Lesson 01 Demo 04

Writing CRUD Operations with PreparedStatement API

Objective: To implement PreparedStatment API with JDBC for writing CRUD operations for managing database records efficiently and securely

Tool required: Eclipse IDE

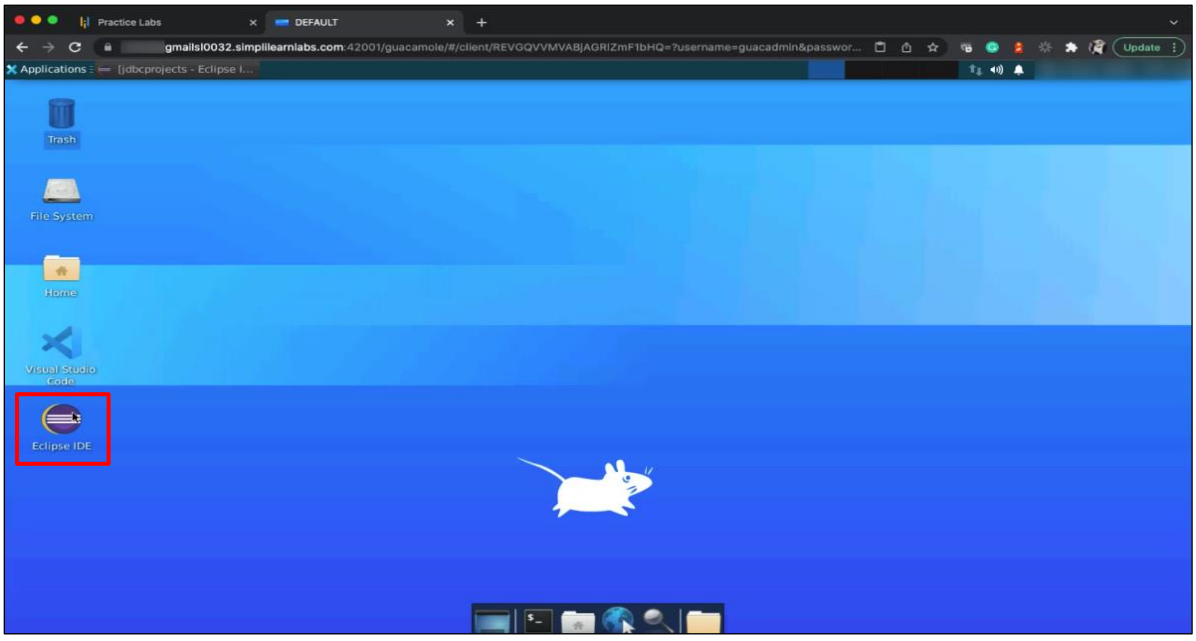
Prerequisites: Lesson 01 Demo 01

Steps to be followed:

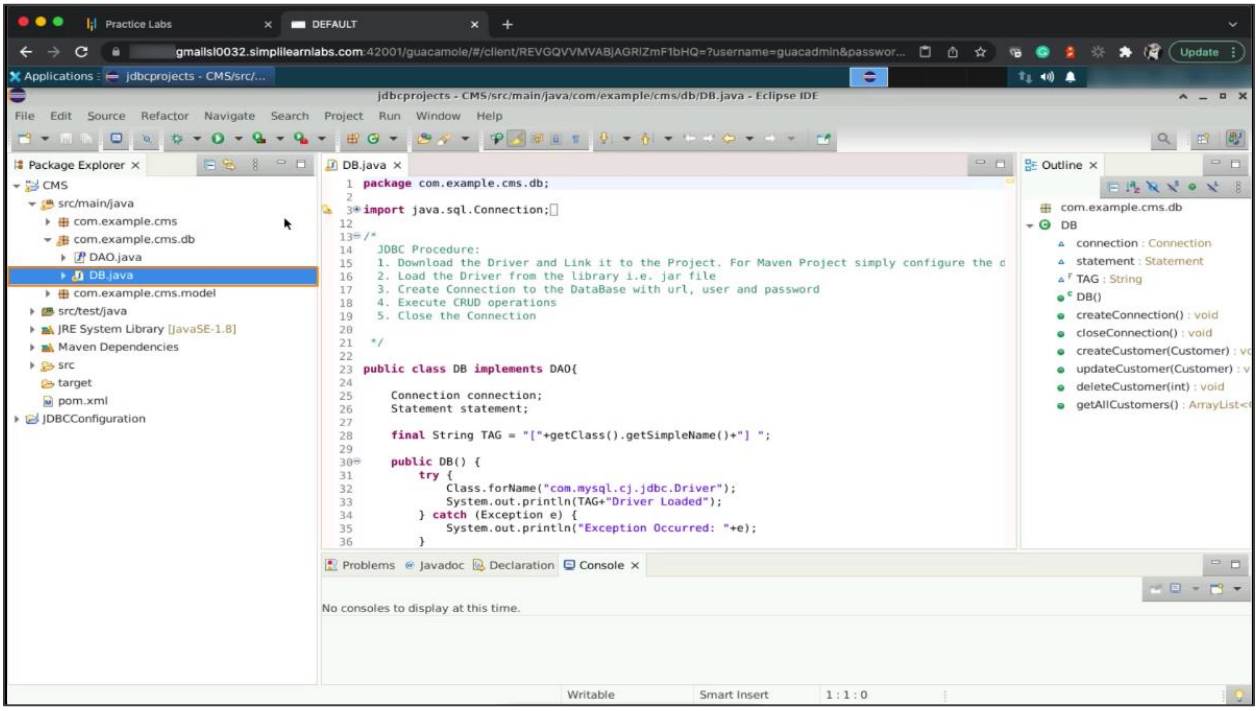
1. Perform the Create operation
2. Perform the Update operation
3. Perform the Delete operation
4. Perform the getAllCustomer operation

Step 1: Perform the Create operation

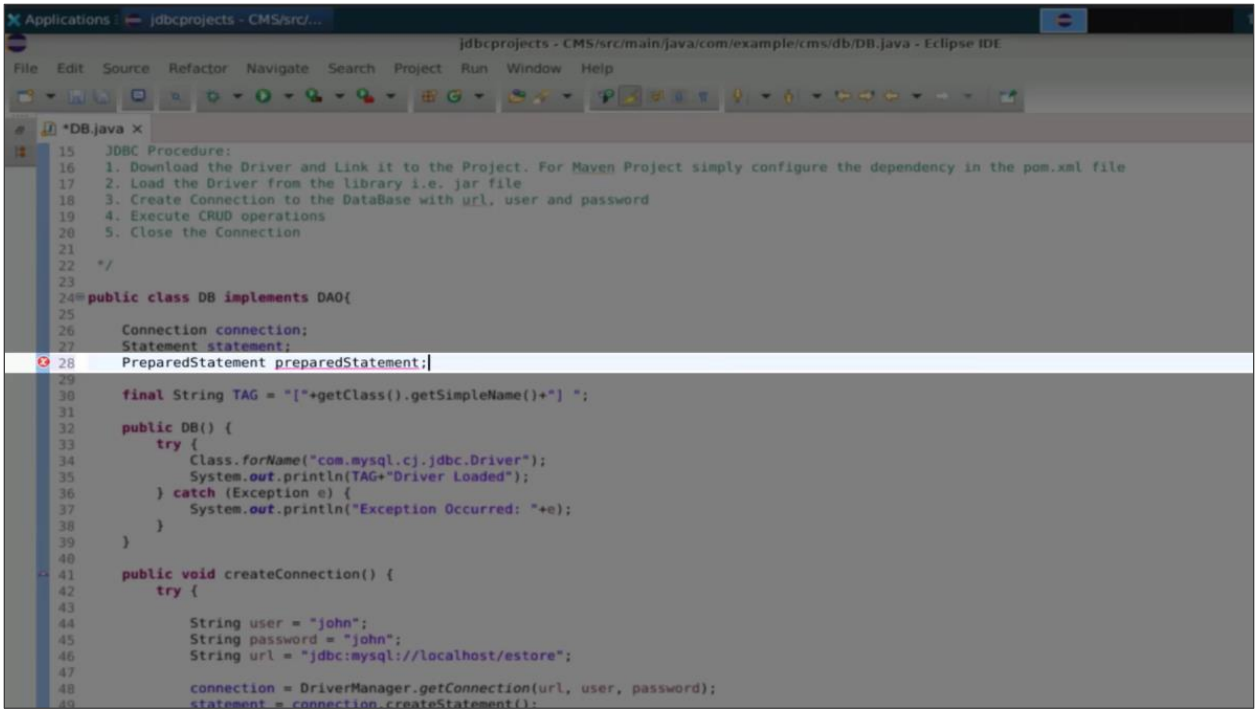
1.1 Open **Eclipse IDE**



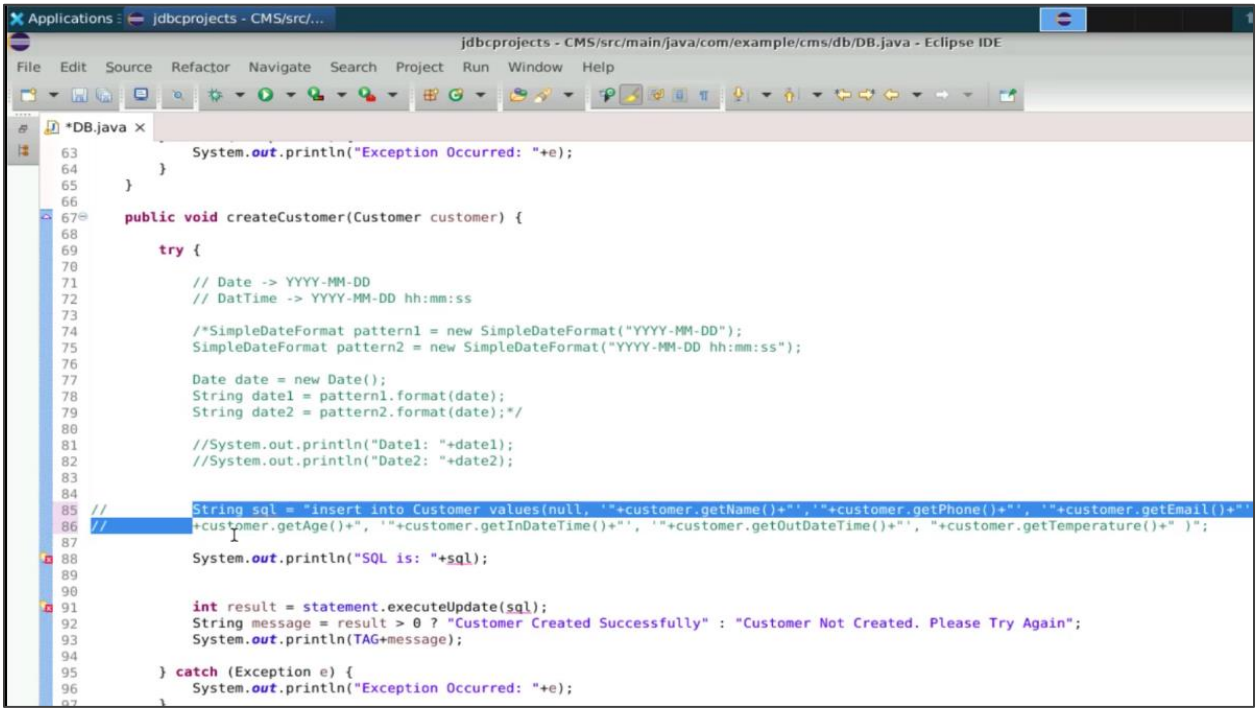
1.2 Open the **DB.java** file in the **CMS** project folder



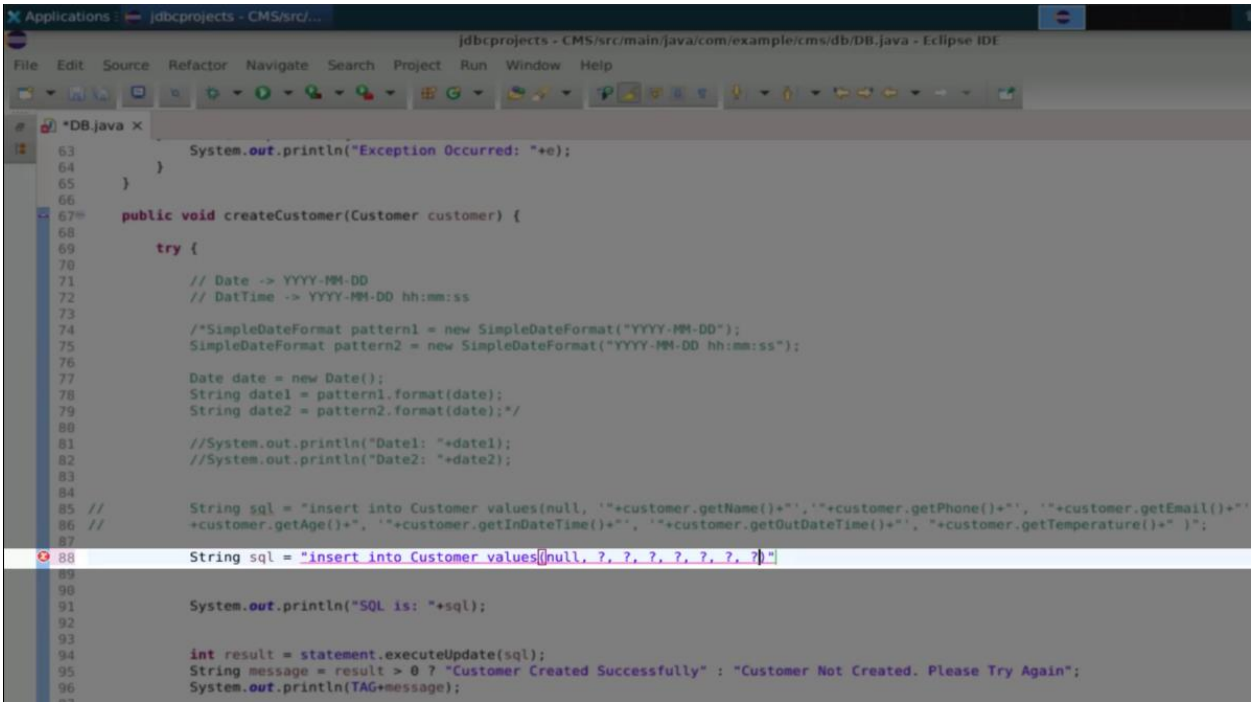
1.3 Write another API for **PreparedStatment** with a reference variable **preparedStatement;**



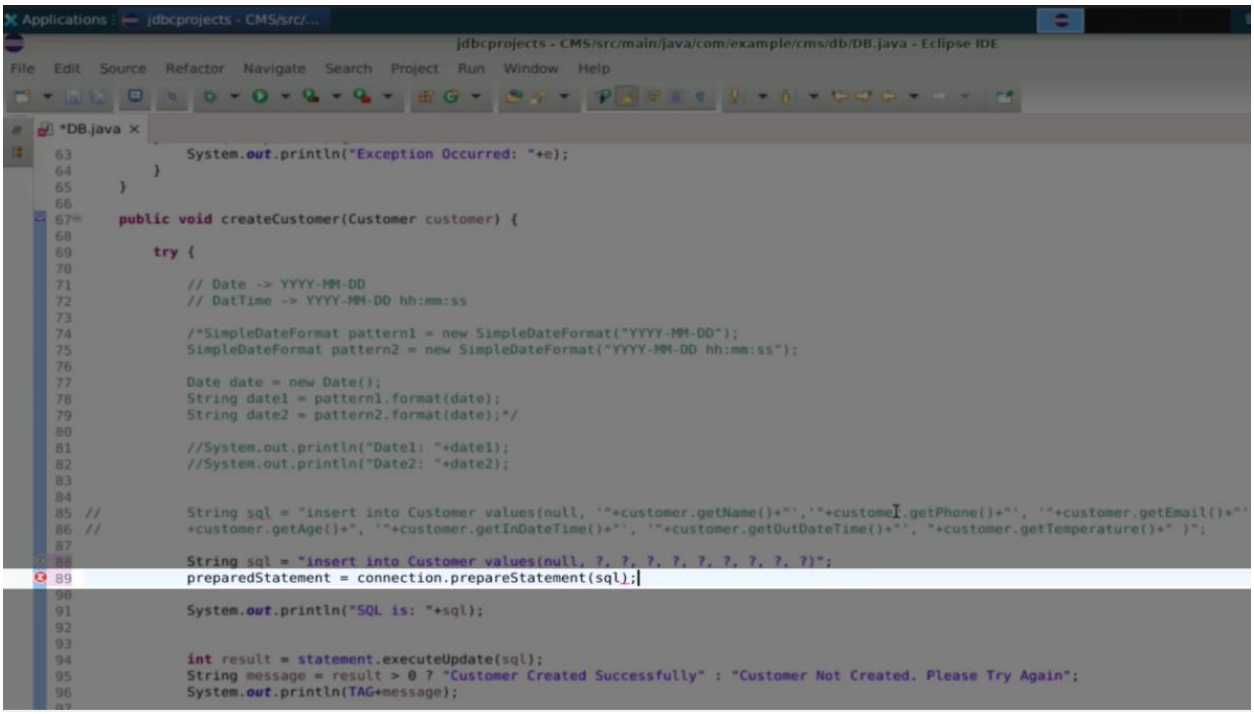
1.4 Comment out the previous SQL statement



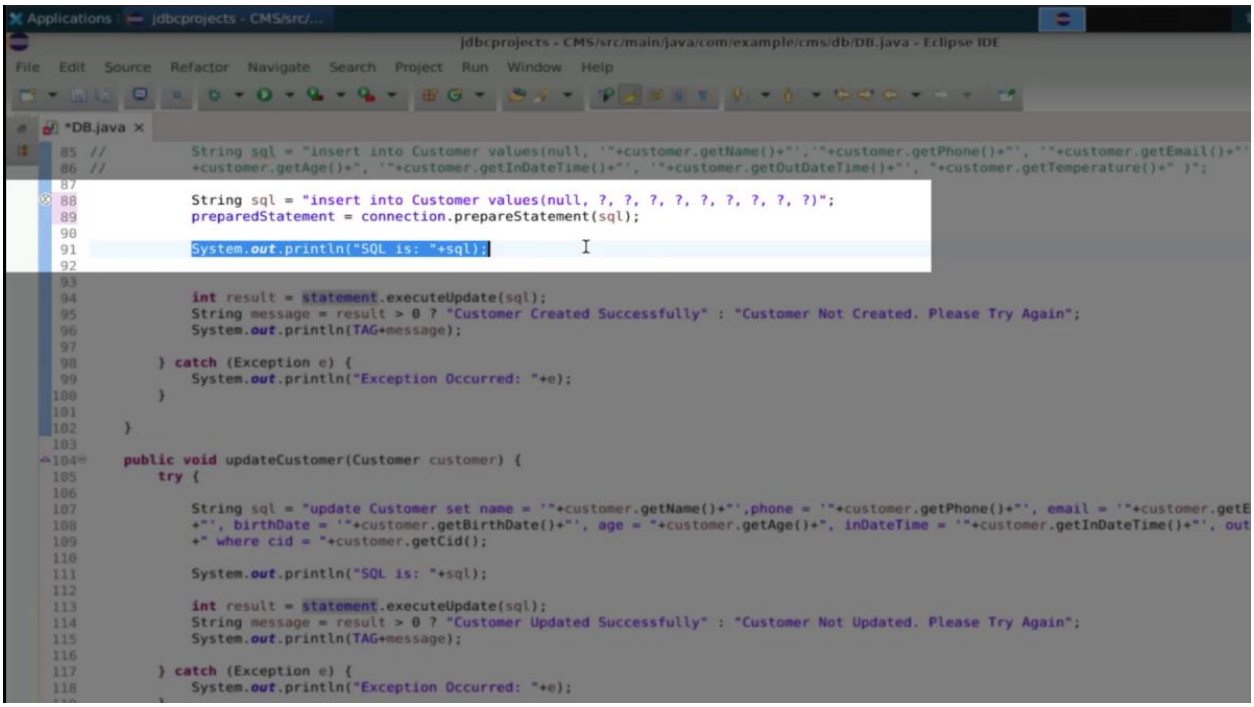
1.5 Write the SQL statement for the **insert** operation in the **Customer** table



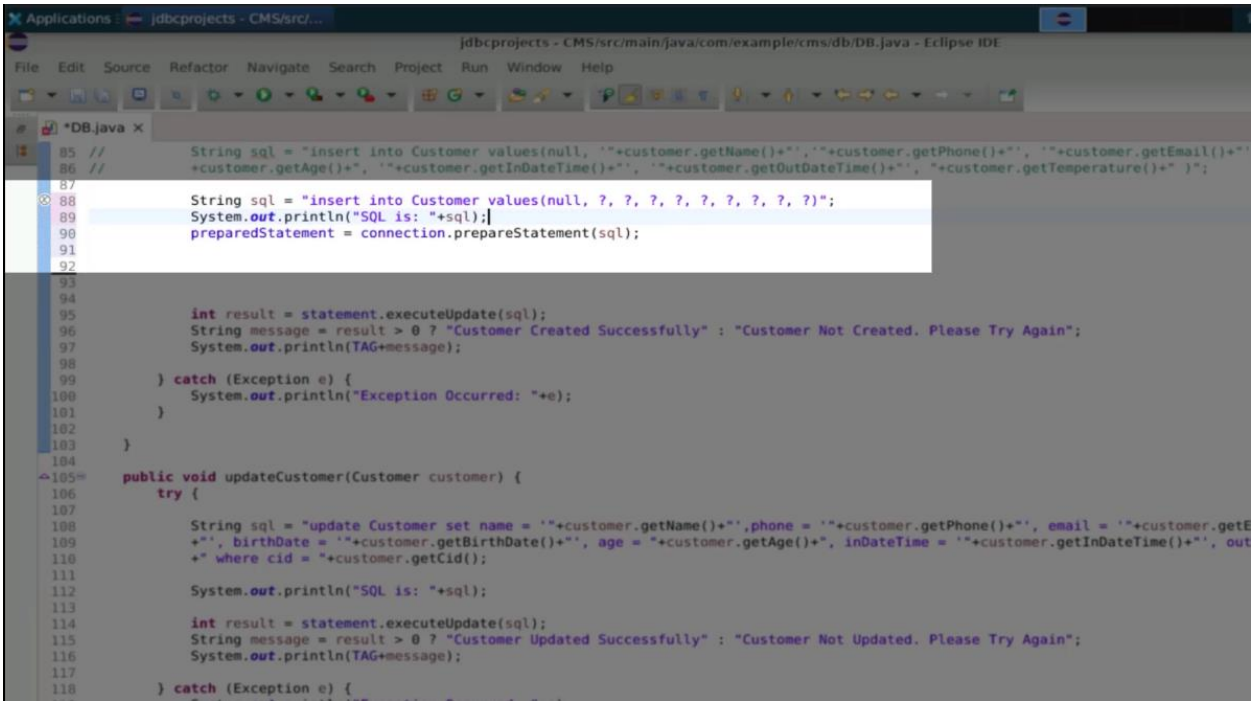
1.6 Initialize the **preparedStatement**



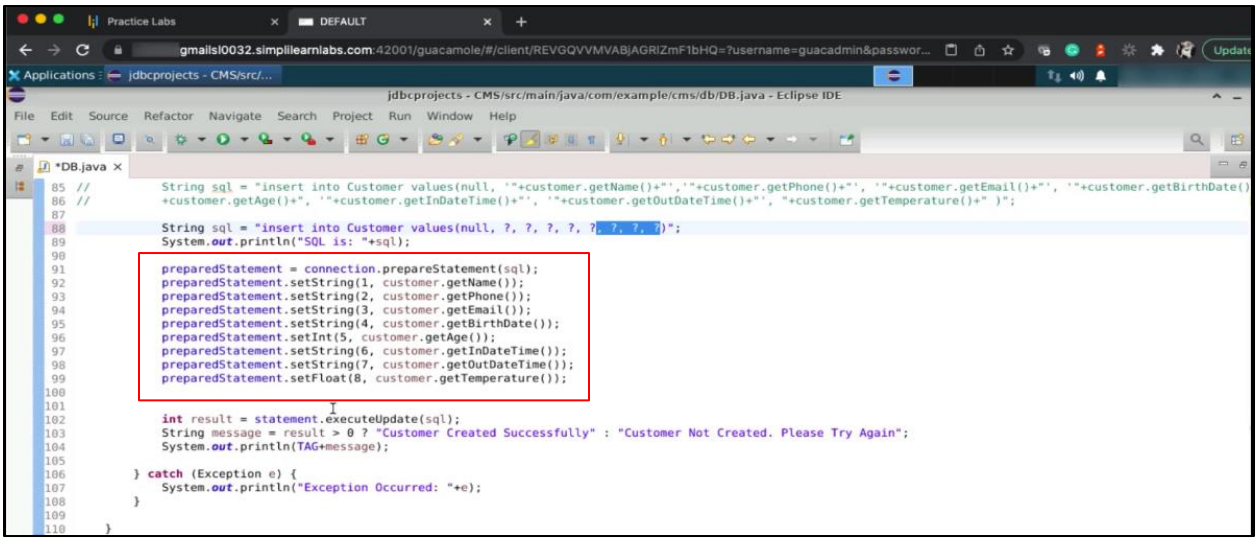
1.7 Cut **(Ctrl+X)** the selected line of code



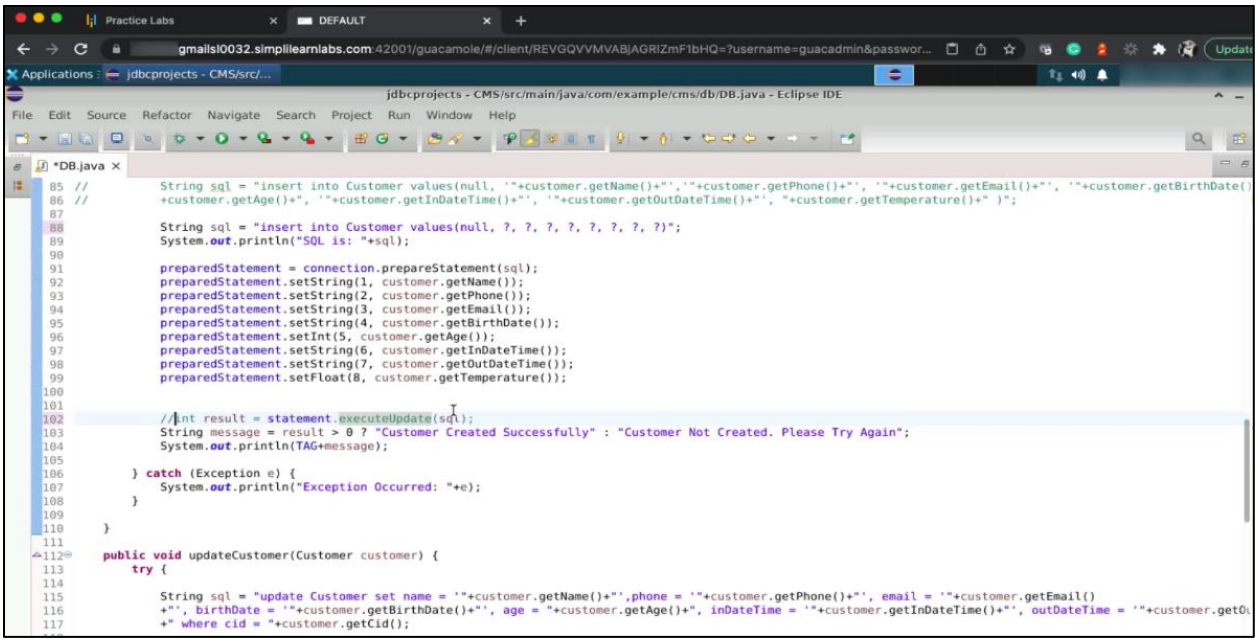
1.8 Paste the code above the preparedStatement object (line 89), and now pass the **String sql**



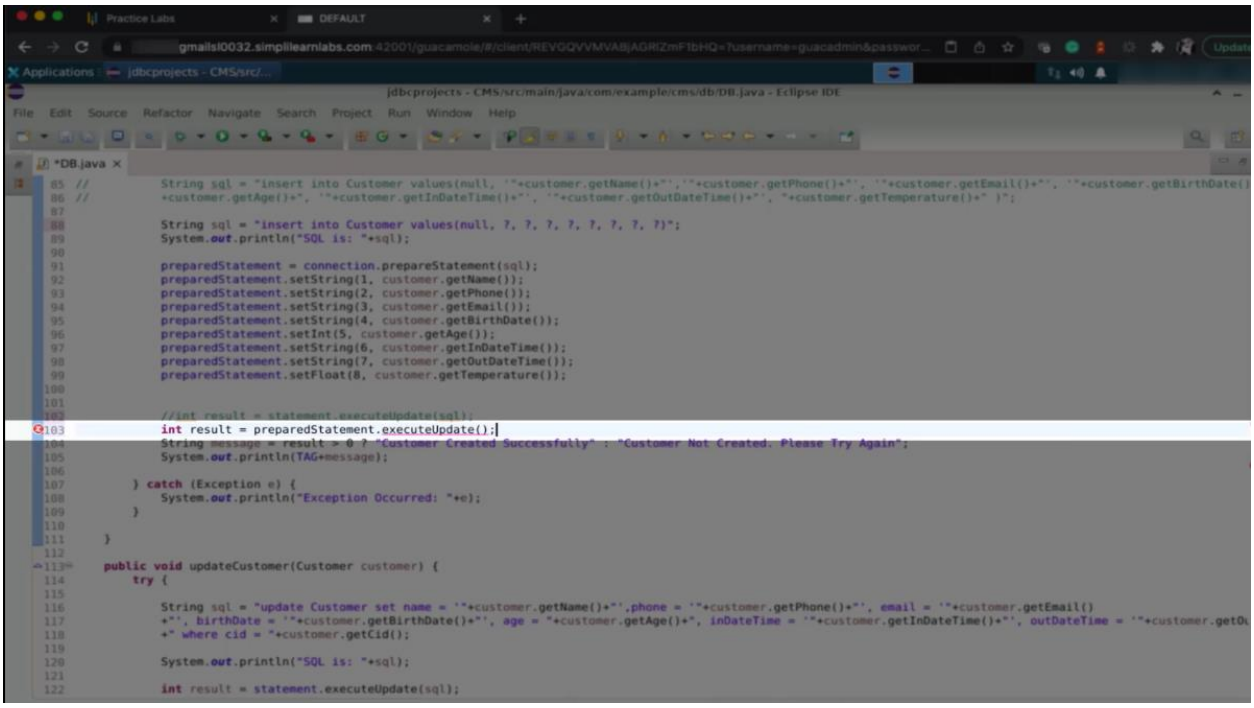
1.9 Substitute the data through the **getter** method, use the **setString()** method and index starting from 1 for each attribute of data, as shown below from line 92 to line 99:



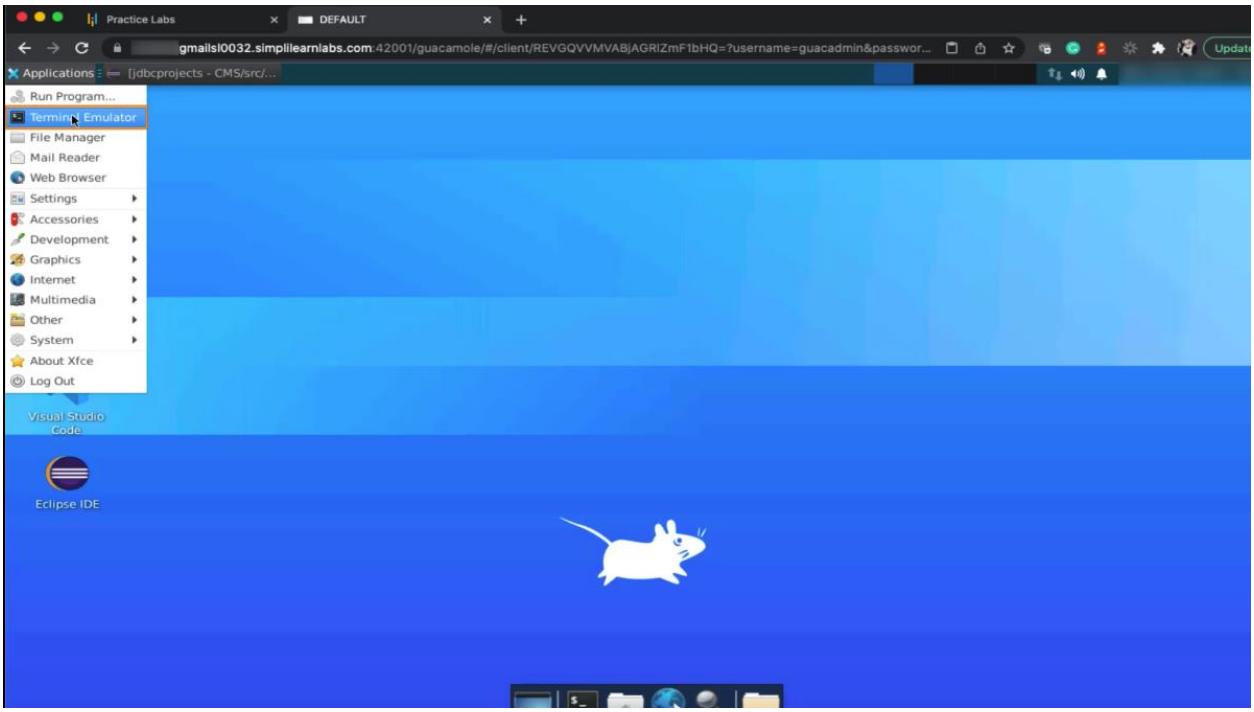
1.10 Comment on the calling of the **executeUpdate()** function, as it will remain similar to the above lines



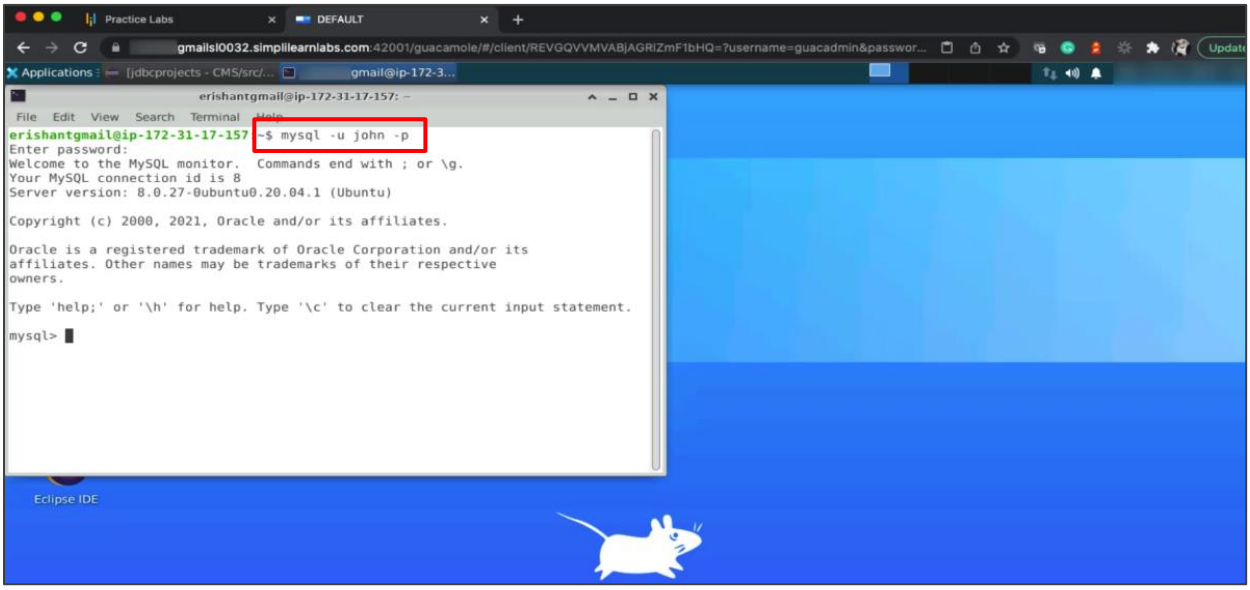
1.11 To execute the method, use **executeUpdate()**



1.12 Open the terminal

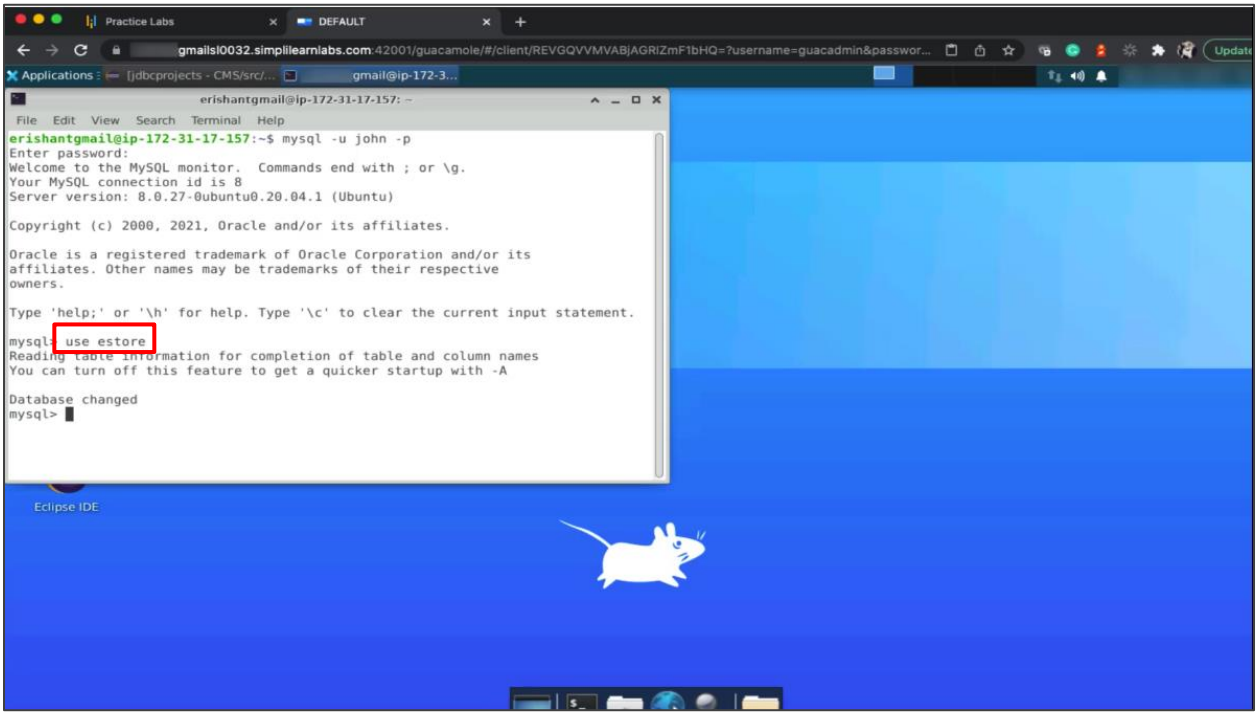


1.13 Log in into **SQL** using the command **mysql -u john -p** and the password is **john**.

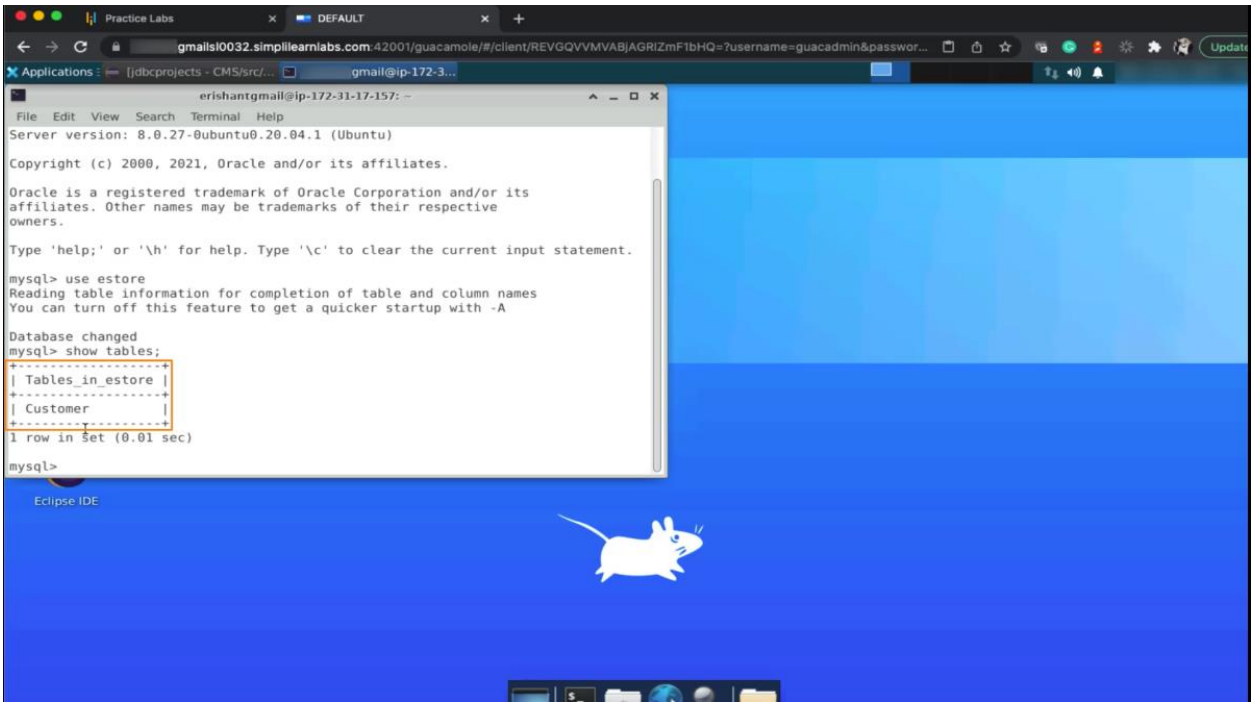


Note: A user named **john** has already been created for the database.

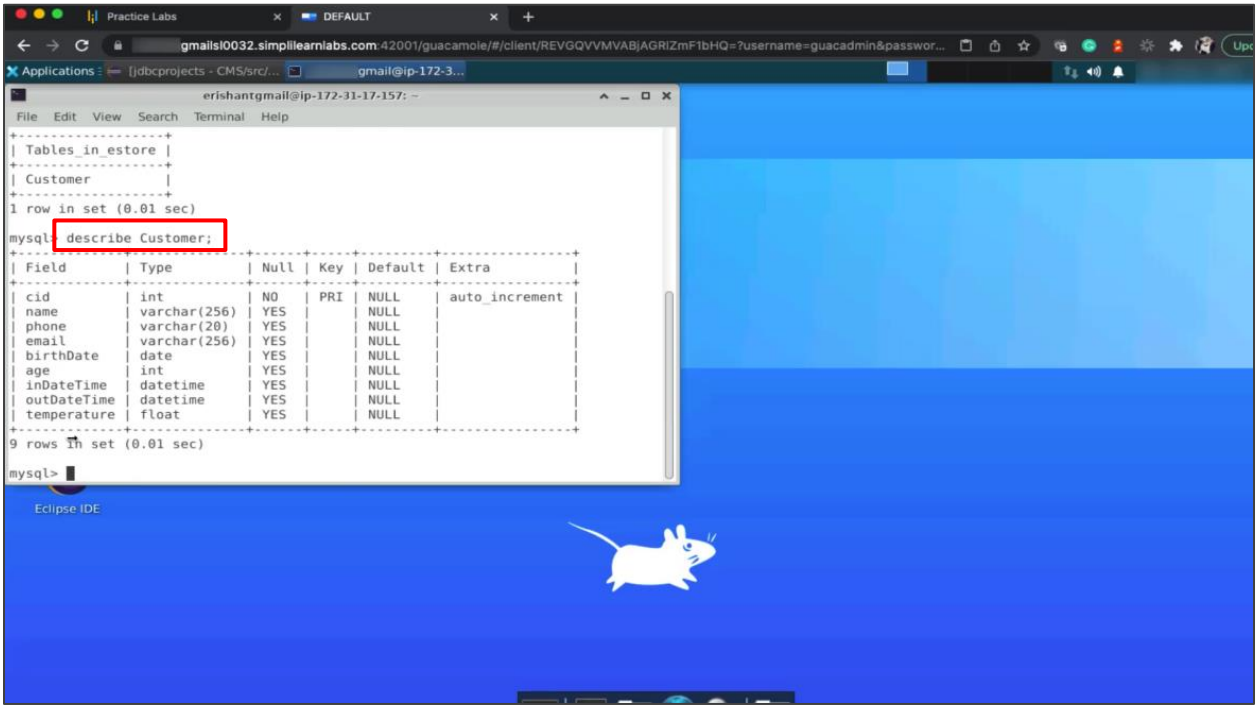
1.14 Run command **use estore;** to change the database



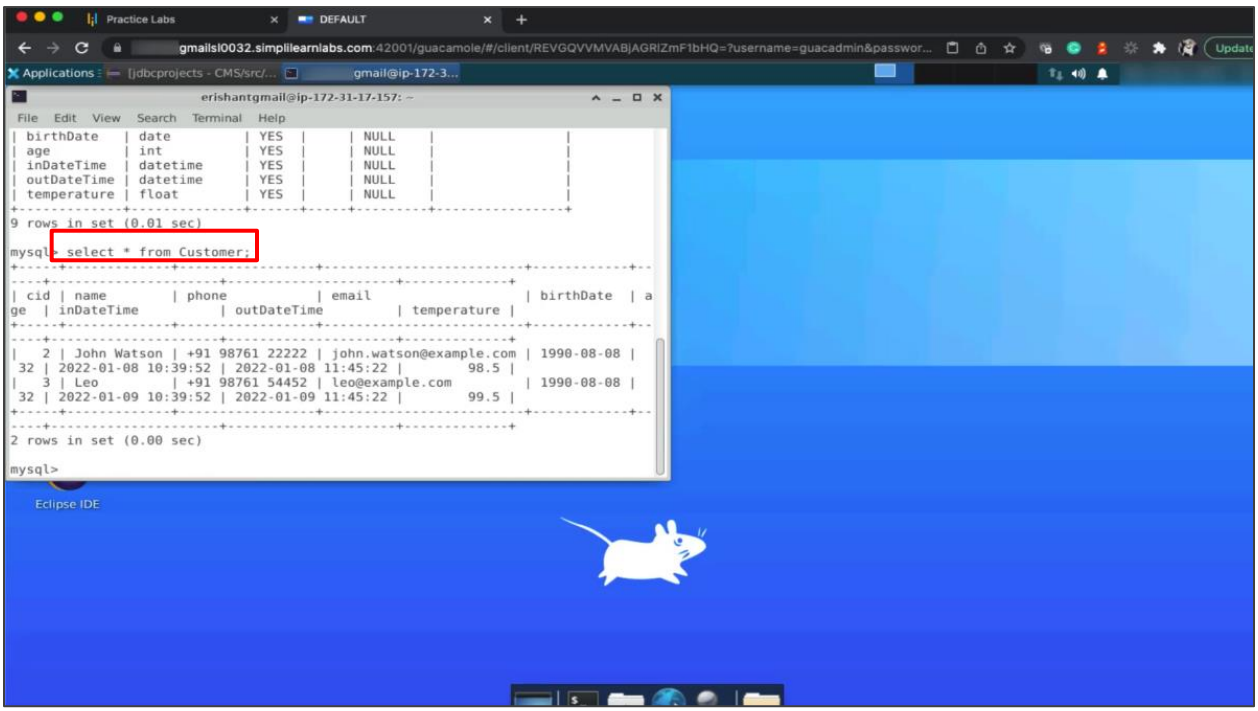
1.15 Run the **show tables;** command to check available tables in the **estore** database



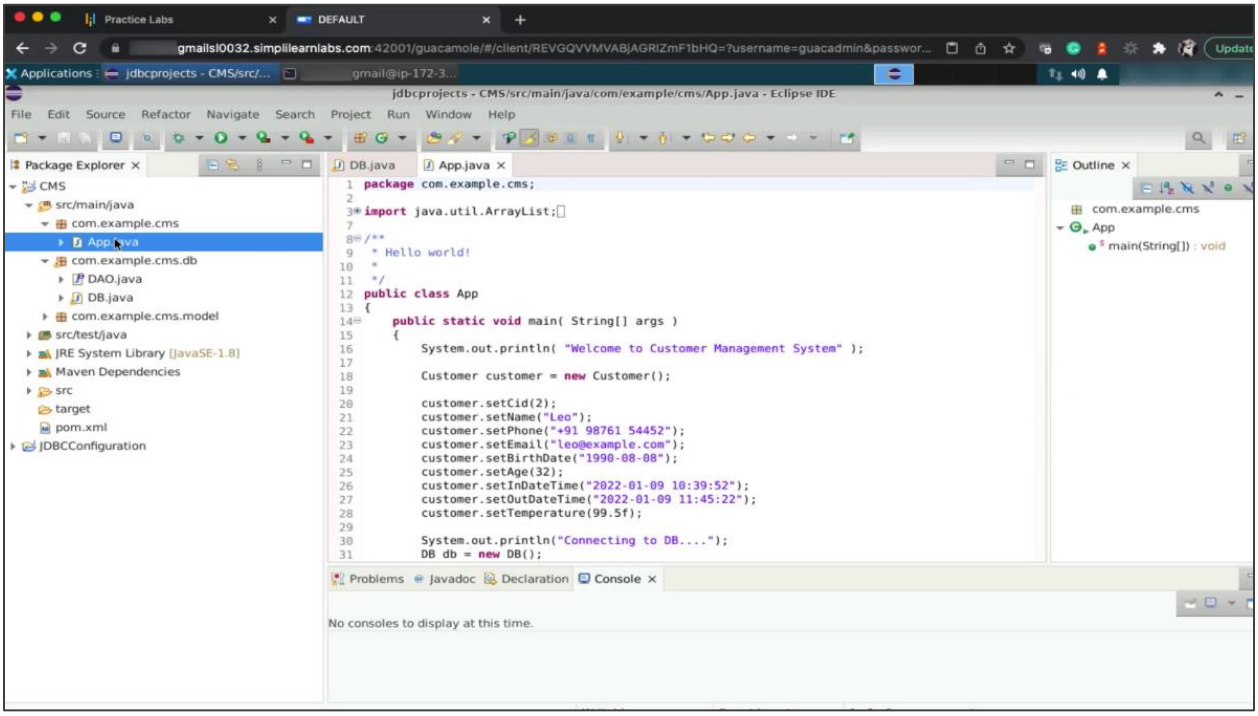
1.16 Run the **describe Customer;** command to see table details



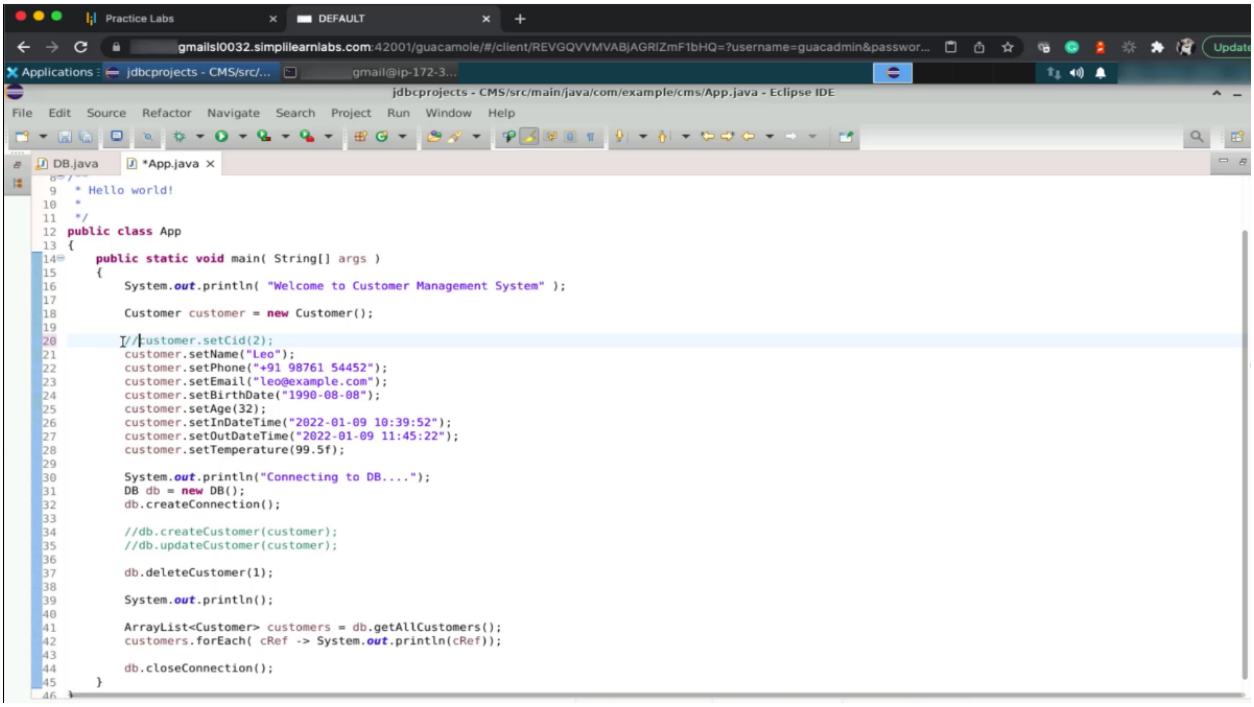
1.17 Run the select command as **select * from Customer;** to see data from the table



1.18 Go back to **Eclipse** and open the **App.java** file to use the insert query



1.19 Comment out **Cid** as it is an auto-increment



1.20 Change the details in the customer fields to create a new customer dataset

```
        System.out.println( "Welcome to Customer Management System" );

        Customer customer = new Customer();

        //customer.setCid(2);
        customer.setName("George");
        customer.setPhone("+91 [illegible] 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(99.5f);

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

```
        //customer.setCid(2);
        customer.setName("George");
        customer.setPhone("+91 99009 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(99.5f);

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

```
        //customer.setCid(2);
        customer.setName("George");
        customer.setPhone("+91 99009 88088");
        customer.setEmail("george@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(99.5f);

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

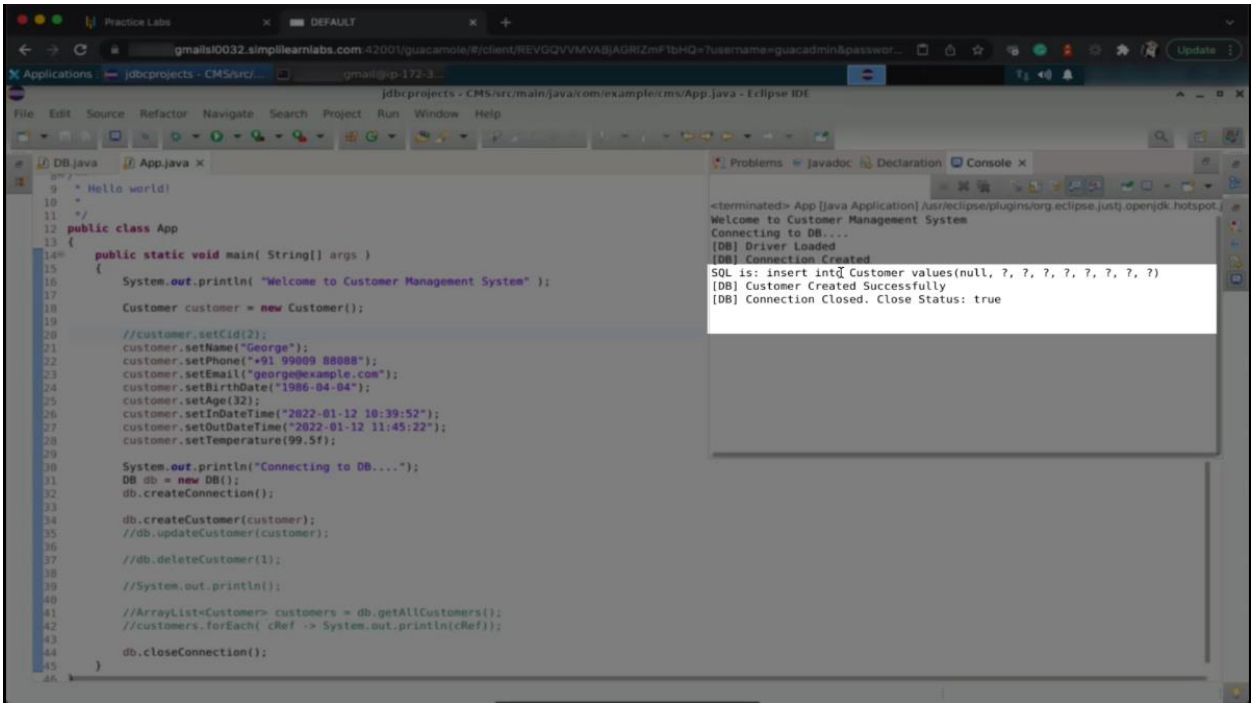
```
        //customer.setCid(2);
        customer.setName("George");
        customer.setPhone("+91 99009 88088");
        customer.setEmail("george@example.com");
        customer.setBirthDate("1986-04-04");
        customer.setAge(32);
        customer.setInDateTime("2022-01-09 10:39:52");
        customer.setOutDateTime("2022-01-09 11:45:22");
        customer.setTemperature(99.5f);

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

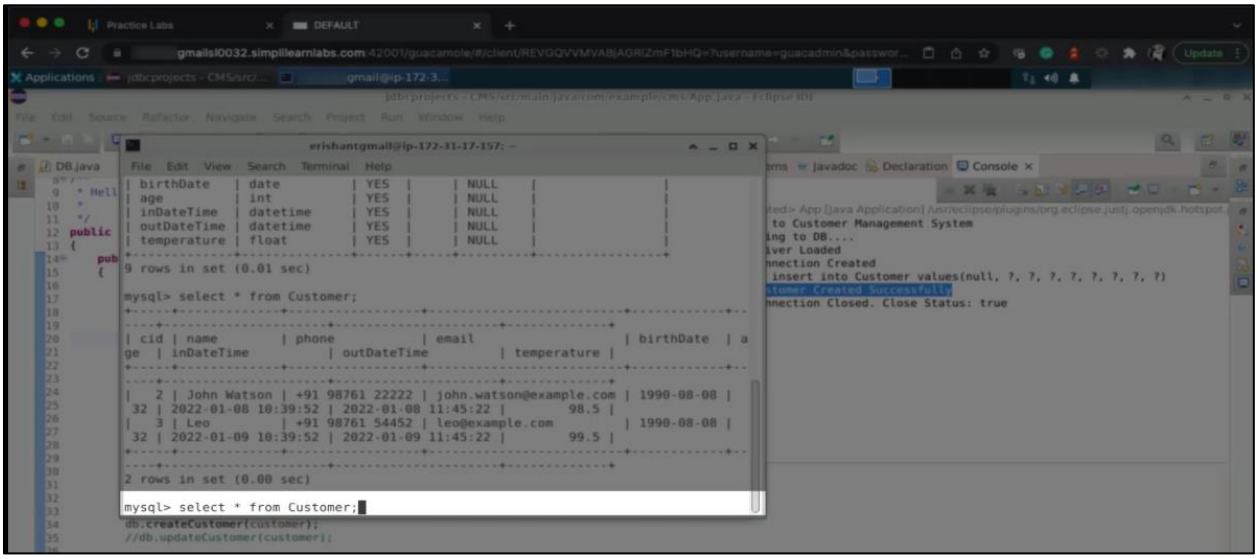
```
        //customer.setCid(2);
        customer.setName("George");
        customer.setPhone("+91 99009 88088");
        customer.setEmail("george@example.com");
        customer.setBirthDate("1986-04-04");
        customer.setAge(32);
        customer.setInDateTime("2022-01-12 10:39:52");
        customer.setOutDateTime("2022-01-12 11:45:22");
        customer.setTemperature(99.5f);

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

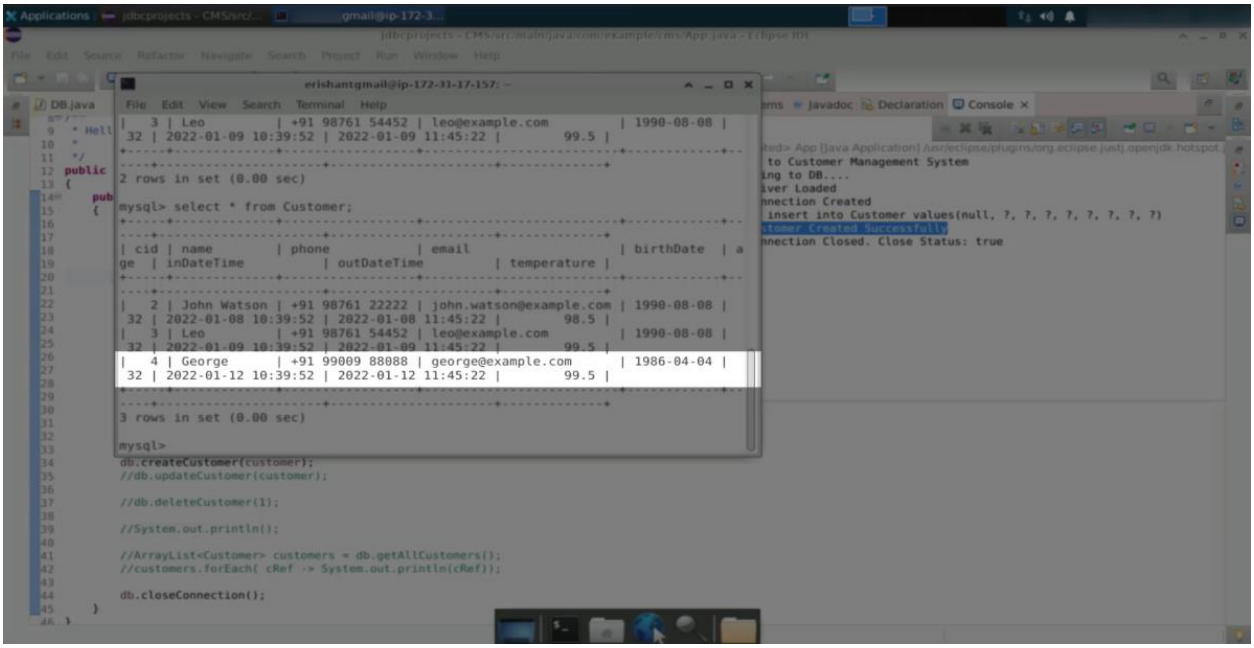
1.21 Run the code, and the output can be seen in the console as **Customer Created Successfully**



1.22 Go back to the terminal and run the **select** command

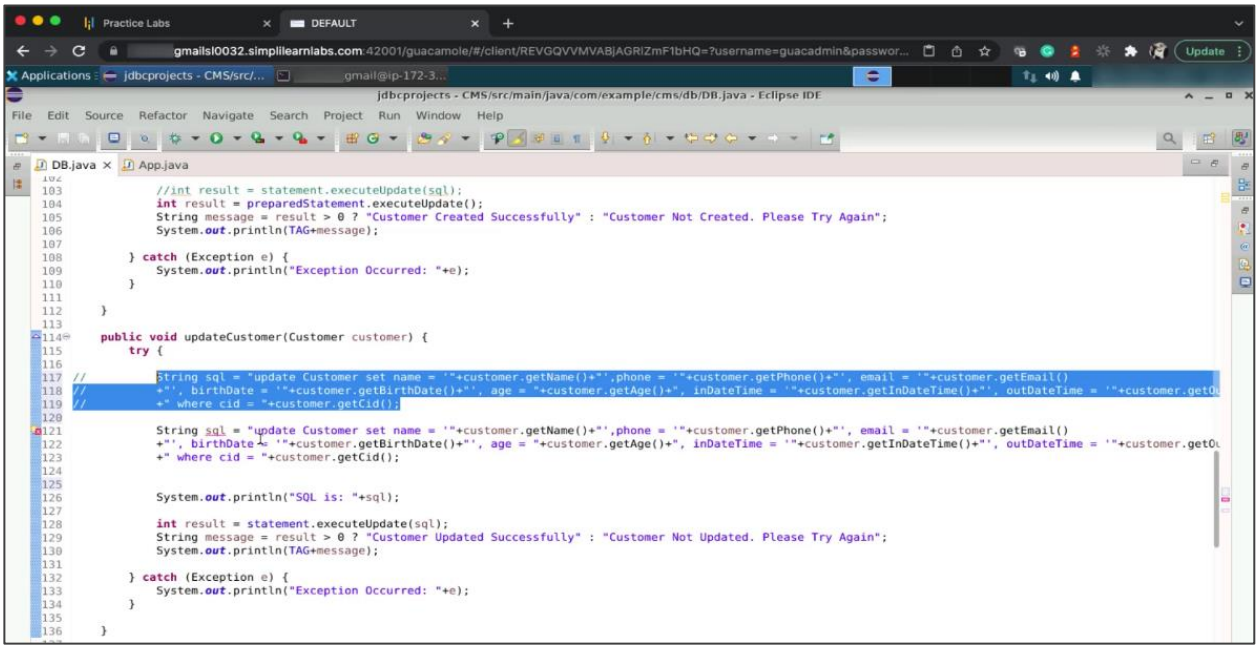


A record inserted can now be seen in the output as:

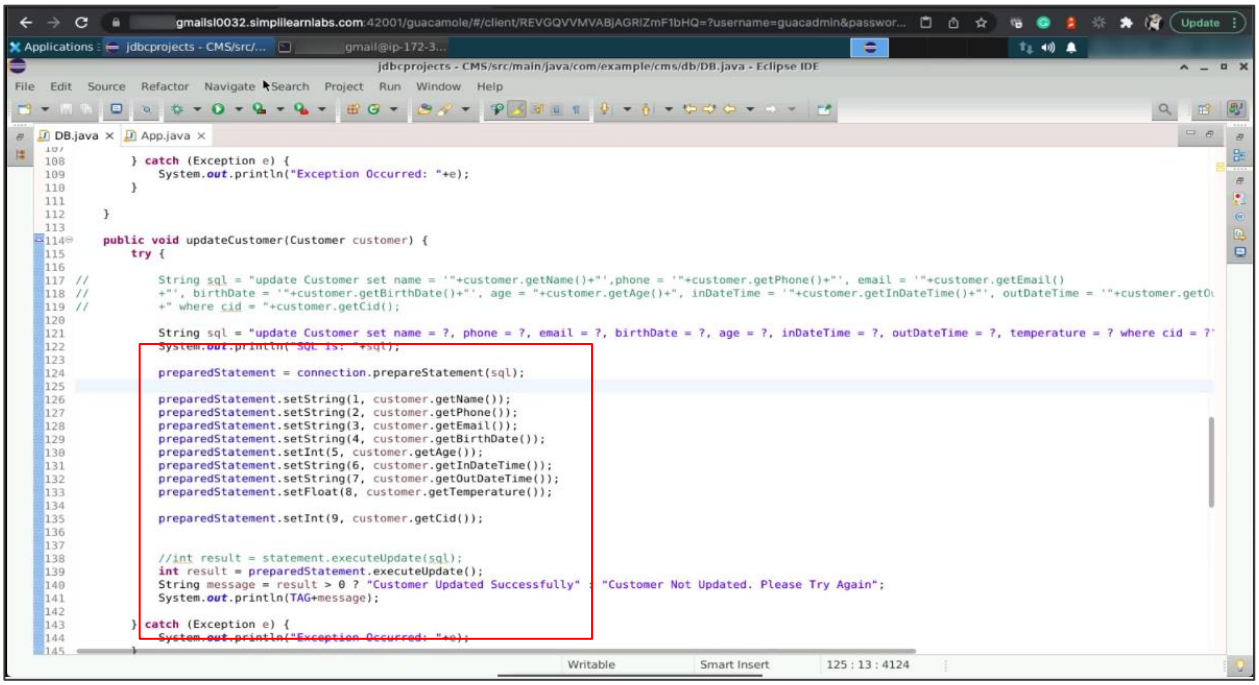


Step 2: Perform the Update operation

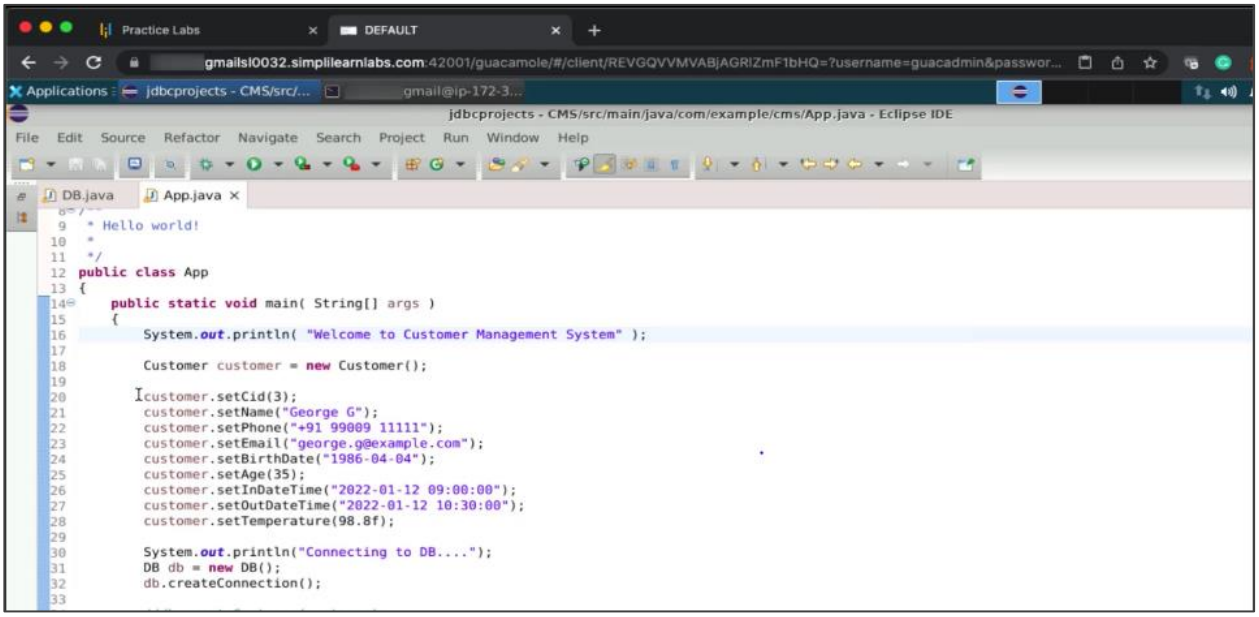
2.1 Open **Eclipse**. Go to the **DB.java** file and find the **updateCustomer()** method. Comment the SQL string and substitute it with the wildcard characters.



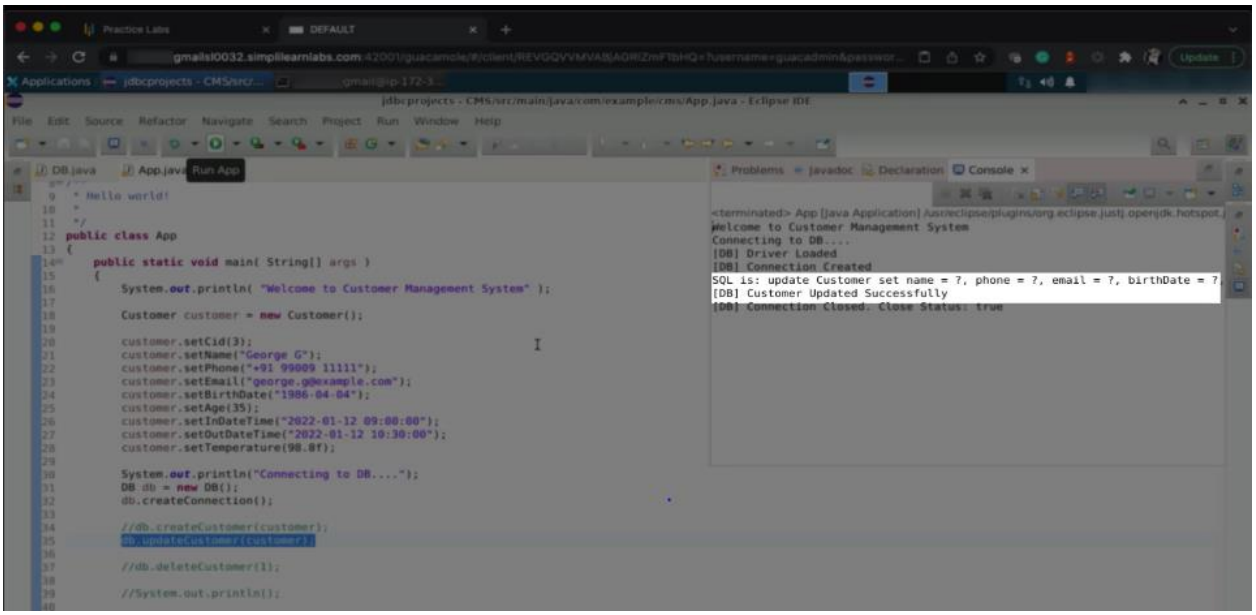
2.2 Write the lines of code from 124 to 141 in which we are creating the **preparedStatment** for updating the customer details



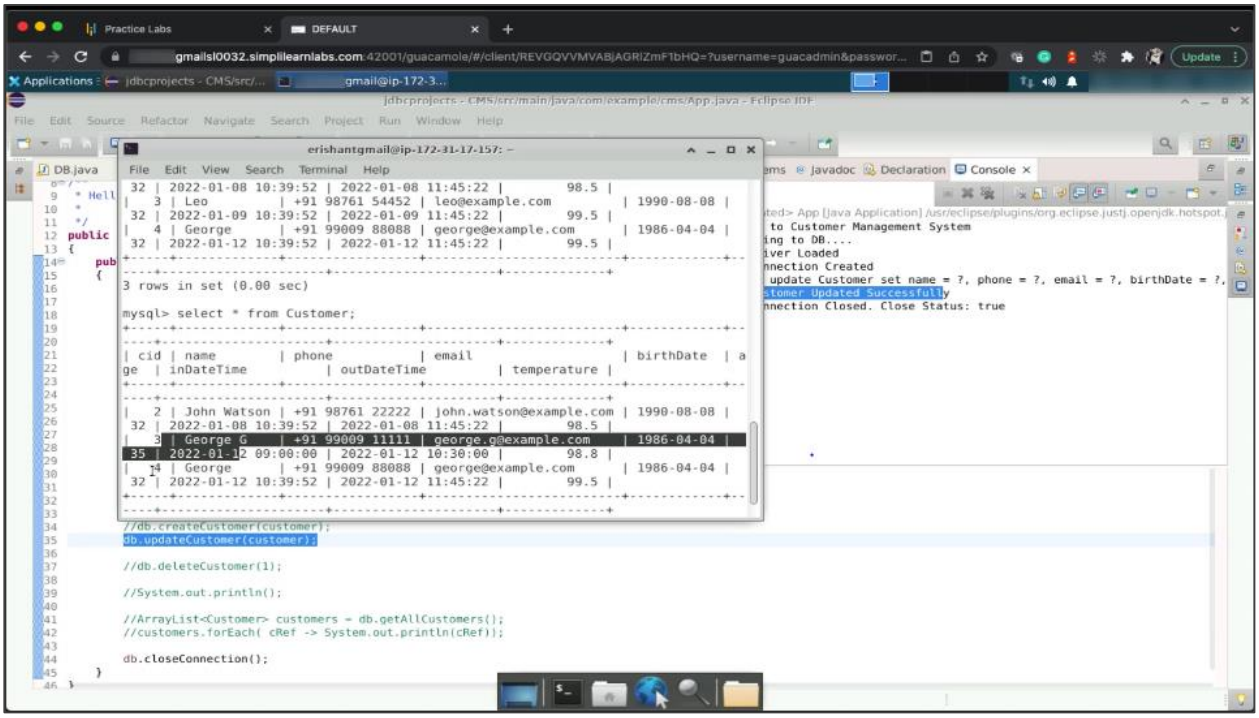
2.3 Go to the **App.java** file, uncomment the **updateCustomer** call, and comment on the **createCustomer** call. Make a few changes in customer details to update.



2.4 Run the code with changes. You can see the updated customer details.

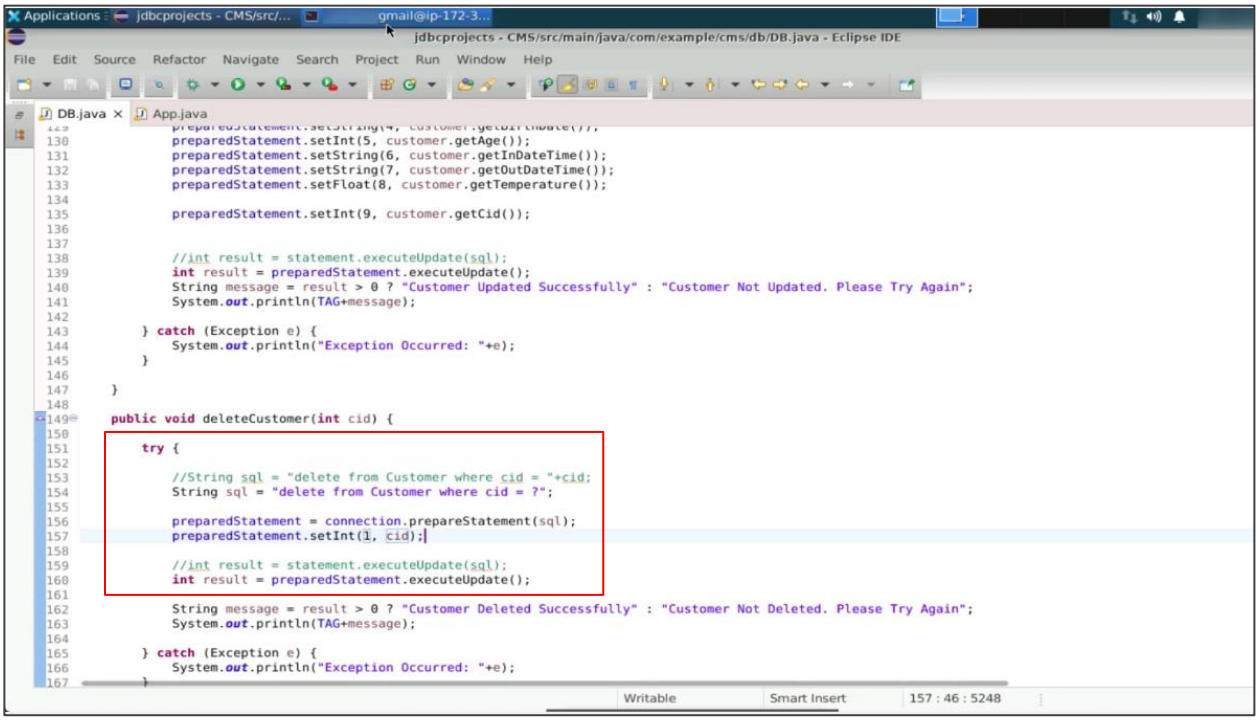


2.5 Open the terminal and run the **select** command. You can see a customer with **id = 3** has been updated.

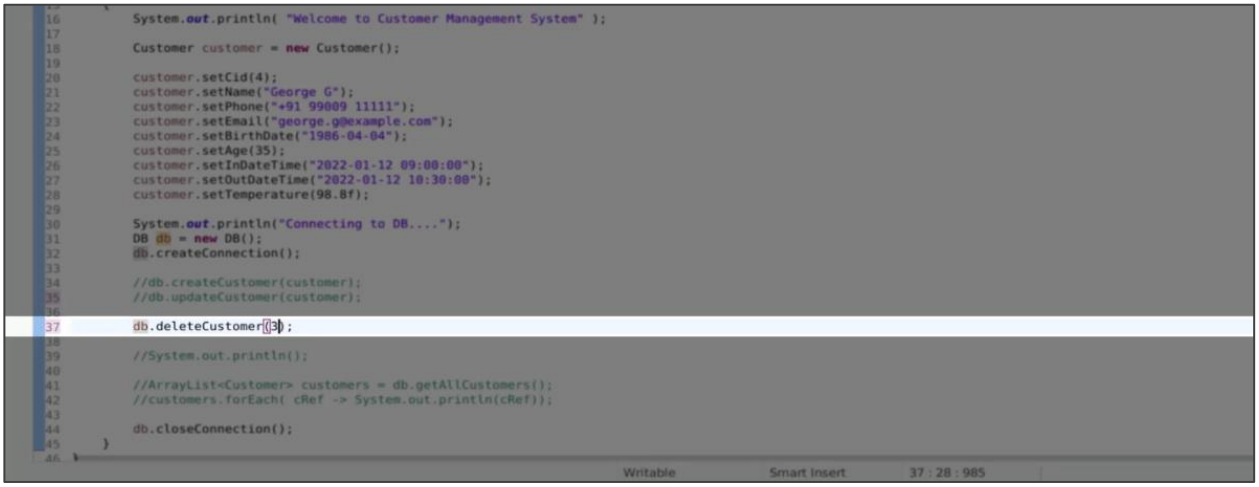


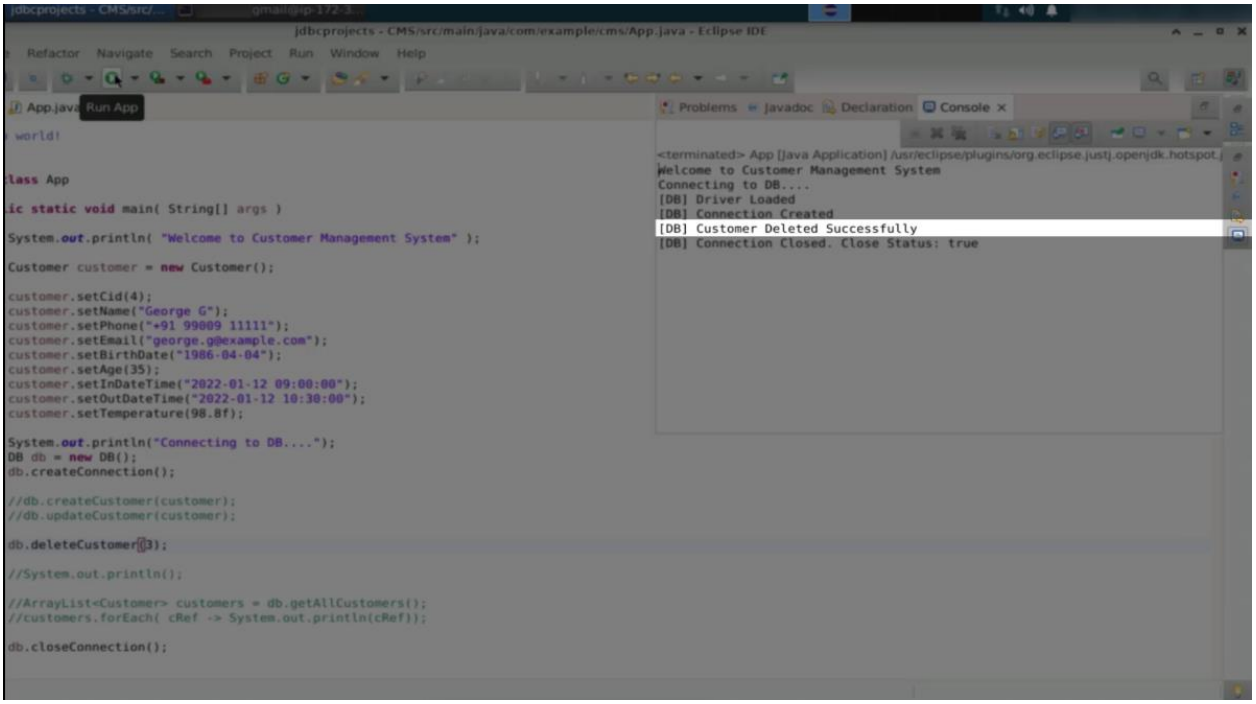
Step 3: Perform the Delete operation

3.1 Use the **delete** operation, use the **executeUpdate()** function and print a message for the successful deletion of the record (line 151 to line 160)



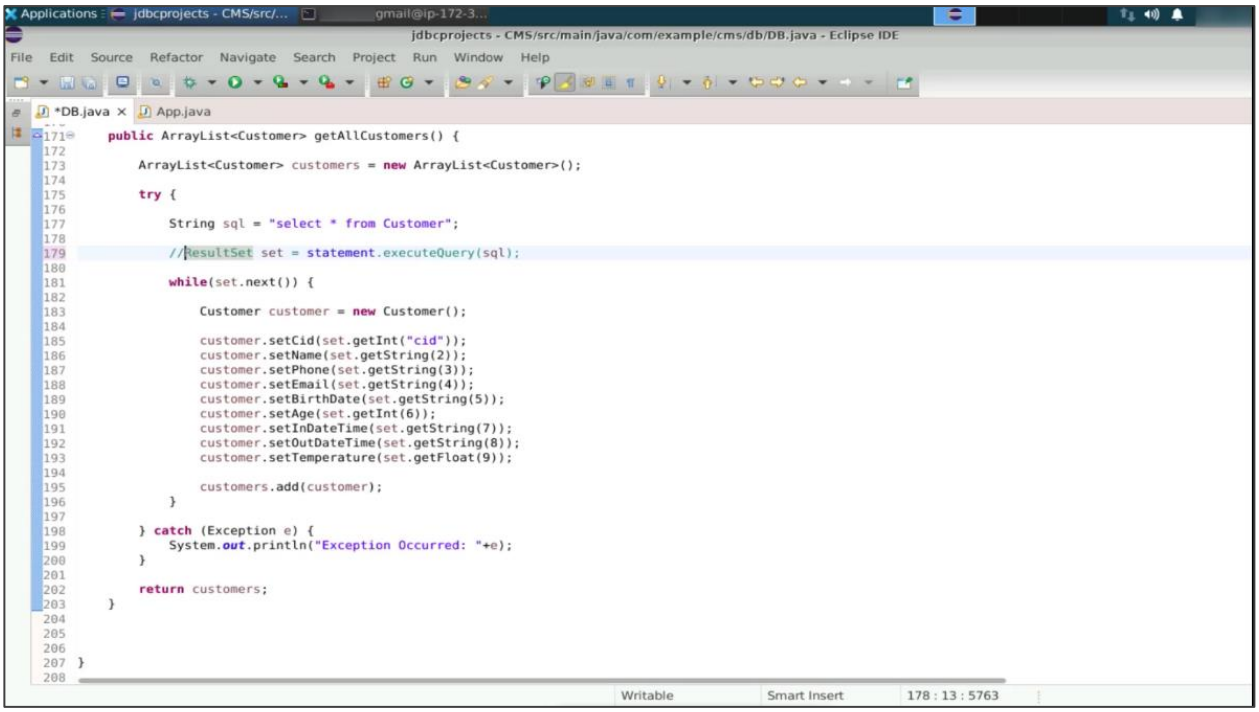
3.2 Call the **deleteCustomer()** method, save the code, and then run it in the **App.java** file. In the output, as you can see the customer record is deleted, and it prints **Customer Deleted Successfully**.



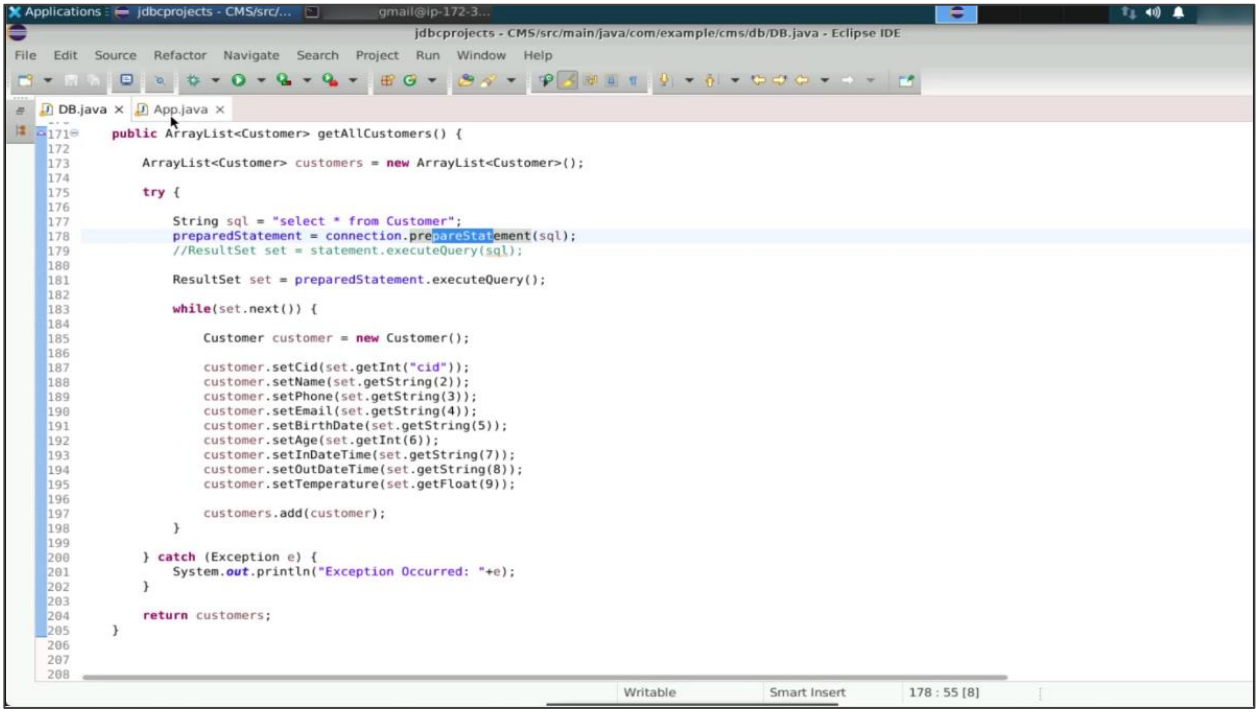


Step 4: Perform the getAllCustomers operation

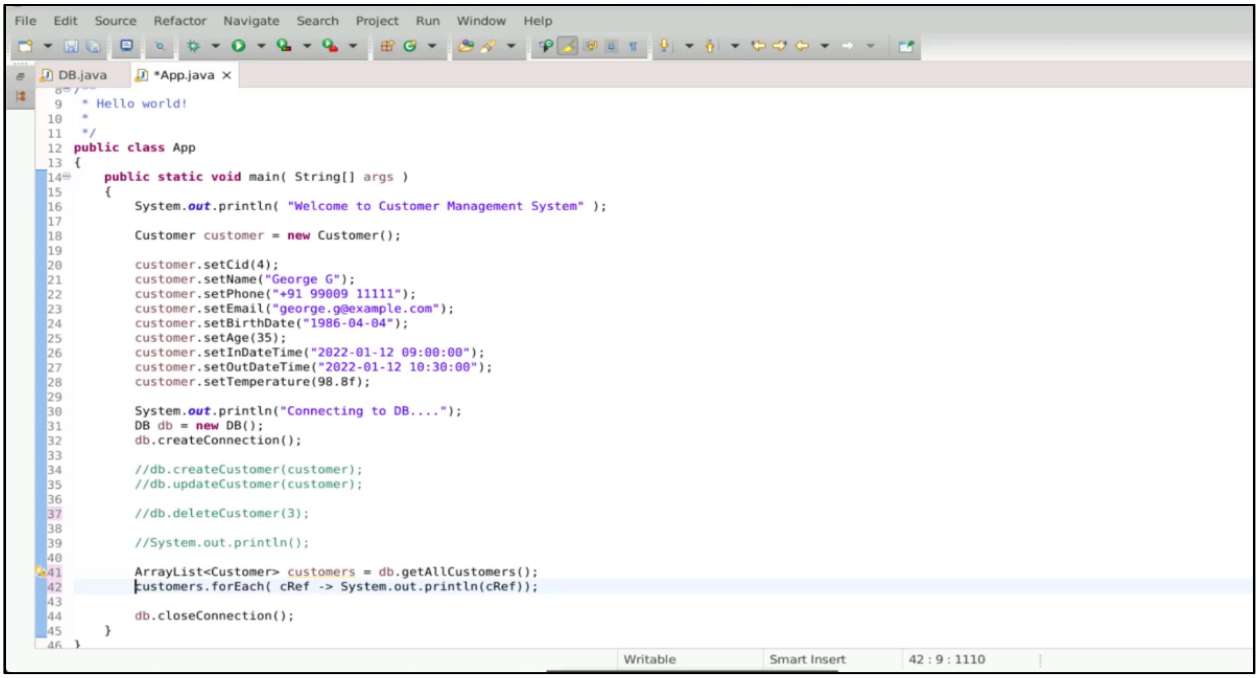
4.1 Comment out the **ResultSet** in the **DB.java** file



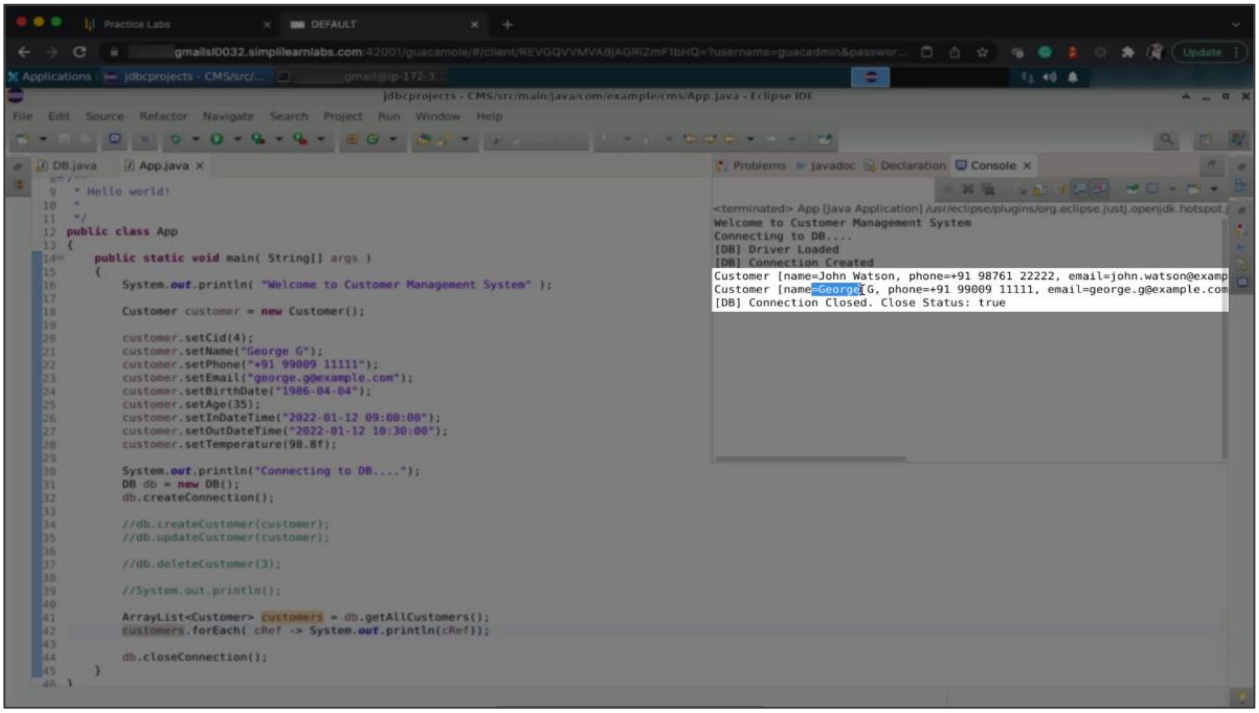
4.2 Create the **preparedStatement** and **ResultSet** object to call the **executeQuery()** function



4.3 Go to the **App.java** file. Uncomment the ArrayList and add a **forEach** loop to get all customer details.



4.4 Run the code, and you will see all customer details as output on the console



By following these steps, you have successfully implemented **PreparedStatment** API with JDBC for writing CRUD operations for managing database records efficiently and securely.