**Hexaware Assignment**

**Courier Management System**

**Task 1**

**Database Design**

**Design a SQL schema for a Courier Management System with tables for Customers, Couriers, Orders,**

**and Parcels. Define the relationships between these tables using appropriate foreign keys.**

**Requirements:**

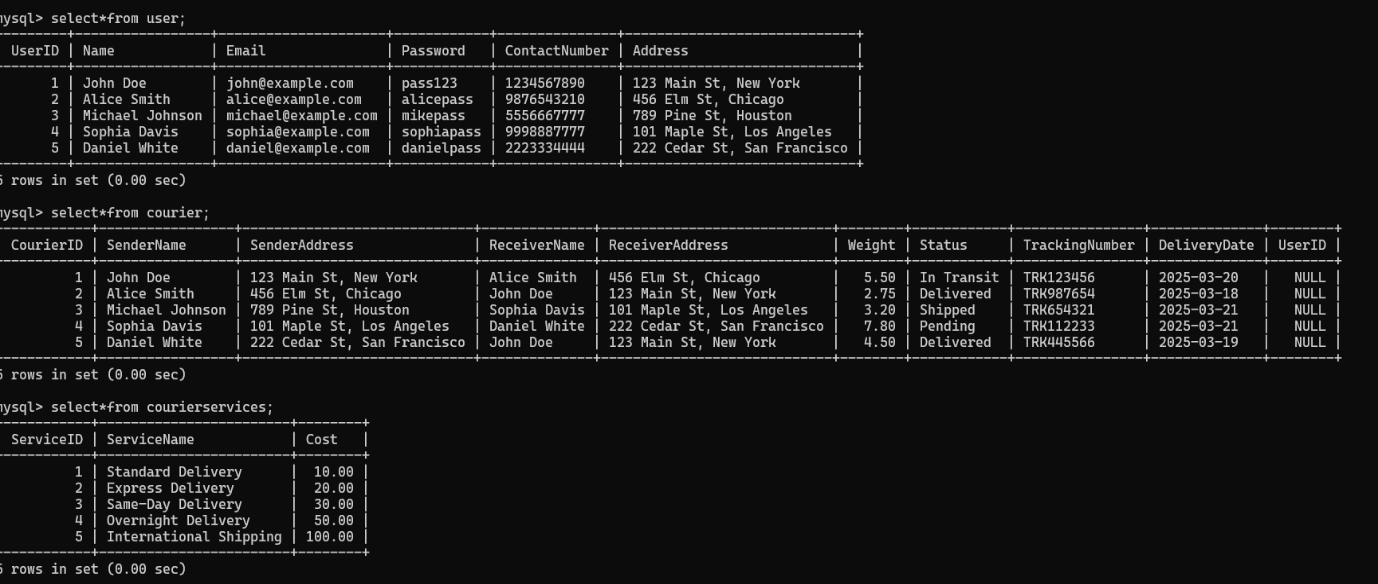
**• Define the Database Schema • Create SQL tables for entities such as User, Courier, Employee,**

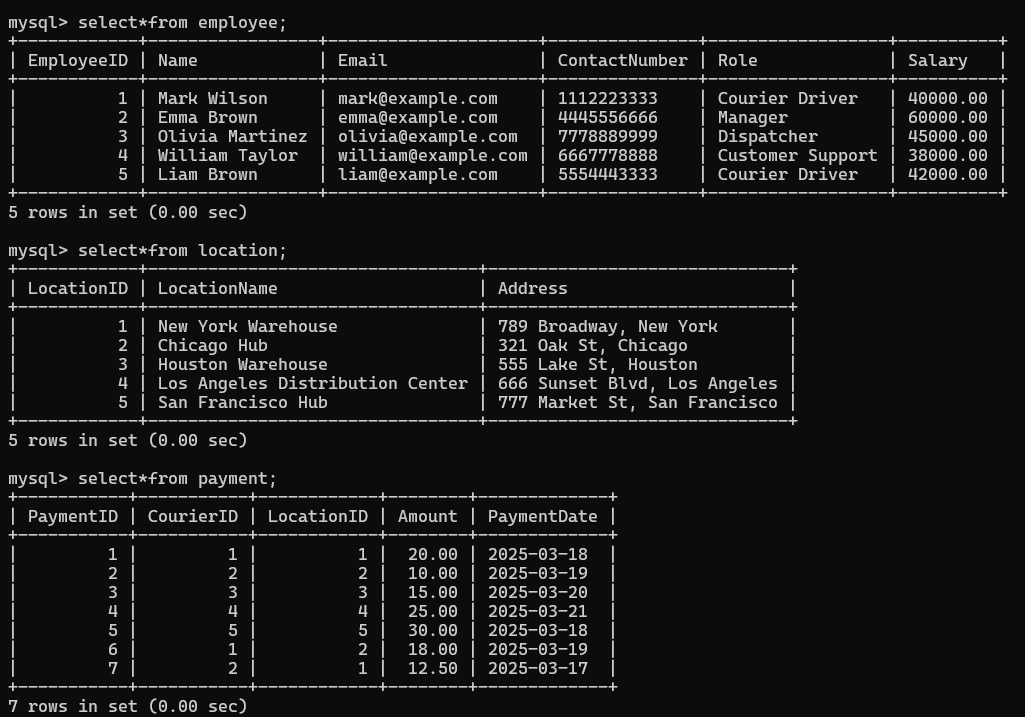
**Location,Payment**

**• Define relationships between these tables (one-to-many, many-to-many, etc.).**

**• Populate Sample Data**

**• Insert sample data into the tables to simulate real-world scenarios**.

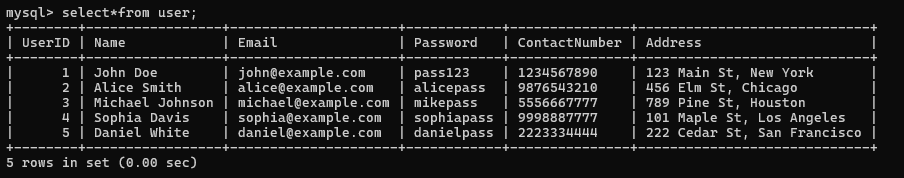




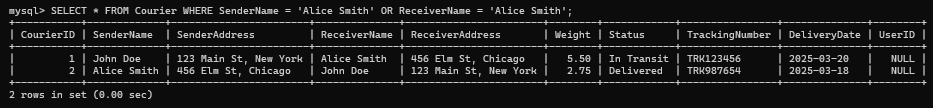
**Task 2**

**Solve the following queries in the Schema that you have created above**

**1. List all customers:**

****

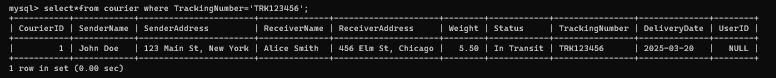
**2. List all orders for a specific customer:**

****

**3. List all couriers:**

****

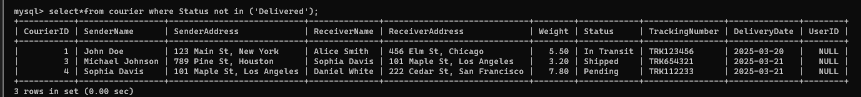
**4. List all packages for a specific order:**

****

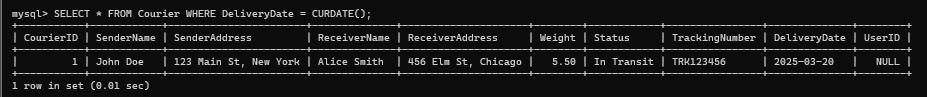
**5. List all deliveries for a specific courier:**

****

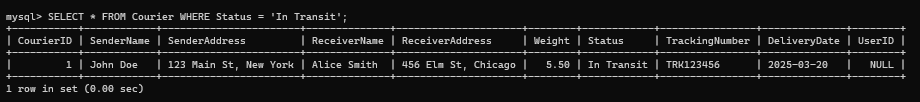
**6. List all undelivered packages:**

****

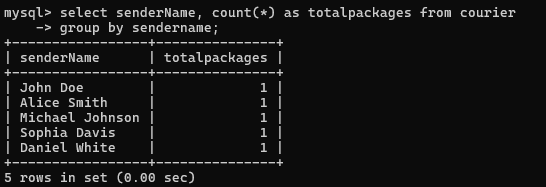
**7. List all packages that are scheduled for delivery today:**

****

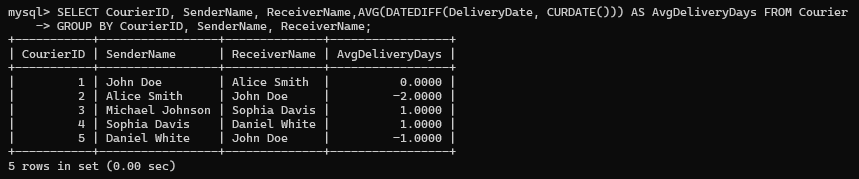
**8. List all packages with a specific status:**

****

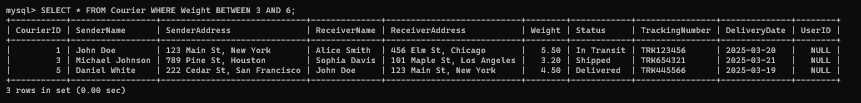
**9. Calculate the total number of packages for each courier.**

****

**10. Find the average delivery time for each courier**

****

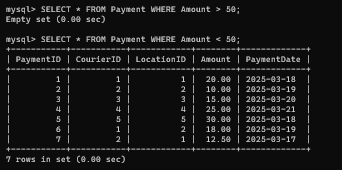
**11. List all packages with a specific weight range:**

****

**12. Retrieve employees whose names contain 'John'**

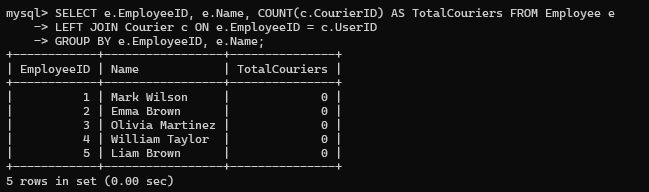
****

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

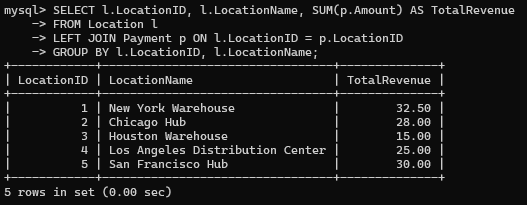
****

**Task 3: Group By, Aggregate Functions, Having, Order By, where**

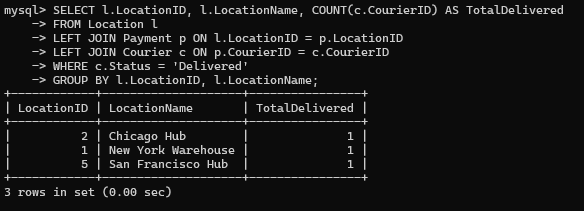
**14. Find the total number of couriers handled by each employee.**

****

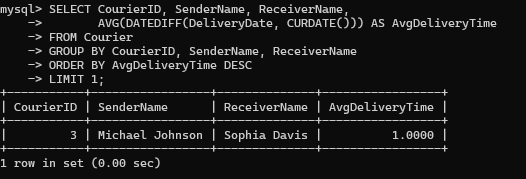
**15. Calculate the total revenue generated by each location**

****

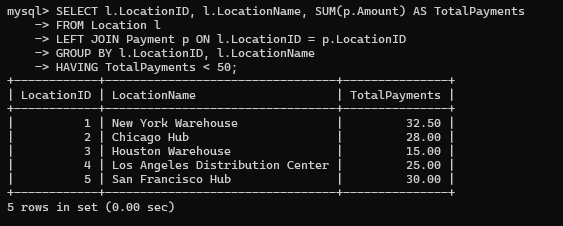
**16. Find the total number of couriers delivered to each location.**

****

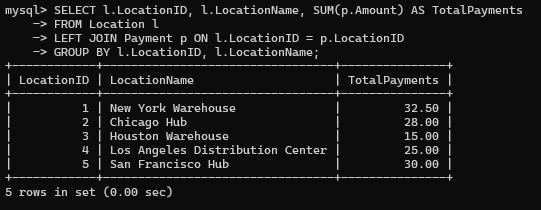
**17. Find the courier with the highest average delivery time:**

****

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

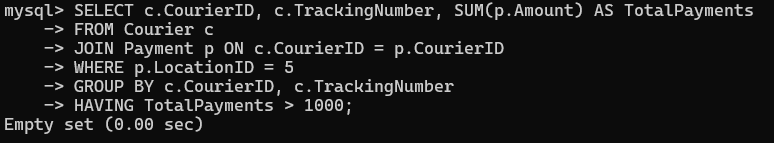
****

**19. Calculate Total Payments per Location**

****

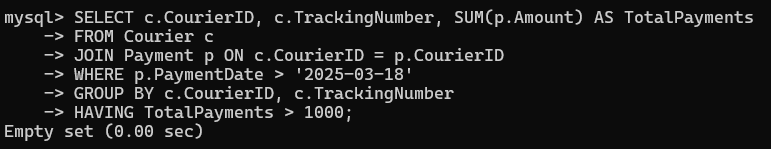
**20. Retrieve couriers who have received payments totaling more than $1000 in a specific location**

**(LocationID = X):**

****

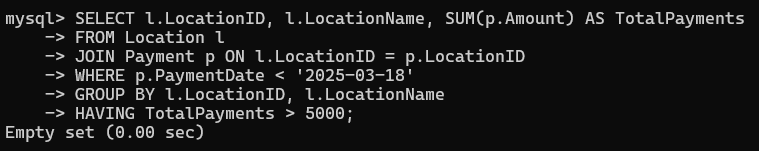
**21. Retrieve couriers who have received payments totaling more than $1000 after a certain date**

**(PaymentDate > 'YYYY-MM-DD'):**

****

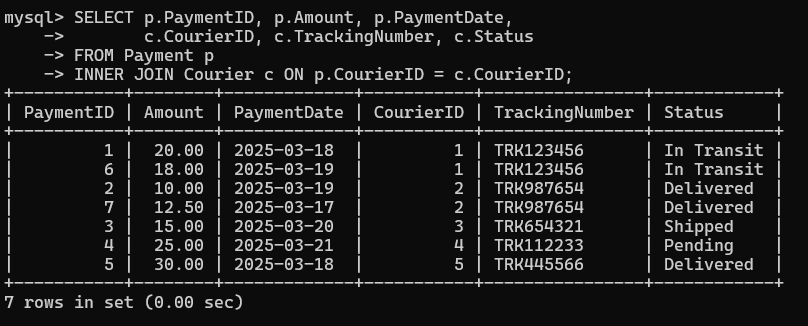
**22. Retrieve locations where the total amount received is more than $5000 before a certain date**

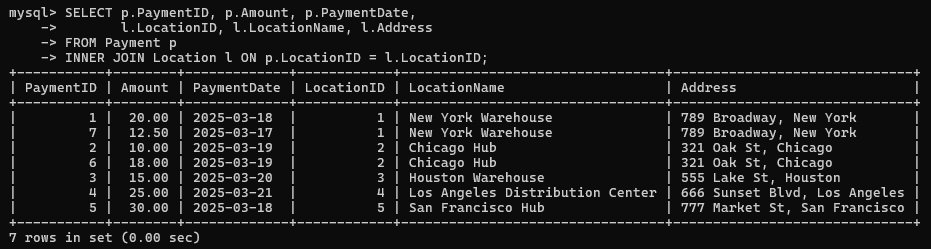
**(PaymentDate > 'YYYY-MM-DD')**

****

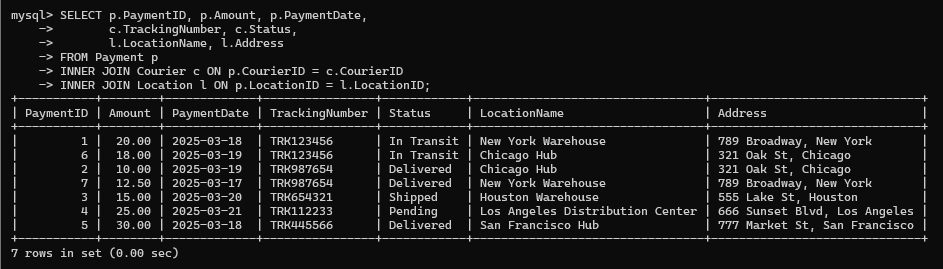
**Task 4: Inner Join,Full Outer Join, Cross Join, Left Outer Join,Right Outer Join**

**23. Retrieve Payments with Courier Information**

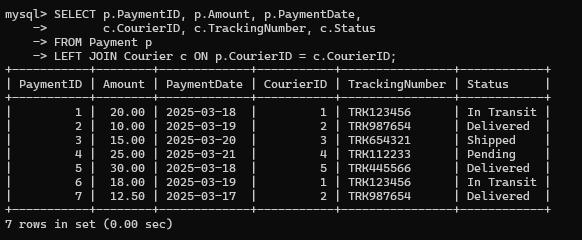
****

**24. Retrieve Payments with Location Information** ****

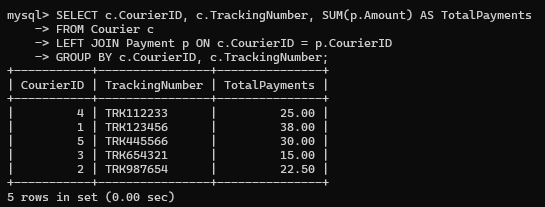
**25. Retrieve Payments with Courier and Location Information**

****

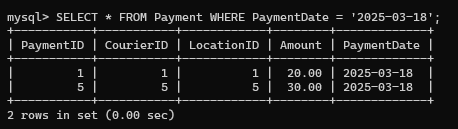
**26. List all payments with courier details**

****

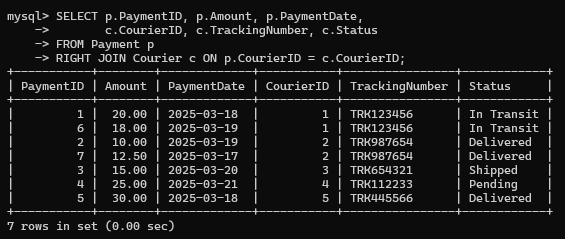
**27. Total payments received for each courier**

****

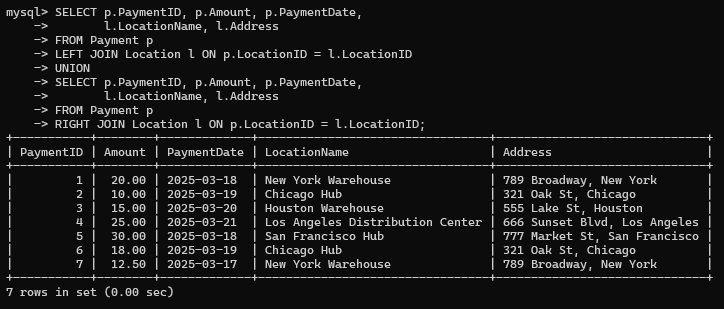
**28. List payments made on a specific date**

****

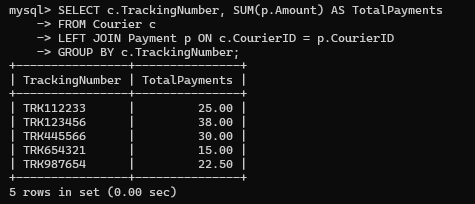
**29. Get Courier Information for Each Payment**

****

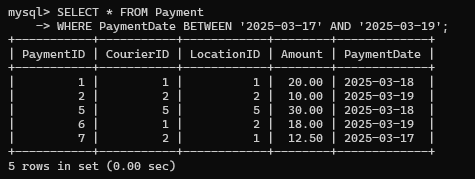
**30. Get Payment Details with Location**

****

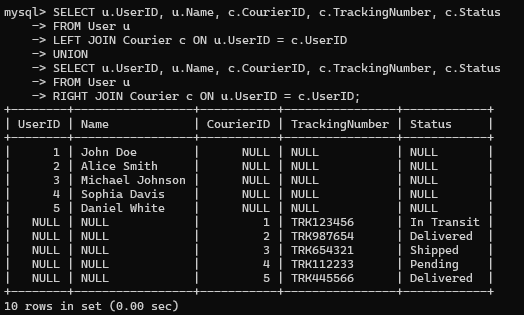
**31. Calculating Total Payments for Each Courier**

****

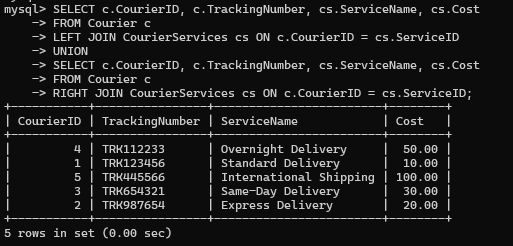
**32. List Payments Within a Date Range**

****

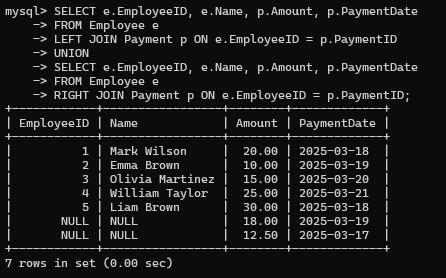
**33. Retrieve a list of all users and their corresponding courier records, including cases where there are no matches on either side**

****

**34. Retrieve a list of all couriers and their corresponding services, including cases where there are no matches on either side**

****

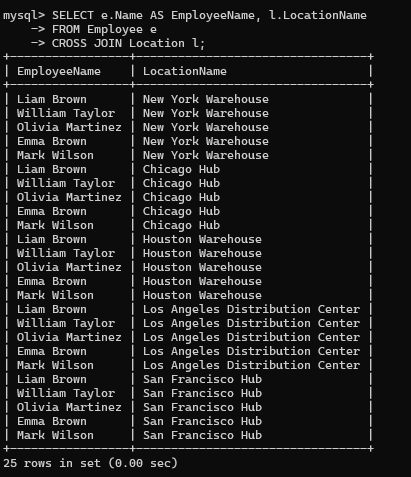
**35. Retrieve a list of all employees and their corresponding payments, including cases where there are no matches on either side**

****

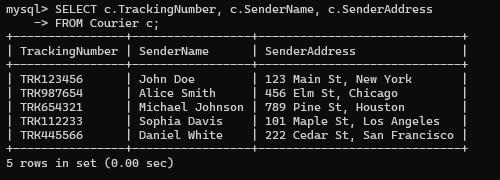
**36. List all users and all courier services, showing all possible combinations.**

****

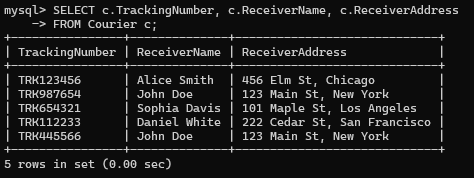
**37. List all employees and all locations, showing all possible combinations:**

****

**38. Retrieve a list of couriers and their corresponding sender information (if available)**

****

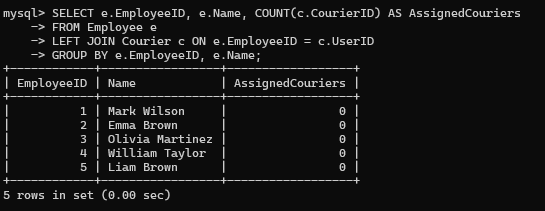
**39. Retrieve a list of couriers and their corresponding receiver information (if available):**

****

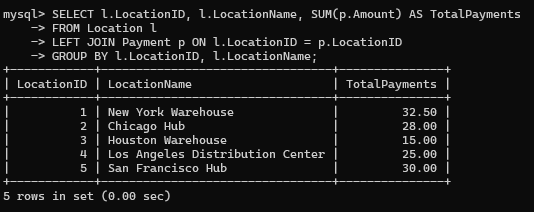
**40. Retrieve a list of couriers along with the courier service details (if available):**

****

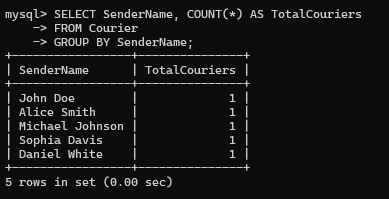
**41. Retrieve a list of employees and the number of couriers assigned to each employee:**

****

**42. Retrieve a list of locations and the total payment amount received at each location:**

****

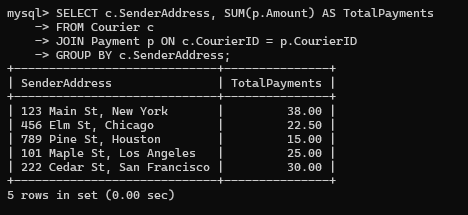
**43. Retrieve all couriers sent by the same sender (based on SenderName).**

****

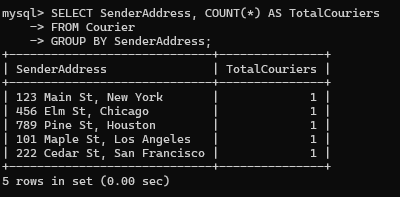
**44. List all employees who share the same role.**

****

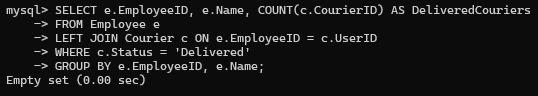
**45. Retrieve all payments made for couriers sent from the same location.**

****

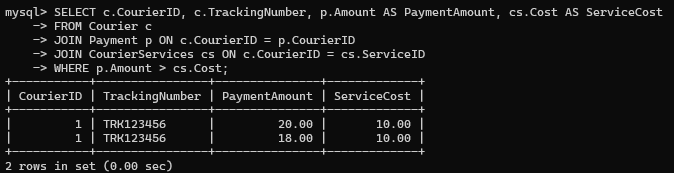
**46. Retrieve all couriers sent from the same location (based on SenderAddress).**

****

**47. List employees and the number of couriers they have delivered:**

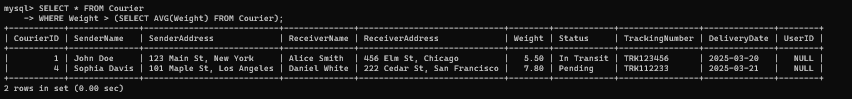
****

**48. Find couriers that were paid an amount greater than the cost of their respective courier services**

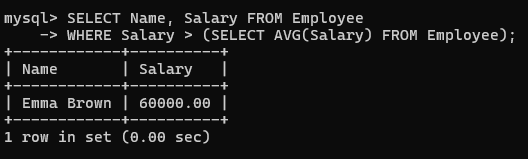
****

**Scope: Inner Queries, Non Equi Joins, Equi joins,Exist,Any,All**

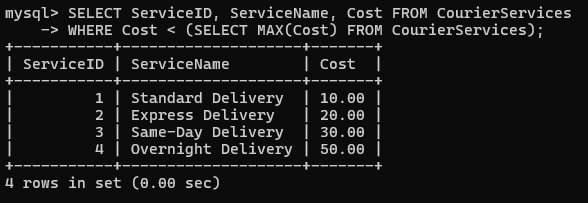
**49. Find couriers that have a weight greater than the average weight of all couriers**

****

**50. Find the names of all employees who have a salary greater than the average salary:**

****

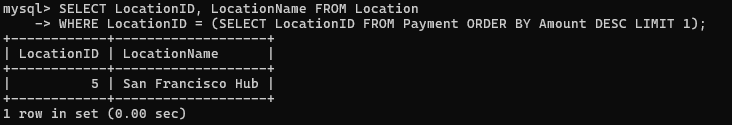
**51. Find the total cost of all courier services where the cost is less than the maximum cost**

****

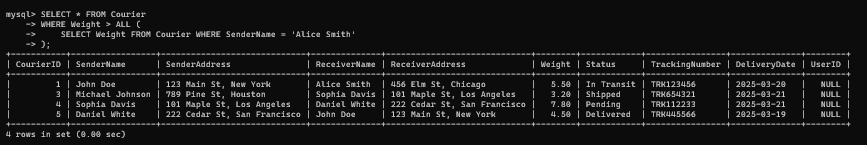
**52. Find all couriers that have been paid for**

****

**53. Find the locations where the maximum payment amount was made**

****

**54. Find all couriers whose weight is greater than the weight of all couriers sent by a specific sender (e.g., 'SenderName'):**

****

**Coding**

**Task 1: Control Flow Statements**

**1. Write a program that checks whether a given order is delivered or not based on its status (e.g.,**

**"Processing," "Delivered," "Cancelled"). Use if-else statements for this.**

|  |
| --- |
| package assignment;  import java.util.HashMap;  import java.util.Map;  import java.util.Scanner;  public class OneIfelse {  public static void main(String[] args) {  Map<Integer, String> orderStatusMap = new HashMap<>();  orderStatusMap.put(101, "Processing");  orderStatusMap.put(102, "Delivered");  orderStatusMap.put(103, "Cancelled");  orderStatusMap.put(104, "Delivered");  orderStatusMap.put(105, "Processing");  Scanner scanner = new Scanner(System.in);  System.out.print("Enter Order ID: ");  int orderId = scanner.nextInt();  if (orderStatusMap.containsKey(orderId)) {  String status = orderStatusMap.get(orderId);  if (status.equalsIgnoreCase("Delivered")) {  System.out.println("The order has been delivered.");  } else if (status.equalsIgnoreCase("Processing")) {  System.out.println("The order is still being processed.");  } else if (status.equalsIgnoreCase("Cancelled")) {  System.out.println("The order was cancelled.");  } else {  System.out.println("Status: " + status);  }  } else {  System.out.println("Order ID not found.");  }  scanner.close();  }  }  **Output:**  Enter Order ID: 101  The order is still being processed. |

**2. Implement a switch-case statement to categorize parcels based on their weight into "Light,"**

**"Medium," or "Heavy."**

|  |
| --- |
| package assignment;  import java.util.Scanner;  public class TwoSwitchcase {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.*in*);  System.*out*.print("Enter parcel weight in kg: ");  int weight = scanner.nextInt();  String category;  switch (weight / 10) {  case 0:  case 1:  category = "Light";  break;  case 2:  case 3:  case 4:  category = "Medium";  break;  default:  category = "Heavy";  }  System.*out*.println("Parcel category: " + category);  scanner.close();  }  }  **Output:**  Enter parcel weight in kg: 20  Parcel category: Medium |

**3. Implement User Authentication 1. Create a login system for employees and customers using Java**

**control flow statements.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **public** **class** ThreeAuthentication {  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);  String employeeUsername = "employee01";  String employeePassword = "emp123";  String customerUsername = "customer01";  String customerPassword = "cust123";  System.***out***.println("Welcome! Are you an Employee or Customer?");  System.***out***.print("Enter role (Employee/Customer): ");  String role = scanner.nextLine().trim();  System.***out***.print("Enter Username: ");  String username = scanner.nextLine().trim();  System.***out***.print("Enter Password: ");  String password = scanner.nextLine().trim();  **if** (role.equalsIgnoreCase("Employee")) {  **if** (username.equals(employeeUsername) && password.equals(employeePassword)) {  System.***out***.println("Employee login successful.");  } **else** {  System.***out***.println("Invalid employee credentials.");  }  } **else** **if** (role.equalsIgnoreCase("Customer")) {  **if** (username.equals(customerUsername) && password.equals(customerPassword)) {  System.***out***.println("Customer login successful.");  } **else** {  System.***out***.println("Invalid customer credentials.");  }  } **else** {  System.***out***.println("Invalid role entered.");  }  scanner.close();  }  }  **Output:**  Welcome! Are you an Employee or Customer?  Enter role (Employee/Customer): employee  Enter Username: employee01  Enter Password: emp123 |

**4. Implement Courier Assignment Logic 1. Develop a mechanism to assign couriers to shipments based on predefined criteria (e.g., proximity, load capacity) using loops.**

|  |
| --- |
| package assignment;  import java.util.\*;  class Courier {  int courierId;  String name;  int maxCapacity;  int currentLoad;  int distanceFromPickup;  public Courier(int courierId, String name, int maxCapacity, int currentLoad, int distanceFromPickup) {  this.courierId = courierId;  this.name = name;  this.maxCapacity = maxCapacity;  this.currentLoad = currentLoad;  this.distanceFromPickup = distanceFromPickup;  }  public boolean canTakeShipment(int shipmentWeight) {  return (maxCapacity - currentLoad) >= shipmentWeight;  }  }  public class FourCourierAssignment {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.*in*);  Courier[] couriers = {  new Courier(1, "Ravi Kumar", 100, 80, 5),  new Courier(2, "Meena Joshi", 120, 50, 2),  new Courier(3, "Arjun Singh", 90, 85, 3)  };  System.*out*.print("Enter shipment weight: ");  int shipmentWeight = scanner.nextInt();  Courier bestCourier = null;  int shortestDistance = Integer.*MAX\_VALUE*;  for (Courier c : couriers) {  if (c.canTakeShipment(shipmentWeight)) {  if (c.distanceFromPickup < shortestDistance) {  bestCourier = c;  shortestDistance = c.distanceFromPickup;  }  }  }  if (bestCourier != null) {  System.*out*.println("Assigned Courier: " + bestCourier.name + " (ID: " + bestCourier.courierId + ")");  } else {  System.*out*.println("No suitable courier available for this shipment.");  }  scanner.close();  }  }  **Output:**  Enter shipment weight: 30  Assigned Courier: Meena Joshi (ID: 2) |

**Task 2: Loops and Iteration**

**5. Write a Java program that uses a for loop to display all the orders for a specific customer.**

|  |
| --- |
| **package** assignment;  **import** java.util.\*;  **class** Order {  **int** orderId;  String customerName;  String item;  String status;  **public** Order(**int** orderId, String customerName, String item, String status) {  **this**.orderId = orderId;  **this**.customerName = customerName;  **this**.item = item;  **this**.status = status;  }  }  **public** **class** FiveLoops {  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);  Order[] orders = {  **new** Order(101, "Samikksha", "Mobile Phone", "Delivered"),  **new** Order(102, "Ravi", "Laptop", "Processing"),  **new** Order(103, "Samikksha", "Headphones", "Shipped"),  **new** Order(104, "Meena", "Camera", "Cancelled"),  **new** Order(105, "Samikksha", "Smartwatch", "Delivered")  };  System.***out***.print("Enter customer name: ");  String customer = scanner.nextLine();  **boolean** found = **false**;  System.***out***.println("Orders for " + customer + ":");  **for** (**int** i = 0; i < orders.length; i++) {  **if** (orders[i].customerName.equalsIgnoreCase(customer)) {  System.***out***.println("Order ID: " + orders[i].orderId + ", Item: " + orders[i].item + ", Status: " + orders[i].status);  found = **true**;  }  }  **if** (!found) {  System.***out***.println("No orders found for this customer.");  }  scanner.close();  }  }  **Output:**  Enter customer name: Samikksha  Orders for Samikksha:  Order ID: 101, Item: Mobile Phone, Status: Delivered  Order ID: 103, Item: Headphones, Status: Shipped  Order ID: 105, Item: Smartwatch, Status: Delivered |

**6. Implement a while loop to track the real-time location of a courier until it reaches its destination.**

|  |
| --- |
| **package** assignment;  **import** java.util.\*;  **public** **class** SixLocationTracking {  @SuppressWarnings("resource")  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);  Map<Integer, String[]> courierRoutes = **new** HashMap<>();  courierRoutes.put(1, **new** String[]{"New York Warehouse", "New Jersey", "Philadelphia", "Baltimore", "Washington D.C."});  courierRoutes.put(2, **new** String[]{"Chicago Hub", "Springfield", "St. Louis", "Kansas City", "Denver"});  courierRoutes.put(3, **new** String[]{"San Francisco Hub", "San Jose", "Fresno", "Bakersfield", "Los Angeles Distribution Center"});  System.***out***.print("Enter Courier ID to track: ");  **int** courierId = scanner.nextInt();  **if** (!courierRoutes.containsKey(courierId)) {  System.***out***.println("Courier ID not found.");  **return**;  }  String[] route = courierRoutes.get(courierId);  **int** currentIndex = 0;  String currentLocation = route[currentIndex];  String destination = route[route.length - 1];  System.***out***.println("Tracking Courier ID " + courierId + "...");  **while** (!currentLocation.equals(destination)) {  System.***out***.println("Current Location: " + currentLocation);  currentIndex++;  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  System.***out***.println("Tracking interrupted");  }  currentLocation = route[currentIndex];  }  System.***out***.println("Current Location: " + currentLocation);  System.***out***.println("Courier has reached the destination.");  scanner.close();  }  }  **Output:**  Enter Courier ID to track: 1  Tracking Courier ID 1...  Current Location: New York Warehouse  Current Location: New Jersey  Current Location: Philadelphia  Current Location: Baltimore  Current Location: Washington D.C.  Courier has reached the destination. |

**Task 3: Arrays and Data Structures**

**7. Create an array to store the tracking history of a parcel, where each entry represents a location**

**update.**

|  |
| --- |
| **package** assignment;  **import** java.util.\*;  **public** **class** SevenTrackingHistory {  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);    List<String> trackingHistory = **new** ArrayList<>();  System.***out***.print("Enter number of location updates: ");  **int** count = scanner.nextInt();  scanner.nextLine();  **for** (**int** i = 0; i < count; i++) {  System.***out***.print("Enter location update " + (i + 1) + ": ");  String location = scanner.nextLine();  trackingHistory.add(location);  }  System.***out***.println("\nTracking History:");  **for** (String location : trackingHistory) {  System.***out***.println("- " + location);  }  scanner.close();  }  }  **Output:**  Enter number of location updates: 1  Enter location update 1: Chennai  Tracking History:  - Chennai |

**8. Implement a method to find the nearest available courier for a new order using an array of couriers.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **class** CourierInfo {  String name;  **int** distanceFromWarehouse;  **boolean** isAvailable;  **public** CourierInfo(String name, **int** distanceFromWarehouse, **boolean** isAvailable) {  **this**.name = name;  **this**.distanceFromWarehouse = distanceFromWarehouse;  **this**.isAvailable = isAvailable;  }  }  **public** **class** EightAvailableCourier {  **public** **static** CourierInfo findNearestCourier(CourierInfo[] couriers, **int** targetDistance) {  CourierInfo nearest = **null**;  **int** minDifference = Integer.***MAX\_VALUE***;  **for** (CourierInfo c : couriers) {  **if** (c.isAvailable) {  **int** diff = Math.*abs*(c.distanceFromWarehouse - targetDistance);  **if** (diff < minDifference) {  minDifference = diff;  nearest = c;  }  }  }  **return** nearest;  }  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);  CourierInfo[] couriers = {  **new** CourierInfo("Ravi Kumar", 10, **true**),  **new** CourierInfo("Meena Joshi", 20, **true**),  **new** CourierInfo("Arjun Singh", 15, **false**),  **new** CourierInfo("Neha Sharma", 25, **true**)  };  System.***out***.print("Enter your delivery location distance from warehouse: ");  **int** userDistance = scanner.nextInt();  CourierInfo nearest = *findNearestCourier*(couriers, userDistance);  **if** (nearest != **null**) {  System.***out***.println("Nearest available courier is: " + nearest.name + " (Distance: " + nearest.distanceFromWarehouse + ")");  } **else** {  System.***out***.println("No available courier found near your location.");  }  scanner.close();  }  }  **Output:**  Enter your delivery location distance from warehouse: 30  Nearest available courier is: Neha Sharma (Distance: 25) |

**Task 4: Strings,2d Arrays, user defined functions,Hashmap**

**9. Parcel Tracking: Create a program that allows users to input a parcel tracking number.Store the**

**tracking number and Status in 2d String Array. Initialize the array with values. Then, simulate the**

**tracking process by displaying messages like "Parcel in transit," "Parcel out for delivery," or "Parcel**

**delivered" based on the tracking number's status.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **public** **class** NineParcelTracking {  **public** **static** **void** main(String[] args) {  String[][] trackingData = {  {"TRK1001", "In Transit"},  {"TRK1002", "Out for Delivery"},  {"TRK1003", "Delivered"},  {"TRK1004", "In Transit"},  {"TRK1005", "Delivered"}  };  Scanner scanner = **new** Scanner(System.***in***);  System.***out***.print("Enter Tracking Number: ");  String inputTracking = scanner.nextLine();  **boolean** found = **false**;  **for** (**int** i = 0; i < trackingData.length; i++) {  **if** (trackingData[i][0].equalsIgnoreCase(inputTracking)) {  found = **true**;  String status = trackingData[i][1];  **if** (status.equalsIgnoreCase("In Transit")) {  System.***out***.println("Parcel is currently in transit.");  } **else** **if** (status.equalsIgnoreCase("Out for Delivery")) {  System.***out***.println("Parcel is out for delivery.");  } **else** **if** (status.equalsIgnoreCase("Delivered")) {  System.***out***.println("Parcel has been delivered.");  } **else** {  System.***out***.println("Unknown status.");  }  **break**;  }  }  **if** (!found) {  System.***out***.println("Tracking number not found.");  }  scanner.close();  }  }  **Output:**  Enter Tracking Number: TRK1003  Parcel has been delivered. |

**10. Customer Data Validation: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name addtress or phone number.Validate customer information based on**

**following critirea. Ensure that names contain only letters and are properly capitalized, addresses do not contain special characters, and phone numbers follow a specific format (e.g., ###-###-####).**

|  |
| --- |
| **package** assignment;  **public** **class** TenCustomerData {  **public** **static** **boolean** validate(String data, String detail) {  **switch** (detail.toLowerCase()) {  **case** "name":    **return** data.matches("([A-Z][a-z]+)(\\s[A-Z][a-z]+)\*");  **case** "address":    **return** data.matches("[\\w\\s.,'-]+");  **case** "phone":    **return** data.matches("\\d{3}-\\d{3}-\\d{4}");  **default**:  **return** **false**;  }  }  **public** **static** **void** main(String[] args) {  System.***out***.println(*validate*("Samikksha Raj", "name")); // true  System.***out***.println(*validate*("123 Main Street", "address")); // true  System.***out***.println(*validate*("123-456-7890", "phone")); // true  System.***out***.println(*validate*("john@doe", "name")); // false  System.***out***.println(*validate*("No.12@# Main Rd", "address")); // false  System.***out***.println(*validate*("1234567890", "phone")); // false  }  }  **Output:**  Enter Tracking Number: TRK1003  Parcel has been delivered. |

**11. Address Formatting: Develop a function that takes an address as input (street, city, state, zip code) and formats it correctly, including capitalizing the first letter of each word and properly formatting the zip code.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **public** **class** ElevenAddressFormatter {  **public** **static** String capitalizeWords(String input) {  String[] words = input.trim().toLowerCase().split(" ");  StringBuilder capitalized = **new** StringBuilder();  **for** (String word : words) {  **if** (!word.isEmpty()) {  capitalized.append(Character.*toUpperCase*(word.charAt(0)))  .append(word.substring(1))  .append(" ");  }  }  **return** capitalized.toString().trim();  }  **public** **static** **boolean** isValidZip(String zip) {  **return** zip.matches("\\d{6}");  }  **public** **static** String formatAddress(String street, String city, String state, String zip) {  **if** (!*isValidZip*(zip)) {  **return** "Invalid ZIP code. Must be 5 digits.";  }  String formattedStreet = *capitalizeWords*(street);  String formattedCity = *capitalizeWords*(city);  String formattedState = *capitalizeWords*(state);  **return** formattedStreet + ", " + formattedCity + ", " + formattedState + " - " + zip;  }  **public** **static** **void** main(String[] args) {  @SuppressWarnings("resource")  Scanner sc = **new** Scanner(System.***in***);  System.***out***.print("Enter Street: ");  String street = sc.nextLine();  System.***out***.print("Enter City: ");  String city = sc.nextLine();  System.***out***.print("Enter State: ");  String state = sc.nextLine();  System.***out***.print("Enter ZIP Code: ");  String zip = sc.nextLine();  String formattedAddress = *formatAddress*(street, city, state, zip);  System.***out***.println("\nFormatted Address:");  System.***out***.println(formattedAddress);  }  }  **Output:**  Enter Street: 1st cross street  Enter City: Chennai  Enter State: Tamil Nadu  Enter ZIP Code: 600024  Formatted Address:  1st Cross Street, Chennai, Tamil Nadu - 600024 |

**12. Order Confirmation Email: Create a program that generates an order confirmation email. The email should include details such as the customer's name, order number, delivery address, and expected delivery date.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **public** **class** TwelveOrderConformation {  **public** **static** String generateEmail(String name, String orderNumber, String address, String deliveryDate) {  String email = "Dear " + name + ",\n\n" +  "Thank you for your order!\n" +  "Your order number is: " + orderNumber + "\n\n" +  "Delivery Address:\n" + address + "\n\n" +  "Expected Delivery Date: " + deliveryDate + "\n\n" +  "We appreciate your business and hope you enjoy your purchase!\n\n" +  "Best regards,\n" +  "Courier Service Team";  **return** email;  }  **public** **static** **void** main(String[] args) {  @SuppressWarnings("resource")  Scanner sc = **new** Scanner(System.***in***);  System.***out***.print("Enter Customer Name: ");  String name = sc.nextLine();  System.***out***.print("Enter Order Number: ");  String orderNumber = sc.nextLine();  System.***out***.print("Enter Delivery Address: ");  String address = sc.nextLine();  System.***out***.print("Enter Expected Delivery Date (e.g., April 10, 2025): ");  String deliveryDate = sc.nextLine();  String emailContent = *generateEmail*(name, orderNumber, address, deliveryDate);  System.***out***.println("\n--- Order Confirmation Email ---\n");  System.***out***.println(emailContent);  }  }  **Output:**  Enter Customer Name: Samikksha  Enter Order Number: 123  Enter Delivery Address: !st cross street Chennai-24  Enter Expected Delivery Date (e.g., April 10, 2025): 12-10-2025  --- Order Confirmation Email ---  Dear Samikksha,  Thank you for your order!  Your order number is: 123  Delivery Address:  !st cross street Chennai-24  Expected Delivery Date: 12-10-2025  We appreciate your business and hope you enjoy your purchase!  Best regards,  Courier Service Team |

**13. Calculate Shipping Costs: Develop a function that calculates the shipping cost based on the distance between two locations and the weight of the parcel. You can use string inputs for the source and destination addresses.**

|  |
| --- |
| **package** assignment;  **import** java.util.Scanner;  **public** **class** ThirteenShippingCost {    **public** **static** **int** getDistance(String source, String destination) {  source = source.toLowerCase();  destination = destination.toLowerCase();  **if** (source.equals(destination)) **return** 0;  **if** ((source.equals("chennai") && destination.equals("bangalore")) ||  (source.equals("bangalore") && destination.equals("chennai")))  **return** 350;  **else** **if** ((source.equals("chennai") && destination.equals("mumbai")) ||  (source.equals("mumbai") && destination.equals("chennai")))  **return** 1330;  **else** **if** ((source.equals("delhi") && destination.equals("mumbai")) ||  (source.equals("mumbai") && destination.equals("delhi")))  **return** 1450;  **else** **if** ((source.equals("kolkata") && destination.equals("delhi")) ||  (source.equals("delhi") && destination.equals("kolkata")))  **return** 1500;  **else**  **return** 1000;  }    **public** **static** **double** calculateShippingCost(String source, String destination, **double** weightKg) {  **int** distance = *getDistance*(source, destination);  **double** baseRate = 50.0;  **double** ratePerKm = 0.5;  **double** ratePerKg = 10.0;  **return** baseRate + (distance \* ratePerKm) + (weightKg \* ratePerKg);  }  **public** **static** **void** main(String[] args) {  @SuppressWarnings("resource")  Scanner sc = **new** Scanner(System.***in***);  System.***out***.print("Enter source location: ");  String source = sc.nextLine();  System.***out***.print("Enter destination location: ");  String destination = sc.nextLine();  System.***out***.print("Enter weight of the parcel (kg): ");  **double** weight = sc.nextDouble();  **double** cost = *calculateShippingCost*(source, destination, weight);  System.***out***.printf("Shipping cost from %s to %s for %.2f kg = %.2f\n",  source, destination, weight, cost);  }  }  **Output:**  Enter source location: Chennai  Enter destination location: Banglore  Enter weight of the parcel (kg): 10  Shipping cost from Chennai to Banglore for 10.00 kg = 650.00 |

**14. Password Generator: Create a function that generates secure passwords for courier system**

**accounts. Ensure the passwords contain a mix of uppercase letters, lowercase letters, numbers, and special characters.**

|  |  |
| --- | --- |
| **package** assignment;  **import** java.util.Random;  **public** **class** FourteenPasswordGenerator {  **public** **static** String generatePassword(**int** length) {  String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  String lower = "abcdefghijklmnopqrstuvwxyz";  String numbers = "0123456789";  String special = "!@#$%^&\*()-\_=+<>?/";  String combined = upper + lower + numbers + special;  Random random = **new** Random();    StringBuilder password = **new** StringBuilder();  password.append(upper.charAt(random.nextInt(upper.length())));  password.append(lower.charAt(random.nextInt(lower.length())));  password.append(numbers.charAt(random.nextInt(numbers.length())));  password.append(special.charAt(random.nextInt(special.length())));    **for** (**int** i = 4; i < length; i++) {  password.append(combined.charAt(random.nextInt(combined.length())));  }  **return** password.toString();  }  **public** **static** **void** main(String[] args) {  **int** desiredLength = 12;  String password = *generatePassword*(desiredLength);  System.***out***.println("Generated Secure Password: " + password);  }  }  **Output:**  Generated Secure Password: Rp2-%jy)jGJ6 |  |

**15. Find Similar Addresses: Implement a function that finds similar addresses in the system. This can be useful for identifying duplicate customer entries or optimizing delivery routes.Use string functions to implement this.**

|  |
| --- |
| **package** assignment;  **public** **class** FifteenSimilarAdd {  // Function to calculate similarity score between two addresses  **public** **static** **int** getSimilarityScore(String address1, String address2) {  address1 = address1.toLowerCase().replaceAll("[^a-z0-9 ]", "");  address2 = address2.toLowerCase().replaceAll("[^a-z0-9 ]", "");  String[] words1 = address1.split(" ");  String[] words2 = address2.split(" ");  **int** common = 0;  **for** (String word1 : words1) {  **for** (String word2 : words2) {  **if** (word1.equals(word2)) {  common++;  **break**;  }  }  }  **return** (**int**) (((**double**) common / Math.*max*(words1.length, words2.length)) \* 100); // return similarity percentage  }  // Function to find similar addresses  **public** **static** **void** findSimilarAddresses(String[] addresses, **int** threshold) {  **for** (**int** i = 0; i < addresses.length; i++) {  **for** (**int** j = i + 1; j < addresses.length; j++) {  **int** score = *getSimilarityScore*(addresses[i], addresses[j]);  **if** (score >= threshold) {  System.***out***.println("Similar Addresses Found (" + score + "% match):");  System.***out***.println(" - " + addresses[i]);  System.***out***.println(" - " + addresses[j]);  System.***out***.println();  }  }  }  }  **public** **static** **void** main(String[] args) {  String[] addressList = {  "123 Main Street, New York",  "123 Main St., New York",  "456 Elm Street, Los Angeles",  "123 Main Street, NY",  "789 Oak Avenue, Chicago",  "123 Main Street New York"  };  **int** similarityThreshold = 70; // percentage  *findSimilarAddresses*(addressList, similarityThreshold);  }  }  **Output:**  Similar Addresses Found (80% match):  - 123 Main Street, New York  - 123 Main St., New York  Similar Addresses Found (100% match):  - 123 Main Street, New York  - 123 Main Street New York  Similar Addresses Found (80% match):  - 123 Main St., New York  - 123 Main Street New York |

**Following tasks are incremental stages to build an application and should be done in a single project**

**Task 5: Object Oriented Programming**

**Scope : Entity classes/Models/POJO, Abstraction/Encapsulation**

**Create the following model/entity classes within package entities with variables declared**

**private, constructors(default and parametrized,getters,setters and toString())**

**1. User Class:**

**Variables:userID , userName , email , password , contactNumber , address**

|  |
| --- |
| **package** entities;  **public** **class** User {  **private** **int** userID;  **private** String userName;  **private** String email;  **private** String password;  **private** String contactNumber;  **private** String address;  **public** User() {}  **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;  }  **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 + '\'' +  ", contactNumber='" + contactNumber + '\'' +  ", address='" + address + '\'' +  '}';  }  } |

**2. Courier Class**

**Variables: courierID , senderName , senderAddress , receiverName , receiverAddress , weight ,**

**status, trackingNumber , deliveryDate ,userId**

|  |
| --- |
| **package** entities;  **public** **class** User {  **private** **int** userID;  **private** String userName;  **private** String email;  **private** String password;  **private** String contactNumber;  **private** String address;  **public** User() {}  **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;  }  **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 + '\'' +  ", contactNumber='" + contactNumber + '\'' +  ", address='" + address + '\'' +  '}';  }  } |

**3. Employee Class:**

**Variables employeeID , employeeName , email , contactNumber , role String, salary**

|  |
| --- |
| **package** entities;  **public** **class** Employee {  **private** **int** employeeID;  **private** String employeeName;  **private** String email;  **private** String contactNumber;  **private** String role;  **private** **double** salary;  **public** Employee() {}  **public** Employee(**int** employeeID, String employeeName, String email, String contactNumber, String role, **double** salary) {  **this**.employeeID = employeeID;  **this**.employeeName = employeeName;  **this**.email = email;  **this**.contactNumber = contactNumber;  **this**.role = role;  **this**.salary = salary;  }  **public** **int** getEmployeeID() { **return** employeeID; }  **public** **void** setEmployeeID(**int** employeeID) { **this**.employeeID = employeeID; }  **public** String getEmployeeName() { **return** employeeName; }  **public** **void** setEmployeeName(String employeeName) { **this**.employeeName = employeeName; }  **public** String getEmail() { **return** email; }  **public** **void** setEmail(String email) { **this**.email = email; }  **public** String getContactNumber() { **return** contactNumber; }  **public** **void** setContactNumber(String contactNumber) { **this**.contactNumber = contactNumber; }  **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; }  @Override  **public** String toString() {  **return** "Employee{" +  "employeeID=" + employeeID +  ", employeeName='" + employeeName + '\'' +  ", email='" + email + '\'' +  ", contactNumber='" + contactNumber + '\'' +  ", role='" + role + '\'' +  ", salary=" + salary +  '}';  }  } |

**4. Location Class**

**Variables LocationID , LocationName , Address**

|  |
| --- |
| **package** entities;  **public** **class** Location {  **private** **int** locationID;  **private** String locationName;  **private** String address;  **public** Location() {}  **public** Location(**int** locationID, String locationName, String address) {  **this**.locationID = locationID;  **this**.locationName = locationName;  **this**.address = address;  }  **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 + '\'' +  '}';  }  } |

**5. CourierCompany Class**

**Variables companyName , courierDetails -collection of Courier Objects, employeeDetails-**

**collection of Employee Objects, locationDetails - collection of Location Objects.**

|  |
| --- |
| package entity;  import java.util.List;  public class CourierCompany {  private String companyName;  private List<Courier> courierDetails;  private List<Employee> employeeDetails;  private List<Location> locationDetails;  // Default constructor  public CourierCompany() {}  // Parameterized constructor  public CourierCompany(String companyName, List<Courier> courierDetails, List<Employee> employeeDetails, List<Location> locationDetails) {  this.companyName = companyName;  this.courierDetails = courierDetails;  this.employeeDetails = employeeDetails;  this.locationDetails = locationDetails;  }  // Getters and Setters  public String getCompanyName() {  return companyName;  }  public void setCompanyName(String companyName) {  this.companyName = companyName;  }  public List<Courier> getCourierDetails() {  return courierDetails;  }  public void setCourierDetails(List<Courier> courierDetails) {  this.courierDetails = courierDetails;  }  public List<Employee> getEmployeeDetails() {  return employeeDetails;  }  public void setEmployeeDetails(List<Employee> employeeDetails) {  this.employeeDetails = employeeDetails;  }  public List<Location> getLocationDetails() {  return locationDetails;  }  public void setLocationDetails(List<Location> locationDetails) {  this.locationDetails = locationDetails;  }  // toString method  @Override  public String toString() {  return "CourierCompany [CompanyName=" + companyName +  ", Couriers=" + courierDetails +  ", Employees=" + employeeDetails +  ", Locations=" + locationDetails + "]";  }  } |

**6. Payment Class:**

**Variables PaymentID long, CourierID long, Amount double, PaymentDate Date**

|  |
| --- |
| package entity;  import java.util.Date;  public class Payment {  private long paymentId;  private long courierId;  private double amount;  private Date paymentDate;  // Default constructor  public Payment() {}  // Parameterized constructor  public Payment(long paymentId, long courierId, double amount, Date paymentDate) {  this.paymentId = paymentId;  this.courierId = courierId;  this.amount = amount;  this.paymentDate = paymentDate;  }  // Getters and Setters  public long getPaymentId() {  return paymentId;  }  public void setPaymentId(long paymentId) {  this.paymentId = paymentId;  }  public long getCourierId() {  return courierId;  }  public void setCourierId(long courierId) {  this.courierId = courierId;  }  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;  }  // ToString Method  @Override  public String toString() {  return "Payment [PaymentID=" + paymentId +  ", CourierID=" + courierId +  ", Amount=" + amount +  ", PaymentDate=" + paymentDate + "]";  }  } |

**Task 6: Service Provider Interface /Abstract class**

**Create 2 Interface /Abstract class ICourierUserService and ICourierAdminService interface**

**ICourierUserService {**

**// Customer-related functions**

**placeOrder()**

**/\*\* Place a new courier order.**

**\* @param courierObj Courier object created using values entered by users**

**\* @return The unique tracking number for the courier order .**

**Use a static variable to generate unique tracking number. Initialize the static variable in Courier**

**class with some random value. Increment the static variable each time in the constructor to**

**generate next values.**

**getOrderStatus();**

**/\*\*Get the status of a courier order.**

**\*@param trackingNumber The tracking number of the courier order.**

**\* @return The status of the courier order (e.g., yetToTransit, In Transit, Delivered).**

**\*/**

**cancelOrder()**

**/\*\* Cancel a courier order.**

**\* @param trackingNumber The tracking number of the courier order to be canceled.**

**\* @return True if the order was successfully canceled, false otherwise.\*/**

**getAssignedOrder();**

**/\*\* Get a list of orders assigned to a specific courier staff member**

**\* @param courierStaffId The ID of the courier staff member.**

**\* @return A list of courier orders assigned to the staff member.\*/**

**// Admin functions**

|  |
| --- |
| package services;  import entities.Courier;  import exceptions.InvalidEmployeeIdException;  import exceptions.TrackingNumberNotFoundException;  public interface ICourierUserService {  String placeOrder(Courier courier) throws InvalidEmployeeIdException;  String getOrderStatus(String trackingNumber) throws TrackingNumberNotFoundException;  boolean cancelOrder(String trackingNumber) throws TrackingNumberNotFoundException;  String getAssignedOrder(String employeeId);  } |

**ICourierAdminService**

**int addCourierStaff(Employee obj);**

**/\*\* Add a new courier staff member to the system.**

**\* @param name The name of the courier staff member.**

**\* @param contactNumber The contact number of the courier staff member.**

**\* @return The ID of the newly added courier staff member.**

**\*/**

|  |
| --- |
| **package** services;  **public** **interface** ICourierAdminService {  String getAllCourierDetails();  } |

**Task 7: Exception Handling**

**(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catchfinally,thow & throws keyword usage)**

**Define the following custom exceptions and throw them in methods whenever needed . Handle all theexcpetionsin main method,**

**1. TrackingNumberNotFoundException :throw this exception when user try to withdraw amount or**

**transfer amount to another acco**

|  |
| --- |
| **package** exceptions;  **public** **class** TrackingNumberNotFoundException **extends** Exception {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **public** TrackingNumberNotFoundException(String message) {  **super**(message);  }  } |

**2. InvalidEmployeeIdException throw this exception when id entered for the employee not existing in the system**

|  |
| --- |
| **package** exceptions;  **public** **class** InvalidEmployeeIdException **extends** RuntimeException {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **public** InvalidEmployeeIdException(String message) {  **super**(message);  }  } |

**Task 8: Collections**

**Scope: ArrayList/Hashmap**

**Task: Improve the Courier Management System by using Java collections:**

**1. Create a new model named CourierCompanyCollection in entity package replacing the Array of**

**Objects with List to accommodate dynamic updates in the CourierCompany class**

|  |
| --- |
| package Main;  import dao.CourierServiceDb;  import java.util.\*;  public class Main {  public static void main(String[] args) {  CourierServiceDb dbService = new CourierServiceDb();    System.out.println("DELIVERY HISTORY for TRK1001:");  List<String> history = dbService.getDeliveryHistory("TRK1001");  if (history.isEmpty()) {  System.out.println("No delivery history found.");  } else {  for (String record : history) {  System.out.println(record);  }  }    System.out.println("\nSHIPMENT STATUS REPORT:");  Map<String, Integer> statusReport = dbService.getShipmentStatusReport();  if (statusReport.isEmpty()) {  System.out.println("No shipment data available.");  } else {  statusReport.forEach((status, count) -> System.out.println(status + ": " + count));  }    System.out.println("\nREVENUE REPORT:");  double revenue = dbService.getTotalRevenue();  System.out.println("Total Revenue: " + revenue);  }  } |

**2. Create a new implementation class CourierUserServiceCollectionImpl class in package dao which**

**implements ICourierUserService interface which holds a variable named companyObj of type**

**CourierCompanyCollection**

|  |
| --- |
| package dao;  import entities.Courier;  import entities.CourierCompanyCollection;  import exceptions.InvalidEmployeeIdException;  import exceptions.TrackingNumberNotFoundException;  import services.ICourierUserService;  import java.util.List;  public class CourierUserServiceCollectionImpl implements ICourierUserService {  protected CourierCompanyCollection companyObj = new CourierCompanyCollection();  @Override  public String placeOrder(Courier courier) {  if (!courier.getEmployeeId().startsWith("EMP")) {  throw new InvalidEmployeeIdException("Invalid Employee ID: " + courier.getEmployeeId());  }  companyObj.addCourier(courier);  return courier.getTrackingNumber();  }  @Override  public String getOrderStatus(String trackingNumber) throws TrackingNumberNotFoundException {  Courier c = companyObj.getCourierByTrackingNumber(trackingNumber);  if (c == null) {  throw new TrackingNumberNotFoundException("Tracking number not found: " + trackingNumber);  }  return c.getStatus();  }  @Override  public boolean cancelOrder(String trackingNumber) throws TrackingNumberNotFoundException {  Courier courier = companyObj.getCourierByTrackingNumber(trackingNumber);  if (courier == null) {  throw new TrackingNumberNotFoundException("Cannot cancel. Tracking number not found: " + trackingNumber);  }  if ("Cancelled".equalsIgnoreCase(courier.getStatus())) {  System.out.println("Order already cancelled.");  return false;  }  courier.setStatus("Cancelled");  return true;    }  @Override  public String getAssignedOrder(String employeeId) {  List<Courier> assignedCouriers = companyObj.getCouriersByEmployee(employeeId);  if (assignedCouriers.isEmpty()) {  return "No couriers assigned to employee ID: " + employeeId;  }  StringBuilder sb = new StringBuilder("Couriers assigned to " + employeeId + ":\n");  for (Courier c : assignedCouriers) {  sb.append("Tracking #: ").append(c.getTrackingNumber())  .append(", Status: ").append(c.getStatus()).append("\n");  }  return sb.toString();  }  } |

**Task 8: Service implementation**

**1.Create CourierUserServiceImpl class which implements ICourierUserService interface which**

**holds a variable named companyObj of type CourierCompany.**

**This variable can be used to access the Object Arrays to access data relevant in method**

**implementations.**

|  |
| --- |
| package services;  import entities.Courier;  import entities.CourierCompany;  import exceptions.InvalidEmployeeIdException;  import exceptions.TrackingNumberNotFoundException;  public class CourierUserServiceImpl implements ICourierUserService {  protected CourierCompany companyObj = new CourierCompany();  @Override  public String placeOrder(Courier courier) {  if (!courier.getEmployeeId().startsWith("EMP")) {  throw new InvalidEmployeeIdException("Invalid Employee ID: " + courier.getEmployeeId());  }  companyObj.addCourier(courier);  return courier.getTrackingNumber();  }  @Override  public String getOrderStatus(String trackingNumber) throws TrackingNumberNotFoundException {  Courier c = companyObj.findCourierByTrackingNumber(trackingNumber);  if (c == null) {  throw new TrackingNumberNotFoundException("Tracking number not found: " + trackingNumber);  }  return c.getStatus();  }  @Override  public boolean cancelOrder(String trackingNumber) throws TrackingNumberNotFoundException {  Courier c = companyObj.findCourierByTrackingNumber(trackingNumber);  if (c == null) {  throw new TrackingNumberNotFoundException("Cannot cancel. Tracking number not found: " + trackingNumber);  }  c.setStatus("Cancelled");  return true;  }  @Override  public String getAssignedOrder(String employeeId) {  Courier[] assigned = companyObj.getCouriersByEmployeeId(employeeId);  if (assigned.length == 0) return "No orders assigned.";  StringBuilder sb = new StringBuilder("Assigned Orders for ").append(employeeId).append(":\n");  for (Courier c : assigned) {  sb.append("Tracking #: ").append(c.getTrackingNumber())  .append(", Status: ").append(c.getStatus()).append("\n");  }  return sb.toString();  }  } |

**2. Create CourierAdminService Impl class which inherits from CourierUserServiceImpl and**

**implements ICourierAdminService interface.**

|  |
| --- |
| package services;  import entities.Courier;  public class CourierAdminServiceImpl extends CourierUserServiceImpl implements ICourierAdminService {  @Override  public String getAllCourierDetails() {  Courier[] allCouriers = companyObj.getAllCouriers();  StringBuilder sb = new StringBuilder("All Courier Details:\n");  for (Courier c : allCouriers) {  sb.append("Tracking #: ").append(c.getTrackingNumber())  .append(", Sender: ").append(c.getSenderName())  .append(", Receiver: ").append(c.getReceiverName())  .append(", Status: ").append(c.getStatus()).append("\n");  }  return sb.toString();  }  } |

**3. Create CourierAdminServiceCollectionImpl class which inherits from**

**CourierUserServiceColectionImpl and implements ICourierAdminService interface.**

|  |
| --- |
| package dao;  import entities.Courier;  import services.ICourierAdminService;  public class CourierAdminServiceCollection extends CourierUserServiceCollectionImpl implements ICourierAdminService {  @Override  public String getAllCourierDetails() {  StringBuilder sb = new StringBuilder("All Courier Details:\n");  for (Courier c : companyObj.getAllCouriers()) {  sb.append("Tracking #: ").append(c.getTrackingNumber())  .append(", Sender: ").append(c.getSenderName())  .append(", Receiver: ").append(c.getReceiverName())  .append(", Status: ").append(c.getStatus()).append("\n");  }  return sb.toString();  }  } |

**Task 9: Database Interaction**

**Connect your application to the SQL database for the Courier Management System**

**1. Write code to establish a connection to your SQL database.**

**Create a class DBConnection in a package connectionutil with a static variable connection of**

**Type Connection and a static method getConnection() which returns connection.**

**Connection properties supplied in the connection string should be read from a property file.**

|  |
| --- |
| package connectionutil;  import java.sql.Connection;  import java.sql.DriverManager;  public class DBConnection {  private static Connection connection;  public static Connection getConnection() {  try {  if (connection == null || connection.isClosed()) {  String url = "jdbc:mysql://localhost:3306/couriermanagement";  String username = "root";  String password = "cats";  String driver = "com.mysql.cj.jdbc.Driver";  Class.forName(driver); // Load the JDBC driver  connection = DriverManager.getConnection(url, username, password);  }  } catch (Exception e) {  e.printStackTrace();  }  return connection;  }  } |

|  |
| --- |
| host=localhost  port=3306  dbname=couriermanagement  username=root  password=cats |

**2. Create a Service class CourierServiceDb in dao with a static variable named connection of**

**type Connection which can be assigned in the constructor by invoking the method in**

**DBConnection Class.**

|  |
| --- |
| package dao;  import java.sql.Connection;  import connectionutil.DBConnection;  public class CourierServiceDb {    private static Connection connection;    public CourierServiceDb() {  connection = DBConnection.getConnection();  if (connection != null) {  System.out.println("Connection established successfully.");  } else {  System.out.println("Failed to establish connection.");  }  }  public boolean isConnected() {  return connection != null;  }  } |

**3. Include methods to insert, update, and retrieve data from the database (e.g., inserting a new**

**order, updating courier status).**

|  |
| --- |
| package dao;  import java.sql.\*;  import connectionutil.DBConnection;  import entities.Courier;  public class CourierServiceDb {  // Static connection variable  private static Connection connection;  // Constructor - assigns connection using DBConnection  public CourierServiceDb() {  connection = DBConnection.getConnection();  if (connection != null) {  System.out.println("Connection established successfully.");  } else {  System.out.println("Failed to establish connection.");  }  }  public boolean insertCourier(Courier courier) {  String sql = "INSERT INTO couriers (tracking\_number, sender\_name, receiver\_name, status, employee\_id) VALUES (?, ?, ?, ?, ?)";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, courier.getTrackingNumber());  ps.setString(2, courier.getSenderName());  ps.setString(3, courier.getReceiverName());  ps.setString(4, courier.getStatus());  ps.setString(5, courier.getEmployeeId());  return ps.executeUpdate() > 0;  } catch (SQLException e) {  System.out.println("Error inserting courier: " + e.getMessage());  return false;  }  }  public boolean updateCourierStatus(String trackingNumber, String newStatus) {  String sql = "UPDATE couriers SET status = ? WHERE tracking\_number = ?";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, newStatus);  ps.setString(2, trackingNumber);  return ps.executeUpdate() > 0;  } catch (SQLException e) {  System.out.println("Error updating status: " + e.getMessage());  return false;  }  }    public Courier getCourierByTrackingNumber(String trackingNumber) {  String sql = "SELECT \* FROM couriers WHERE tracking\_number = ?";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, trackingNumber);  ResultSet rs = ps.executeQuery();  if (rs.next()) {  return new Courier(  rs.getString("tracking\_number"),  rs.getString("sender\_name"),  rs.getString("receiver\_name"),  rs.getString("employee\_id"),  rs.getString("status")  );  }  } catch (SQLException e) {  System.out.println("Error retrieving courier: " + e.getMessage());  }  return null;  }  } |

**4. Implement a feature to retrieve and display the delivery history of a specific parcel by**

**querying the database. 1. Generate and display reports using data retrieved from the database**

**(e.g., shipment status report, revenue report).**

|  |
| --- |
| package dao;  import java.sql.\*;  import java.util.\*;  import connectionutil.DBConnection;  import entities.Courier;  public class CourierServiceDb {  private static Connection connection;    public CourierServiceDb() {  connection = DBConnection.getConnection();  if (connection != null) {  System.out.println("Database connection established in CourierServiceDb.");  } else {  System.out.println("Failed to establish database connection.");  }  }    public boolean insertCourier(Courier courier) {  String sql = "INSERT INTO couriers (tracking\_number, sender\_name, receiver\_name, status, employee\_id) VALUES (?, ?, ?, ?, ?)";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, courier.getTrackingNumber());  ps.setString(2, courier.getSenderName());  ps.setString(3, courier.getReceiverName());  ps.setString(4, courier.getStatus());  ps.setString(5, courier.getEmployeeId());  return ps.executeUpdate() > 0;  } catch (SQLException e) {  System.out.println("Error inserting courier: " + e.getMessage());  return false;  }  }    public boolean updateCourierStatus(String trackingNumber, String newStatus) {  String sql = "UPDATE couriers SET status = ? WHERE tracking\_number = ?";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, newStatus);  ps.setString(2, trackingNumber);  return ps.executeUpdate() > 0;  } catch (SQLException e) {  System.out.println("Error updating status: " + e.getMessage());  return false;  }  }    public Courier getCourierByTrackingNumber(String trackingNumber) {  String sql = "SELECT \* FROM couriers WHERE tracking\_number = ?";  try (PreparedStatement ps = connection.prepareStatement(sql)) {  ps.setString(1, trackingNumber);  ResultSet rs = ps.executeQuery();  if (rs.next()) {  return new Courier(  rs.getString("tracking\_number"),  rs.getString("sender\_name"),  rs.getString("receiver\_name"),  rs.getString("employee\_id"),  rs.getString("status")  );  }  } catch (SQLException e) {  System.out.println("Error retrieving courier: " + e.getMessage());  }  return null;  }  public List<String> getDeliveryHistory(String trackingNumber) {  List<String> history = new ArrayList<>();  String sql = "SELECT status, location, timestamp FROM delivery\_history WHERE tracking\_number = ? ORDER BY timestamp";  try (PreparedStatement pstmt = connection.prepareStatement(sql)) {  pstmt.setString(1, trackingNumber);  ResultSet rs = pstmt.executeQuery();  while (rs.next()) {  String status = rs.getString("status");  String location = rs.getString("location");  String time = rs.getTimestamp("timestamp").toString();  history.add("[" + time + "] " + status + " at " + location);  }  } catch (SQLException e) {  e.printStackTrace();  }  return history;  }    public Map<String, Integer> getShipmentStatusReport() {  Map<String, Integer> report = new HashMap<>();  String sql = "SELECT status, COUNT(\*) AS count FROM couriers GROUP BY status";  try (Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sql)) {  while (rs.next()) {  String status = rs.getString("status");  int count = rs.getInt("count");  report.put(status, count);  }  } catch (SQLException e) {  e.printStackTrace();  }  return report;  }    public double getTotalRevenue() {  double total = 0.0;  String sql = "SELECT SUM(amount) FROM payments";  try (Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sql)) {  if (rs.next()) {  total = rs.getDouble(1);  }  } catch (SQLException e) {  e.printStackTrace();  }  return total;  }  } |

|  |
| --- |
| **OUTPUT:**  Database connection established in CourierServiceDb.  DELIVERY HISTORY for TRK1001:  [2025-04-07 01:10:33.0] Shipped at Warehouse A  [2025-04-07 01:10:33.0] In Transit at City Hub  [2025-04-07 01:10:33.0] Out for Delivery at Zone 5  [2025-04-07 01:10:33.0] Delivered at Customer Address  SHIPMENT STATUS REPORT:  null: 3  Delivered: 1  REVENUE REPORT:  Total Revenue: 950.0 |