Lesson 01 Demo 06

Implementing Transaction Management

Objective: To implement transaction management to execute all the SQL statements together, ensuring data integrity and consistency within the database

Tool required: Eclipse IDE

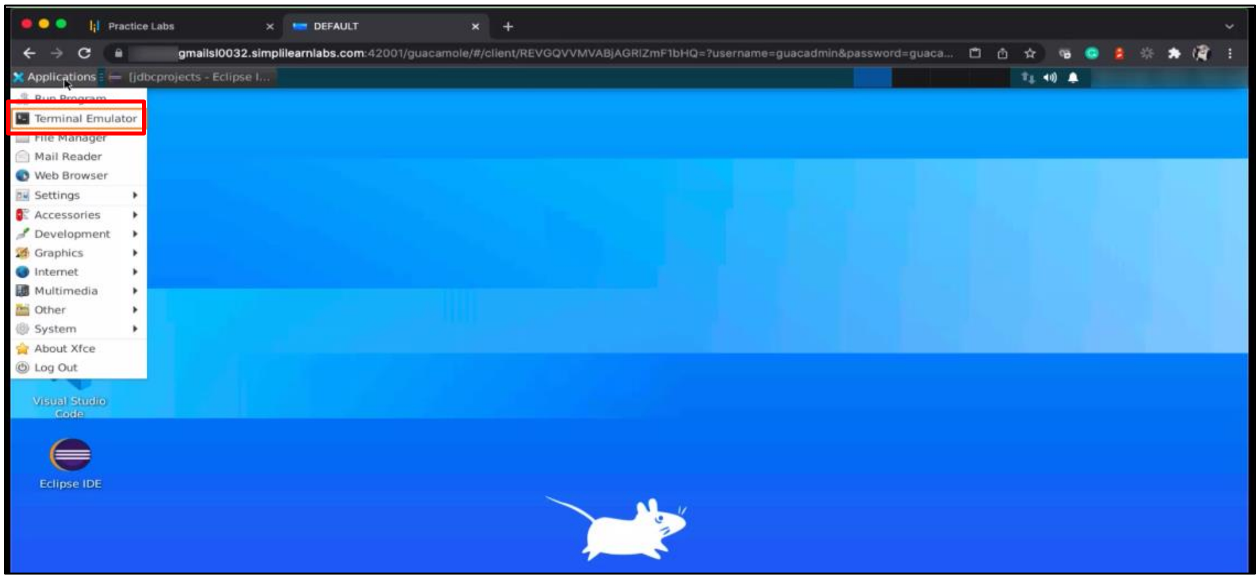
Prerequisites: None

Steps to be followed:

1. Create new table name orders
2. Create the executionTransaction method
3. Call the created method and checking the output
4. Use the auto-commit feature
5. Use insert and update operations
6. Use the delete operation

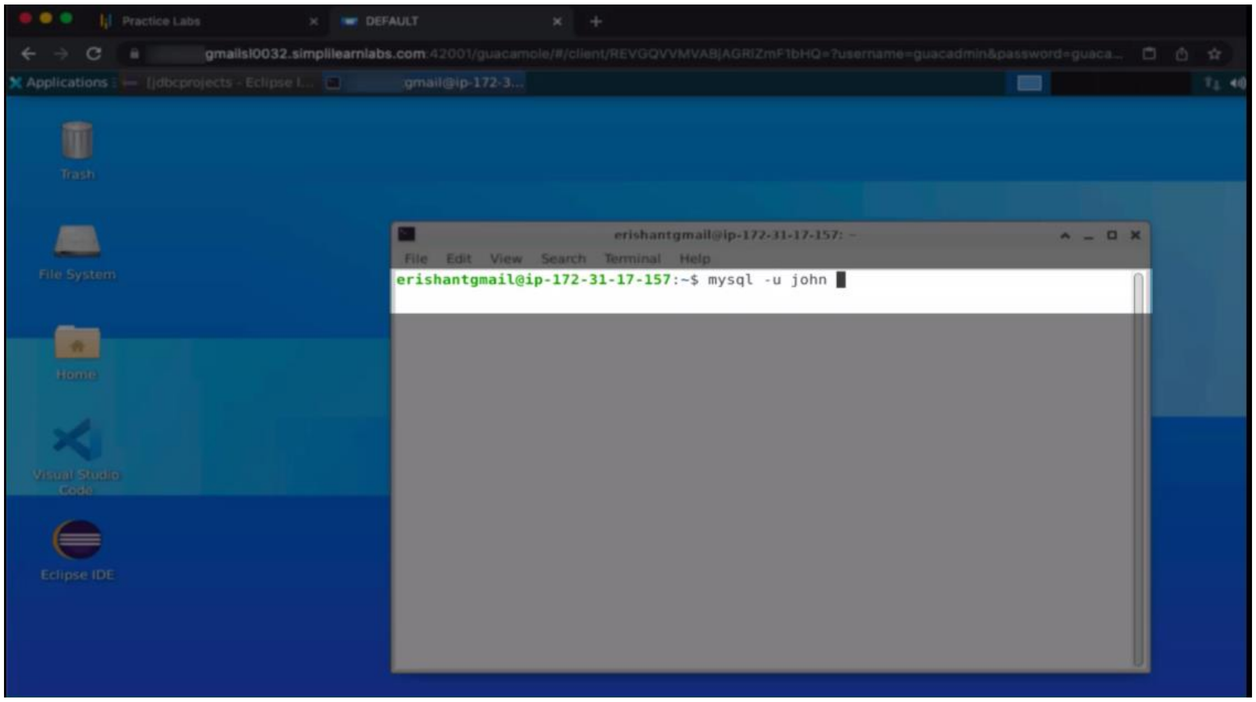
Step 1: Create new table name orders

1.1 Open the **Terminal Emulator**



1.2 Login to MySQL using the command:

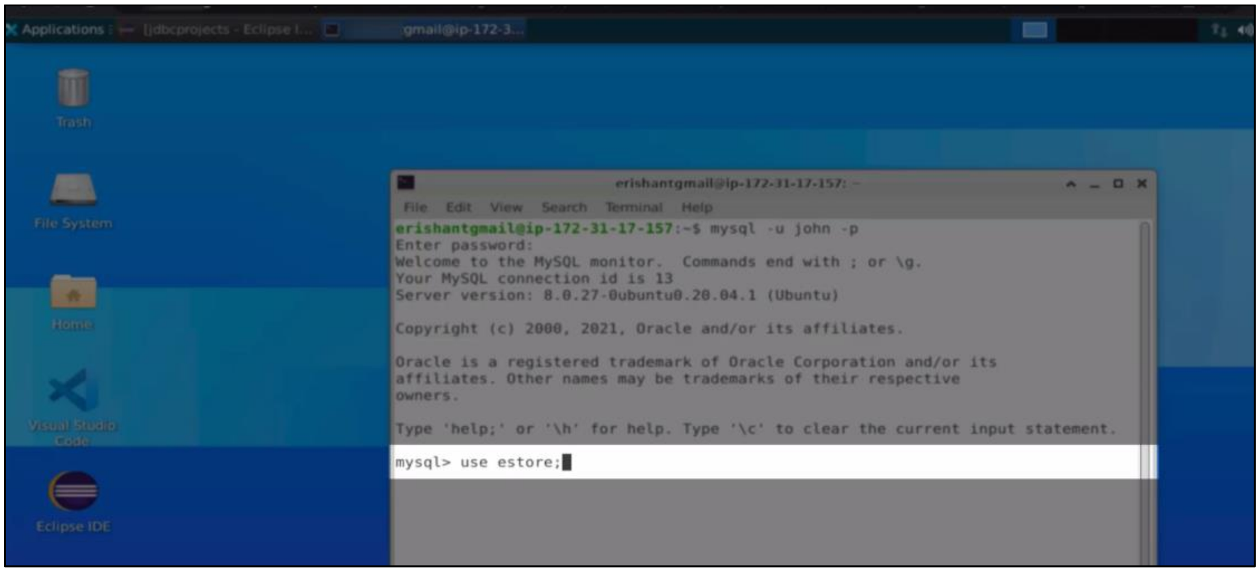
mysql -u john -p



```
erishantgmail@ip-172-31-17-157:~$ mysql -u john
```

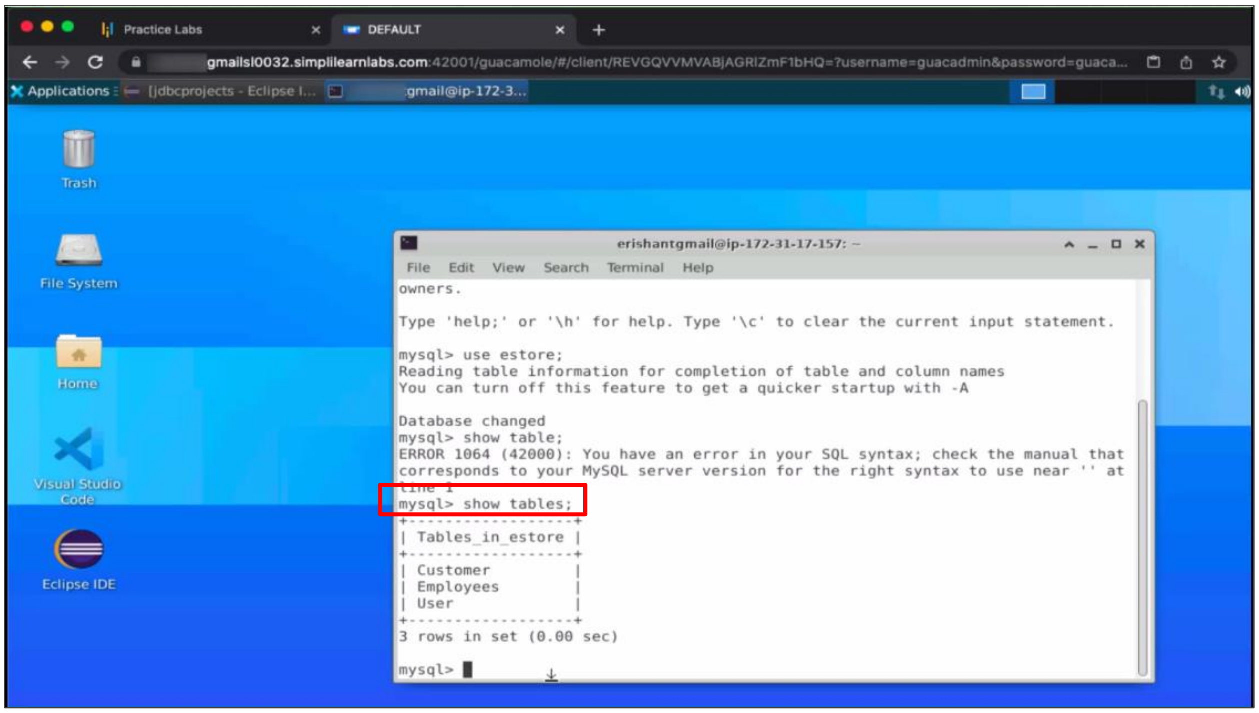
Note: A user named john has already been created for the database.

1.3 Enter the command **use estore;** to change the database



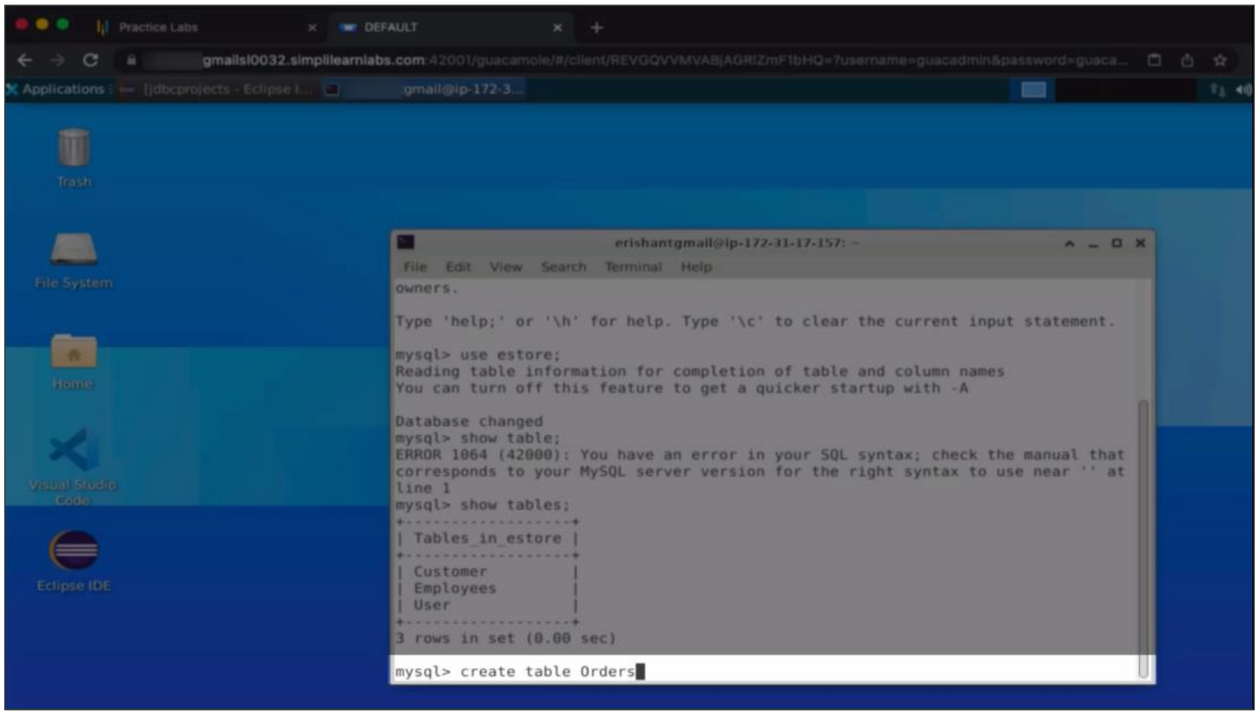
```
erishantgmail@ip-172-31-17-157:~$ mysql -u john -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

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

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

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

mysql> use estore;
```

1.4 Run the **show tables;** command to list the tables in the **estore** database


Practice Labs
DEFAULT
Applications
[jdbcprojects - Eclipse I...
Trash
File System
Home
Visual Studio Code
Eclipse IDE
erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

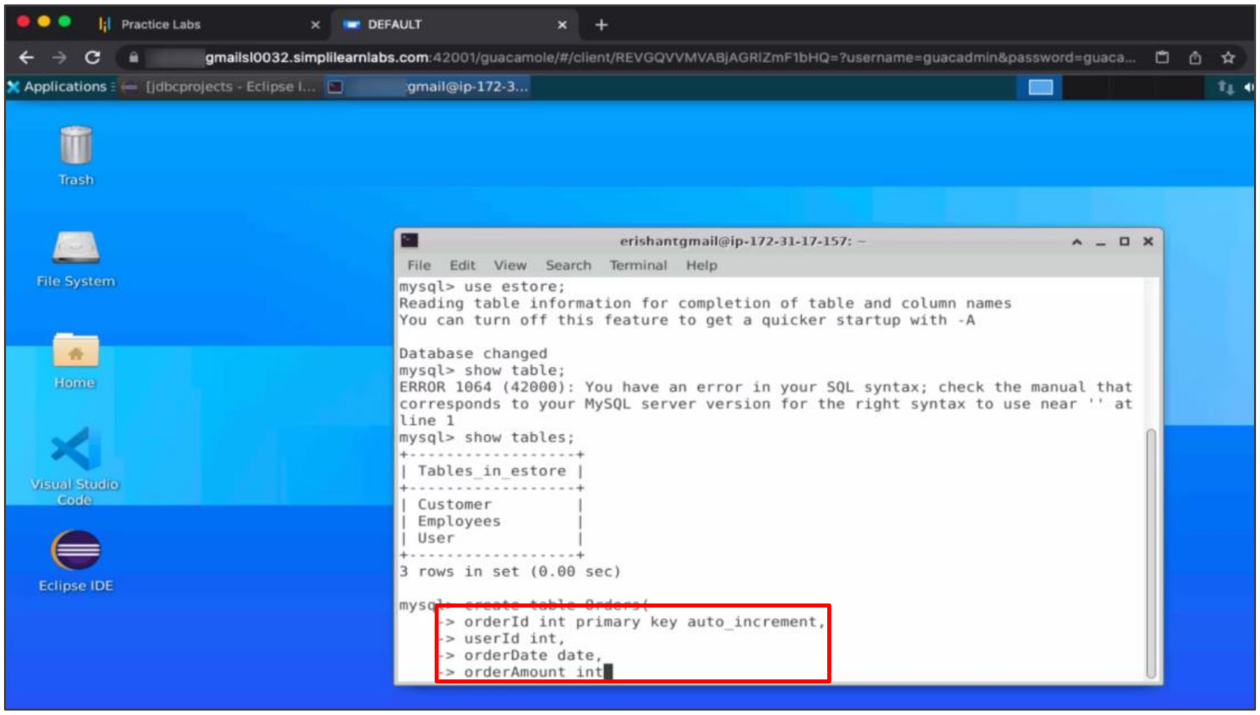
```
owners.

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

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

Database changed
mysql> show table;
ERROR 1064 (42000): 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 '' at
line 1
mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Customer         |
| Employees        |
| User             |
+------------------+
3 rows in set (0.00 sec)

mysql> 
```

1.5 Write **create table Orders** to create a new table in the **estore** database


Practice Labs
DEFAULT
Applications
[jdbcprojects - Eclipse I...
Trash
File System
Home
Visual Studio Code
Eclipse IDE
erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

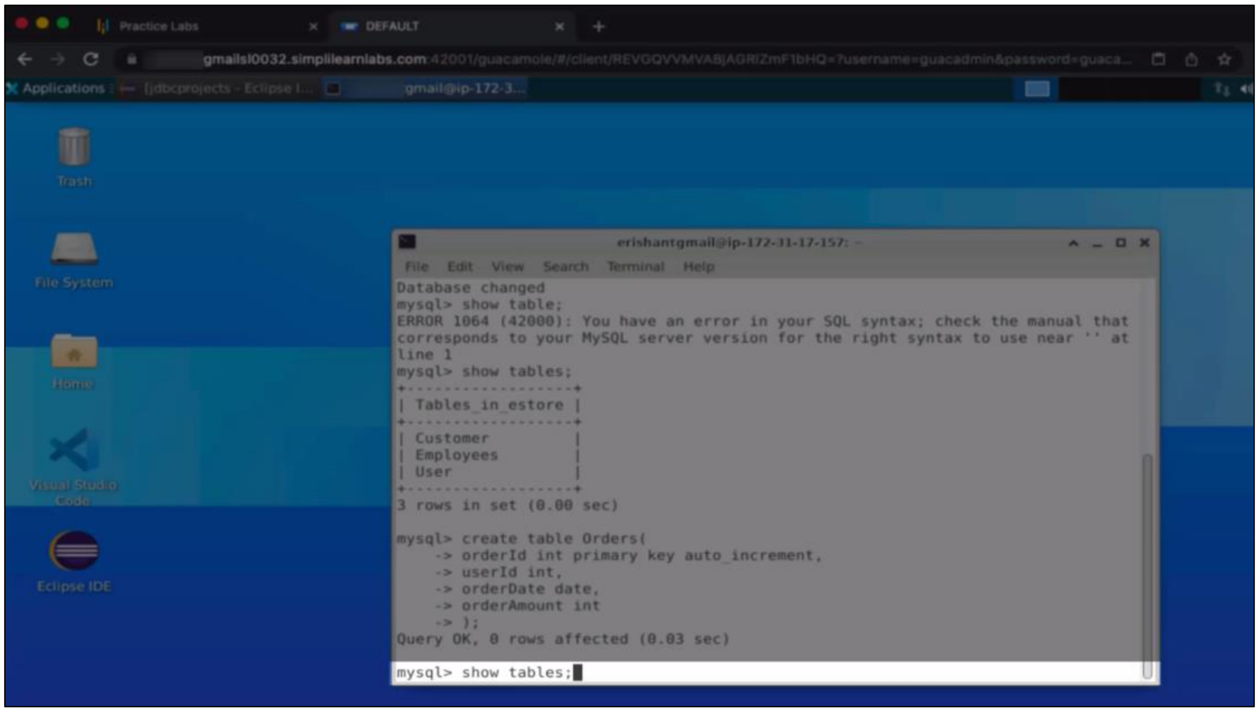
```
owners.

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

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

Database changed
mysql> show table;
ERROR 1064 (42000): 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 '' at
line 1
mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Customer         |
| Employees        |
| User             |
+------------------+
3 rows in set (0.00 sec)

mysql> create table Orders
```

1.6 Define **attributes** for the table Orders



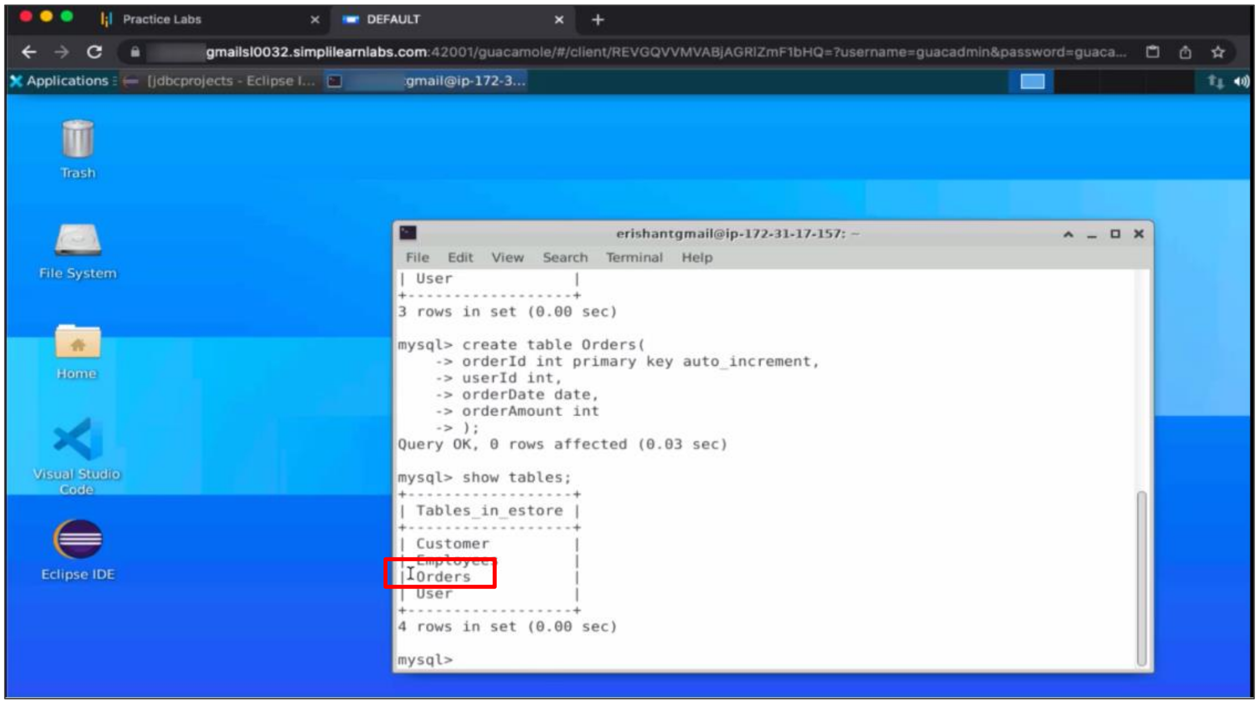
```
mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show table;
ERROR 1064 (42000): 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 '' at
line 1
mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Customer         |
| Employees        |
| User             |
+------------------+
3 rows in set (0.00 sec)

mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
```

1.7 Run the **show tables;** command



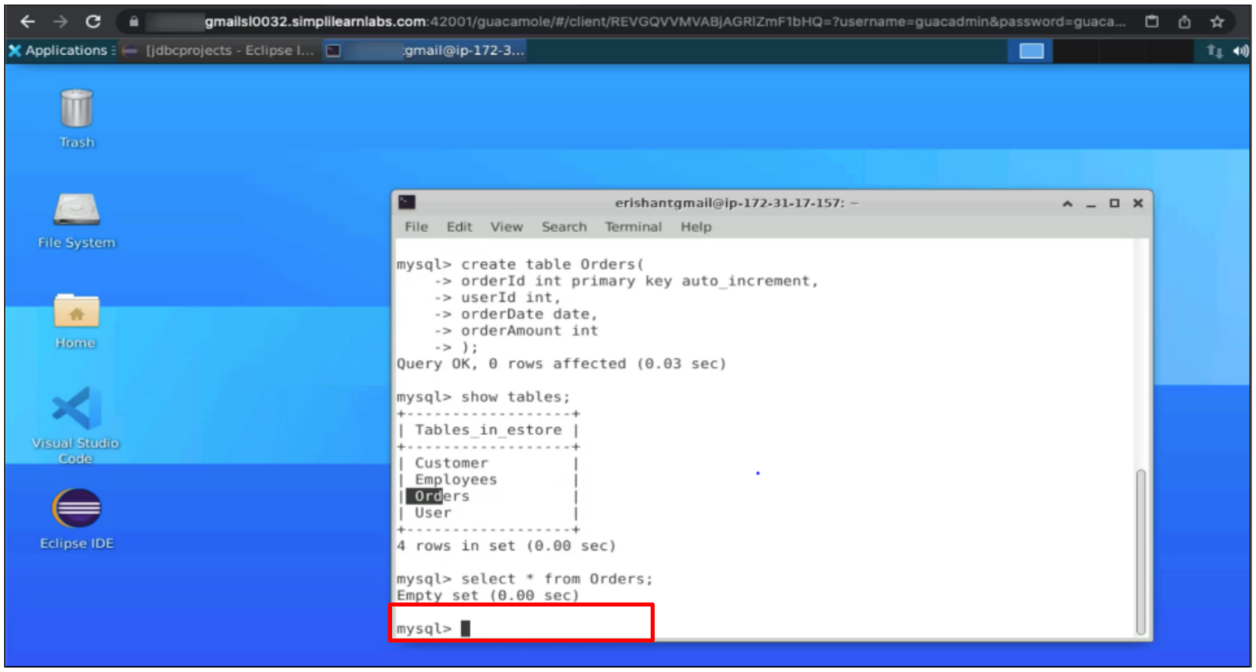
```
Database changed
mysql> show table;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '' at
line 1
mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Customer         |
| Employees        |
| User             |
+------------------+
3 rows in set (0.00 sec)

mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
```

You can see the **Orders** table created.



```
| User             |
+------------------+
3 rows in set (0.00 sec)

mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
    -> );
Query OK, 0 rows affected (0.03 sec)

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

mysql>
```

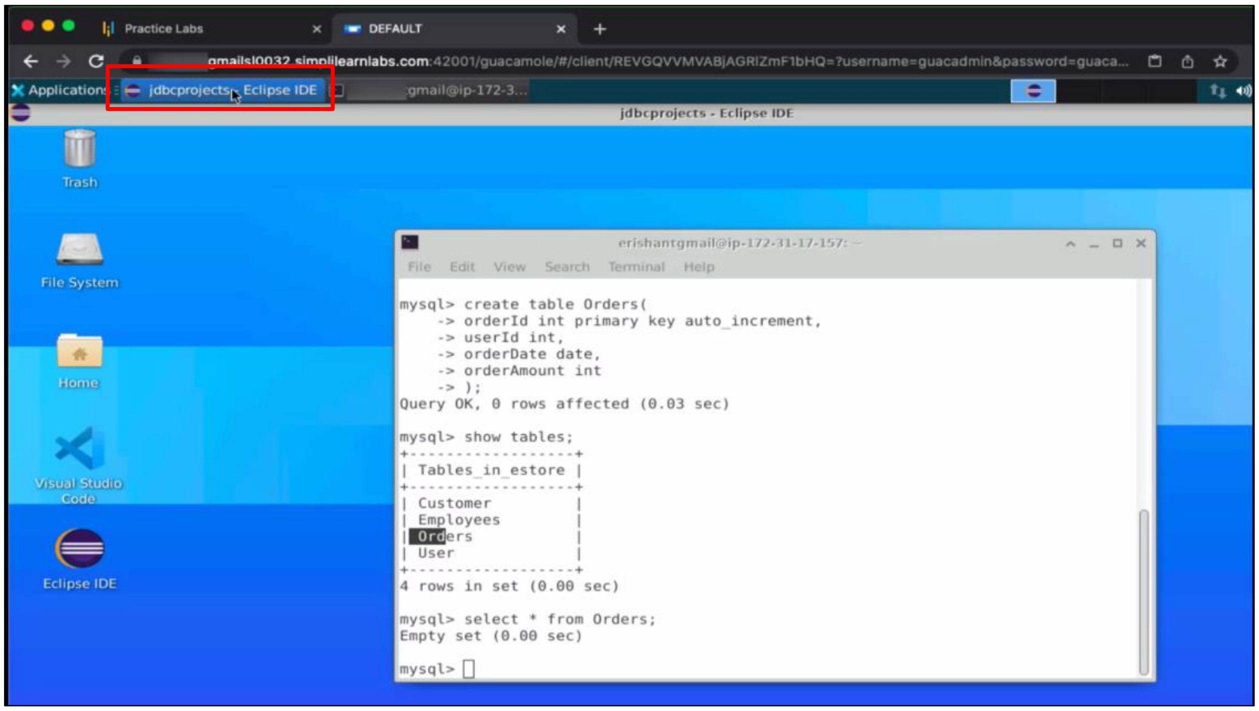
1.8 Run the **select * from Orders;** command to see the empty set in the Orders table



```
mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
    -> );
Query OK, 0 rows affected (0.03 sec)

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

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

mysql>
```

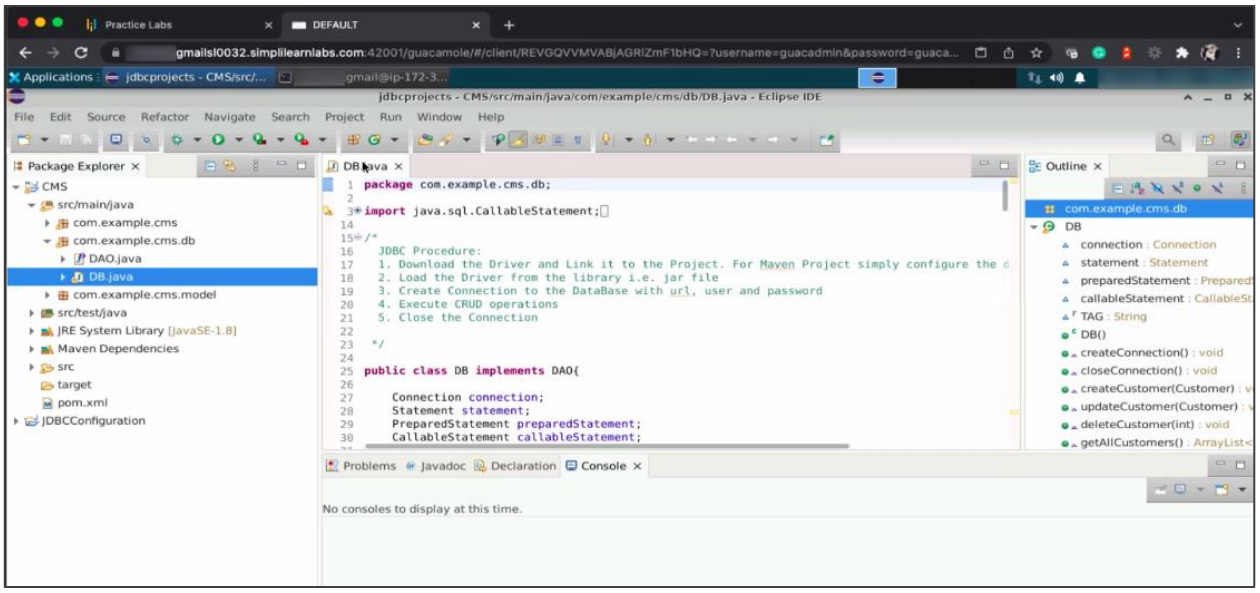
Step 2: Create the executionTransaction method

2.1 Open **Eclipse IDE**



```
mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
    -> );
Query OK, 0 rows affected (0.03 sec)

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

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

mysql> 
```

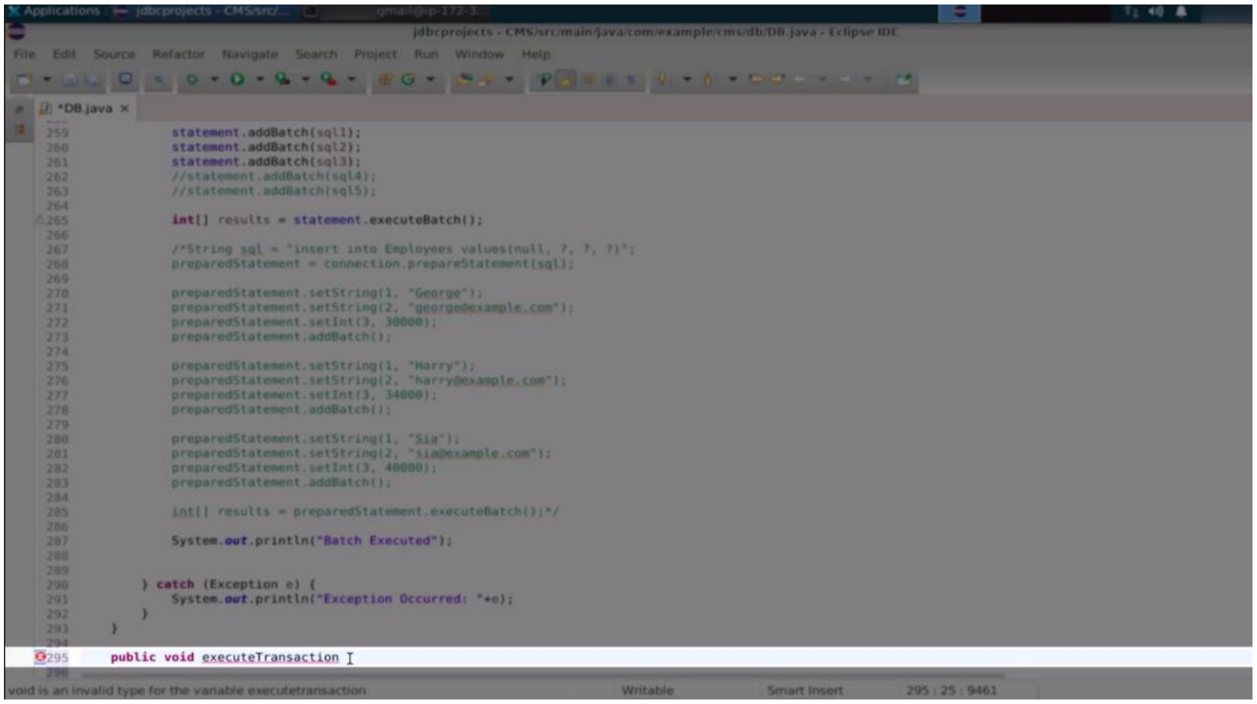
2.2 Open the **DB.java** file


Practice Labs
DEFAULT
jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
CMS
src/main/java
com.example.cms
com.example.cms.db
DAO.java
DB.java
com.example.cms.model
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
pom.xml
JDBCConfiguration
DB.java
Outline
com.example.cms.db
DB
connection : Connection
statement : Statement
preparedStatement : Prepared
callableStatement : CallableSt
TAG : String
DB()
createConnection() : void
closeConnection() : void
createCustomer(Customer) : v
updateCustomer(Customer) : v
deleteCustomer(int) : void
getAllCustomers() : ArrayList<
Problems Javadoc Declaration Console

```
package com.example.cms.db;

import java.sql.CallableStatement;

/*
  JDBC Procedure:
  1. Download the Driver and Link it to the Project. For Maven Project simply configure the c
  2. Load the Driver from the library i.e. jar file
  3. Create Connection to the DataBase with url, user and password
  4. Execute CRUD operations
  5. Close the Connection

 */

public class DB implements DAO{

    Connection connection;
    Statement statement;
    PreparedStatement preparedStatement;
    CallableStatement callableStatement;
```

No consoles to display at this time.

2.3 Create an **executeTransaction** method to write transaction management code

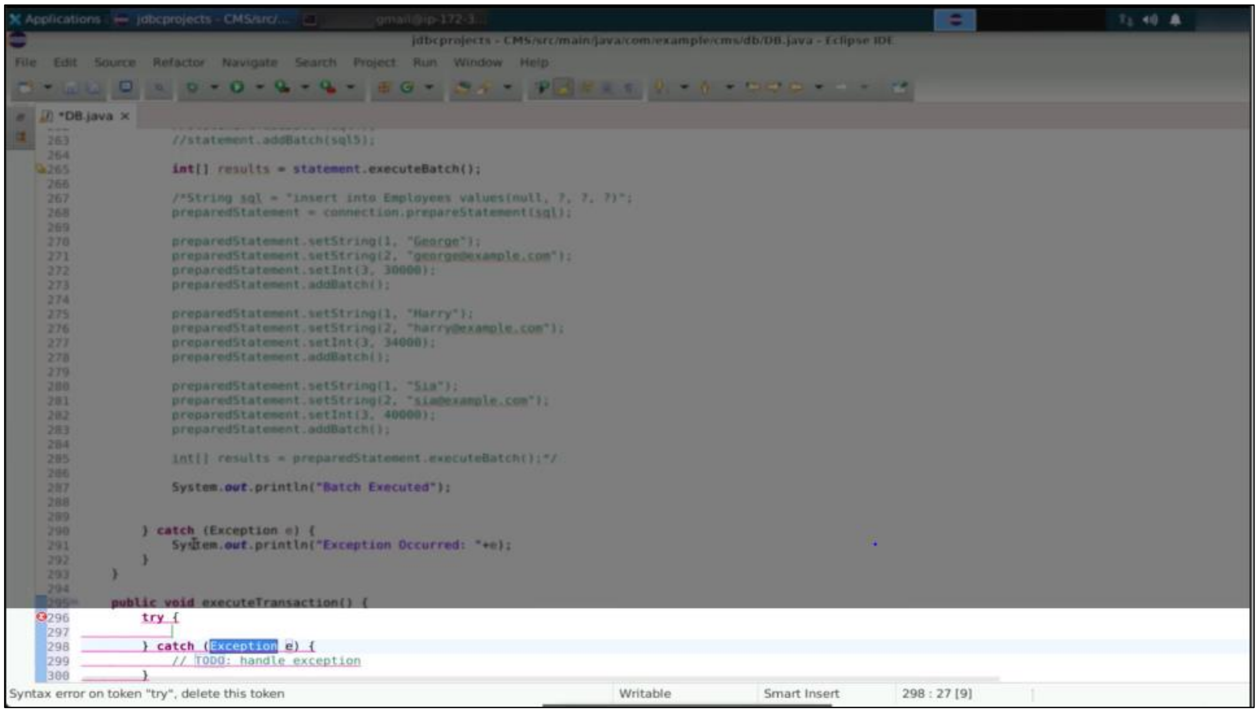


jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE

```java
            statement.addBatch(sql1);
            statement.addBatch(sql2);
            statement.addBatch(sql3);
            //statement.addBatch(sql4);
            //statement.addBatch(sql5);

            int[] results = statement.executeBatch();

            /*String sql = "insert into Employees values(null, ?, ?, ?)";
            preparedStatement = connection.prepareStatement(sql);

            preparedStatement.setString(1, "George");
            preparedStatement.setString(2, "george@example.com");
            preparedStatement.setInt(3, 30000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Harry");
            preparedStatement.setString(2, "harry@example.com");
            preparedStatement.setInt(3, 34000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction
```

```
void is an invalid type for the variable executetransaction    Writable    Smart Insert    295 : 25 : 9461
```

2.4 Write a **try-catch** block to manage all potential errors during code execution

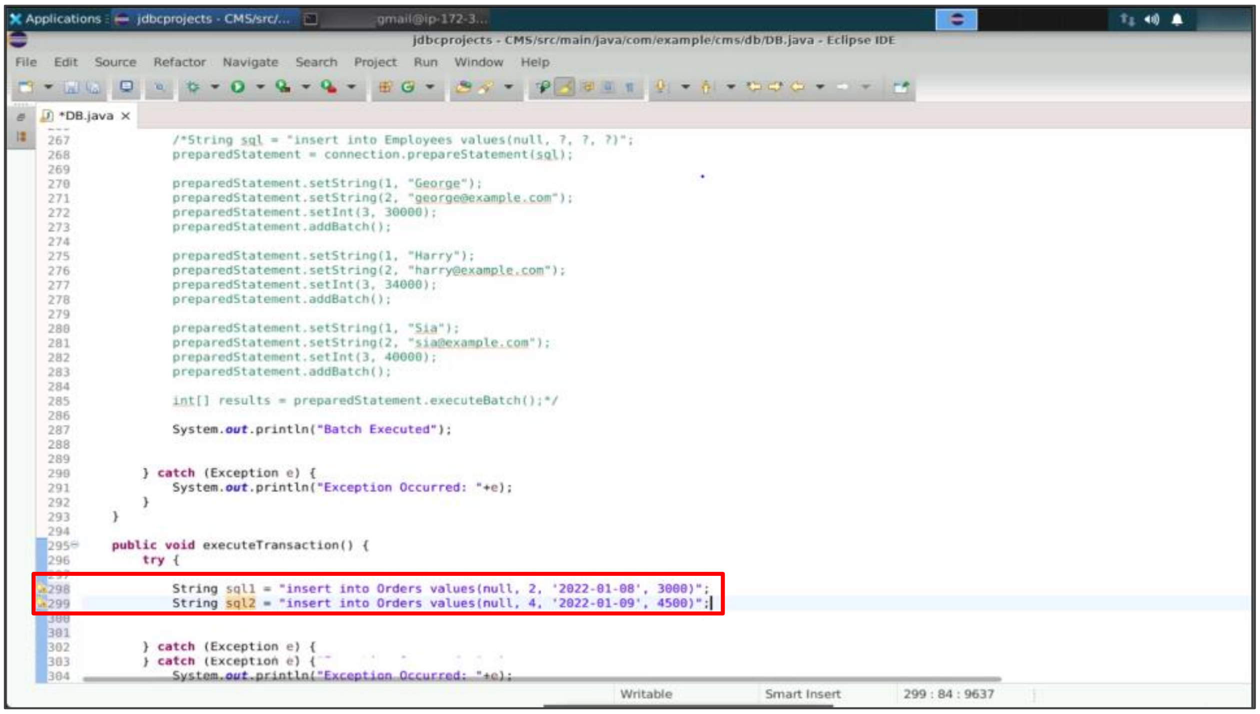


jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE

```java
            //statement.addBatch(sql5);

            int[] results = statement.executeBatch();

            /*String sql = "insert into Employees values(null, ?, ?, ?)";
            preparedStatement = connection.prepareStatement(sql);

            preparedStatement.setString(1, "George");
            preparedStatement.setString(2, "george@example.com");
            preparedStatement.setInt(3, 30000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Harry");
            preparedStatement.setString(2, "harry@example.com");
            preparedStatement.setInt(3, 34000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

        } catch (Exception e) {
            // TODO: handle exception
        }
```

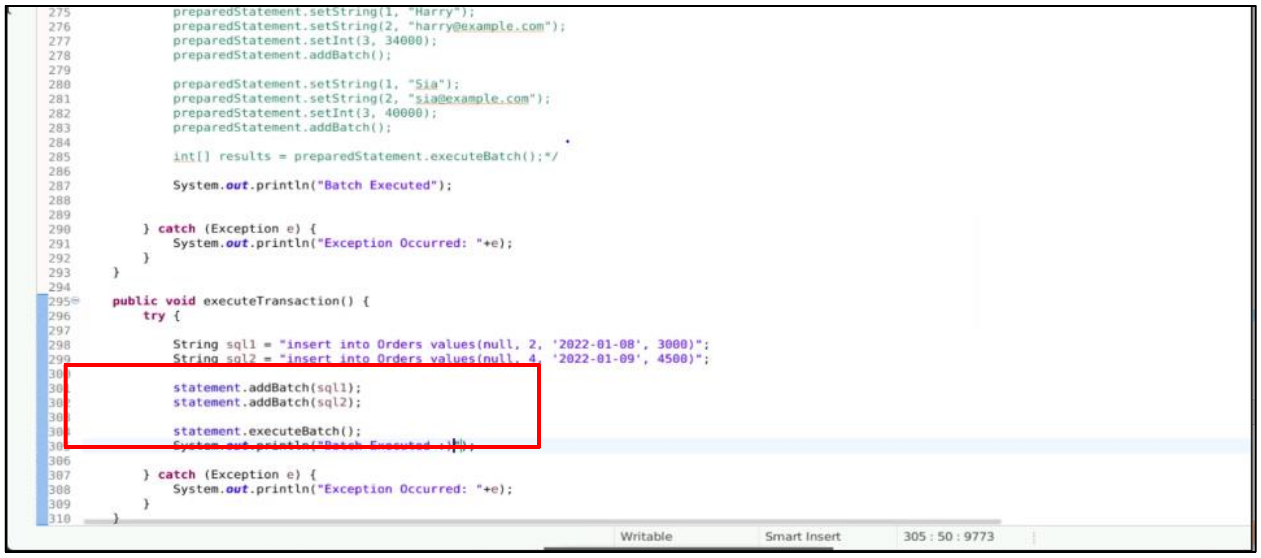
```
Syntax error on token "try", delete this token    Writable    Smart Insert    298 : 27 [9]
```

2.5 Create two SQL statements to perform the **insert** operation



```
            /*String sql = "insert into Employees values(null, ?, ?, ?)";
            preparedStatement = connection.prepareStatement(sql);

            preparedStatement.setString(1, "George");
            preparedStatement.setString(2, "george@example.com");
            preparedStatement.setInt(3, 30000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Harry");
            preparedStatement.setString(2, "harry@example.com");
            preparedStatement.setInt(3, 34000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            String sql1 = "insert into Orders values(null, 2, '2022-01-08', 3000)";
            String sql2 = "insert into Orders values(null, 4, '2022-01-09', 4500)";


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

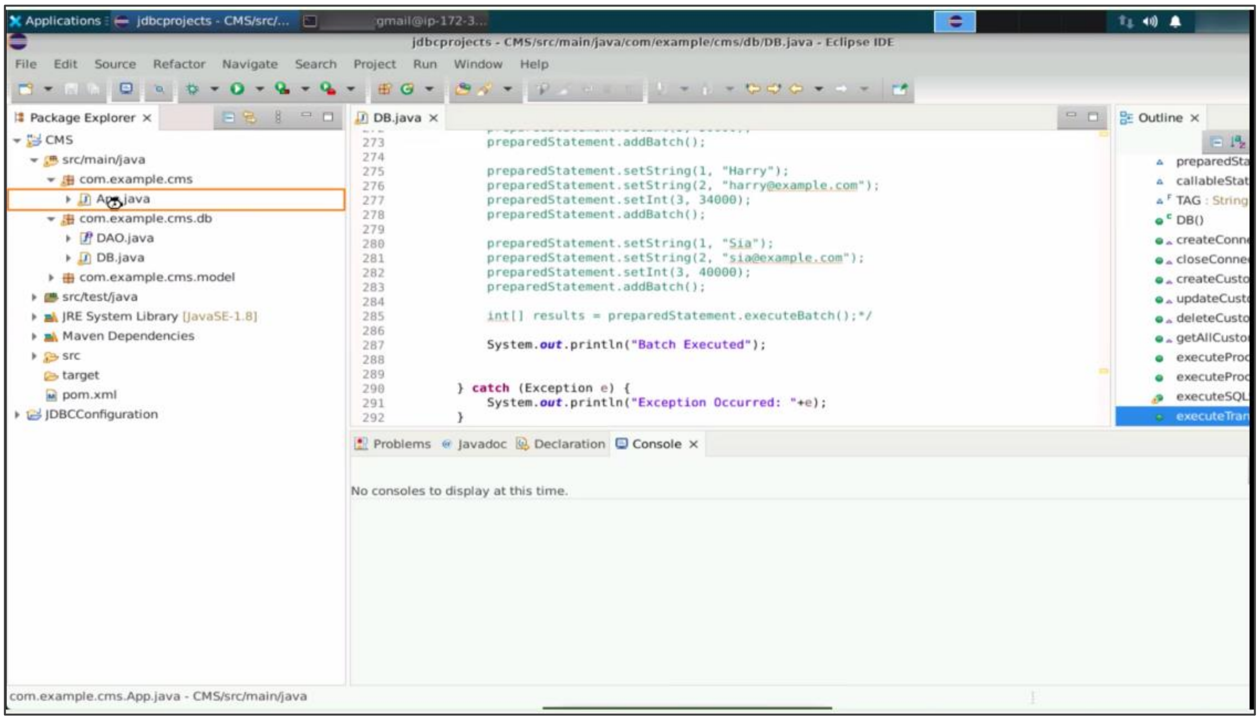
2.6 Create a batch and execute the batch statement to run the SQL statements created above



```
            preparedStatement.setString(1, "Harry");
            preparedStatement.setString(2, "harry@example.com");
            preparedStatement.setInt(3, 34000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            String sql1 = "insert into Orders values(null, 2, '2022-01-08', 3000)";
            String sql2 = "insert into Orders values(null, 4, '2022-01-09', 4500)";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            System.out.println("Batch Executed :)");

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

Step 3: Call the created method and checking the output

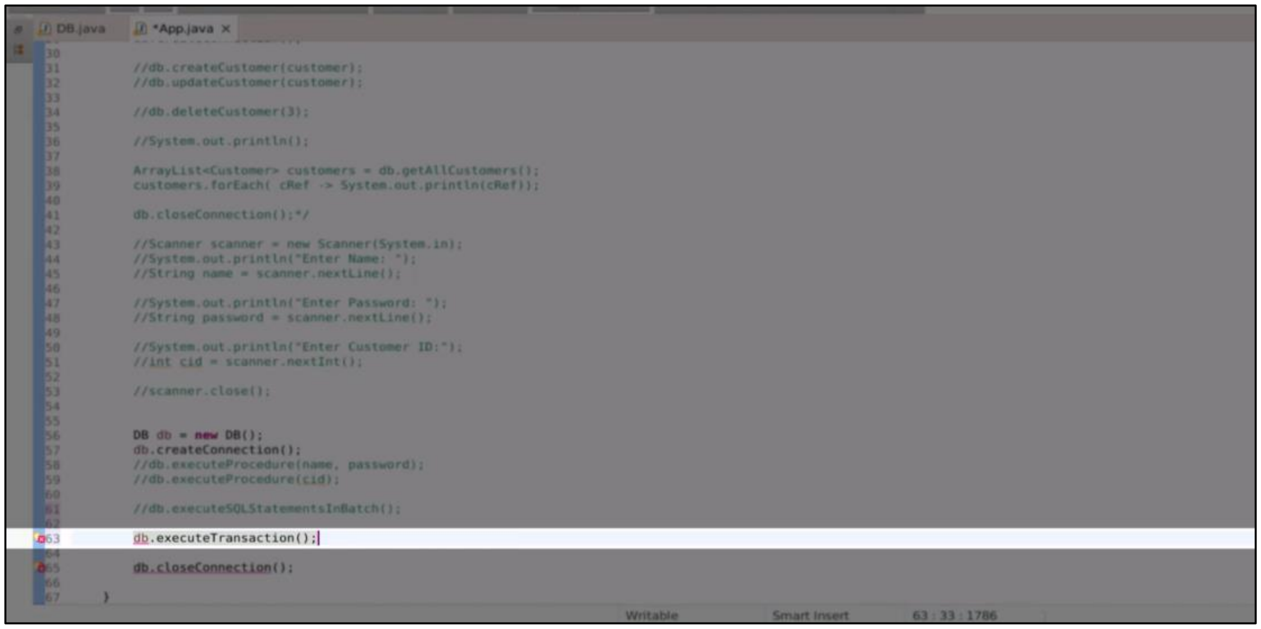
3.1 Open the **App.java** file



```
        preparedStatement.addBatch();

        preparedStatement.setString(1, "Harry");
        preparedStatement.setString(2, "harry@example.com");
        preparedStatement.setInt(3, 34000);
        preparedStatement.addBatch();

        preparedStatement.setString(1, "Sia");
        preparedStatement.setString(2, "sia@example.com");
        preparedStatement.setInt(3, 40000);
        preparedStatement.addBatch();

        int[] results = preparedStatement.executeBatch();*/

        System.out.println("Batch Executed");


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

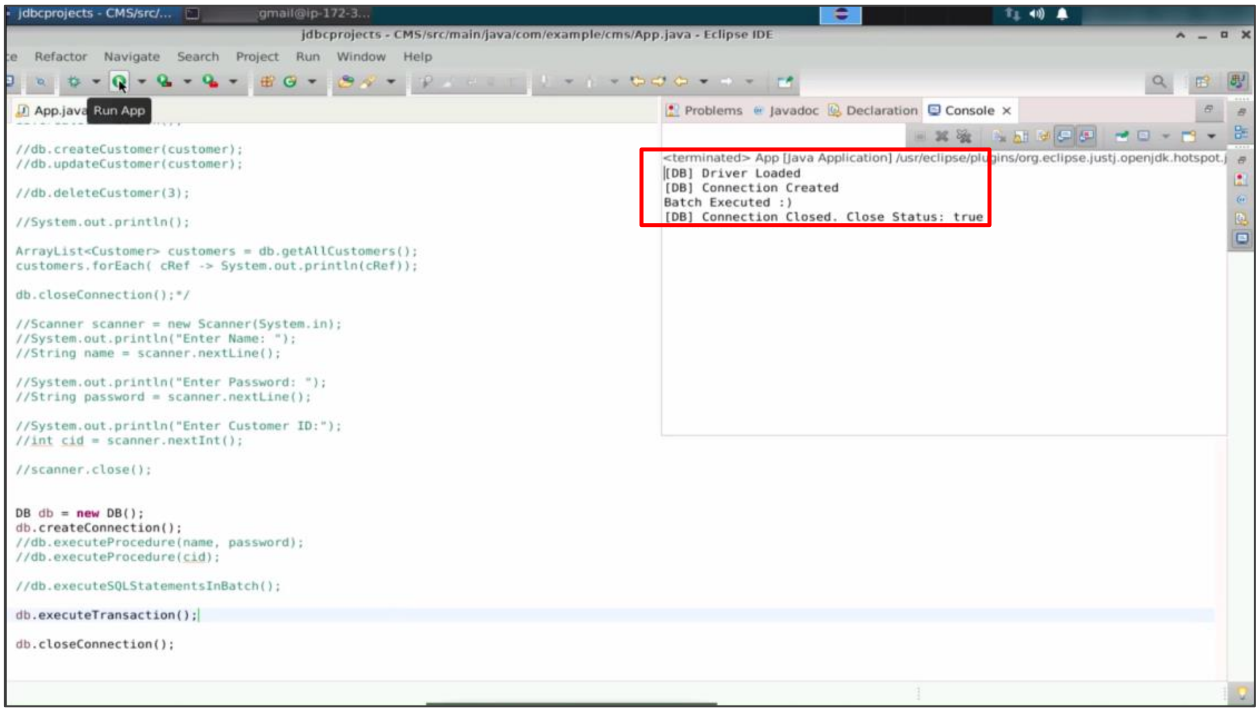
```
No consoles to display at this time.
```

3.2 Write **db.executeTransaction();** to call the created method



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

        //db.deleteCustomer(3);

        //System.out.println();

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

        db.closeConnection();*/

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

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

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

        //scanner.close();


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

        //db.executeSQLStatementsInBatch();

        db.executeTransaction();

        db.closeConnection();

    }
```

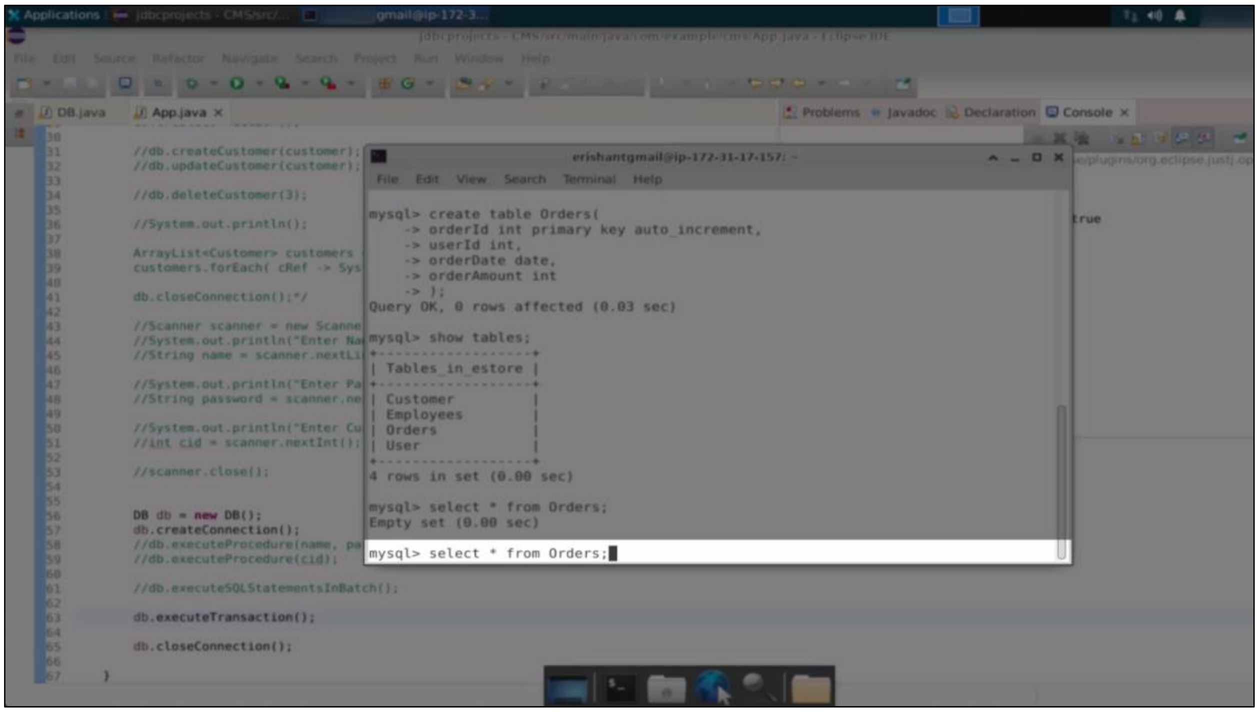
3.3 Run the code, and you will see the output **Batch Executed :)** with no errors


jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
Refactor Navigate Search Project Run Window Help
App.java
Run App
Problems
Javadoc
Declaration
Console

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

//db.deleteCustomer(3);

//System.out.println();

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

db.closeConnection();*/

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

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

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

//scanner.close();


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

//db.executeSQLStatementsInBatch();

db.executeTransaction();

db.closeConnection();
```

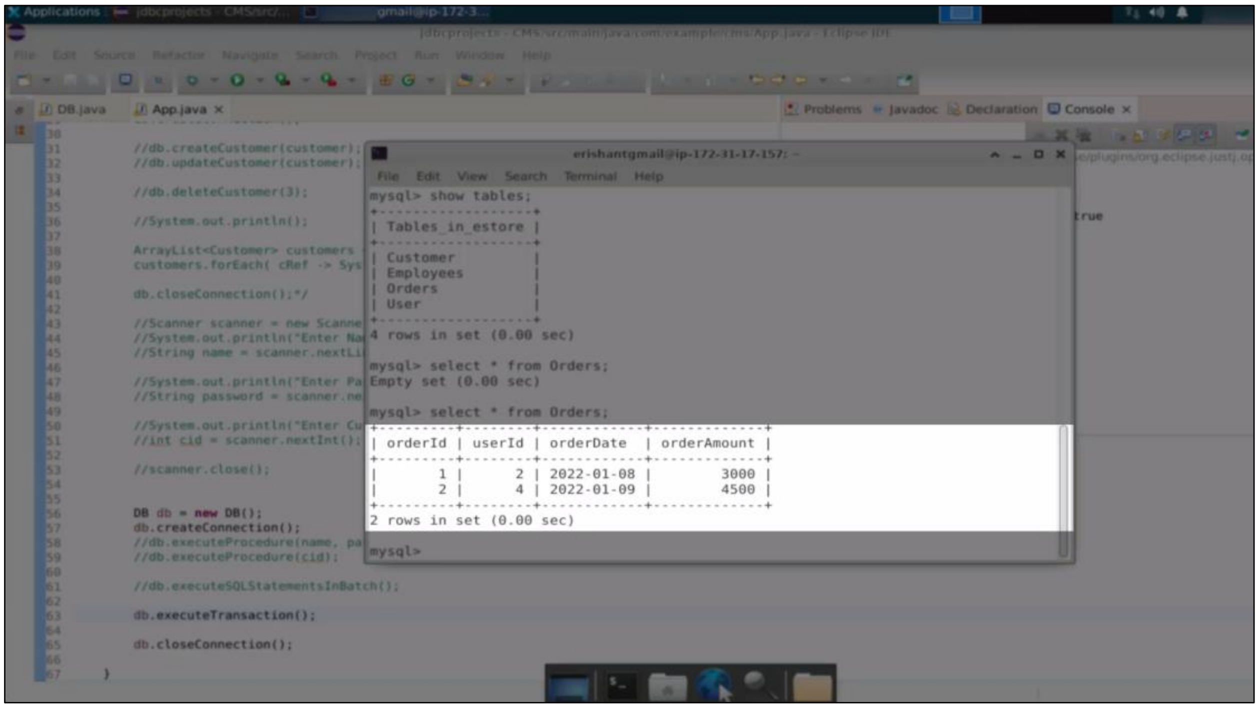
```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
[DB] Driver Loaded
[DB] Connection Created
Batch Executed :)
[DB] Connection Closed. Close Status: true
```

3.4 Return to the Terminal Emulator and run the **select * from Orders;** command


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

```
mysql> create table Orders(
    -> orderId int primary key auto_increment,
    -> userId int,
    -> orderDate date,
    -> orderAmount int
    -> );
Query OK, 0 rows affected (0.03 sec)

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

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

mysql> select * from Orders;
```

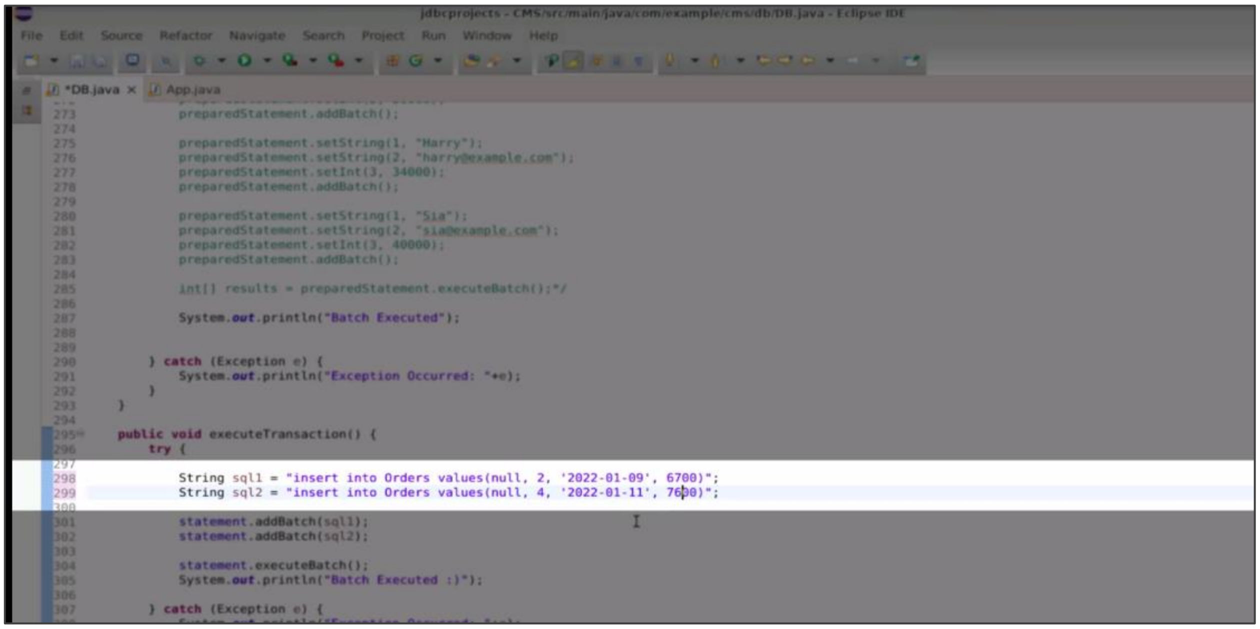
You will see the following orders inserted in the **Orders** table:



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

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

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
+---------+--------+------------+-------------+
2 rows in set (0.00 sec)

mysql>
```

Step 4: Use the auto-commit feature

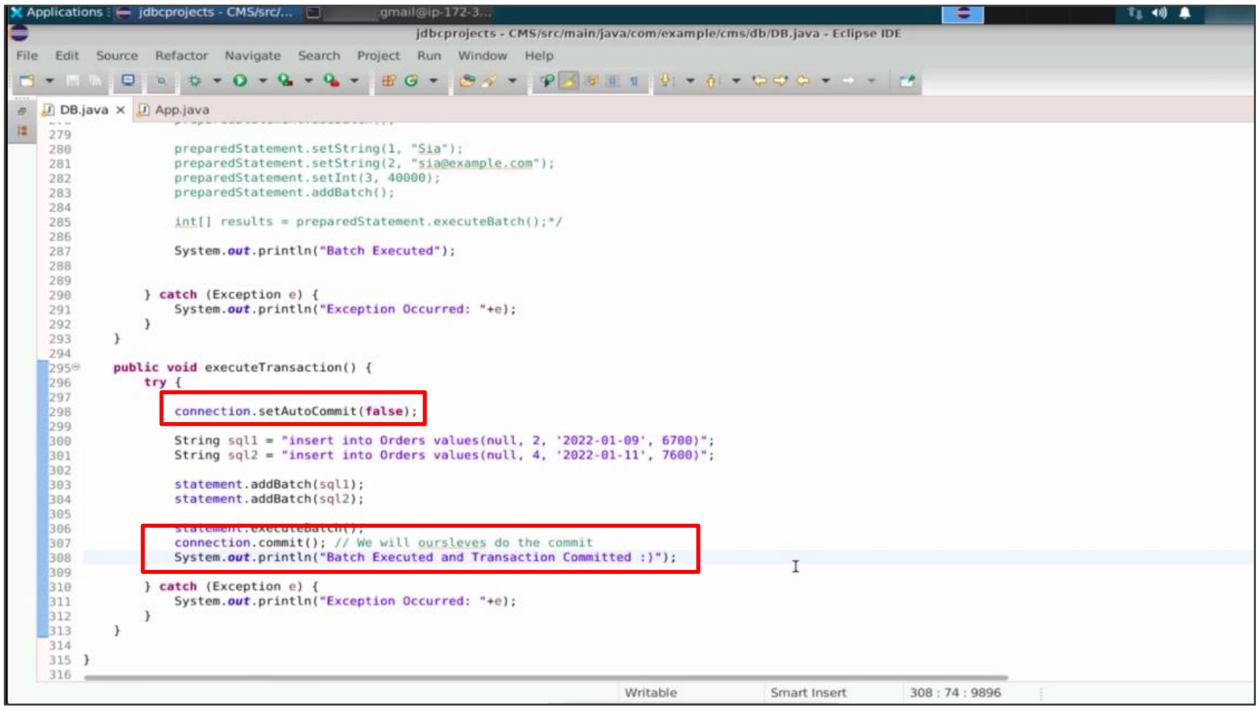
4.1 Return to the **DB.java** file and change it to insert two more orders



jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*DB.java
App.java

```java
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Harry");
            preparedStatement.setString(2, "harry@example.com");
            preparedStatement.setInt(3, 34000);
            preparedStatement.addBatch();

            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            String sql1 = "insert into Orders values(null, 2, '2022-01-09', 6700)";
            String sql2 = "insert into Orders values(null, 4, '2022-01-11', 7600)";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            System.out.println("Batch Executed :)");

        } catch (Exception e) {
```

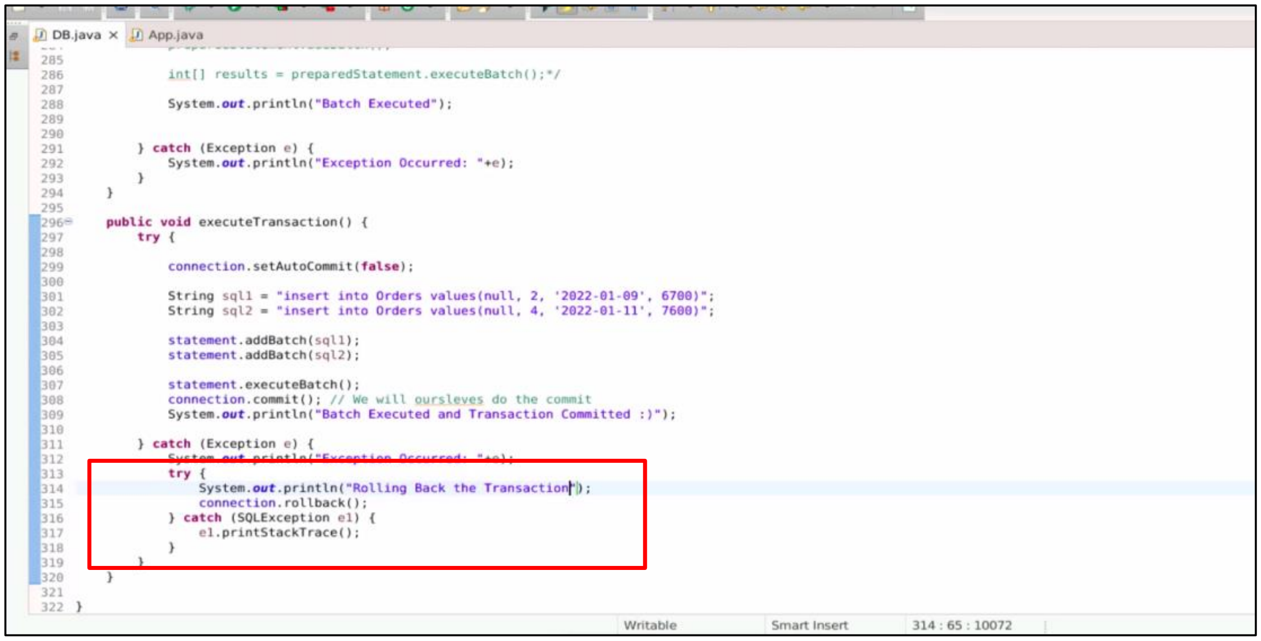
4.2 Add the auto-commit feature in a **try-catch** block to disable auto-commit and add **commit()**



Applications
jdbcprojects - CMS/src/...
jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DB.java
App.java
Writable
Smart Insert
308 : 74 : 9896

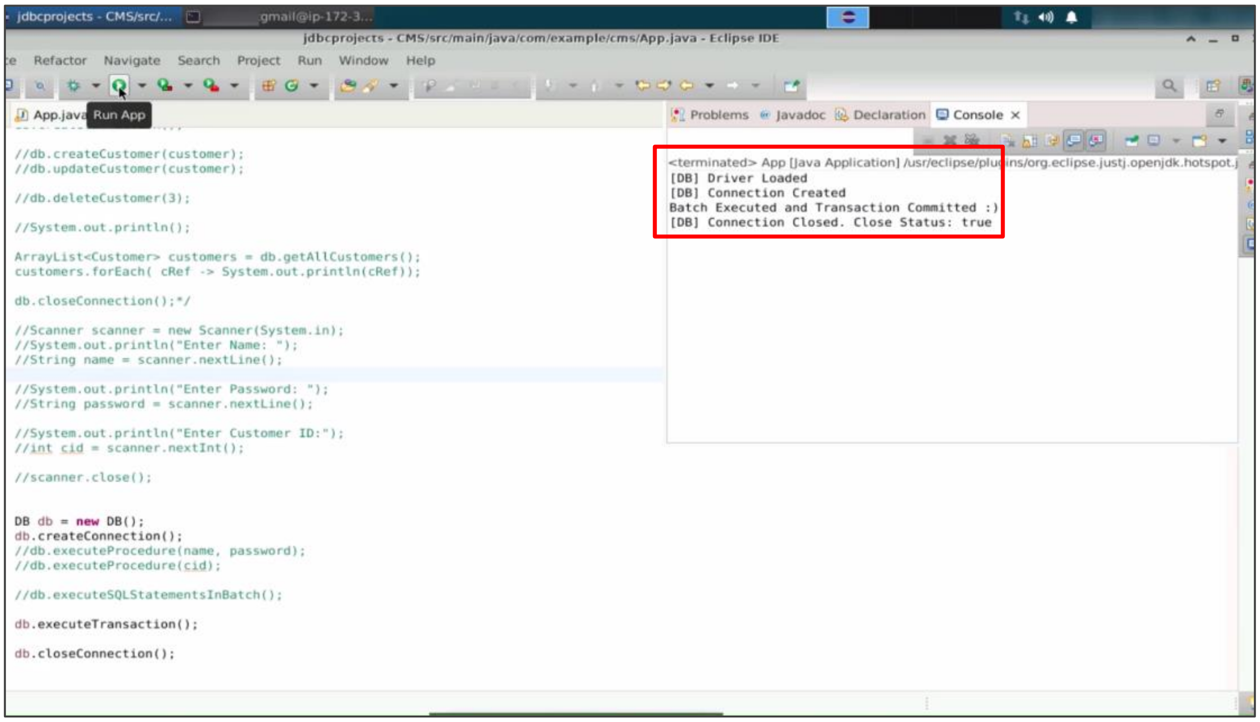
```java
            preparedStatement.setString(1, "Sia");
            preparedStatement.setString(2, "sia@example.com");
            preparedStatement.setInt(3, 40000);
            preparedStatement.addBatch();

            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            connection.setAutoCommit(false);

            String sql1 = "insert into Orders values(null, 2, '2022-01-09', 6700)";
            String sql2 = "insert into Orders values(null, 4, '2022-01-11', 7600)";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            connection.commit(); // We will oursleves do the commit
            System.out.println("Batch Executed and Transaction Committed :)");

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

}
```

4.3 Add a **rollback** method within the catch block for any errors that may occur during execution



```java
            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            connection.setAutoCommit(false);

            String sql1 = "insert into Orders values(null, 2, '2022-01-09', 6700)";
            String sql2 = "insert into Orders values(null, 4, '2022-01-11', 7600)";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            connection.commit(); // We will oursleves do the commit
            System.out.println("Batch Executed and Transaction Committed :)");

        } catch (Exception e) {
            System.out.println("Exception Occurred: "+e);
            try {
                System.out.println("Rolling Back the Transaction");
                connection.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }

}
```

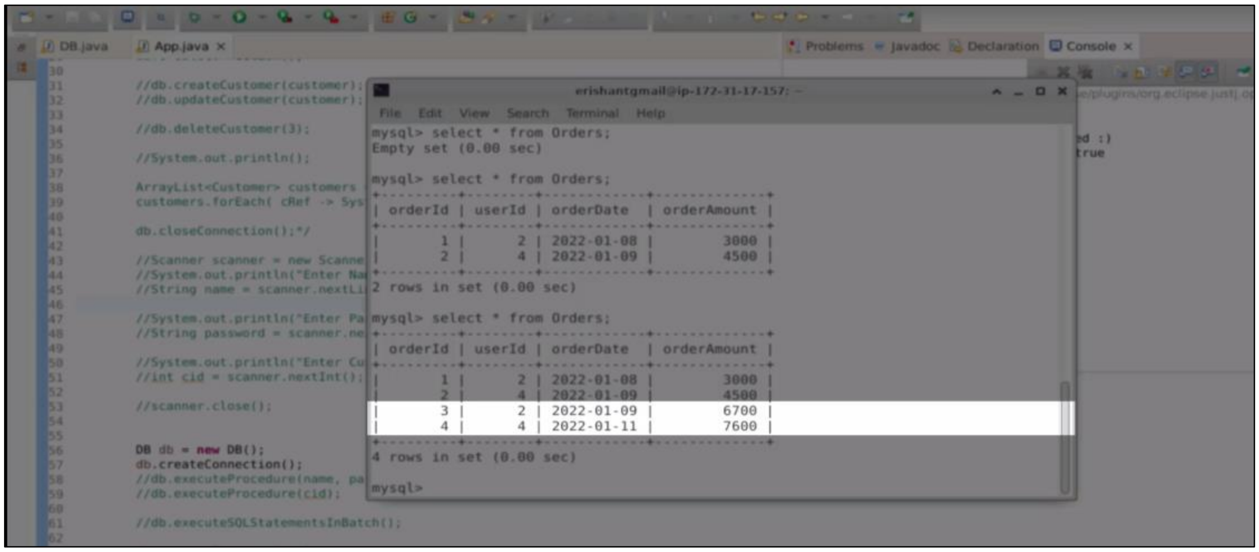
4.4 Re-run this code, and you will see **Batch Executed and Transaction committed :)** in the output without error


jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
Refactor Navigate Search Project Run Window Help
App.java
Run App
Problems
Javadoc
Declaration
Console

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

//db.deleteCustomer(3);

//System.out.println();

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

db.closeConnection();*/

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

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

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

//scanner.close();


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

//db.executeSQLStatementsInBatch();

db.executeTransaction();

db.closeConnection();
```

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
[DB] Driver Loaded
[DB] Connection Created
Batch Executed and Transaction Committed :)
[DB] Connection Closed. Close Status: true
```

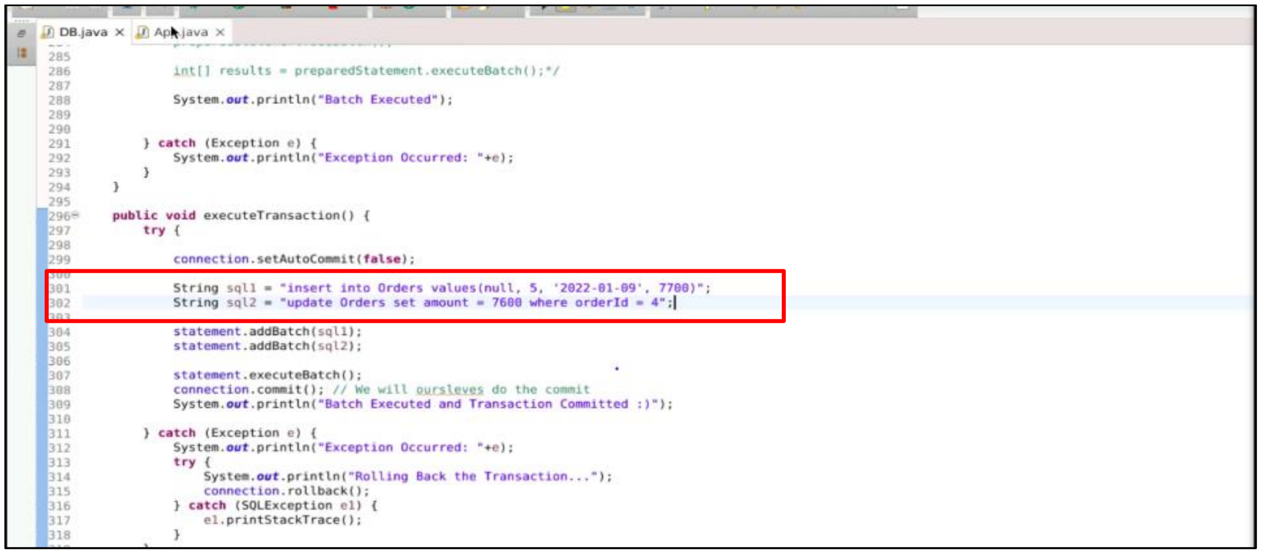
4.5 Go back to the **Terminal Emulator** and run the **select * from Orders;** command. You will see two more orders inserted in the Orders table.



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

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
+---------+--------+------------+-------------+
2 rows in set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql>
```

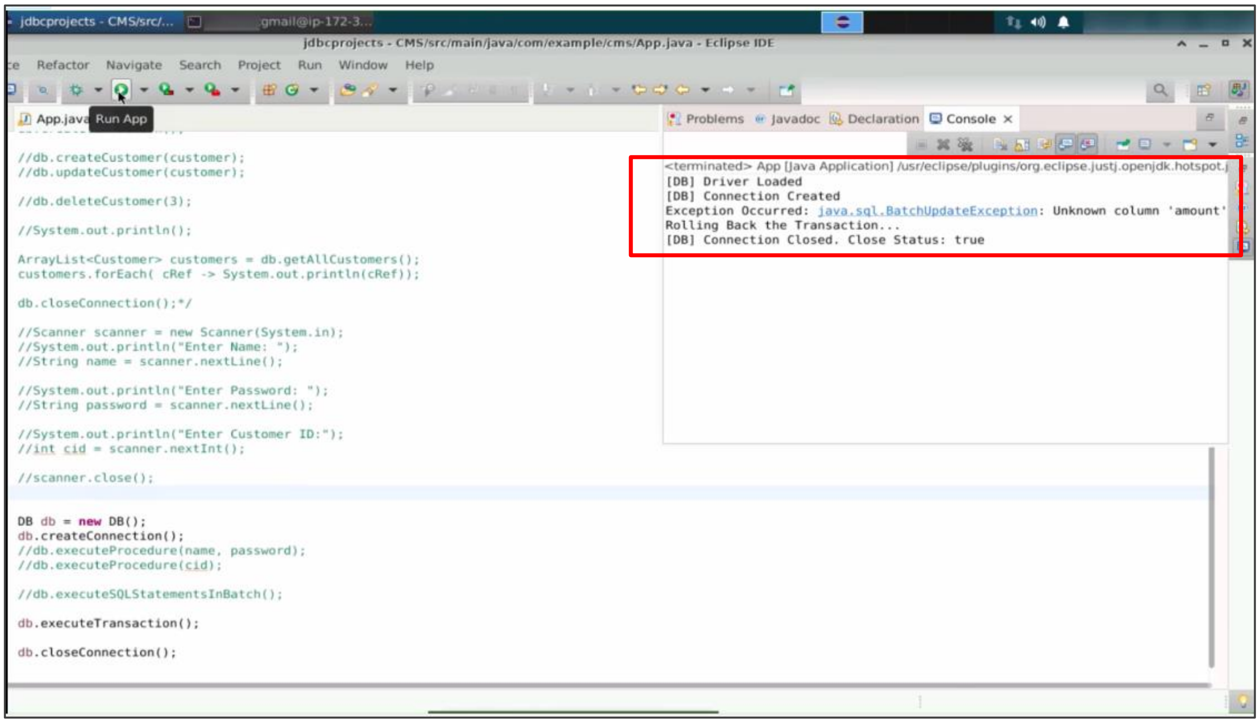
Step 5: Use the insert and update operations

5.1 Write an insert and an update operation. Insert a new order and update the old order with ID 4



```java
        int[] results = preparedStatement.executeBatch();*/

        System.out.println("Batch Executed");


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

public void executeTransaction() {
    try {

        connection.setAutoCommit(false);

        String sql1 = "insert into Orders values(null, 5, '2022-01-09', 7700)";
        String sql2 = "update Orders set amount = 7600 where orderId = 4";

        statement.addBatch(sql1);
        statement.addBatch(sql2);

        statement.executeBatch();
        connection.commit(); // We will oursleves do the commit
        System.out.println("Batch Executed and Transaction Committed :)");

    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
        try {
            System.out.println("Rolling Back the Transaction...");
            connection.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
```

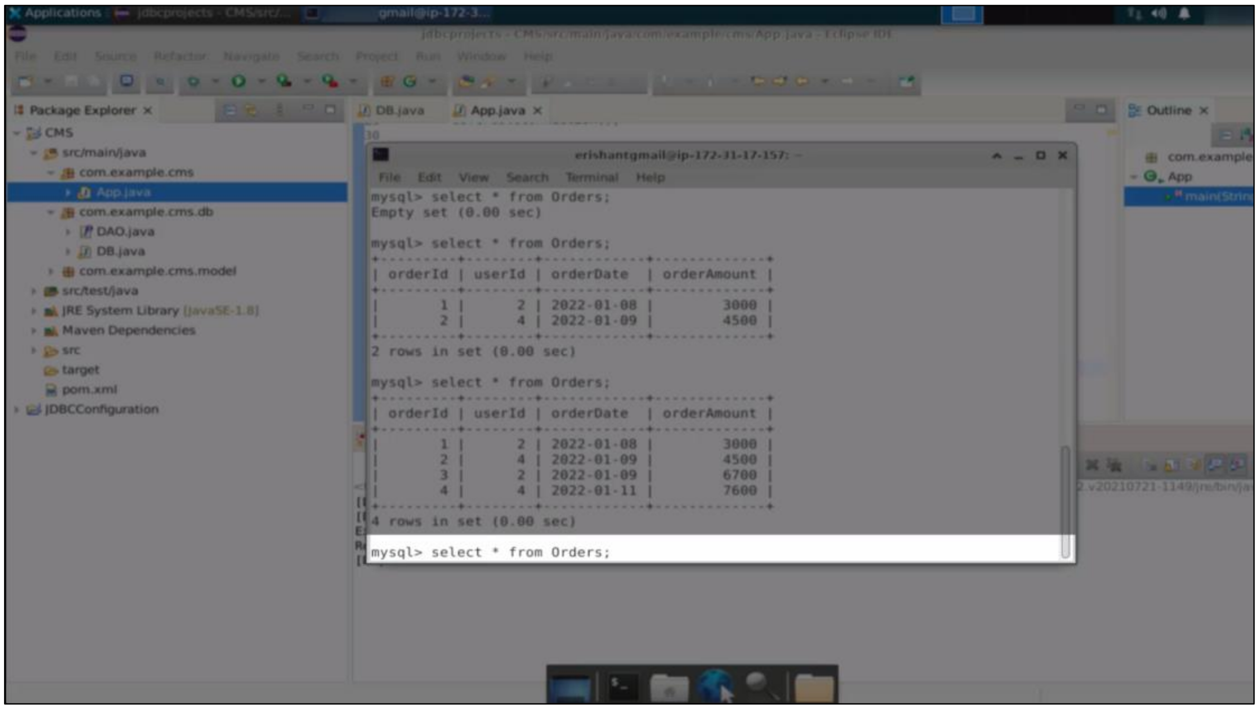
5.2 Re-run the code, and you will see **Exception Occurred** in the console



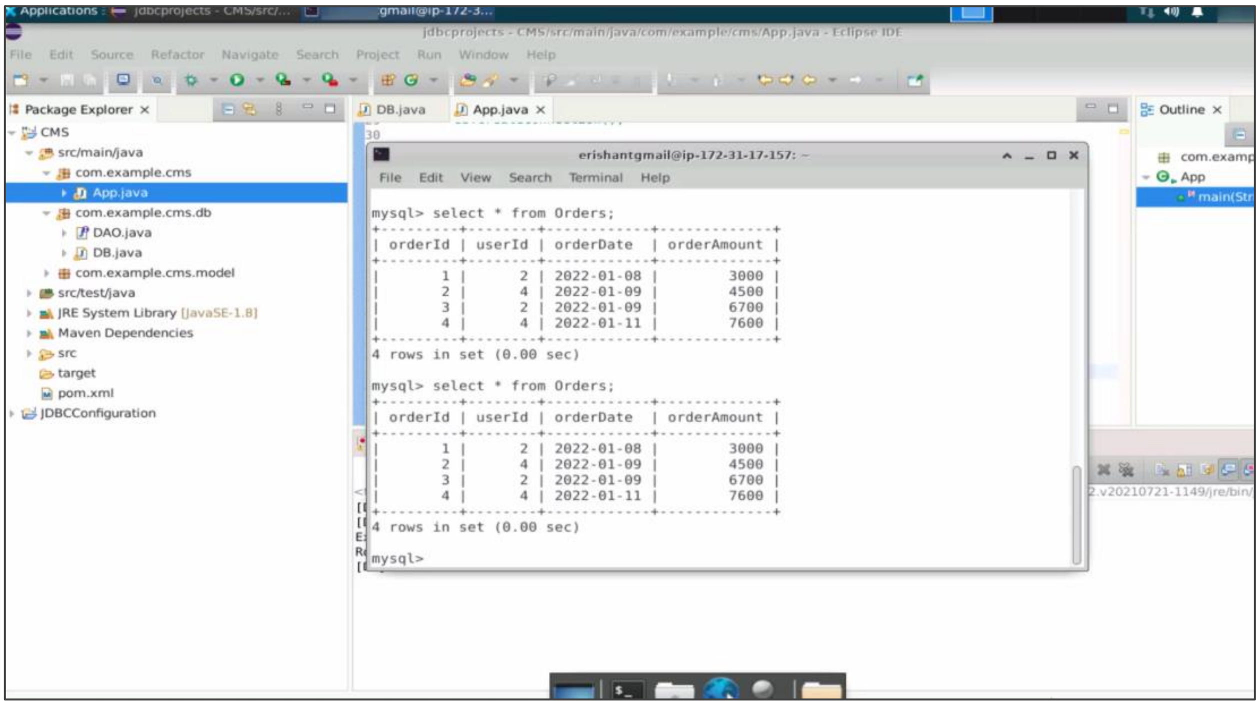
```java
//db.createCustomer(customer);
//db.updateCustomer(customer);

//db.deleteCustomer(3);

//System.out.println();

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

db.closeConnection();*/

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

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

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

//scanner.close();


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

//db.executeSQLStatementsInBatch();

db.executeTransaction();

db.closeConnection();
```

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
[DB] Driver Loaded
[DB] Connection Created
Exception Occurred: java.sql.BatchUpdateException: Unknown column 'amount'
Rolling Back the Transaction...
[DB] Connection Closed. Close Status: true
```

5.3 Return to the terminal and run the **select * from Orders;** command. You will see that the new order is not inserted, and the old order is also not updated

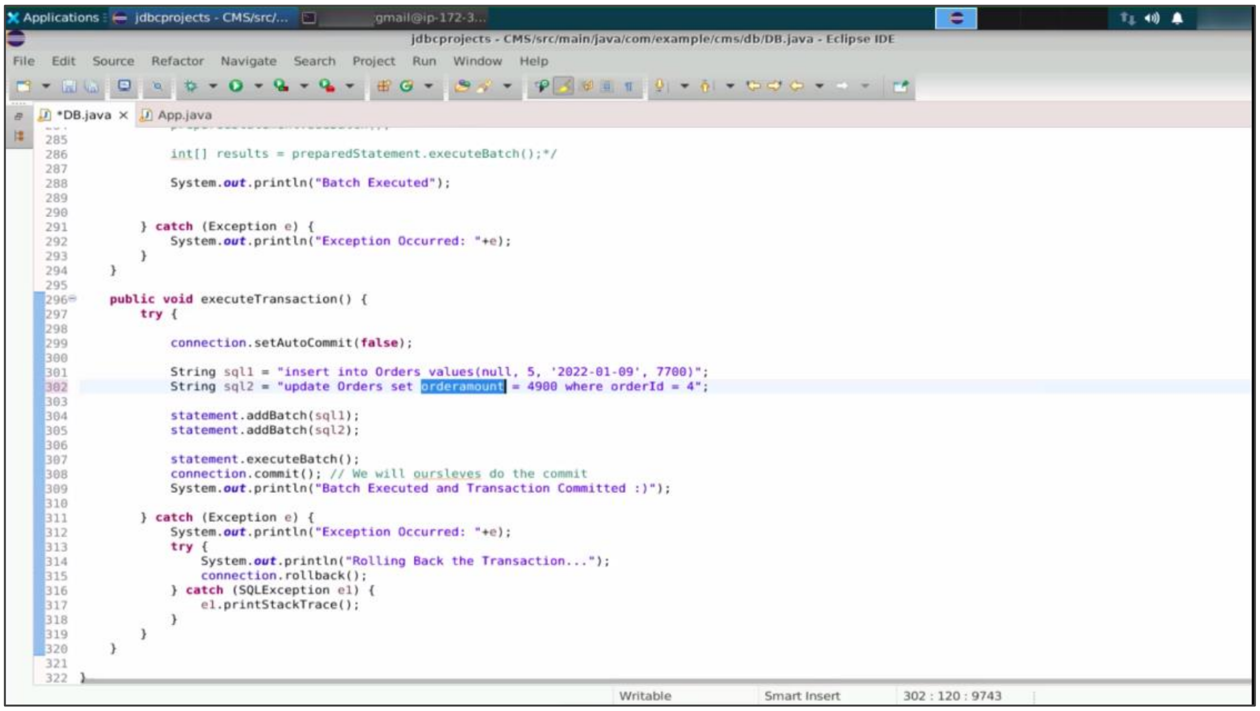


```
mysql> select * from Orders;
Empty set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
+---------+--------+------------+-------------+
2 rows in set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql> select * from Orders;
```



```
mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql>
```

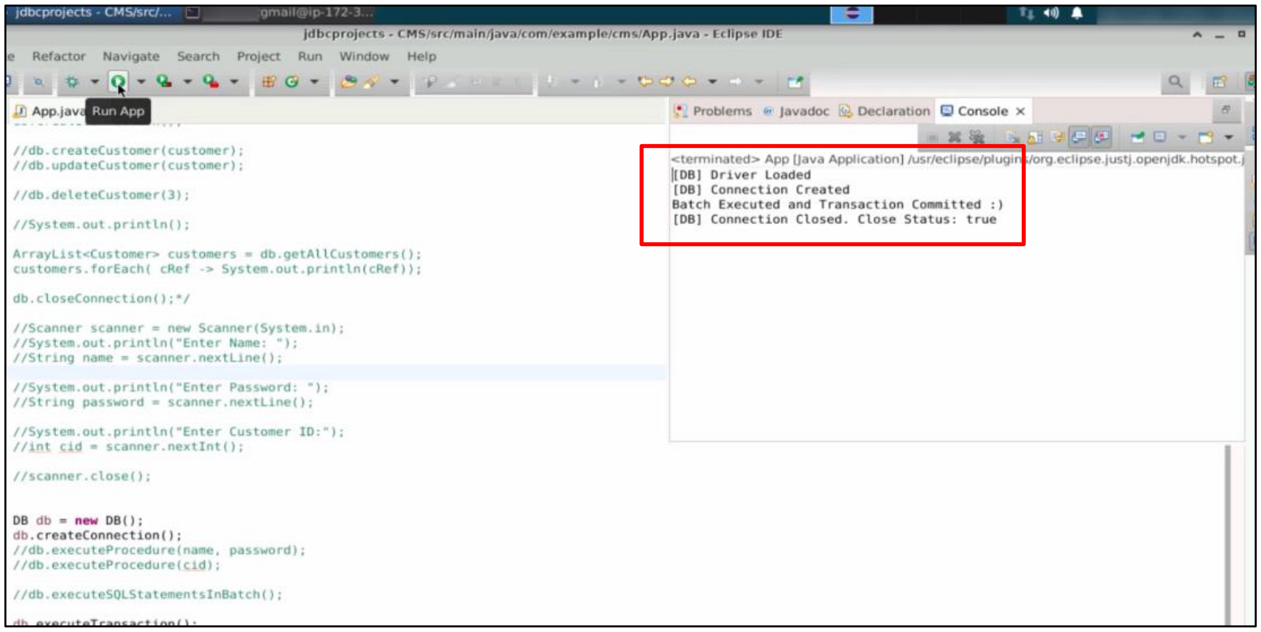
5.4 Change the column name to **orderamount** in the updated SQL code



jdbcprojects - CMS/src/main/java/com/example/cms/db/DB.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*DB.java
App.java
Writable
Smart Insert
302 : 120 : 9743

```
            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            connection.setAutoCommit(false);

            String sql1 = "insert into Orders values(null, 5, '2022-01-09', 7700)";
            String sql2 = "update Orders set orderamount = 4900 where orderId = 4";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            connection.commit(); // We will oursleves do the commit
            System.out.println("Batch Executed and Transaction Committed :)");

        } catch (Exception e) {
            System.out.println("Exception Occurred: "+e);
            try {
                System.out.println("Rolling Back the Transaction...");
                connection.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

5.5 Re-run the code, and you will see the code executed without any errors

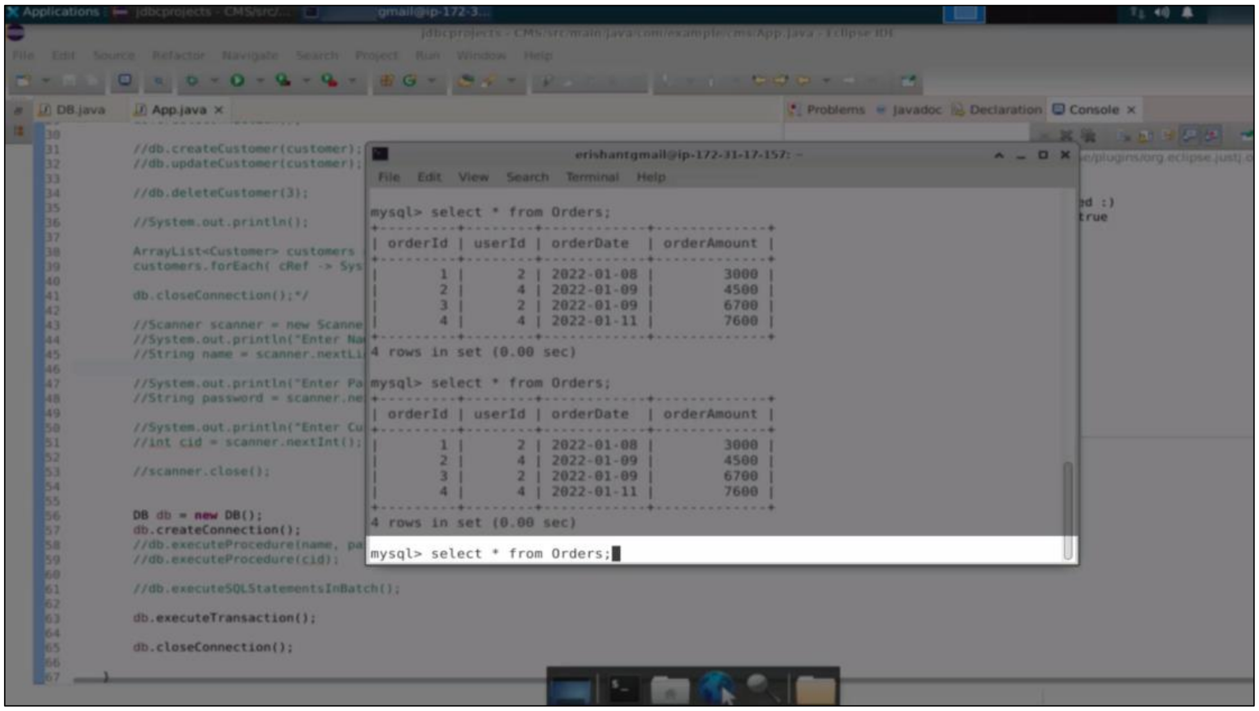


jdbcprojects - CMS/src/main/java/com/example/cms/App.java - Eclipse IDE
Refactor Navigate Search Project Run Window Help
App.java
Run App
Problems
Javadoc
Declaration
Console

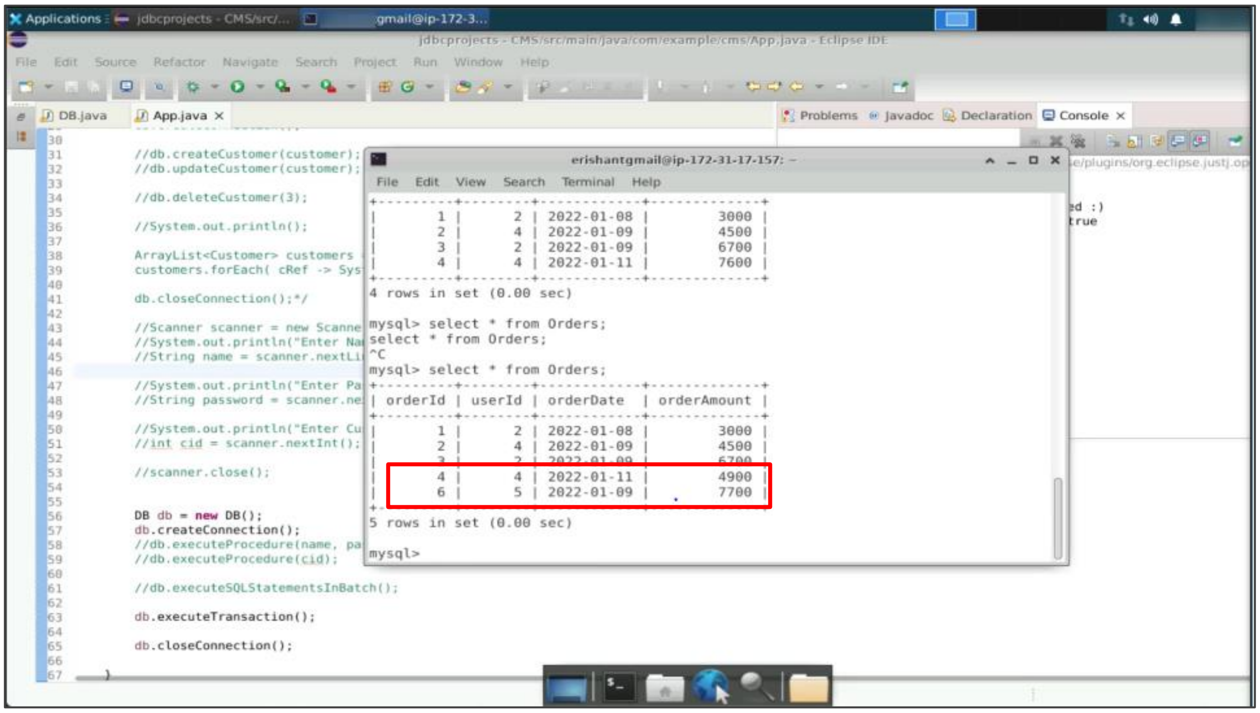
```
//db.createCustomer(customer);
//db.updateCustomer(customer);

//db.deleteCustomer(3);

//System.out.println();

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

db.closeConnection();*/

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

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

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

//scanner.close();


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

//db.executeSQLStatementsInBatch();
```

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
[DB] Driver Loaded
[DB] Connection Created
Batch Executed and Transaction Committed :)
[DB] Connection Closed. Close Status: true
```

5.6 Return to the terminal and run the **select * from Orders** command. You will see a new row added and the **orderamount** of order ID 4 gets updated.



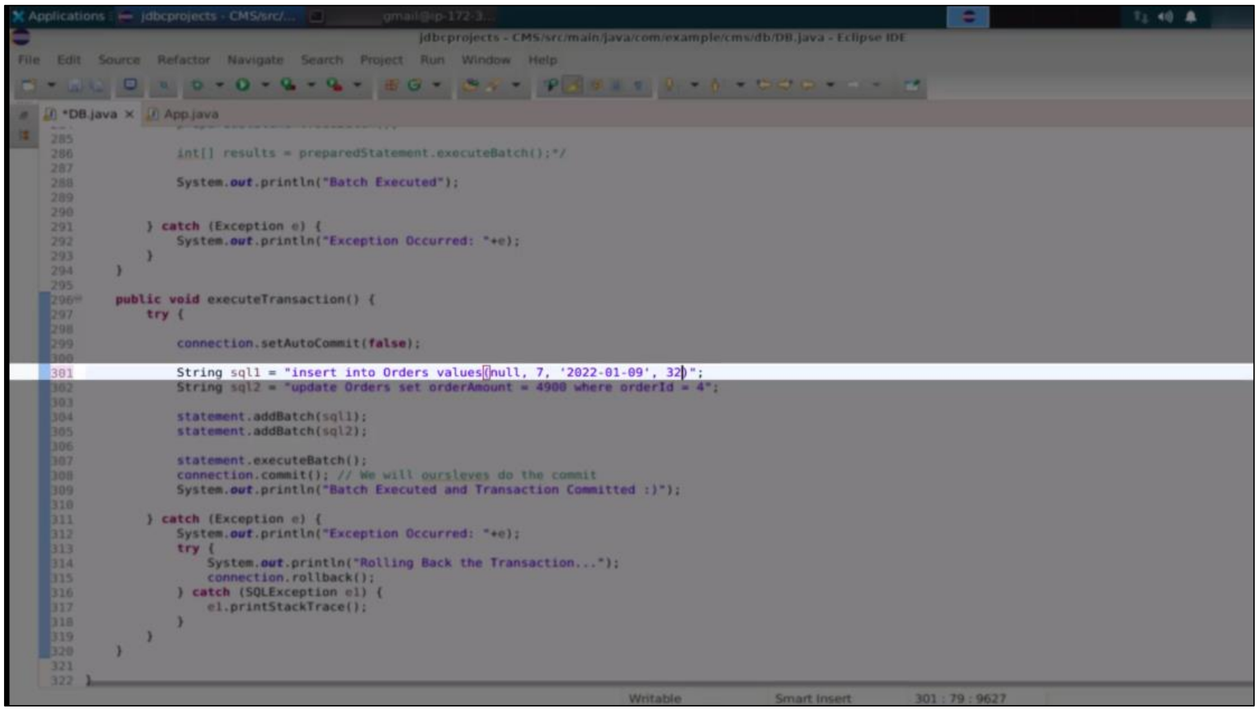
```
mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql> select * from Orders;
```



```
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        7600 |
+---------+--------+------------+-------------+
4 rows in set (0.00 sec)

mysql> select * from Orders;
select * from Orders;
^C
mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        4900 |
|       6 |      5 | 2022-01-09 |        7700 |
+---------+--------+------------+-------------+
5 rows in set (0.00 sec)

mysql>
```

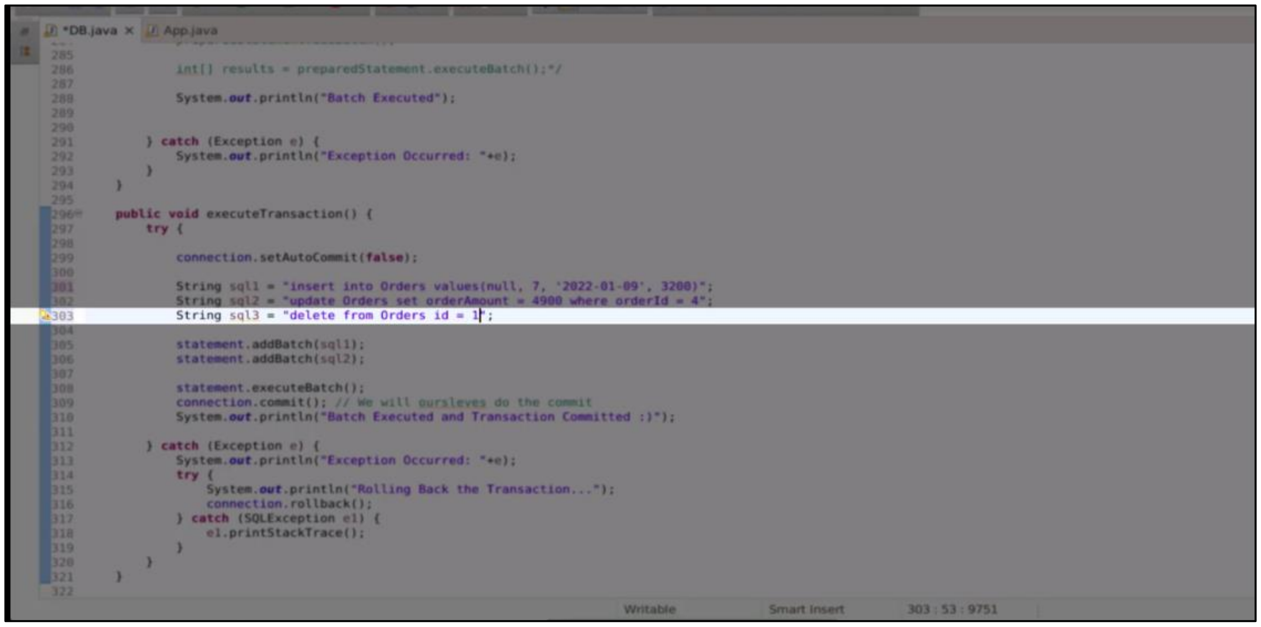
Step 6: Use the delete operation

6.1 Insert another order by changing the values in the insert statement



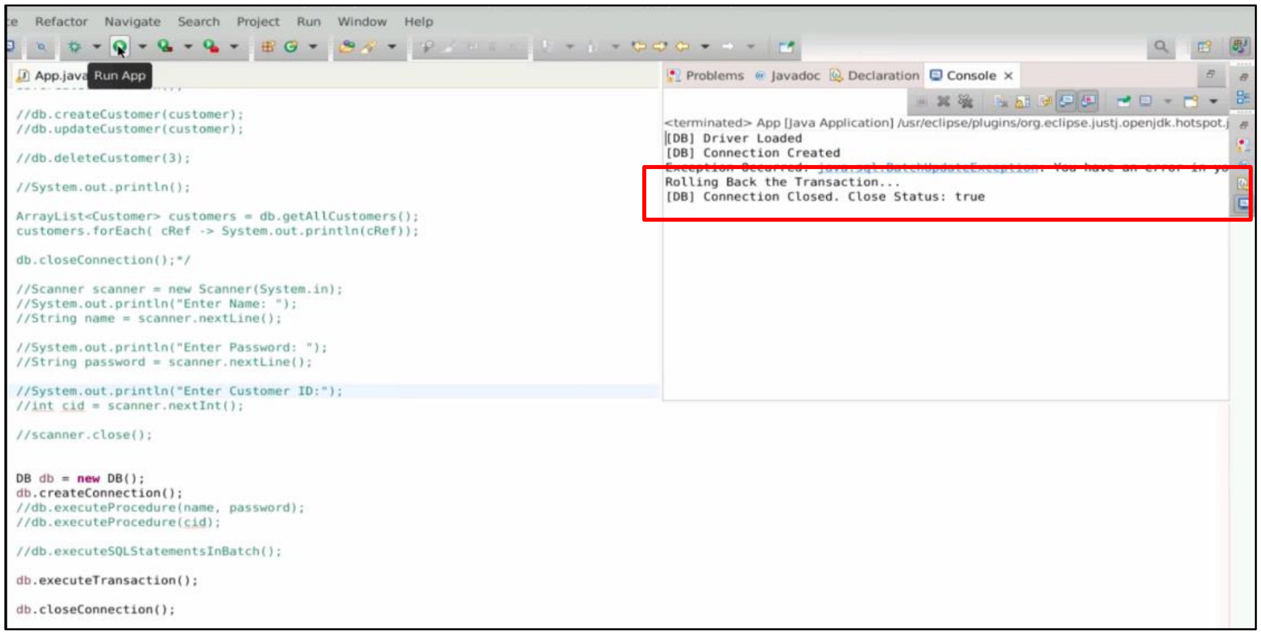
```java
            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            connection.setAutoCommit(false);

            String sql1 = "insert into Orders values(null, 7, '2022-01-09', 32)";
            String sql2 = "update Orders set orderAmount = 4900 where orderId = 4";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            connection.commit(); // We will oursleves do the commit
            System.out.println("Batch Executed and Transaction Committed :)");

        } catch (Exception e) {
            System.out.println("Exception Occurred: "+e);
            try {
                System.out.println("Rolling Back the Transaction...");
                connection.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

6.2 Write the **delete** statement



```java
            int[] results = preparedStatement.executeBatch();*/

            System.out.println("Batch Executed");


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

    public void executeTransaction() {
        try {

            connection.setAutoCommit(false);

            String sql1 = "insert into Orders values(null, 7, '2022-01-09', 3200)";
            String sql2 = "update Orders set orderAmount = 4900 where orderId = 4";
            String sql3 = "delete from Orders id = 1";

            statement.addBatch(sql1);
            statement.addBatch(sql2);

            statement.executeBatch();
            connection.commit(); // We will oursleves do the commit
            System.out.println("Batch Executed and Transaction Committed :)");

        } catch (Exception e) {
            System.out.println("Exception Occurred: "+e);
            try {
                System.out.println("Rolling Back the Transaction...");
                connection.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }
```

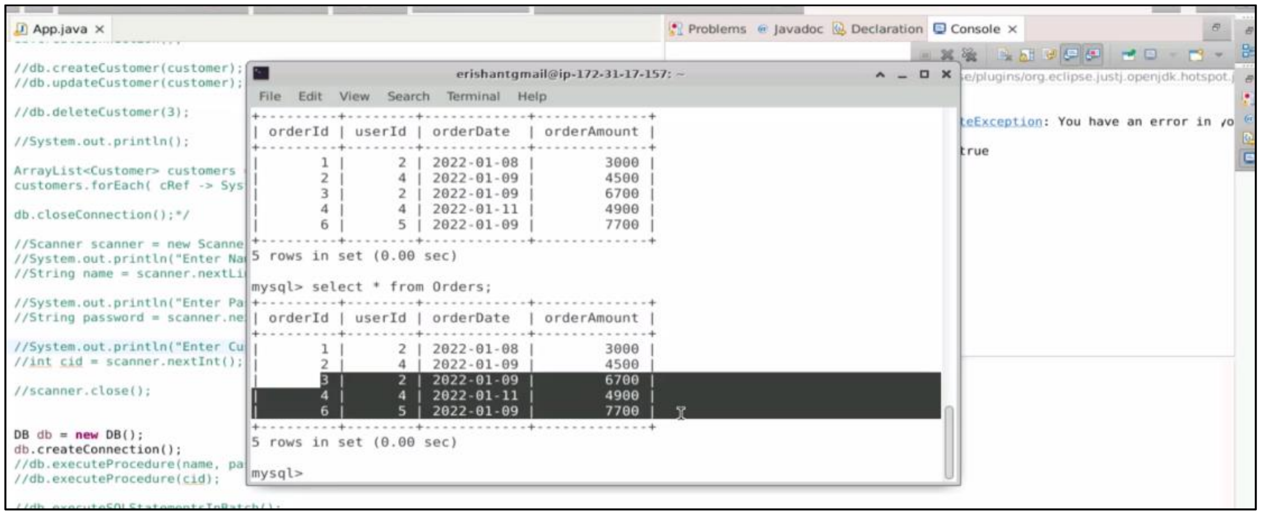
6.3 Re-run the code, and you will see the rollback function is called because one or more batches failed during execution



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

//db.deleteCustomer(3);

//System.out.println();

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

db.closeConnection();*/

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

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

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

//scanner.close();


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

//db.executeSQLStatementsInBatch();

db.executeTransaction();

db.closeConnection();
```

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
[DB] Driver Loaded
[DB] Connection Created
Exception Occurred: java.sql.BatchUpdateException: You have an error in yo
Rolling Back the Transaction...
[DB] Connection Closed. Close Status: true
```

6.4 Run the **select * from Orders;** command in the Terminal Emulator, and you will see that there is no change in the table



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

//db.deleteCustomer(3);

//System.out.println();

ArrayList<Customer> customers
customers.forEach( cRef -> Sys

db.closeConnection();*/

//Scanner scanner = new Scanne
//System.out.println("Enter Na
//String name = scanner.nextLi

//System.out.println("Enter Pa
//String password = scanner.ne

//System.out.println("Enter Cu
//int cid = scanner.nextInt();

//scanner.close();


DB db = new DB();
db.createConnection();
//db.executeProcedure(name, pa
//db.executeProcedure(cid);
```

```
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        4900 |
|       6 |      5 | 2022-01-09 |        7700 |
+---------+--------+------------+-------------+
5 rows in set (0.00 sec)

mysql> select * from Orders;
+---------+--------+------------+-------------+
| orderId | userId | orderDate  | orderAmount |
+---------+--------+------------+-------------+
|       1 |      2 | 2022-01-08 |        3000 |
|       2 |      4 | 2022-01-09 |        4500 |
|       3 |      2 | 2022-01-09 |        6700 |
|       4 |      4 | 2022-01-11 |        4900 |
|       6 |      5 | 2022-01-09 |        7700 |
+---------+--------+------------+-------------+
5 rows in set (0.00 sec)

mysql>
```

```
se/plugins/org.eclipse.justj.openjdk.hotspot.

teException: You have an error in yo

true
```

By following these steps, you have successfully managed transaction execution to perform all SQL statements together, ensuring data integrity and consistency within the database.