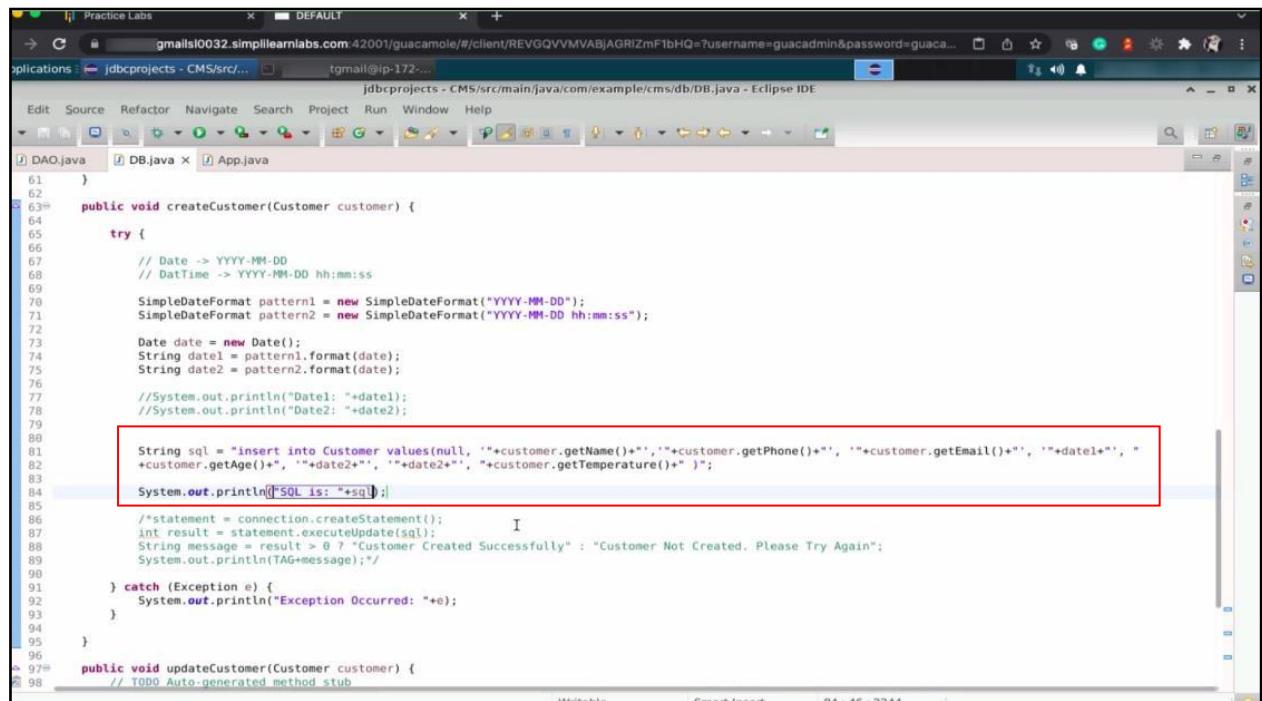
- Title Bar:** Applications : jdbcprojects - CMS/src... | gmail@ip-172-3...
- Toolbar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Left Panel:** Shows three files: DAO.java, DB.java, and App.java.
- Code Editor (App.java):**

```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  *
8  */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         customer.setName("Fionna");
18         customer.setPhone("+91 99999 1111");
19
20         System.out.println("Connecting to DB....");
21         DB db = new DB();
22         db.createConnection();
23
24         db.createCustomer(customer);
25
26         db.closeConnection();
27     }
28 }
```
- Console View:** Displays the application's output:

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.jst.jdt.op
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
Date1: 2022-01-08
Date2: 2022-01-08 11:31:52
[DB] Connection Closed. Close Status: true
```

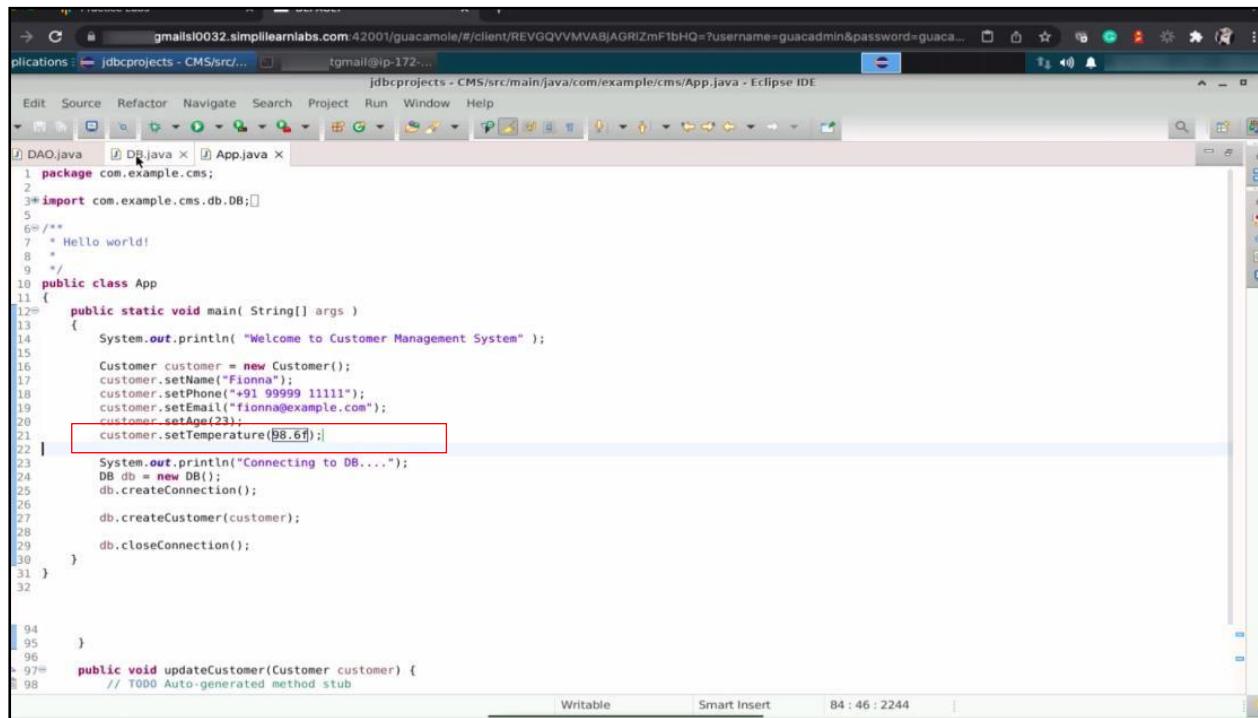
## Step 3: Create a SQL query for the insert operation

### 3.1 Create a SQL query to perform the insert operation and enter details into the Customer table



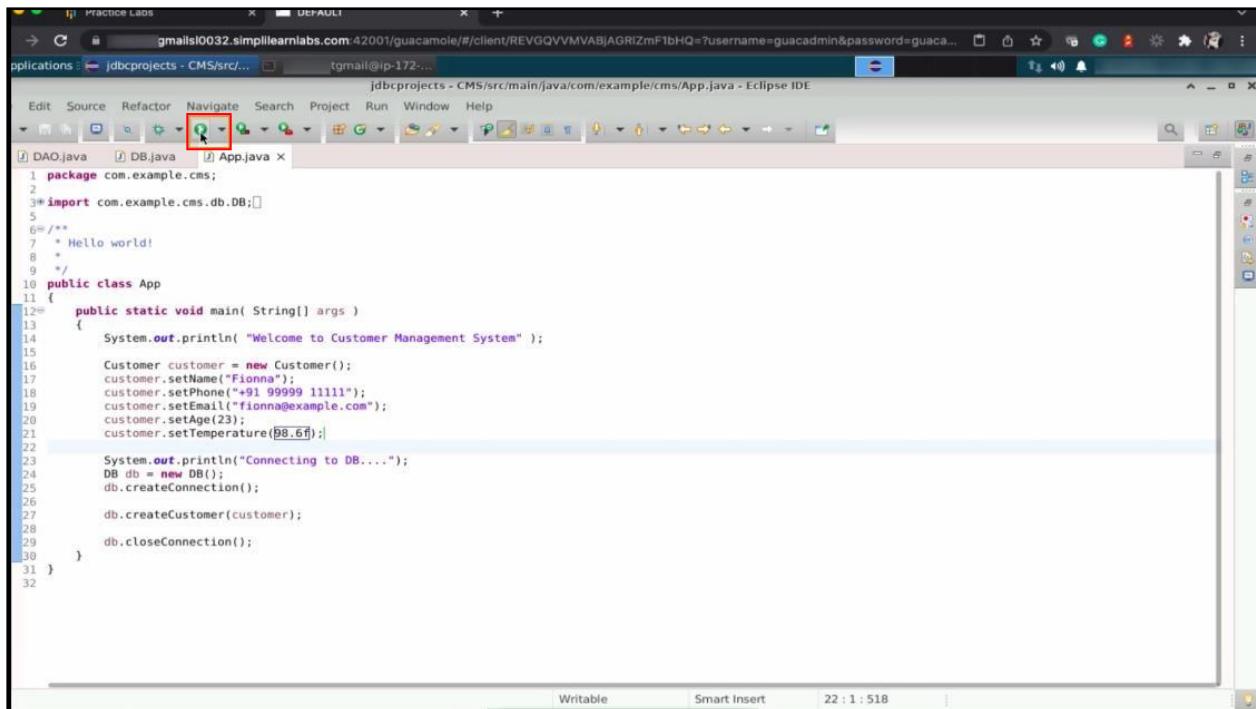
```
 61  }
 62
 63  public void createCustomer(Customer customer) {
 64    try {
 65
 66      // Date -> YYYY-MM-DD
 67      // DateTime -> YYYY-MM-DD hh:mm:ss
 68
 69      SimpleDateFormat pattern1 = new SimpleDateFormat("YYYY-MM-DD");
 70      SimpleDateFormat pattern2 = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");
 71
 72      Date date = new Date();
 73      String date1 = pattern1.format(date);
 74      String date2 = pattern2.format(date);
 75
 76      //System.out.println("Date1: "+date1);
 77      //System.out.println("Date2: "+date2);
 78
 79
 80      String sql = "insert into Customer values(null, '"+customer.getName()+"','"+customer.getPhone()+"','"+customer.getEmail()+"', '"+date1+"', "
 81      "+customer.getAge()+"", "+date2+", "+date2+", "+customer.getTemperature()+" )";
 82
 83      System.out.println("SQL is: "+sql);
 84
 85
 86      /*statement = connection.createStatement();           I
 87      int result = statement.executeUpdate(sql);
 88      String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
 89      System.out.println(TAG+message);*/
 90
 91    } catch (Exception e) {
 92      System.out.println("Exception Occurred: "+e);
 93    }
 94  }
 95
 96
 97  public void updateCustomer(Customer customer) {
 98    // TODO Auto-generated method stub
```

### 3.2 Return to the App.java file and add more attributes like age and temperature



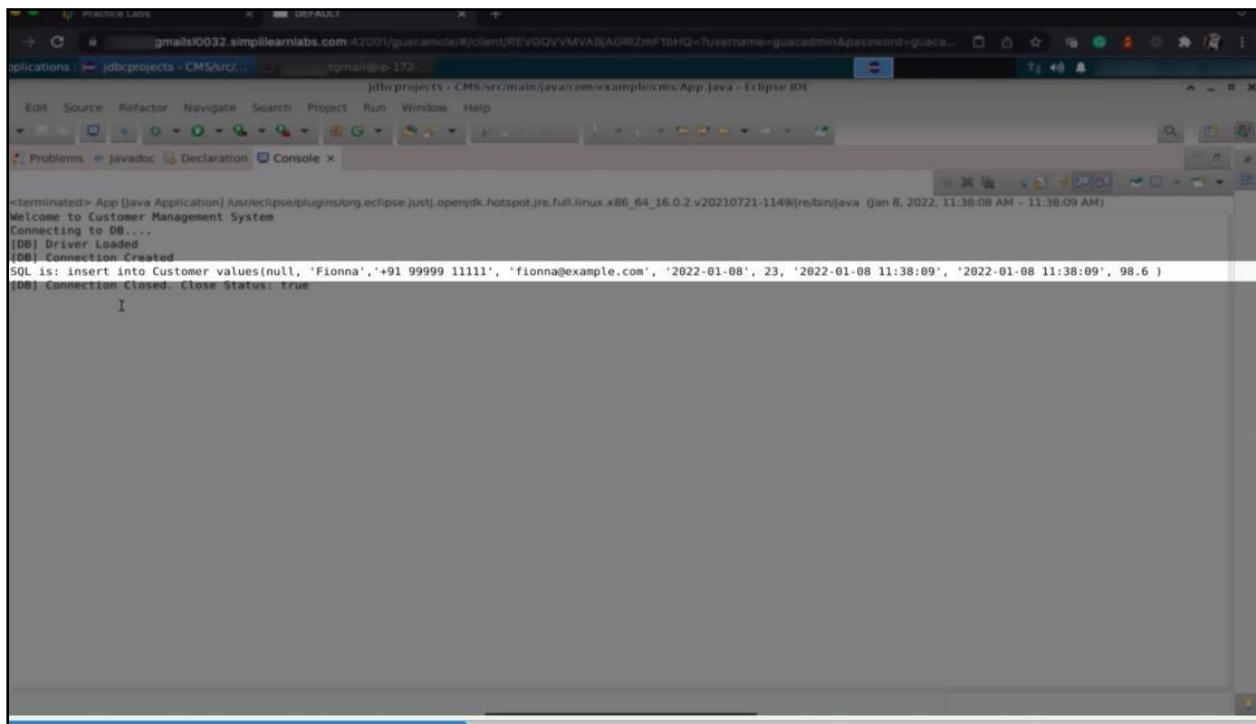
```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6 * Hello world!
7 */
8
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         customer.setName("Fionna");
18         customer.setPhone("+91 99999 1111");
19         customer.setEmail("fionna@example.com");
20         customer.setAge(23);
21         customer.setTemperature(88.6f);
22
23         System.out.println("Connecting to DB....");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47 public void updateCustomer(Customer customer) {
48     // TODO Auto-generated method stub
49 }
```

### 3.3 Save and run the code



```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  */
8
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         customer.setName("Fionna");
18         customer.setPhone("+91 99999 11111");
19         customer.setEmail("fionna@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.6);
22
23         System.out.println("Connecting to DB....");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
```

The SQL statement will be printed as written in the print statement.



```
Welcome to Customer Management System
Connecting to DB....
[DB] Driver Loaded
[DB] Connection Created
SQL is: insert into Customer values(null, 'Fionna', '+91 99999 11111', 'fionna@example.com', '2022-01-08', 23, '2022-01-08 11:38:09', '2022-01-08 11:38:09', 98.6 )
[DB] Connection Closed. Close Status: true
```

### 3.4 Navigate to the **DB.java** file and uncomment the connection string part to establish a connection to the database for further operations:

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6 * Hello world!
7 */
8
9
10 public class App {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14
15         Customer customer = new Customer();
16         customer.setName("Fionna");
17         customer.setPhone("+91 99999 11111");
18         customer.setEmail("fionna@example.com");
19         customer.setAge(23);
20         customer.setTemperature(98.6);
21
22         System.out.println("Connecting to DB....");
23         DB db = new DB();
24         db.createConnection();
25
26         db.createCustomer(customer);
27
28         db.closeConnection();
29     }
30 }
31
32

```

### 3.5 Run the code and observe the output **Customer Created Successfully** confirming the successful creation of the customer.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6 * Hello world!
7 */
8
9
10 public class App {
11     public static void main( String[] args )
12     {
13         System.out.println( "Welcome to Customer Management System" );
14
15         Customer customer = new Customer();
16         customer.setName("Fionna");
17         customer.setPhone("+91 99999 11111");
18         customer.setEmail("fionna@example.com");
19         customer.setAge(23);
20         customer.setTemperature(98.6);
21
22         System.out.println("Connecting to DB....");
23         DB db = new DB();
24         db.createConnection();
25
26         db.createCustomer(customer);
27
28         db.closeConnection();
29     }
30 }
31
32

```

Output in the Console tab:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jdk11
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
SQL is: insert into Customer values(null, 'Fionna', '+91 99999 11111', 'fionna@example.com', 23, 98.6)
[DB] Customer Created Successfully
[DB] Connection Closed. Close Status: true

```

3.6 Go to the terminal and run the select command:

```
select * from Customer;
```

The screenshot shows the Eclipse IDE interface with a Java project named "cms" open. The code editor displays a file named "App.java" containing Java code that interacts with a MySQL database. The terminal window shows the execution of SQL commands to describe a "Customer" table and insert data into it. The MySQL output indicates 9 rows inserted.

```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6 * Hello world!
7 */
8
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setName("Fionna");
18         customer.setPhone("+91 99999 11111");
19         customer.setEmail("fionna@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.6);
22
23         System.out.println("Customer created successfully!");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32
```

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

mysql> describe Customer;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| cid | int | NO | PRI | NULL | auto_increment |
| name | varchar(256) | YES | | NULL | |
| phone | varchar(20) | YES | | NULL | |
| email | varchar(256) | YES | | NULL | |
| birthDate | date | YES | | NULL | |
| age | int | YES | | NULL | |
| inDateTime | datetime | YES | | NULL | |
| outDateTime | datetime | YES | | NULL | |
| temperature | float | YES | | NULL | |
+-----+
9 rows in set (0.00 sec)

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

mysql> select * from Customer;
```

You can view the inserted data in the table **Customer**.

The screenshot shows the Eclipse IDE interface with several open windows:

- Left pane:** Shows the file structure with DAO.java, DB.java, and App.java.
- Middle pane:** Displays the Java code for App.java:

```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6 * Hello world!
7 */
8
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setName("Fionna");
18         customer.setPhone("+91 99999 11111");
19         customer.setEmail("fionna@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.6);
22
23         System.out.println("Customer created successfully!");
24
25         DB db = new DB();
26         db.createConnection();
27
28         db.createCustomer(customer);
29
30         db.closeConnection();
31     }
32 }
```

- Right pane:** Shows the MySQL terminal output and the Java application's console output.

**MySQL Terminal Output:**

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

mysql> select * from Customer;
+----+-----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | age | inDate
| outDateTime | temperature |
+----+-----+-----+-----+-----+-----+
| 1 | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 | 2022-01-08 11:38:57 | 98.6 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

**Java Application Console Output:**

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justmyjava/jdk-hotspot
System
values(null, 'Fionna', '+91 99999 11111', 'fionna@example.com', null, 23, null)
Status: true
```

### 3.7 Insert another record in the Customer table using the **set()** operation

```

1 package com.example.cms;
2
3* import com.example.cms.db.DB;[]
4
5 /**
6 * Hello world!
7 *
8 */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         customer.setName("John");
18         customer.setPhone("+91 99999 22222");
19         customer.setEmail("john@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.2f);
22
23         System.out.println("Connecting to DB....");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32

```

### 3.8 Run the code to get the output as **Customer Created Successfully**

```

1 package com.example.cms;
2
3* import com.example.cms.db.DB;[]
4
5 /**
6 * Hello world!
7 *
8 */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         customer.setName("John");
18         customer.setPhone("+91 99999 22222");
19         customer.setEmail("john@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.2f);
22
23         System.out.println("Connecting to DB....");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32

```

Output in Console:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jre.internal
[0] Welcome to Customer Management System
[0] Connecting to DB...
[0] [DB] Driver Loaded
[0] [DB] Connection Created
[0] SQL: insert into Customer values(null, 'John', '+91 99999 22222', 'john@example.com', 23, 98.2)
[0] [DB] Customer Created Successfully
[0] [DB] Connection Closed. Close Status: true

```

3.9 Return to the terminal and rerun the select command:

```
select * from Customer;
```

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor with Java code. On the right, there is a terminal window showing MySQL command-line interface output.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  */
8
9
10 public class App {
11
12     public static void main( String[] args ) {
13
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setName("John");
18         customer.setPhone("+91 99999 22222");
19         customer.setEmail("john@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.6);
22
23         System.out.println("Customer created successfully!");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32

```

MySQL terminal output:

```

erishant@gmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
| cid | name | phone | email | birthDate | age | inDate
| 1 | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 | 2022-01-08 11:38:57 |
| 2 | John | +91 99999 22222 | john@example.com | 2022-01-08 | 23 | 2022-01-08 11:39:52 |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from Customer;
Empty set (0.00 sec)
mysql> select * from Customer;
+----+-----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | age | inDate
| 1 | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 | 2022-01-08 11:38:57 |
| 2 | John | +91 99999 22222 | john@example.com | 2022-01-08 | 23 | 2022-01-08 11:39:52 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from Customer;

```

You will see another record is inserted in the table.

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor with Java code. On the right, there is a terminal window showing MySQL command-line interface output.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4
5 /**
6  * Hello world!
7  */
8
9
10 public class App {
11
12     public static void main( String[] args ) {
13
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setName("John");
18         customer.setPhone("+91 99999 22222");
19         customer.setEmail("john@example.com");
20         customer.setAge(23);
21         customer.setTemperature(98.6);
22
23         System.out.println("Customer created successfully!");
24         DB db = new DB();
25         db.createConnection();
26
27         db.createCustomer(customer);
28
29         db.closeConnection();
30     }
31 }
32

```

MySQL terminal output:

```

erishant@gmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
| cid | name | phone | email | birthDate | age | inDate
| 1 | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 | 2022-01-08 11:38:57 |
| 2 | John | +91 99999 22222 | john@example.com | 2022-01-08 | 23 | 2022-01-08 11:39:52 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

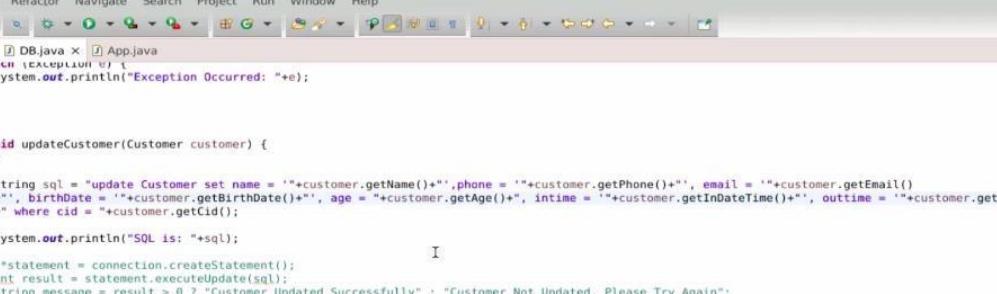
mysql> select * from Customer;
Empty set (0.00 sec)
mysql> select * from Customer;
+----+-----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | age | inDate
| 1 | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 | 2022-01-08 11:38:57 |
| 2 | John | +91 99999 22222 | john@example.com | 2022-01-08 | 23 | 2022-01-08 11:39:52 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from Customer;

```

## Step 4: Write an update operation

4.1 Write an update operation in the **try-catch** block



The screenshot shows the Eclipse IDE interface with the DAO.java file open in the editor. The code implements a CustomerDAO interface with methods for updating and deleting customers, and a getAllCustomers method. It uses JDBC to execute SQL statements and handle exceptions.

```
1 package com.example.cms.db;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 import com.example.cms.model.Customer;
11
12 public class DAO {
13
14     private Connection connection;
15
16     public DAO() {
17         // TODO Auto-generated constructor stub
18     }
19
20     public void updateCustomer(Customer customer) {
21         try {
22             String sql = "update Customer set name = '"+customer.getName()+"', phone = '"+customer.getPhone()+"', email = '"+customer.getEmail()+"', birthDate = '"+customer.getBirthDate()+"', age = '"+customer.getAge()+"', intime = '"+customer.getInDateTime()+"', outtime = '"+customer.getOutDateTime()+"' where cid = "+customer.getCid();
23
24             System.out.println("SQL is: "+sql);
25
26             PreparedStatement statement = connection.createStatement();
27             int result = statement.executeUpdate(sql);
28             String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
29             System.out.println(TAG+message);
30
31         } catch (Exception e) {
32             System.out.println("Exception Occurred: "+e);
33         }
34     }
35
36     public void deleteCustomer(int cid) {
37         // TODO Auto-generated method stub
38     }
39
40     public ArrayList<Customer> getAllCustomers() {
41         // TODO Auto-generated method stub
42         return null;
43     }
44 }
```

4.2 Return to the **App.java** file and set the **customer id**

```
5
6 /**
7  * Hello world!
8 *
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17         Integer cid;
18         customer.setCustomerId(2);
19         customer.setName("John");
20         customer.setPhone("+91 99999 22222");
21         customer.setEmail("john@example.com");
22         customer.setAge(23);
23         customer.setTemperature(98.2f);
24
25         System.out.println("Connecting to DB....");
26         DB db = new DB();
27         db.createConnection();
28
29         db.createCustomer(customer);
30
31         db.closeConnection();
32     }
33 }
34 }
```

#### 4.3 Update all the details of customers using the **set()** method

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App {
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.setId(2);
18         customer.setName("John Watson");
19         customer.setPhone("+91 98761 22222");
20         customer.setEmail("john.watson@example.com");
21         customer.setBirthDate("1990-08-08");
22         customer.setAge(32);
23         customer.setInDateTime("2022-01-08 10:39:52");
24         customer.setInDateTime("2022-01-08 11:45:22");
25         customer.setTemperature(98.5f);
26
27         System.out.println("Connecting to DB....");
28         DB db = new DB();
29         db.createConnection();
30
31         db.createCustomer(customer);
32
33         db.closeConnection();
34
35     }
36 }
37

```

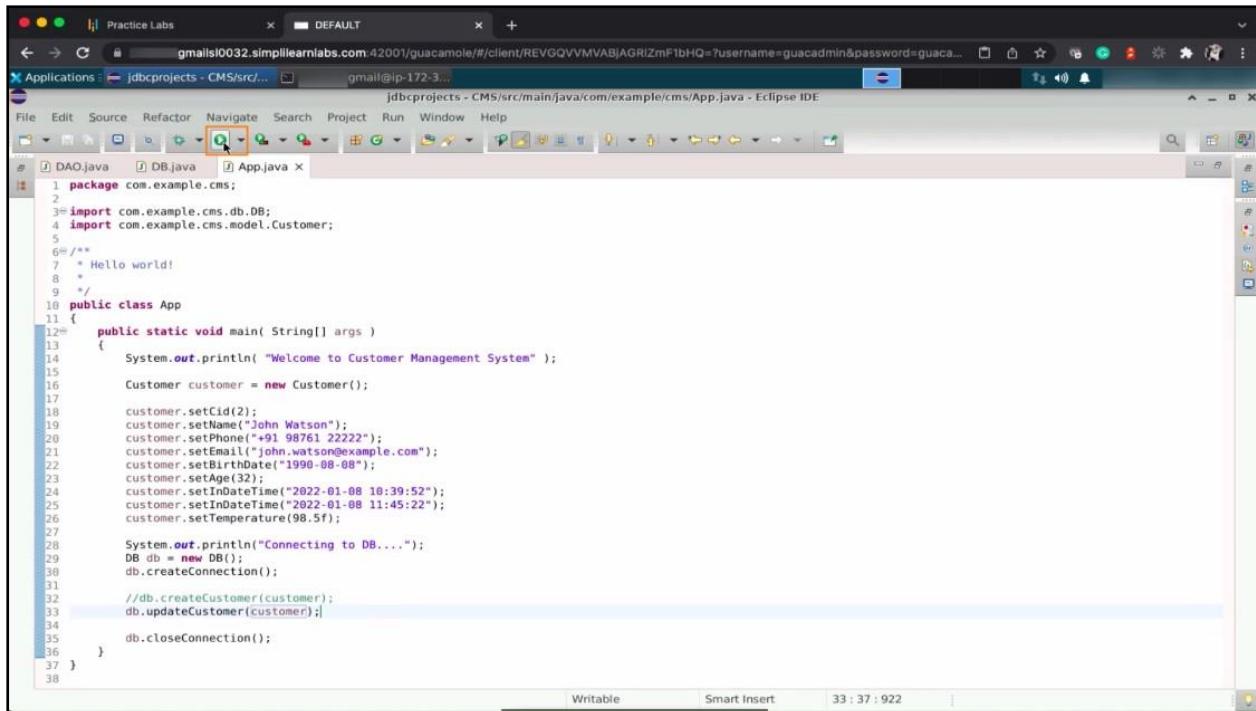
#### 4.4 Write the **db.updateCustomer()** method for updating details

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App {
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.setId(2);
18         customer.setName("John Watson");
19         customer.setPhone("+91 98761 22222");
20         customer.setEmail("john.watson@example.com");
21         customer.setBirthDate("1990-08-08");
22         customer.setAge(32);
23         customer.setInDateTime("2022-01-08 10:39:52");
24         customer.setInDateTime("2022-01-08 11:45:22");
25         customer.setTemperature(98.5f);
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.closeConnection();
35
36     }
37 }
38

```

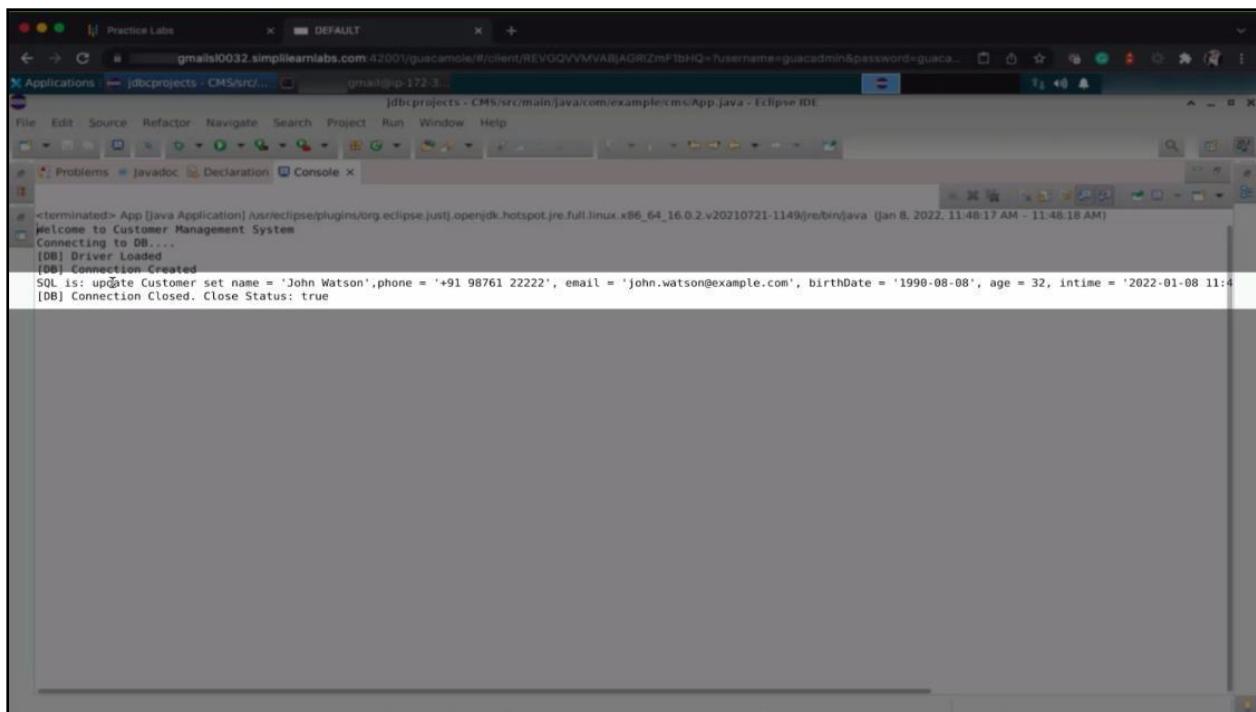
#### 4.5 Save and run the code



The screenshot shows the Eclipse IDE interface with the 'App.java' file open in the editor. The code implements a simple customer management system. It defines a Customer class and uses it to create a new customer object with specific details. This object is then passed to a DB class to perform database operations like connecting to the DB and updating the customer record.

```
1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17
18         customer.setCustomerId(2);
19         customer.setName("John Watson");
20         customer.setPhone("+91 98761 22222");
21         customer.setEmail("john.watson@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-08 10:39:52");
25         customer.setInDateTime("2022-01-08 11:45:22");
26         customer.setTemperature(98.5f);
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.closeConnection();
36     }
37 }
```

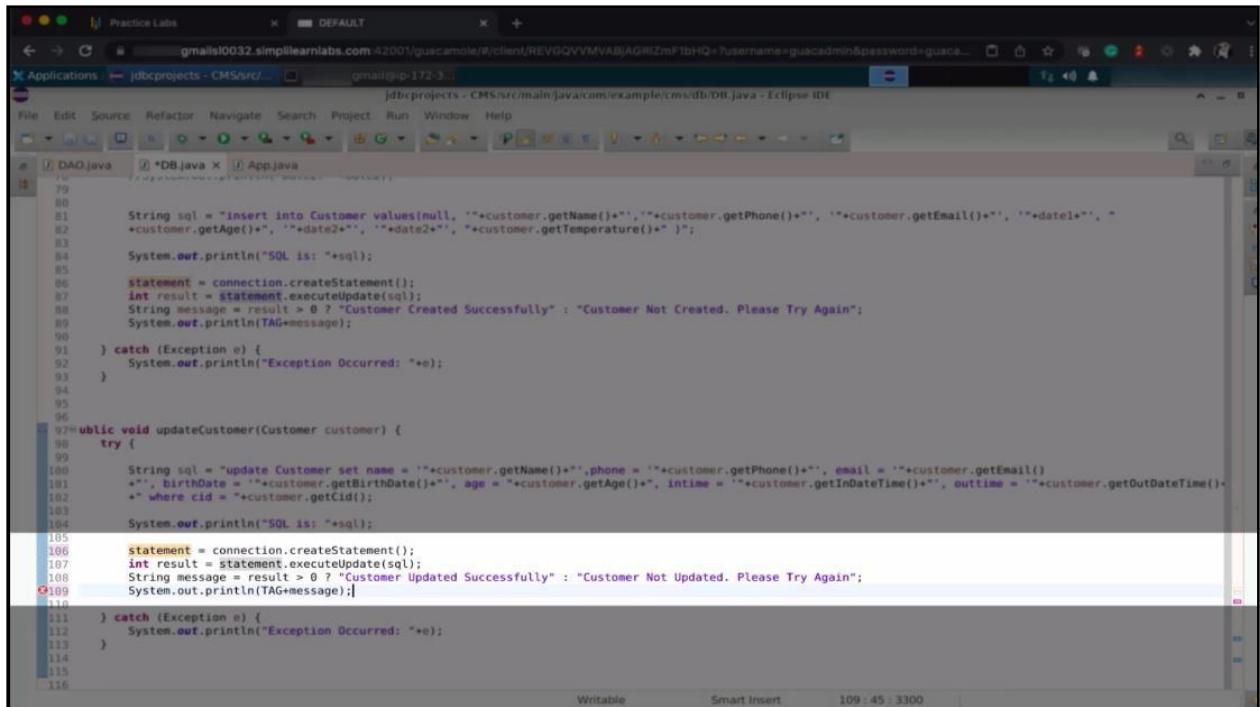
You will see that the customer details are updated.



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the results of running the application. It shows the application's welcome message, the connection to the database, and the successful update of the customer record. The SQL statement used for the update is also printed.

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.hotspot.jre.full.linux.x86_64_16.0.2.v20210721-1149/jre/bin/java [Jan 8, 2022, 11:48:17 AM - 11:48:18 AM]
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
SQL is: update Customer set name = 'John Watson', phone = '+91 98761 22222', email = 'john.watson@example.com', birthDate = '1990-08-08', age = 32, intime = '2022-01-08 11:45:22'
[DB] Connection Closed. Close Status: true
```

#### 4.6 Return to the DB.java file and uncomment the connection string statement

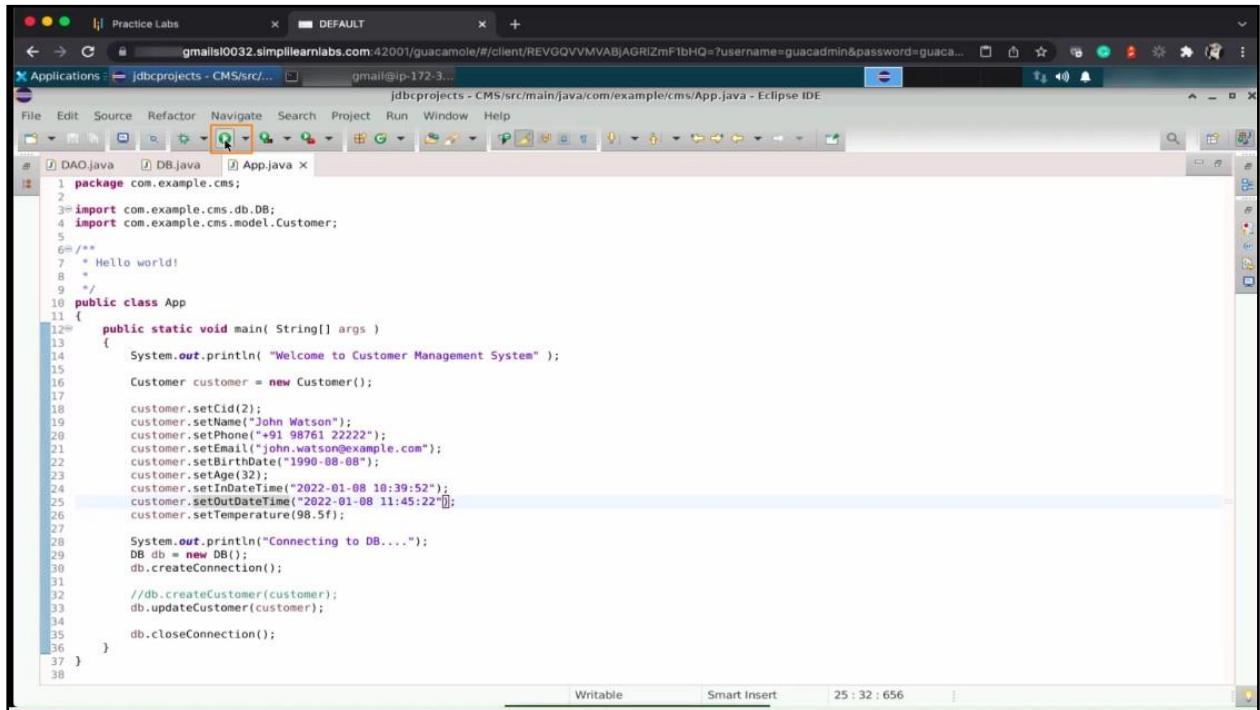


```

79
80     String sql = "insert into Customer values(null, '"+customer.getName()+"','"+customer.getPhone()+"', '"+customer.getEmail()+"', '"+date1+"', "
81     +" "+customer.getAge()+"', '"+date2+"', '"+date2+"', "+customer.getTemperature()+"')";
82
83     System.out.println("SQL is: "+sql);
84
85     statement = connection.createStatement();
86     int result = statement.executeUpdate(sql);
87     String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
88     System.out.println(TAG+message);
89
90 } catch (Exception e) {
91     System.out.println("Exception Occurred: "+e);
92 }
93
94
95
96
97 public void updateCustomer(Customer customer) {
98     try {
99
100         String sql = "update Customer set name = '"+customer.getName()+"', phone = '"+customer.getPhone()+"', email = '"+customer.getEmail()+"',
101         birthDate = '"+customer.getBirthDate()+"', age = "+customer.getAge()+"', intime = '"+customer.getInDateTime)+"', outtime = '"+customer.getOutDateTime()+"-
102         where cid = "+customer.getId();
103
104         System.out.println("SQL is: "+sql);
105
106         statement = connection.createStatement();
107         int result = statement.executeUpdate(sql);
108         String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
109         System.out.println(TAG+message);
110
111     } catch (Exception e) {
112         System.out.println("Exception Occurred: "+e);
113     }
114 }
115
116

```

#### 4.7 Save and run the code



```

1 package com.example.cms;
2
3 import com.example.cms.db;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17
18         customer.setId(2);
19         customer.setName("John Watson");
20         customer.setPhone("+91 98761 22222");
21         customer.setEmail("john.watson@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-08 10:39:52");
25         customer.setOutDateTime("2022-01-08 11:45:22");
26         customer.setTemperature(98.5f);
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.closeConnection();
36     }
37 }

```

After the update operation is completed successfully, the output **Customer Updated Successfully** will be shown.

4.8 Return to the terminal and run the **select** command:

```
select * from Customer
```

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

- Project Explorer:** Shows files DAO.java, DB.java, and App.java.
- Code Editor:** Displays Java code for creating a Customer object and connecting to a MySQL database to insert data.
- Terminal:** A terminal window titled "erishant@gmail@ip-172-31-17-157: ~" showing MySQL command-line output.
- Output:** Shows the results of the MySQL queries, including the creation of the Customer table and the insertion of a new customer record.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println("Connecting to DB....");
15
16         Customer customer = new Customer();
17
18         customer.setCid(2);
19         customer.setName("John Watson");
20         customer.setPhone("+91 98761 54452");
21         customer.setEmail("john.watson@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-08 10:39:52");
25         customer.setOutDateTime("2022-01-08 11:45:22");
26         customer.setTemperature(99.5f);
27
28         System.out.println("Customer created successfully!");
29         DB db = new DB();
30         db.createConnection();
31
32         db.createCustomer(customer);
33         //db.updateCustomer(customer);
34
35         db.closeConnection();
36     }
}

```

4.9 Insert one more customer detail by updating the details and running the code again in the **App.java** file

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

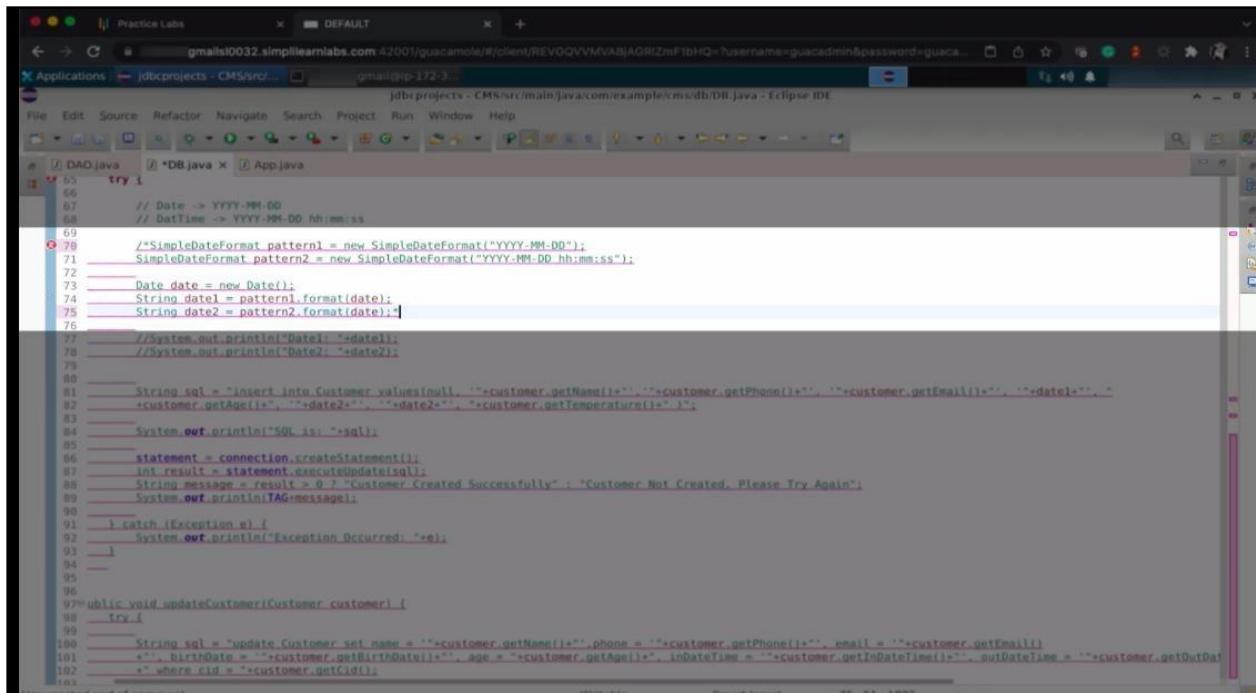
- Project Explorer:** Shows files DAO.java, DB.java, and App.java.
- Code Editor:** Displays Java code for creating a Customer object and connecting to a MySQL database to insert data.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7 * Hello world!
8 *
9 */
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println("Welcome to Customer Management System");
15
16         Customer customer = new Customer();
17
18         customer.setCid(2);
19         customer.setName("Leo");
20         customer.setPhone("+91 98761 54452");
21         customer.setEmail("leo@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-08 10:39:52");
25         customer.setOutDateTime("2022-01-08 11:45:22");
26         customer.setTemperature(99.5f);
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.closeConnection();
36     }
}

```

4.10 Return to the **DB.java** file and comment on the date format code to prevent potential errors in the future if the entered format doesn't match the expected format



The screenshot shows the Eclipse IDE interface with the DB.java file open in the editor. The code is as follows:

```
try {
    // Date -> YYYY-MM-DD
    // DateTime -> YYYY-MM-DD hh:mm:ss
    SimpleDateFormat pattern1 = new SimpleDateFormat("YYYY-MM-DD");
    SimpleDateFormat pattern2 = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");

    Date date = new Date();
    String date1 = pattern1.format(date);
    String date2 = pattern2.format(date);

    //System.out.println("Date1: "+date1);
    //System.out.println("Date2: "+date2);

    String sql = "insert into Customer values(null, '"+customer.getName()+"','"+customer.getPhone()+"', '"+customer.getEmail()+"', '"+date1+"', "+customer.getAge()+" , '"+date2+"', '"+date2+"', "+customer.getTemperature()+" )";
    System.out.println("SQL is: "+sql);

    statement = connection.createStatement();
    int result = statement.executeUpdate(sql);
    String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
    System.out.println(TAG+message);

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

}

public void updateCustomer(Customer customer) {
    try {
        String sql = "update Customer set name = '"+customer.getName()+"', phone = '"+customer.getPhone()+"', email = '"+customer.getEmail()+"', "+customer.getBirthDate()+"', age = "+customer.getAge()+" , inDateTime = '"+customer.getInDateTime()+"', outDateTime = '"+customer.getOutDate()+"' where cid = "+customer.getCid();
        System.out.println("SQL is: "+sql);

        statement = connection.createStatement();
        int result = statement.executeUpdate(sql);
        String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
        System.out.println(TAG+message);

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

}
```

4.11 Change the query as shown:

```

try {
    // Date -> YYYY-MM-DD
    // DateTime -> YYYY-MM-DD hh:mm:ss
    SimpleDateFormat pattern1 = new SimpleDateFormat("YYYY-MM-DD");
    SimpleDateFormat pattern2 = new SimpleDateFormat("YYYY-MM-DD hh:mm:ss");

    Date date = new Date();
    String date1 = pattern1.format(date);
    String date2 = pattern2.format(date);

    //System.out.println("Date1: "+date1);
    //System.out.println("Date2: "+date2);

    String sql = "insert into Customer values(null, '"+customer.getName()+"','"+customer.getPhone()+"', '"+customer.getEmail()+"', '"+customer.getBirthDate()+"",
    +customer.getAge()+"','"+customer.getInDateTime()+"','"+customer.getOutDateTime()+"','"+customer.getTemperature()+"')";

    System.out.println("SQL is: "+sql);

    statement = connection.createStatement();
    int result = statement.executeUpdate(sql);
    String message = result > 0 ? "Customer Created Successfully" : "Customer Not Created. Please Try Again";
    System.out.println(TAG+message);

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

public void updateCustomer(Customer customer) {
    try {

        String sql = "update Customer set name ='"+customer.getName()+"', phone ='"+customer.getPhone()+"', email = '"+customer.getEmail()+"',
        birthDate = '"+customer.getBirthDate()+"', age = '"+customer.getAge()+"', inDateTime = '"+customer.getInDateTime()+"', outDateTime = '"+customer.getOutDateTime()+"'
        where cid = '"+customer.getCid()';

        System.out.println("SQL is: "+sql);

        statement = connection.createStatement();
        int result = statement.executeUpdate(sql);
        String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
        System.out.println(TAG+message);

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

```

#### 4.12 Return to the App.java and change the setInDateTime and the setOutDateTime

```

package com.example.cms;

import com.example.cms.db.DB;
import com.example.cms.model.Customer;

/**
 * Hello world!
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Customer Management System" );

        Customer customer = new Customer();

        customer.setCid(2);
        customer.setName("Leo");
        customer.setPhone("+91 98761 54452");
        customer.setEmail("leo@example.com");
        customer.setBirthDate("1990-08-08");
        customer.setAge(32);
        customer.setInDateTime("2022-01-09 10:39:52");
        customer.setOutDateTime("2022-01-09 11:45:22");
        customer.setTemperature(99.5);

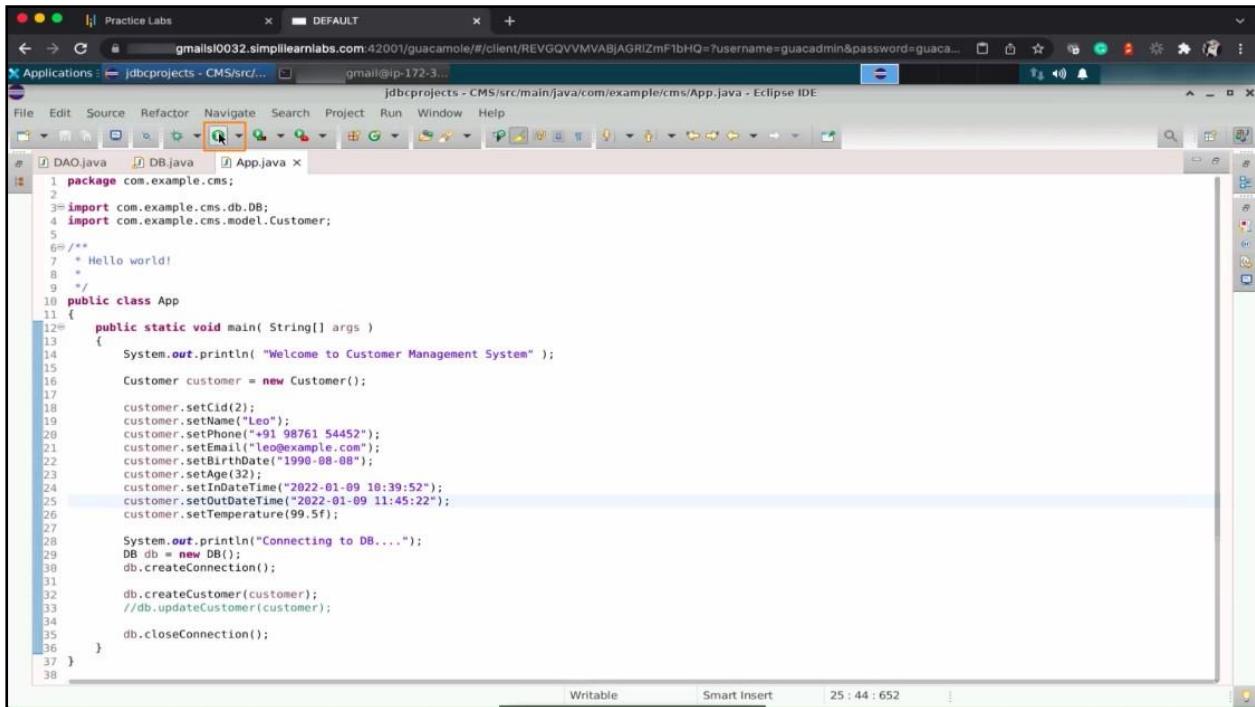
        System.out.println("Connecting to DB....");
        DB db = new DB();
        db.createConnection();

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

        db.closeConnection();
    }
}

```

#### 4.13 Save and run the code

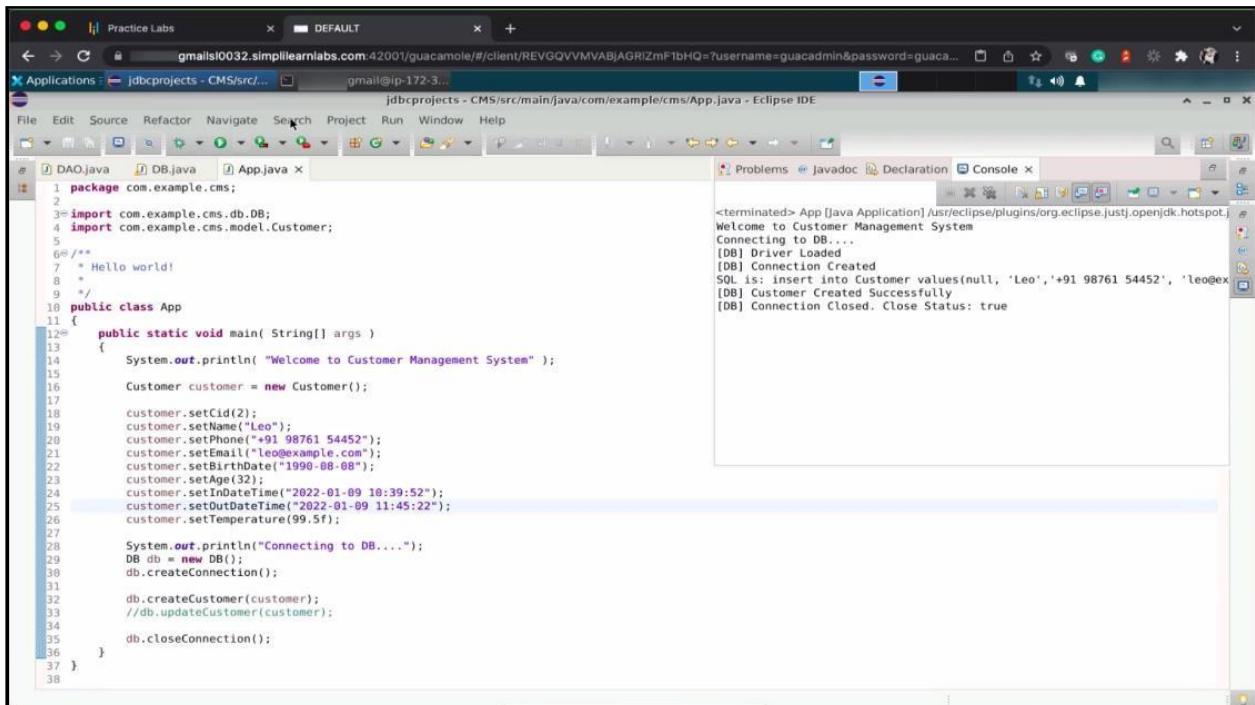


```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7  * Hello world!
8  */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17
18         customer.setCid(2);
19         customer.setName("Leo");
20         customer.setPhone("+91 98761 54452");
21         customer.setEmail("leo@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-09 10:39:52");
25         customer.setOutDateTime("2022-01-09 11:45:22");
26         customer.setTemperature(99.5f);
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.closeConnection();
36     }
37 }
38

```

You can see the output as **Customer Created Successfully.**



```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7  * Hello world!
8  */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Welcome to Customer Management System" );
15
16         Customer customer = new Customer();
17
18         customer.setCid(2);
19         customer.setName("Leo");
20         customer.setPhone("+91 98761 54452");
21         customer.setEmail("leo@example.com");
22         customer.setBirthDate("1990-08-08");
23         customer.setAge(32);
24         customer.setInDateTime("2022-01-09 10:39:52");
25         customer.setOutDateTime("2022-01-09 11:45:22");
26         customer.setTemperature(99.5f);
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.closeConnection();
36     }
37 }
38

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jvm.dll  
 Welcome to Customer Management System  
 Connecting to DB...  
 [DB] Driver Loader...  
 [DB] Connection Created  
 SQL is: insert into Customer values(null, 'Leo', '+91 98761 54452', 'leo@example.com', '1990-08-08', 32, '2022-01-09 10:39:52', '2022-01-09 11:45:22', 99.5)  
 [DB] Customer Created Successfully  
 [DB] Connection Closed. Close Status: true

#### 4.14 Go to the terminal and run the **select** command:

```
select * from Customers;
```

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

- Java Code:** The code in `App.java` creates a `Customer` object and prints its details to the console.
- MySQL Terminal:** The terminal window shows the execution of a `SELECT * FROM Customer` query. The results are displayed as a table with columns: cid, name, phone, email, birthDate, and temperature. Two rows are returned, corresponding to the customers created in the previous steps.
- Console Output:** The right-hand panel shows the output of the `System.out.println` statements and the MySQL query results.

```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7  * Hello world!
8 */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setCid(2);
18         customer.setName("Leo");
19         customer.setPhone("+91 98761 54452");
20         customer.setEmail("leo@example.com");
21         customer.setBirthDate("1990-08-08");
22         customer.setAge(32);
23         customer.setInDateTime("2022-01-08 11:38:57");
24         customer.setOutDateTime("2022-01-08 11:38:57");
25         customer.setTemperature(98.5);
26
27         System.out.println("Customer created!");
28
29         DB db = new DB();
30         db.createConnection();
31
32         db.createCustomer(customer);
33         //db.updateCustomer(customer);
34
35     }
36 }

```

You can see the updated data in the table:

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

- Java Code:** The code in `App.java` creates a `Customer` object and prints its details to the console.
- MySQL Terminal:** The terminal window shows the execution of a `SELECT * FROM Customer` query. The results are displayed as a table with columns: cid, name, phone, email, birthDate, and temperature. Three rows are returned, including the new entry for 'Leo' and the existing entries for 'Fionna' and 'John Watson'.
- Console Output:** The right-hand panel shows the output of the `System.out.println` statements and the MySQL query results.

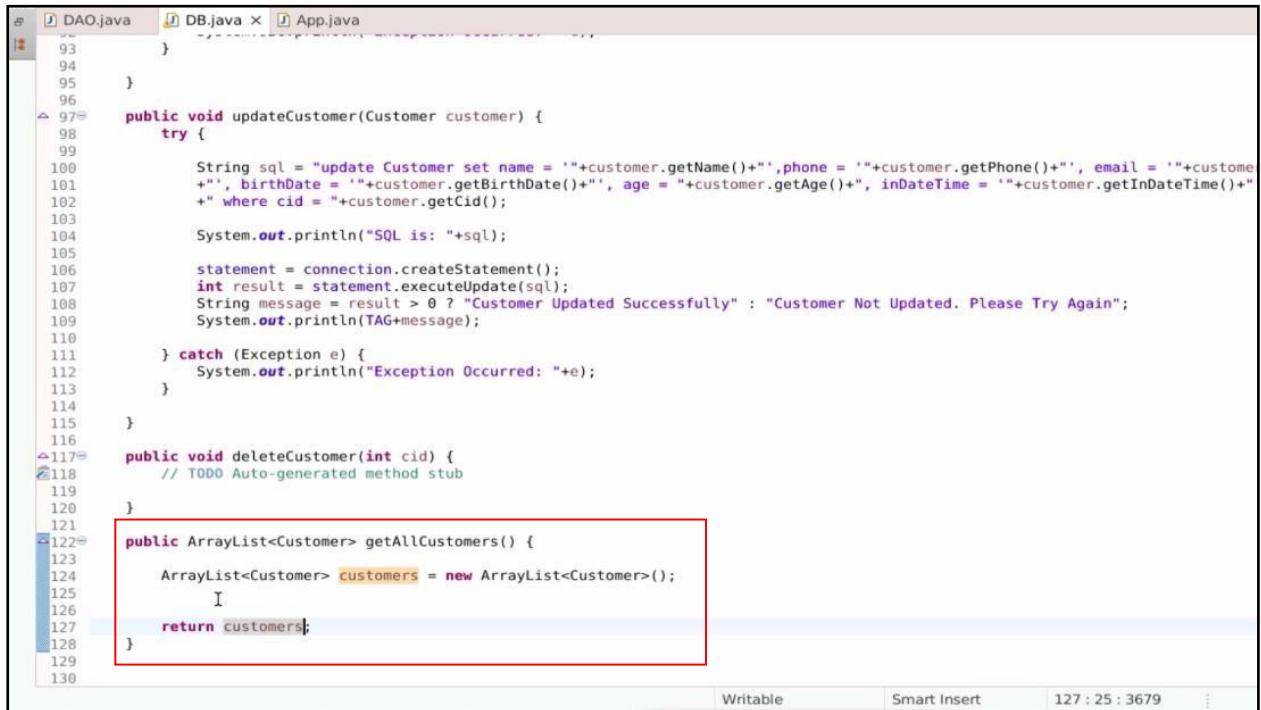
```

1 package com.example.cms;
2
3 import com.example.cms.db.DB;
4 import com.example.cms.model.Customer;
5
6 /**
7  * Hello world!
8 */
9
10 public class App
11 {
12     public static void main( String[] args )
13     {
14         System.out.println( "Hello World!" );
15
16         Customer customer = new Customer();
17         customer.setCid(2);
18         customer.setName("Leo");
19         customer.setPhone("+91 98761 54452");
20         customer.setEmail("leo@example.com");
21         customer.setBirthDate("1990-08-08");
22         customer.setAge(32);
23         customer.setInDateTime("2022-01-08 10:39:52");
24         customer.setOutDateTime("2022-01-08 11:45:22");
25         customer.setTemperature(98.5);
26
27         System.out.println("Customer created!");
28
29         DB db = new DB();
30         db.createConnection();
31
32         db.createCustomer(customer);
33         //db.updateCustomer(customer);
34
35     }
36 }

```

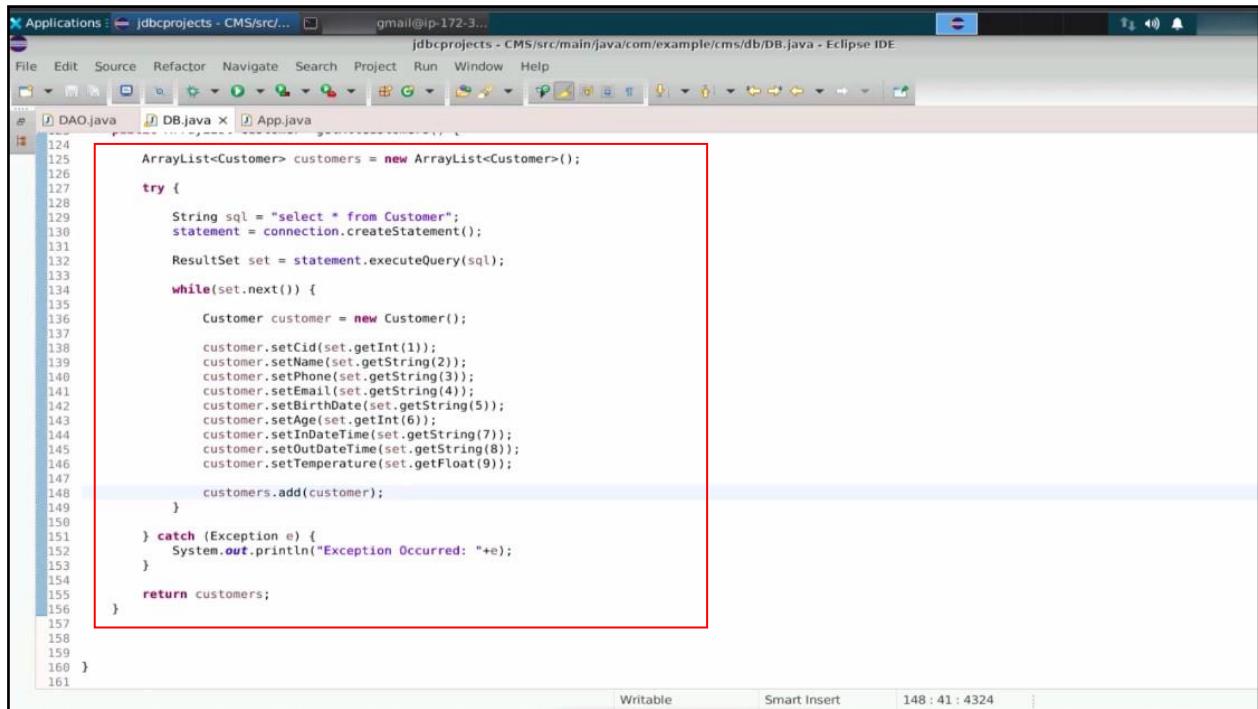
## Step 5: Perform the getallcustomer operations

### 5.1 Use the `getAllCustomers()` method to fetch details of all the customers



```
93     }
94 }
95
96
97 public void updateCustomer(Customer customer) {
98     try {
99
100         String sql = "update Customer set name = '"+customer.getName()+"', phone = '"+customer.getPhone()+"', email = '"+customer.getEmail()+"', birthDate = '"+customer.getBirthDate()+"', age = "+customer.getAge()+" , inDateTime = '"+customer.getInDateTime()+"' where cid = '"+customer.getCid()+"';
101
102         System.out.println("SQL is: "+sql);
103
104         statement = connection.createStatement();
105         int result = statement.executeUpdate(sql);
106         String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
107         System.out.println(TAG+message);
108
109     } catch (Exception e) {
110         System.out.println("Exception Occurred: "+e);
111     }
112 }
113
114
115
116
117 public void deleteCustomer(int cid) {
118     // TODO Auto-generated method stub
119 }
120
121
122 public ArrayList<Customer> getAllCustomers() {
123
124     ArrayList<Customer> customers = new ArrayList<Customer>();
125
126
127     return customers;
128 }
129
130
```

## 5.2 Enclose the select statement and connection string within a **try-catch block** to manage any potential exceptions



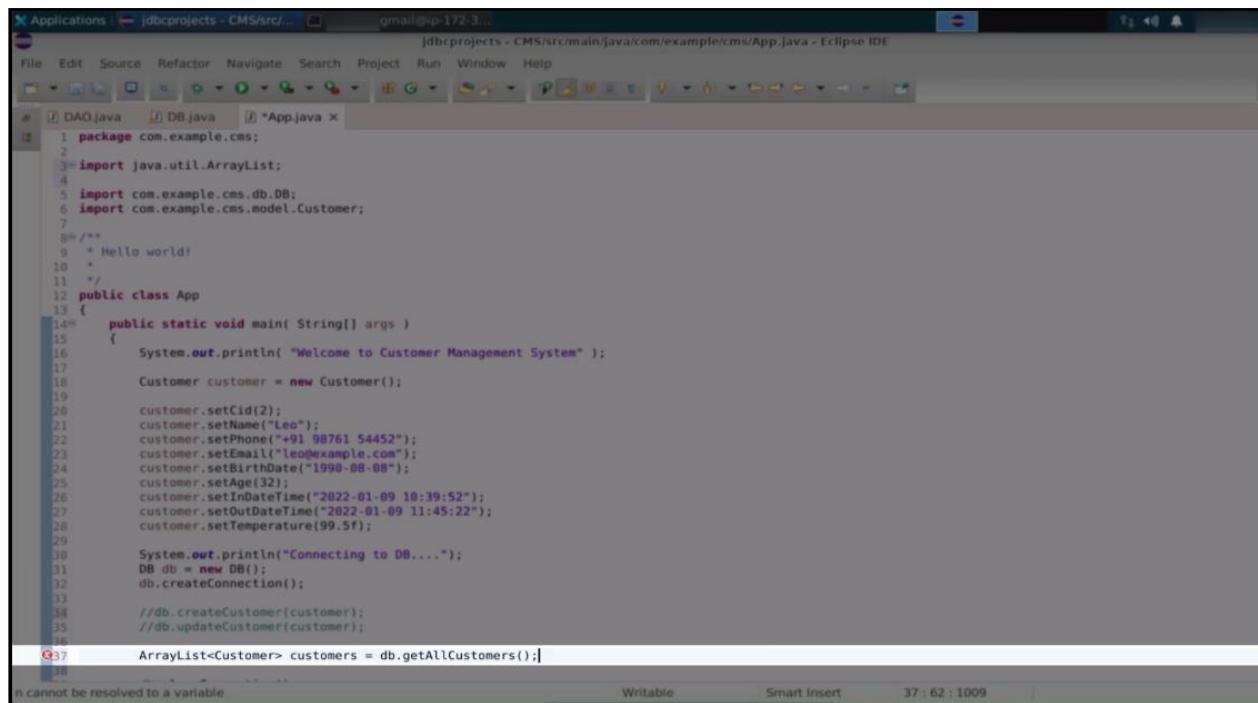
The screenshot shows a Java code editor in the Eclipse IDE. The code is located in a file named DB.java. A red rectangular box highlights the **try-catch** block used for exception handling. The code retrieves customer data from a database and adds it to a list.

```
ArrayList<Customer> customers = new ArrayList<Customer>();

try {
    String sql = "select * from Customer";
    statement = connection.createStatement();
    ResultSet set = statement.executeQuery(sql);
    while(set.next()) {
        Customer customer = new Customer();
        customer.setCustomerId(set.getInt(1));
        customer.setName(set.getString(2));
        customer.setPhone(set.getString(3));
        customer.setEmail(set.getString(4));
        customer.setBirthDate(set.getString(5));
        customer.setAge(set.getInt(6));
        customer.setInDate(new Date(set.getString(7)));
        customer.setOutDate(new Date(set.getString(8)));
        customer.setTemperature(set.getFloat(9));
        customers.add(customer);
    }
} catch (Exception e) {
    System.out.println("Exception Occurred: "+e);
}

return customers;
}
```

### 5.3 Return to the App.java file and call the getAllCustomers() method

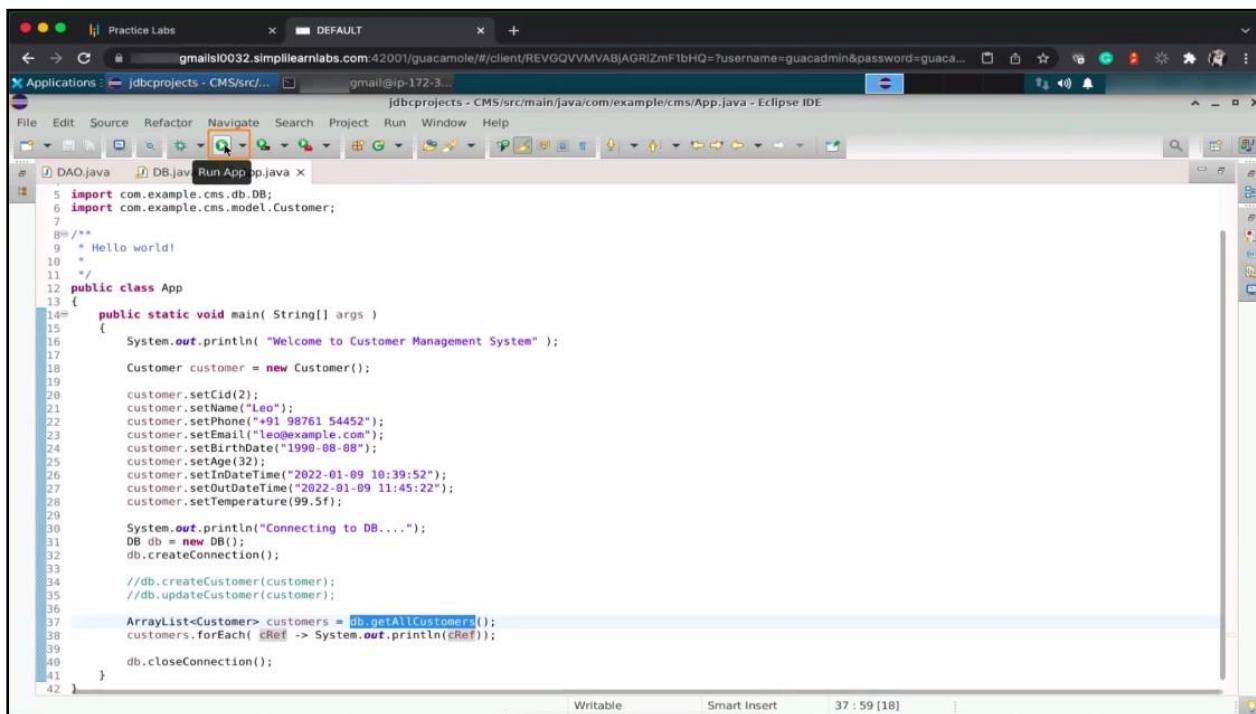


The screenshot shows the Eclipse IDE interface with the title bar "Applications - jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The left sidebar shows files DAO.java, DB.java, and App.java. The main editor window contains the following Java code:

```
1 package com.example.cms;
2
3 import java.util.ArrayList;
4
5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 *
11 */
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Welcome to Customer Management System" );
17
18         Customer customer = new Customer();
19
20         customer.setId(2);
21         customer.setName("Leo");
22         customer.setPhone("+91 98761 54452");
23         customer.setEmail("leode@example.com");
24         customer.setBirthDate("1990-08-08");
25         customer.setAge(32);
26         customer.setInDateTime("2022-01-09 10:39:52");
27         customer.setOutDateTime("2022-01-09 11:45:22");
28         customer.setTemperature(99.5f);
29
30         System.out.println("Connecting to DB....");
31         DB db = new DB();
32         db.createConnection();
33
34         //db.createCustomer(customer);
35         //db.updateCustomer(customer);
36
37     ArrayList<Customer> customers = db.getAllCustomers();
```

A tooltip "in Cannot be resolved to a variable" appears over the line "ArrayList<Customer> customers = db.getAllCustomers();". The status bar at the bottom shows "Writable", "Smart Insert", and "37 : 62 : 1009".

## 5.4 Save and run the code



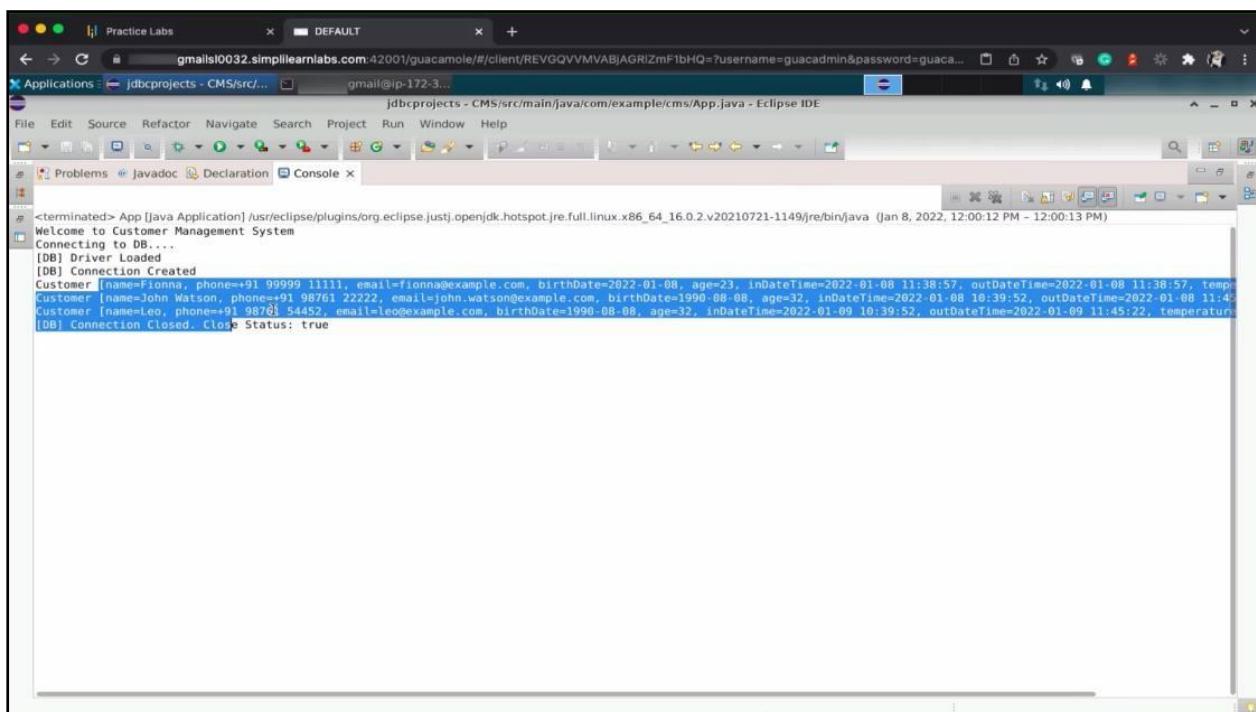
The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT
- Project Bar:** Applications - jdbcprojects - CMS/src/...
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Run, Stop, and Help.
- Code Editor:** Displays the `App.java` file content. The code initializes a customer object, sets its properties, and prints a welcome message. It then connects to a database, creates a customer record, retrieves all customers, and prints them. Finally, it closes the connection.

```
5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 */
11
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Welcome to Customer Management System" );
17
18         Customer customer = new Customer();
19
20         customer.setCustomerId(2);
21         customer.setName("Leo");
22         customer.setPhone("+91 98761 54452");
23         customer.setEmail("leo@example.com");
24         customer.setBirthDate("1990-08-08");
25         customer.setAge(32);
26         customer.setInDate("2022-01-09 10:39:52");
27         customer.setOutDate("2022-01-09 11:45:22");
28         customer.setTemperature(99.5f);
29
30         System.out.println("Connecting to DB....");
31         DB db = new DB();
32         db.createConnection();
33
34         //db.createCustomer(customer);
35         //db.updateCustomer(customer);
36
37         ArrayList<Customer> customers = db.getAllCustomers();
38         customers.forEach( cRef -> System.out.println(cRef));
39
40         db.closeConnection();
41     }
42 }
```

- Status Bar:** Writable, Smart Insert, 37 : 59 [18]

After running the code, the console will display the customer list as the output.



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

- Title Bar:** Practice Labs - DEFAULT
- Project Bar:** Applications - jdbcprojects - CMS/src/...
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Run, Stop, and Help.
- Console Tab:** Shows the output of the Java application. The output includes the welcome message, the connection log, and the list of customers.

```
* terminated-> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_16.0.2.v20210721-1149/jre/bin/java (Jan 8, 2022, 12:00:12 PM - 12:00:13 PM)
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
Customer [name=Fionna, phone=+91 99999 1111, email=fionna@example.com, birthDate=2022-01-08, age=23, inDateTime=2022-01-08 11:38:57, outDateTime=2022-01-08 11:38:57, temperature=99.5]
Customer [name=John Watson, phone=+91 98761 2222, email=john.watson@example.com, birthDate=1990-08-08, age=32, inDateTime=2022-01-08 10:39:52, outDateTime=2022-01-08 11:45:22]
Customer [name=Leo, phone=+91 98761 54452, email=leo@example.com, birthDate=1990-08-08, age=32, inDateTime=2022-01-09 10:39:52, outDateTime=2022-01-09 11:45:22, temperature=99.5]
[DB] Connection Closed. Close Status: true
```

## **Step 6: Perform the delete operation**

6.1 Type a delete operation inside the **try-catch** block, execute the **deleteCustomer()** method, and add a message to it

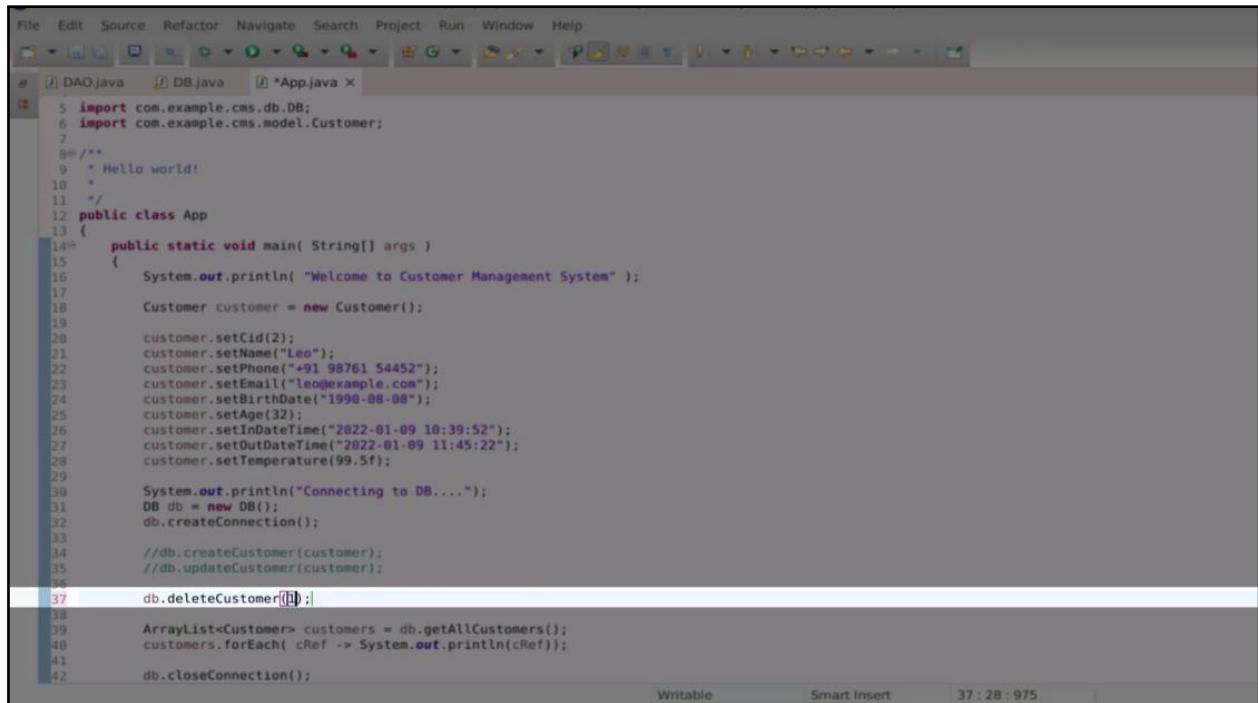


```
File Edit Source Refactor Navigate Search Project Run Window Help

DAO.java DB.java App.java

106
107     statement = connection.createStatement();
108     int result = statement.executeUpdate(sql);
109     String message = result > 0 ? "Customer Updated Successfully" : "Customer Not Updated. Please Try Again";
110     System.out.println(TAG+message);
111
112 } catch (Exception e) {
113     System.out.println("Exception Occurred: "+e);
114 }
115
116 }
117
118 public void deleteCustomer(int cid) {
119
120     try {
121
122         String sql = "delete from Customer where cid = "+cid;
123         statement = connection.createStatement();
124         int result = statement.executeUpdate(sql);
125
126         String message = result > 0 ? "Customer Deleted Successfully" : "Customer Not Deleted. Please Try Again";
127         System.out.println(TAG+message);
128
129     } catch (Exception e) {
130         System.out.println("Exception Occurred: "+e);
131     }
132
133 }
134
135 public ArrayList<Customer> getAllCustomers() {
136
137     ArrayList<Customer> customers = new ArrayList<Customer>();
138
139     try {
140
141         String sql = "select * from Customer";
142         statement = connection.createStatement();
143     }
```

6.2 Return to the **App.java** file and write **db.deleteCustomer()** to call the function with the object **db**



The screenshot shows a Java code editor with the file **App.java** open. The code is as follows:

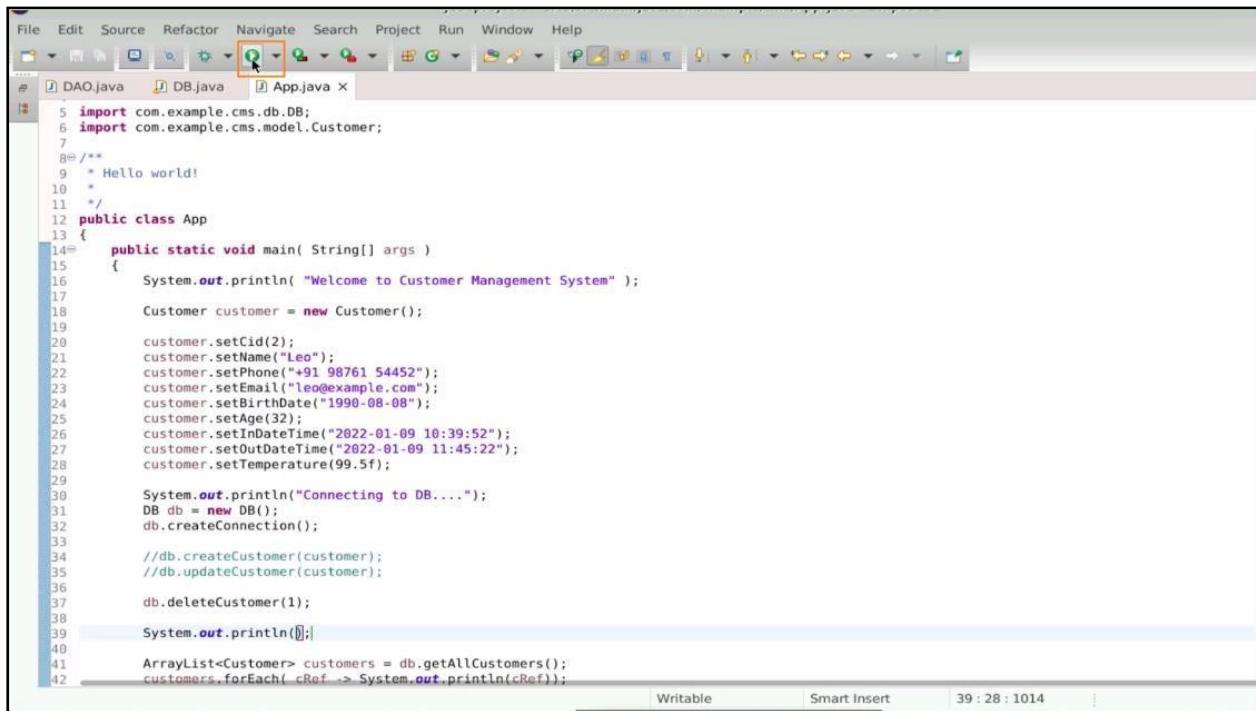
```
File Edit Source Refactor Navigate Search Project Run Window Help
DAO.java DB.java *App.java X

5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 */
11
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Welcome to Customer Management System" );
17
18         Customer customer = new Customer();
19
20         customer.setCid(2);
21         customer.setName("Leo");
22         customer.setPhone("+91 98761 54452");
23         customer.setEmail("leo@example.com");
24         customer.setBirthDate("1990-08-08");
25         customer.setAge(32);
26         customer.setInDateTime("2022-01-09 10:39:52");
27         customer.setOutDateTime("2022-01-09 11:45:22");
28         customer.setTemperature(99.5f);
29
30         System.out.println("Connecting to DB....");
31         DB db = new DB();
32         db.createConnection();
33
34         //db.createCustomer(customer);
35         //db.updateCustomer(customer);
36
37         db.deleteCustomer([b]);
38
39         ArrayList<Customer> customers = db.getAllCustomers();
40         customers.forEach( cRef -> System.out.println(cRef));
41
42         db.closeConnection();

```

The cursor is positioned at line 37, where the method **db.deleteCustomer()** is being typed. The IDE interface includes a toolbar, a menu bar, and status bars at the bottom indicating "Writable", "Smart Insert", and "37 : 28 : 975".

### 6.3 Save and run the code



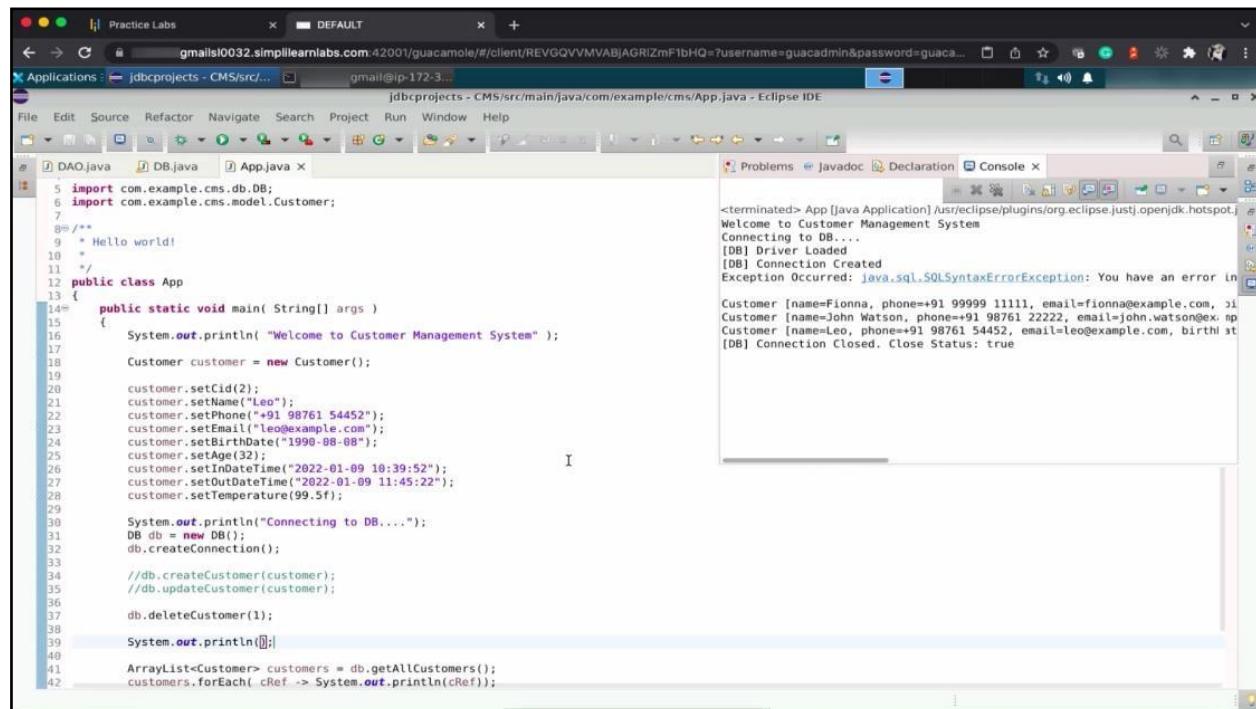
The screenshot shows the Eclipse IDE interface with the 'Run' button highlighted in orange. The code editor displays Java code for a Customer Management System. The code includes imports for DAO.java, DB.java, and App.java, and defines a main() method that prints a welcome message, creates a customer object, connects to a database, and prints the customer details.

```

5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 */
11
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Welcome to Customer Management System" );
17
18         Customer customer = new Customer();
19
20         customer.setCid(2);
21         customer.setName("Leo");
22         customer.setPhone("+91 98761 54452");
23         customer.setEmail("leo@example.com");
24         customer.setBirthDate("1990-08-08");
25         customer.setAge(32);
26         customer.setInDateTime("2022-01-09 10:39:52");
27         customer.setOutDateTime("2022-01-09 11:45:22");
28         customer.setTemperature(99.5f);
29
30         System.out.println("Connecting to DB....");
31         DB db = new DB();
32         db.createConnection();
33
34         //db.createCustomer(customer);
35         //db.updateCustomer(customer);
36
37         db.deleteCustomer(1);
38
39         System.out.println();
40
41         ArrayList<Customer> customers = db.getAllCustomers();
42         customers.forEach( cRef -> System.out.println(cRef));
    }
}

```

In the output, you will encounter an error indicated by the message **Exception occurred**.



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the execution of the Java application. It shows the connection to the database, creation of a customer record, and deletion of the record. However, it ends with an `java.sql.SQLSyntaxErrorException` indicating an error in the database connection.

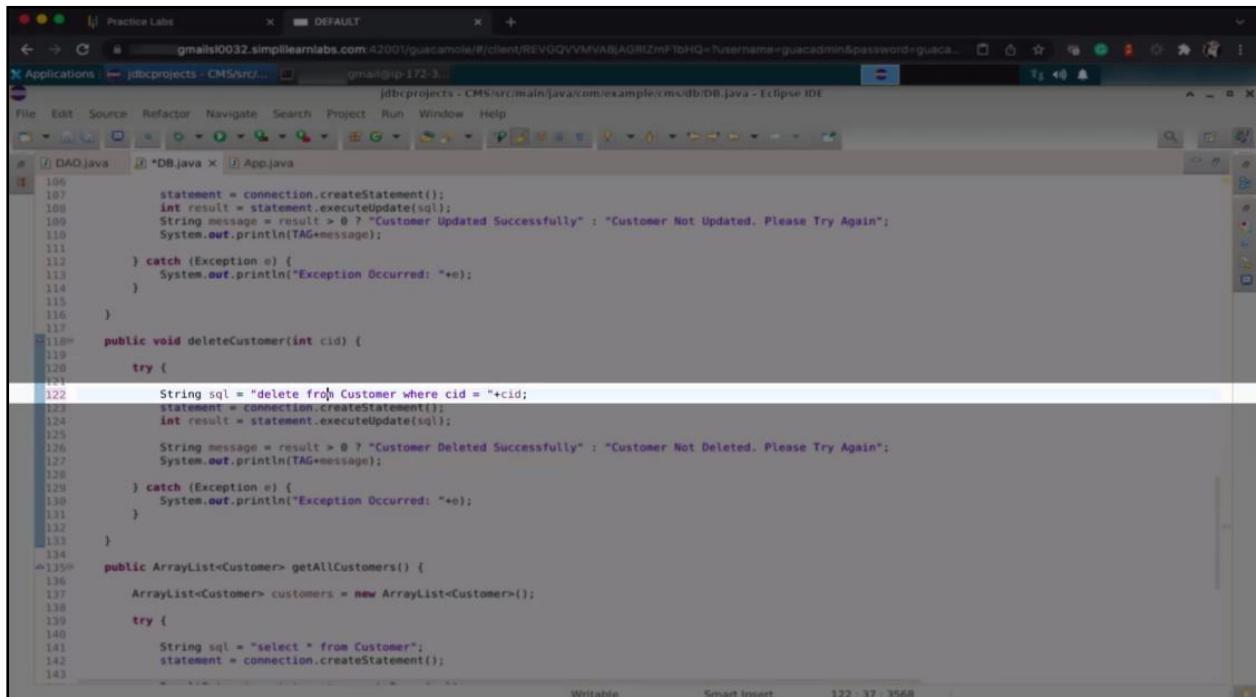
```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj/openjdk.hotspot
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
Exception Occurred: java.sql.SQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Customer [name=Fionna, phone=+91 99999 11111, email=fionna@example.com, id=1]' at line 1
Customer [name=John Watson, phone=+91 98761 22222, email=john.watson@example.com, birthdate=1990-08-08, age=32, inDateTime='2022-01-09 10:39:52', outDateTime='2022-01-09 11:45:22', temperature=99.5]
[DB] Connection Closed. Close Status: true

```

6.4 Update the following line of code to correct the delete query that caused the error:

```
String sql= "delete from Customer where cid = " +cid;
```

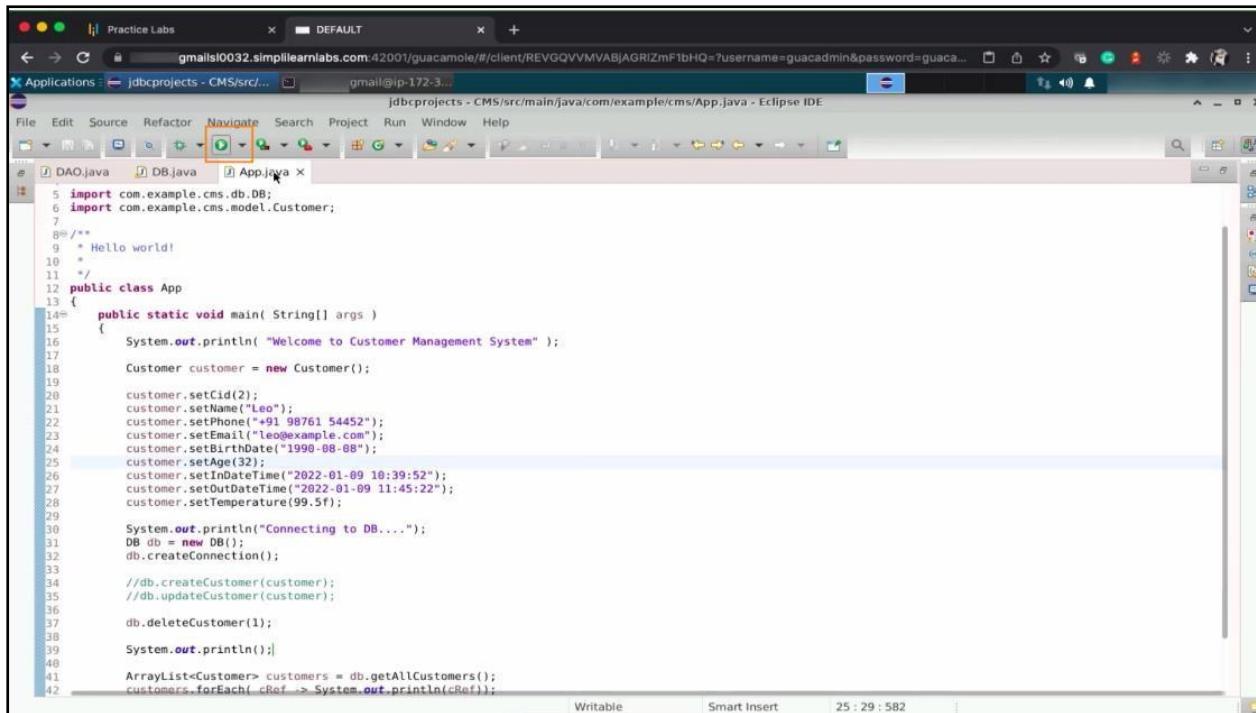


The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT - jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
- Toolbar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Left Sidebar:** Shows DAO.java, \*DB.java (selected), and App.java.
- Code Editor:** Displays the content of DB.java. The line causing the error is highlighted in blue:

```
String sql= "delete from Customer where cid = " +cid;
```
- Bottom Status Bar:** Shows symbols for available, saved, and modified status, along with line numbers 122, 123, 124, 125.

## 6.5 Return to the App.java file and run the code

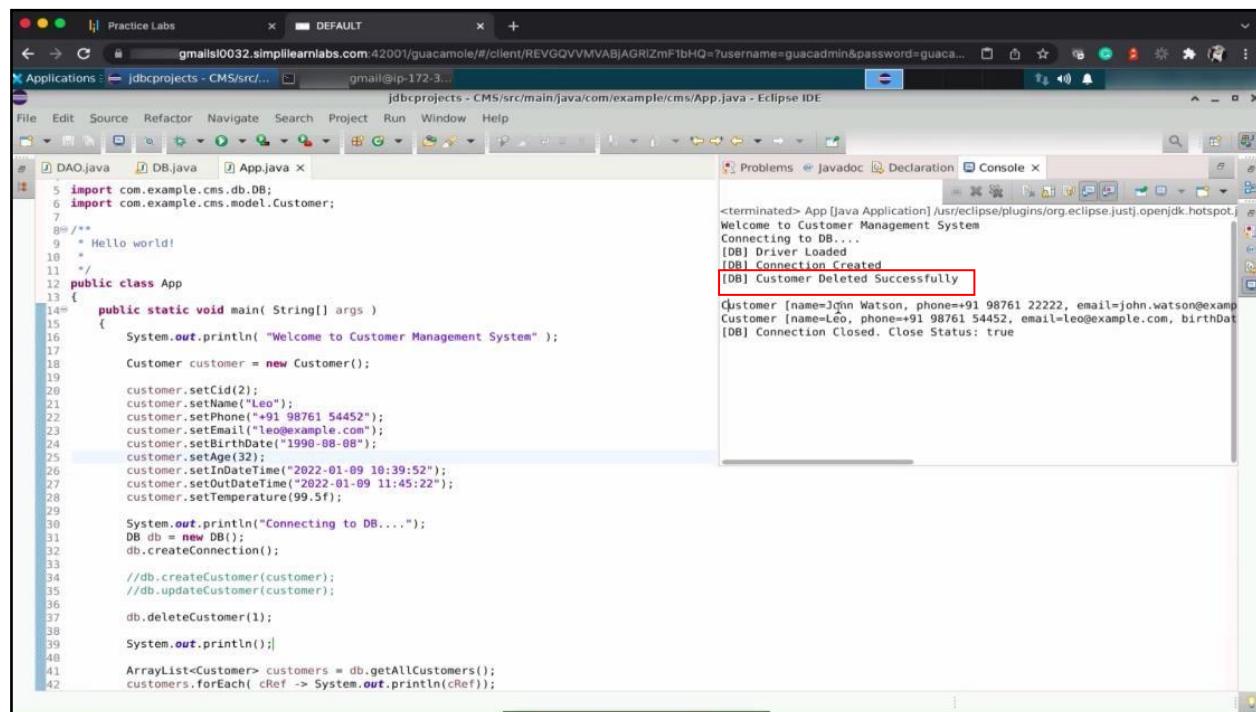


The screenshot shows the Eclipse IDE interface with the 'App.java' file open in the editor. The 'Run' button in the toolbar is highlighted with an orange box. The code in the editor is as follows:

```

5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 */
11 */
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Welcome to Customer Management System" );
17
18         Customer customer = new Customer();
19
20         customer.setId(2);
21         customer.setName("Leo");
22         customer.setPhone("+91 98761 54452");
23         customer.setEmail("leo@example.com");
24         customer.setBirthDate("1990-08-08");
25         customer.setAge(32);
26         customer.setInDateTime("2022-01-09 10:39:52");
27         customer.setOutDateTime("2022-01-09 11:45:22");
28         customer.setTemperature(99.5f);
29
30         System.out.println("Connecting to DB....");
31         DB db = new DB();
32         db.createConnection();
33
34         //db.createCustomer(customer);
35         //db.updateCustomer(customer);
36
37         db.deleteCustomer(1);
38
39         System.out.println();
40
41         ArrayList<Customer> customers = db.getAllCustomers();
42         customers.forEach( cRef -> System.out.println(cRef));
    
```

By following these steps, you will successfully deleted the customer, as indicated by the message **Customer Deleted Successfully**.



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following messages:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.just/jdk.hotspot.j
Welcome to Customer Management System
Connecting to DB...
[DB] Driver Loaded
[DB] Connection Created
[DB] Customer Deleted Successfully
Customer [name=John Watson, phone=<+91 98761 22222, email=john.watson@example.com]
Customer [name=Leo, phone=<+91 98761 54452, email=leo@example.com, birthDate=1990-08-08]
[DB] Connection Closed. Close Status: true
    
```

6.6 Return to the terminal and run the select command:

**select \* from Customers;**

The screenshot shows the Eclipse IDE interface with several tabs open: DAO.java, DB.java, and App.java. The App.java tab contains Java code for interacting with a MySQL database. The code includes creating a Customer object, setting its attributes, and printing it to System.out. It then creates a DB object, connects to the database, and performs a SELECT query on the Customer table. The MySQL terminal output shows the results of the query, listing three customers with their details: name, phone, email, birthDate, and temperature.

```
import com.example.cms.db;
import com.example.cms.model.Customer;

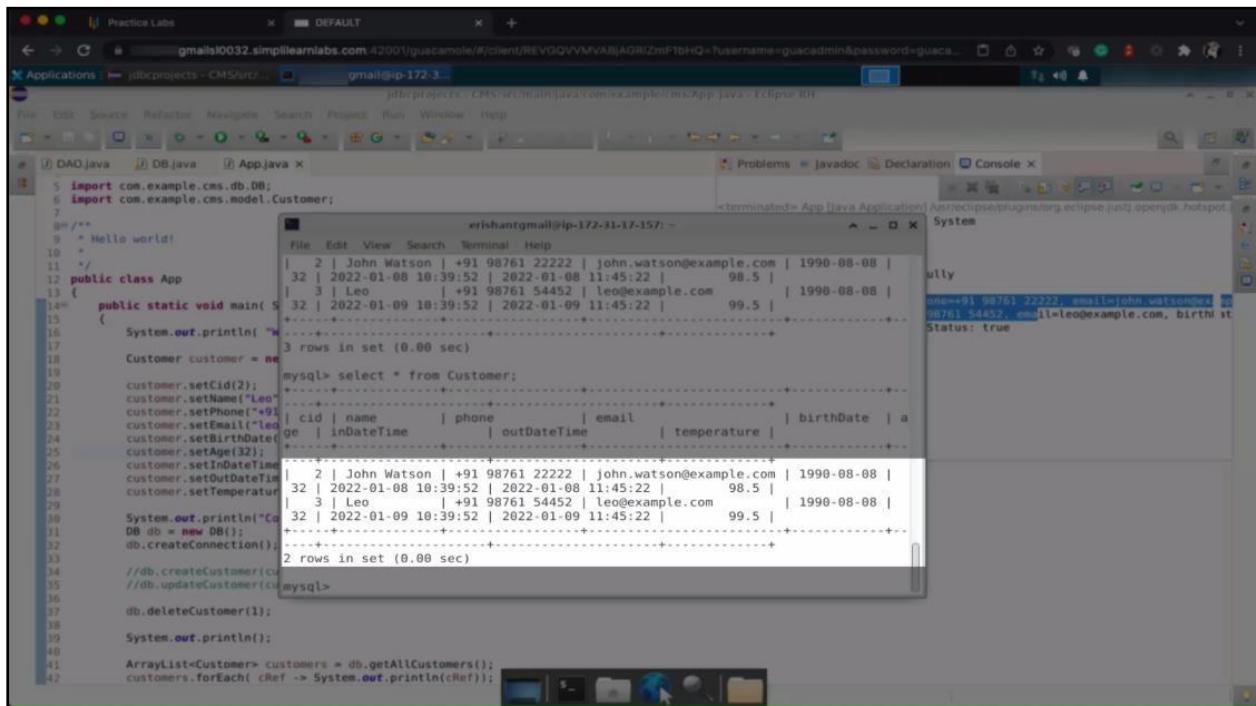
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello world!" );
        Customer customer = new Customer();
        customer.setCid(2);
        customer.setName("Leo");
        customer.setPhone("+91 98761 54452");
        customer.setEmail("leo@example.com");
        customer.setBirthDate("1990-08-08");
        customer.setAge(32);
        customer.setInDateTime("2022-01-08 10:39:52");
        customer.setOutDateTime("2022-01-08 11:45:22");
        customer.setTemperature(98.5);

        System.out.println("Customer created successfully!");
        DB db = new DB();
        db.createConnection();

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

```
mysql> select * from Customer;
+----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | age |
+----+-----+-----+-----+-----+
| 1   | Fionna | +91 99999 11111 | fionna@example.com | 2022-01-08 | 23 |
| 2   | John Watson | +91 98761 22222 | john.watson@example.com | 1990-08-08 | 32 |
| 3   | Leo | +91 98761 54452 | leo@example.com | 1990-08-08 | 32 |
+----+-----+-----+-----+-----+
```

You will notice that the customer data has been deleted.



```

5 import com.example.cms.db.DB;
6 import com.example.cms.model.Customer;
7
8 /**
9  * Hello world!
10 */
11
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Hello World!" );
17
18         Customer customer = new Customer();
19         customer.setCustomerId(2);
20         customer.setName("Leo");
21         customer.setPhone("+91 98761 54452");
22         customer.setEmail("leo@example.com");
23         customer.setBirthDate("1990-08-08");
24         customer.setAge(32);
25
26         customer.setInDateTime("2022-01-08 10:39:52");
27         customer.setOutDateTime("2022-01-09 11:45:22");
28         customer.setTemperature(99.5);
29
30         System.out.println("Customer created successfully!");
31
32         DB db = new DB();
33         db.createConnection();
34
35         //db.createCustomer(customer);
36         //db.updateCustomer(customer);
37
38         db.deleteCustomer(1);
39
40         System.out.println();
41
42         ArrayList<Customer> customers = db.getAllCustomers();
43         customers.forEach(cRef -> System.out.println(cRef));
44
45     }
46
47 }

```

The terminal output shows the following data:

```

mysql> select * from Customer;
+----+-----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | temperature |
+----+-----+-----+-----+-----+-----+
| 2   | John Watson | +91 98761 22222 | john.watson@example.com | 1990-08-08 | 98.5 |
| 3   | Leo          | +91 98761 54452 | leo@example.com      | 1990-08-08 | 99.5 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from Customer;
+----+-----+-----+-----+-----+-----+
| cid | name | phone | email | birthDate | temperature |
+----+-----+-----+-----+-----+-----+
| 2   | John Watson | +91 98761 22222 | john.watson@example.com | 1990-08-08 | 98.5 |
| 3   | Leo          | +91 98761 54452 | leo@example.com      | 1990-08-08 | 99.5 |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

By following these steps, you have successfully demonstrated how CRUD (create, read, update, delete) operations can be performed in a Java project, reflecting changes in the database. We executed an insert query to add a new customer, an update query to modify existing customer details, the **getAllCustomers** operation to retrieve all customer records, and the **deleteCustomer()** function to remove a customer's record.