JAVA FOUNDATION TRAINING PROGRAM

ASSIGNMENT TASKS

Name: Kavin Kaarthik Date: 20/03/25

Batch No.: 3 Superset ID: 5371276  
  
  
Task 1:   
Create schema using these queries  
CREATE TABLE User (

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255) NOT NULL,

Email VARCHAR(255) UNIQUE NOT NULL,

Password VARCHAR(255) NOT NULL,

ContactNumber VARCHAR(20) NOT NULL,

Address TEXT NOT NULL

);

CREATE TABLE Courier (

CourierID INT PRIMARY KEY AUTO\_INCREMENT,

SenderUserID INT NOT NULL,

ReceiverUserID INT NOT NULL,

PackageName VARCHAR(255) NOT NULL,

NumberOfPackages INT NOT NULL DEFAULT 1,

Weight DECIMAL(5,2) NOT NULL,

Status ENUM('Processing', 'Delivered', 'Cancelled') NOT NULL,

TrackingNumber VARCHAR(20) UNIQUE NOT NULL,

CreatedDate Date NOT NULL,

DeliveryDate DATE,

ServiceID INT NOT NULL,

EmployeeID INT,

FOREIGN KEY (SenderUserID) REFERENCES User(UserID) ON DELETE CASCADE,

FOREIGN KEY (ReceiverUserID) REFERENCES User(UserID) ON DELETE CASCADE,

FOREIGN KEY (ServiceID) REFERENCES CourierServices(ServiceID) ON DELETE CASCADE,

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID) ON DELETE SET NULL

);

CREATE TABLE CourierServices (

ServiceID INT PRIMARY KEY AUTO\_INCREMENT,

ServiceName VARCHAR(100) NOT NULL,

Cost DECIMAL(8,2) NOT NULL

);

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT UNIQUE NOT NULL,

Role VARCHAR(50) NOT NULL,

Salary DECIMAL(10,2) NOT NULL,

FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE

);

CREATE TABLE Location (

LocationID INT PRIMARY KEY AUTO\_INCREMENT,

LocationName VARCHAR(100) NOT NULL,

Address TEXT NOT NULL

);

CREATE TABLE Payment (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

CourierID INT NOT NULL UNIQUE,

LocationID INT NOT NULL,

EmployeeID INT,

Amount DECIMAL(10,2) NOT NULL,

PaymentDate DATE NOT NULL,

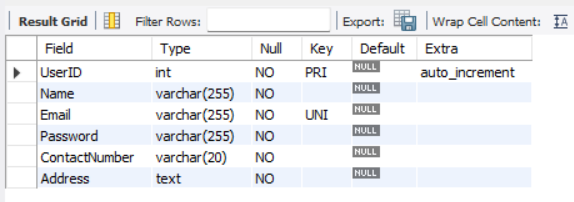
FOREIGN KEY (CourierID) REFERENCES Courier(CourierID) ON DELETE CASCADE,

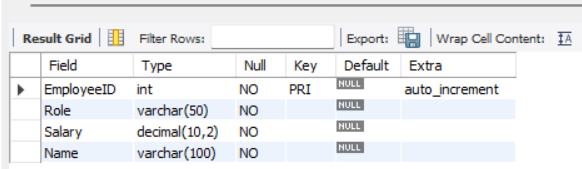
FOREIGN KEY (LocationID) REFERENCES Location(LocationID) ON DELETE CASCADE,

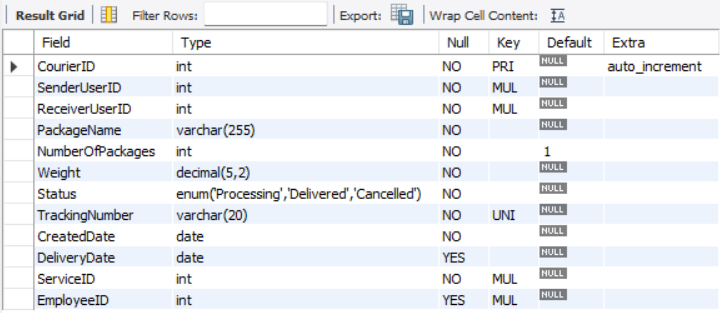
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID) ON DELETE SET NULL

);

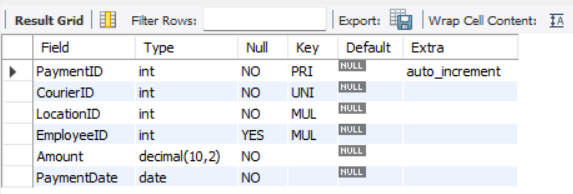
This is the schematic of each table created:  
  
User table:

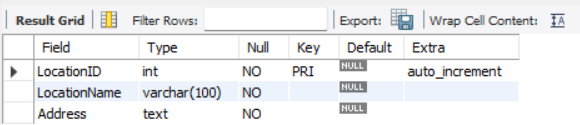


Employee table:   


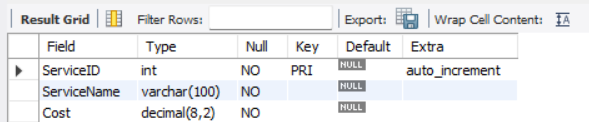
Courier table:   


Payment Table:



Location Table:   


Courier Services Table:



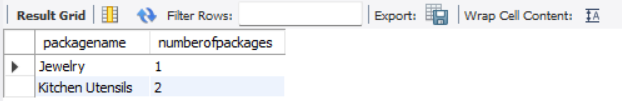
Task 2:

1. List all customers: select Name from user;



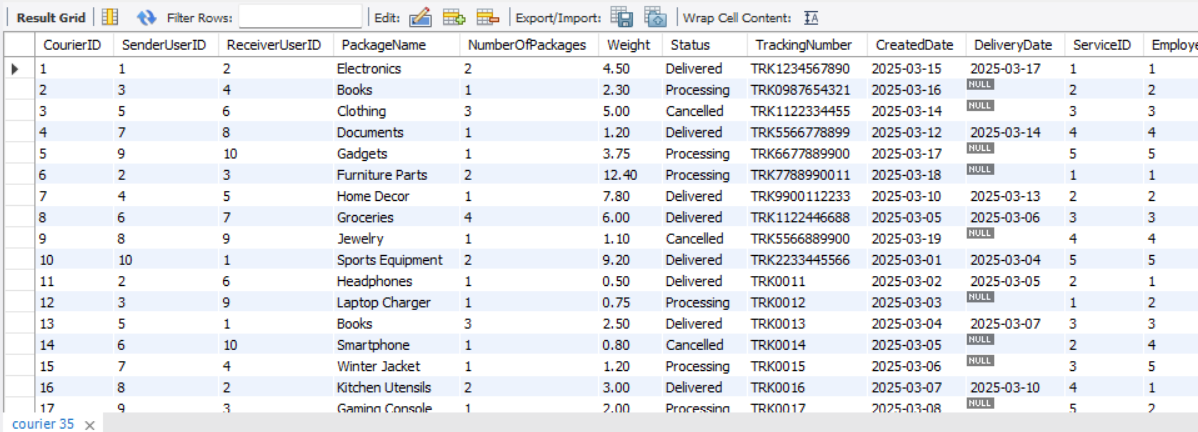
1. List all orders for a specific customer:

select packagename, numberofpackages from courier where senderuserid=8;



1. List all couriers:

select \* from courier;

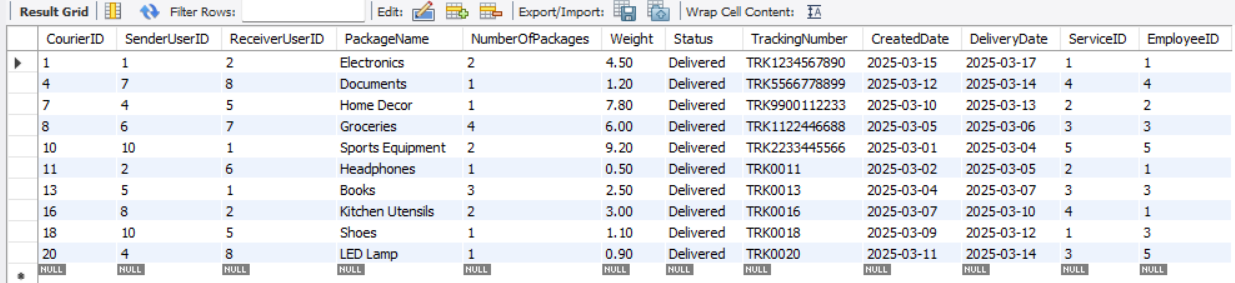


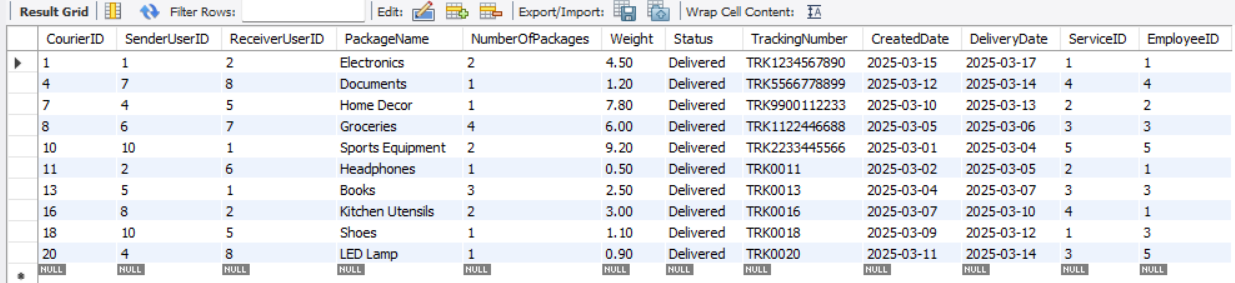
4. List all packages for a specific order:

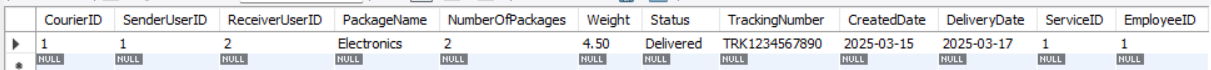
select packagename, numberofpackages from courier where courierid =13;



1. List all deliveries for a specific courier:  
   select \* from courier where status='delivered';



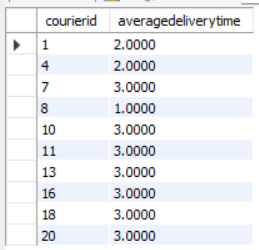
6. List all undelivered packages:  
select \* from courier where not status='delivered';  


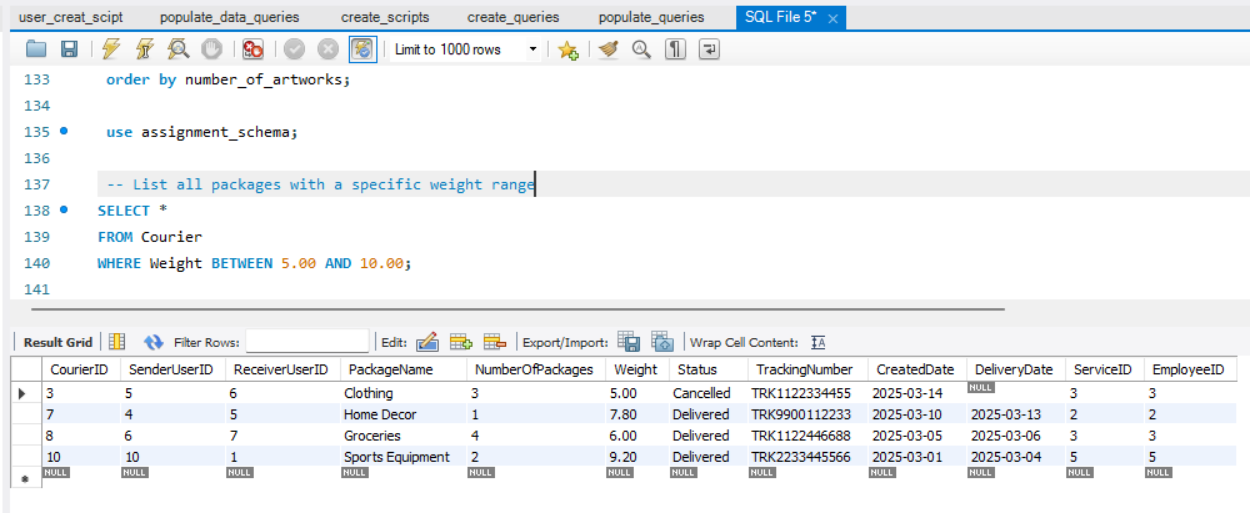
7. List all packages that are scheduled for delivery today:   
select \* from courier where DeliveryDate='2025-03-17';  


8. List all packages with a specific status:   
select \* from courier where status ='cancelled';  

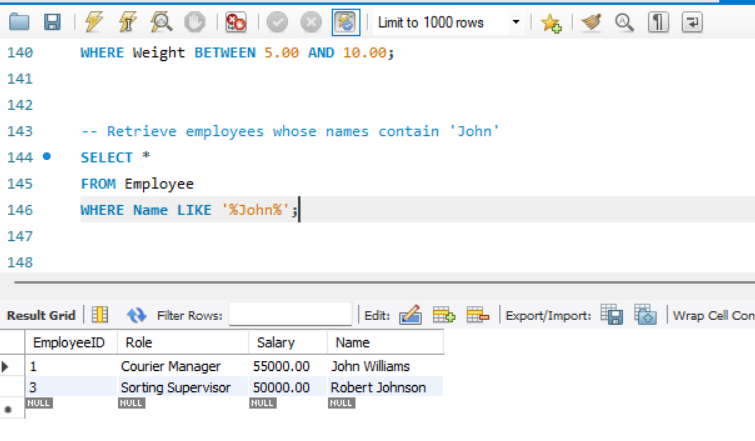

9. Calculate the total number of packages for each courier:  
select numberofpackages from courier where CourierID=3;  


10. Find the average delivery time for each courier

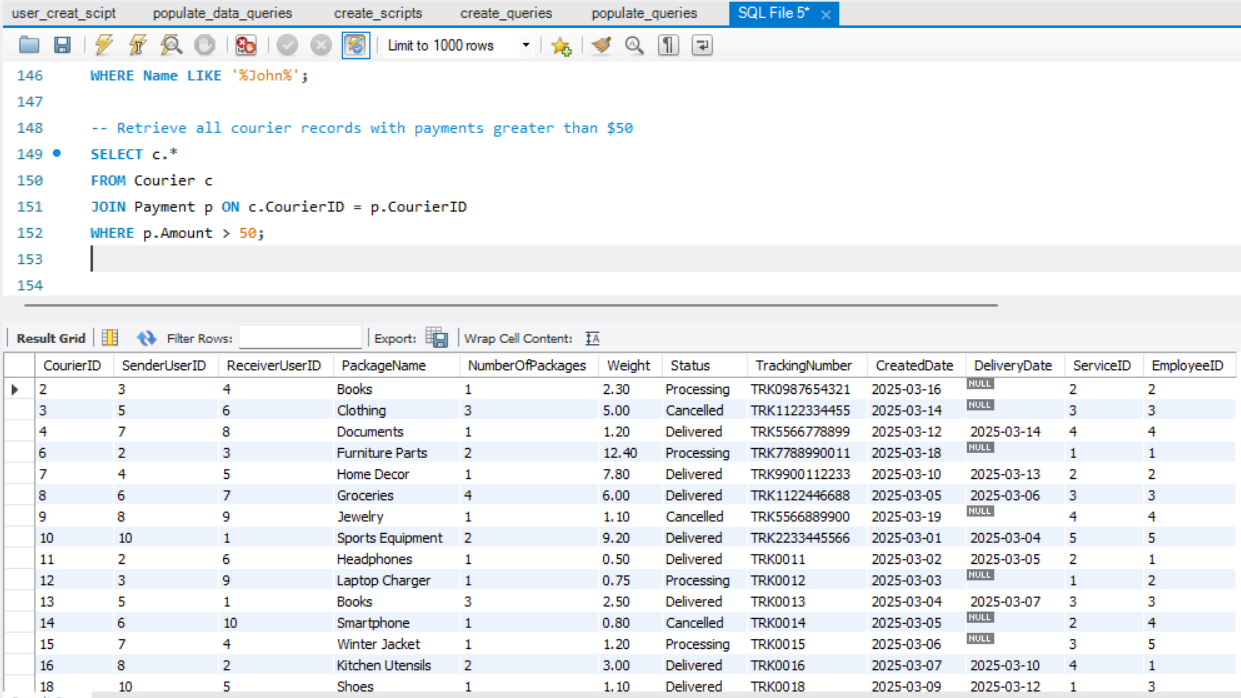
select courierid, avg(datediff(deliverydate, createddate)) as averagedeliverytime from courier where status='delivered' group by courierid;  


11. 

12. Retrieve employees whose names contain 'John'

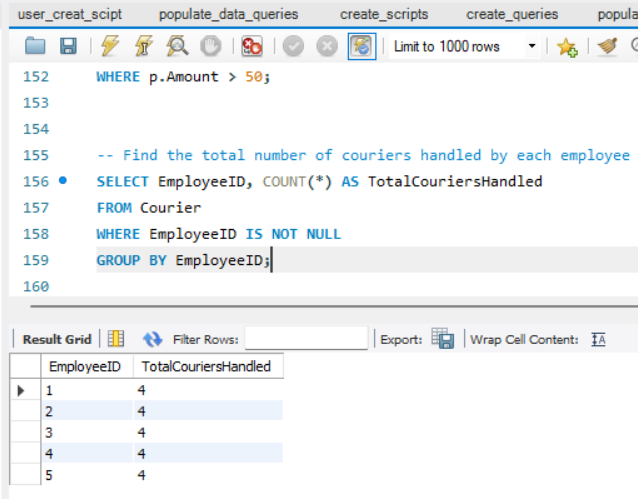


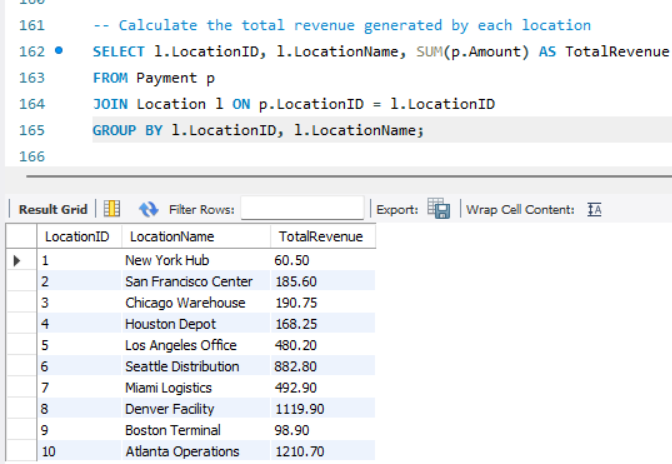
13. Retrieve all courier records with payments greater than $50.



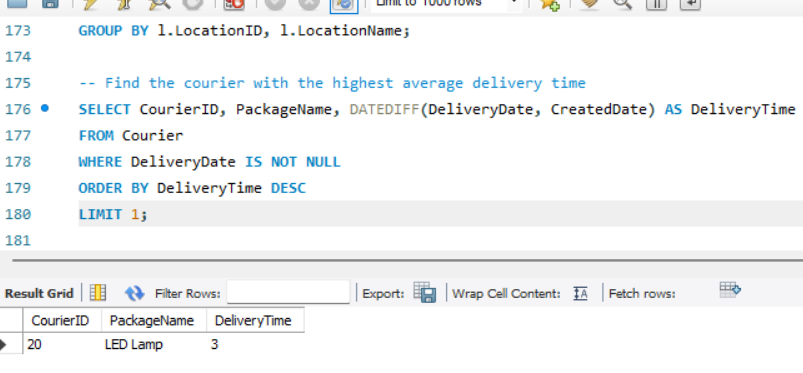
Task:3:

14.

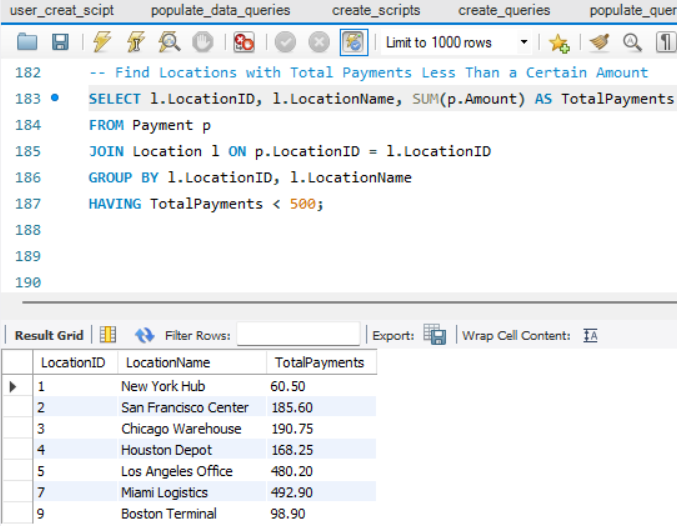
.

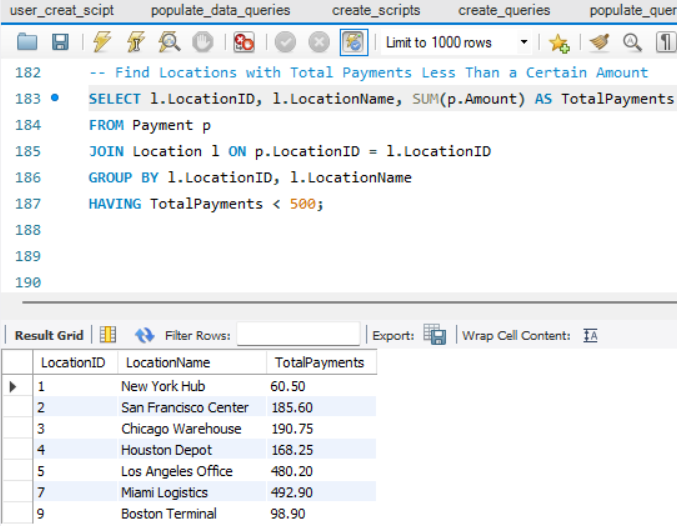
15. 

17. Find the courier with the highest average delivery time

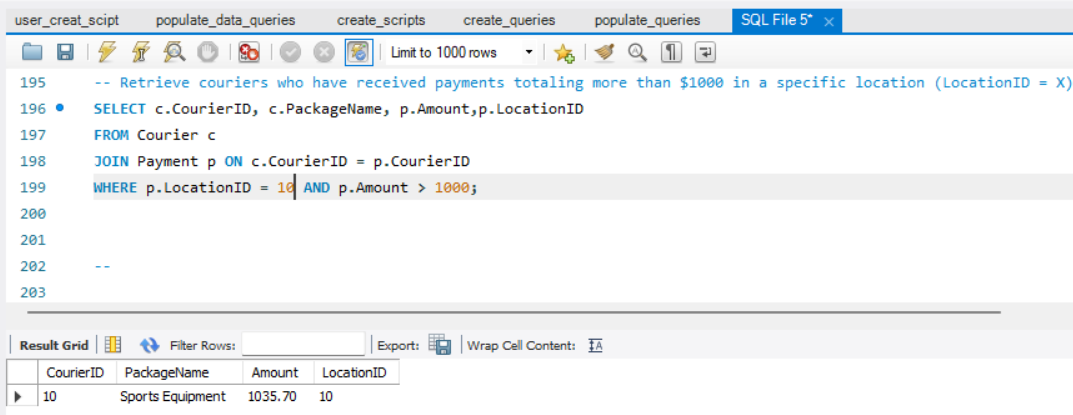


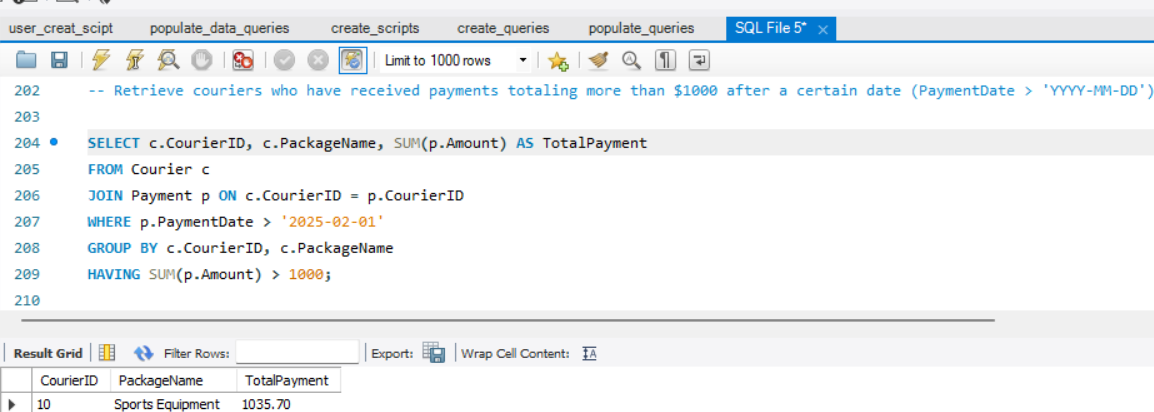
18. Find Locations with Total Payments Less Than a Certain Amount



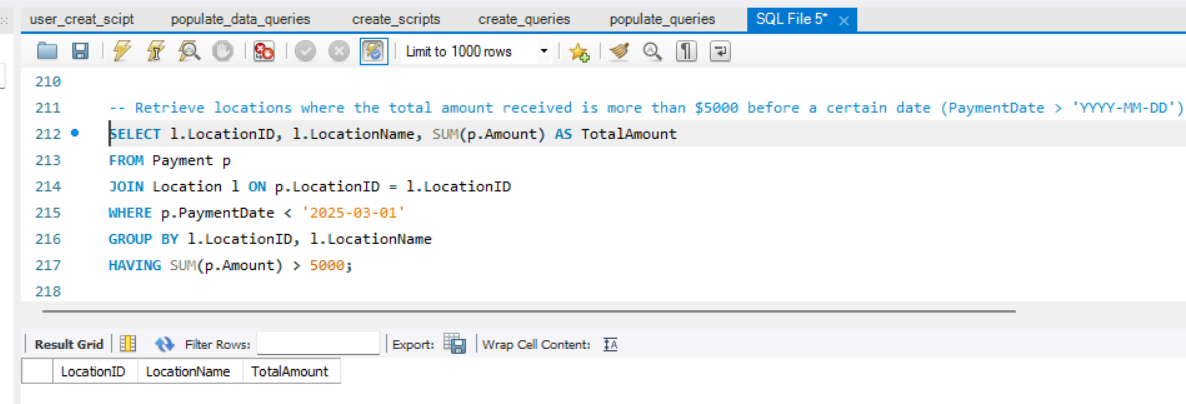
19. 

20. Retrieve couriers who have received payments totalling more than $1000 in a specific location.



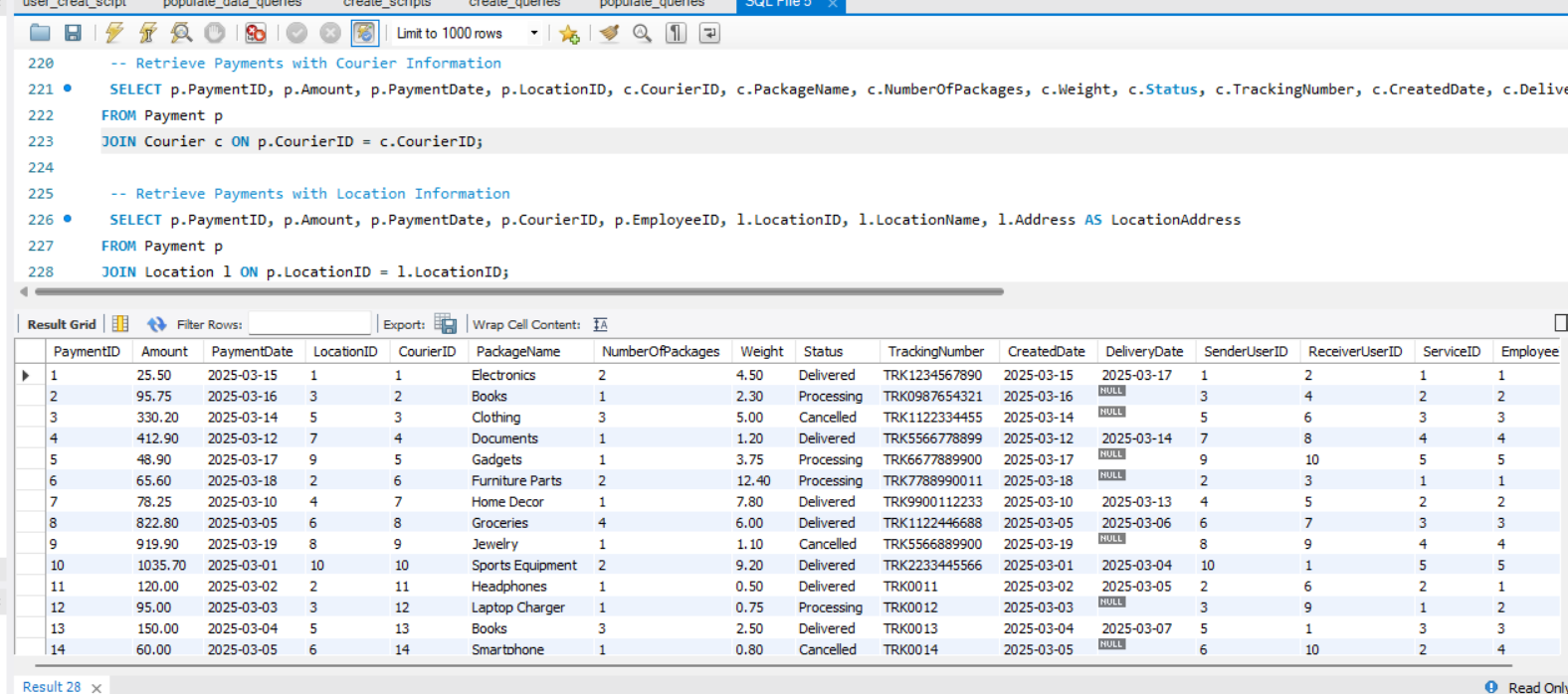
21. 

22.

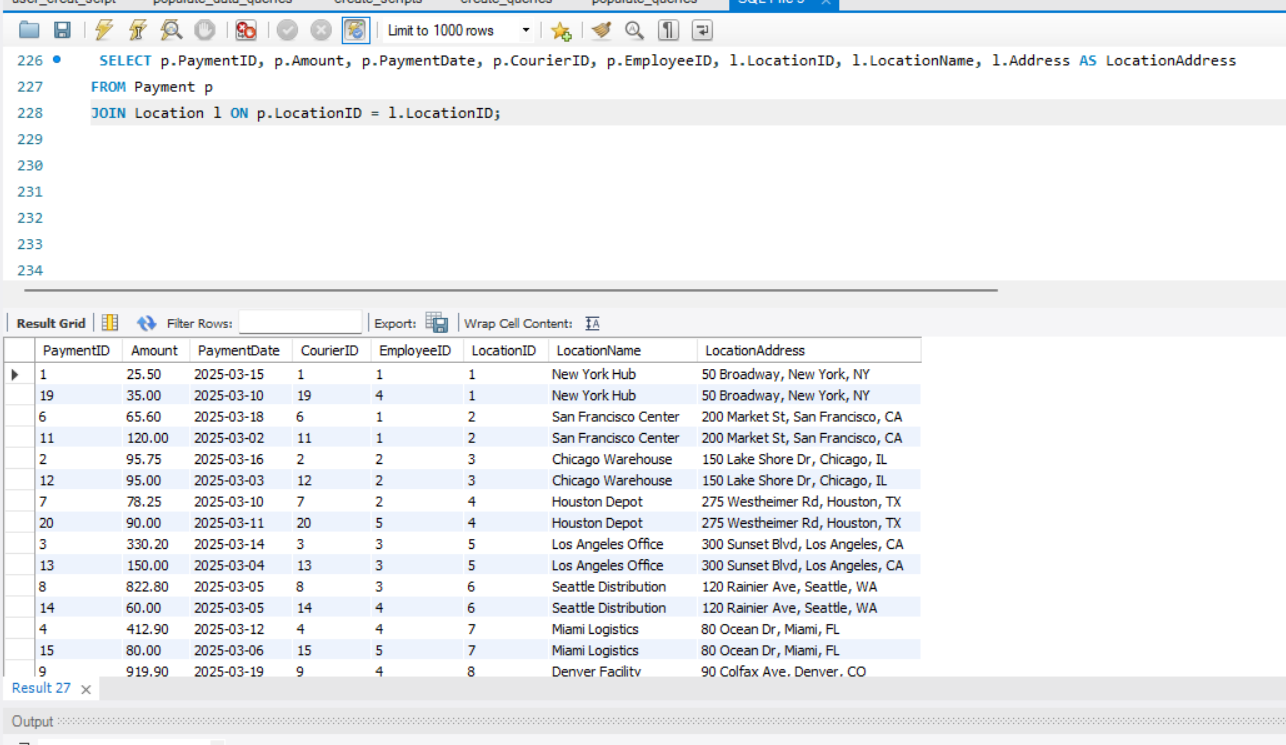


Task 4:

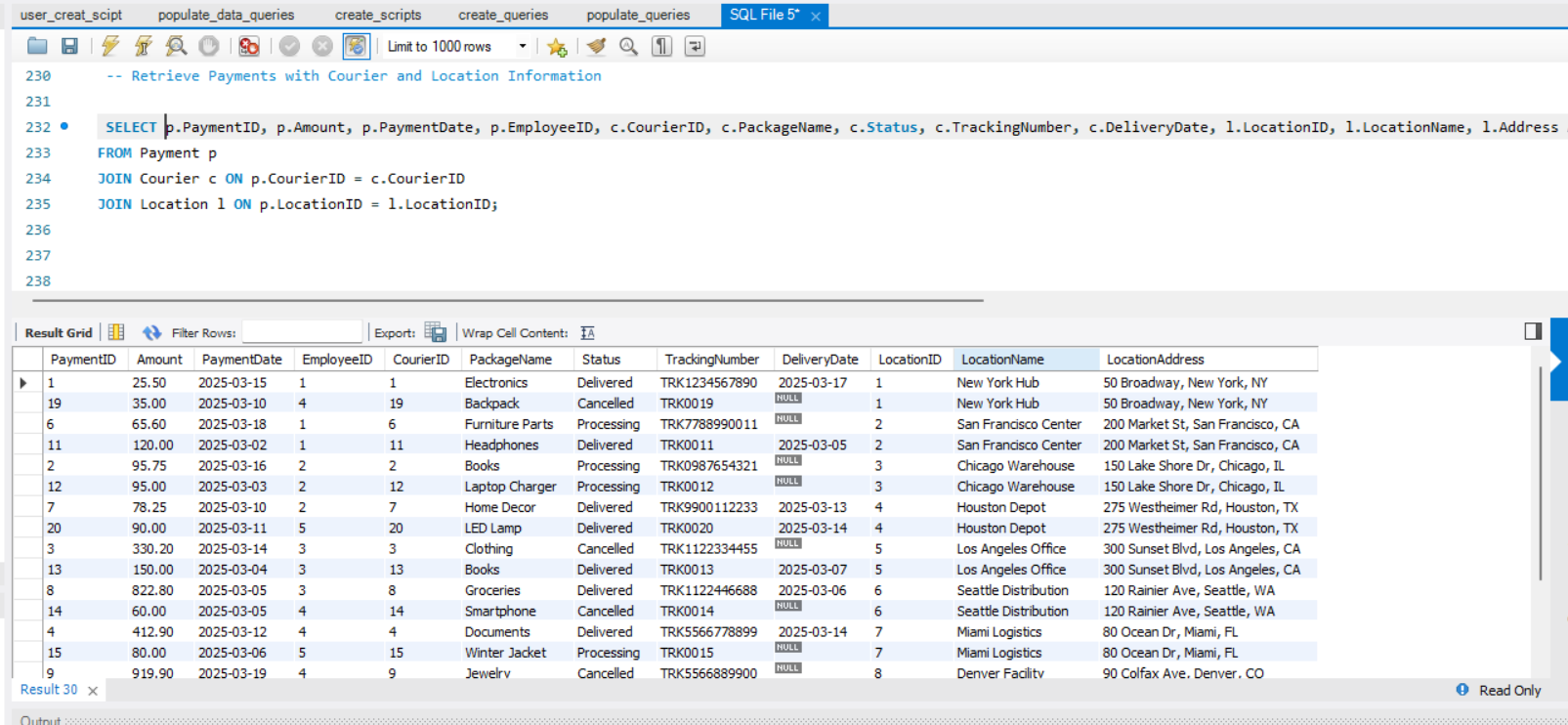
23.



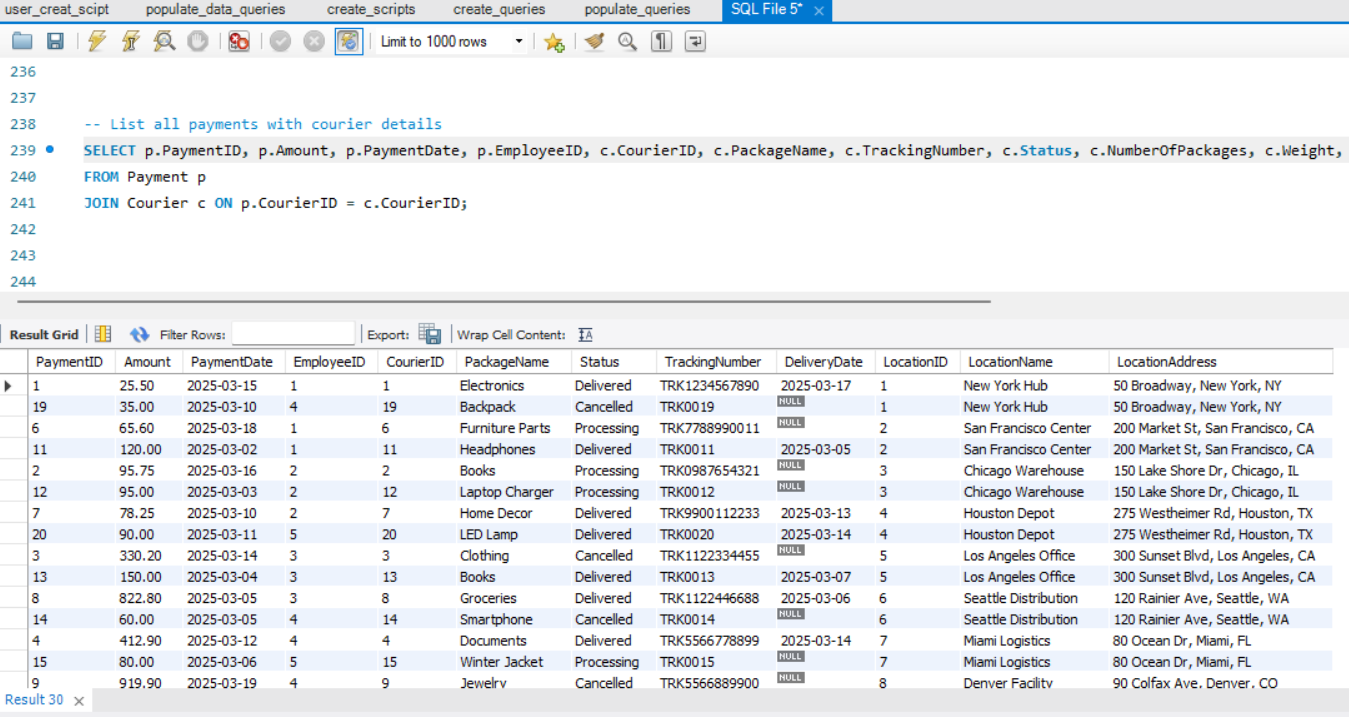
24.



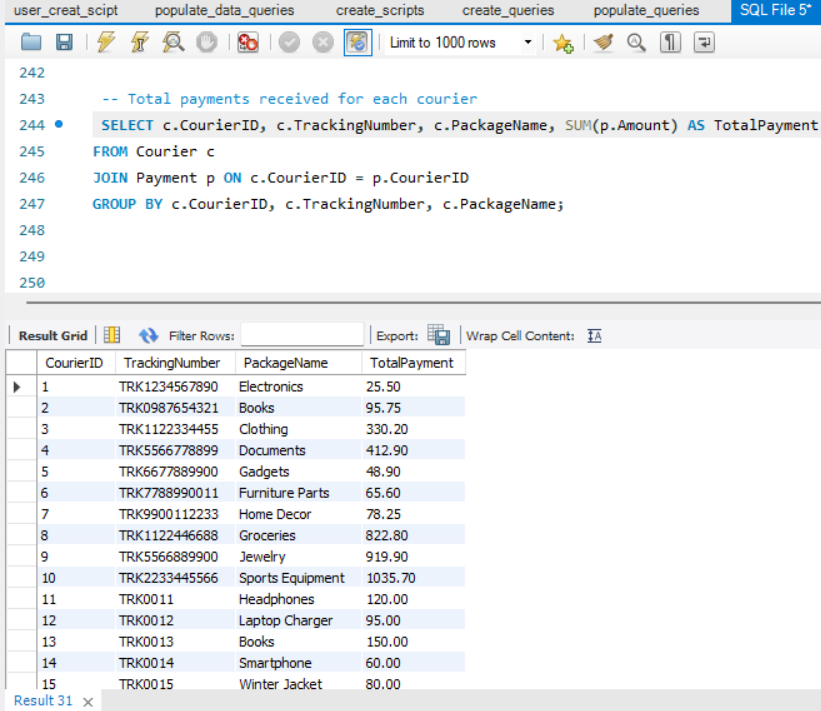
25.



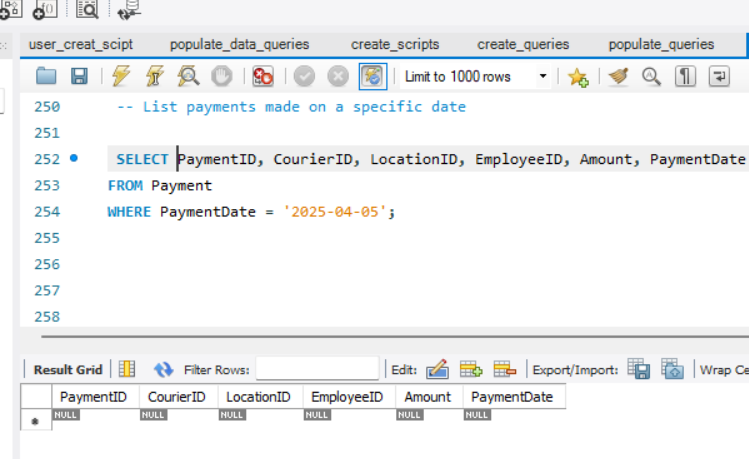
26. List all payments with courier details



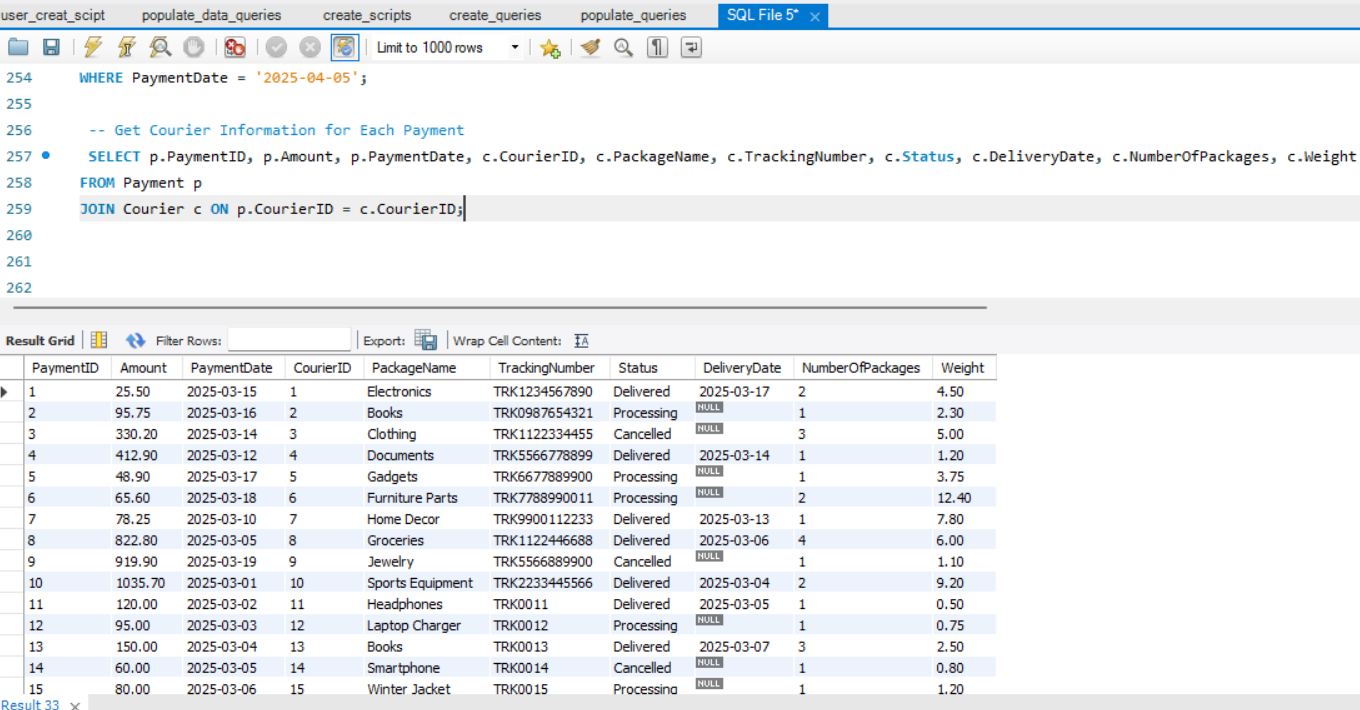
27.

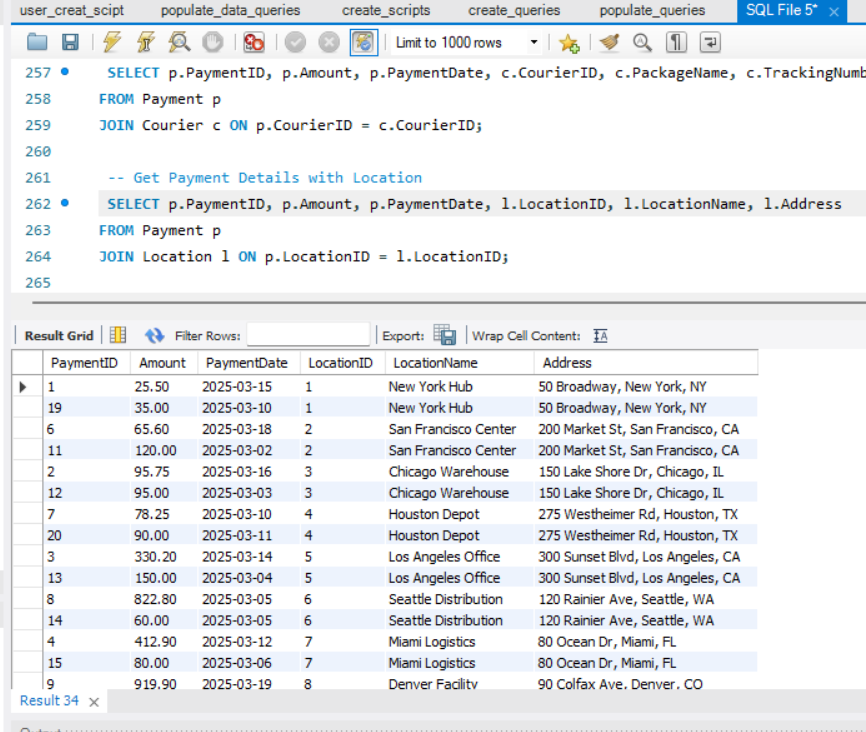


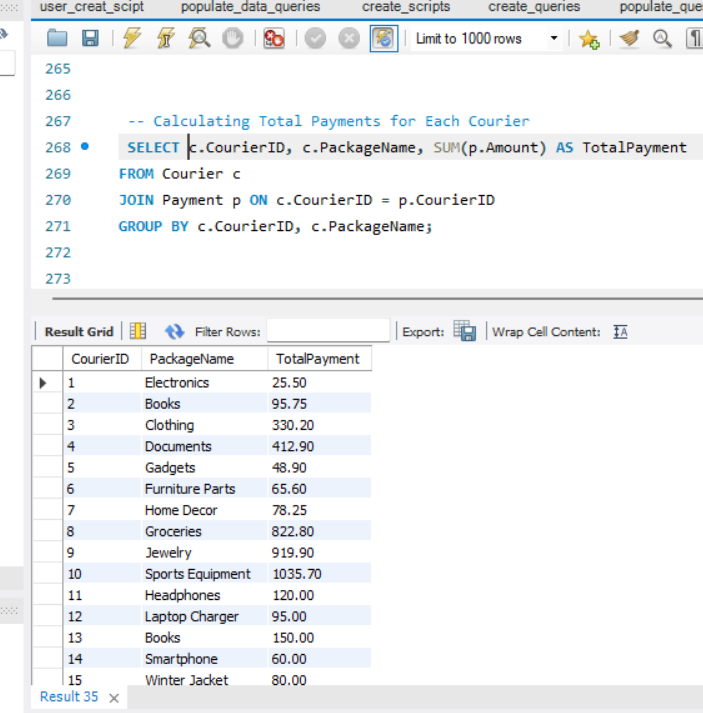
28.



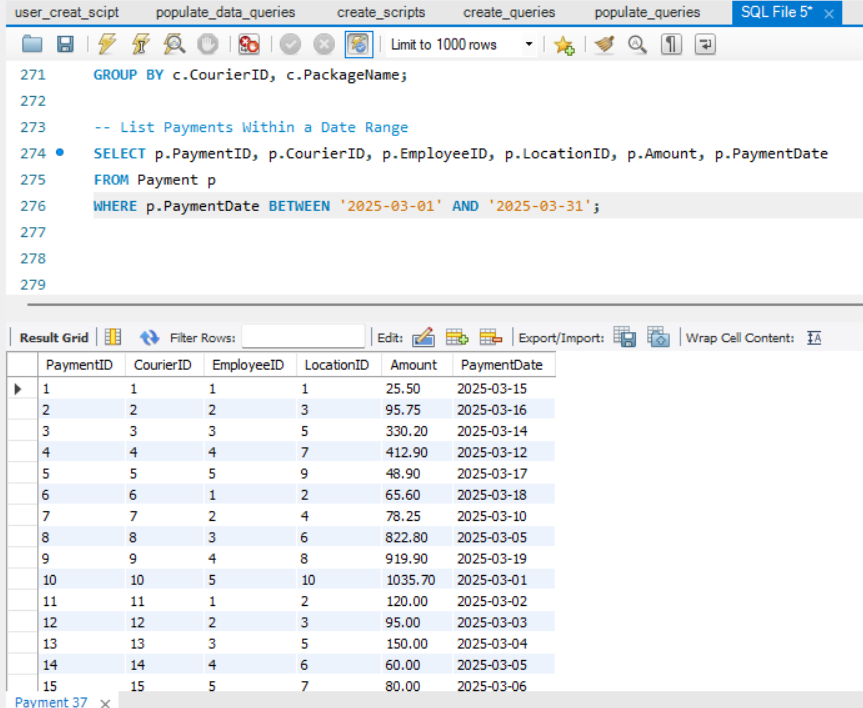
29.



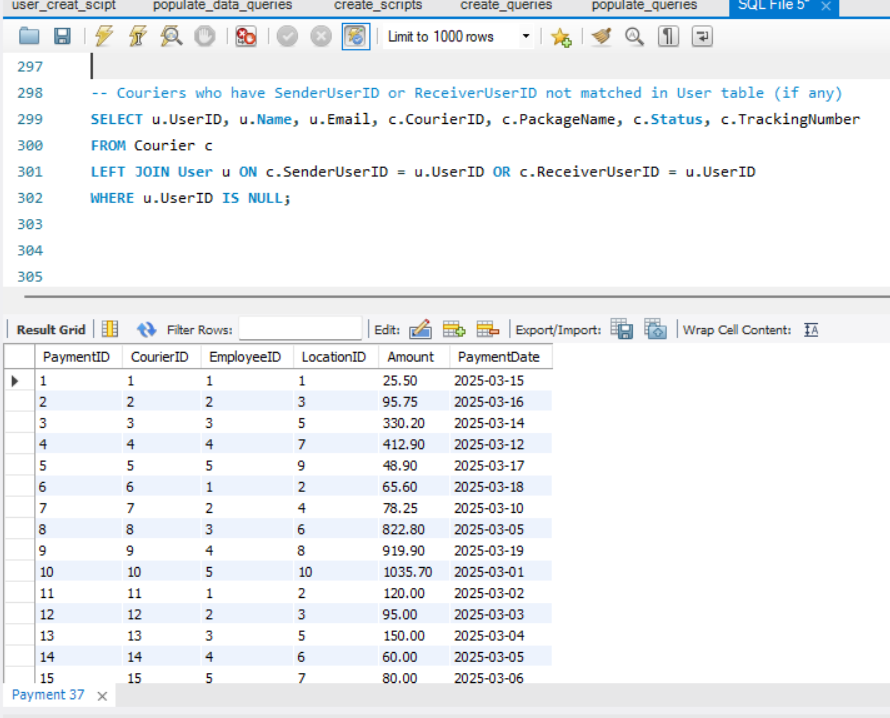
30.

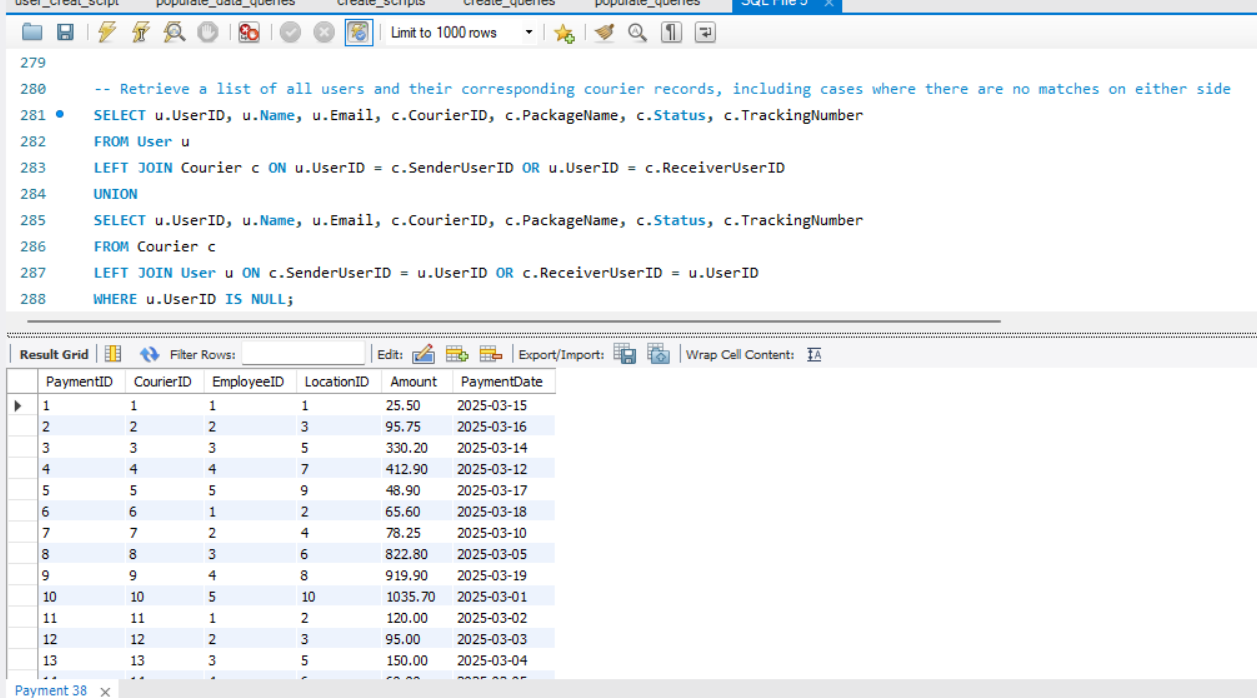
31.  


32.

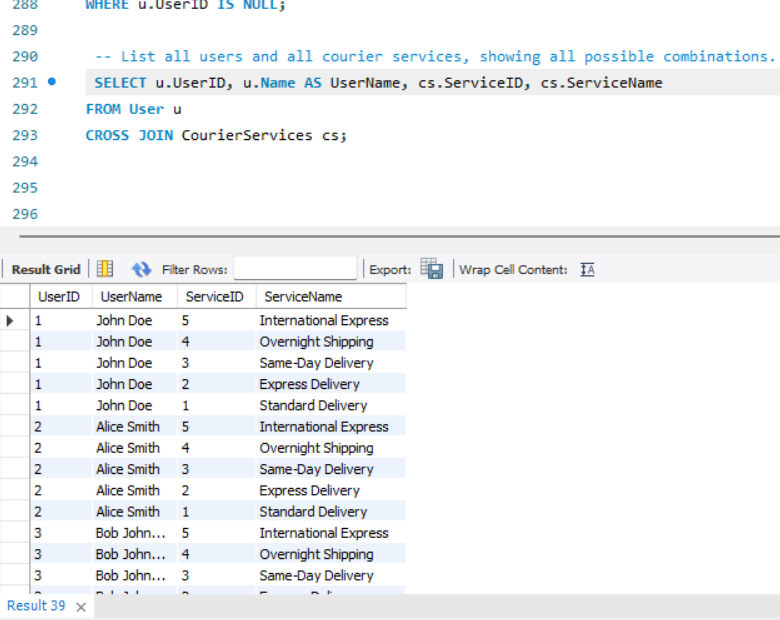


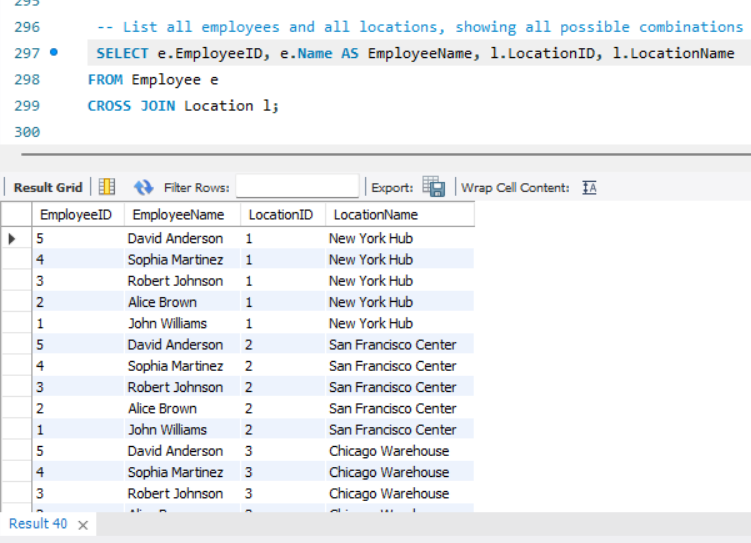
33.



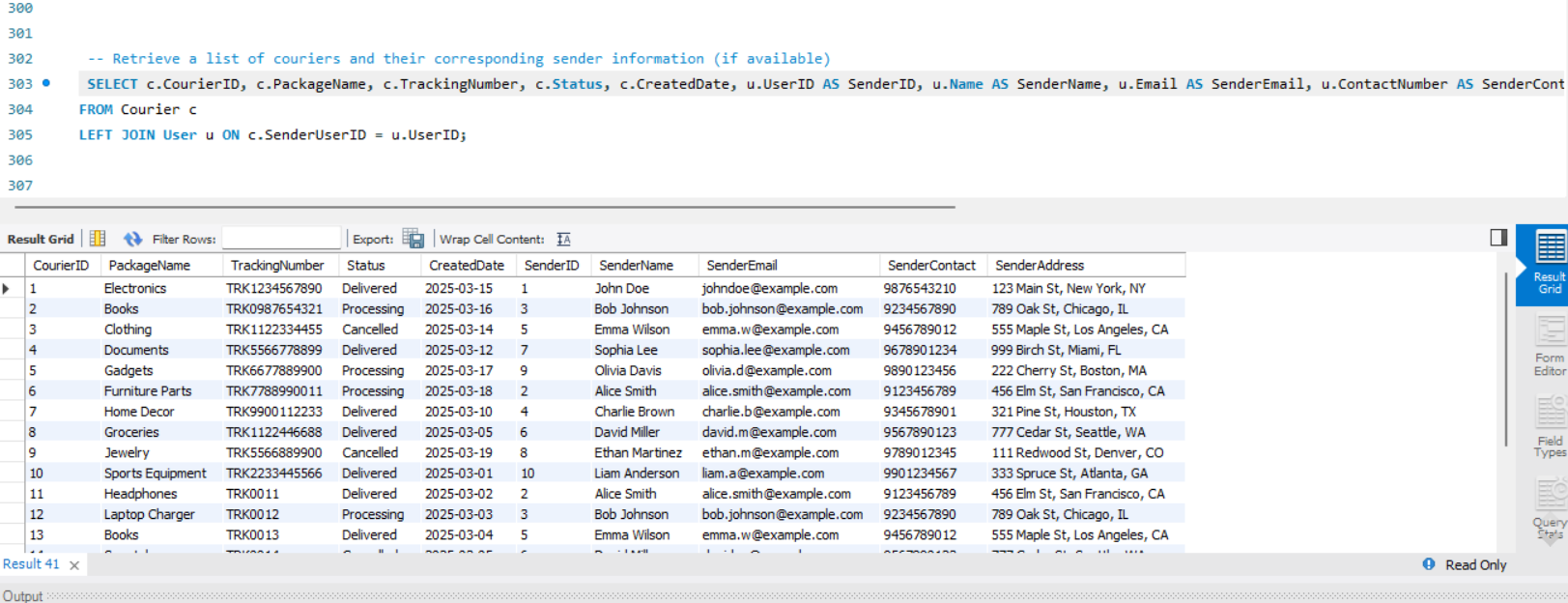
34.

36.

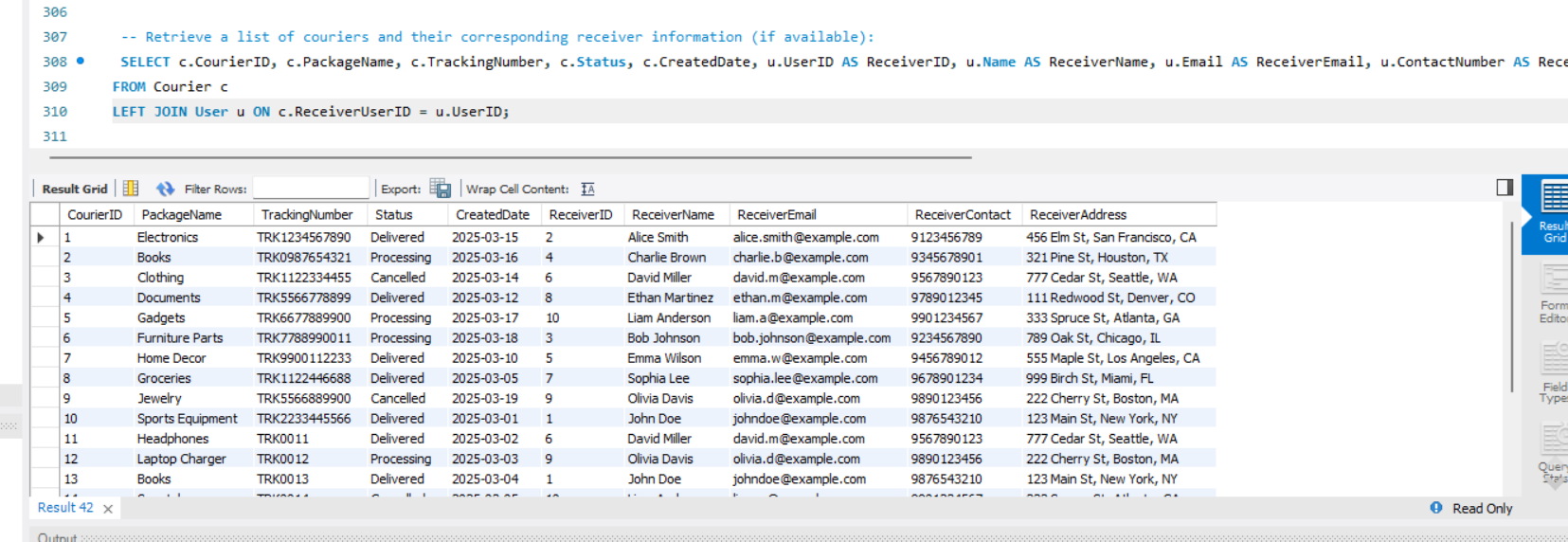


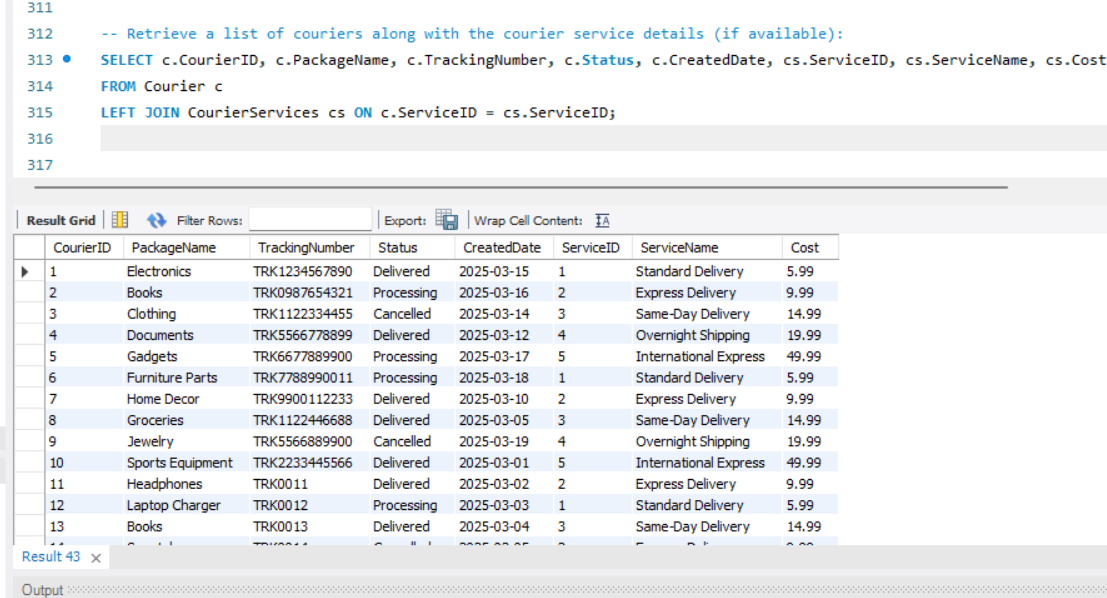
37.

38.

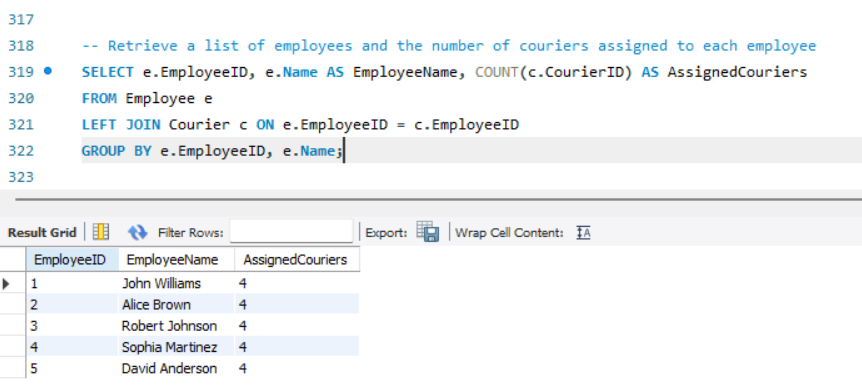


39.

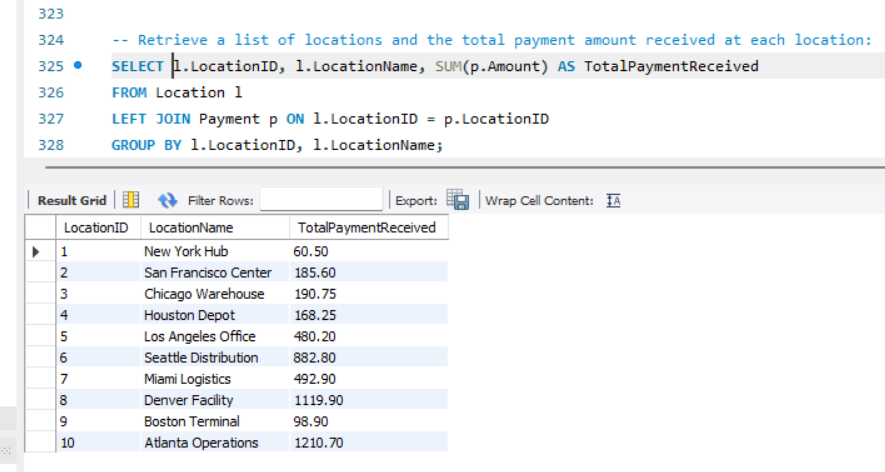


40. 

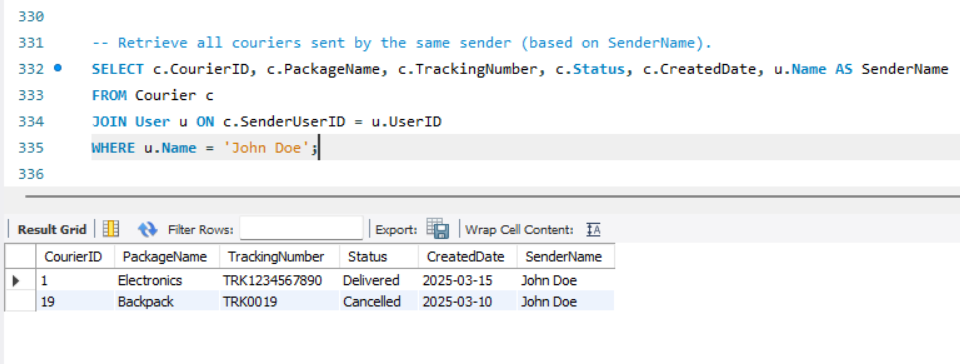
41.



42.

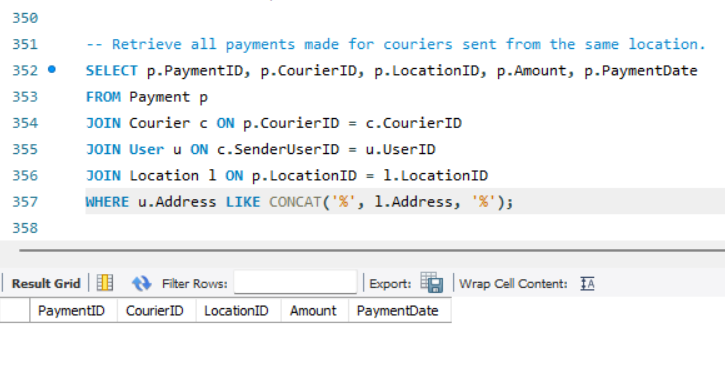


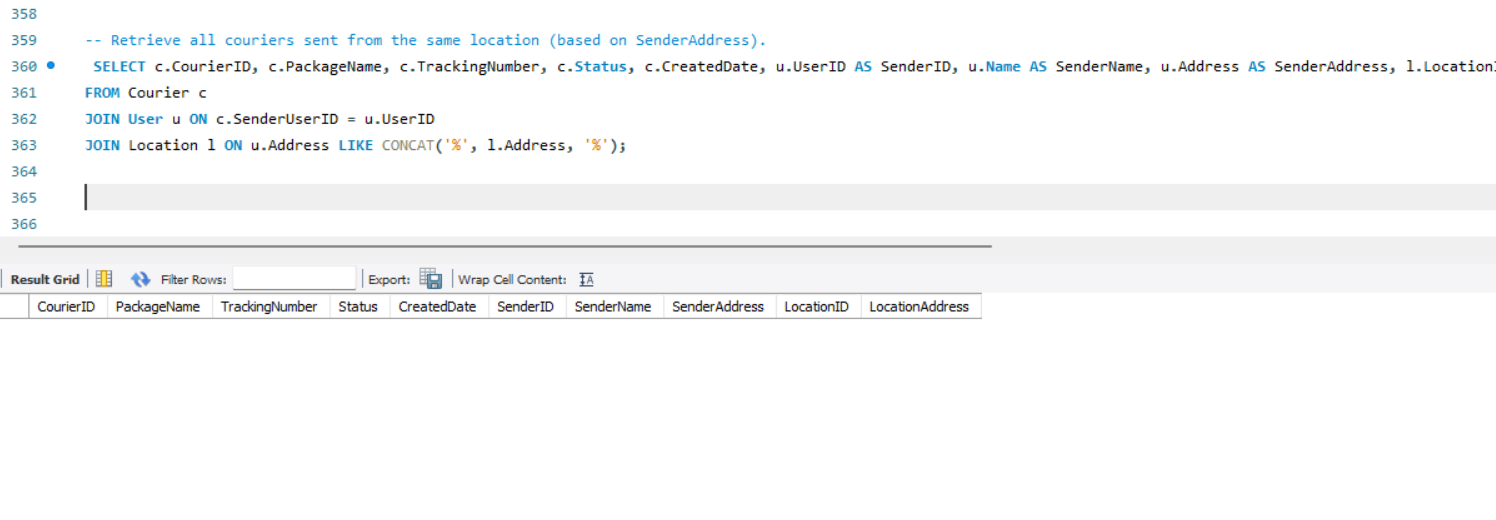
43.



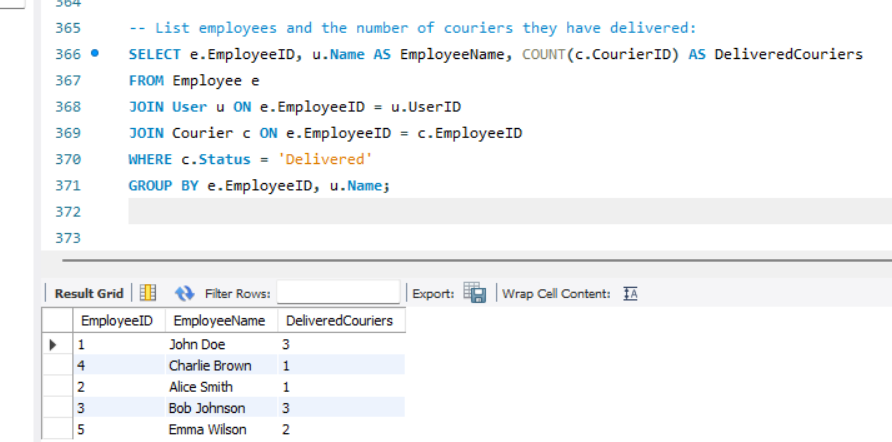
44. 

45.

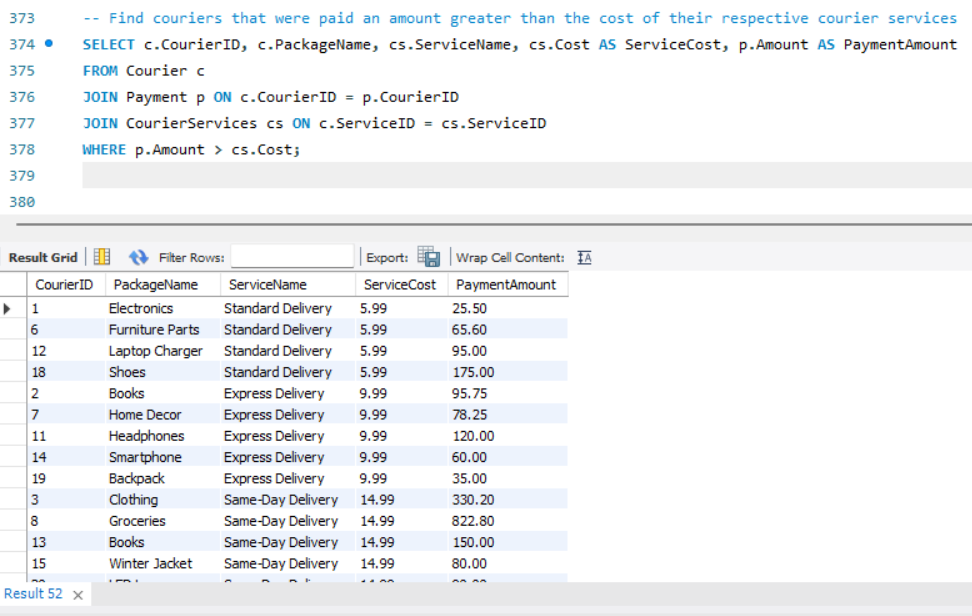
46.



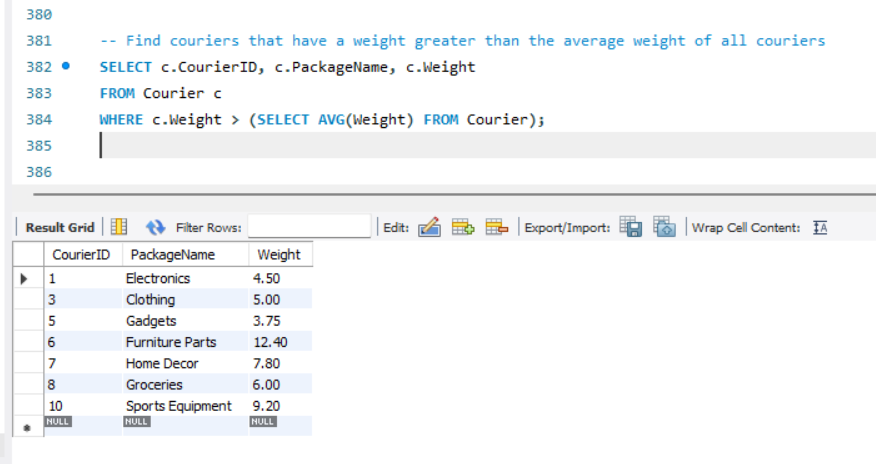
47.



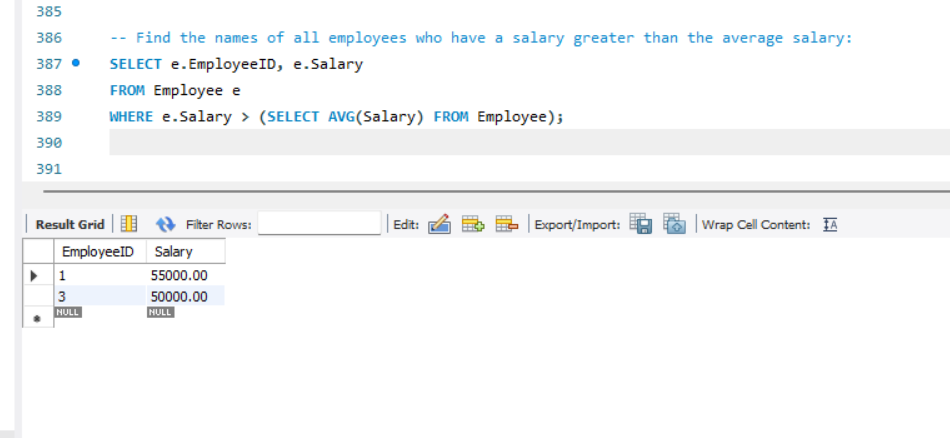
48.



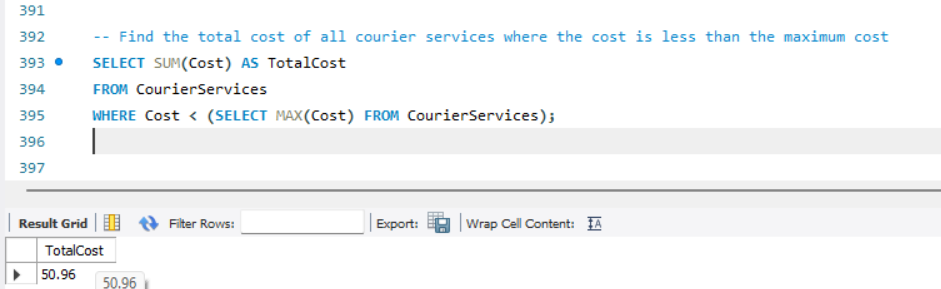
49.



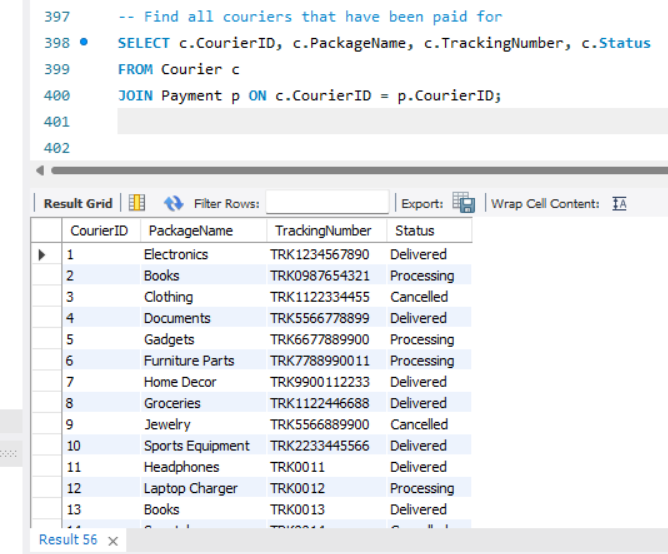
50.



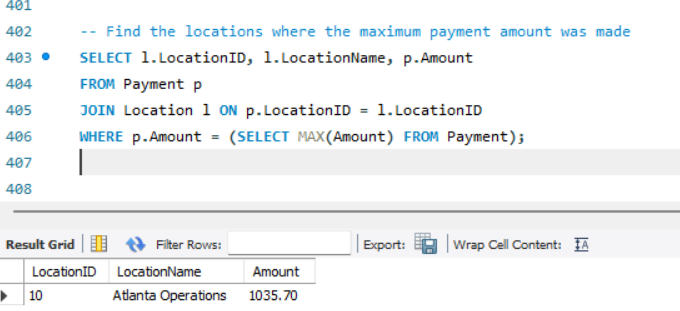
51.



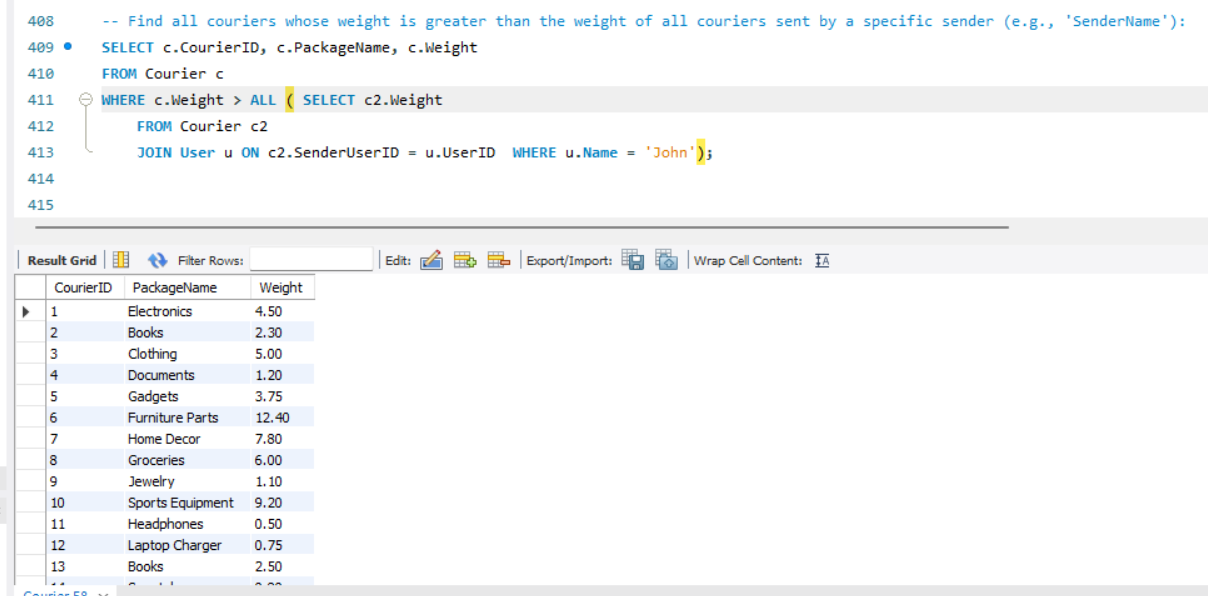
52.



53.



54.



TASKS (java implementation)  
  
package com.hexaware.assignment;

public class AddressFormatting {

public static String formatAddress(String address) {

String[] words = address.split(" ");

StringBuilder formattedAddress = new StringBuilder();

for (String word : words) {

formattedAddress.append(Character.*toUpperCase*(word.charAt(0)) + word.substring(1).toLowerCase()).append(" ");

}

return formattedAddress.toString().trim();

}

public static String formatZipCode(String zipCode) {

return zipCode.replaceAll("[^0-9]", ""); // Remove non-numeric characters

}

public static void main(String[] args) {

String address = "123 main st, new york, ny 10001";

System.***out***.println("Formatted Address: " + *formatAddress*(address));

System.***out***.println("Formatted Zip Code: " + *formatZipCode*("1000A1"));

}

}

package com.hexaware.assignment;

class Courier {

int courierId;

int senderId;

int receiverId;

String packageName;

String status;

public Courier(int courierId, int senderId, int receiverId, String packageName, String status) {

this.courierId = courierId;

this.senderId = senderId;

this.receiverId = receiverId;

this.packageName = packageName;

this.status = status;

}

}

package com.hexaware.assignment;

public class CourierAssignment {

public static void main(String[] args) {

String[] employees = {"John", "Alice", "Bob"};

int[] loadCapacity = {2, 1, 0}; // Number of packages they can still handle

boolean assigned = false;

for (int i = 0; i < employees.length; i++) {

if (loadCapacity[i] > 0) {

System.***out***.println("Courier assigned to: " + employees[i]);

loadCapacity[i]--;

assigned = true;

break;

}

}

if (!assigned) {

System.***out***.println("No employee available for courier assignment.");

}

}

}

package com.hexaware.assignment;

import java.util.\*;

public class CourierManager {

// Method to find the nearest available courier (first with status "Available")

public static Courier findNearestAvailableCourier(List<Courier> couriers) {

for (Courier c : couriers) {

if (c.status.equalsIgnoreCase("Available")) {

return c;

}

}

return null; // No available courier found

}

public static void main(String[] args) {

// Sample list of couriers

List<Courier> couriers = new ArrayList<>();

couriers.add(new Courier(101, 1, 5, "Laptop", "Busy"));

couriers.add(new Courier(102, 2, 6, "Phone", "Available"));

couriers.add(new Courier(103, 3, 7, "Documents", "Busy"));

couriers.add(new Courier(104, 4, 8, "Clothes", "Available"));

// Finding nearest available courier

Courier nearest = *findNearestAvailableCourier*(couriers);

if (nearest != null) {

System.***out***.println("Nearest available courier found:");

System.***out***.println("Courier ID: " + nearest.courierId);

System.***out***.println("Sender ID: " + nearest.senderId);

System.***out***.println("Receiver ID: " + nearest.receiverId);

System.***out***.println("Package: " + nearest.packageName);

System.***out***.println("Status: " + nearest.status);

} else {

System.***out***.println("No available courier found.");

}

}

}

package com.hexaware.assignment;

public class CustomerDataValidation {

public static boolean validateData(String data, String detail) {

if (detail.equalsIgnoreCase("name")) {

return data.matches("^[A-Z][a-z]\*$"); // Name should only contain letters and start with uppercase

} else if (detail.equalsIgnoreCase("address")) {

return !data.matches(".\*[!@#$%^&\*(),.?\":{}|<>].\*"); // Address should not contain special characters

} else if (detail.equalsIgnoreCase("phone")) {

return data.matches("\\d{3}-\\d{3}-\\d{4}"); // Phone number should match the format ###-###-####

}

return false;

}

public static void main(String[] args) {

System.***out***.println(*validateData*("John", "name"));

System.***out***.println(*validateData*("123 Main St.", "address"));

System.***out***.println(*validateData*("123-456-7890", "phone"));

}

}

package com.hexaware.assignment;

import java.util.\*;

public class CustomerOrders {

public static void main(String[] args) {

List<Courier> couriers = List.*of*(

new Courier(11, 2, 6, "Headphones", "Delivered"),

new Courier(12, 3, 9, "Laptop Charger", "Processing"),

new Courier(13, 5, 1, "Books", "Delivered"),

new Courier(14, 2, 10, "Smartphone", "Cancelled"),

new Courier(15, 2, 4, "Winter Jacket", "Processing")

);

int specificCustomerId = 2; // You can change this ID for other customers

System.***out***.println("Orders placed by Customer ID: " + specificCustomerId);

for (Courier c : couriers) {

if (c.senderId == specificCustomerId) {

System.***out***.println("Courier ID: " + c.courierId + " | Package: " + c.packageName + " | Status: " + c.status);

}

}

}

}

package com.hexaware.assignment;

public class OrderConfirmation {

public static String generateOrderEmail(String customerName, String orderNumber, String deliveryAddress, String deliveryDate) {

return "Hello " + customerName + ",\n\n" +

"Thank you for your order. Your order number is " + orderNumber + ".\n" +

"The delivery address is:\n" + deliveryAddress + "\n\n" +

"Your order is expected to be delivered on: " + deliveryDate + "\n\n" +

"Best regards,\n" +

"Courier System";

}

public static void main(String[] args) {

String email = *generateOrderEmail*("John Doe", "ORD12345", "123 Main St, New York, NY", "2025-04-20");

System.***out***.println(email);

}

}

package com.hexaware.assignment;

public class OrderStatusChecker {

public static void main(String[] args) {

String status = "Delivered"; // Test status

if (status.equalsIgnoreCase("Delivered")) {

System.***out***.println("The order has been delivered.");

} else if (status.equalsIgnoreCase("Processing")) {

System.***out***.println("The order is still processing.");

} else if (status.equalsIgnoreCase("Cancelled")) {

System.***out***.println("The order was cancelled.");

} else {

System.***out***.println("Unknown order status.");

}

}

}

package com.hexaware.assignment;

import java.util.Scanner;

public class ParcelTracking {

public static void main(String[] args) {

// Initialize the 2D array with parcel tracking numbers and their status

String[][] parcels = {

{"12345", "In Transit"},

{"67890", "Out for Delivery"},

{"11223", "Delivered"},

{"44556", "In Transit"},

{"78901", "Delivered"}

};

// User input for tracking number

Scanner scanner = new Scanner(System.***in***);

System.***out***.print("Enter Parcel Tracking Number: ");

String trackingNumber = scanner.nextLine();

// Simulate tracking

boolean found = false;

for (String[] parcel : parcels) {

if (parcel[0].equals(trackingNumber)) {

found = true;

System.***out***.println("Tracking Number: " + trackingNumber);

System.***out***.println("Parcel Status: " + parcel[1]);

break;

}

}

if (!found) {

System.***out***.println("Parcel not found.");

}

scanner.close();

}

}

package com.hexaware.assignment;

public class ParcelWeightCategory {

public static void main(String[] args) {

double weight = 12.5;

String category;

if (weight <= 5) {

category = "Light";

} else if (weight <= 15) {

category = "Medium";

} else {

category = "Heavy";

}

switch (category) {

case "Light":

System.***out***.println("The parcel is Light.");

break;

case "Medium":

System.***out***.println("The parcel is Medium.");

break;

case "Heavy":

System.***out***.println("The parcel is Heavy.");

break;

default:

System.***out***.println("Unknown category.");

}

}

}

package com.hexaware.assignment;

import java.security.SecureRandom;

public class PasswordGenerator {

private static final String ***UPPERCASE*** = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private static final String ***LOWERCASE*** = "abcdefghijklmnopqrstuvwxyz";

private static final String ***DIGITS*** = "0123456789";

private static final String ***SPECIAL*** = "!@#$%^&\*()-\_+=<>?";

public static String generatePassword(int length) {

String allChars = ***UPPERCASE*** + ***LOWERCASE*** + ***DIGITS*** + ***SPECIAL***;

SecureRandom random = new SecureRandom();

StringBuilder password = new StringBuilder();

for (int i = 0; i < length; i++) {

int index = random.nextInt(allChars.length());

password.append(allChars.charAt(index));

}

return password.toString();

}

public static void main(String[] args) {

String password = *generatePassword*(12);

System.***out***.println("Generated Password: " + password);

}

}

package com.hexaware.assignment;

public class ShippingCost {

public static double calculateShippingCost(double distance, double weight) {

double costPerMile = 0.5; // Cost per mile

double costPerKg = 1.2; // Cost per kg

return (distance \* costPerMile) + (weight \* costPerKg);

}

public static void main(String[] args) {

double distance = 100; // in miles

double weight = 10; // in kg

double cost = *calculateShippingCost*(distance, weight);

System.***out***.println("Shipping cost: $" + cost);

}

}

package com.hexaware.assignment;

import java.util.ArrayList;

import java.util.List;

public class SimilarAddresses {

public static List<String> findSimilarAddresses(List<String> addresses, String searchAddress) {

List<String> similarAddresses = new ArrayList<>();

for (String address : addresses) {

if (address.contains(searchAddress)) {

similarAddresses.add(address);

}

}

return similarAddresses;

}

public static void main(String[] args) {

List<String> addresses = new ArrayList<>();

addresses.add("123 Main St, New York, NY");

addresses.add("124 Main St, New York, NY");

addresses.add("456 Oak St, New York, NY");

List<String> similar = findSimilarAddresses(addresses, "Main St");

for (String address : similar) {

System.out.println(address);

}

}

}

package com.hexaware.assignment;

import java.util.Scanner;

public class UserAuthentication {

public static void main(String[] args) {

String storedUsername = "employee123";

String storedPassword = "password123";

Scanner scanner = new Scanner(System.***in***);

System.***out***.print("Enter username: ");

String inputUsername = scanner.nextLine();

System.***out***.print("Enter password: ");

String inputPassword = scanner.nextLine();

if (inputUsername.equals(storedUsername) && inputPassword.equals(storedPassword)) {

System.***out***.println("Login successful!");

} else {

System.***out***.println("Invalid username or password.");

}

scanner.close();

}

}

DATABASE INERACTION IMPLEMENTATION

package dao;

import entity.Courier;

import entity.Employee;

import exceptions.InvalidEmployeeIdException;

import interfaces.ICourierAdminService;

public class CourierAdminServiceCollectionImpl extends CourierUserServiceCollectionImpl implements ICourierAdminService {

@Override

public void addCourierStaff(Employee emp) {

companyObj.addEmployee(emp);

}

@Override

public boolean assignEmployeeToOrder(String trackingNumber, int employeeId) {

Courier courier = companyObj.findCourierByTrackingNumber(trackingNumber);

if (courier == null) {

return false;

}

if (companyObj.findEmployeeById(employeeId) == null) {

throw new InvalidEmployeeIdException("Employee ID not found: " + employeeId);

}

courier.setEmployeeID(employeeId);

courier.setStatus("inTransit");

return true;

}

@Override

public boolean deleteCourier(String trackingNumber) {

return companyObj.removeCourierByTrackingNumber(trackingNumber);

}

}

package dao;

import entity.Courier;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class CourierServiceDb {

private Connection connection;

// ✅ Constructor to initialize DB connection

public CourierServiceDb(Connection connection) {

this.connection = connection;

}

// ✅ Insert a new courier

public void addCourier(Courier courier) {

String sql = "INSERT INTO Courier (senderUserID, receiverUserID, packageName, numberOfPackages, weight, serviceID, trackingNumber, status, createdDate) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setInt(1, courier.getSenderUserID());

stmt.setInt(2, courier.getReceiverUserID());

stmt.setString(3, courier.getPackageName());

stmt.setInt(4, courier.getNumberOfPackages());

stmt.setDouble(5, courier.getWeight());

stmt.setInt(6, courier.getServiceID());

stmt.setString(7, courier.getTrackingNumber());

stmt.setString(8, courier.getStatus());

stmt.setDate(9, new java.sql.Date(courier.getCreatedDate().getTime()));

stmt.executeUpdate();

System.out.println("Courier added successfully.");

} catch (SQLException e) {

System.err.println("Error adding courier: " + e.getMessage());

}

}

// ✅ Assign a courier to an employee

public void assignCourier(int courierId, int employeeId) {

String sql = "UPDATE Courier SET employeeID = ? WHERE courierID = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setInt(1, employeeId);

stmt.setInt(2, courierId);

int rows = stmt.executeUpdate();

if (rows > 0) {

System.out.println("Courier assigned successfully.");

} else {

System.out.println("Courier ID not found.");

}

} catch (SQLException e) {

System.err.println("Error assigning courier: " + e.getMessage());

}

}

// ✅ Update courier status using tracking number

public void updateCourierStatus(String trackingNumber, String newStatus) {

String sql = "UPDATE Courier SET status = ? WHERE trackingNumber = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, newStatus);

stmt.setString(2, trackingNumber);

int rows = stmt.executeUpdate();

if (rows > 0) {

System.out.println("Courier status updated.");

} else {

System.out.println("Tracking number not found.");

}

} catch (SQLException e) {

System.err.println("Error updating status: " + e.getMessage());

}

}

// ✅ Overloaded method: Update courier status using courierID

public void updateCourierStatus(int courierId, String newStatus) {

String sql = "UPDATE Courier SET status = ? WHERE courierID = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, newStatus);

stmt.setInt(2, courierId);

int rows = stmt.executeUpdate();

if (rows > 0) {

System.out.println("Courier status updated.");

} else {

System.out.println("Courier ID not found.");

}

} catch (SQLException e) {

System.err.println("Error updating status: " + e.getMessage());

}

}

// ✅ Fetch delivery history by tracking number

public void getDeliveryHistory(String trackingNumber) {

String sql = "SELECT \* FROM Courier WHERE trackingNumber = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, trackingNumber);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

System.out.println("Courier ID: " + rs.getInt("courierID"));

System.out.println("Status: " + rs.getString("status"));

System.out.println("Created Date: " + rs.getDate("createdDate"));

System.out.println("Delivery Date: " + rs.getDate("deliveryDate"));

} else {

System.out.println("No courier found with this tracking number.");

}

} catch (SQLException e) {

System.err.println("Error retrieving history: " + e.getMessage());

}

}

// ✅ Generate shipment status report

public void generateShipmentStatusReport() {

String sql = "SELECT status, COUNT(\*) AS count FROM Courier GROUP BY status";

try (Statement stmt = connection.createStatement()) {

ResultSet rs = stmt.executeQuery(sql);

System.out.println("\nShipment Status Report:");

while (rs.next()) {

System.out.println(rs.getString("status") + ": " + rs.getInt("count"));

}

} catch (SQLException e) {

System.err.println("Error generating report: " + e.getMessage());

}

}

// ✅ Generate revenue report (assumes delivery charge exists)

public void generateRevenueReport() {

String sql = "SELECT SUM(cs.Cost \* c.NumberOfPackages) AS totalRevenue " +

"FROM Courier c " +

"JOIN CourierServices cs ON c.ServiceID = cs.ServiceID";

try (Statement stmt = connection.createStatement()) {

ResultSet rs = stmt.executeQuery(sql);

if (rs.next()) {

System.out.printf("Total Revenue: ₹%.2f\n", rs.getDouble("totalRevenue"));

} else {

System.out.println("No revenue data found.");

}

} catch (SQLException e) {

System.err.println("Error generating revenue report: " + e.getMessage());

}

}

}

package dao;

import java.util.List;

import entity.Courier;

import entity.CourierCompanyCollection;

import exceptions.TrackingNumberNotFoundException;

import interfaces.ICourierUserService;

public class CourierUserServiceCollectionImpl implements ICourierUserService {

protected CourierCompanyCollection companyObj = new CourierCompanyCollection();

@Override

public String placeOrder(Courier courier) {

companyObj.addCourier(courier);

return courier.getTrackingNumber();

}

@Override

public Courier getOrderDetails(String trackingNumber) {

return companyObj.findCourierByTrackingNumber(trackingNumber);

}

@Override

public List<Courier> getAssignedOrder(int employeeId) {

return companyObj.getCouriersByEmployeeId(employeeId);

}

@Override

public String getOrderStatus(String trackingNumber) throws TrackingNumberNotFoundException {

Courier c = companyObj.findCourierByTrackingNumber(trackingNumber);

if (c == null) throw new TrackingNumberNotFoundException("Tracking number not found");

return c.getStatus();

}

@Override

public boolean cancelOrder(String trackingNumber) throws TrackingNumberNotFoundException {

Courier c = companyObj.findCourierByTrackingNumber(trackingNumber);

if (c == null) throw new TrackingNumberNotFoundException("Tracking number not found");

c.setStatus("cancelled");

return true;

}

}

package entity;

import java.util.Date;

public class Courier {

private static int *trackingNumberSeed* = 1000; // Static seed for tracking number

private int courierID;

private int senderUserID;

private int receiverUserID;

private String packageName;

private int numberOfPackages;

private double weight;

private String status; // Allowed: "Processing", "Delivered", "Cancelled"

private String trackingNumber;

private Date createdDate;

private Date deliveryDate;

private int serviceID;

private Integer employeeID; // Nullable

// Default constructor - auto generates tracking number and sets default status

public Courier() {

this.trackingNumber = "TRK" + (++*trackingNumberSeed*);

this.createdDate = new Date(); // Assign current date

this.status = "Processing"; // ✅ Default status matches ENUM in DB

}

// Constructor with fields except tracking number (auto-generated)

public Courier(int senderUserID, int receiverUserID, String packageName,

int numberOfPackages, double weight, int serviceID) {

this(); // calls the default constructor

this.senderUserID = senderUserID;

this.receiverUserID = receiverUserID;

this.packageName = packageName;

this.numberOfPackages = numberOfPackages;

this.weight = weight;

this.serviceID = serviceID;

}

// Getters and Setters

public int getCourierID() {

return courierID;

}

public void setCourierID(int courierID) {

this.courierID = courierID;

}

public int getSenderUserID() {

return senderUserID;

}

public void setSenderUserID(int senderUserID) {

this.senderUserID = senderUserID;

}

public int getReceiverUserID() {

return receiverUserID;

}

public void setReceiverUserID(int receiverUserID) {

this.receiverUserID = receiverUserID;

}

public String getPackageName() {

return packageName;

}

public void setPackageName(String packageName) {

this.packageName = packageName;

}

public int getNumberOfPackages() {

return numberOfPackages;

}

public void setNumberOfPackages(int numberOfPackages) {

this.numberOfPackages = numberOfPackages;

}

public double getWeight() {

return weight;

}

public void setWeight(double weight) {

this.weight = weight;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public String getTrackingNumber() {

return trackingNumber;

}

public void setTrackingNumber(String trackingNumber) {

this.trackingNumber = trackingNumber;

}

public Date getCreatedDate() {

return createdDate;

}

public void setCreatedDate(Date createdDate) {

this.createdDate = createdDate;

}

public Date getDeliveryDate() {

return deliveryDate;

}

public void setDeliveryDate(Date deliveryDate) {

this.deliveryDate = deliveryDate;

}

public int getServiceID() {

return serviceID;

}

public void setServiceID(int serviceID) {

this.serviceID = serviceID;

}

public Integer getEmployeeID() {

return employeeID;

}

public void setEmployeeID(Integer employeeID) {

this.employeeID = employeeID;

}

public static int getTrackingNumberSeed() {

return *trackingNumberSeed*;

}

public static void setTrackingNumberSeed(int trackingNumberSeed) {

Courier.*trackingNumberSeed* = trackingNumberSeed;

}

}

package entity;

import java.util.ArrayList;

import java.util.List;

public class CourierCompanyCollection {

private List<Courier> couriers = new ArrayList<>();

private List<Employee> employees = new ArrayList<>();

// Getter for courier list

public List<Courier> getCouriers() {

return couriers;

}

// Add a courier to the list

public void addCourier(Courier courier) {

couriers.add(courier);

}

// Remove courier by tracking number

public boolean removeCourierByTrackingNumber(String trackingNumber) {

return couriers.removeIf(c -> c.getTrackingNumber().equals(trackingNumber));

}

// ✅ Correct method name: findCourierByTrackingNumber

public Courier findCourierByTrackingNumber(String trackingNumber) {

for (Courier c : couriers) {

if (c.getTrackingNumber().equals(trackingNumber)) {

return c;

}

}

return null;

}

// Get couriers assigned to a particular employee

public List<Courier> getCouriersByEmployeeId(int employeeId) {

List<Courier> assigned = new ArrayList<>();

for (Courier c : couriers) {

if (c.getEmployeeID() != null && c.getEmployeeID() == employeeId) {

assigned.add(c);

}

}

return assigned;

}

// Getter for employee list

public List<Employee> getEmployees() {

return employees;

}

// Add employee and return their ID

public int addEmployee(Employee emp) {

employees.add(emp);

return emp.getEmployeeID(); // assumes employeeID is set externally or auto-generated

}

// Find employee by ID

public Employee findEmployeeById(int id) {

for (Employee emp : employees) {

if (emp.getEmployeeID() == id) {

return emp;

}

}

return null;

}

}

package entity;

public class CourierServices {

private int serviceID;

private String serviceName;

private double cost;

// Default Constructor

public CourierServices() {}

// Parameterized Constructor

public CourierServices(int serviceID, String serviceName, double cost) {

this.serviceID = serviceID;

this.serviceName = serviceName;

this.cost = cost;

}

// Getters and Setters

public int getServiceID() {

return serviceID;

}

public void setServiceID(int serviceID) {

this.serviceID = serviceID;

}

public String getServiceName() {

return serviceName;

}

public void setServiceName(String serviceName) {

this.serviceName = serviceName;

}

public double getCost() {

return cost;

}

public void setCost(double cost) {

this.cost = cost;

}

*@Override*

public String toString() {

return "CourierServices [serviceID=" + serviceID + ", serviceName=" + serviceName + ", cost=" + cost + "]";

}

}

package entity;

public class Employee {

private int employeeID;

private int userID;

private String role;

private double salary;

private String name;

// Default Constructor

public Employee() {}

// Parameterized Constructor

public Employee(int employeeID, int userID, String role, double salary, String name) {

this.employeeID = employeeID;

this.userID = userID;

this.role = role;

this.salary = salary;

this.name = name;

}

// Getters and Setters

public int getEmployeeID() {

return employeeID;

}

public void setEmployeeID(int employeeID) {

this.employeeID = employeeID;

}

public int getUserID() {

return userID;

}

public void setUserID(int userID) {

this.userID = userID;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

*@Override*

public String toString() {

return "Employee [employeeID=" + employeeID + ", userID=" + userID + ", role=" + role + ", salary=" + salary

+ ", name=" + name + "]";

}

}

package entity;

public class Location {

private int locationID;

private String locationName;

private String address;

// Default Constructor

public Location() {}

// Parameterized Constructor

public Location(int locationID, String locationName, String address) {

this.locationID = locationID;

this.locationName = locationName;

this.address = address;

}

// Getters and Setters

public int getLocationID() {

return locationID;

}

public void setLocationID(int locationID) {

this.locationID = locationID;

}

public String getLocationName() {

return locationName;

}

public void setLocationName(String locationName) {

this.locationName = locationName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

*@Override*

public String toString() {

return "Location [locationID=" + locationID + ", locationName=" + locationName + ", address=" + address + "]";

}

}

package entity;

import java.util.Date;

public class Payment {

private int paymentID;

private int courierID;

private int locationID;

private Integer employeeID; // Nullable

private double amount;

private Date paymentDate;

// Default Constructor

public Payment() {}

// Parameterized Constructor

public Payment(int paymentID, int courierID, int locationID, Integer employeeID, double amount, Date paymentDate) {

this.paymentID = paymentID;

this.courierID = courierID;

this.locationID = locationID;

this.employeeID = employeeID;

this.amount = amount;

this.paymentDate = paymentDate;

}

// Getters and Setters

public int getPaymentID() {

return paymentID;

}

public void setPaymentID(int paymentID) {

this.paymentID = paymentID;

}

public int getCourierID() {

return courierID;

}

public void setCourierID(int courierID) {

this.courierID = courierID;

}

public int getLocationID() {

return locationID;

}

public void setLocationID(int locationID) {

this.locationID = locationID;

}

public Integer getEmployeeID() {

return employeeID;

}

public void setEmployeeID(Integer employeeID) {

this.employeeID = employeeID;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public Date getPaymentDate() {

return paymentDate;

}

public void setPaymentDate(Date paymentDate) {

this.paymentDate = paymentDate;

}

*@Override*

public String toString() {

return "Payment [paymentID=" + paymentID + ", courierID=" + courierID + ", locationID=" + locationID

+ ", employeeID=" + employeeID + ", amount=" + amount + ", paymentDate=" + paymentDate + "]";

}

}

package entity;

public class User {

private int userID;

private String userName;

private String email;

private String password;

private String contactNumber;

private String address;

// Default Constructor

public User() {}

// Parameterized Constructor

public User(int userID, String userName, String email, String password, String contactNumber, String address) {

this.userID = userID;

this.userName = userName;

this.email = email;

this.password = password;

this.contactNumber = contactNumber;

this.address = address;

}

// Getters and Setters

public int getUserID() {

return userID;

}

public void setUserID(int userID) {

this.userID = userID;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getContactNumber() {

return contactNumber;

}

public void setContactNumber(String contactNumber) {

this.contactNumber = contactNumber;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

*@Override*

public String toString() {

return "User [userID=" + userID + ", userName=" + userName + ", email=" + email + ", password=" + password

+ ", contactNumber=" + contactNumber + ", address=" + address + "]";

}

}

package entity;

public class User {

private int userID;

private String userName;

private String email;

private String password;

private String contactNumber;

private String address;

// Default Constructor

public User() {}

// Parameterized Constructor

public User(int userID, String userName, String email, String password, String contactNumber, String address) {

this.userID = userID;

this.userName = userName;

this.email = email;

this.password = password;

this.contactNumber = contactNumber;

this.address = address;

}

// Getters and Setters

public int getUserID() {

return userID;

}

public void setUserID(int userID) {

this.userID = userID;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getContactNumber() {

return contactNumber;

}

public void setContactNumber(String contactNumber) {

this.contactNumber = contactNumber;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

*@Override*

public String toString() {

return "User [userID=" + userID + ", userName=" + userName + ", email=" + email + ", password=" + password

+ ", contactNumber=" + contactNumber + ", address=" + address + "]";

}

}

package exceptions;

public class TrackingNumberNotFoundException extends Exception {

public TrackingNumberNotFoundException(String message) {

super(message);

}

}

package interfaces;

import entity.Employee;

public interface ICourierAdminService extends ICourierUserService {

void addCourierStaff(Employee emp);

boolean assignEmployeeToOrder(String trackingNumber, int employeeId);

boolean deleteCourier(String trackingNumber);

}

package interfaces;

import java.util.List;

import entity.Courier;

import exceptions.TrackingNumberNotFoundException;

public interface ICourierUserService {

String placeOrder(Courier courier);

Courier getOrderDetails(String trackingNumber);

List<Courier> getAssignedOrder(int employeeId);

String getOrderStatus(String trackingNumber) throws TrackingNumberNotFoundException;

boolean cancelOrder(String trackingNumber) throws TrackingNumberNotFoundException;

}

package main;

import dao.CourierServiceDb;

import entity.Courier;

import java.sql.Connection;

import java.util.Scanner;

import util.DBConnUtil;

import util.DBPropertyUtil;

public class MainApp {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String connectionString = DBPropertyUtil.getConnectionString("db.properties");

Connection connection = DBConnUtil.getConnection(connectionString);

if (connection == null) {

System.err.println("Exiting: Unable to establish DB connection.");

return;

}

CourierServiceDb service = new CourierServiceDb(connection);

boolean exit = false;

while (!exit) {

System.out.println("\n========== Courier Management Menu ==========");

System.out.println("1. Add New Courier");

System.out.println("2. Assign Courier to Employee");

System.out.println("3. Update Courier Status");

System.out.println("4. Get Delivery History");

System.out.println("5. Generate Shipment Status Report");

System.out.println("6. Generate Revenue Report");

System.out.println("7. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Clear newline

switch (choice) {

case 1:

System.out.print("Sender User ID: ");

int senderId = scanner.nextInt();

System.out.print("Receiver User ID: ");

int receiverId = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Package Name: ");

String packageName = scanner.nextLine();

System.out.print("Number of Packages: ");

int numPackages = scanner.nextInt();

System.out.print("Weight (kg): ");

double weight = scanner.nextDouble();

System.out.print("Service ID: ");

int serviceId = scanner.nextInt();

Courier newCourier = new Courier(senderId, receiverId, packageName, numPackages, weight, serviceId);

service.addCourier(newCourier);

System.out.println("Courier added successfully with tracking number: " + newCourier.getTrackingNumber());

break;

case 2:

System.out.print("Enter Courier ID: ");

int courierId = scanner.nextInt();

System.out.print("Enter Employee ID to assign: ");

int employeeId = scanner.nextInt();

service.assignCourier(courierId, employeeId);

break;

case 3:

System.out.print("Enter Courier ID: ");

int cid = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Enter new Status(Proceesing/Cancelled/Delivered): ");

String newStatus = scanner.nextLine();

service.updateCourierStatus(cid, newStatus);

break;

case 4:

System.out.print("Enter Tracking Number: ");

String trackingNumber = scanner.nextLine();

service.getDeliveryHistory(trackingNumber);

break;

case 5:

service.generateShipmentStatusReport();

break;

case 6:

service.generateRevenueReport();

break;

case 7:

System.out.println("Exiting program...");

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

try {

if (connection != null && !connection.isClosed()) {

connection.close();

System.out.println("Database connection closed.");

}

} catch (Exception e) {

System.err.println("Error closing DB connection: " + e.getMessage());

}

scanner.close();

System.out.println("Scanner closed. Goodbye!");

}

}

package util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnUtil {

public static Connection getConnection(String connectionString) {

Connection connection = null;

try {

connection = DriverManager.getConnection(connectionString);

System.out.println("Database connection established successfully.");

} catch (SQLException e) {

System.err.println("Failed to connect to DB: " + e.getMessage());

}

return connection;

}

}

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class DBPropertyUtil {

public static String getConnectionString(String propertyFileName) {

String connectionString = null;

try {

System.out.println("🔍 Attempting to load DB properties from: " + propertyFileName);

Properties props = new Properties();

FileInputStream fis = new FileInputStream(propertyFileName);

props.load(fis);

String url = props.getProperty("db.url"); // Already includes jdbc:mysql://...

String username = props.getProperty("db.username");

String password = props.getProperty("db.password");

if (url == null || username == null || password == null) {

System.err.println("Missing one or more properties (url, username, password).");

} else {

connectionString = url + "?user=" + username + "&password=" + password;

}

} catch (IOException e) {

System.err.println("Error loading properties: " + e.getMessage());

}

return connectionString;

}

}