CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

LAB ASSIGNMENT - 4

Banner ID: B00948977

Git Assignment Link :
https://git.cs.dal.ca/sukumaran/csci5408_f23_b00948977_balaji_sukumaran

Table of contents

Problem Statement 1: **Create a local database with –**

User table (attributes – id, name, email, phone, address)

Created the User table in local database.

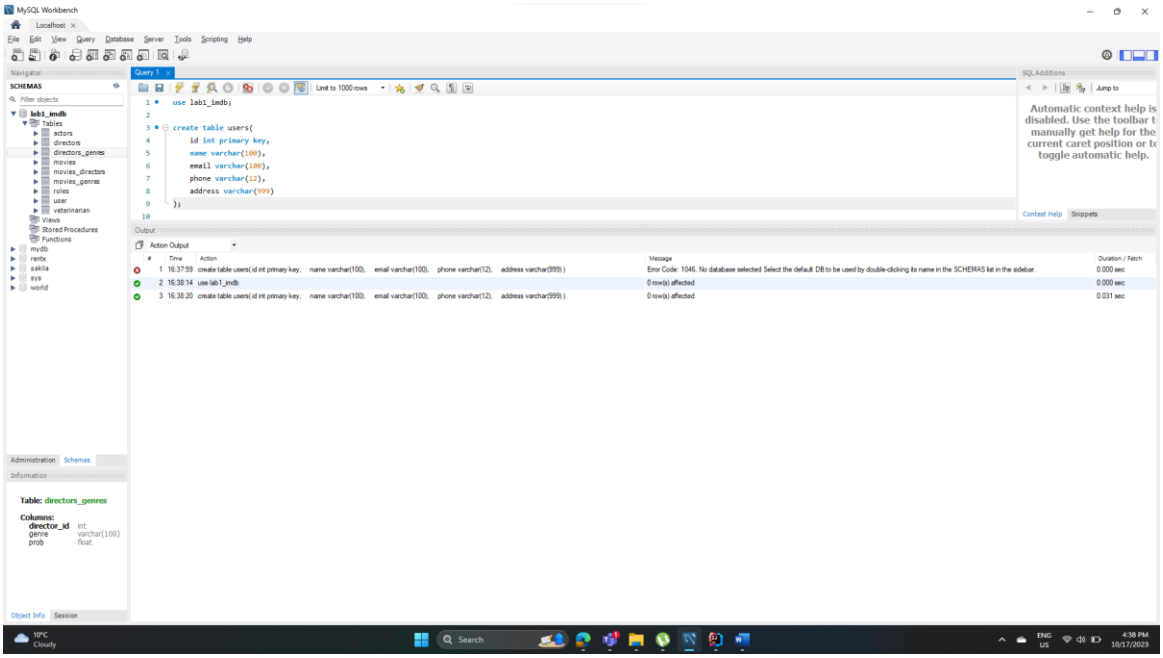


Figure 1: create statement for user in local database

Inserted sample data in user table

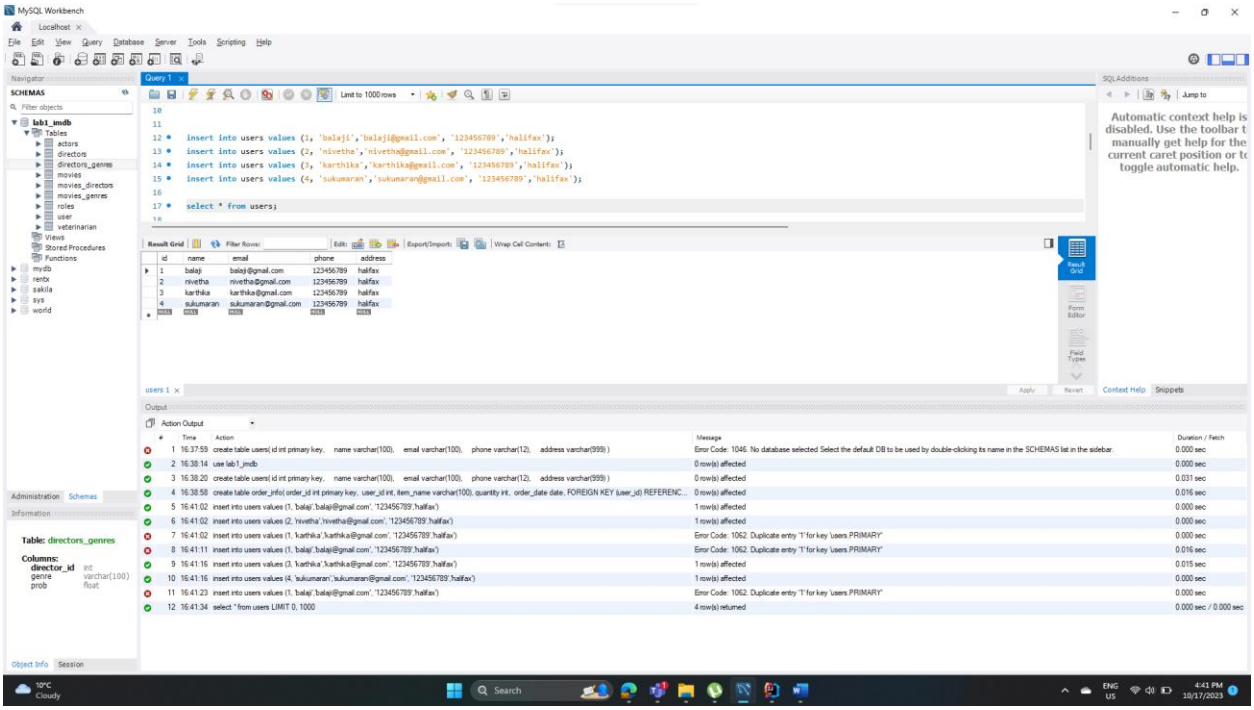


Figure 2: Sample data for users table

Order_info table (order_id, user_id,item_name, quantity, order_date)

Created the order info table in local database

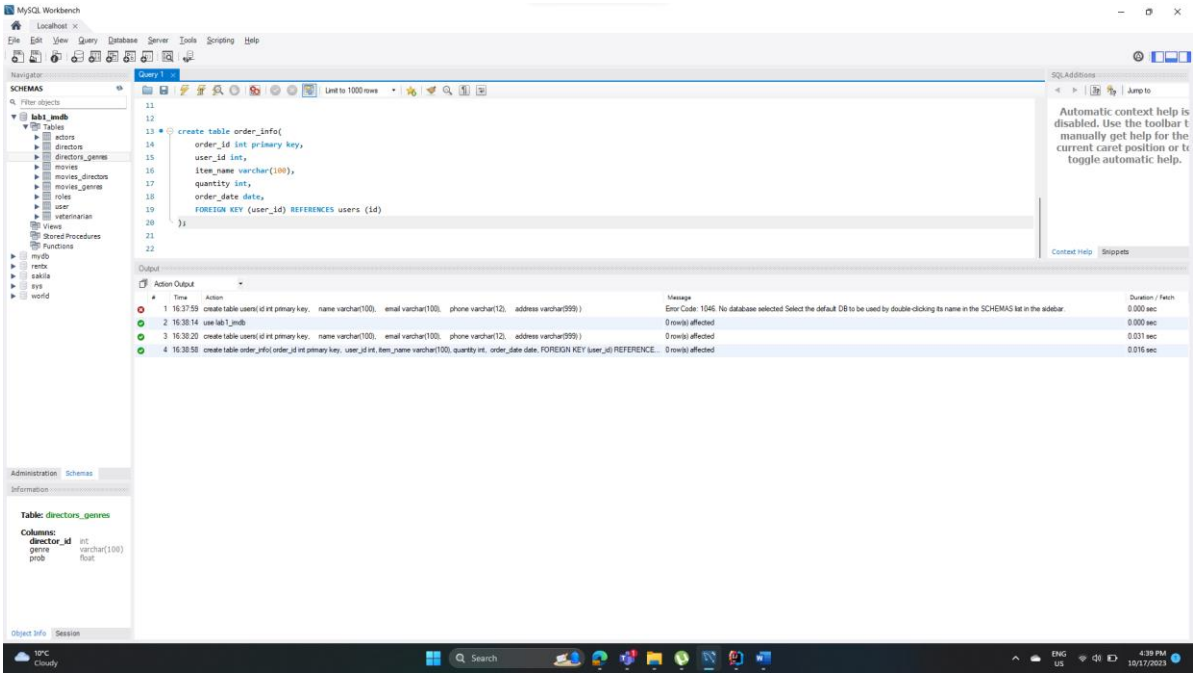


Figure 3: create statment for order_info table in local database

Problem Statement 2: Create a remote database in G C P with Inventory table (item_id,item_name, available_quantity)

Created a GCP MySql instance

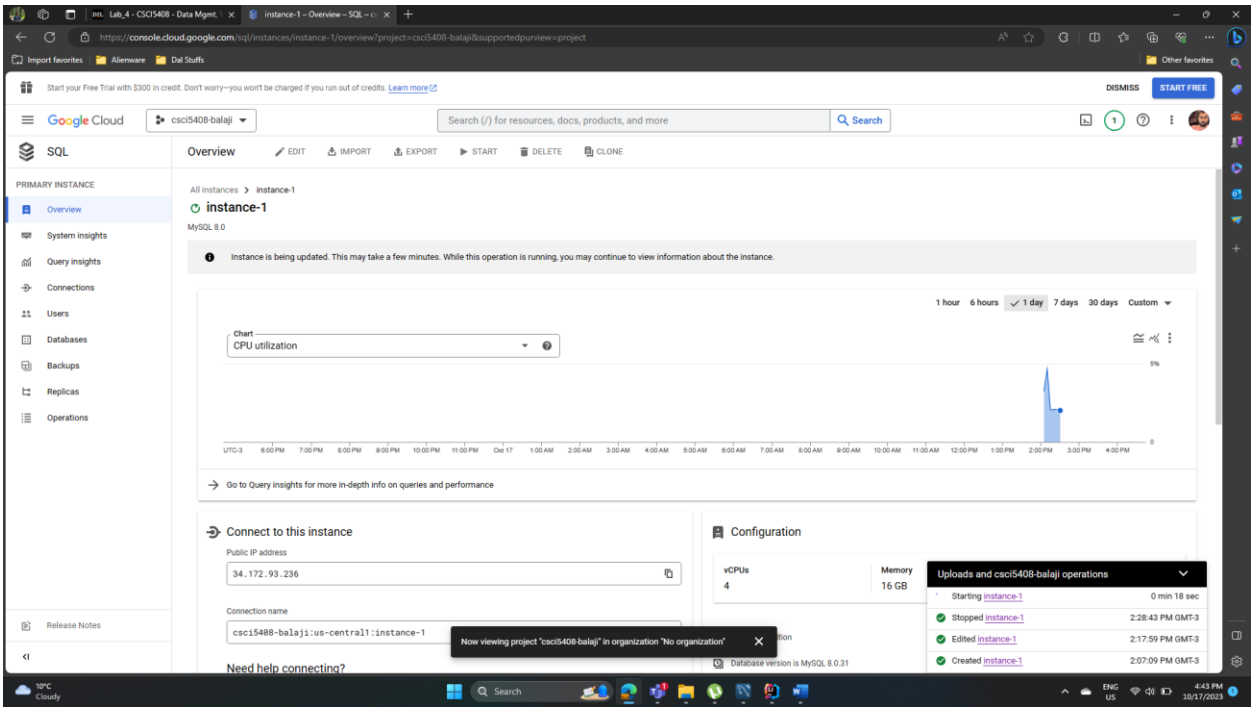


Figure 4: Screenshot of GCP MySql instance

Created an Inventory table in the remote MySql instance and inserted sample data

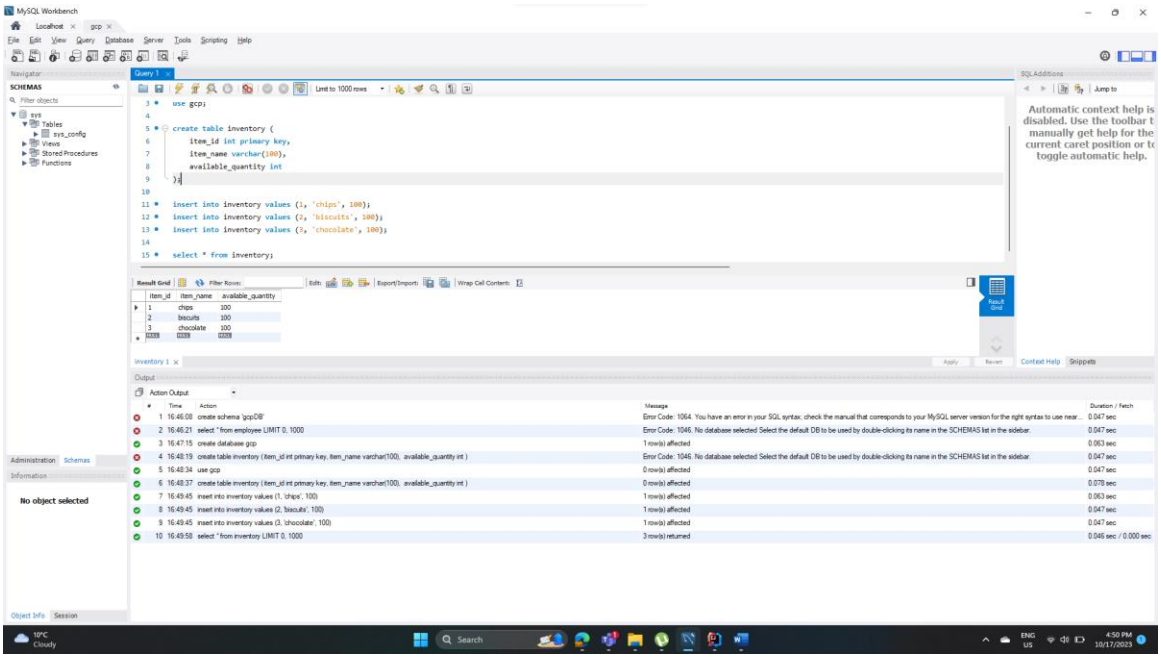
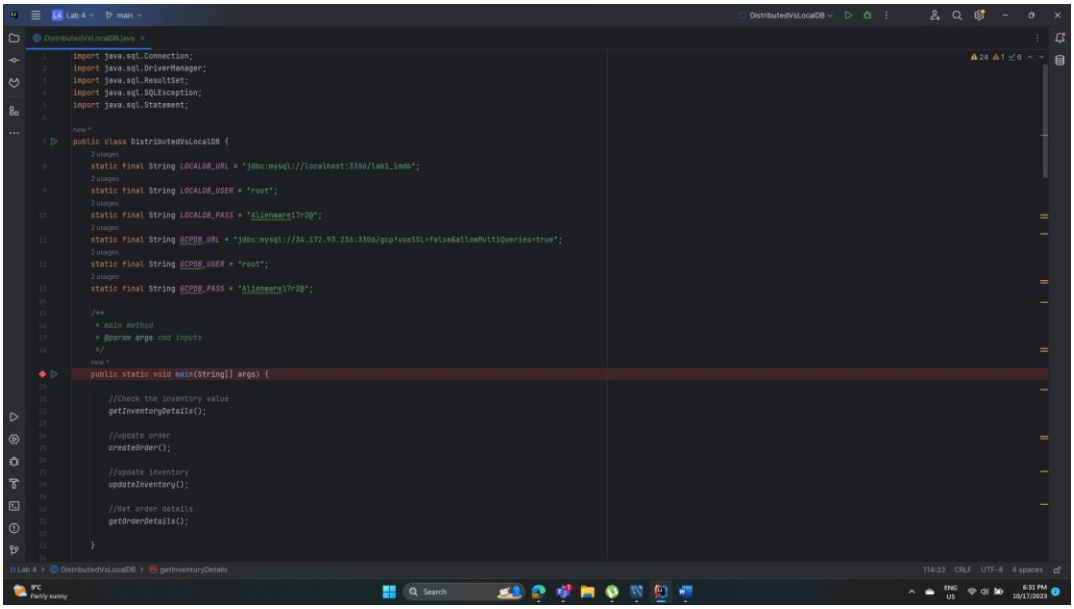


Figure 5: Create and insert commands for inventory table

Problem Statement 3: **Write a Java program that –**

- **Fetches item details from the remote database**
- **Creates an order in local database**
- **Writes the updated quantity back to the remote database**

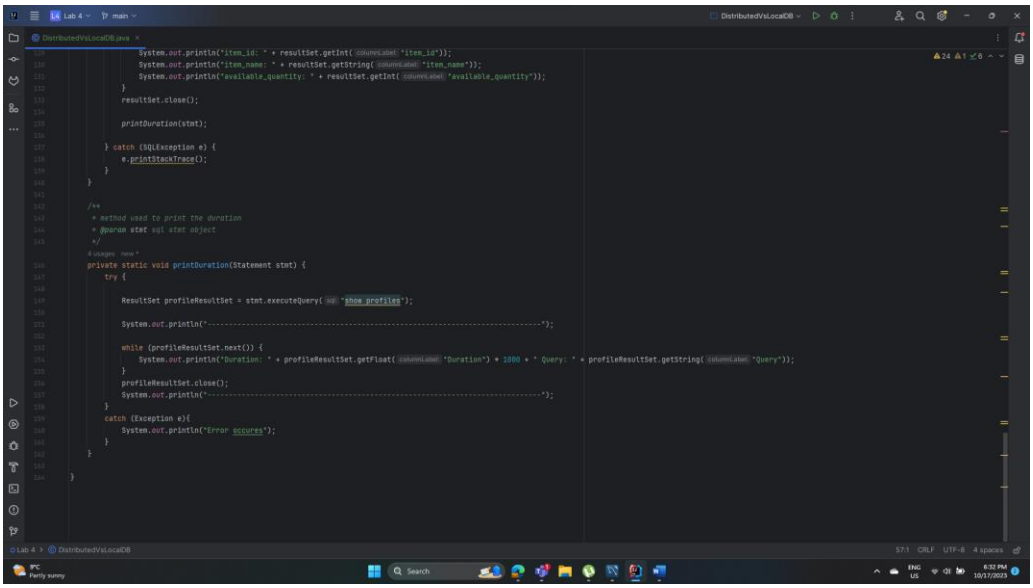
Created a Java program



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DistributedVsLocalDB {
    static final String LOCALDB_URL = "jdbc:mysql://localhost:3306/lab1_imdb";
    static final String LOCALDB_USER = "root";
    static final String LOCALDB_PASS = "Alienware17r2@";
    static final String GCPDB_URL = "jdbc:mysql://34.172.93.236:3306/gcp?useSSL=false&allowMultiQueries=true";
    static final String GCPDB_USER = "root";
    static final String GCPDB_PASS = "Alienware17r2@";

    /**
     * main method
     * @param args cmd inputs
     */
    public static void main(String[] args) {

        //Check the inventory value
        getInventoryDetails();

        //update order
        createOrder();

        //update inventory
        updateInventory();

        //Get order details
        getOrderDetails();

    }
```

Figure 6: java program main method

Turned the profiling on and used show profiles to get the intermediate durations



```
                System.out.println("item_id: " + resultSet.getInt( columnLabel: "item_id"));
                System.out.println("item_name: " + resultSet.getString( columnLabel: "item_name"));
                System.out.println("available_quantity: " + resultSet.getInt( columnLabel: "available_quantity"));
            }
            resultSet.close();

            printDuration(stmt);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * method used to print the duration
     * @param stmt sql stmt object
     */
    4 usages  new *
    private static void printDuration(Statement stmt) {
        try {

            ResultSet profileResultSet = stmt.executeQuery( sql: "show profiles");

            System.out.println("----------------------------------------------------------------------------");

            while (profileResultSet.next()) {
                System.out.println("Duration: " + profileResultSet.getFloat( columnLabel: "Duration") * 1000 + " Query: " + profileResultSet.getString( columnLabel: "Query"));
            }
            profileResultSet.close();
            System.out.println("----------------------------------------------------------------------------");
        }
        catch (Exception e){
            System.out.println("Error occures");
        }
    }
}
```

Figure 7: method to print the SQL execution duration

Which executes the following business logic in order:

1. Get the inventory details – touches the remote DB

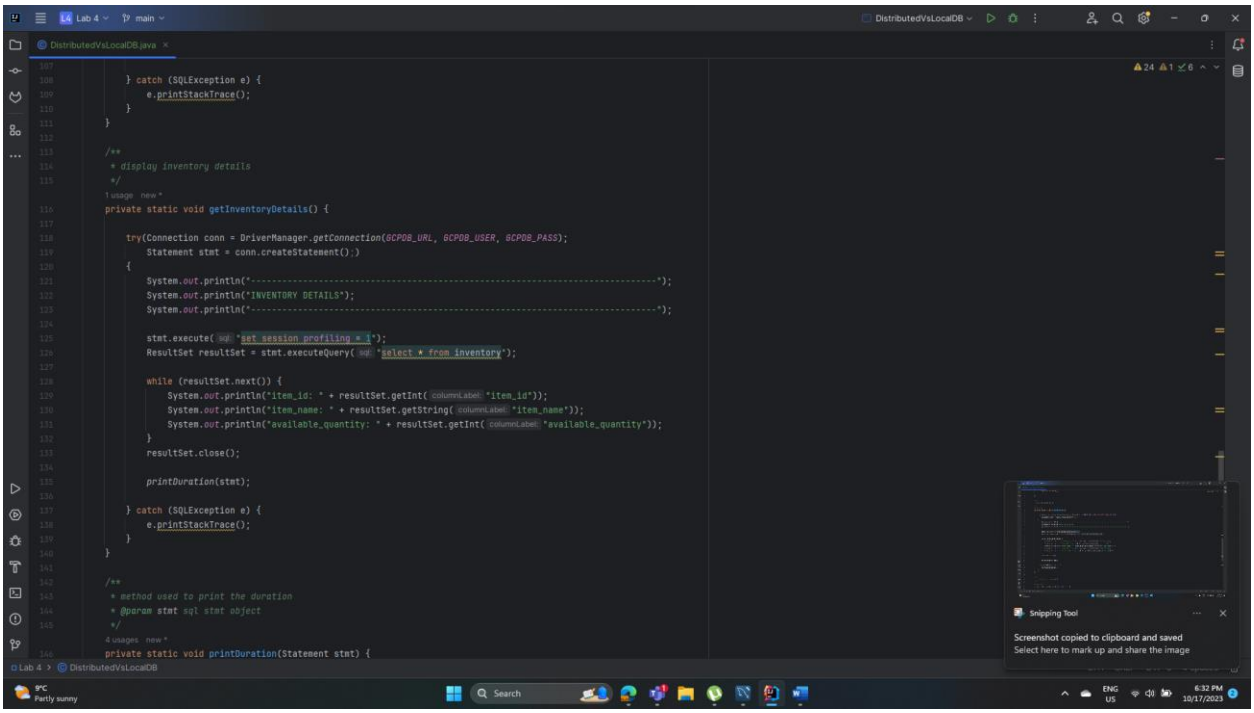


Figure 8: method to get the remote inventory details

2. Create the order – touches the local DB

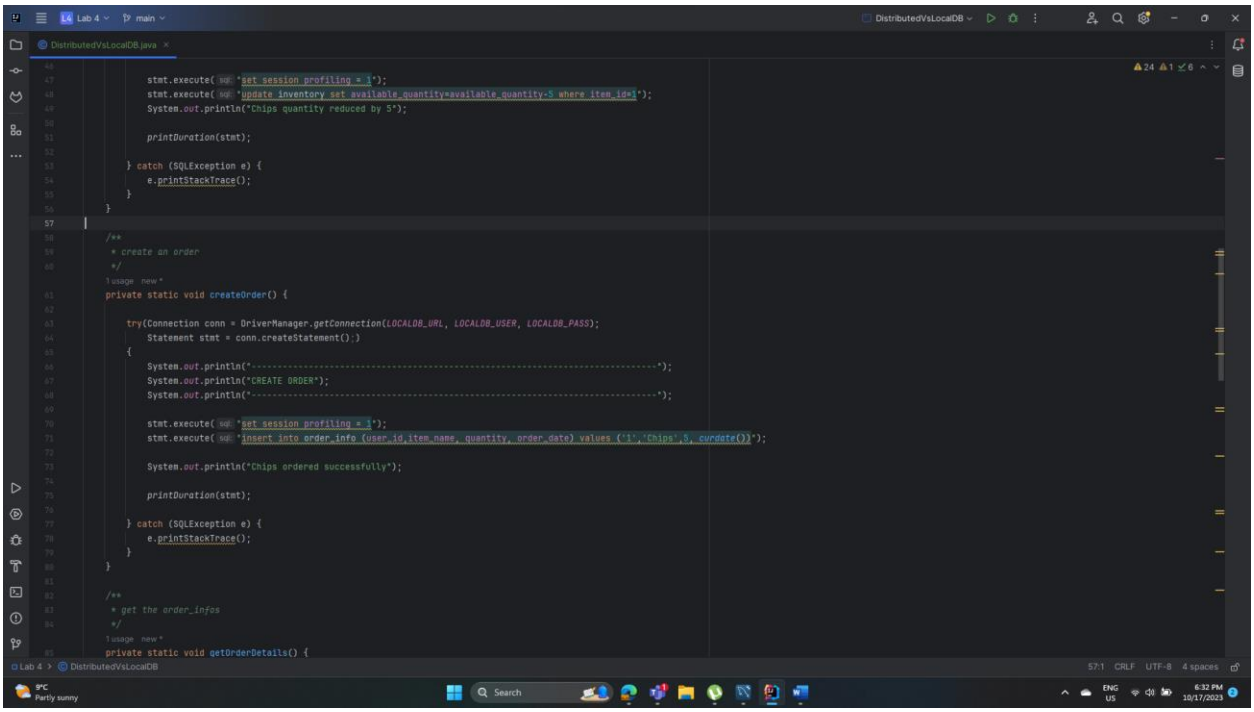
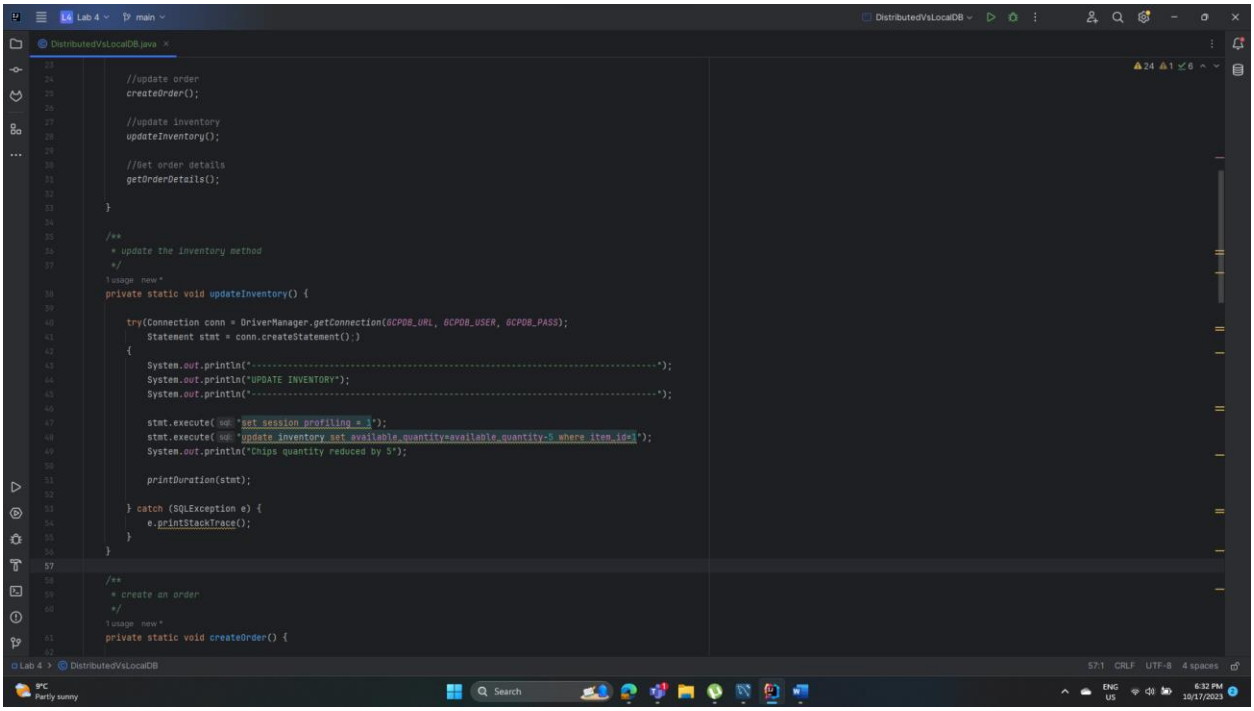


Figure 9: method to create the order

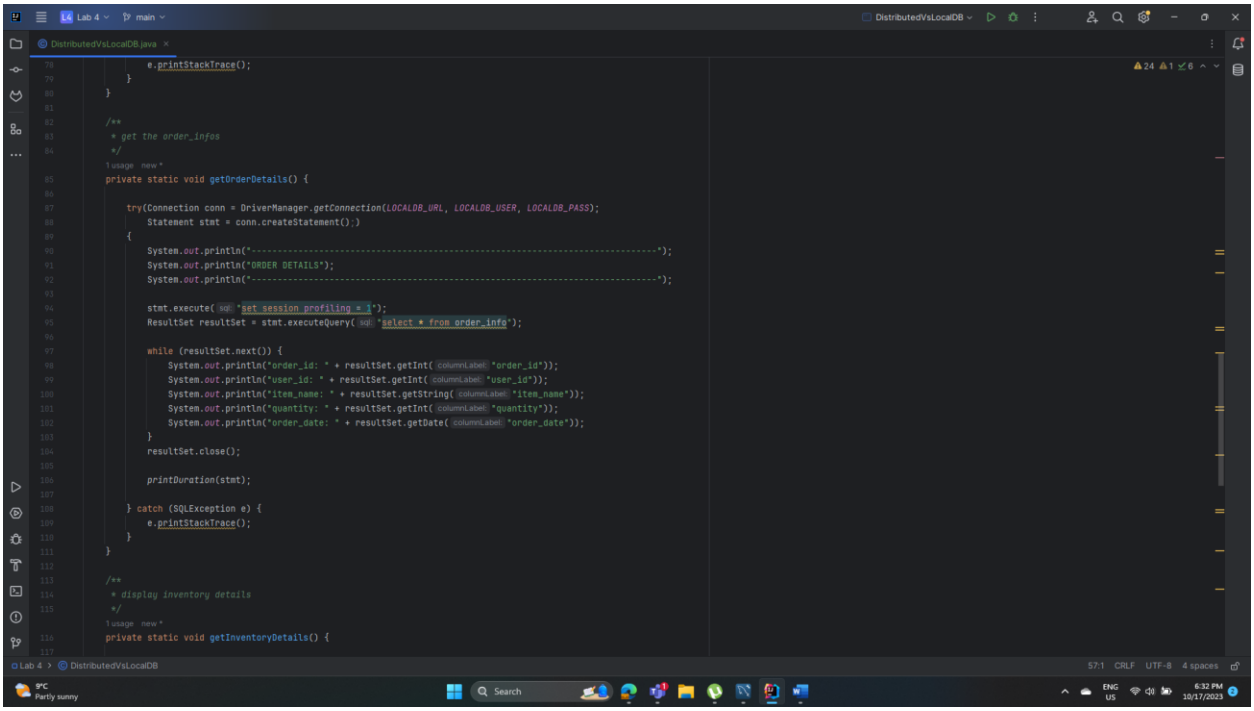
3. Update the inventory – touches the remote DB



```
        //update order
        createOrder();

        //update inventory
        updateInventory();

        //Get order details
        getOrderDetails();

    }

    /**
     * update the inventory method
     */
    1 usage  new *
    private static void updateInventory() {

        try(Connection conn = DriverManager.getConnection(BCPDB_URL, BCPDB_USER, BCPDB_PASS);
            Statement stmt = conn.createStatement();)
        {
            System.out.println("--------------------------------------------------------------------------------");
            System.out.println("UPDATE INVENTORY");
            System.out.println("--------------------------------------------------------------------------------");

            stmt.execute( sql: "set session profiling = 1");
            stmt.execute( sql: "update inventory set available_quantity=available_quantity-5 where item_id=1");
            System.out.println("Chips quantity reduced by 5");

            printDuration(stmt);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * create an order
     */
    1 usage  new *
    private static void createOrder() {
```

Figure 10: method to update the remote inventory

4. Get the order details - touches the local DB



```
            e.printStackTrace();
        }
    }

    /**
     * get the order_infos
     */
    1 usage  new *
    private static void getOrderDetails() {

        try(Connection conn = DriverManager.getConnection(LOCALDB_URL, LOCALDB_USER, LOCALDB_PASS);
            Statement stmt = conn.createStatement();)
        {
            System.out.println("--------------------------------------------------------------------------------");
            System.out.println("ORDER DETAILS");
            System.out.println("--------------------------------------------------------------------------------");

            stmt.execute( sql: "set session profiling = 1");
            ResultSet resultSet = stmt.executeQuery( sql: "select * from order_info");

            while (resultSet.next()) {
                System.out.println("order_id: " + resultSet.getInt( columnLabel: "order_id"));
                System.out.println("user_id: " + resultSet.getInt( columnLabel: "user_id"));
                System.out.println("item_name: " + resultSet.getString( columnLabel: "item_name"));
                System.out.println("quantity: " + resultSet.getInt( columnLabel: "quantity"));
                System.out.println("order_date: " + resultSet.getDate( columnLabel: "order_date"));
            }
            resultSet.close();

            printDuration(stmt);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * display inventory details
     */
    1 usage  new *
    private static void getInventoryDetails() {
```

Figure 11: Method to get the order details

Problem Statement 4: Print query execution time at every step.

(try to deduce the reason behind the time differences)

Note: converted the duration to X1000 for readability

As we can see here the time taken to insert into the local DB table order_info took only 1.4605 whereas the updating the remote inventory took 5.70075 which is significantly higher than the local,

reason could be the following,

Since the remote DB is in different geographical location than the local, delay could be due to the network latency to fetch from different geographical location or Network setup in the local works best for the locally executed java program than the network setup of remote Database

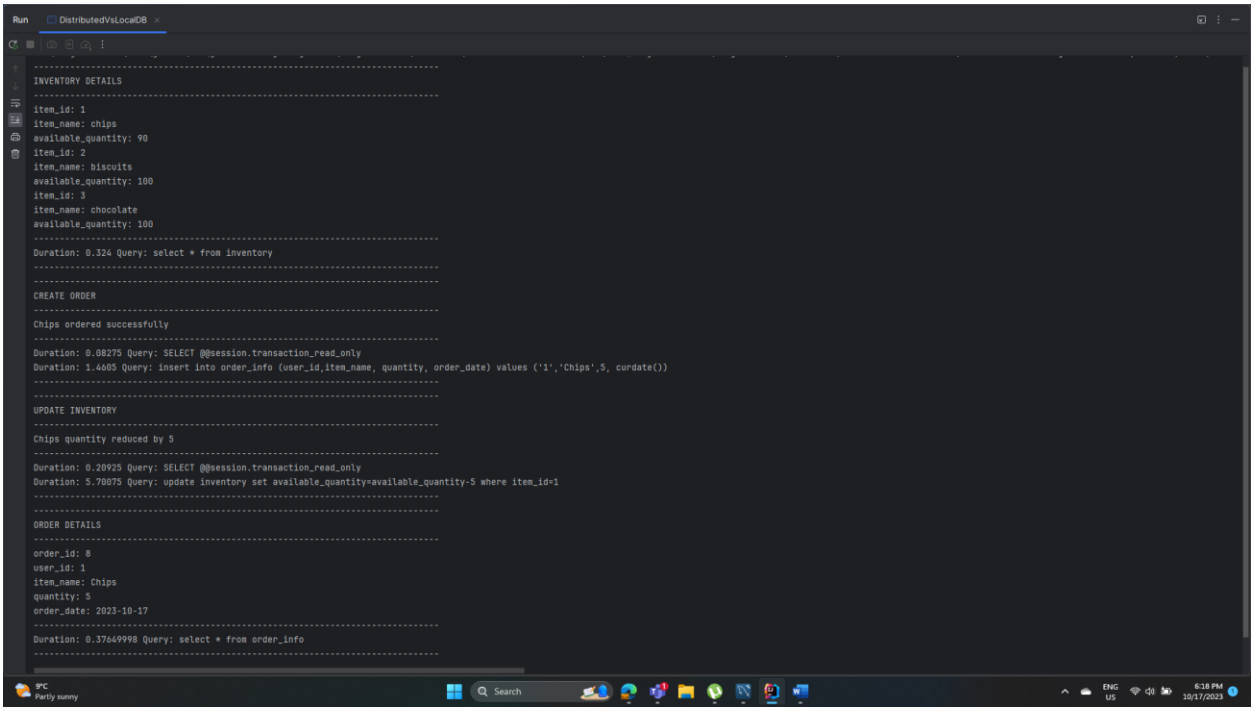


Figure 12: Java program output