

Lesson 01 Demo 02

Creating a Table in DB and Implementing ORM with OOPs

Objective: To create a customer model, a table in DB, and implement ORM with OOPs for managing customer data in a Java application

Tool required: Eclipse IDE

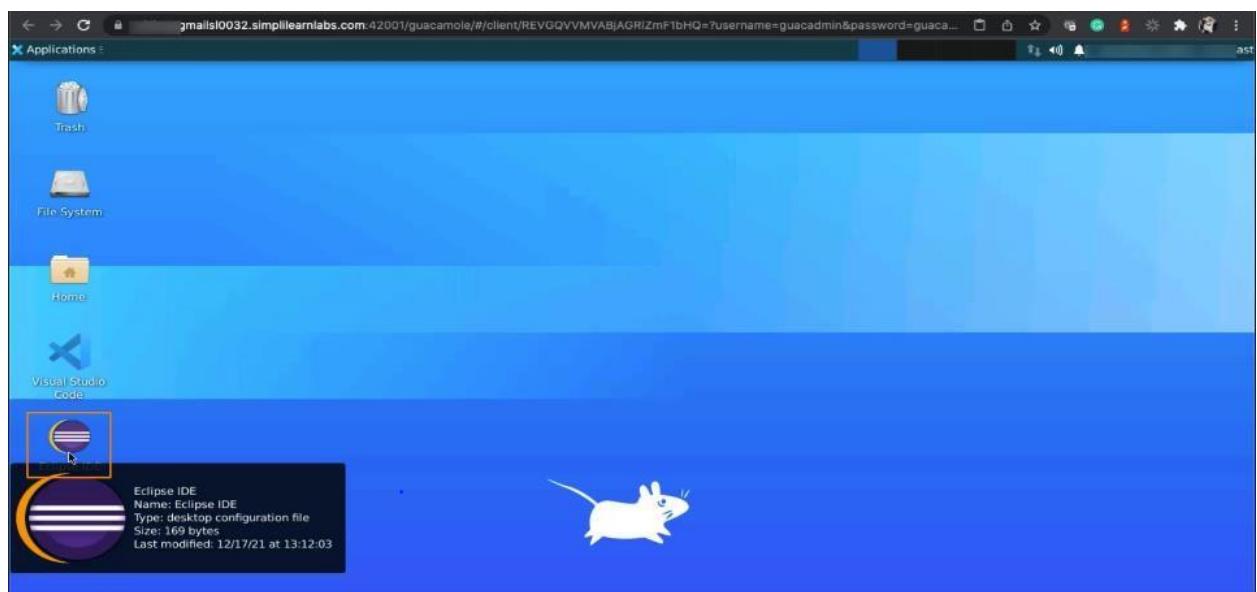
Prerequisites: Lesson 01 Demo 01

Steps to be followed:

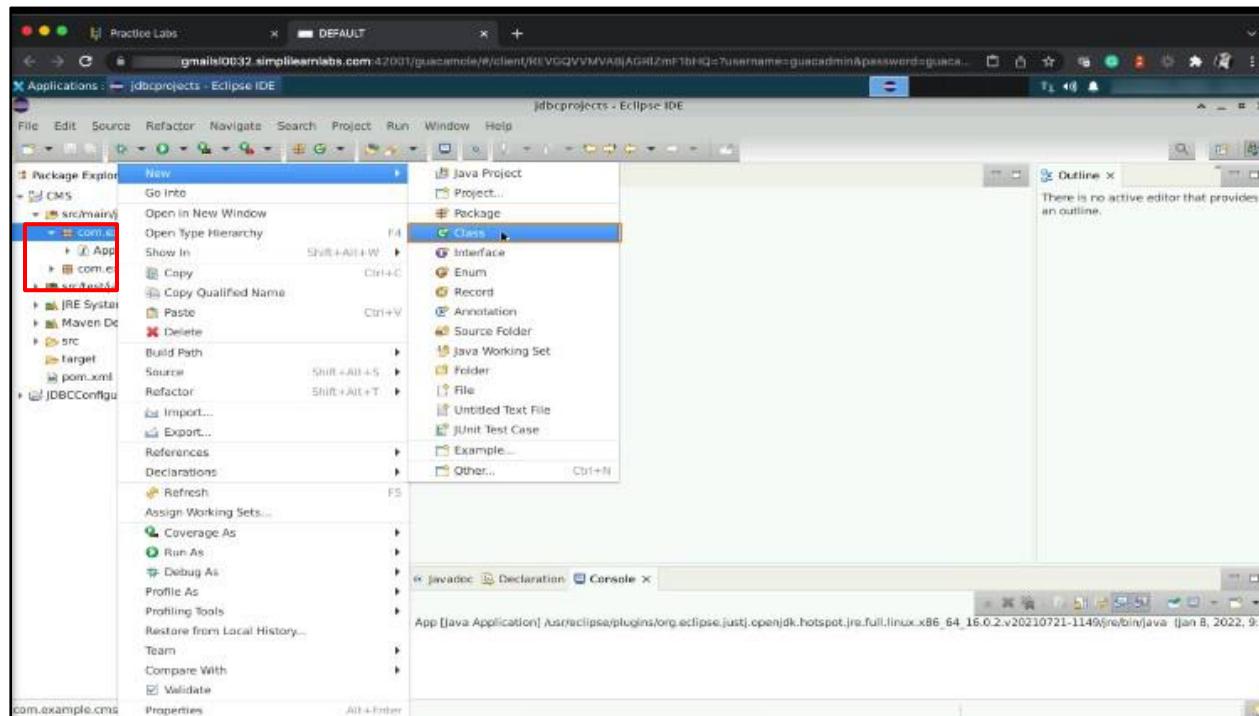
1. Create a new class
2. Write a customer object in App.java
3. Open the terminal window
4. Declare a method for a customer in the DAO interface

Step 1: Create a new class

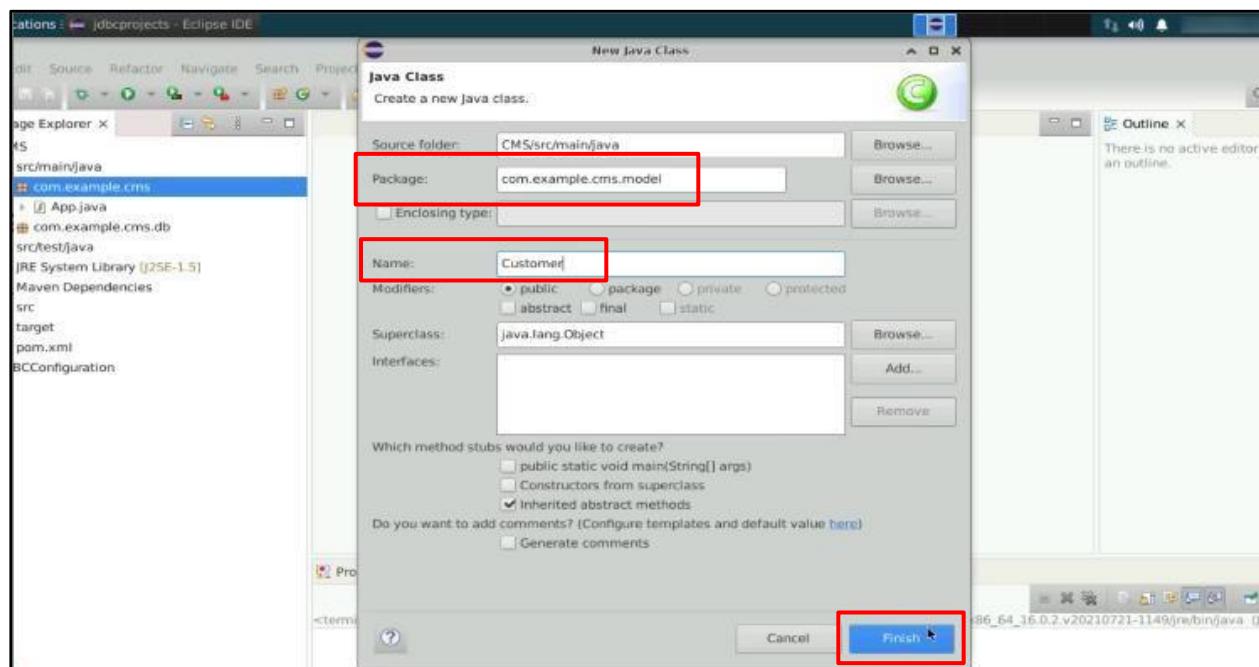
1.1 Open **Eclipse IDE** in your lab as shown in the screenshot below:



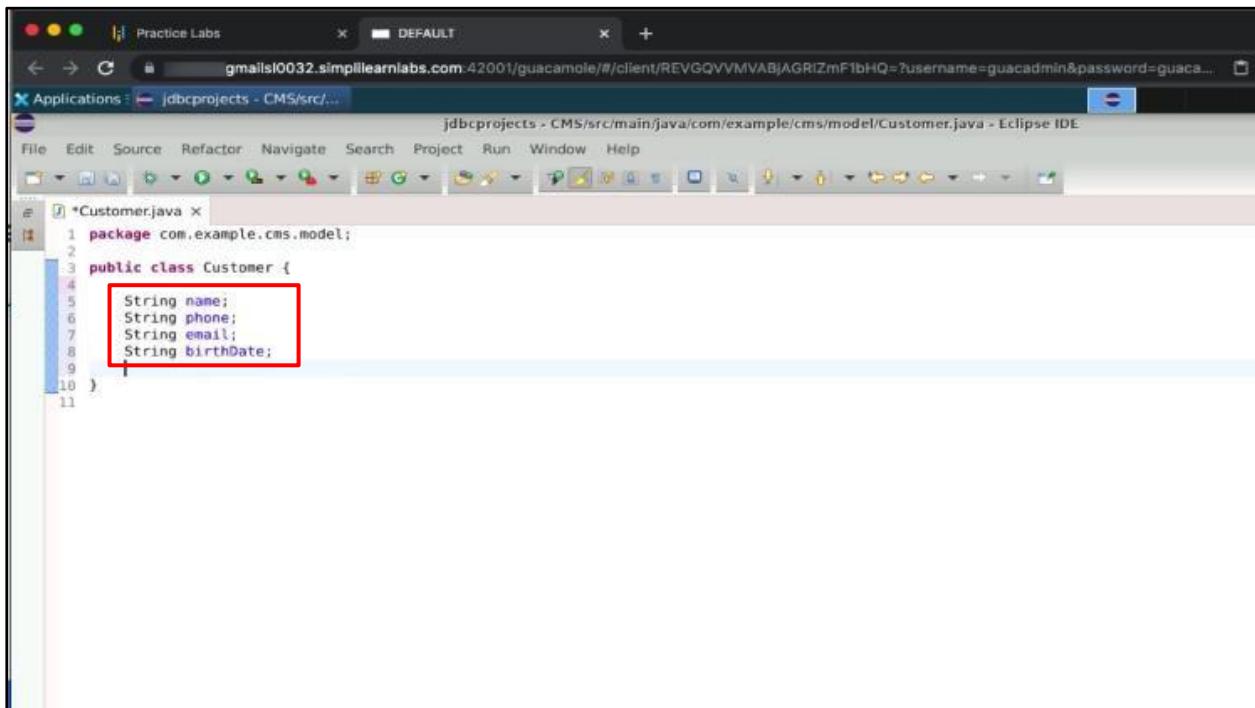
1.2 Right-click on **com.example.cms** in the **src** folder, select **New**, and click on **Class** to create a new class as shown in the screenshot below:



1.3 Add a name for class, package and click on **Finish** as shown in the screenshot below:



1.4 Add string attributes for the **Customer** database as shown in the below screenshot:



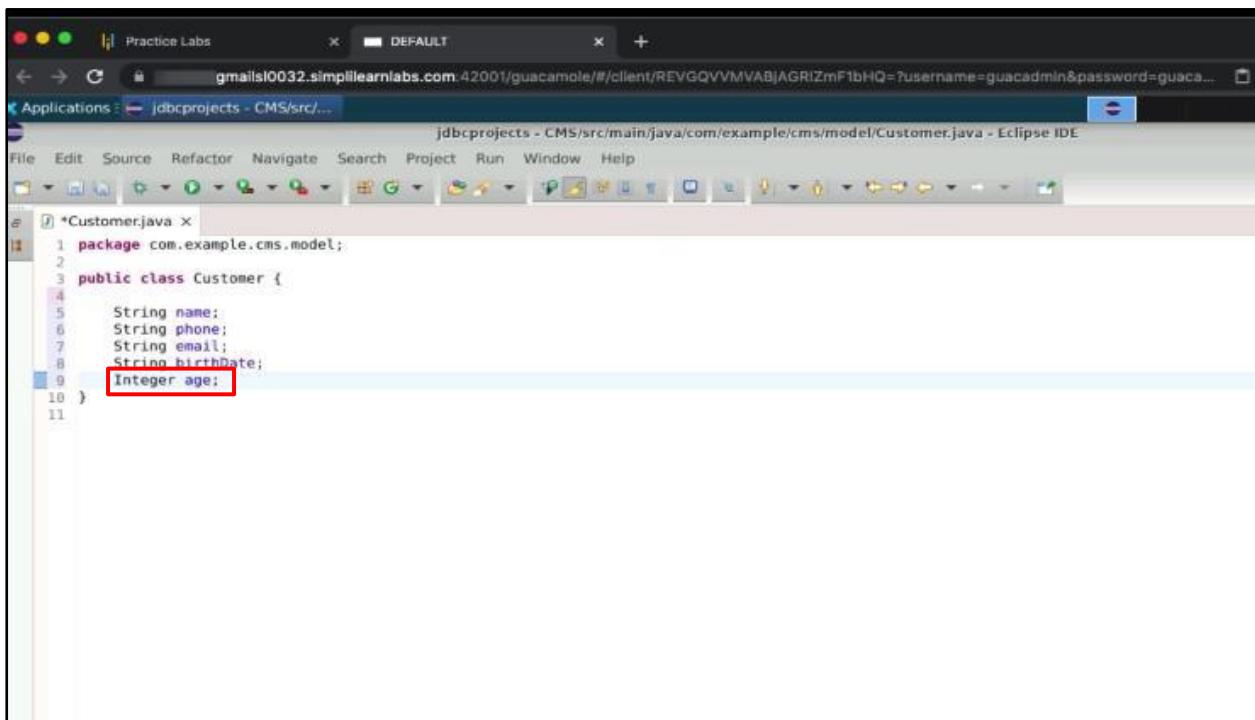
```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8 }
9
10
11
```

A screenshot of the Eclipse IDE interface. The title bar shows "Practice Labs" and "DEFAULT". 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 displays a Java file named "Customer.java". The class definition is as follows:

```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8 }
9
10
11
```

The lines containing the attribute declarations (String name; String phone; String email; String birthDate;) are highlighted with a red rectangular box.

1.5 Add an integer attribute for **age** as shown in the below screenshot:



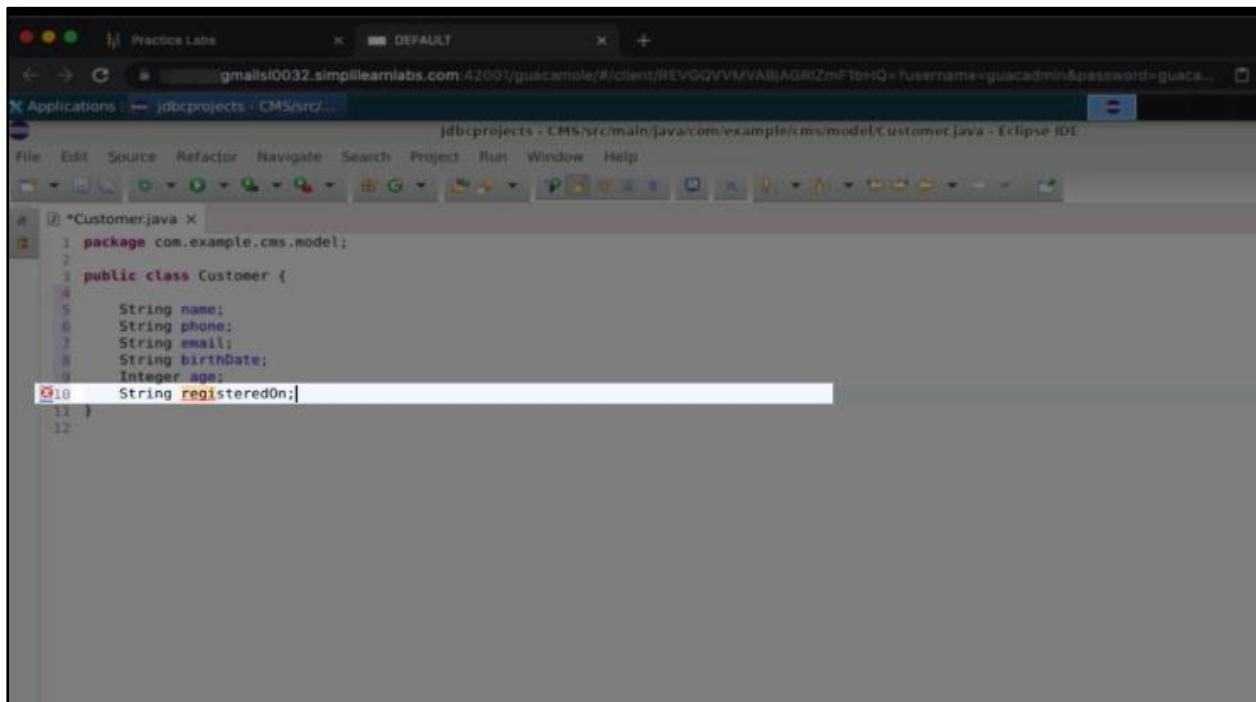
```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8     Integer age;
9 }
10
11
```

A screenshot of the Eclipse IDE interface, identical to the previous one but with a different code modification. The class definition is now:

```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8     Integer age;
9 }
10
11
```

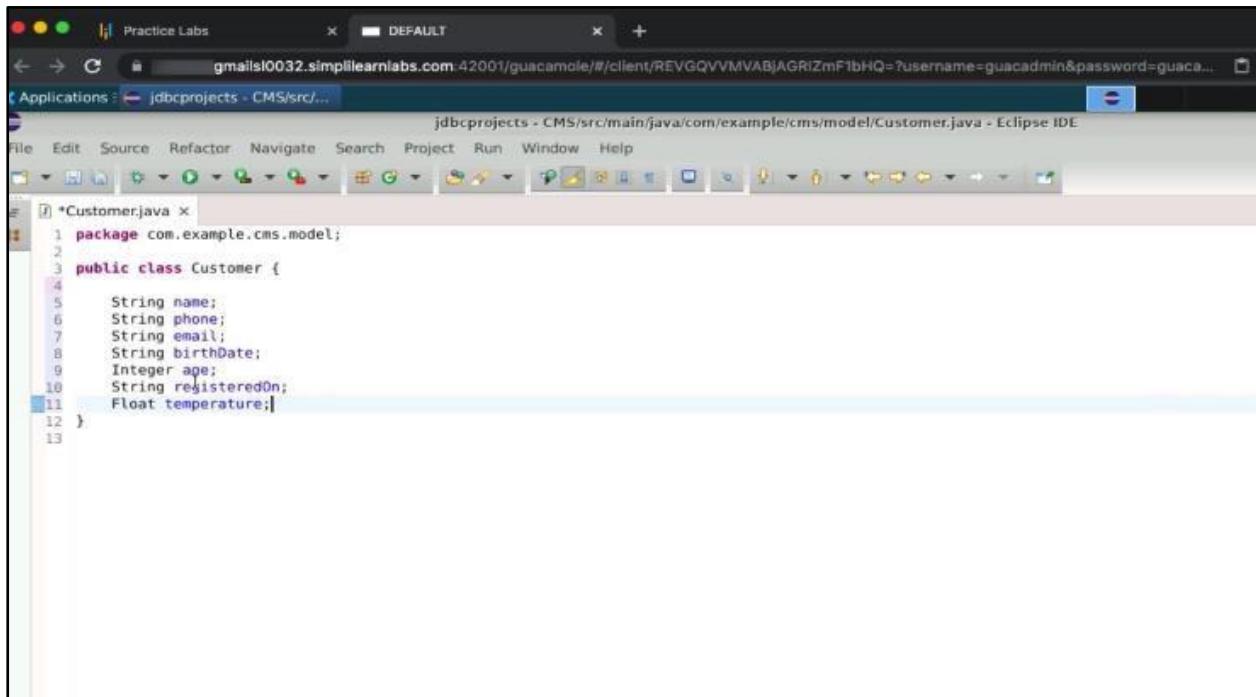
The line containing the attribute declaration (Integer age;) is highlighted with a red rectangular box.

1.6 Add a string attribute for **registeredOn**:



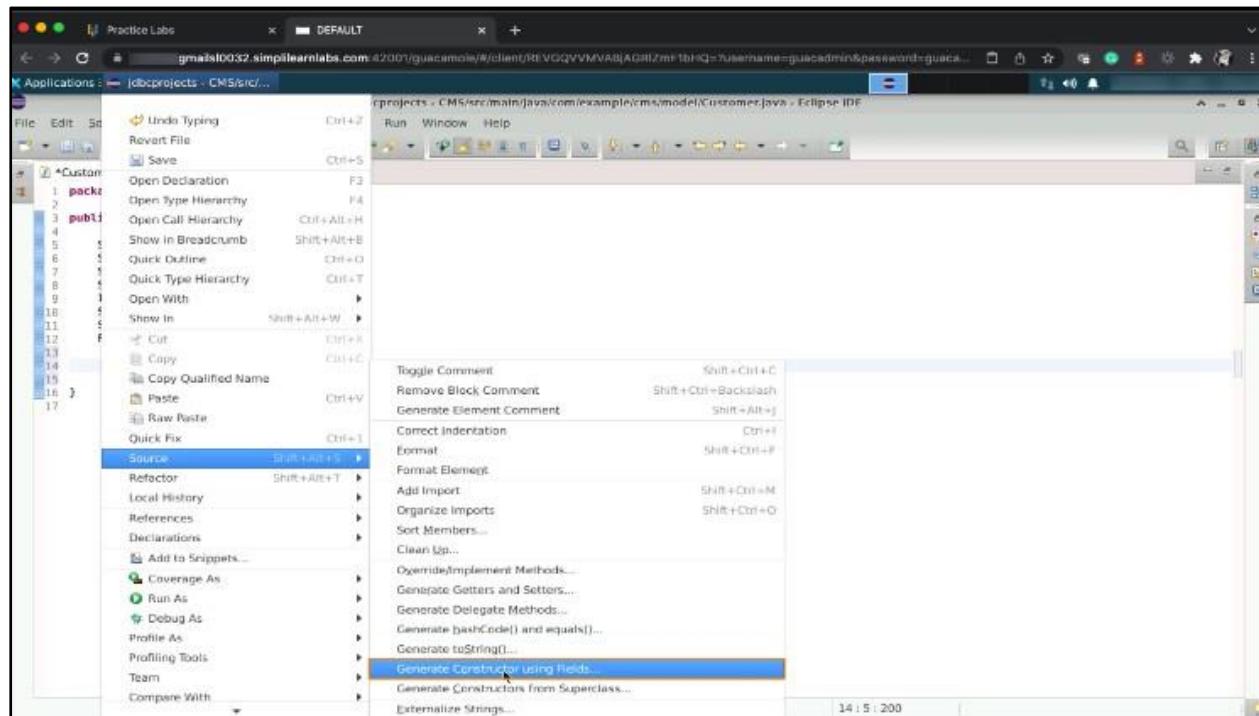
```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8     Integer age;
9
10    String registeredOn;
11 }
12
```

1.7 Add a floating-point attribute for **temperature**

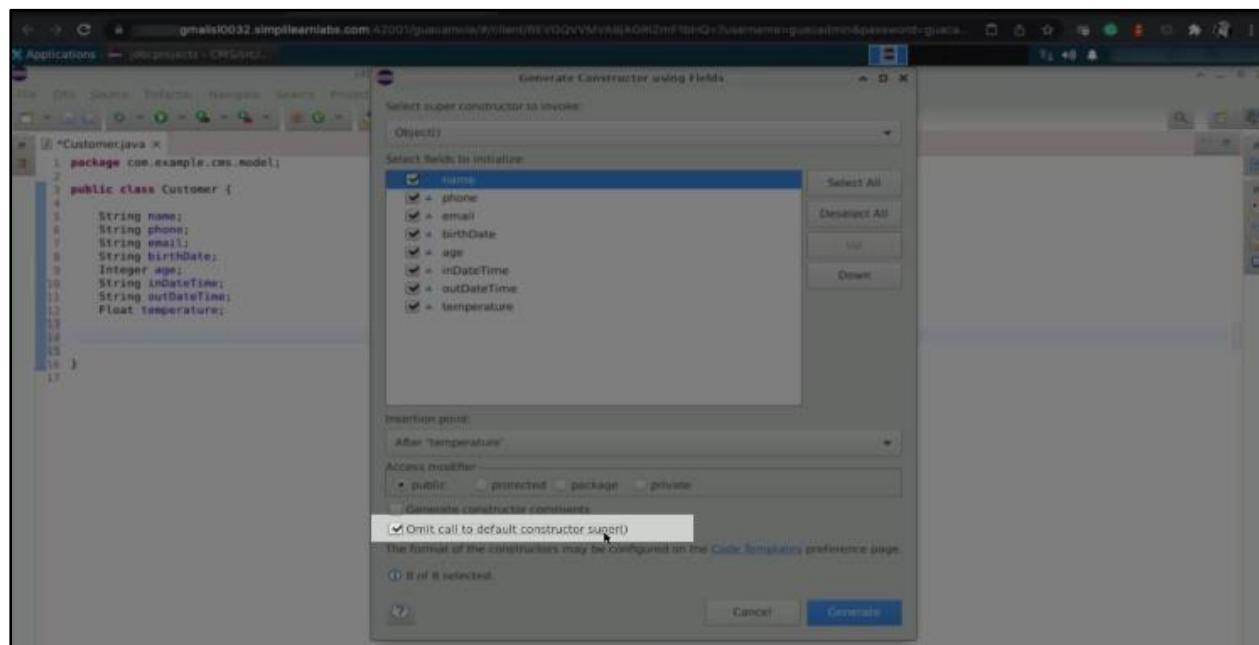


```
*Customer.java x
1 package com.example.cms.model;
2
3 public class Customer {
4     String name;
5     String phone;
6     String email;
7     String birthDate;
8     Integer age;
9     String registeredOn;
10    Float temperature;
11 }
12
13
```

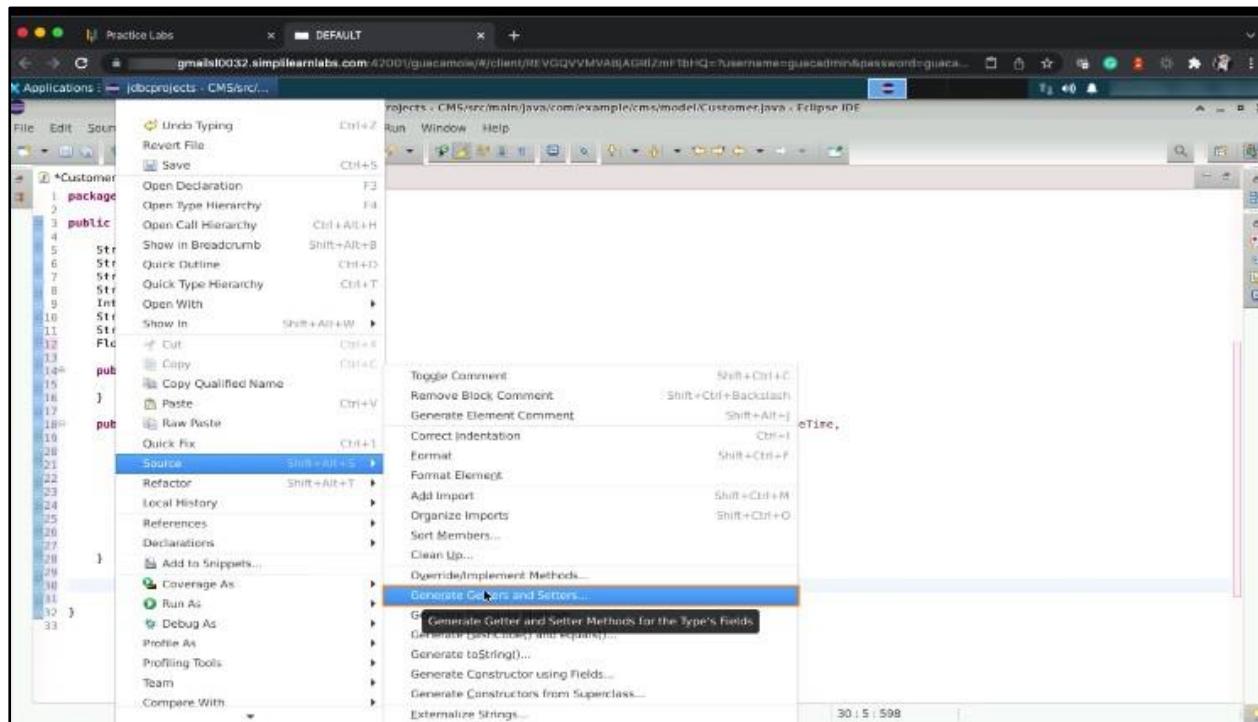
1.8 Right-click on class, select **Source**, and click on **Generate Constructor using Fields** to create a constructor



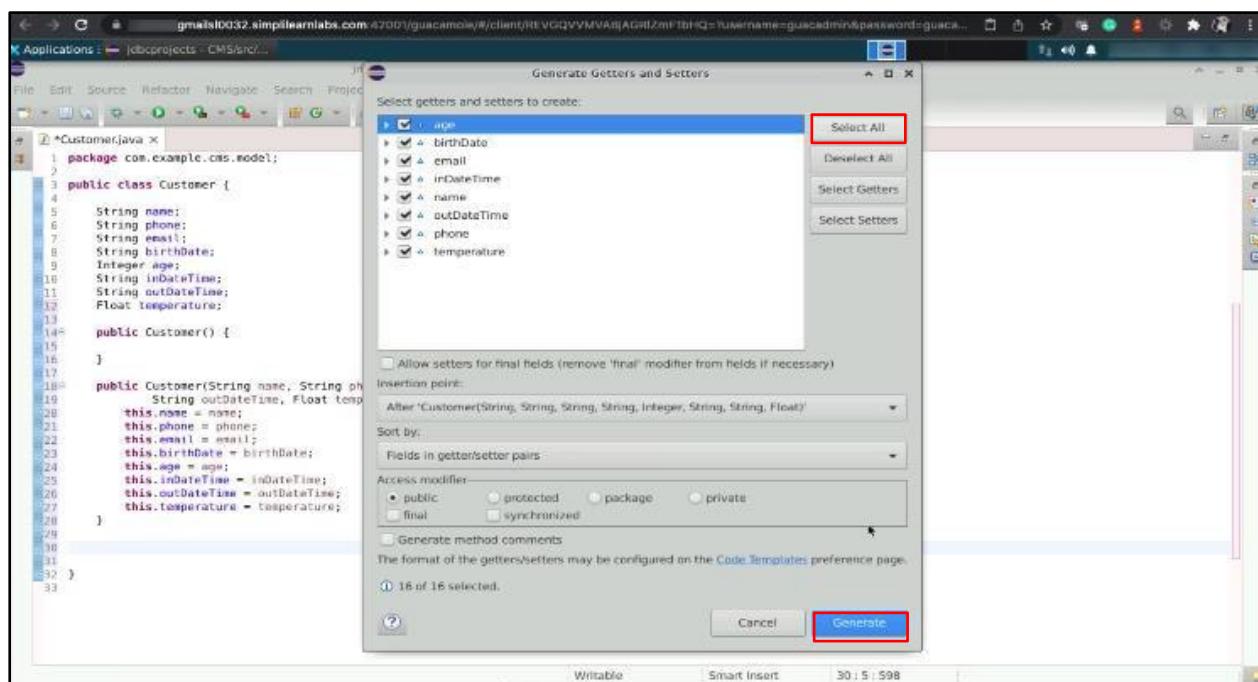
1.9 Mark the checkbox **Omit call to default constructor super()** and click on **Generate**. The constructor will be created.



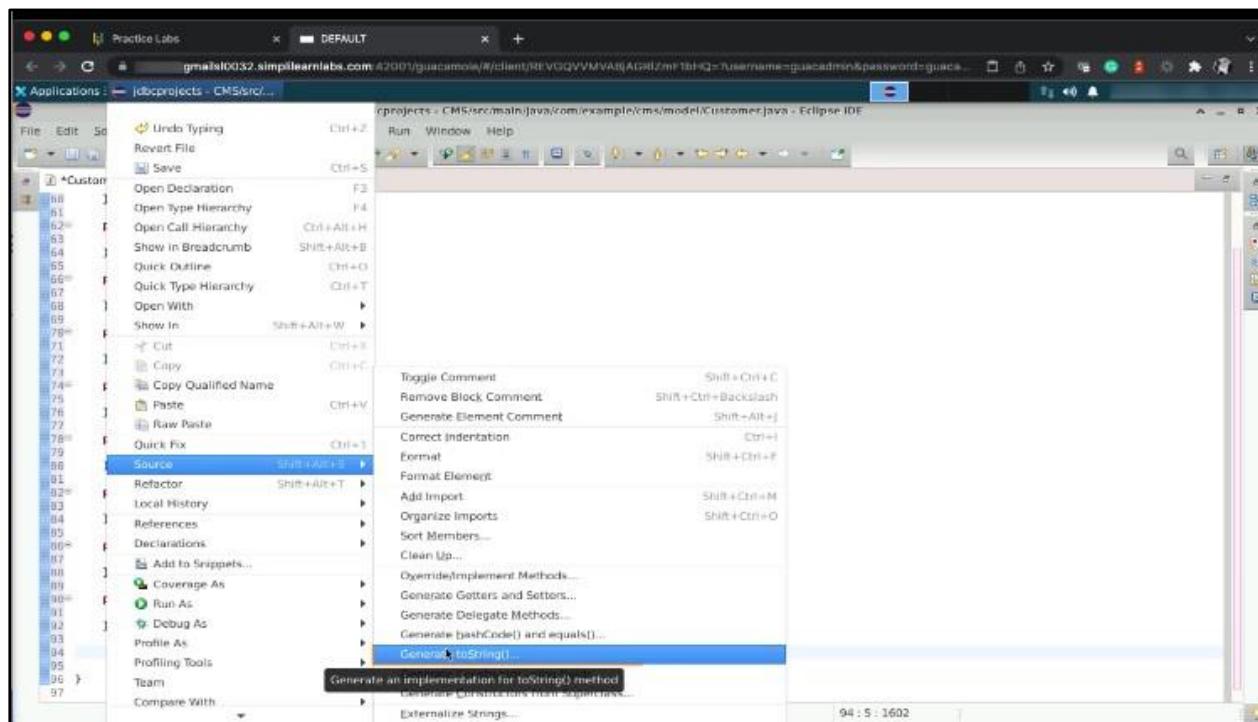
- 1.10 Right-click on class, select **Source**, and click on **Generate Getters and Setters** to write individual attributes



- 1.11 Click on **Select All** and then on **Generate**



1.12 Right-click **Source** and select **Generate toString()** to decide the location of data in the object as shown in the screenshot below:



Step 2: Write a customer object in App.java

2.1 Define a **Customer** object in the **App.java** file

```

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         Customer customer = new Customer();
16
17         System.out.println("Connecting to DB....");
18         DB db = new DB();
19         db.createConnection();
20         db.closeConnection();
21
22     }
23 }

```

The screenshot shows the Eclipse IDE code editor with an open file named 'App.java'. The code defines a class 'App' with a main method. Inside the main method, there is a comment block, a declaration of a 'Customer' object, a print statement, and database connection logic using a 'DB' class. The code editor interface, including tabs for 'Customer.java' and 'App.java', is visible at the top.

2.2 Set the name for the **Customer** object as shown in the screenshot below:

```
Customer.java  [App.java X]
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         customer.setName("John");
18
19         System.out.println("Connecting to DB....");
20         DB db = new DB();
21         db.createConnection();
22         db.closeConnection();
23     }
24
25 }
```

String literal is not properly closed by a double-quote

Writable Smart Insert

17 | 27 | 344

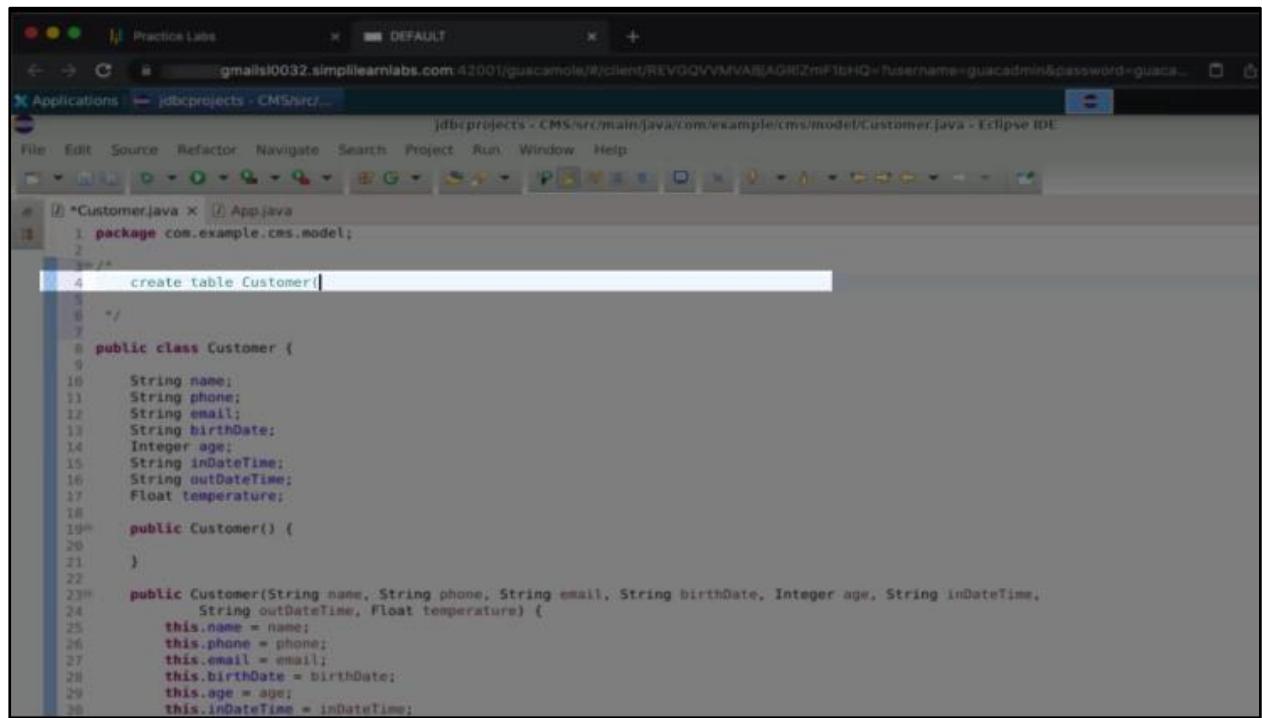
2.3 Create a method called **setPhone** with details as shown in the below screenshot:

```
Customer.java  [App.java X]
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         customer.setName("John");
18         customer.setPhone("+91 99999 1111");
19
20         System.out.println("Connecting to DB....");
21         DB db = new DB();
22         db.createConnection();
23         db.closeConnection();
24     }
25 }
```

Writable Smart Insert

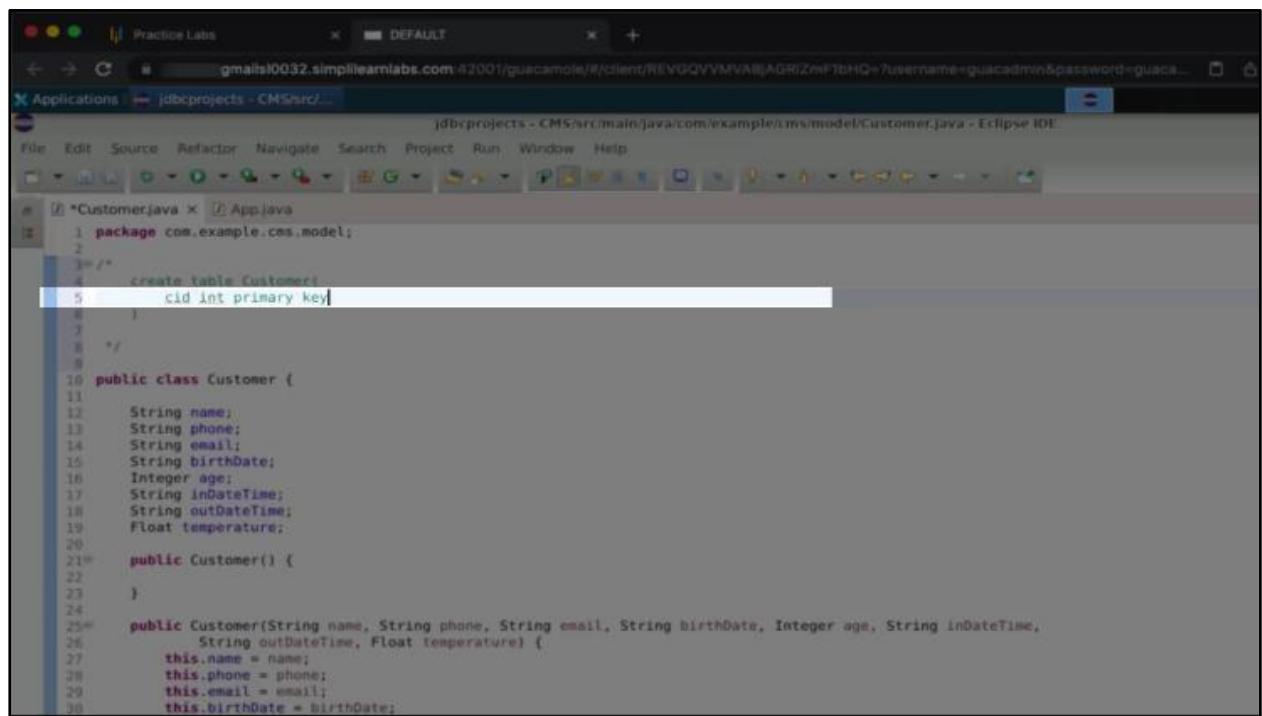
18 | 42 | 395

2.4 Use the **Customer** class and type **create table Customer** for object-relational mapping



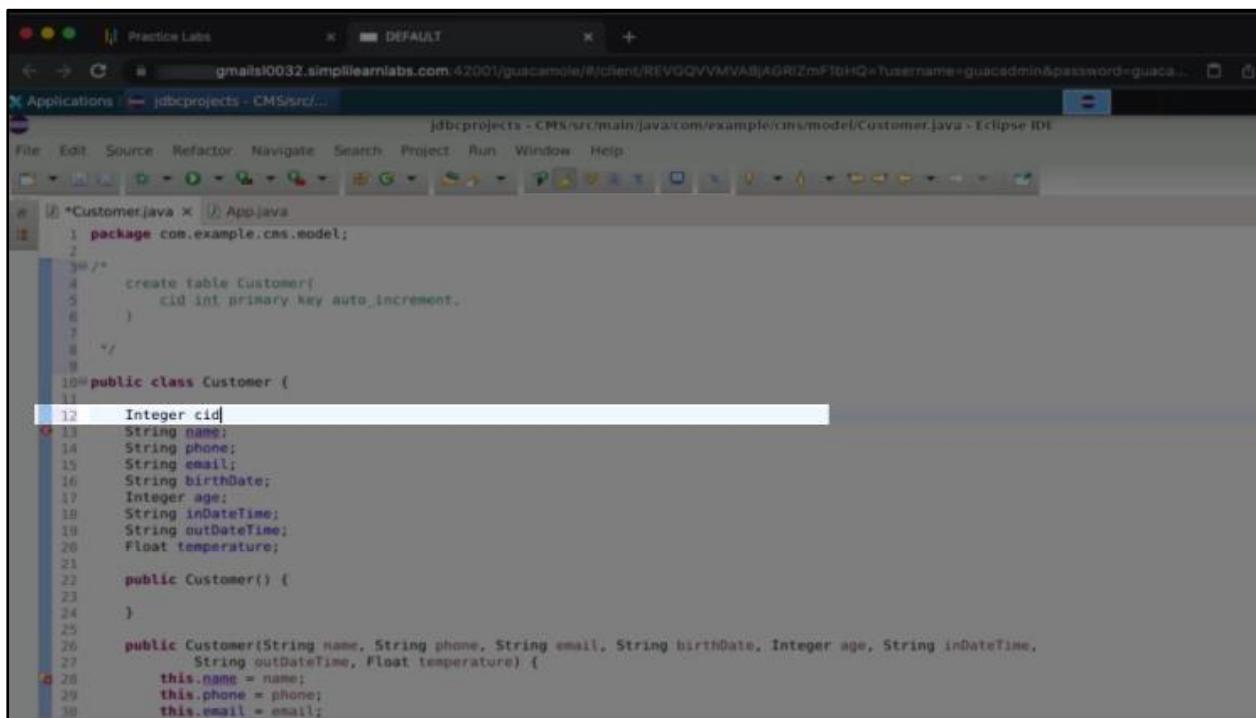
```
 1 package com.example.cms.model;
 2
 3 /**
 4  * create table Customer
 5  */
 6
 7
 8 public class Customer {
 9
10     String name;
11     String phone;
12     String email;
13     String birthDate;
14     Integer age;
15     String inDateTime;
16     String outDateTime;
17     Float temperature;
18
19     public Customer() {
20
21     }
22
23     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
24                     String outDateTime, Float temperature) {
25         this.name = name;
26         this.phone = phone;
27         this.email = email;
28         this.birthDate = birthDate;
29         this.age = age;
30         this.inDateTime = inDateTime;
31     }
32 }
```

2.5 Add a primary key as **cid** for a unique identifier as shown below:



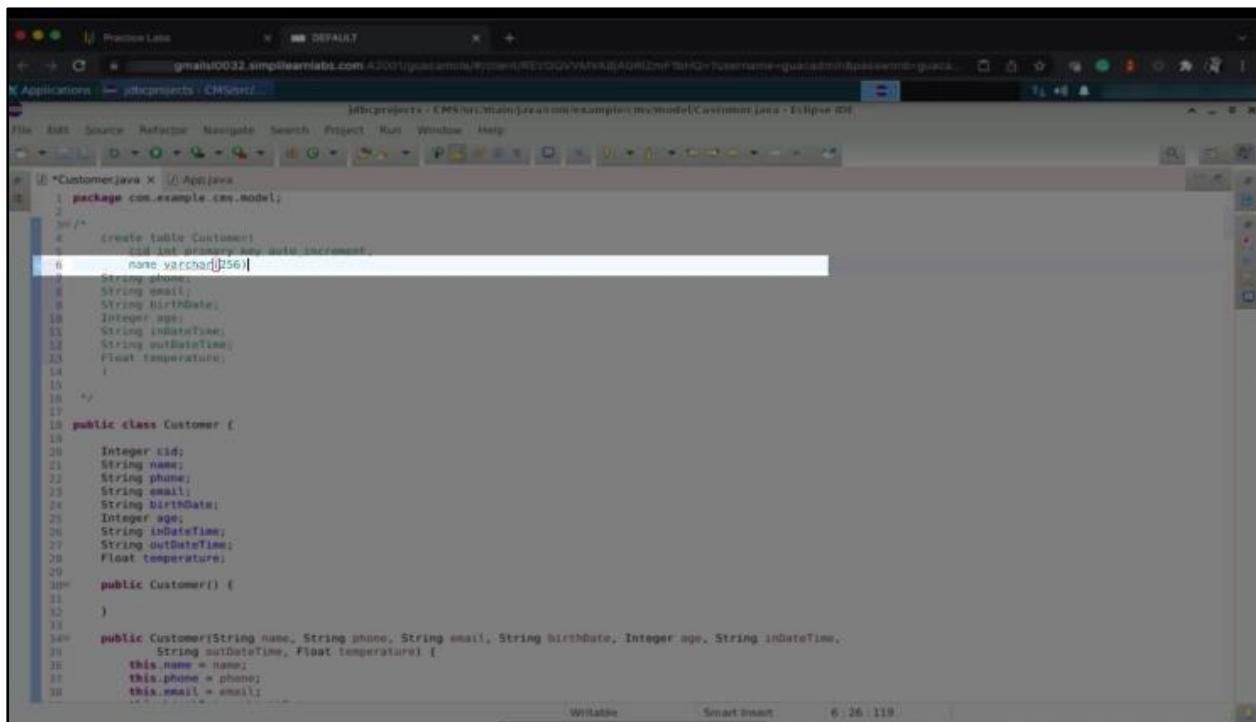
```
 1 package com.example.cms.model;
 2
 3 /**
 4  * create table Customer
 5  *     cid int primary key
 6  */
 7
 8
 9 public class Customer {
10
11     String name;
12     String phone;
13     String email;
14     String birthDate;
15     Integer age;
16     String inDateTime;
17     String outDateTime;
18     Float temperature;
19
20     public Customer() {
21
22     }
23
24     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
25                     String outDateTime, Float temperature) {
26         this.name = name;
27         this.phone = phone;
28         this.email = email;
29         this.birthDate = birthDate;
30     }
31 }
```

2.6 Add an integer attribute **cid** as shown in the below screenshot:



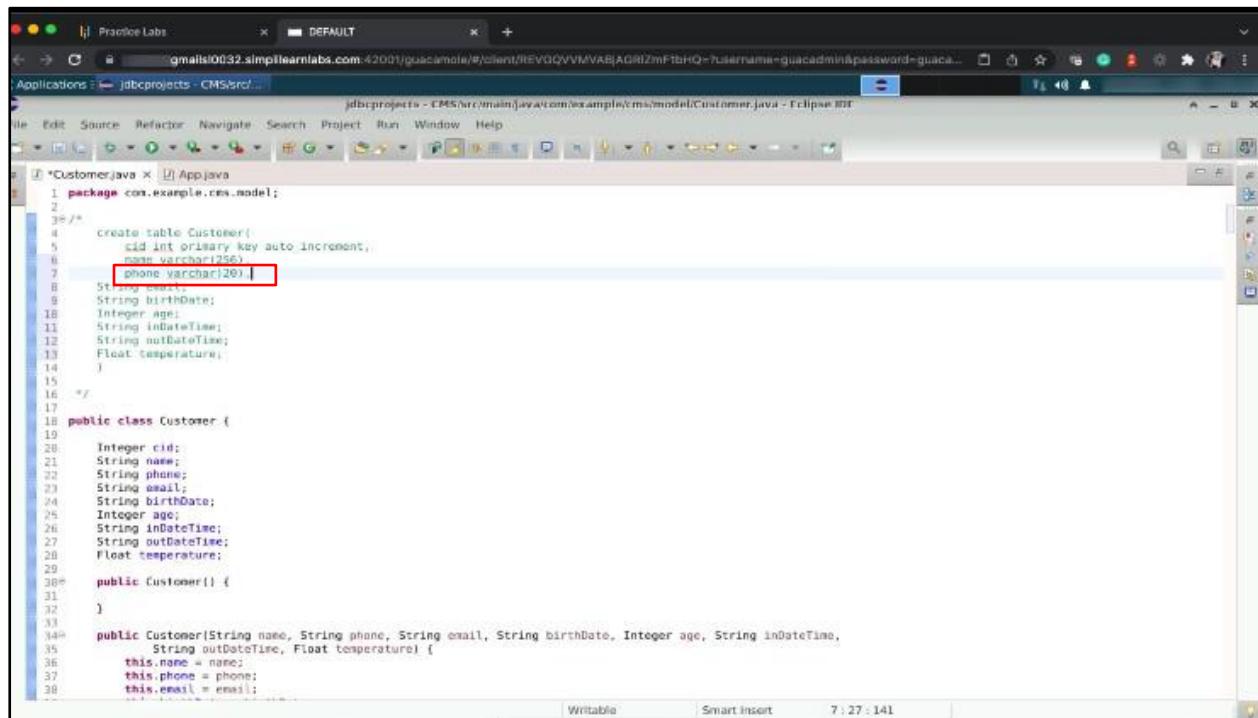
```
*Customer.java x App.java
1 package com.example.cms.model;
2
3 /**
4  * create table Customer(
5  * cid int primary key auto_increment,
6  * name varchar(256),
7  * phone varchar(256),
8  * email varchar(256),
9  * birthDate date,
10 * age int,
11 * inDateTime time,
12 * outDateTime time,
13 * temperature float
14 */
15
16 public class Customer {
17
18     Integer cid;
19     String name;
20     String phone;
21     String email;
22     String birthDate;
23     Integer age;
24     String inDateTime;
25     String outDateTime;
26     Float temperature;
27
28     public Customer() {
29
30     }
31
32     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
33                     String outDateTime, Float temperature) {
34         this.name = name;
35         this.phone = phone;
36         this.email = email;
37     }
38 }
```

2.7 Specify the **name** attribute with the datatype **varchar(256)** as shown in the below screenshot:



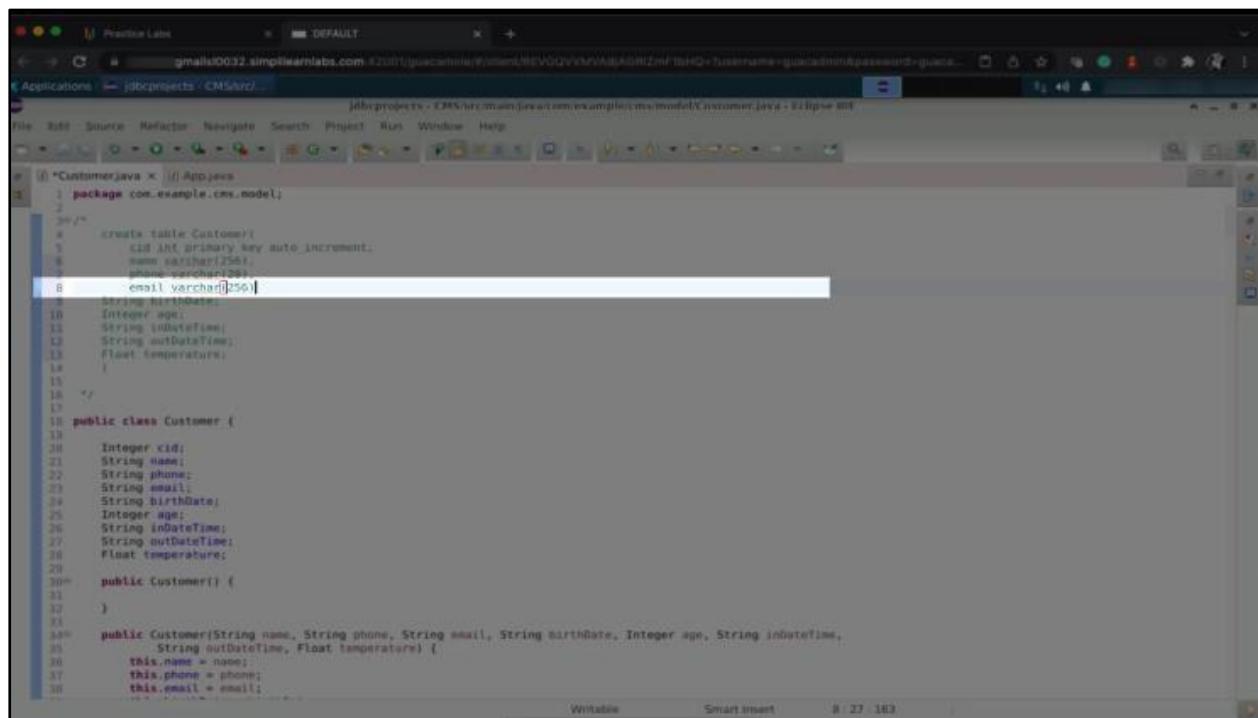
```
*Customer.java x App.java
1 package com.example.cms.model;
2
3 /**
4  * create table Customer(
5  * cid int primary key auto_increment,
6  * name varchar(256),
7  * phone varchar(256),
8  * email varchar(256),
9  * birthDate date,
10 * age int,
11 * inDateTime time,
12 * outDateTime time,
13 * temperature float
14 */
15
16 public class Customer {
17
18     Integer cid;
19     String name;
20     String phone;
21     String email;
22     String birthDate;
23     Integer age;
24     String inDateTime;
25     String outDateTime;
26     Float temperature;
27
28     public Customer() {
29
30     }
31
32     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
33                     String outDateTime, Float temperature) {
34         this.name = name;
35         this.phone = phone;
36         this.email = email;
37     }
38 }
```

2.8 Add the **phone** attribute with the datatype **varchar(20)** as shown in the below screenshot:



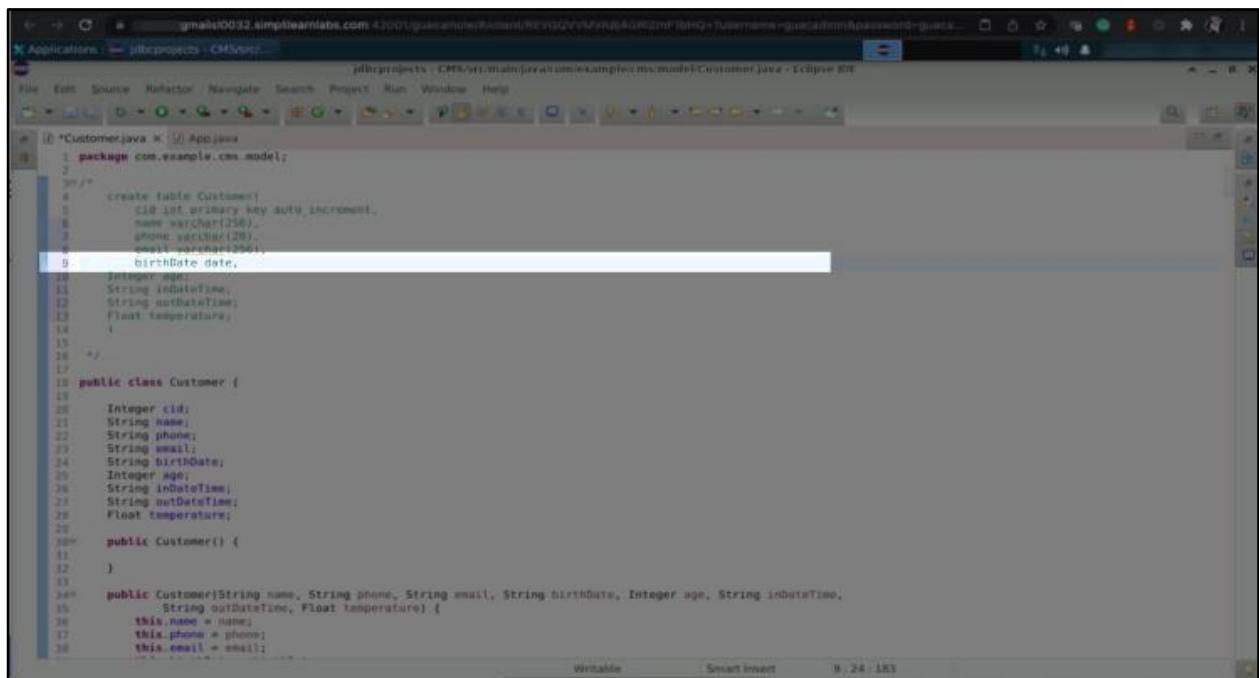
```
Customer.java
1 package com.example.cms.model;
2
3 /**
4  * create table Customer(
5  *     cid int primary key auto_increment,
6  *     name varchar(256),
7  *     phone varchar(20),
8  *     String birthDate;
9  *     Integer age;
10 *     String inDateTime;
11 *     String outDateTime;
12 *     Float temperature;
13 * );
14 */
15
16
17 public class Customer {
18
19     Integer cid;
20     String name;
21     String phone;
22     String email;
23     String birthDate;
24     Integer age;
25     String inDateTime;
26     String outDateTime;
27     Float temperature;
28
29     public Customer() {
30
31     }
32
33
34     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
35                     String outDateTime, Float temperature) {
36         this.name = name;
37         this.phone = phone;
38         this.email = email;
39     }
40 }
```

2.9 Add the **email** attribute with the datatype **varchar(256)** as shown in the below screenshot:



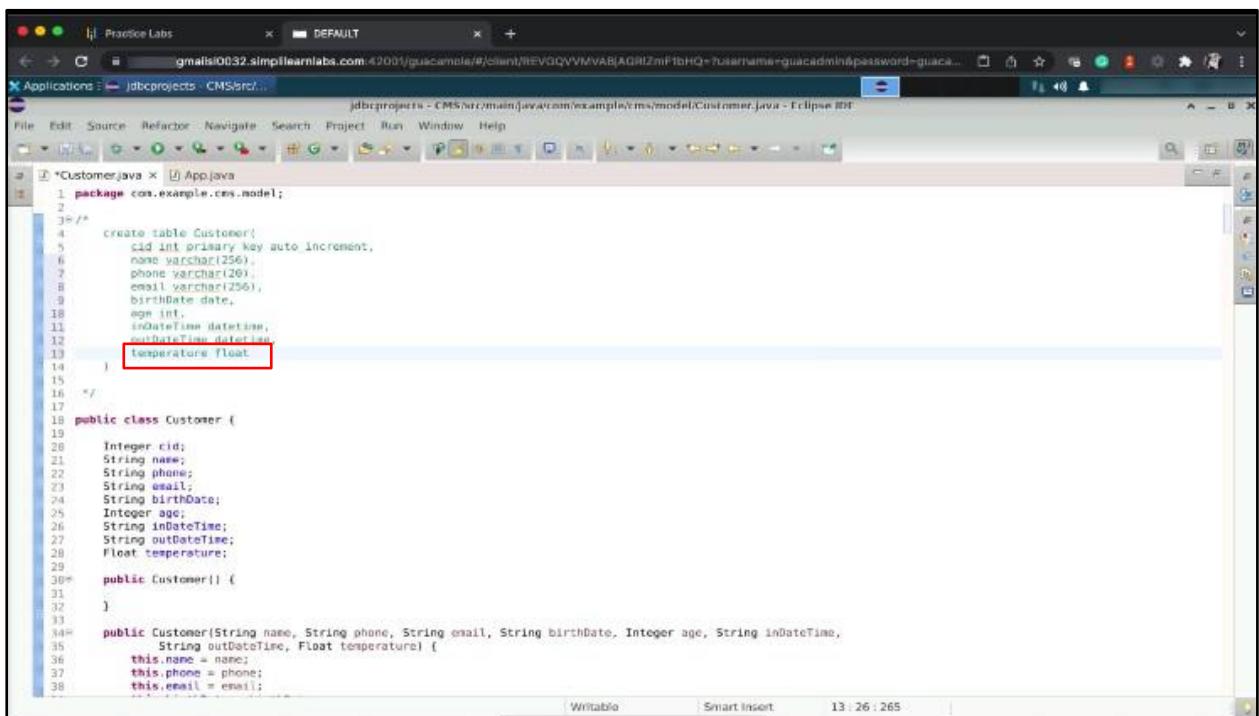
```
Customer.java
1 package com.example.cms.model;
2
3 /**
4  * create table Customer(
5  *     cid int primary key auto_increment,
6  *     name varchar(256),
7  *     phone varchar(20),
8  *     birthDate,
9  *     Integer age;
10 *     String inDateTime;
11 *     String outDateTime;
12 *     Float temperature;
13 * );
14 */
15
16
17 public class Customer {
18
19     Integer cid;
20     String name;
21     String phone;
22     String email;
23     String birthDate;
24     Integer age;
25     String inDateTime;
26     String outDateTime;
27     Float temperature;
28
29     public Customer() {
30
31     }
32
33
34     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
35                     String outDateTime, Float temperature) {
36         this.name = name;
37         this.phone = phone;
38         this.email = email;
39     }
40 }
```

2.10 Set the **birthday** attribute with the datatype **date** as shown in the screenshot below:



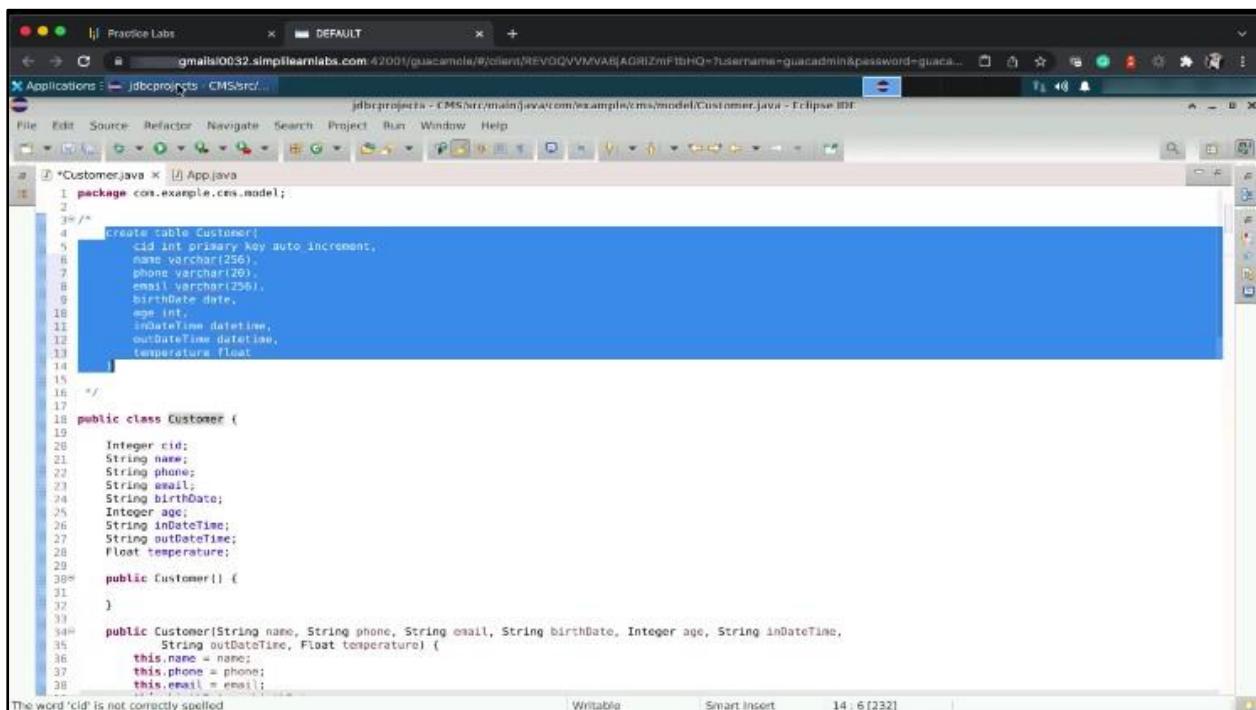
```
*Customer.java X [ ] AppJava
1 package com.example.cms.model;
2
3/*
4  * create table Customer(
5  *     cid int primary key auto increment,
6  *     name varchar(256),
7  *     phone varchar(20),
8  *     email varchar(256),
9  *     birthDate date,
10 *     age int,
11 *     inDateTime,
12 *     outDateTime,
13 *     float temperature;
14 *
15 */
16
17 public class Customer {
18
19     Integer cid;
20     String name;
21     String phone;
22     String email;
23     String birthDate;
24     Integer age;
25     String inDateTime;
26     String outDateTime;
27     Float temperature;
28
29     public Customer() {
30
31     }
32
33     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
34     String outDateTime, Float temperature) {
35         this.name = name;
36         this.phone = phone;
37         this.email = email;
38     }
39 }
```

2.11 Add a floating-point attribute for **temperature** and a datetime attribute for **outDataTime** as shown in the screenshot below:



```
*Customer.java X [ ] AppJava
1 package com.example.cms.model;
2
3/*
4  * create table Customer(
5  *     cid int primary key auto increment,
6  *     name varchar(256),
7  *     phone varchar(20),
8  *     email varchar(256),
9  *     birthDate date,
10 *     age int,
11 *     inDateTime datetime,
12 *     outDateTime datetime,
13 *     temperature float
14 *)
15
16 */
17
18 public class Customer {
19
20     Integer cid;
21     String name;
22     String phone;
23     String email;
24     String birthDate;
25     Integer age;
26     String inDateTime;
27     String outDateTime;
28     Float temperature;
29
30     public Customer() {
31
32     }
33
34     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
35     String outDateTime, Float temperature) {
36         this.name = name;
37         this.phone = phone;
38         this.email = email;
39     }
40 }
```

2.12 Copy the entire Customer table query

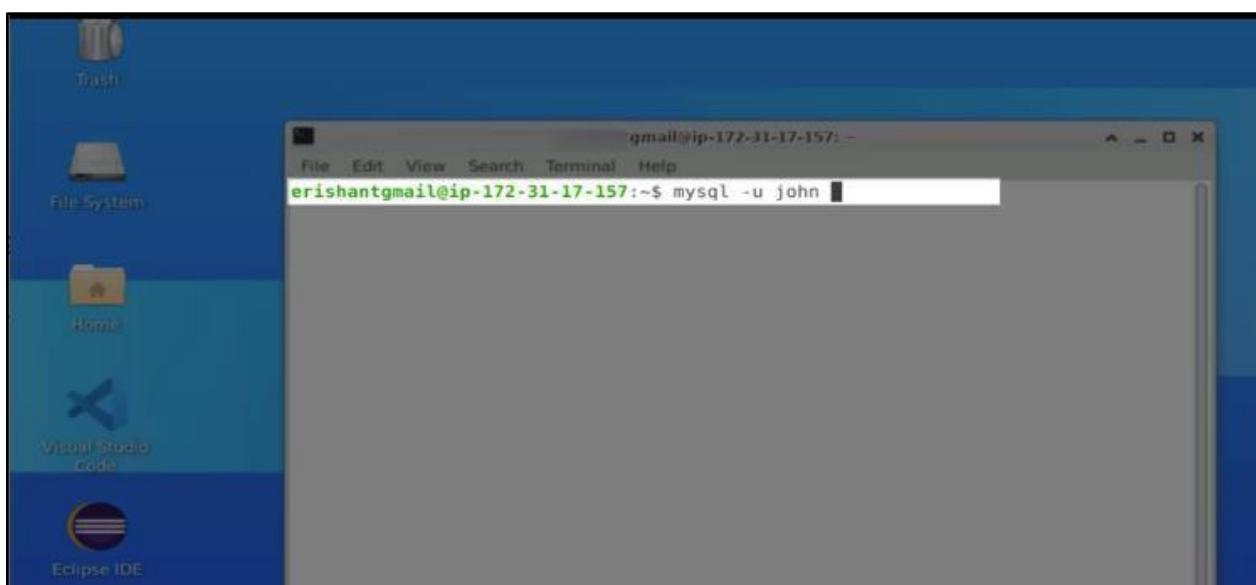


The screenshot shows the Eclipse IDE interface with the Customer.java file open. The code defines a Customer class with attributes cid, name, phone, email, birthDate, age, inDateTime, outDateTime, and temperature. It also includes a constructor and a copy constructor.

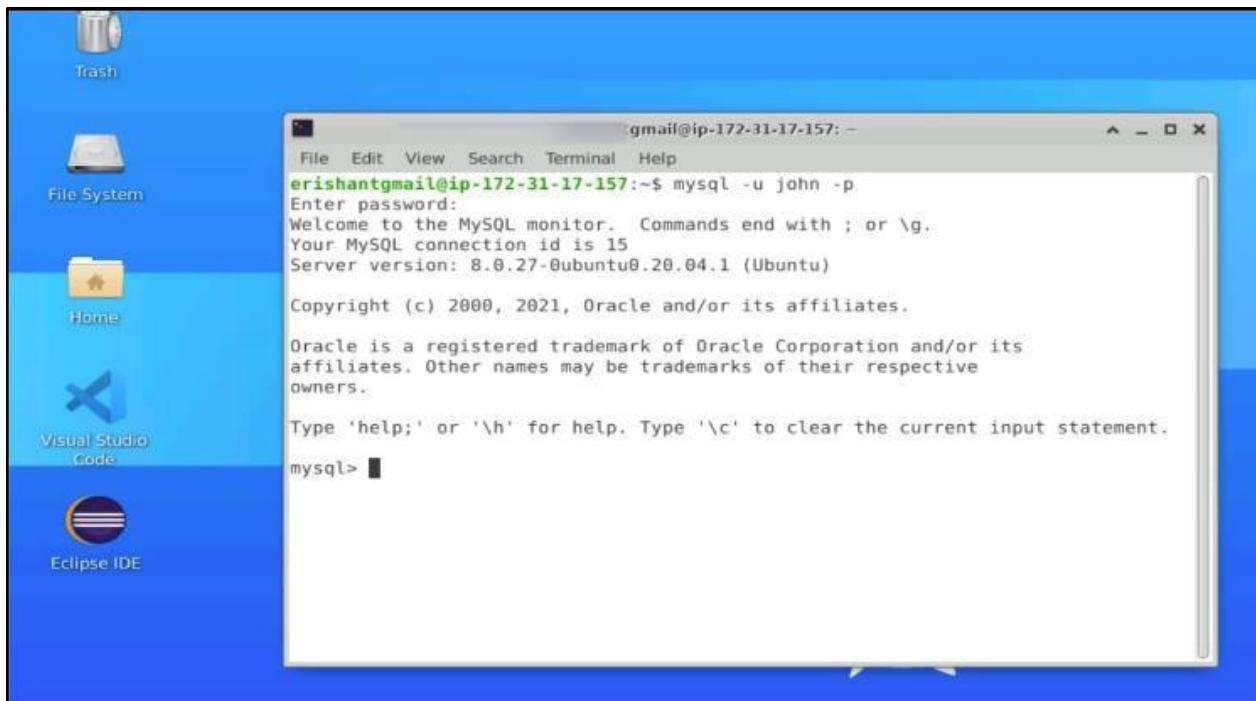
```
1 package com.example.cms.model;
2
3 /**
4  * Create table Customer!
5  * cid int primary key auto_increment,
6  * name varchar(256),
7  * phone varchar(20),
8  * email varchar(256),
9  * birthDate date,
10 * age int,
11 * inDateTime datetime,
12 * outDateTime datetime,
13 * temperature float
14 */
15
16
17 public class Customer {
18
19     Integer cid;
20     String name;
21     String phone;
22     String email;
23     String birthDate;
24     Integer age;
25     String inDateTime;
26     String outDateTime;
27     Float temperature;
28
29     public Customer() {
30
31     }
32
33
34     public Customer(String name, String phone, String email, String birthDate, Integer age, String inDateTime,
35                     String outDateTime, Float temperature) {
36         this.name = name;
37         this.phone = phone;
38         this.email = email;
39         this.birthDate = birthDate;
40         this.age = age;
41         this.inDateTime = inDateTime;
42         this.outDateTime = outDateTime;
43         this.temperature = temperature;
44     }
45 }
```

Step 3: Open the terminal window

3.1 Open the terminal window and log in to MySQL using the command **mysql -u john -p**

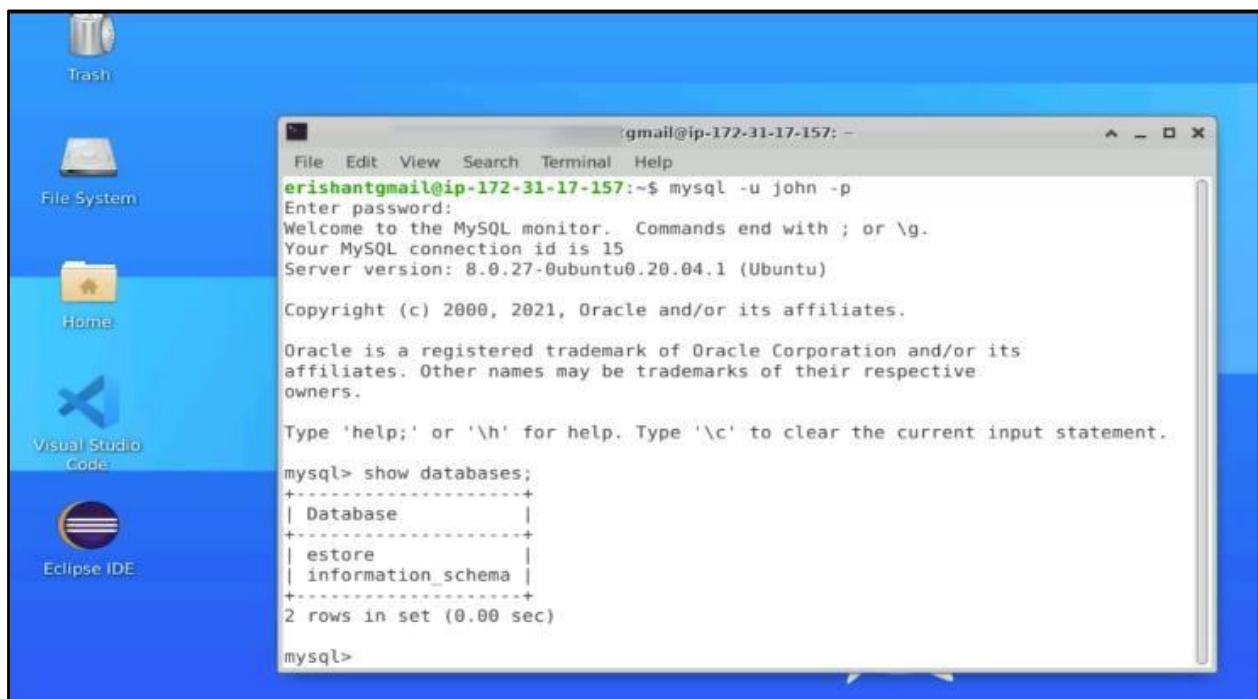


3.2 Enter the password

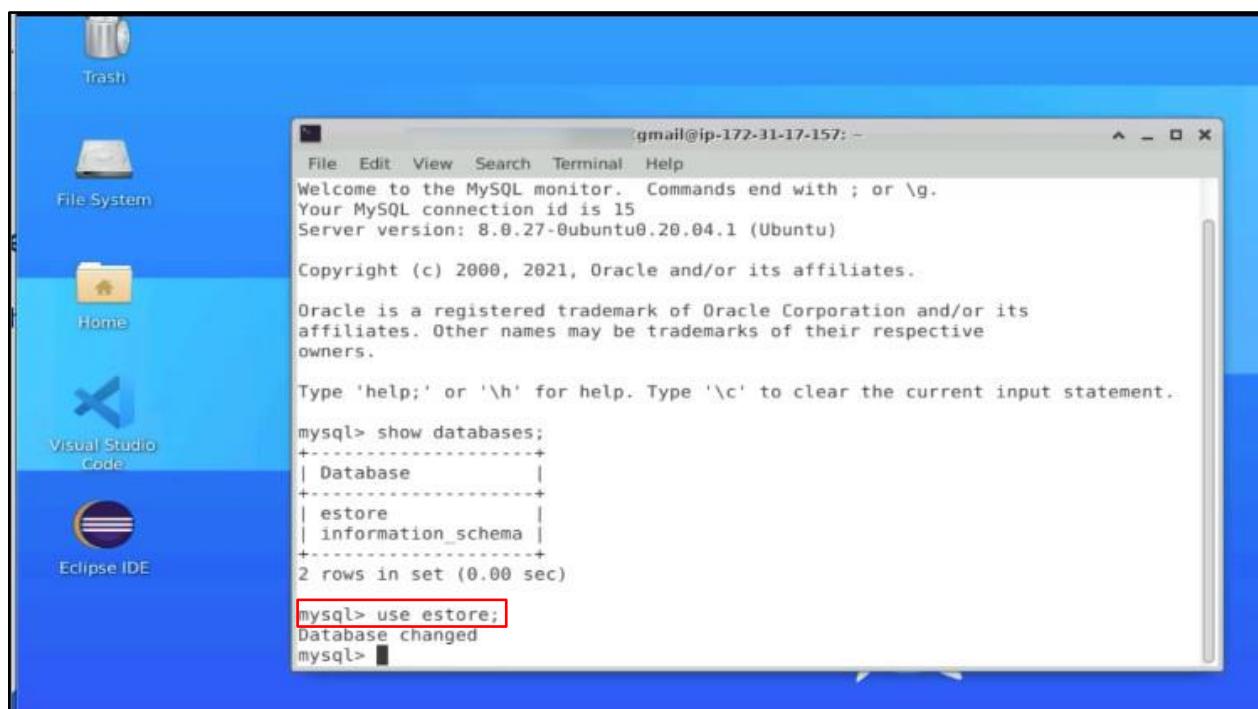


Note: A user named **John** has already been created for the database.

3.3 Type the **show databases;** command to check the available database names as shown in the screenshot below:



3.4 Enter **use estore;** to work on the estore databases shown in the screenshot below:



The screenshot shows a Linux desktop environment with a blue-themed desktop. 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 "gmail@ip-172-31-17-157: ~" is open, displaying the MySQL monitor. The terminal output is as follows:

```
File Edit View Search Terminal Help
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

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

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

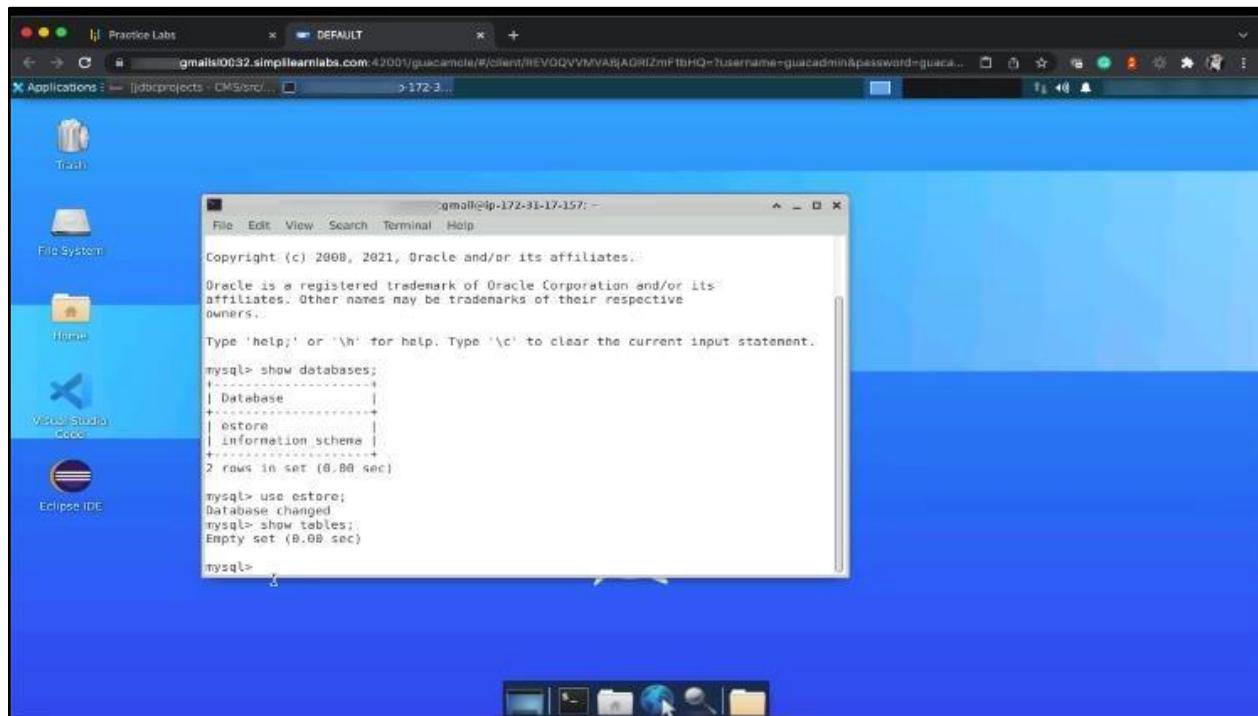
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| estore   |
| information_schema |
+-----+
2 rows in set (0.00 sec)

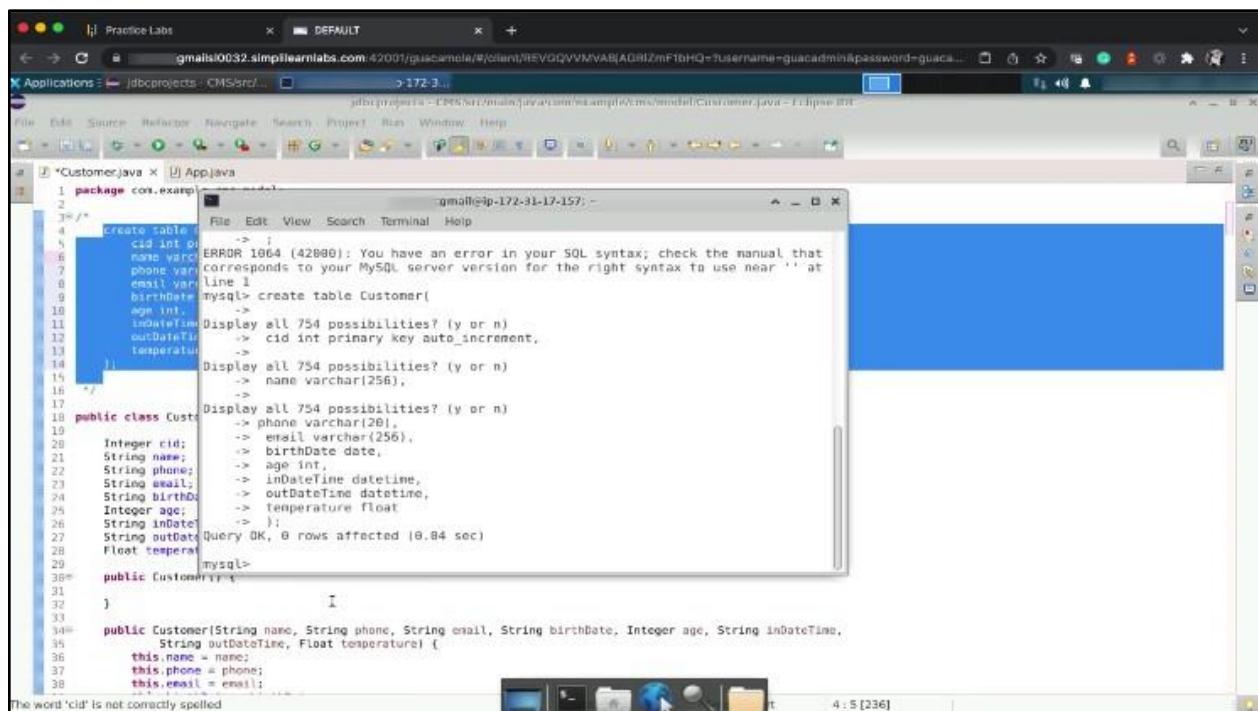
mysql> use estore;
Database changed
mysql> █
```

The command `use estore;` is highlighted with a red box.

3.5 Enter **show tables;** to check tables available in the estore database



3.6 Paste the **Customer** table copied in step 2.12



3.7 Run the **show tables;** command to see the changes

The screenshot shows the Eclipse IDE interface with the Customer.java file open. In the terminal window, the user runs the 'show tables;' command, which returns a single row: 'Customer'. This indicates that the table has been successfully created.

```

Customer.java x App.java
1 package com.example;
2
3 /*
4  * create table
5  * cid int primary key
6  * name varchar(256)
7  * phone varchar(20)
8  * email varchar(256)
9  * birthDate date
10 * age int,
11 * inDateTime datetime,
12 * outDateTime datetime,
13 * temperature float
14 */
15
16 public class Customer {
17
18     Integer cid;
19     String name;
20     String phone;
21     String email;
22     String birthDate;
23     Integer age;
24     String inDate;
25     String outDate;
26     Float temperat
27
28     public Customer() {
29
30     }
31
32 }

```

mysql> show tables;

Tables_in_estore
Customer

1 row in set (0.00 sec)

mysql>

3.8 Enter the **describe Customer;** command to view the customer table details

The screenshot shows the Eclipse IDE interface with the Customer.java file open. In the terminal window, the user runs the 'describe Customer;' command, which displays the table structure:

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>

3.9 Enter **select * from Customer;** to check the data in the table

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor window titled "Customer.java" containing Java code. The code includes a MySQL create table statement and a Java class definition for a Customer object with fields cid, name, phone, email, birthDate, age, inDateTime, outDateTime, and temperature. On the right, there is a terminal window titled "mysql>" showing MySQL commands and their results. The commands include "describe Customer;", "select * from Customer;", and "Empty set (0.00 sec)". The terminal output also shows "1 row in set (0.00 sec)" and "9 rows in set (0.00 sec)".

```
Customer.java
package com.example;
create table Customer (
    cid int primary key,
    name varchar(256),
    phone varchar(20),
    email varchar(256),
    birthDate date,
    age int,
    inDateTime datetime,
    outDateTime datetime,
    temperature float
);
public class Customer {
    Integer cid;
    String name;
    String phone;
    String email;
    String birthDate;
    Integer age;
    String inDateTime;
    String outDateTime;
    Float temperature;
}
public Customer() { }

App.java
```

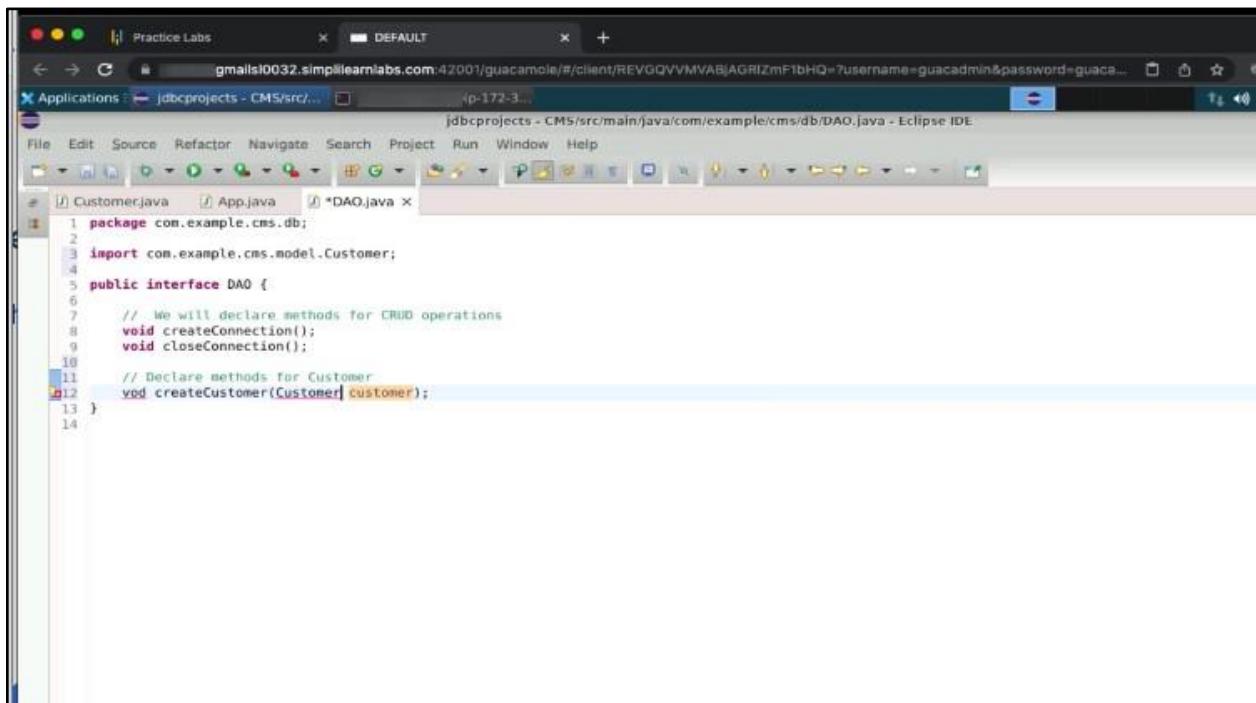
```
mysql> describe Customer;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| cid   | int    | NO   | PRI | NULL    | auto_increment |
| name  | varchar(256) | YES  |     | NULL    |             |
| phone | varchar(20)  | YES  |     | 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>
```

Step 4: Declare a method for a customer in the DAO interface

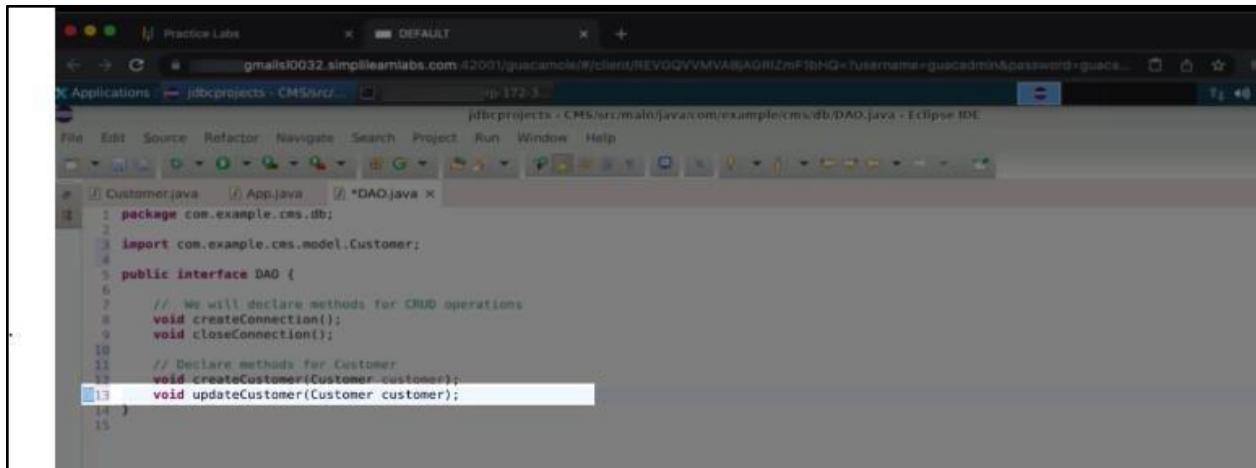
4.1 Declare methods for a customer in the DAO interface. The first one is **createCustomer()** that takes a single customer as input.



The screenshot shows the Eclipse IDE interface with the DAO.java file open in the editor. The code defines a public interface DAO with methods for CRUD operations and a specific method for creating a customer.

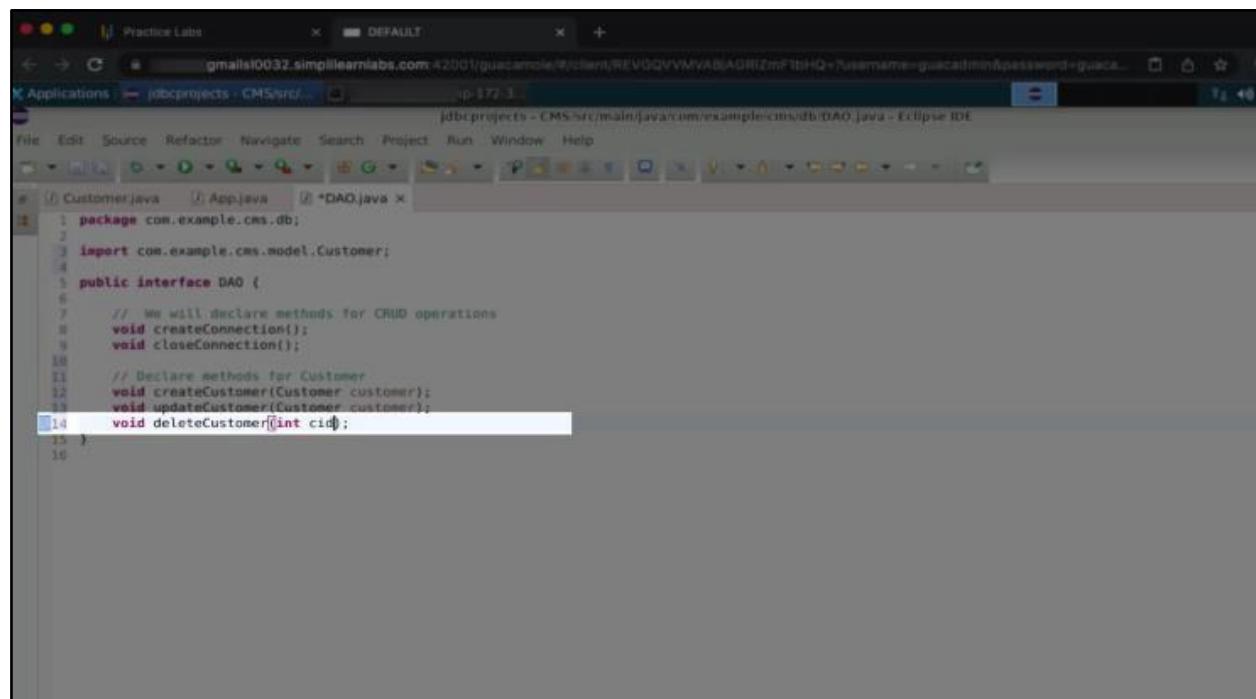
```
1 package com.example.cms.db;
2
3 import com.example.cms.model.Customer;
4
5 public interface DAO {
6
7     // We will declare methods for CRUD operations
8     void createConnection();
9     void closeConnection();
10
11    // Declare methods for Customer
12    void createCustomer(Customer customer);
13}
```

4.2 Additionally, implement **updateCustomer()** which also takes a customer as input for making updates



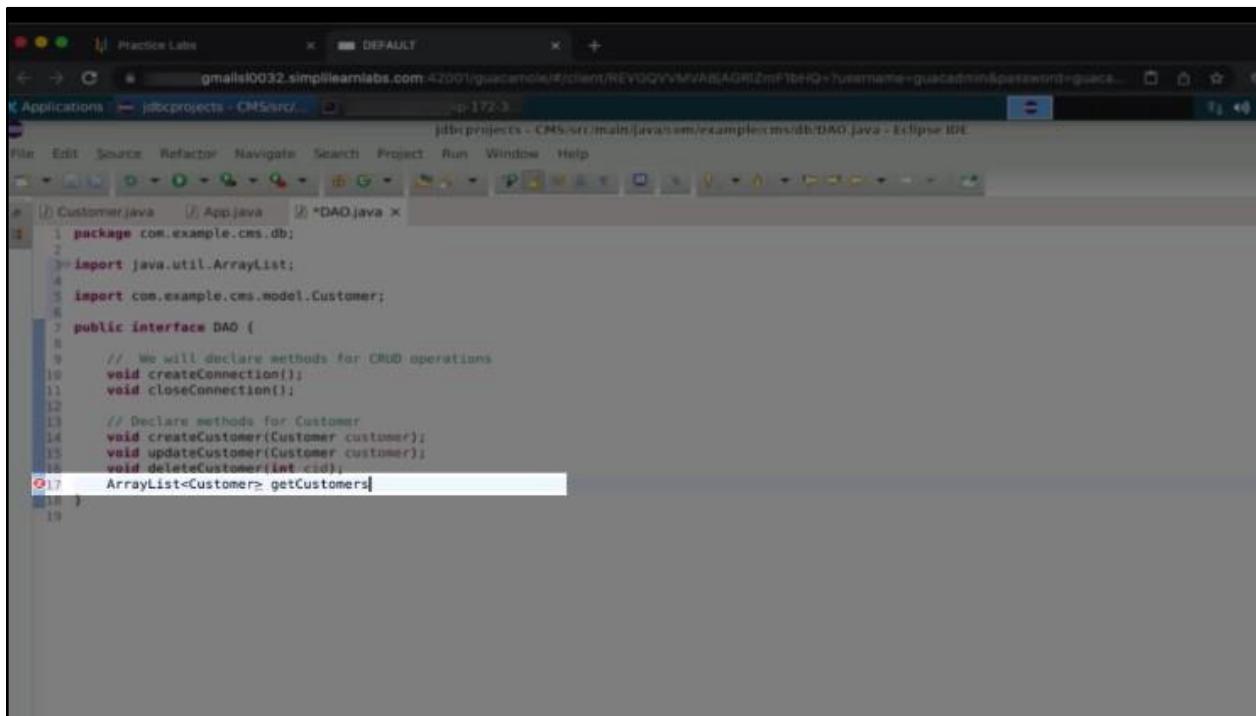
```
1 package com.example.cms.db;
2
3 import com.example.cms.model.Customer;
4
5 public interface DAO {
6
7     // We will declare methods for CRUD operations
8     void createConnection();
9     void closeConnection();
10
11    // Declare methods for Customer
12    void createCustomer(Customer customer);
13    void updateCustomer(Customer customer);
14 }
```

4.3 Create **deleteCustomer()**, which takes a single input **cid**



```
1 package com.example.cms.db;
2
3 import com.example.cms.model.Customer;
4
5 public interface DAO {
6
7     // We will declare methods for CRUD operations
8     void createConnection();
9     void closeConnection();
10
11    // Declare methods for Customer
12    void createCustomer(Customer customer);
13    void updateCustomer(Customer customer);
14    void deleteCustomer(int cid);
15 }
```

4.4 Implement the `getAllCustomers()` method to retrieve the list of customers



The screenshot shows the Eclipse IDE interface with the DAO.java file open in the editor. The code implements a DAO interface for managing Customer objects.

```
1 package com.example.cms.db;
2
3 import java.util.ArrayList;
4
5 import com.example.cms.model.Customer;
6
7 public interface DAO {
8
9     // We will declare methods for CRUD operations
10    void createConnection();
11    void closeConnection();
12
13    // Declare methods for Customer
14    void createCustomer(Customer customer);
15    void updateCustomer(Customer customer);
16    void deleteCustomer(int cid);
17    ArrayList<Customer> getCustomers();
18
19}
```

4.5 Navigate to the **DB.java** file where you can see the error, and click on the error to add the unimplemented methods

The screenshot shows the Eclipse IDE interface with the title bar "Practice Labs" and "DEFAULT". The URL in the browser is "gmails10032.simplilearnlabs.com:42001/guacamole/#/client/REVGQVVVMVABjAGRIZmF1bHQ=?username=guacadmin&password=guaca...". The project "jobprojects - CMS/src..." is selected. The current file is "DB.java". The code editor contains the following Java code:

```
6  /*
7   * JDBC Procedure:
8   * 1. Download the Driver and Link it to the Project. For Maven Project simply configure the dependency in the pom.xml file
9   * 2. Load the Driver from the library i.e. jar file
10  * 3. Create Connection to the DataBase with url, user and password
11  * 4. Execute CRUD operations
12  * 5. Close the Connection
13  */
14 
15 public class DB implements DAO{
16     Connection con;
17     final String finalStr;
18     final String finalStr;
19     final String finalStr;
20     final String finalStr;
21     public DB() {
22         try {
23             Class.forName("com.mysql.jdbc.Driver");
24         } catch (Exception e) {
25             e.printStackTrace();
26         }
27     }
28 
29     public void connect() {
30         try {
31             String user = "john";
32             String password = "john";
33             String url = "jdbc:mysql://localhost/estore";
34             connection = DriverManager.getConnection(url, user, password);
35         } catch (Exception e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

A code completion tooltip is displayed over the line "public class DB implements DAO{". The tooltip has two sections: "Add unimplemented methods" and "4 methods to implement:".

- Add unimplemented methods
 - Create new JUnit test case for 'DB.java'
 - Make type 'DB' abstract
 - Rename in file (Ctrl+2 R)
 - Rename in workspace (Shift+Alt+R)
- 4 methods to implement:
 - com.example.cms.db.DAO.createCustomer(...)
 - com.example.cms.db.DAO.updateCustomer(...)
 - com.example.cms.db.DAO.deleteCustomer(...)
 - com.example.cms.db.DAO.getAllCustomers()

By following these steps, you have successfully created a model named **Customer** and established a corresponding table in the database using ORM. Additionally, we have implemented methods following the DAO design pattern to perform CRUD operations on the data.