Courier Management System

Project Title: Courier Management System
Submitted By: C.S. Deepta
Date: 04/04/2025
Trainer: Mrs.Karthika Madan
Organization: Hexaware

TABLE OF CONTENTS

PURPOSE OF THE PROJECT

The purpose of the **Courier Management System** project is to design and implement a system that efficiently handles courier-related operations such as order placement, courier tracking, payment processing, and employee management. The system aims to provide a streamlined solution for managing couriers, deliveries, and customers, ensuring smooth and accurate transactions.

Key objectives include:

- **Database Design**: Structuring an SQL schema to manage entities like users, couriers, employees, locations, and payments, and defining their relationships.
- **Data Retrieval**: Implementing SQL queries to manage and retrieve data on couriers, orders, customers, payments, and employees.
- **Control Flow & Logic**: Using control flow statements and loops in Python to handle courier statuses, package categorization, authentication, and assignment of tasks.
- **Object-Oriented Design**: Leveraging object-oriented principles in Python to structure entities like users, couriers, employees, and payments, promoting code reusability and maintainability.
- **Exception Handling**: Ensuring smooth operation of the system through proper exception handling, including user-defined exceptions and standard Python exceptions.
- **Collections**: Enhancing the system with Python collections such as lists, dictionaries, and sets for dynamic data handling and updates.
- **Service Provider Implementation**: Building interfaces and implementation classes in Python to model customer and admin services related to courier management.
- **Database Interaction**: Integrating database operations to handle CRUD (Create, Read, Update, Delete) operations efficiently within the system.

Overall, the project is designed to create a comprehensive and reliable Courier Management System using Python, SQL, and Object-Oriented Programming principles to facilitate real-time courier tracking, order management, and customer interaction.

SCOPE OF THE PROJECT

Overview:

The Courier Management System (CMS) is designed to streamline the process of managing couriers, deliveries, orders, customers, and payments within a courier company. The system will facilitate functionalities like user authentication, courier order placements, package tracking, courier assignment, payments management, and report generation. The project will use Python for its implementation, along with SQL for database interactions.

Modules & Structure:

The system consists of multiple modules, each focusing on specific operations such as managing users, couriers, orders, payments, and employees. The main components are described below.

1. Database Design:

I. Entities:

- **Users:** Users can be customers or employees.
- **Couriers:** Represents the delivery details for each shipment.
- **Orders:** Tracks customer orders and their statuses.
- **Parcels:** Information about the individual parcels within an order.
- **Employees:** Employees responsible for managing couriers and shipments.
- **Locations:** Locations involved in the courier process (sender/receiver addresses, etc.)
- **Payments:** Details of the payment transactions for the couriers.

II. Relationships:

- A **User** can place many **Orders**.
- An **Order** can have many **Parcels**.
- A **Courier** corresponds to an **Order** and has multiple **Parcels**.
- An **Employee** is responsible for **Couriers**.
- **Payments** are linked to specific **Couriers** and **Locations**.

III SQL Tables & Schema:

- The database consists of tables which has rows and columns
- Foreign keys and relationships (one-to-many, many-to-many) will be defined to establish connections between the tables.

2. Python Program Structure:

I. User Authentication & Session Management:

- **Login System:** Allows users to authenticate themselves as customers or employees, ensuring proper access rights.
- **User Validation:** Input validation for user data (name, contact number, etc.) using regular expressions.

II. Courier and Order Management:

- **Order Placement:** Customers can place orders by specifying courier details like sender, receiver, weight, and delivery date.

- **Tracking:** Customers can track the status of their orders using a unique tracking number.
- **Courier Assignment:** Admin can assign couriers to specific shipments based on predefined criteria such as weight, proximity, or load capacity.

III. Payment Management:

- **Payment Recording:** Payments for shipments are tracked, and details are stored in the database.
- **Revenue Reports:** The system will generate reports such as total revenue by location or payment status for each courier.

IV. Employee Management:

- **Employee Details:** The system stores employee details such as name, role, contact number, and salary.
- **Courier Assignment to Employees:** Employees are assigned couriers based on their role and load capacity.

3. Functionalities:

Task 1: Control Flow Statements

- **Order Delivery Status:** Check if an order is delivered based on its status (Delivered, Cancelled, etc.).
- **Parcel Weight Classification:** Categorize parcels as "Light," "Medium," or "Heavy" based on their weight.

Task 2: Loops & Iteration

- **Order List for Customer:** Display all orders placed by a specific customer using a loop.
- **Real-time Courier Location:** Track the location of a courier using a loop until the destination is reached.

Task 3: Arrays & Data Structures

- **Tracking History:** Use a list to store parcel tracking updates (location, status).
- **Nearest Available Courier:** Find the nearest available courier using a list of courier objects.

Task 4: String Operations and Validation

- **Order Confirmation Email:** Generate an order confirmation email with details like the customer's name, delivery address, and expected delivery date.
- **Shipping Cost Calculation:** Calculate the shipping cost based on the distance between two locations and the weight of the parcel.

Task 5: Object-Oriented Programming (OOP)

- **Entities as Classes:** Define classes for **User**, **Courier**, **Employee**, **Location**, **Payment**, and **CourierCompany** using OOP principles (encapsulation, abstraction).
- **Methods for Functionalities:** Implement methods such as placing an order, tracking a parcel, updating courier status, and generating reports.

Task 6: Service Provider Interfaces (Abstract Classes)

- **ICourierUserService and ICourierAdminService:** Define interfaces for customer-related and admin-related functionalities. Implement these interfaces in respective service classes.

Task 7: Exception Handling

- **Custom Exceptions:** Implement custom exceptions like **TrackingNumberNotFoundException** and **InvalidEmployeeIdException** for specific error handling scenarios.

Task 8: Collections and Data Management

- **Collection Management:** Use Python's list and dict for managing courier details and orders.
- **Courier Assignment & Tracking:** Implement collection-based methods to dynamically assign and track couriers.

Task 9: Service Implementation

- **CourierUserServiceImpl:** Create the CourierUserServiceImpl class that implements the ICourierUserService interface, managing courier and order data through a companyObj variable.
- **CourierAdminService Implementation:** Extend the CourierUserServiceImpl class to create CourierAdminServiceImpl and CourierAdminServiceCollectionImpl, implementing admin-level functionalities like courier management and order tracking.

Task 10: Database Interaction

- **DBConnection Class:** ImplementI the DBConnection class to manage database connections, with connection properties loaded from a properties file.
- **CourierServiceDb Class:** Create the CourierServiceDb class in the dao package to interact with the database, supporting data insertion, updates, and retrieval of courier and order information.

Technologies Used for the Project

1. **MySQL:** Used as the relational database management system to store and manage courier details, orders, and tracking information. SQL queries are employed to insert, update, and retrieve data from the database.
2. **PyCharm:** The Integrated Development Environment (IDE) used for Python development. It is utilized for writing, debugging, and testing the Python code for the Courier Management System.
3. **GitHub:** A platform for version control and collaboration. GitHub is used to manage the project's source code, track changes, and collaborate with team members, ensuring efficient version control throughout the development process.

PART -1 SQL

Task 1: Database Design for Courier Management System

Objective:

The goal of this task is to design a relational database schema for the Courier Management System (CMS). The schema should include tables for core entities such as Customers, Couriers, Orders, Parcels, and other related entities. We will define the relationships between these tables using foreign keys and populate the tables with sample data to simulate real-world scenarios.

Step 1: Creating Database

To begin, we will create a database called CouriersManagementSystem in MySQL. This will serve as the container for all the tables related to the CMS.

```
CREATE DATABASE CouriersManagementSystem;
USE CouriersManagementSystem;
```

Step 2 : Creating Table

- **User Table:** The User table stores information about the users of the courier management system, which includes both customers and employees interacting with the system. Each user has a unique identifier (UserID), name, email, password, contact number, and an address. The email field is unique to ensure that there are no duplicate accounts in the system. This table plays a key role in identifying and authenticating users.

  Attributes:

  - UserID: The primary key, uniquely identifying each user.
  - Name: The name of the user.
  - Email: The user's email, which is unique for each user.
  - Password: The user's password for login authentication.
  - ContactNumber: The user's contact number.
  - Address: The user's address.

```
CREATE TABLE User (
UserID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(255) NOT NULL,
Email VARCHAR(255) UNIQUE NOT NULL,
Password VARCHAR(255) NOT NULL,
ContactNumber VARCHAR(20),
Address TEXT
);
```

- **CourierServices Table:** The CourierServices table contains a list of services offered by the courier company. Each service has a unique ID, a name (such as "Same Day Delivery," "Next Day Express," etc.), and a cost associated with it. This table allows the system to manage different types of services offered and their pricing.

Attributes:

- ServiceID: The unique identifier for each service.
- ServiceName: The name of the courier service.
- Cost: The cost of the service.

```
CREATE TABLE CourierServices (
ServiceID INT PRIMARY KEY AUTO_INCREMENT,
ServiceName VARCHAR(100) NOT NULL,
Cost DECIMAL(8, 2) NOT NULL
);
```

- **Courier Table:** The Courier table stores details of the parcels being shipped, including both the sender's and receiver's information, as well as the parcel's weight, current status, and a unique tracking number for tracking purposes. This table helps track the journey of a courier shipment, from pick-up to delivery.

Attributes:

- CourierID: A unique identifier for each courier.
- SenderName and ReceiverName: The names of the sender and receiver.
- SenderAddress and ReceiverAddress: The addresses of the sender and receiver.
- Weight: The weight of the parcel being shipped.
- Status: The current status of the courier (e.g., in transit, delivered).
- TrackingNumber: A unique identifier used to track the courier.
- DeliveryDate: The date when the courier is expected to be delivered.

```
CREATE TABLE Courier (
CourierID INT PRIMARY KEY AUTO_INCREMENT,
SenderName VARCHAR(255) NOT NULL,
SenderAddress TEXT NOT NULL,
ReceiverName VARCHAR(255) NOT NULL,
ReceiverAddress TEXT NOT NULL,
Weight DECIMAL(5, 2) NOT NULL,
Status VARCHAR(50) DEFAULT 'Pending',
TrackingNumber VARCHAR(20) UNIQUE NOT NULL,
DeliveryDate DATE,
UserID INT,
ServiceID INT,
FOREIGN KEY (UserID) REFERENCES User(UserID),
FOREIGN KEY (ServiceID) REFERENCES CourierServices(ServiceID)
);
```

- **Employee Table:** The Employee table stores information about the employees working in the courier company. It contains details such as the employee's ID, name, contact information, role, and salary. This table helps to manage and track employee data within the system.

Attributes:

- EmployeeID: The unique identifier for each employee.

- Name: The employee's name.
- Email: The employee's email address.
- ContactNumber: The employee's contact number.
- Role: The role of the employee in the company (e.g., driver, manager).
- Salary: The salary of the employee.

```
CREATE TABLE Employee (
EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(255) NOT NULL,
Email VARCHAR(255) UNIQUE NOT NULL,
ContactNumber VARCHAR(20),
Role VARCHAR(50) NOT NULL,
Salary DECIMAL(10, 2) NOT NULL
);
```

- **Location Table:** The Location table stores information about various locations the courier company operates in. This includes cities or regions where the company provides services. Each location has a unique ID, a name, and an address associated with it. The table helps track and manage operational areas within the courier system.

Attributes:

- LocationID: The unique identifier for each location.
- LocationName: The name of the location (e.g., city or region).
- Address: The address of the location.

```
CREATE TABLE Location (
LocationID INT PRIMARY KEY AUTO_INCREMENT,
LocationName VARCHAR(100) NOT NULL,
Address TEXT NOT NULL
);
```

- **Payment Table:** The Payment table stores payment-related information for couriers. It records the payment for a specific courier order, including the courier's ID, the associated location, the payment amount, and the date of payment. The table is linked to the Courier and Location tables via foreign keys to indicate which courier the payment is related to and which location it pertains to.

Attributes:

- PaymentID: The unique identifier for each payment transaction.
- CourierID: A foreign key referring to the Courier table, identifying which courier the payment is for.
- LocationID: A foreign key referring to the Location table, identifying the location where the payment is associated.
- Amount: The total amount paid for the courier service.
- PaymentDate: The date on which the payment was made.

```
CREATE TABLE Payment (
```

```
PaymentID INT PRIMARY KEY AUTO_INCREMENT,
CourierID INT,
LocationID INT,
Amount DECIMAL(10, 2) NOT NULL,
PaymentDate DATE,
FOREIGN KEY (CourierID) REFERENCES Courier(CourierID),
FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);
```

Step 3 : Inserting values into the table

Inserting values into the tables involves adding data to each table to populate the database with real-world information. This is done using the INSERT INTO SQL command, specifying the table name and the corresponding values for each column. It is important to ensure that the data types and constraints (such as primary and foreign keys) match the table definitions. Proper insertion of values is crucial for maintaining data consistency and integrity across the system. Additionally, sample data helps simulate the actual operations of the Courier Management System, allowing for effective testing and validation of the system's functionality.

```
INSERT INTO User (Name, Email, Password, ContactNumber, Address) VALUES
('Chhota Bheem', 'bheem@dholakpur.com', 'bheem123', '9876543210', 'Dholakpur Village'),
('Shinchan Nohara', 'shinchan@kasukabe.com', 'shinchan123', '9876543211', 'Kasukabe, Japan'),
('Doraemon', 'doraemon@future.com', 'doraemon123', '9876543212', 'Tokyo, Japan'),
('Ninja Hattori', 'hattori@ninja.com', 'hattori123', '9876543213', 'Shinobi Village'),
('Jackie Chan', 'jackie@kungfu.com', 'jackie123', '9876543214', 'Hong Kong'),
('Heidi', 'heidi@alps.com', 'heidi123', '9876543215', 'Swiss Alps'),
('Raju', 'raju@dholakpur.com', 'raju123', '9876543216', 'Dholakpur Village'),
('Chutki', 'chutki@dholakpur.com', 'chutki123', '9876543217', 'Dholakpur Village'),
('Kalia', 'kalia@dholakpur.com', 'kalia123', '9876543218', 'Dholakpur Village'),
('Indumati', 'indu@dholakpur.com', 'indu123', '9876543219', 'Dholakpur Village'),
('Nobita Nobi', 'nobita@future.com', 'nobita123', '9876543220', 'Tokyo, Japan'),
('Shizuka', 'shizuka@future.com', 'shizuka123', '9876543221', 'Tokyo, Japan'),
('Gian', 'gian@future.com', 'gian123', '9876543222', 'Tokyo, Japan'),
('Suneo', 'suneo@future.com', 'suneo123', '9876543223', 'Tokyo, Japan'),
('Dekisugi', 'dekisugi@future.com', 'dekisugi123', '9876543224', 'Tokyo, Japan');

INSERT INTO CourierServices (ServiceName, Cost) VALUES
('Standard Delivery', 100.00),
('Express Delivery', 250.00),
('Overnight Delivery', 500.00),
('Same-Day Delivery', 700.00),
('International Delivery', 1500.00),
('Heavy Parcel Delivery', 1200.00),
('Fragile Item Delivery', 800.00),
('Medical Supply Delivery', 900.00),
('Food Delivery', 300.00),
('Document Delivery', 200.00),
('Electronics Delivery', 1000.00),
('Gift Delivery', 600.00),
('Bulk Delivery', 2000.00),
('Eco-Friendly Delivery', 400.00),
('VIP Delivery', 3000.00);

INSERT INTO Courier (SenderName, SenderAddress, ReceiverName, ReceiverAddress, Weight,
Status, TrackingNumber, DeliveryDate, UserID, ServiceID) VALUES
```

('Chhota Bheem', 'Dholakpur Village', 'Shinchan Nohara', 'Kasukabe, Japan', 2.5, 'In Transit', 'TRK123456', '2023-10-25', 1, 1),
('Doraemon', 'Tokyo, Japan', 'Ninja Hattori', 'Shinobi Village', 1.8, 'Delivered', 'TRK123457', '2023-10-20', 3, 2),
('Heidi', 'Swiss Alps', 'Jackie Chan', 'Hong Kong', 3.0, 'Pending', 'TRK123458', '2023-10-30', 6, 3),
('Raju', 'Dholakpur Village', 'Chutki', 'Dholakpur Village', 5.0, 'In Transit', 'TRK123459', '2023-10-26', 7, 4),
('Kalia', 'Dholakpur Village', 'Indumati', 'Dholakpur Village', 4.2, 'Delivered', 'TRK123460', '2023-10-22', 9, 5),
('Nobita Nobi', 'Tokyo, Japan', 'Shizuka Minamoto', 'Tokyo, Japan', 1.5, 'Pending', 'TRK123461', '2023-10-31', 11, 6),
('Gian', 'Tokyo, Japan', 'Suneo Honekawa', 'Tokyo, Japan', 6.0, 'In Transit', 'TRK123462', '2023-10-27', 13, 7),
('Dekisugi', 'Tokyo, Japan', 'Nobita Nobi', 'Tokyo, Japan', 2.0, 'Delivered', 'TRK123463', '2023-10-23', 15, 8),
('Shinchan Nohara', 'Kasukabe, Japan', 'Doraemon', 'Tokyo, Japan', 3.5, 'Pending', 'TRK123464', '2023-11-01', 2, 9),
('Jackie Chan', 'Hong Kong', 'Heidi', 'Swiss Alps', 7.0, 'In Transit', 'TRK123465', '2023-10-28', 5, 10),
('Chutki', 'Dholakpur Village', 'Raju', 'Dholakpur Village', 1.0, 'Delivered', 'TRK123466', '2023-10-24', 8, 11),
('Indumati', 'Dholakpur Village', 'Kalia', 'Dholakpur Village', 2.3, 'Pending', 'TRK123467', '2023-11-02', 10, 12),
('Shizuka', 'Tokyo, Japan', 'Gian', 'Tokyo, Japan', 4.7, 'In Transit', 'TRK123468', '2023-10-29', 12, 13),
('Suneo', 'Tokyo, Japan', 'Dekisugi', 'Tokyo, Japan', 3.8, 'Delivered', 'TRK123469', '2023-10-21', 14, 14),
('Ninja Hattori', 'Shinobi Village', 'Chhota Bheem', 'Dholakpur Village', 5.5, 'Pending', 'TRK123470', '2023-11-03', 4, 15);

INSERT INTO Employee (Name, Email, ContactNumber, Role, Salary) VALUES
('Raju', 'raju@dholakpur.com', '9876543220', 'Delivery Boy', 20000.00),
('Jaggu', 'jaggu@dholakpur.com', '9876543221', 'Manager', 50000.00),
('Kalia', 'kalia@dholakpur.com', '9876543222', 'Warehouse Staff', 15000.00),
('Indumati', 'indu@dholakpur.com', '9876543223', 'Customer Support', 18000.00),
('Chutki', 'chutki@dholakpur.com', '9876543224', 'Delivery Boy', 20000.00),
('Nobita Nobi', 'nobita@future.com', '9876543225', 'IT Support', 25000.00),
('Shizuka', 'shizuka@future.com', '9876543226', 'HR Manager', 40000.00),
('Gian', 'gian@future.com', '9876543227', 'Security', 22000.00),
('Suneo', 'suneo@future.com', '9876543228', 'Accountant', 30000.00),
('Dekisugi', 'dekisugi@future.com', '9876543229', 'Developer', 35000.00),
('Heidi', 'heidi@alps.com', '9876543230', 'Delivery Boy', 20000.00),
('Jackie Chan', 'jackie@kungfu.com', '9876543231', 'Security', 22000.00),
('Ninja Hattori', 'hattori@ninja.com', '9876543232', 'Delivery Boy', 20000.00),
('Doraemon', 'doraemon@future.com', '9876543233', 'IT Support', 25000.00),
('Shinchan Nohara', 'shinchan@kasukabe.com', '9876543234', 'Customer Support', 18000.00);

INSERT INTO Location (LocationName, Address) VALUES
('Dholakpur Branch', 'Dholakpur Village, India'),
('Kasukabe Branch', 'Kasukabe, Japan'),
('Tokyo Branch', 'Tokyo, Japan'),
('Hong Kong Branch', 'Hong Kong'),
('Swiss Alps Branch', 'Swiss Alps'),
('Mumbai Branch', 'Mumbai, India'),
('Delhi Branch', 'Delhi, India'),
('Bangalore Branch', 'Bangalore, India'),
('Chennai Branch', 'Chennai, India'),
('Kolkata Branch', 'Kolkata, India'),
('New York Branch', 'New York, USA'),

('London Branch', 'London, UK'),
('Paris Branch', 'Paris, France'),
('Sydney Branch', 'Sydney, Australia'),
('Dubai Branch', 'Dubai, UAE');

INSERT INTO Payment (CourierID, LocationID, Amount, PaymentDate) VALUES
(1, 1, 100.00, '2023-10-24'),
(2, 2, 250.00, '2023-10-19'),
(3, 3, 500.00, '2023-10-29'),
(4, 4, 700.00, '2023-10-26'),
(5, 5, 1500.00, '2023-10-22'),
(6, 6, 1200.00, '2023-10-31'),
(7, 7, 800.00, '2023-10-27'),
(8, 8, 900.00, '2023-10-23'),
(9, 9, 300.00, '2023-11-01'),
(10, 10, 200.00, '2023-10-28'),
(11, 11, 1000.00, '2023-10-24'),
(12, 12, 600.00, '2023-11-02'),
(13, 13, 2000.00, '2023-10-29'),
(14, 14, 400.00, '2023-10-21'),
(15, 15, 3000.00, '2023-11-03');

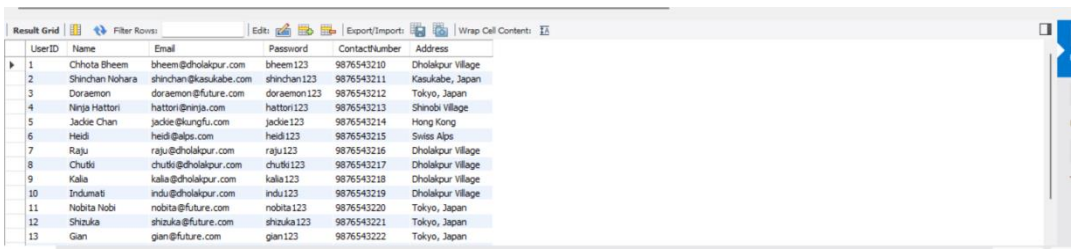
Task 2: Writing queries using Select,Where

SELECT and **WHERE** are SQL clauses used to retrieve and filter data from a database.

- **SELECT**: The SELECT statement is used to specify which columns of data you want to retrieve from a database table. You can choose to select all columns using the * wildcard or specific columns by listing them after the SELECT keyword. This is the primary way to query data from one or more tables.
- **WHERE**: The WHERE clause is used to filter records based on specific conditions. It helps to narrow down the data retrieved by specifying criteria that must be met by the rows. Conditions can be based on comparison operators like =, >, <, >=, <=, <>, or logical operators such as AND, OR, and NOT.

1. List all customers:

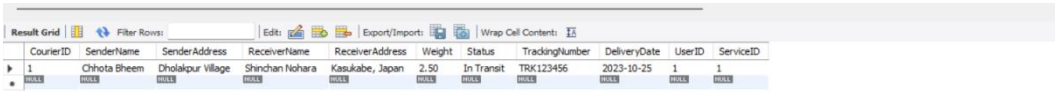
SELECT * FROM User;



UserID	Name	Email	Password	ContactNumber	Address
1	Chhota Bheem	bheem@dholakpur.com	bheem123	9876543210	Dholakpur Village
2	Shinchan Nohara	shinchan@kasukabe.com	shinchan123	9876543211	Kasukabe, Japan
3	Doraemon	doraemon@future.com	doraemon123	9876543212	Tokyo, Japan
4	Ninja Hattori	hattori@ninja.com	hattori123	9876543213	Shinobi Village
5	Jackie Chan	jackie@kungfu.com	jackie123	9876543214	Hong Kong
6	Heidi	heidi@alps.com	heidi123	9876543215	Swiss Alps
7	Raju	raju@dholakpur.com	raju123	9876543216	Dholakpur Village
8	Chutki	chutki@dholakpur.com	chutki123	9876543217	Dholakpur Village
9	Kalia	kalia@dholakpur.com	kalia123	9876543218	Dholakpur Village
10	Indumati	indu@dholakpur.com	indu123	9876543219	Dholakpur Village
11	Nobita Nobi	nobita@future.com	nobita123	9876543220	Tokyo, Japan
12	Shizuka	shizuka@future.com	shizuka123	9876543221	Tokyo, Japan
13	Gian	gian@future.com	gian123	9876543222	Tokyo, Japan

2. List all orders for a specific customer:

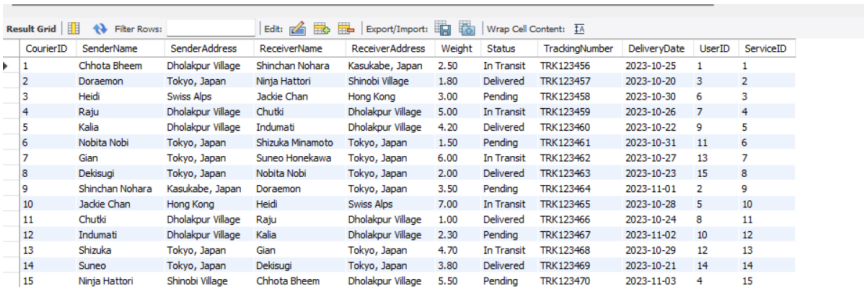
SELECT * FROM Courier WHERE UserID = 1;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1

3. List all couriers:’

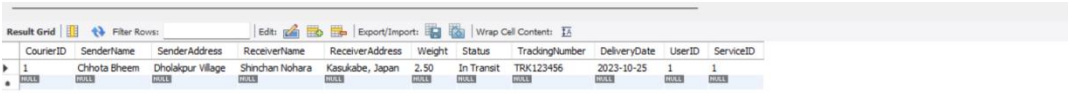
SELECT * FROM Courier;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15

4. List all packages for a specific order:

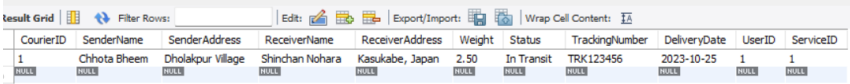
SELECT * FROM Courier WHERE CourierID = 1;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1

5. List all deliveries for a specific courier:

SELECT * FROM Courier WHERE CourierID = 1;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1

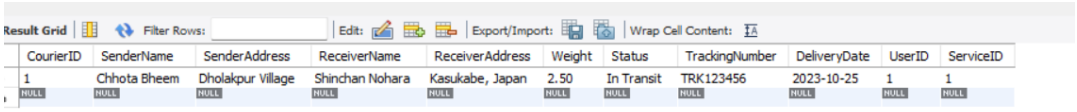
6. List all undelivered packages:

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

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15

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

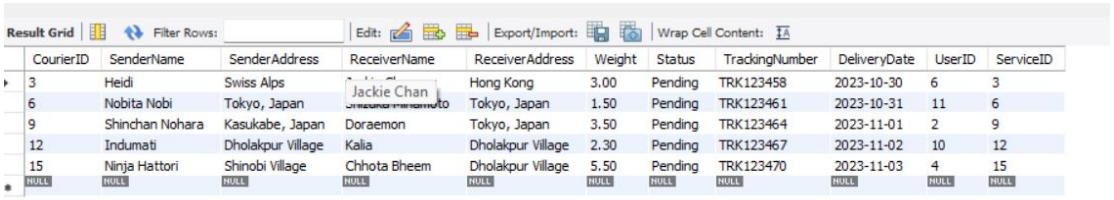
SELECT * FROM Courier WHERE DeliveryDate = '2023-10-25';



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1

8. List all packages with a specific status:

SELECT * FROM Courier WHERE Status = 'Pending';



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID
3	Heidi	Swiss Alps	[illegible]	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3
6	Nobita Nobi	Tokyo, Japan	[illegible]	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15

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

SELECT CourierID, COUNT(*) AS TotalPackages
FROM Courier
GROUP BY CourierID;

CourierID	TotalPackages
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1

10. Find the average delivery time for each courier

SELECT CourierID, AVG(DATEDIFF(DeliveryDate, OrderDate)) AS AvgDeliveryTime
FROM Courier
GROUP BY CourierID;

CourierID	AvgDeliveryTime
1	2.0000
2	2.0000
3	2.0000
4	2.0000
5	2.0000
6	2.0000
7	2.0000
8	2.0000
9	2.0000
10	2.0000
11	2.0000
12	2.0000
13	2.0000
14	2.0000
15	2.0000

11. List all packages with a specific weight range:

SELECT *

FROM Courier
WHERE Weight BETWEEN 2.0 AND 5.0;

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

12. Retrieve employees whose names contain 'John'

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

EmployeeID	Name	Email	ContactNumber	Role	Salary
NULL	NULL	NULL	NULL	NULL	NULL

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

SELECT *
FROM Courier
WHERE CourierID IN (
SELECT CourierID
FROM Payment
WHERE Amount > 50
);

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01

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

- **GROUP BY**:
  The GROUP BY clause in SQL is used to group rows that have the same values into summary rows, like "total," "average," or "count." It is typically used with aggregate functions to perform operations on each group of data. This is useful when you want to combine similar data for reporting purposes (e.g., grouping sales by month or region).
- **Aggregate Functions**:
  Aggregate functions are SQL functions that perform calculations on a set of values and return a single result. Common aggregate functions include:

  i. COUNT(): Counts the number of rows in a group.

ii. SUM(): Calculates the total sum of a numeric column.
iii. AVG(): Computes the average of a numeric column.
iv. MAX(): Returns the maximum value in a column.
v. MIN(): Returns the minimum value in a column.

- **HAVING**:
  The HAVING clause is used to filter records after the GROUP BY operation. While the WHERE clause filters rows before grouping, the HAVING clause filters groups after the aggregation has been performed. This is useful when you need to filter groups based on the result of an aggregate function.

- **ORDER BY**:
  The ORDER BY clause is used to sort the result set of a query based on one or more columns. By default, ORDER BY sorts the data in ascending order (smallest to largest), but you can use the DESC keyword to sort it in descending order (largest to smallest). It helps in arranging the data in a desired order, such as by price, name, or date.

- **WHERE**:
  The WHERE clause is used to filter rows before any grouping or aggregation. It is the first filter applied when querying a table and helps in specifying conditions that rows must satisfy to be included in the result. It can be used with comparison operators, logical operators, and pattern matching functions (e.g., LIKE, BETWEEN, etc.).

These clauses together allow you to filter, aggregate, and organize data efficiently when retrieving it from a database.

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

```
SELECT EmployeeID, COUNT(*) AS TotalCouriersHandled
FROM Courier
GROUP BY EmployeeID;
```

EmployeeID	TotalCouriersHandled
1	1
2	1
3	1

15. Calculate the total revenue generated by each location

```
SELECT LocationID, SUM(Amount) AS TotalRevenue
FROM Payment
GROUP BY LocationID;
```

LocationID	TotalRevenue
1	100.00
2	250.00
3	500.00
4	700.00
5	1500.00
6	1200.00
7	800.00
8	900.00
9	300.00
10	200.00
11	1000.00
12	600.00
13	2000.00
14	400.00
15	3000.00

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

SELECT LocationID, COUNT(*) AS TotalCouriersDelivered
FROM Payment
GROUP BY LocationID;

LocationID	TotalCouriersDelivered
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1

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

SELECT CourierID, AVG(DATEDIFF(DeliveryDate, OrderDate)) AS AvgDeliveryTime
FROM Courier
GROUP BY CourierID
ORDER BY AvgDeliveryTime DESC
LIMIT 1;

CourierID	AvgDeliveryTime
1	2.0000

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

SELECT LocationID, SUM(Amount) AS TotalPayments

FROM Payment
GROUP BY LocationID
HAVING TotalPayments < 1000;

LocationID	TotalPayments
1	100.00
2	250.00
3	500.00
4	700.00
7	800.00
8	900.00
9	300.00
10	200.00
12	600.00
14	400.00

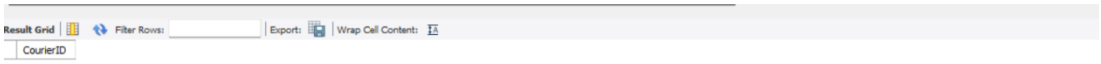
19. Calculate Total Payments per Location

SELECT LocationID, SUM(Amount) AS TotalPayments
FROM Payment
GROUP BY LocationID;

LocationID	TotalPayments
1	100.00
2	250.00
3	500.00
4	700.00
5	1500.00
6	1200.00
7	800.00
8	900.00
9	300.00
10	200.00
11	1000.00
12	600.00
13	2000.00
14	400.00
15	3000.00

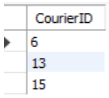
20. Retrieve couriers who have received payments totaling more than $1000 in a specific location (LocationID = X):

SELECT CourierID
FROM Payment
WHERE LocationID = 3
GROUP BY CourierID
HAVING SUM(Amount) > 1000;



21. Retrieve couriers who have received payments totaling more than $1000 after a certain date (PaymentDate > 'YYYY-MM-DD'):

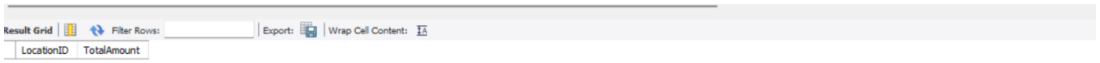
```
SELECT CourierID
FROM Payment
WHERE PaymentDate > '2023-10-25'
GROUP BY CourierID
HAVING SUM(Amount) > 1000;
```



CourierID
6
13
15

22. Retrieve locations where the total amount received is more than $5000 before a certain date (PaymentDate > 'YYYY-MM-DD')

```
SELECT LocationID, SUM(Amount) AS TotalAmount
FROM Payment
WHERE PaymentDate < '2023-10-30'
GROUP BY LocationID
HAVING TotalAmount > 5000;
```



LocationID	TotalAmount

Task 4: Queries with Inner Join,Full Outer Join, Cross Join, Left Outer Join,Right Outer Join

- **INNER JOIN**:
  An INNER JOIN is the most common type of join in SQL. It returns only the rows where there is a match in both tables based on a specified condition. If there is no match between the two tables, the row is not included in the result set. This type of join is used when you want to retrieve records that have related data in both tables.
- **FULL OUTER JOIN**:
  A FULL OUTER JOIN returns all rows when there is a match in one of the tables. It combines the results of both LEFT OUTER JOIN and RIGHT OUTER JOIN. If there is no match, the result will still contain rows from both tables, but with NULL values in columns where there is no matching data. This type of join is useful when you want to include all records from both tables, whether or not they have matching rows.
- **CROSS JOIN**:
  A CROSS JOIN returns the Cartesian product of two tables. That means it combines every row of the first table with every row of the second table. This type of join does not require any condition and results in a large number of rows in the output. It's typically used when you need to combine every possibility between two sets of data, but it should be used cautiously due to the large number of rows it can produce.
- **LEFT OUTER JOIN**:
  A LEFT OUTER JOIN (or simply LEFT JOIN) returns all rows from the left table and the matching rows from the right table. If there is no match, the result

will still include the rows from the left table with NULL values in the columns from the right table. This join is useful when you want to retrieve all records from the left table, even if they don't have corresponding entries in the right table.

- **RIGHT OUTER JOIN**:
  A RIGHT OUTER JOIN (or simply RIGHT JOIN) is the opposite of a LEFT OUTER JOIN. It returns all rows from the right table and the matching rows from the left table. If there is no match, the result will include the rows from the right table with NULL values in the columns from the left table. This join is useful when you want to retrieve all records from the right table, even if there are no matching records in the left table.

These joins help in retrieving and combining data from multiple tables based on relationships between them.

23. Retrieve Payments with Courier Information

```
SELECT Payment.*, Courier.SenderName, Courier.ReceiverName
FROM Payment
INNER JOIN Courier ON Payment.CourierID = Courier.CourierID;
```

PaymentID	CourierID	LocationID	Amount	PaymentDate	SenderName	ReceiverName
1	1	1	100.00	2023-10-24	Chhota Bheem	Shinchan Nohara
2	2	2	250.00	2023-10-19	Doraemon	Ninja Hattori
3	3	3	500.00	2023-10-29	Heidi	Jackie Chan
4	4	4	700.00	2023-10-26	Raju	Chutki
5	5	5	1500.00	2023-10-22	Kalia	Indumati
6	6	6	1200.00	2023-10-31	Nobita Nobi	Shizuka Minamoto
7	7	7	800.00	2023-10-27	Gian	Suneo Honekawa
8	8	8	900.00	2023-10-23	Dekisugi	Nobita Nobi
9	9	9	300.00	2023-11-01	Shinchan Nohara	Doraemon
10	10	10	200.00	2023-10-28	Jackie Chan	Heidi
11	11	11	1000.00	2023-10-24	Chutki	Raju
12	12	12	600.00	2023-11-02	Indumati	Kalia
13	13	13	2000.00	2023-10-29	Shizuka	Gian
14	14	14	400.00	2023-10-21	Suneo	Dekisugi
15	15	15	3000.00	2023-11-03	Ninja Hattori	Chhota Bheem

24. Retrieve Payments with Location Information

```
SELECT Payment.*, Location.LocationName
FROM Payment
INNER JOIN Location ON Payment.LocationID = Location.LocationID;
```

PaymentID	CourierID	LocationID	Amount	PaymentDate	LocationName
1	1	1	100.00	2023-10-24	Dholakpur Branch
2	2	2	250.00	2023-10-19	Kasukabe Branch
3	3	3	500.00	2023-10-29	Tokyo Branch
4	4	4	700.00	2023-10-26	Hong Kong Branch
5	5	5	1500.00	2023-10-22	Swiss Alps Branch
6	6	6	1200.00	2023-10-31	Mumbai Branch
7	7	7	800.00	2023-10-27	Delhi Branch
8	8	8	900.00	2023-10-23	Bangalore Branch
9	9	9	300.00	2023-11-01	Chennai Branch
10	10	10	200.00	2023-10-28	Kolkata Branch
11	11	11	1000.00	2023-10-24	New York Branch
12	12	12	600.00	2023-11-02	London Branch
13	13	13	2000.00	2023-10-29	Paris Branch
14	14	14	400.00	2023-10-21	Sydney Branch
15	15	15	3000.00	2023-11-03	Dubai Branch

25. Retrieve Payments with Courier and Location Information

```
SELECT Payment.*, Courier.SenderName, Courier.ReceiverName, Location.LocationName
FROM Payment
```

INNER JOIN Courier ON Payment.CourierID = Courier.CourierID
INNER JOIN Location ON Payment.LocationID = Location.LocationID;

PaymentID	CourierID	LocationID	Amount	PaymentDate	SenderName	ReceiverName	LocationName
1	1	1	100.00	2023-10-24	Chhota Bheem	Shinchan Nohara	Dholakpur Branch
2	2	2	250.00	2023-10-19	Doraemon	Ninja Hattori	Kasukabe Branch
3	3	3	500.00	2023-10-29	Heidi	Jackie Chan	Tokyo Branch
4	4	4	700.00	2023-10-26	Raju	Chutki	Hong Kong Branch
5	5	5	1500.00	2023-10-22	Kalia	Indumati	Swiss Alps Branch
6	6	6	1200.00	2023-10-31	Nobita Nobi	Shizuka Minamoto	Mumbai Branch
7	7	7	800.00	2023-10-27	Gian	Suneo Honekawa	Delhi Branch
8	8	8	900.00	2023-10-23	Dekisugi	Nobita Nobi	Bangalore Branch
9	9	9	300.00	2023-11-01	Shinchan Nohara	Doraemon	Chennai Branch
10	10	10	200.00	2023-10-28	Jackie Chan	Heidi	Kolkata Branch
11	11	11	1000.00	2023-10-24	Chutki	Raju	New York Branch
12	12	12	600.00	2023-11-02	Indumati	Kalia	London Branch
13	13	13	2000.00	2023-10-29	Shizuka	Gian	Paris Branch
14	14	14	400.00	2023-10-21	Suneo	Dekisugi	Sydney Branch
15	15	15	3000.00	2023-11-03	Ninja Hattori	Chhota Bheem	Dubai Branch

26. List all payments with courier details

SELECT Payment.*, Courier.*
FROM Payment
INNER JOIN Courier ON Payment.CourierID = Courier.CourierID;

PaymentID	CourierID	LocationID	Amount	PaymentDate	CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	Orde
1	1	1	100.00	2023-10-24	1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-
2	2	2	250.00	2023-10-19	2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-
3	3	3	500.00	2023-10-29	3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-
4	4	4	700.00	2023-10-26	4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-
5	5	5	1500.00	2023-10-22	5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-
6	6	6	1200.00	2023-10-31	6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-
7	7	7	800.00	2023-10-27	7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-
8	8	8	900.00	2023-10-23	8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-
9	9	9	300.00	2023-11-01	9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-
10	10	10	200.00	2023-10-28	10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-
11	11	11	1000.00	2023-10-24	11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-
12	12	12	600.00	2023-11-02	12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-
13	13	13	2000.00	2023-10-29	13	12 jika	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-
14	14	14	400.00	2023-10-21	14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-
15	15	15	3000.00	2023-11-03	15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-

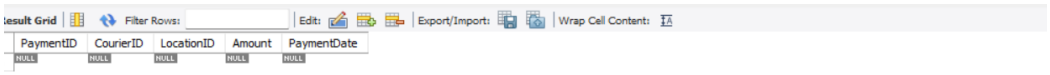
27. Total payments received for each courier

SELECT Courier.CourierID, Courier.SenderName, SUM(Payment.Amount) AS TotalPayments
FROM Payment
INNER JOIN Courier ON Payment.CourierID = Courier.CourierID
GROUP BY Courier.CourierID;

CourierID	SenderName	TotalPayments
1	Chhota Bheem	100.00
2	Doraemon	250.00
3	Heidi	500.00
4	Raju	700.00
5	Kalia	1500.00
6	Nobita Nobi	1200.00
7	Gian	800.00
8	Dekisugi	900.00
9	Shinchan Nohara	300.00
10	Jackie Chan	200.00
11	Chutki	1000.00
12	Indumati	600.00
13	Shizuka	2000.00
14	Suneo	400.00
15	Ninja Hattori	3000.00

28. List payments made on a specific date

SELECT *
FROM Payment
WHERE PaymentDate = '2023-10-25';



29. Get Courier Information for Each Payment

SELECT Payment.*, Courier.*
FROM Payment
LEFT JOIN Courier ON Payment.CourierID = Courier.CourierID;

PaymentID	CourierID	LocationID	Amount	PaymentDate	CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	Order
1	1	1	100.00	2023-10-24	1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-
2	2	2	250.00	2023-10-19	2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-
3	3	3	500.00	2023-10-29	3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-
4	4	4	700.00	2023-10-26	4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-
5	5	5	1500.00	2023-10-22	5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-
6	6	6	1200.00	2023-10-31	6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-
7	7	7	800.00	2023-10-27	7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-
8	8	8	900.00	2023-10-23	8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-
9	9	9	300.00	2023-11-01	9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-
10	10	10	200.00	2023-10-28	10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-
11	11	11	1000.00	2023-10-24	11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-
12	12	12	600.00	2023-11-02	12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-
13	13	13	2000.00	2023-10-29	13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-
14	14	14	400.00	2023-10-21	14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-
15	15	15	3000.00	2023-11-03	15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-

30. Get Payment Details with Location

SELECT Payment.*, Location.*
FROM Payment
LEFT JOIN Location ON Payment.LocationID = Location.LocationID;

PaymentID	CourierID	LocationID	Amount	PaymentDate	LocationID	LocationName	Address
1	1	1	100.00	2023-10-24	1	Dholakpur Branch	Dholakpur Village, India
2	2	2	250.00	2023-10-19	2	Kasukabe Branch	Kasukabe, Japan
3	3	3	500.00	2023-10-29	3	Tokyo Branch	Tokyo, Japan
4	4	4	700.00	2023-10-26	4	Hong Kong Branch	Hong Kong
5	5	5	1500.00	2023-10-22	5	Swiss Alps Branch	Swiss Alps
6	6	6	1200.00	2023-10-31	6	Mumbai Branch	Mumbai, India
7	7	7	800.00	2023-10-27	7	Delhi Branch	Delhi, India
8	8	8	900.00	2023-10-23	8	Bangalore Branch	Bangalore, India
9	9	9	300.00	2023-11-01	9	Chennai Branch	Chennai, India
10	10	10	200.00	2023-10-28	10	Kolkata Branch	Kolkata, India
11	11	11	1000.00	2023-10-24	11	New York Branch	New York, USA
12	12	12	600.00	2023-11-02	12	London Branch	London, UK
13	13	13	2000.00	2023-10-29	13	Paris Branch	Paris, France
14	14	14	400.00	2023-10-21	14	Sydney Branch	Sydney, Australia
15	15	15	3000.00	2023-11-03	15	Dubai Branch	Dubai, UAE

31. Calculating Total Payments for Each Courier

SELECT Courier.CourierID, Courier.SenderName, SUM(Payment.Amount) AS TotalPayments
FROM Payment
INNER JOIN Courier ON Payment.CourierID = Courier.CourierID
GROUP BY Courier.CourierID;

CourierID	SenderName	TotalPayments
1	Chhota Bheem	100.00
2	Doraemon	250.00
3	Heidi	500.00
4	Raju	700.00
5	Kalia	1500.00
6	Nobita Nobi	1200.00
7	Gian	800.00
8	Dekisugi	900.00
9	Shinchan Nohara	300.00
10	Jackie Chan	200.00
11	Chutki	1000.00
12	Indumati	600.00
13	Shizuka	2000.00
14	Suneo	400.00
15	Ninja Hattori	3000.00

32. List Payments Within a Date Range

SELECT *
FROM Payment
WHERE PaymentDate BETWEEN '2023-10-20' AND '2023-10-30';

PaymentID	CourierID	LocationID	Amount	PaymentDate
1	1	1	100.00	2023-10-24
3	3	3	500.00	2023-10-29
4	4	4	700.00	2023-10-26
5	5	5	1500.00	2023-10-22
7	7	7	800.00	2023-10-27
8	8	8	900.00	2023-10-23
10	10	10	200.00	2023-10-28
11	11	11	1000.00	2023-10-24
13	13	13	2000.00	2023-10-29
14	14	14	400.00	2023-10-21
NULL	NULL	NULL	NULL	NULL

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

SELECT User.*, Courier.*
FROM User
LEFT JOIN Courier ON User.UserID = Courier.UserID;

UserID	Name	Email	Password	ContactNumber	Address	CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber
1	Chhota Bheem	bheem@dholakpur.com	bheem123	9876543210	Dholakpur Village	1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456
2	Shinchan Nohara	shinchan@kasukabe.com	shinchan123	9876543211	Kasukabe, Japan	9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464
3	Doraemon	doraemon@future.com	doraemon123	9876543212	Tokyo, Japan	2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457
4	Ninja Hattori	hattori@ninja.com	hattori123	9876543213	Shinobi Village	15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470
5	Jackie Chan	jackie@kungfu.com	jackie123	9876543214	Hong Kong	10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465
6	Heidi	heidi@alps.com	heidi123	9876543215	Swiss Alps	3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458
7	Raju	raju@dholakpur.com	raju123	9876543216	Dholakpur Village	4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459
8	Chutki	chutki@dholakpur.com	chutki123	9876543217	Dholakpur Village	11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466
9	Kalia	kalia@dholakpur.com	kalia123	9876543218	Dholakpur Village	5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460
10	Indumati	indu@dholakpur.com	indu123	9876543219	Dholakpur Village	12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467
11	Nobita Nobi	nobita@future.com	nobita123	9876543220	Tokyo, Japan	6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461
12	Shizuka	shizuka@future.com	shizuka123	9876543221	Tokyo, Japan	13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468
13	Gian	gian@future.com	gian123	9876543222	Tokyo, Japan	7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462
14	Suneo	suneo@future.com	suneo123	9876543223	Tokyo, Japan	14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469
15	Dekisugi	dekisugi@future.com	dekisugi123	9876543224	Tokyo, Japan	8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463

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

SELECT Courier.*, CourierServices.*
FROM Courier
LEFT JOIN CourierServices ON Courier.ServiceID = CourierServices.ServiceID;

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID	ServiceID	ServiceName	Cost
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1	1	Standard Delivery	100.0(
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18	2	2	Express Delivery	250.0(
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3	3	Overnight Delivery	500.0(
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL	4	Same-Day Delivery	700.0(
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL	5	International Delivery	1500.(
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29	NULL	6	Heavy Parcel Delivery	1200.(
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL	7	Fragile Item Delivery	800.0(
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21	NULL	8	Medical Supply Delivery	900.0(
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL	9	Food Delivery	300.0(
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL	10	Document Delivery	200.0(
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL	11	Electronics Delivery	1000.(
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL	12	Gift Delivery	600.0(
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL	13	Bulk Delivery	2000.(
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL	14	Eco-Friendly Delivery	400.0(
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL	15	VIP Delivery	3000.(

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

SELECT Employee.*, Payment.*
FROM Employee
LEFT JOIN Payment ON Employee.EmployeeID = Payment.CourierID;

EmployeeID	Name	Email	ContactNumber	Role	Salary	PaymentID	CourierID	LocationID	Amount	PaymentDate
1	Raju	raju@dholakpur.com	9876543220	Delivery Boy	20000.00	1	1	1	100.00	2023-10-24
2	Jaggu	jaggu@dholakpur.com	9876543221	Manager	50000.00	2	2	2	250.00	2023-10-19
3	Kalia	kalia@dholakpur.com	9876543222	Warehouse Staff	15000.00	3	3	3	500.00	2023-10-29
4	Indumati	indu@dholakpur.com	9876543223	Customer Support	18000.00	4	4	4	700.00	2023-10-26
5	Chutki	chutki@dholakpur.com	9876543224	Delivery Boy	20000.00	5	5	5	1500.00	2023-10-22
6	Nobita Nobi	nobita@future.com	9876543225	IT Support	25000.00	6	6	6	1200.00	2023-10-31
7	Shizuka	shizuka@future.com	9876543226	HR Manager	40000.00	7	7	7	800.00	2023-10-27
8	Gian	gian@future.com	9876543227	Security	22000.00	8	8	8	900.00	2023-10-23
9	Suneo	suneo@future.com	9876543228	Accountant	30000.00	9	9	9	300.00	2023-11-01
10	Dekisugi	dekisugi@future.com	9876543229	Developer	35000.00	10	10	10	200.00	2023-10-28
11	Heidi	heidi@alps.com	9876543230	Delivery Boy	20000.00	11	11	11	1000.00	2023-10-24
12	Jackie Chan	jackie@kungfu.com	9876543231	Security	22000.00	12	12	12	600.00	2023-11-02
13	Ninja Hattori	hattori@ninja.com	9876543232	Delivery Boy	20000.00	13	13	13	2000.00	2023-10-29
14	Doraemon	doraemon@future.com	9876543233	IT Support	25000.00	14	14	14	400.00	2023-10-21
15	Shinchan ...	shinchan@kasukabe....	9876543234	Customer Support	18000.00	15	15	15	3000.00	2023-11-03

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

SELECT User.*, CourierServices.*
FROM User
CROSS JOIN CourierServices;

UserID	Name	Email	Password	ContactNumber	Address	ServiceID	ServiceName	Cost
15	Dekisugi	dekisugi@future.com	dekisugi123	9876543224	Tokyo, Japan	1	Standard Delivery	100.00
14	Suneo	suneo@future.com	suneo123	9876543223	Tokyo, Japan	1	Standard Delivery	100.00
13	Gian	gian@future.com	gian123	9876543222	Tokyo, Japan	1	Standard Delivery	100.00
12	Shizuka	shizuka@future.com	shizuka123	9876543221	Tokyo, Japan	1	Standard Delivery	100.00
11	Nobita Nobi	nobita@future.com	nobita123	9876543220	Tokyo, Japan	1	Standard Delivery	100.00
10	Indumati	indu@dholakpur.com	indu123	9876543219	Dholakpur Village	1	Standard Delivery	100.00
9	Kalia	kalia@dholakpur.com	kalia123	9876543218	Dholakpur Village	1	Standard Delivery	100.00
8	Chutki	chutki@dholakpur.com	chutki123	9876543217	Dholakpur Village	1	Standard Delivery	100.00
7	Raju	raju@dholakpur.com	raju123	9876543216	Dholakpur Village	1	Standard Delivery	100.00
6	Heidi	heidi@alps.com	heidi123	9876543215	Swiss Alps	1	Standard Delivery	100.00
5	Jackie Chan	jackie@kungfu.com	jackie123	9876543214	Hong Kong	1	Standard Delivery	100.00
4	Ninja Hattori	hattori@ninja.com	hattori123	9876543213	Shinobi Village	1	Standard Delivery	100.00
3	Doraemon	doraemon@future.com	doraemon...	9876543212	Tokyo, Japan	1	Standard Delivery	100.00
2	Shinchan ...	shinchan@kasukabe....	shinchan123	9876543211	Kasukabe, Japan	1	Standard Delivery	100.00
1	Chhota Bh...	bheem@dholakpur.com	bheem123	9876543210	Dholakpur Village	1	Standard Delivery	100.00
15	Dekisugi	dekisugi@future.com	dekisugi123	9876543224	Tokyo, Japan	2	Express Delivery	250.00
14	Suneo	suneo@future.com	suneo123	9876543223	Tokyo, Japan	2	Express Delivery	250.00
13	Gian	gian@future.com	gian123	9876543222	Tokyo, Japan	2	Express Delivery	250.00
12	Shizuka	shizuka@future.com	shizuka123	9876543221	Tokyo, Japan	2	Express Delivery	250.00
11	Nobita Nobi	nobita@future.com	nobita123	9876543220	Tokyo, Japan	2	Express Delivery	250.00

Result 51

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

SELECT Employee.*, Location.*
FROM Employee
CROSS JOIN Location;

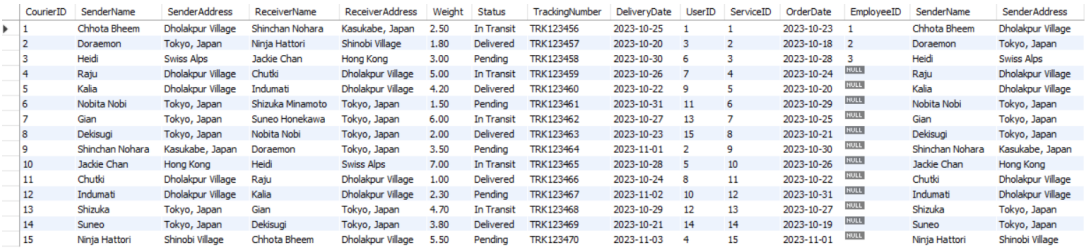
EmployeeID	Name	Email	ContactNumber	Role	Salary	LocationID	LocationName	Address
15	Shinchan Nohara	shinchan@kasukabe.com	9876543234	Customer Support	18000.00	1	Dholakpur Branch	Dholakpur Village, India
14	Doraemon	doraemon@future.com	9876543233	IT Support	25000.00	1	Dholakpur Branch	Dholakpur Village, India
13	Ninja Hattori	hattori@ninja.com	9876543232	Delivery Boy	20000.00	1	Dholakpur Branch	Dholakpur Village, India
12	Jackie Chan	jackie@kungfu.com	9876543231	Security	22000.00	1	Dholakpur Branch	Dholakpur Village, India
11	Heidi	heidi@alps.com	9876543230	Delivery Boy	20000.00	1	Dholakpur Branch	Dholakpur Village, India
10	Dekisugi	dekisugi@future.com	9876543229	Developer	35000.00	1	Dholakpur Branch	Dholakpur Village, India
9	Suneo	suneo@future.com	9876543228	Accountant	30000.00	1	Dholakpur Branch	Dholakpur Village, India
8	Gian	gian@future.com	9876543227	Security	22000.00	1	Dholakpur Branch	Dholakpur Village, India
7	Shizuka	shizuka@future.com	9876543226	HR Manager	40000.00	1	Dholakpur Branch	Dholakpur Village, India
6	Nobita Nobi	nobita@future.com	9876543225	IT Support	25000.00	1	Dholakpur Branch	Dholakpur Village, India
5	Chutki	chutki@dholakpur.com	9876543224	Delivery Boy	20000.00	1	Dholakpur Branch	Dholakpur Village, India
4	Indumati	indu@dholakpur.com	9876543223	Customer Support	18000.00	1	Dholakpur Branch	Dholakpur Village, India
3	Kalia	kalia@dholakpur.com	9876543222	Warehouse Staff	15000.00	1	Dholakpur Branch	Dholakpur Village, India
2	Jaggu	jaggu@dholakpur.com	9876543221	Manager	50000.00	1	Dholakpur Branch	Dholakpur Village, India
1	Raju	raju@dholakpur.com	9876543220	Delivery Boy	20000.00	1	Dholakpur Branch	Dholakpur Village, India
15	Shinchan Nohara	shinchan@kasukabe.com	9876543234	Customer Support	18000.00	2	Kasukabe Branch	Kasukabe, Japan
14	Doraemon	doraemon@future.com	9876543233	IT Support	25000.00	2	Kasukabe Branch	Kasukabe, Japan
13	Ninja Hattori	hattori@ninja.com	9876543232	Delivery Boy	20000.00	2	Kasukabe Branch	Kasukabe, Japan
12	Jackie Chan	jackie@kungfu.com	9876543231	Security	22000.00	2	Kasukabe Branch	Kasukabe, Japan
11	Heidi	heidi@alps.com	9876543230	Delivery Boy	20000.00	2	Kasukabe Branch	Kasukabe, Japan

Result 52

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

SELECT Courier.*, User.Name AS SenderName, User.Address AS SenderAddress
FROM Courier

LEFT JOIN User ON Courier.UserID = User.UserID;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID	SenderName	SenderAddress
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1	Chhota Bheem	Dholakpur Village
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18	2	Doraemon	Tokyo, Japan
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3	Heidi	Swiss Alps
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL	Raju	Dholakpur Village
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL	Kalia	Dholakpur Village
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29	NULL	Nobita Nobi	Tokyo, Japan
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL	Gian	Tokyo, Japan
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21	NULL	Dekisugi	Tokyo, Japan
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL	Shinchan Nohara	Kasukabe, Japan
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL	Jackie Chan	Hong Kong
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL	Chutki	Dholakpur Village
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL	Indumati	Dholakpur Village
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL	Shizuka	Tokyo, Japan
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL	Suneo	Tokyo, Japan
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL	Ninja Hattori	Shinobi Village

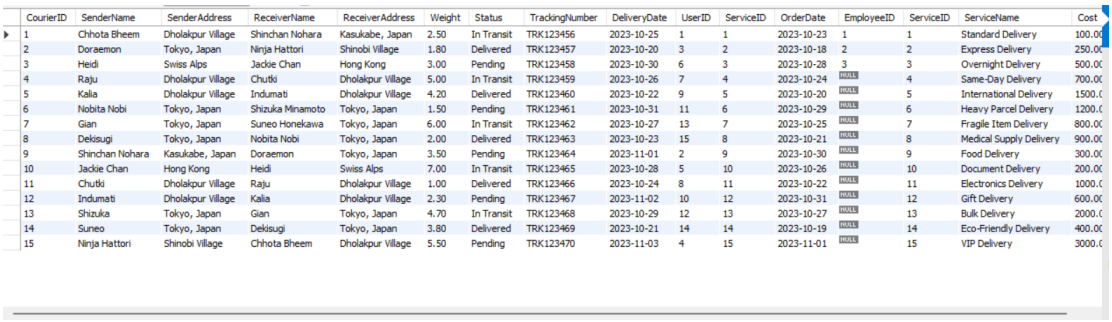
39. Retrieve a list of couriers and their corresponding receiver information (if available):

SELECT Courier.*, User.Name AS ReceiverName, User.Address AS ReceiverAddress
FROM Courier
LEFT JOIN User ON Courier.UserID = User.UserID;

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID	ReceiverName	ReceiverAddress
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1	Chhota Bheem	Dholakpur Village
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18	2	Doraemon	Tokyo, Japan
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3	Heidi	Swiss Alps
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL	Raju	Dholakpur Village
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL	Kalia	Dholakpur Village
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29	NULL	Nobita Nobi	Tokyo, Japan
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL	Gian	Tokyo, Japan
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21	NULL	Dekisugi	Tokyo, Japan
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL	Shinchan Nohara	Kasukabe, Japan
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL	Jackie Chan	Hong Kong
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL	Chutki	Dholakpur Village
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL	Indumati	Dholakpur Village
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL	Shizuka	Tokyo, Japan
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL	Suneo	Tokyo, Japan
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL	Ninja Hattori	Shinobi Village

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

SELECT Courier.*, CourierServices.*
FROM Courier
LEFT JOIN CourierServices ON Courier.ServiceID = CourierServices.ServiceID;



CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID	ServiceID	ServiceName	Cost
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1	1	Standard Delivery	100.00
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18	2	2	Express Delivery	250.00
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3	3	Overnight Delivery	500.00
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL	4	Same-Day Delivery	700.00
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL	5	International Delivery	1500.0
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29	NULL	6	Heavy Parcel Delivery	1200.0
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL	7	Fragile Item Delivery	800.00
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21	NULL	8	Medical Supply Delivery	900.00
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL	9	Food Delivery	300.00
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL	10	Document Delivery	200.00
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL	11	Electronics Delivery	1000.0
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL	12	Gift Delivery	600.00
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL	13	Bulk Delivery	2000.0
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL	14	Eco-Friendly Delivery	400.00
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL	15	VIP Delivery	3000.0

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

SELECT Employee.EmployeeID, Employee.Name, COUNT(Courier.CourierID) AS TotalCouriers
FROM Employee
LEFT JOIN Courier ON Employee.EmployeeID = Courier.EmployeeID
GROUP BY Employee.EmployeeID;

EmployeeID	Name	TotalCouriers
1	Raju	1
2	Jaggu	1
3	Kalia	1
4	Indumati	0
5	Chutki	0
6	Nobita Nobi	0
7	Shizuka	0
8	Gian	0
9	Suneo	0
10	Dekisugi	0
11	Heidi	0
12	Jackie Chan	0
13	Ninja Hattori	0
14	Doraemon	0
15	Shinchan ...	0

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

SELECT Location.LocationID, Location.LocationName, SUM(Payment.Amount) AS TotalPayments
FROM Location
LEFT JOIN Payment ON Location.LocationID = Payment.LocationID
GROUP BY Location.LocationID;

LocationID	LocationName	TotalPayments
1	Dholakpur Branch	100.00
2	Kasukabe Branch	250.00
3	Tokyo Branch	500.00
4	Hong Kong Branch	700.00
5	Swiss Alps Branch	1500.00
6	Mumbai Branch	1200.00
7	Delhi Branch	800.00
8	Bangalore Branch	900.00
9	Chennai Branch	300.00
10	Kolkata Branch	200.00
11	New York Branch	1000.00
12	London Branch	600.00
13	Paris Branch	2000.00
14	Sydney Branch	400.00
15	Dubai Branch	3000.00

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

SELECT *
FROM Courier
WHERE SenderName = 'Chhota Bheem';

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

44. List all employees who share the same role.

SELECT Role, COUNT(*) AS TotalEmployees
FROM Employee
GROUP BY Role

Role	TotalEmployees
Delivery Boy	4
Customer Support	2
IT Support	2
Security	2

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

SELECT *
FROM Payment
WHERE LocationID = 1;

PaymentID	CourierID	LocationID	Amount	PaymentDate
1	1	1	100.00	2023-10-24
NULL	NULL	NULL	NULL	NULL

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

SELECT *
FROM Courier
WHERE SenderAddress = 'Dholakpur Village';

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

SELECT Employee.EmployeeID, Employee.Name, COUNT(Courier.CourierID) AS TotalCouriersDelivered
FROM Employee
LEFT JOIN Courier ON Employee.EmployeeID = Courier.EmployeeID
GROUP BY Employee.EmployeeID;

EmployeeID	Name	TotalCouriersDelivered
1	Raju	1
2	Jaggu	1
3	Kalia	1
4	Indumati	0
5	Chutki	0
6	Nobita Nobi	0
7	Shizuka	0
8	Gian	0
9	Suneo	0
10	Dekisugi	0
11	Heidi	0
12	Jackie Chan	0
13	Ninja Hattori	0
14	Doraemon	0
15	Shinchan ...	0

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

SELECT Courier.*
FROM Courier
INNER JOIN Payment ON Courier.CourierID = Payment.CourierID
INNER JOIN CourierServices ON Courier.ServiceID = CourierServices.ServiceID
WHERE Payment.Amount > CourierServices.Cost;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID

Task 5: Queries with Inner Queries, Non Equi Joins, Equi joins,Exist,Any,All

- **Inner Queries (Subqueries):**
  Inner queries, also known as *subqueries*, are queries nested inside another SQL query. They are typically used to retrieve data that will be used in the main query's condition. Subqueries can be placed in SELECT, FROM, or WHERE clauses and help break complex queries into manageable steps.

- **Equi Joins:**
  An **Equi Join** is a type of join that uses an equality (=) operator to match rows from two or more tables based on a common column. It is the most common form of join and is used to retrieve related data from different tables that have a common field.

- **Non-Equi Joins:**
  A **Non-Equi Join** is a join condition that uses comparison operators other than equality, such as <, >, <=, or >=. These are used when rows need to be matched based on a range or condition rather than exact equality.

- **EXISTS:**
  The EXISTS keyword is used to test for the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records. It's often used in WHERE clauses to check for related records efficiently.

- **ANY:**
  The ANY operator is used to compare a value to *any* value in a set returned by a subquery. It's typically used with comparison operators like =, >, <, etc. It returns TRUE if *at least one* of the values in the set satisfies the condition.

- **ALL:**
  The ALL operator is used to compare a value to *all* values in a subquery result. It returns TRUE only if the condition holds true for *every* value in the set. It is useful when you want a condition to be satisfied against all rows returned by a subquery.

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

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

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

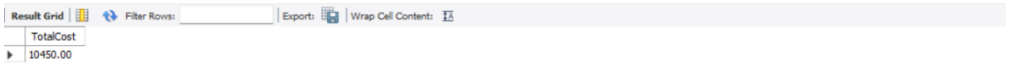
50. Find the names of all employees who have a salary greater than the average salary:

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

Name
Jaggu
Shizuka
Suneo
Dekisugi

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

```
SELECT SUM(Cost) AS TotalCost
FROM CourierServices
WHERE Cost < (SELECT MAX(Cost) FROM CourierServices);
```



TotalCost
10450.00

52. Find all couriers that have been paid for

```
SELECT *
FROM Courier
WHERE CourierID IN (SELECT CourierID FROM Payment);
```

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID
1	Chhota Bheem	Dholakpur Village	Shinchan Nohara	Kasukabe, Japan	2.50	In Transit	TRK123456	2023-10-25	1	1	2023-10-23	1
2	Doraemon	Tokyo, Japan	Ninja Hattori	Shinobi Village	1.80	Delivered	TRK123457	2023-10-20	3	2	2023-10-18	2
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL
6	Nobita Nobi	Tokyo, Japan	Shizuka Minamoto	Tokyo, Japan	1.50	Pending	TRK123461	2023-10-31	11	6	2023-10-29	NULL
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL
8	Dekisugi	Tokyo, Japan	Nobita Nobi	Tokyo, Japan	2.00	Delivered	TRK123463	2023-10-23	15	8	2023-10-21	NULL
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL
11	Chutki	Dholakpur Village	Raju	Dholakpur Village	1.00	Delivered	TRK123466	2023-10-24	8	11	2023-10-22	NULL
12	Indumati	Dholakpur Village	Kalia	Dholakpur Village	2.30	Pending	TRK123467	2023-11-02	10	12	2023-10-31	NULL
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
SELECT LocationID, LocationName
FROM Location
WHERE LocationID = (
   SELECT LocationID
   FROM Payment
   WHERE Amount = (SELECT MAX(Amount) FROM Payment)
   );
```

LocationID	LocationName
15	Dubai Branch
NULL	NULL

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

```
SELECT *
FROM Courier
WHERE Weight > ALL (
   SELECT Weight
   FROM Courier
   WHERE SenderName = 'Chhota Bheem'
);
```

CourierID	SenderName	SenderAddress	ReceiverName	ReceiverAddress	Weight	Status	TrackingNumber	DeliveryDate	UserID	ServiceID	OrderDate	EmployeeID
3	Heidi	Swiss Alps	Jackie Chan	Hong Kong	3.00	Pending	TRK123458	2023-10-30	6	3	2023-10-28	3
4	Raju	Dholakpur Village	Chutki	Dholakpur Village	5.00	In Transit	TRK123459	2023-10-26	7	4	2023-10-24	NULL
5	Kalia	Dholakpur Village	Indumati	Dholakpur Village	4.20	Delivered	TRK123460	2023-10-22	9	5	2023-10-20	NULL
7	Gian	Tokyo, Japan	Suneo Honekawa	Tokyo, Japan	6.00	In Transit	TRK123462	2023-10-27	13	7	2023-10-25	NULL
9	Shinchan Nohara	Kasukabe, Japan	Doraemon	Tokyo, Japan	3.50	Pending	TRK123464	2023-11-01	2	9	2023-10-30	NULL
10	Jackie Chan	Hong Kong	Heidi	Swiss Alps	7.00	In Transit	TRK123465	2023-10-28	5	10	2023-10-26	NULL
13	Shizuka	Tokyo, Japan	Gian	Tokyo, Japan	4.70	In Transit	TRK123468	2023-10-29	12	13	2023-10-27	NULL
14	Suneo	Tokyo, Japan	Dekisugi	Tokyo, Japan	3.80	Delivered	TRK123469	2023-10-21	14	14	2023-10-19	NULL
15	Ninja Hattori	Shinobi Village	Chhota Bheem	Dholakpur Village	5.50	Pending	TRK123470	2023-11-03	4	15	2023-11-01	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

PART - 2 Coding

Connecting PyCharm to a SQL Database (MySQL)

To interact with a MySQL database from your Python project in PyCharm, a connection needs to be established using a database connector. This allows you to send SQL queries directly from your Python application and manage data dynamically.

1. Install MySQL Connector:

Before connecting, install the required connector by running the following command in PyCharm's terminal:

```
pip install mysql-connector-python
```

2. Import and Establish Connection:

Use the mysql.connector module in your Python script. A connection is created by specifying host, user, password, and database name

```
import mysql.connector

def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="CouriersManagementSystem"
    )
```

Task 1: Control Flow Statements

Control flow statements determine the order in which instructions in a program are executed. They help make decisions, repeat actions, and control how a program flows based on conditions.

1. Conditional Statements (if, elif, else):

Used to execute code blocks based on specific conditions.

- if checks a condition.
- elif checks another condition if the previous one is false.
- else executes when none of the above conditions are true.

2. Loops (for, while):

Used to repeat actions.

- for loop is used to iterate over sequences (like lists, strings, etc.).
- while loop continues as long as a condition is true.

3. Loop Control Statements:

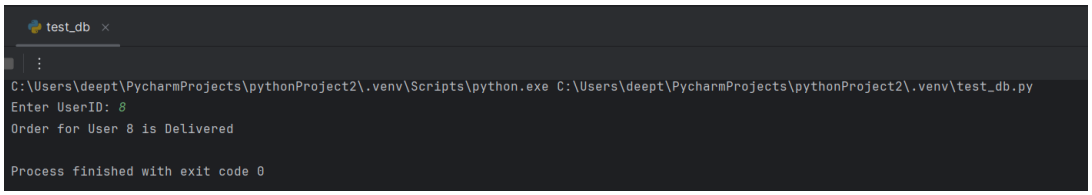
- break: Exits the loop prematurely.
- continue: Skips the current iteration and goes to the next one.
- pass: Does nothing; a placeholder for future code.

4. Nested Control Flow:

- You can nest if statements and loops inside each other for more complex logic.

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.

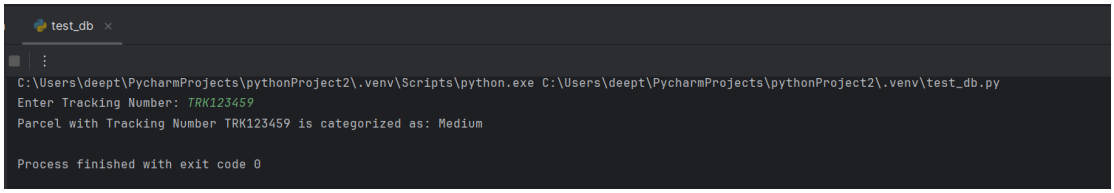
```
cursor = db.cursor()
def check_order_status(user_id):
    """Check and print the order status for a given user ID."""
    conn = get_db_connection()
    cursor = conn.cursor()

    query = "SELECT Status FROM Courier WHERE UserID = %s"
    cursor.execute(query, (user_id,))
    result = cursor.fetchone()
    if result:
        status = result[0]
        if status == "Delivered":
            print(f"Order for User {user_id} is Delivered")
        elif status == "Pending":
            print(f"Order for User {user_id} is Pending")
        elif status == "In Transit":
            print(f"Order for User {user_id} is In Transit")
        else:
            print(f"Order for User {user_id} has an unknown status.")
    else:
        print(f"No order found for User {user_id}")
    cursor.close()
    conn.close()
user_id = input("Enter UserID: ")
check_order_status(user_id)
```



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

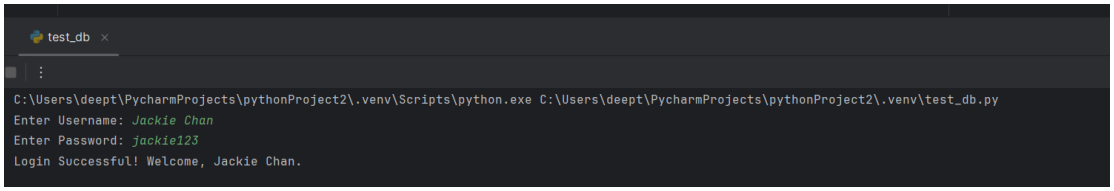
```
cursor = db.cursor()
def categorize_parcel(tracking_number):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "SELECT weight FROM courier WHERE TrackingNumber = %s"
```

```
    cursor.execute(query, (tracking_number,))
    result = cursor.fetchone()
    if result:
        weight = result[0]
        match weight:
            case w if w < 5:
                category = "Light"
            case w if 5 <= w < 20:
                category = "Medium"
            case _:
                category = "Heavy"
        print(f"Parcel with Tracking Number {tracking_number} is categorized as: {category}")
    else:
        print(f"No parcel found with Tracking Number: {tracking_number}")
    cursor.close()
    conn.close()
tracking_number = input("Enter Tracking Number: ")
categorize_parcel(tracking_number)
```



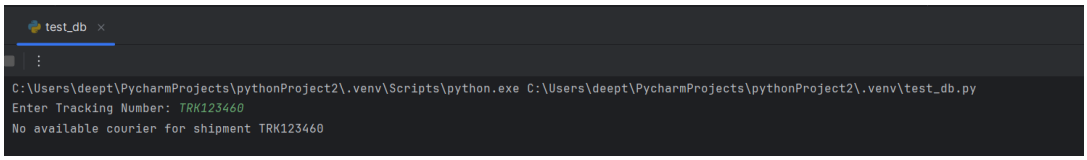
3. Implement User Authentication **1.** Create a login system for employees and customers using python **control flow statements.**

```
cursor = db.cursor()
def authenticate_user():
    conn = get_db_connection()
    cursor = conn.cursor()
    username = input("Enter Username: ")
    password = input("Enter Password: ")
    query = "SELECT * FROM user WHERE name = %s AND password = %s"
    cursor.execute(query, (username, password))
    result = cursor.fetchone()
    if result:
        print(f"Login Successful! Welcome, {username}. ")
    else:
        print("Invalid Credentials! Please try again. ")
    cursor.close()
    conn.close()

authenticate_user()
```



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.

```
cursor = db.cursor()
def assign_courier(tracking_number):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "SELECT SenderAddress, Weight FROM courier WHERE TrackingNumber = %s"
    cursor.execute(query, (tracking_number,))
    result = cursor.fetchone()
    if not result:
        print(f"No shipment found with Tracking Number: {tracking_number}")
        return
    sender_address, weight = result
    query = """
        SELECT TrackingNumber FROM courier
        WHERE SenderAddress = %s AND Weight + %s <= 50 AND Status = 'Available'
        ORDER BY DeliveryDate ASC LIMIT 1
    """
    cursor.execute(query, (sender_address, weight))
    courier = cursor.fetchone()
    if courier:
        assigned_courier = courier[0]
        print(f"Shipment {tracking_number} assigned to Courier {assigned_courier} ")
    else:
        print(f"No available courier for shipment {tracking_number} ")
    cursor.close()
    conn.close()
tracking_number = input("Enter Tracking Number: ")
assign_courier(tracking_number)
```



Task 2: Loops and Iteration

Loops and iteration in Python allow you to execute a block of code multiple times. The for loop is used to iterate over a sequence like lists or strings, while the while loop continues as long as a condition is true. Python also provides control statements like break, continue, and pass to manage loop flow. These loops are essential for automating tasks and handling repetitive operations efficiently.

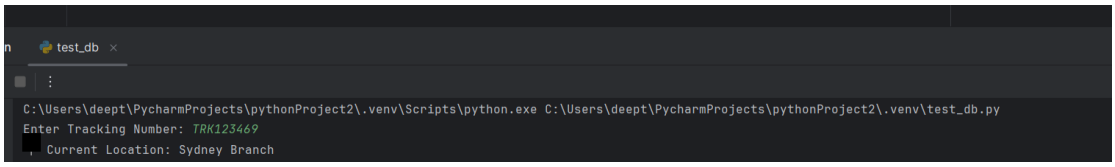
5. Write a python program that uses a for loop to display all the orders for a specific customer.

```
cursor = db.cursor()
def display_orders(user_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = """
    SELECT c.CourierID, c.Status, u.name
    FROM courier c
    JOIN user u ON c.UserID = u.UserID
    WHERE c.UserID = %s
    """
    cursor.execute(query, (user_id,))
    orders = cursor.fetchall()
    if orders:
```

```
        print(f"Orders for User ID {user_id}:")
        print("CourierID | Status | User Name")
        print("-" * 30)
        for order in orders:
            print(f"{order[0]} | {order[1]} | {order[2]}")
    else:
        print(f"No orders found for User ID {user_id}")
    cursor.close()
    conn.close()
user_id = input("Enter User ID: ")
display_orders(user_id)
```

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

```
cursor = db.cursor()
def track_courier(tracking_number):
    while True:
        query = "SELECT LocationName FROM location WHERE TrackingNumber = %s"
        cursor.execute(query, (tracking_number,))
        result = cursor.fetchone()
        if result:
            current_location = result[0]
            print(f" Current Location: {current_location}")
            if current_location.lower() == "delivered":
                print("The courier has reached its destination!")
                break
        else:
            print("Tracking number not found!")
tracking_number = input("Enter Tracking Number: ")
track_courier(tracking_number)
```



Task 3: Arrays and Data Structures

- **Arrays:**
  In Python, arrays are used to store multiple values in a single variable. Although the built-in `list` data type functions like an array, for more efficient numerical operations, the `array` module or third-party libraries like `NumPy` are used. Arrays store elements of the same data type and allow indexing, slicing, and looping for easy access and manipulation. They are especially useful in situations where performance and memory optimization are important.

- **Data Structures:**
  Python offers various built-in data structures such as lists, tuples, sets, and dictionaries. Lists are ordered and mutable, tuples are ordered but immutable, sets are unordered collections of unique items, and dictionaries store key-value pairs for fast data retrieval. These structures are the foundation for writing efficient, readable, and scalable Python programs.

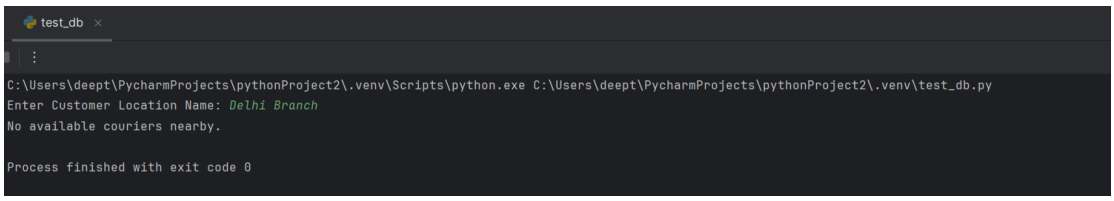
7. Create an array to store the tracking history of a parcel, where each entry represents a location update.

```
cursor = db.cursor()
def get_tracking_history(tracking_number):
    query = """
    SELECT locationname, address FROM location WHERE TrackingNumber = %s
    """
    cursor.execute(query, (tracking_number,))
    history = cursor.fetchall()
    if history:
        print("\nTracking History:")
        for entry in history:
            print(f"Location: {entry[0]}, Address: {entry[1]}")
    else:
        print("\nNo tracking history found for this number.")

tracking_number = input("Enter Tracking Number: ")
get_tracking_history(tracking_number)

cursor.close()
db.close()
```

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

```
cursor = conn.cursor()
def find_nearest_courier(customer_location):
    query = """
    SELECT c.CourierID, l.LOCATIONNAME, l.ADDRESS
    FROM COURIER c
    JOIN LOCATION l ON c.TrackingNumber = l.TrackingNumber
    WHERE l.LOCATIONNAME = %s AND c.STATUS = 'pending'
    ORDER BY c.CourierID LIMIT 1;
    """
    cursor.execute(query, (customer_location,))
    result = cursor.fetchone()
    if result:
        print(f"Nearest Available Courier: {result[0]}")
        print(f"Location: {result[1]}, Address: {result[2]}")
    else:
        print("No available couriers nearby.")
customer_location = input("Enter Customer Location Name: ")
```

```
find_nearest_courier(customer_location)
cursor.close()
conn.close()
```



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

Strings:

Strings in Python are sequences of characters enclosed in single or double quotes. They are immutable, meaning once created, they cannot be changed. Python provides many built-in string methods like .upper(), .lower(), .replace(), and .split() to manipulate text easily. Strings support indexing and slicing, allowing access to specific characters or substrings.

2D Arrays:

Two-dimensional arrays are arrays within arrays and are used to represent tabular data, like rows and columns. In Python, a 2D array can be created using nested lists (e.g., [[1, 2], [3, 4]]). Libraries like NumPy offer more efficient and powerful ways to work with 2D arrays using array operations and broadcasting.

User-Defined Functions:

User-defined functions in Python are blocks of reusable code that perform a specific task. They are created using the def keyword followed by the function name and parentheses. Functions can take parameters, perform operations, and return results using the return statement, helping make programs modular and clean.

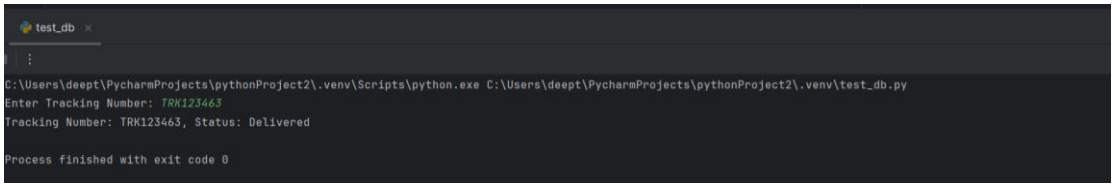
HashMap (Dictionaries in Python):

A HashMap in Python is implemented using the dict data type. It stores data as key-value pairs, allowing for fast lookups and updates. Each key must be unique, and values can be of any data type. Common operations include adding, updating, deleting elements, and retrieving values using keys.

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.

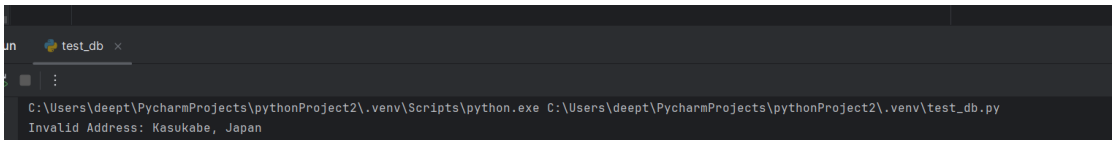
```
cursor = conn.cursor()

def track_parcel(tracking_number):
    query = "SELECT TrackingNumber, Status FROM COURIER WHERE TrackingNumber
= %s"
    cursor.execute(query, (tracking_number,))
    result = cursor.fetchone()
    if result:
        print(f"Tracking Number: {result[0]}, Status: {result[1]}")
```

```
    else:
        print("Invalid Tracking Number")

tracking_number = input("Enter Tracking Number: ")
track_parcel(tracking_number)

cursor.close()
conn.close()
```



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., ###-###-####).

```
cursor = conn.cursor()

def validate_customer_data():
    query = "SELECT Name, Address, ContactNumber FROM user"
    cursor.execute(query)
    results = cursor.fetchall()

    for name, address, contact in results:
        if not re.match(r"^[A-Za-z\s]+$", name):
            print(f"Invalid Name: {name}")
        if not re.match(r"^[A-Za-z0-9\s]+$", address):
            print(f"Invalid Address: {address}")
        if not re.match(r"^\d{10}$", contact):
            print(f"Invalid Contact Number: {contact}")
validate_customer_data()
cursor.close()
conn.close()
```



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.

```
cursor = conn.cursor()
def format_address_interactively():

    address = input("Enter an address: ")
    formatted_address = ' '.join([word.capitalize() for word in address.split()])
    print(f"Formatted Address: {formatted_address}")

format_address_interactively()
```

```
cursor.close()
conn.close()
```

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.

```
cursor = conn.cursor()

def generate_order_confirmation_email():
    order_id = input("Enter Order Number (Courier ID): ")
    print(f"Looking for Order Number: {order_id}")
    query = """
    SELECT u.Name, u.ContactNumber, u.Address, c.DeliveryDate
    FROM courier c
    JOIN user u ON c.UserID = u.UserID
    WHERE c.CourierID = %s
    """
    cursor.execute(query, (order_id,))
    result = cursor.fetchone()

    if result:
        name, contact_number, address, delivery_date = result
        email_content = f"""
        Dear {name},

        Thank you for your order with us. Below are the details of your order:

        Order Number: {order_id}
        Customer Name: {name}
        Contact Number: {contact_number}
        Delivery Address: {address}
        Expected Delivery Date: {delivery_date}

        We are processing your order, and you will be notified once it is out for delivery.

        If you have any questions or need assistance, please feel free to contact us.

        Best regards,
        Courier Management System
        """
        print(email_content)
    else:
        print("No order found with the provided order number.")
generate_order_confirmation_email()
cursor.close()
conn.close()
```

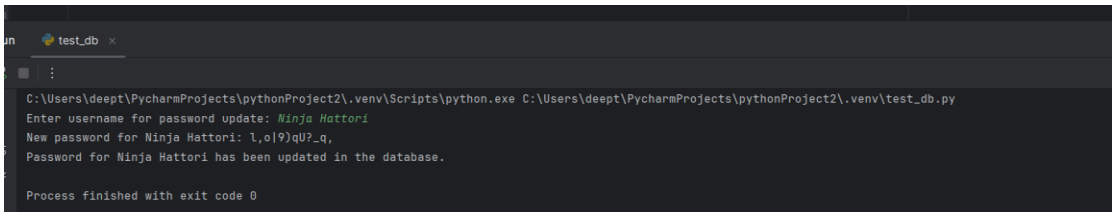
13. **Calculate Shipping Costs**: Develop a function that calculates the shipping cost based on the distance between two locations and the weight of the parcel. You can use string inputs for the source and destination addresses.

```
cursor = conn.cursor()
def calculate_shipping_cost():
    tracking_number = input("Enter the tracking number to fetch data: ")
    query = """
    SELECT senderaddress, receiveraddress, weight
    FROM courier
    WHERE trackingnumber = %s
    """
    cursor.execute(query, (tracking_number,))
    result = cursor.fetchone()
    if result:
        sender_address, receiver_address, weight = result
        distance_rate = 5
        weight_rate = 10
        distance = 100
        cost = (distance * distance_rate) + (weight * weight_rate)
        print(f"Shipping cost for parcel from {sender_address} to {receiver_address} (Weight:
{weight}kg) is: ₹{cost}")
    else:
        print("Tracking number not found in the database.")
calculate_shipping_cost()
cursor.close()
conn.close()
```

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.

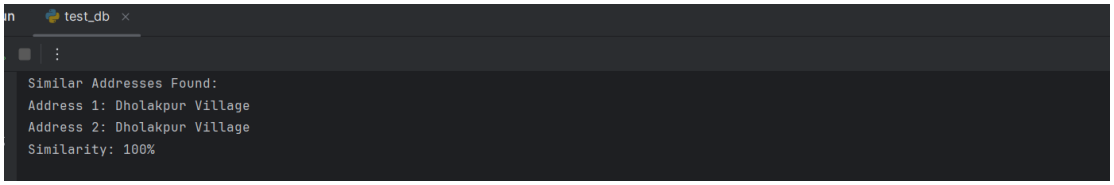
```
import mysql.connector
import random
```

```
import string
conn = mysql.connector.connect(host="localhost", user="root", password="root",
database="CouriersManagementSystem")
cursor = conn.cursor()
def generate_password():
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for i in range(12))  # Generates a 12-character
password
    return password
def store_password_in_db(username):
    password = generate_password()
    print(f"New password for {username}: {password}")
    query = "UPDATE user SET password = %s WHERE name = %s"
    cursor.execute(query, (password, username))
    conn.commit()
    print(f"Password for {username} has been updated in the database.")
username = input("Enter username for password update: ")
store_password_in_db(username)
cursor.close()
conn.close()
```



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.

```
import mysql.connector
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
conn = mysql.connector.connect(host="localhost", user="root", password="root",
database="CouriersManagementSystem")
cursor = conn.cursor()
def find_similar_addresses():
    query = "SELECT Address FROM user"
    cursor.execute(query)
    addresses = cursor.fetchall()
    if len(addresses) < 2:
        print("Not enough addresses to compare.")
        return
    for i in range(len(addresses)):
        for j in range(i + 1, len(addresses)):
            addr1 = addresses[i][0]
            addr2 = addresses[j][0]
            similarity = fuzz.ratio(addr1, addr2)
            if similarity > 80:
                print(f"Similar Addresses Found:")
                print(f"Address 1: {addr1}")
                print(f"Address 2: {addr2}")
                print(f"Similarity: {similarity}%\n")
find_similar_addresses()
cursor.close()
conn.close()
```



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())

- **Entity Classes/Models/POJO:**

These classes represent real-world objects like User, Courier, or Payment. They contain attributes (variables) and getters/setters to access or modify the data, following the Plain Old Python Object (POPO) style.

- **Abstraction/Encapsulation:**

Abstraction hides complex implementation details and shows only the necessary features, while encapsulation protects the data by bundling it with methods and restricting direct access through private variables and getter/setter methods.

1. User Class:
Variables:
userID , userName , email , password , contactNumber , address

- Creating a folder named entities
- Creating a file named User under the folder entities (User.py)

```
class User:
   def _init_(self, userID=None, userName=None, email=None, password=None,
contactNumber=None, address=None):
      self.__userID = userID
      self.__userName = userName
      self.__email = email
      self.__password = password
      self.__contactNumber = contactNumber
      self.__address = address

    def get_userID(self):
      return self.__userID

   def set_userID(self, userID):
      self.__userID = userID

   def get_userName(self):
      return self.__userName

   def set_userName(self, userName):
      self.__userName = userName

   def get_email(self):
      return self.__email
```

```
    def set_email(self, email):
        self.__email = email

    def get_password(self):
        return self.__password

    def set_password(self, password):
        self.__password = password

    def get_contactNumber(self):
        return self.__contactNumber

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def get_address(self):
        return self.__address

    def set_address(self, address):
        self.__address = address

    def _str_(self):
        return f"User(userID={self._userID}, userName={self.userName}, email={self._email}, " \
               f"contactNumber={self._contactNumber}, address={self._address})"
```

2. Courier Class

Variables: courierID , senderName , senderAddress , receiverName , receiverAddress , weight , status, trackingNumber , deliveryDate ,userId

- Creating a file named Courier under the folder entities (Courier.py)

```
class Courier:
    def _init_(self, courierID=None, senderName=None, senderAddress=None,
receiverName=None, receiverAddress=None,
               weight=None, status=None, trackingNumber=None, deliveryDate=None, userID=None):
        self.__courierID = courierID
        self.__senderName = senderName
        self.__senderAddress = senderAddress
        self.__receiverName = receiverName
        self.__receiverAddress = receiverAddress
        self.__weight = weight
        self.__status = status
        self.__trackingNumber = trackingNumber
        self.__deliveryDate = deliveryDate
        self.__userID = userID

    def get_courierID(self):
        return self.__courierID

    def set_courierID(self, courierID):
        self.__courierID = courierID

    def get_senderName(self):
        return self.__senderName

    def set_senderName(self, senderName):
        self.__senderName = senderName

    def get_senderAddress(self):
```

```
        return self.__senderAddress

    def set_senderAddress(self, senderAddress):
        self.__senderAddress = senderAddress

    def get_receiverName(self):
        return self.__receiverName

    def set_receiverName(self, receiverName):
        self.__receiverName = receiverName

    def get_receiverAddress(self):
        return self.__receiverAddress

    def set_receiverAddress(self, receiverAddress):
        self.__receiverAddress = receiverAddress

    def get_weight(self):
        return self.__weight

    def set_weight(self, weight):
        self.__weight = weight

    def get_status(self):
        return self.__status

    def set_status(self, status):
        self.__status = status

    def get_trackingNumber(self):
        return self.__trackingNumber

    def set_trackingNumber(self, trackingNumber):
        self.__trackingNumber = trackingNumber

    def get_deliveryDate(self):
        return self.__deliveryDate

    def set_deliveryDate(self, deliveryDate):
        self.__deliveryDate = deliveryDate

    def get_userID(self):
        return self.__userID

    def set_userID(self, userID):
        self.__userID = userID

    def _str_(self):
        return f"Courier(courierID={self._courierID}, senderName={self.senderName},
senderAddress={self._senderAddress}, " \
               f"receiverName={self._receiverName}, receiverAddress={self.receiverAddress},
weight={self._weight}, " \
               f"status={self._status}, trackingNumber={self.trackingNumber},
deliveryDate={self._deliveryDate}, " \
               f"userID={self.__userID})"
```

3. Employee Class:

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

- Creating a file named Employee under the folder entities (Employee.py)

```
class Employee:
    def _init_(self, employeeID=None, employeeName=None, email=None, contactNumber=None,
role=None, salary=None):
        self.__employeeID = employeeID
        self.__employeeName = employeeName
        self.__email = email
        self.__contactNumber = contactNumber
        self.__role = role
        self.__salary = salary

    def get_employeeID(self):
        return self.__employeeID

    def set_employeeID(self, employeeID):
        self.__employeeID = employeeID

    def get_employeeName(self):
        return self.__employeeName

    def set_employeeName(self, employeeName):
        self.__employeeName = employeeName

    def get_email(self):
        return self.__email

    def set_email(self, email):
        self.__email = email

    def get_contactNumber(self):
        return self.__contactNumber

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def get_role(self):
        return self.__role

    def set_role(self, role):
        self.__role = role

    def get_salary(self):
        return self.__salary

    def set_salary(self, salary):
        self.__salary = salary

    def _str_(self):
        return f"Employee(employeeID={self._employeeID}, employeeName={self.employeeName},
email={self._email}, " \
               f"contactNumber={self._contactNumber}, role={self.role}, salary={self._salary})"
```

4. Location Class

Variables LocationID , LocationName , Address

- Creating a file named Location under the folder entities (Location.py)

```
class Location:
```

```
    def _init_(self, LocationID=None, LocationName=None, Address=None):
        self.__LocationID = LocationID
        self.__LocationName = LocationName
        self.__Address = Address
    def get_LocationID(self):
        return self.__LocationID

    def set_LocationID(self, LocationID):
        self.__LocationID = LocationID

    def get_LocationName(self):
        return self.__LocationName

    def set_LocationName(self, LocationName):
        self.__LocationName = LocationName

    def get_Address(self):
        return self.__Address

    def set_Address(self, Address):
        self.__Address = Address

    def _str_(self):
        return f"Location(LocationID={self._LocationID}, LocationName={self.LocationName},
Address={self._Address})"
```

5. CourierCompany Class

Variables companyName , courierDetails -collection of Courier Objects, employeeDetailscollection of Employee Objects, locationDetails - collection of Location Objects.

- Creating a file named CourierCompany under the folder entities (CourierCompany.py)

```
class CourierCompany:
    def _init_(self, companyName=None, courierDetails=None, employeeDetails=None,
locationDetails=None):
        self.__companyName = companyName
        self.__courierDetails = courierDetails if courierDetails else []
        self.__employeeDetails = employeeDetails if employeeDetails else []
        self.__locationDetails = locationDetails if locationDetails else []

    def get_companyName(self):
        return self.__companyName

    def set_companyName(self, companyName):
        self.__companyName = companyName

    def get_courierDetails(self):
        return self.__courierDetails

    def set_courierDetails(self, courierDetails):
        self.__courierDetails = courierDetails

    def get_employeeDetails(self):
        return self.__employeeDetails

    def set_employeeDetails(self, employeeDetails):
```

```
        self.__employeeDetails = employeeDetails

    def get_locationDetails(self):
        return self.__locationDetails

    def set_locationDetails(self, locationDetails):
        self.__locationDetails = locationDetails

    def _str_(self):
        return f"CourierCompany(companyName={self.__companyName}, " \
               f"courierDetails={self._courierDetails}, employeeDetails={self._employeeDetails}, " \
               f"locationDetails={self.__locationDetails})"
```

6. Payment Class:

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

- Creating a file named Payment under the folder entities (Payment.py)

```
class Payment:
    def _init_(self, PaymentID=None, CourierID=None, Amount=None, PaymentDate=None):
        self.__PaymentID = PaymentID
        self.__CourierID = CourierID
        self.__Amount = Amount
        self.__PaymentDate = PaymentDate

    def get_PaymentID(self):
        return self.__PaymentID

    def set_PaymentID(self, PaymentID):
        self.__PaymentID = PaymentID

    def get_CourierID(self):
        return self.__CourierID

    def set_CourierID(self, CourierID):
        self.__CourierID = CourierID

    def get_Amount(self):
        return self.__Amount

    def set_Amount(self, Amount):
        self.__Amount = Amount

    def get_PaymentDate(self):
        return self.__PaymentDate

    def set_PaymentDate(self, PaymentDate):
        self.__PaymentDate = PaymentDate

    def _str_(self):
        return f"Payment(PaymentID={self._PaymentID}, CourierID={self._CourierID}, " \
               f"Amount={self._Amount}, PaymentDate={self._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
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.
*/

- **Abstract Class:**
  An abstract class in Python acts as a blueprint for other classes. It can contain both abstract methods (without implementation) and concrete methods (with implementation). It ensures that all subclasses implement specific methods.
- **Service Provider Interface:**
  This interface defines a set of services that must be provided by the implementing classes. It promotes consistency and standardizes method signatures, making it easier to manage and scale the application's services.

Creating **abstract base classes** (interfaces in Python) for the service classes as specified. We'll implement two classes:

- **ICourierUserService** (for customer-related functions).
- **ICourierAdminService** (for admin-related functions).

Step 1: Creating the Interface Files

- Creating a new Python file named icourieruserservice.py under entities folder

```
from abc import ABC, abstractmethod

class ICourierUserService(ABC):
    @abstractmethod
    def placeOrder(self, courierObj):
        pass

    @abstractmethod
    def getOrderStatus(self, trackingNumber):
        pass

    @abstractmethod
    def cancelOrder(self, trackingNumber):
        pass

    @abstractmethod
    def getAssignedOrder(self, courierStaffId):
        pass
```

- Creating a new Python file named icourieradminservice.py under entities folder

```
from abc import ABC, abstractmethod

class ICourierAdminService(ABC):
    @abstractmethod
    def addCourierStaff(self, name, contactNumber):
        pass
```

Step 2: Implementing the Concrete Classes

- Creating a new Python file named courieruserservice.py under entities folder

```
from entities.icourieruserservice import ICourierUserService
from entities.courier import Courier

class CourierUserService(ICourierUserService):
    tracking_number = 1000

    def placeOrder(self, courierObj):
        self.tracking_number += 1
        print(f"Order placed with Tracking Number: {self.tracking_number}")
        return self.tracking_number

    def getOrderStatus(self, trackingNumber):
        print(f"Fetching status for order {trackingNumber}")
        return "In Transit"

    def cancelOrder(self, trackingNumber):
        print(f"Order with tracking number {trackingNumber} has been cancelled.")
        return True

    def getAssignedOrder(self, courierStaffId):
        print(f"Fetching orders assigned to courier staff with ID {courierStaffId}")
        return ["Order1", "Order2", "Order3"]
```

- Creating a new Python file named courieradminservice.py under entities folder

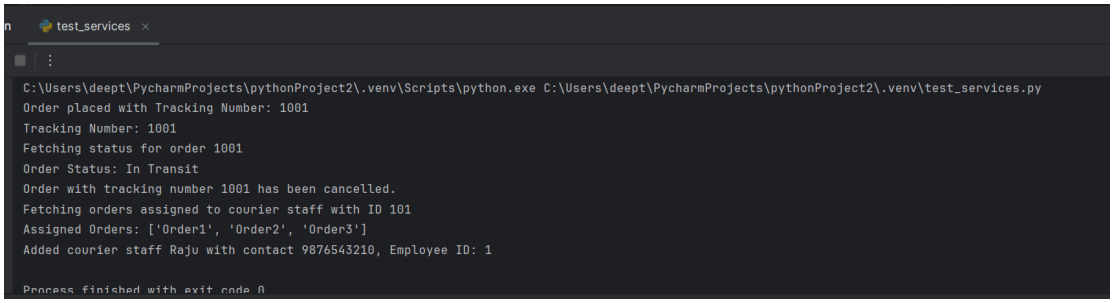
```
from entities.icourieradminservice import ICourierAdminService
from entities.employee import Employee

class CourierAdminService(ICourierAdminService):
    def addCourierStaff(self, name, contactNumber):
        employee_id = 101
        print(f"Added courier staff {name} with contact {contactNumber}, Employee ID:
{employee_id}")
        return employee_id
```

Step 3: Testing the Implementation

- Creating a new Python file named test_services.py under dao folder

```
from entities.courieruserservice import CourierUserService
from entities.courieradminservice import CourierAdminService

user_service = CourierUserService()
courier_obj = {"sender": "Raju", "receiver": "Jane", "address": "123 Street", "weight": 5}

tracking_number = user_service.placeOrder(courier_obj)
print("Tracking Number:", tracking_number)

status = user_service.getOrderStatus(tracking_number)
print("Order Status:", status)

user_service.cancelOrder(tracking_number)

assigned_orders = user_service.getAssignedOrder(101)
print("Assigned Orders:", assigned_orders)

admin_service = CourierAdminService()

admin_service.addCourierStaff("Alice", "9876543210")
```

Running the test_services.py file give the following output



Task 7: Exception Handling
(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch,finally,thow & throws keyword usage)

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

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

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

- Exception handling in Python is used to manage errors gracefully without crashing the program. It is done using try, except, else, and finally blocks. The try block contains the code that may raise an exception. If an error occurs, the except block handles it. The else block runs if no exceptions occur, and finally runs regardless of the result, often used for cleanup actions like closing files or connections. This ensures the program remains stable and user-friendly.

- Creating a new Python file named exceptions.py under entities folder

```
class TrackingNumberNotFoundException(Exception):
    """Exception raised when the tracking number is not found."""
    def __init__(self, message="Tracking number not found!"):
        self.message = message
        super().__init__(self.message)

class InvalidEmployeeIdException(Exception):
    """Exception raised when an invalid employee ID is entered."""
    def __init__(self, message="Invalid employee ID!"):
        self.message = message
        super().__init__(self.message)
```

- Creating a new Python file named courier_service.py under entities folder

```
from entities.exceptions import TrackingNumberNotFoundException,
InvalidEmployeeIdException
class CourierService:

    def __init__(self):
        self.employees = {101: "John Doe", 102: "Jane Smith"}  # Simulated employee records
        self.tracking_numbers = {"TRK123456": "In Transit", "TRK123457": "Delivered"}  #
Simulated tracking numbers

    def withdraw(self, tracking_number):
        """Simulate withdrawing an amount or accessing courier details."""
        if tracking_number not in self.tracking_numbers:
            raise TrackingNumberNotFoundException(f"Tracking number {tracking_number} not
found!")
        return f"Status of {tracking_number}: {self.tracking_numbers[tracking_number]}"

    def transfer(self, employee_id):
        """Simulate transferring amount or accessing employee data."""
        if employee_id not in self.employees:
            raise InvalidEmployeeIdException(f"Employee ID {employee_id} not found!")
        return f"Employee found: {self.employees[employee_id]}"
```

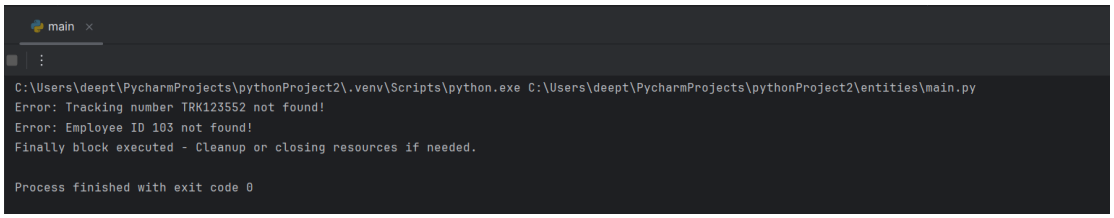
- Creating a new Python file named main.py under entities folder

This file will use the CourierService and handle the exceptions using try, except, and finally blocks.

```
from entities.courier_service import CourierService
```

```
from entities.exceptions import TrackingNumberNotFoundException,
InvalidEmployeeIdException

def main():
    service = CourierService()

    try:
        tracking_number = "TRK123552"
        print(service.withdraw(tracking_number))
    except TrackingNumberNotFoundException as e:
        print(f"Error: {e}")
    try:
        employee_id = 103
        print(service.transfer(employee_id))
    except InvalidEmployeeIdException as e:
        print(f"Error: {e}")

    finally:
        print("Finally block executed - Cleanup or closing resources if needed.")


if __name__ == "__main__":
    main()
```

Running the main.py produces the following output



Task 8: Collections
Scope: ArrayList/Hashmap
Task: Improve the Courier Management System by using python 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
2. Create a new implementation class **CourierUserServiceCollectionImpl** class in package **dao** which implements **ICourierUserService** interface which holds a variable named companyObj of type **CourierCompanyCollection**

- Creating a new Python file named CourierCompanyCollection.py under entities folder

```
from entities.Courier import Courier
from entities.Employee import Employee
from entities.Location import Location

class CourierCompanyCollection:
    def __init__(self, companyName):
        self.companyName = companyName
        self.courierDetails = []
```

```
        self.employeeDetails = []
        self.locationDetails = []

    def add_courier(self, courier):
        self.courierDetails.append(courier)

    def add_employee(self, employee):
        self.employeeDetails.append(employee)

    def add_location(self, location):
        self.locationDetails.append(location)

    def get_couriers(self):
        return self.courierDetails

    def get_employees(self):
        return self.employeeDetails

    def get_locations(self):
        return self.locationDetails

    def __str__(self):
        return f"Courier Company: {self.companyName}, Couriers: {len(self.courierDetails)},
Employees: {len(self.employeeDetails)}, Locations: {len(self.locationDetails)}"
```

- Creating a new Python file named CourierUserServiceCollectionImpl.py under dao folder

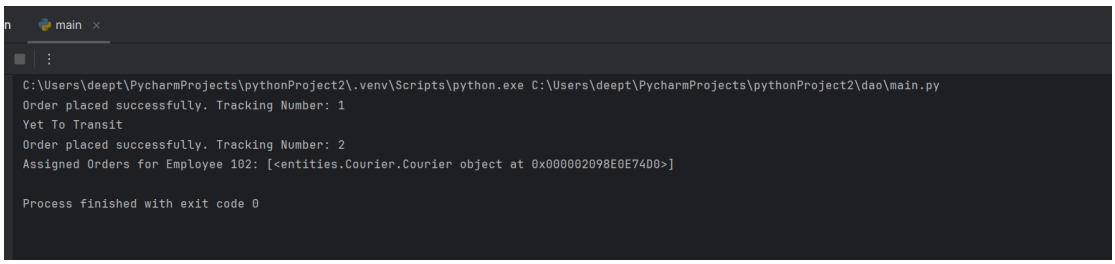
```
from entities.CourierCompanyCollection import CourierCompanyCollection
from entities.Courier import Courier

class CourierUserServiceCollectionImpl:
    def __init__(self, company_name):
        self.companyObj = CourierCompanyCollection(company_name)

    def place_order(self, courier_obj):
        # Generate a unique tracking number (for simplicity, using length of couriers list)
        tracking_number = len(self.companyObj.get_couriers()) + 1
        courier_obj.tracking_number = tracking_number
        self.companyObj.add_courier(courier_obj)
        print(f"Order placed successfully. Tracking Number: {tracking_number}")
        return tracking_number

    def get_order_status(self, tracking_number):
        for courier in self.companyObj.get_couriers():
            if courier.tracking_number == tracking_number:
                return courier.status
        return "Tracking number not found."

    def cancel_order(self, tracking_number):
        for courier in self.companyObj.get_couriers():
            if courier.tracking_number == tracking_number:
                courier.status = "Canceled"
                return True
        return False

    def get_assigned_order(self, courier_staff_id):

        assigned_orders = []
```

```
        for courier in self.companyObj.get_couriers():
            if courier.employee_id == courier_staff_id:
                assigned_orders.append(courier)
        return assigned_orders
```

- Creating a new Python file named main.py under dao folder

```
from dao.CourierUserServiceCollectionImpl import CourierUserServiceCollectionImpl
from entities.Courier import Courier

company_service = CourierUserServiceCollectionImpl("FastExpress")
courier1 = Courier("John Doe", "123 Elm St", "Jane Smith", "456 Oak St", 5, employee_id=101)

tracking_number1 = company_service.place_order(courier1)
print(company_service.get_order_status(tracking_number1))

courier2 = Courier("Alice Brown", "789 Pine St", "Bob White", "101 Maple St", 3,
employee_id=102)
tracking_number2 = company_service.place_order(courier2)


assigned_orders = company_service.get_assigned_order(102)
print("Assigned Orders for Employee 102:", assigned_orders)
```

Running the main.py produces the following output



Task 9: 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.
2. Create **CourierAdminService Impl** class which inherits from **CourierUserServiceImpl** and
implements **ICourierAdminService** interface.
3. Create **CourierAdminServiceCollectionImpl** class which inherits from **CourierUserServiceColectionImpl** and implements **ICourierAdminService** interface.

- Creating a new Python file named CourierUserServiceImpl.py under dao folder

```
from entities.icourieruserservice import ICourierUserService
from entities.CourierCompany import CourierCompany
from entities.Courier import Courier

class CourierUserServiceImpl(ICourierUserService):
```

```
    def __init__(self, company_name):
        self.companyObj = CourierCompany(company_name)

    def place_order(self, courierObj):
        """Place a new courier order."""
        tracking_number = courierObj.tracking_number or
f"TN{len(self.companyObj.courierDetails) + 1}"
        courierObj.tracking_number = tracking_number
        self.companyObj.courierDetails.append(courierObj)
        return tracking_number

    def get_order_status(self, tracking_number):
        """Get the status of a courier order."""
        for courier in self.companyObj.courierDetails:
            if courier.tracking_number == tracking_number:
                return courier.status
        return "Order not found"

    def cancel_order(self, tracking_number):
        """Cancel a courier order."""
        for courier in self.companyObj.courierDetails:
            if courier.tracking_number == tracking_number:
                courier.status = "Cancelled"
                return True
        return False

    def get_assigned_order(self, employee_id):
        """Get a list of orders assigned to a specific courier staff member."""
        assigned_orders = []
        for courier in self.companyObj.courierDetails:
            if courier.employee_id == employee_id:
                assigned_orders.append(courier)
        return assigned_orders
```

- Creating a new Python file named CourierAdminServiceImpl.py under dao folder

```
from dao.CourierUserServiceImpl import CourierUserServiceImpl
from entities.icourieradminservice import ICourierAdminService
from entities.Employee import Employee

class CourierAdminServiceImpl(CourierUserServiceImpl, ICourierAdminService):
    def __init__(self, company_name):
        super().__init__(company_name)

    def add_courier_staff(self, name, contact_number):
        """Add a new courier staff member to the system."""
        new_staff_id = len(self.companyObj.employeeDetails) + 1
        new_employee = Employee(new_staff_id, name, "email@example.com", contact_number,
"Courier", 30000)
        self.companyObj.employeeDetails.append(new_employee)
        return new_staff_id
```

- Creating a new Python file named CourierAdminServiceCollectionImpl.py under dao folder

```
from entities.CourierUserServiceCollectionImpl import CourierUserServiceCollectionImpl
from entities.ICourierAdminService import ICourierAdminService
from entities.Employee import Employee
```

```
class CourierAdminServiceCollectionImpl(CourierUserServiceCollectionImpl,
ICourierAdminService):
    def __init__(self, company_name):
        super().__init__(company_name)

    def add_courier_staff(self, name, contact_number):
        """Add a new courier staff member to the system."""
        new_staff_id = len(self.companyObj.employeeDetails) + 1
        new_employee = Employee(new_staff_id, name, "email@example.com", contact_number,
"Courier", 35000)
        self.companyObj.employeeDetails.append(new_employee)
        return new_staff_id
```

Task 10: 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.

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.

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

- Creating a folder named connectionutil.Inside it, creating a Python file named DBConnection.py.
- Inside DBConnection.py, Defining a class DBConnection.Creating a static variable called connection.Defining a static method getConnection()
- Creating the db.properties file in config folder.

```
host=localhost
user=root
password=root
database=CourierManagementSystem
```

- Creating the DBConnection class

```
import mysql.connector
from configparser import ConfigParser
import os

class DBConnection:
    connection = None

    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            config = ConfigParser()
            config.read(os.path.join("config", "db.properties"))

            DBConnection.connection = mysql.connector.connect(
                host=config.get("DEFAULT", "host"),
```

```
            user=config.get("DEFAULT", "user"),
            password=config.get("DEFAULT", "password"),
            database=config.get("DEFAULT", "database")
        )
```

- Creating the main.py and running the given code

```
from connectionutil.DBConnection import DBConnection
conn = DBConnection.getConnection()
if conn.is_connected():
    print("Connected to database successfully!")
else:
    print("Connection failed.")
```

```
C:\Users\deept\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\deept\PycharmProjects\pythonProject2\.venv\test_db.py
Connected to database successfully!

Process finished with exit code 0
```

- Creating the python file CourierServiceDb.py in the dao folder

```
from connectionutil.DBConnection import DBConnection

class CourierServiceDb:
    connection = None

    def __init__(self):
        if CourierServiceDb.connection is None:
            CourierServiceDb.connection = DBConnection.getConnection()
        self.cursor = CourierServiceDb.connection.cursor()
```

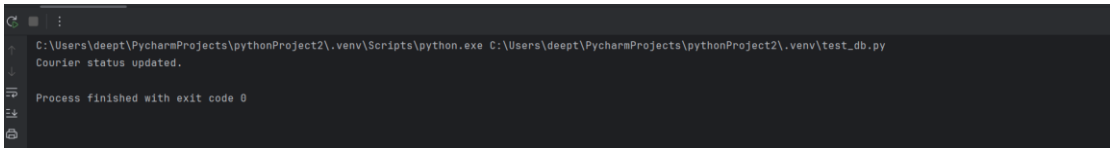
- Inserting a new courier/order

```
    def insert_order(self, sender, receiver, source, destination, status):
        sql = """INSERT INTO courier (sender, receiver, source, destination, status)
                 VALUES (%s, %s, %s, %s, %s)"""
        values = (sender, receiver, source, destination, status)
        self.cursor.execute(sql, values)
        CourierServiceDb.connection.commit()
        print("Order inserted successfully.")
```

```
C:\Users\deept\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\deept\PycharmProjects\pythonProject2\.venv\test_db.py
Order inserted successfully!

Process finished with exit code 0
```

- Updating the courier status

```
    def update_status(self, courier_id, new_status):
        sql = "UPDATE courier SET status = %s WHERE id = %s"
        values = (new_status, courier_id)
        self.cursor.execute(sql, values)
        CourierServiceDb.connection.commit()
        print("Courier status updated.")
```



Conclusion

The Courier Management System project successfully demonstrates the integration of database-driven backend operations with Python-based service logic to efficiently manage courier-related tasks. From designing a robust SQL schema with essential tables like User, Courier, Employee, Location, and Payment, to implementing service layers and database connectivity using PyCharm and MySQL, the system is built to simulate real-world courier operations.

The use of Python concepts such as object-oriented programming, collections, loops, functions, and exception handling ensures modularity and maintainability. SQL functionalities like joins, subqueries, and aggregate functions enhance data retrieval and reporting capabilities. By utilizing tools like GitHub for version control and PyCharm for development, this project not only fulfills the technical requirements but also equips learners with practical industry-level exposure.Therefore, the Courier Management System stands as a comprehensive solution for tracking, managing, and analyzing courier operations with accuracy and efficiency.