ASSIGNMENT-4 (SQL TASK)

TASK-1:

CREATION:

```
-- User Table
CREATE TABLE User (
   UserID INT PRIMARY KEY,
   Name VARCHAR(255),
   Email VARCHAR(255) UNIQUE,
   Password VARCHAR(255),
   ContactNumber VARCHAR(20),
   Address TEXT
);

-- CourierServices Table
CREATE TABLE CourierServices (
   ServiceID INT PRIMARY KEY,
   ServiceName VARCHAR(100),
   Cost DECIMAL(8, 2)
);

-- Employee Table
CREATE TABLE Employee (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(255),
   Email VARCHAR(255) UNIQUE,
   ContactNumber VARCHAR(20),
   Role VARCHAR(50),
   Salary DECIMAL(10, 2)
```

```sql
);

-- Location Table
CREATE TABLE Location (
    LocationID INT PRIMARY KEY,
    LocationName VARCHAR(100),
    Address TEXT
);

-- Courier Table
CREATE TABLE Courier (
    CourierID INT PRIMARY KEY,
    SenderName VARCHAR(255),
    SenderAddress TEXT,
    ReceiverName VARCHAR(255),
    ReceiverAddress TEXT,
    Weight DECIMAL(5, 2),
    Status VARCHAR(50),
    TrackingNumber VARCHAR(20) UNIQUE,
    DeliveryDate DATE,
    ServiceID INT,
    EmployeeID INT,
    FOREIGN KEY (ServiceID) REFERENCES CourierServices(ServiceID),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);

-- Payment Table
CREATE TABLE Payment (
    PaymentID INT PRIMARY KEY,
    CourierID INT,
    LocationID INT,
    Amount DECIMAL(10, 2),
    PaymentDate DATE,
    FOREIGN KEY (CourierID) REFERENCES Courier(CourierID),
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);
```

INSERTION:

```
--
INSERT INTO User (UserID, Name, Email, Password, ContactNumber, Address)
VALUES
(1, 'Alice Johnson', 'alice@example.com', 'password123', '1234567890', '123 Main St'),
(2, 'Bob Smith', 'bob@example.com', 'password456', '0987654321', '456 Elm St');
--
INSERT INTO CourierServices (ServiceID, ServiceName, Cost)
VALUES
(1, 'Standard Delivery', 50.00),
(2, 'Express Delivery', 100.00);
--
INSERT INTO Employee (EmployeeID, Name, Email, ContactNumber, Role, Salary)
VALUES
(1, 'John Doe', 'john@example.com', '1112223333', 'Delivery Agent', 30000.00),
(2, 'Jane Smith', 'jane@example.com', '4445556666', 'Manager', 50000.00);


--
INSERT INTO Location (LocationID, LocationName, Address)
```

VALUES

(1, 'Warehouse A', '789 Oak St'),

(2, 'Warehouse B', '321 Pine St');

--

INSERT INTO Courier (CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, ServiceID, EmployeeID)

VALUES

(1, 'Alice Johnson', '123 Main St', 'Charlie Brown', '987 Maple St', 2.5, 'Delivered', 'TRK123456', '2025-04-15', 1, 1),

(2, 'Bob Smith', '456 Elm St', 'Daisy Miller', '654 Spruce St', 1.2, 'In Transit', 'TRK654321', '2025-04-17', 2, 1);

--

INSERT INTO Payment (PaymentID, CourierID, LocationID, Amount, PaymentDate)

VALUES

(1, 1, 1, 50.00, '2025-04-15'),

(2, 2, 2, 100.00, '2025-04-16');

Task 2: SELECT and WHERE Queries

1)List all customers:

SELECT * FROM User;

2) List all orders for a specific customer:

SELECT * FROM Courier WHERE SenderName = 'Alice Johnson';

3) List all couriers:

SELECT * FROM Courier;

4) List all packages for a specific order:

SELECT * FROM Courier WHERE CourierID = 1;

5) List all deliveries for a specific courier:

SELECT * FROM Courier WHERE TrackingNumber = 'TRK123456';

6) List all undelivered packages:

```
SELECT * FROM Courier WHERE Status != 'Delivered';
```

7) List all packages that are scheduled for delivery today:

```
SELECT * FROM Courier WHERE DeliveryDate = CURRENT_DATE;
```

8) List all packages with a specific status:

```
SELECT * FROM Courier WHERE Status = 'In Transit';
```

9)Calculate the total number of packages for each courier:

```
SELECT EmployeeID, COUNT(*) AS TotalPackages

FROM Courier

GROUP BY EmployeeID;
```

10) Find the average delivery time for each courier:

```
SELECT EmployeeID, AVG(DATEDIFF(DeliveryDate, DispatchDate)) AS AvgDeliveryTime

FROM Courier

GROUP BY EmployeeID;
```

11) List all packages with a specific weight range:

```
SELECT * FROM Courier WHERE Weight BETWEEN 1.0 AND 3.0;
```

12) Retrieve employees whose names contain 'John':

```
SELECT * FROM Employee WHERE Name LIKE '%John%';
```

13) Retrieve all courier records with payments greater than $50:

```
SELECT c.*

FROM Courier c

JOIN Payment p ON c.CourierID = p.CourierID

WHERE p.Amount > 50.00;
```

Task 3: GroupBy, Aggregate Functions, Having, Order By, where

14) Find the total number of couriers handled by each employee:

```
SELECT EmployeeID, COUNT(*) AS TotalCouriers

FROM Courier

GROUP BY EmployeeID;
```

15) Calculate the total revenue generated by each location:

```
SELECT LocationID, SUM(Amount) AS TotalRevenue

FROM Payment

GROUP BY LocationID;
```

16) Find the total number of couriers delivered to each location:

```
SELECT ReceiverAddress, COUNT(*) AS TotalCouriers

FROM Courier

GROUP BY ReceiverAddress;
```

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

```
SELECT CourierID, AVG(DATEDIFF(DeliveryDate, DispatchDate)) AS AvgDeliveryTime

FROM Courier

GROUP BY CourierID

ORDER BY AvgDeliveryTime DESC

LIMIT 1;
```

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

```
SELECT LocationID, SUM(Amount) AS TotalPayments

FROM Payment

GROUP BY LocationID
```

```
HAVING SUM(Amount) < 1000;
```

19) Calculate Total Payments per Location:

```
SELECT LocationID, SUM(Amount) AS TotalPayments

FROM Payment

GROUP BY LocationID;
```

20) Couriers who received payments > $1000 in a specific location:

```
SELECT CourierID, SUM(Amount) AS Total

FROM Payment

WHERE LocationID = X

GROUP BY CourierID

HAVING SUM(Amount) > 1000;
```

21) Couriers with payments > $1000 after a specific date:

```
SELECT CourierID, SUM(Amount) AS Total

FROM Payment

WHERE PaymentDate > '2025-01-01'

GROUP BY CourierID

HAVING SUM(Amount) > 1000;
```

22) Locations with total amount > $5000 before a date:

```
SELECT LocationID, SUM(Amount) AS Total

FROM Payment

WHERE PaymentDate < '2025-01-01'

GROUP BY LocationID

HAVING SUM(Amount) > 5000;
```

Task 4: JOINs

23) Retrieve Payments with Courier Information:

```
SELECT p.*, c.*

FROM Payment p

JOIN Courier c ON p.CourierID = c.CourierID;
```

24) Payments with Location Info:

```
SELECT p.*, l.*

FROM Payment p

JOIN Location l ON p.LocationID = l.LocationID;
```

25. Payments with Courier and Location Info:

```
SELECT p.*, c.*, l.*

FROM Payment p

JOIN Courier c ON p.CourierID = c.CourierID

JOIN Location l ON p.LocationID = l.LocationID;
```

26. List all payments with courier details:

```
SELECT p.*, c.*

FROM Payment p

JOIN Courier c ON p.CourierID = c.CourierID;
```

27. Total payments received for each courier:

```
SELECT CourierID, SUM(Amount) AS TotalReceived

FROM Payment

GROUP BY CourierID;
```

28. Payments on a specific date:

```
SELECT * FROM Payment

WHERE PaymentDate = '2025-04-15';
```

29. Courier Info for Each Payment:

```
SELECT p.PaymentID, c.*

FROM Payment p

JOIN Courier c ON p.CourierID = c.CourierID;
```

30. Payment Details with Location:

```
SELECT p.*, l.LocationName

FROM Payment p

JOIN Location l ON p.LocationID = l.LocationID;
```

31. Total Payments for Each Courier:

```
SELECT CourierID, SUM(Amount) AS TotalPayment

FROM Payment

GROUP BY CourierID;
```

32. Payments Within Date Range:

```
SELECT * FROM Payment

WHERE PaymentDate BETWEEN '2025-04-01' AND '2025-04-17';
```

33. All Users and Their Couriers:

```
SELECT u.*, c.*

FROM User u

LEFT JOIN Courier c ON u.Name = c.SenderName

UNION

SELECT u.*, c.*
```

```
FROM User u

RIGHT JOIN Courier c ON u.Name = c.SenderName;
```

34. All Couriers and Their Services:

```
SELECT c.*, s.*

FROM Courier c

LEFT JOIN CourierServices s ON c.ServiceID = s.ServiceID

UNION

SELECT c.*, s.*

FROM Courier c

RIGHT JOIN CourierServices s ON c.ServiceID = s.ServiceID;
```

35. Employees and Their Payments:

```
SELECT e.*, p.*

FROM Employee e

LEFT JOIN Courier c ON e.EmployeeID = c.EmployeeID

LEFT JOIN Payment p ON c.CourierID = p.CourierID;
```

36. All Users and All Courier Services:

```
SELECT u.*, s.*

FROM User u

CROSS JOIN CourierServices s;
```

37. Employees and All Locations:

```
SELECT e.*, l.*

FROM Employee e

CROSS JOIN Location l;
```

38. Couriers and Sender Info:

```
SELECT c.*, u.*

FROM Courier c

LEFT JOIN User u ON c.SenderName = u.Name;
```

39. Couriers and Receiver Info:

```
SELECT c.*, u.*

FROM Courier c

LEFT JOIN User u ON c.ReceiverName = u.Name;
```

40. Couriers with Service Details:

```
SELECT c.*, s.*

FROM Courier c

LEFT JOIN CourierServices s ON c.ServiceID = s.ServiceID;
```

41. Employees and Courier Count:

```
SELECT e.EmployeeID, e.Name, COUNT(c.CourierID) AS NumCouriers

FROM Employee e

LEFT JOIN Courier c ON e.EmployeeID = c.EmployeeID

GROUP BY e.EmployeeID, e.Name;
```

42. Locations and Total Payments:

```
SELECT l.LocationID, l.LocationName, SUM(p.Amount) AS TotalAmount

FROM Location l

LEFT JOIN Payment p ON l.LocationID = p.LocationID

GROUP BY l.LocationID, l.LocationName;
```

43. Couriers from Same Sender:

```
SELECT * FROM Courier

WHERE SenderName = 'Alice Johnson';
```

44. Employees with Same Role:

```
SELECT * FROM Employee

WHERE Role IN (

   SELECT Role FROM Employee

   GROUP BY Role

   HAVING COUNT(*) > 1

);
```

45. Payments for Couriers from Same Location:

```
SELECT p.*

FROM Payment p

JOIN Courier c ON p.CourierID = c.CourierID

WHERE c.SenderAddress = '123 Main St';
```

46. Couriers Sent from Same Location:

```
SELECT * FROM Courier

WHERE SenderAddress = '123 Main St';
```

47. Employees and Couriers Delivered:

```
SELECT e.EmployeeID, e.Name, COUNT(c.CourierID) AS DeliveredCount

FROM Employee e

JOIN Courier c ON e.EmployeeID = c.EmployeeID

WHERE c.Status = 'Delivered'

GROUP BY e.EmployeeID, e.Name;
```

48. Couriers Paid More Than Service Cost:

```
SELECT c.CourierID, SUM(p.Amount) AS Paid, s.Cost

FROM Courier c

JOIN Payment p ON c.CourierID = p.CourierID

JOIN CourierServices s ON c.ServiceID = s.ServiceID

GROUP BY c.CourierID, s.Cost

HAVING SUM(p.Amount) > s.Cost;
```

Task 5: Subqueries and Advanced Logic

49. Couriers Heavier than Average Weight:

```
SELECT * FROM Courier

WHERE Weight > (SELECT AVG(Weight) FROM Courier);
```

50. Employees with Salary > Average:

```
SELECT * FROM Employee

WHERE Salary > (SELECT AVG(Salary) FROM Employee);
```

51. Total Cost of Services Below Max Cost:

```
SELECT SUM(Cost) AS Total

FROM CourierServices

WHERE Cost < (SELECT MAX(Cost) FROM CourierServices);
```

52. Couriers That Have Been Paid For:

```
SELECT DISTINCT c.*

FROM Courier c

JOIN Payment p ON c.CourierID = p.CourierID;
```

53. Locations with Max Payment:

```sql
SELECT LocationID, Amount

FROM Payment

WHERE Amount = (SELECT MAX(Amount) FROM Payment);
```

54. Couriers Heavier Than All from Specific Sender:

```sql
SELECT * FROM Courier

WHERE Weight > ALL (

   SELECT Weight FROM Courier

   WHERE SenderName = 'Alice Johnson'

);
```

PYTHON :

Task 1: Control Flow Statements

1. **Check order delivery status:**

```python
status = input("Enter order status (Processing, Delivered, Cancelled): ")


if status == "Delivered":
   print("Order has been delivered.")
elif status == "Processing":
   print("Order is still processing.")
elif status == "Cancelled":
   print("Order has been cancelled.")
else:
   print("Unknown status.")
```

2. **Switch-case weight categorization:**

```python
weight = float(input("Enter parcel weight (kg): "))



match weight:
    case w if w <= 2:
        print("Light")
    case w if 2 < w <= 5:
        print("Medium")
    case _:
        print("Heavy")
```

3. **User Authentication:**

```
def login(user_type):
    username = input("Enter username: ")
    password = input("Enter password: ")

    if user_type == "employee":
        if username == "emp123" and password == "emp@pass":
            print("Employee logged in.")
        else:
            print("Invalid employee credentials.")
    elif user_type == "customer":
        if username == "cust456" and password == "cust@pass":
            print("Customer logged in.")
        else:
            print("Invalid customer credentials.")

login("customer")  # or login("employee")
```

4. **Courier Assignment Logic:**

```
couriers = [
    {"name": "John", "location": "Downtown", "load": 3},
    {"name": "Sarah", "location": "Uptown", "load": 1},
    {"name": "Mike", "location": "Midtown", "load": 2}
]
```

```
def assign_courier(required_location):
    for courier in couriers:
        if courier["location"] == required_location and courier["load"] < 5:
            print(f"Assigned courier: {courier['name']}")
            courier["load"] += 1
            return
    print("No available courier found.")

assign_courier("Uptown")
```

Task 2: Loops

5. **For loop to show orders:**

```
orders = {
    "cust1": ["order001", "order002"],
    "cust2": ["order003"]
}
customer_id = "cust1"
print(f"Orders for {customer_id}:")
for order in orders.get(customer_id, []):
    print(order)
```

6. **While loop for tracking:**

```
location_updates = ["Warehouse", "City Hub", "On the Way", "Delivered"]
i = 0
while i < len(location_updates):
    print(f"Current location: {location_updates[i]}")
    if location_updates[i] == "Delivered":
        break
    i += 1
```

Task 3: Arrays and Data Structures

7. **Parcel tracking history (array):**

```
tracking_history = ["Dispatched", "Arrived at hub", "Out for delivery", "Delivered"]
print("Tracking History:")
```

```
for status in tracking_history:
    print(status)
```

8. **Nearest courier logic:**

```
couriers = [
    {"name": "John", "distance": 10},
    {"name": "Jane", "distance": 5},
    {"name": "Rick", "distance": 8}
]

nearest = min(couriers, key=lambda x: x["distance"])
print(f"Nearest courier: {nearest['name']}, Distance: {nearest['distance']}km")
```

Task 4: Strings, 2D Arrays, User-Defined Functions, HashMap

9. **Parcel Tracking using 2D String Array:**

```
parcels = [
    ["TRK001", "In Transit"],
    ["TRK002", "Out for Delivery"],
    ["TRK003", "Delivered"]
]

def track_parcel(tracking_number):
    for parcel in parcels:
        if parcel[0] == tracking_number:
            status = parcel[1]
            print(f"Tracking Number: {tracking_number} - Status: {status}")
            return
    print("Parcel not found.")

track_parcel("TRK002")
```

10. **Customer Data Validation:**

```
import re

def validate(data, detail):
    if detail == "name":
        return data.isalpha() and data.istitle()
    elif detail == "address":
        return all(c.isalnum() or c.isspace() for c in data)
    elif detail == "phone":
        return re.match(r"^\d{3}-\d{3}-\d{4}$", data) is not None
    return False

# Test
print(validate("John", "name"))         # True
print(validate("123 Main St", "address")) # True
print(validate("987-654-3210", "phone"))  # True
```

11. Address Formatting:

```
def format_address(street, city, state, zip_code):
    formatted = f"{street.title()}, {city.title()}, {state.upper()} - {zip_code}"
    return formatted

# Test
print(format_address("123 main street", "boston", "ma", "02115"))
```

12. Order Confirmation Email Generator:

```
def generate_email(name, order_num, address, delivery_date):
    print(f"""
    Hello {name},

    Your order #{order_num} has been confirmed!
    Delivery Address: {address}
    Expected Delivery Date: {delivery_date}

    Thank you for choosing our courier service.
    """)

generate_email("Alice", "ORD123", "123 Main St, Boston, MA", "2025-04-20")
```

13. Calculate Shipping Cost:

```
def calculate_shipping_cost(weight_kg, distance_km):
    cost_per_km = 0.5
    cost_per_kg = 1.0
    return (weight_kg * cost_per_kg) + (distance_km * cost_per_km)

print("Shipping Cost: $", calculate_shipping_cost(3.5, 20))
```

14. Secure Password Generator:

```
 import random

import string


def generate_password(length=12):

   chars = string.ascii_letters + string.digits + "!@#$%^&*()"

   return ''.join(random.choice(chars) for _ in range(length))


print("Generated Password:", generate_password())
```

15. Find Similar Addresses:

```
from difflib import SequenceMatcher

addresses = [
    "123 Main Street",
    "124 Main St.",
    "123 Maine Street",
    "99 Elm Avenue"
]

def find_similar(input_addr):
    print(f"Similar addresses to '{input_addr}':")
    for addr in addresses:
        similarity = SequenceMatcher(None, input_addr.lower(), addr.lower()).ratio()
        if similarity > 0.7 and addr != input_addr:
            print(f"- {addr} (Similarity: {round(similarity, 2)})")

find_similar("123 Main Street")
```

Task 5: OOP Model/Entity Classes

1. **User Class:**

```
class User:
    def __init__(self, userID, userName, email, password, contactNumber,
address):
        self.__userID = userID
        self.__userName = userName
        self.__email = email
        self.__password = password
        self.__contactNumber = contactNumber
        self.__address = address

    def __str__(self):
        return f"User[{self.__userID}] - {self.__userName}, {self.__email}"

    # Getters & Setters
    def get_userID(self): return self.__userID
    def set_userID(self, uid): self.__userID = uid
    # Add others similarly if needed
```

2. **Courier Class:**

```
class Courier:
    tracking_number_seed = 1000  # Static tracking number generator
    def __init__(self, courierID, senderName, senderAddress, receiverName,
receiverAddress, weight, status, deliveryDate, userId):
        self.__courierID = courierID
        self.__senderName = senderName
        self.__senderAddress = senderAddress
        self.__receiverName = receiverName
        self.__receiverAddress = receiverAddress
        self.__weight = weight
        self.__status = status
        self.__trackingNumber = f"TRK{Courier.tracking_number_seed}"
        self.__deliveryDate = deliveryDate
        self.__userId = userId
        Courier.tracking_number_seed += 1

    def __str__(self):
        return f"Courier[{self.__courierID}] - {self.__trackingNumber}, Status:
{self.__status}"
```

3. **Employee Class:**

```
class Employee:
    def __init__(self, employeeID, employeeName, email, contactNumber, role,
salary):
        self.__employeeID = employeeID
        self.__employeeName = employeeName
        self.__email = email
        self.__contactNumber = contactNumber
        self.__role = role
        self.__salary = salary

    def __str__(self):
        return f"Employee[{self.__employeeID}] - {self.__employeeName},
{self.__role}"
```

4. **Location Class:**

```
class Location:
    def __init__(self, locationID, locationName, address):
        self.__locationID = locationID
        self.__locationName = locationName
        self.__address = address

    def __str__(self):
        return f"Location[{self.__locationID}] - {self.__locationName}"
```

5. **CourierCompany Class:**

```
class CourierCompany:
    def __init__(self, companyName):
        self.__companyName = companyName
        self.__courierDetails = []
        self.__employeeDetails = []
        self.__locationDetails = []

    def __str__(self):
        return f"CourierCompany: {self.__companyName}, Couriers:
{len(self.__courierDetails)}, Employees: {len(self.__employeeDetails)}"
```

6. **Payment Class:**

```
from datetime import date

class Payment:
    def __init__(self, paymentID, courierID, amount, paymentDate: date):
        self.__paymentID = paymentID
        self.__courierID = courierID
        self.__amount = amount
        self.__paymentDate = paymentDate

    def __str__(self):
        return f"Payment[{self.__paymentID}] - Courier: {self.__courierID}, Amount:
${self.__amount}"
```

Task 6: Service Provider Interface / Abstract Class:

ICourierUserService:

```
from abc import ABC, abstractmethod

class ICourierUserService(ABC):

    @abstractmethod
    def placeOrder(self, courierObj):
        """Place a new courier order and return tracking number"""
        pass

    @abstractmethod
    def getOrderStatus(self, trackingNumber):
        """Get the status of a courier order"""
        pass

    @abstractmethod
    def cancelOrder(self, trackingNumber):
        """Cancel the order using tracking number"""
        pass

    @abstractmethod
    def getAssignedOrder(self, courierStaffId):
        """Get orders assigned to a specific courier staff member"""
        Pass
```

IcourierAdminService:

```
class ICourierAdminService(ABC):

    @abstractmethod
    def addCourierStaff(self, employeeObj):
        """Add a new courier staff member to the system"""
        Pass
```

Task 7: Custom Exceptions:

1. **TrackingNumberNotFoundException:**

```
class TrackingNumberNotFoundException(Exception):
    def __init__(self, tracking_number):
        super().__init__(f"Tracking number {tracking_number} not found.")
```

2. **InvalidEmployeeIdException:**

```
class InvalidEmployeeIdException(Exception):
    def __init__(self, employee_id):
        super().__init__(f"Employee ID {employee_id} is invalid.")
```

Task 8: Collections:

1. **Create a model: CourierCompanyCollection:**

```
class CourierCompanyCollection:
    def __init__(self, company_name):
        self.company_name = company_name
        self.courier_details = []     # List of Courier objects
        self.employee_details = []    # List of Employee objects
        self.location_details = []    # List of Location objects

    def __str__(self):
        return f"CourierCompany({self.company_name})"
```

2. **Implementation class: CourierUserServiceCollectionImpl:**

```
class CourierUserServiceCollectionImpl:
    def __init__(self, company_obj):
        self.company_obj = company_obj
        self.tracking_number_seq = 1000  # Simulating static tracking number

    def place_order(self, courier_obj):
        self.tracking_number_seq += 1
        courier_obj.tracking_number = f"TRK{self.tracking_number_seq}"
        self.company_obj.courier_details.append(courier_obj)
        return courier_obj.tracking_number

    def get_order_status(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.tracking_number == tracking_number:
                return courier.status
        raise TrackingNumberNotFoundException(tracking_number)

    def cancel_order(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.tracking_number == tracking_number:
                courier.status = "Cancelled"
                return True
        raise TrackingNumberNotFoundException(tracking_number)
```

Service Implementation:

```
class CourierUserServiceImpl:
    def __init__(self, company_obj):
        self.company_obj = company_obj
        self.tracking_number_seq = 1000

    def place_order(self, courier_obj):
        self.tracking_number_seq += 1
        courier_obj.tracking_number = f"TRK{self.tracking_number_seq}"
        self.company_obj.courier_details.append(courier_obj)
        return courier_obj.tracking_number

    def get_order_status(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.tracking_number == tracking_number:
                return courier.status
        raise TrackingNumberNotFoundException(tracking_number)
    def cancel_order(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.tracking_number == tracking_number:
                courier.status = "Cancelled"
                return True
        raise TrackingNumberNotFoundException(tracking_number)

    def get_assigned_order(self, courier_staff_id):
        return [c for c in self.company_obj.courier_details if c.user_id ==
courier_staff_id]
```

3. **CourierAdminServiceImpl Class:**

```
class CourierAdminServiceImpl(CourierUserServiceImpl):
    def add_courier_staff(self, employee_obj):
        self.company_obj.employee_details.append(employee_obj)
        return employee_obj.employeeID
```