

GROUP – 4

Topic: Database Management System for UPS Deliveries

College of Business, California State University, Long Beach

Course: IS680 Database Management Systems Sec 02 11140 - 2024

Instructor: Prof. Jose Pineda

Team:

1. Vishakha Maruti Sonmore
2. Saumya Chandrakant Prasad
3. Shloka Daga
4. Prithviraju Venkataraman
5. Bommalapura Panjaru Shashidhar Reddy

1. INTRODUCTION

1.1. United Parcel Service (UPS)

“The business model is changing, and our focus is changing. Our customers are demanding broader, deeper services, whether that’s running telephone centres for them or doing their billing for them or going out and repairing and replacing their computer parts for them. We’re doing all kinds of things we didn’t dream of doing ten years ago.” —Jim Kelly, CEO and Chairman of the Board, UPS

Jim Kelly, CEO of United Parcel Service (UPS)

A company best known for reliable, efficient package delivery, was commenting in late 2000 on how his company had become far more than a package delivery company. For a decade, UPS had been capitalizing on opportunities created by information technology to increase the efficiency and flexibility of its core business.

In the ever-evolving landscape of logistics and delivery services, United Parcel Service (UPS) stands as a titan, renowned for its global reach and steadfast commitment to efficiency. While the focus of delivery services often centres around the transport of goods and packages, the intricacies of managing data and optimizing operations are paramount for sustained success.

1.2. Statement of Problem

As UPS continues to expand its operations and adapt to the evolving demands of the modern world, the challenges of managing vast amounts of data become increasingly apparent. With a multitude of packages, routes, and customer preferences to account for, the need for streamlined data management solutions becomes imperative.

This project endeavours to address the burgeoning complexities faced by UPS by developing a comprehensive database application tailored to its specific needs. At its core, this endeavour seeks to streamline data storage, retrieval, and analysis processes, empowering UPS to operate with unparalleled efficiency and precision.

1.3. Purpose

The database application will encompass a multifaceted approach, incorporating elements such as conceptual modelling, logical design, and a robust data dictionary. By meticulously cataloguing information ranging from customer profiles to delivery details, the system will serve as a cornerstone for UPS's operational infrastructure.

Key functionalities of the database will include the ability to manage customer preferences, track package movements in real-time, analyse delivery routes for optimization, and generate insightful reports to inform strategic decision-making.

Through the implementation of this innovative database application, UPS aims to revolutionize its data management practices, fostering enhanced operational agility and customer satisfaction. By leveraging cutting-edge technology to streamline processes and unlock actionable insights, UPS reaffirms its commitment to excellence in the realm of logistics and delivery services.

1.4. Aim/Objective

The primary aim of our database is to streamline and optimize the logistical operations of United Parcel Service (UPS). It needs to centralize and manage vast amounts of data related to package tracking, customer information, delivery routes, and operational performance. By doing so, it aims to enhance efficiency, accuracy, and customer satisfaction while reducing operational costs and errors. Ultimately, the success of the database hinges on its ability to facilitate smoother operations, provide actionable insights, and contribute to UPS's overarching goal of delivering excellence in logistics services.

Specifically, the project involves:

- Identification of the problems being experienced at United Parcel Services.
- Design of a database system that will overcome the identified problems and enhance efficiency in service delivery as well as database management system timely decision making.
- Implement the developed system.

1.5. Users and Benefits

The database's users comprise a range of UPS stakeholders, including as but not restricted to:

- **Logistics managers:** They follow delivery routes, keep an eye on package movements, and streamline logistical procedures using the database. Real-time package locations, delivery statuses, route efficiency indicators, and performance analytics are among the essential data they need.
- **Customer service representatives:** They use the database to respond to questions from clients, handle delivery problems, and give precise status updates on shipments. Having access to delivery history, feedback data, and customer profiles is crucial for providing outstanding customer service.

- **Data analysts:** They use the database to get insightful information, analyse performance, and provide reports that help with strategic decision-making. This entails assessing delivery patterns, pinpointing regions in need of development, and projecting demand trends going forward.
- **IT Administrators:** IT administrators are responsible for managing and maintaining UPS's IT infrastructure, including hardware, software, networks, and security systems. They ensure the reliability, availability, and security of UPS's systems and applications. IT administrators interact with UPS's systems to perform tasks such as system configuration, software updates, security patches, and troubleshooting technical issues. They monitor system performance, implement data backup and recovery procedures, and enforce security policies to protect UPS's data and infrastructure from cyber threats and vulnerabilities.
- **Delivery Drivers:** Delivery drivers are responsible for transporting packages from distribution centres to customers' doorsteps. They follow predefined routes, deliver packages within designated timeframes, provide exceptional service to UPS's customers. Delivery drivers may interact with UPS's systems through mobile devices or handheld scanners to receive delivery instructions, update package statuses, and report any delivery exceptions or issues. They rely on the system to access real-time navigation tools, delivery manifests, and customer instructions to ensure accurate and efficient delivery of packages.

1.6. Inputs and Information Management

Inputs to the database encompass a wide array of data points collected throughout the logistics process. These include:

- Package tracking events: Information Regarding Package Shipment, Sender, Tracking, Company, Transaction.
- Customer information: Recipient, Customer.
- Delivery routes and schedules: Employee, Vehicle, Warehouse, Location.

1.7. Information Stored in the Database

The database will need to store a comprehensive range of information to support UPS's logistical operations effectively. This includes:

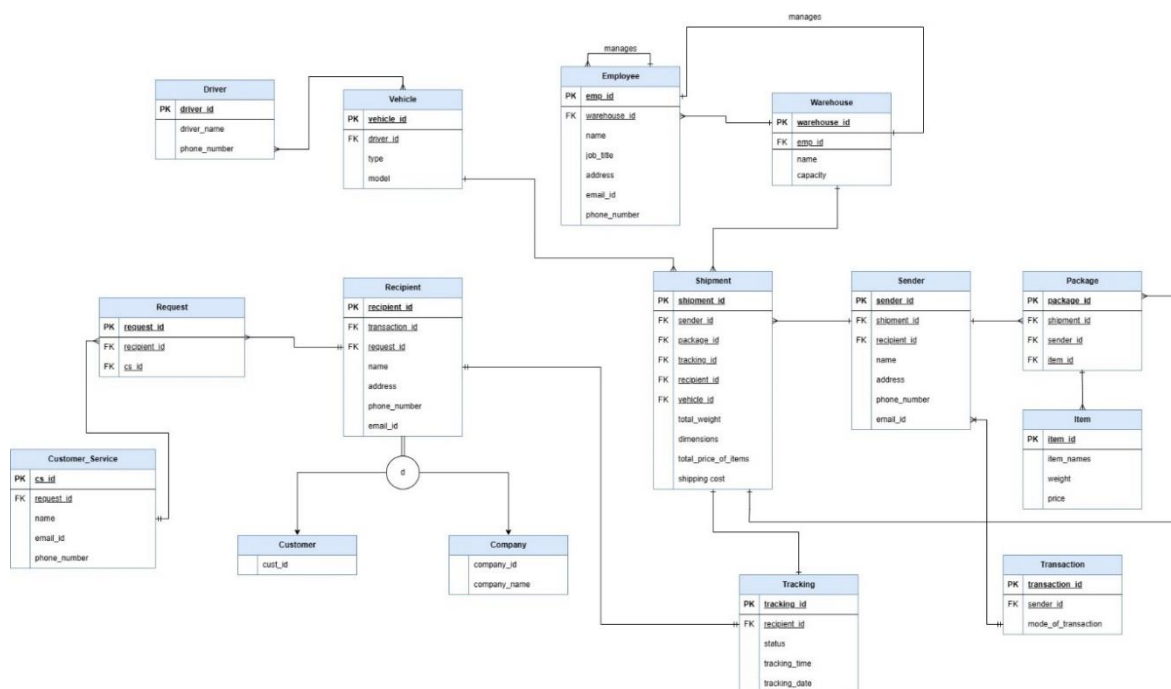
- Customer Information: Name, contact details, delivery preferences, billing information.

- Package Data: Tracking numbers, sender and recipient details, package dimensions, and weight.
- Operational Metrics: Delivery times, transit durations, status.

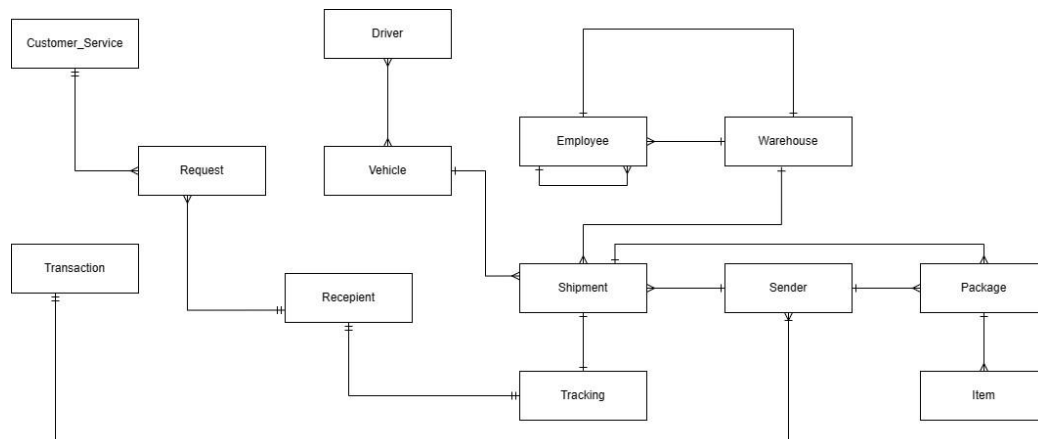
UPS's database will be a strong framework for improving customer experiences, streamlining logistical processes, and fostering operational excellence by methodically organizing and maintaining this abundance of data.

2. ER DIAGRAM & CONCEPTUAL MODEL

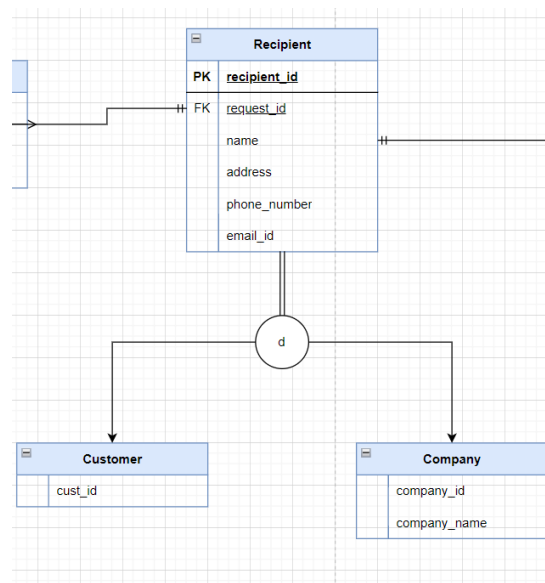
Creating an Entity-Relationship (ER) diagram for United Parcel Service (UPS) could involve identifying the main entities, their attributes, and the relationships between them. Here's a simplified ER diagram for UPS:



For the CONCEPTUAL model of the United Parcel Service (UPS) data management system, we have identified 15 key entities with appropriate relationships. To ensure clarity and efficacy, we've utilized various relationship types, including mandatory one, mandatory many, and optional many, to establish robust connections between entities.



At the core of our model lies the **Shipment** entity, serving as the central hub for UPS's logistical operations. **Shipment** acts as the linchpin, orchestrating the flow of data throughout the system. For instance, the relationship between the **Sender** entity and the **Shipment** & **Packages** entity is designated as mandatory one, signifying that every package must be associated with a corresponding **Sender** or **Receptient**. Likewise, the relationship between the **Shipment** entity and the **Tracking** entity is also mandatory one, ensuring that all packages are linked to a predefined **Tracking** for efficient transportation.



We've incorporated the Supertype-Subtype Identity transform to establish an identifying relationship between a supertype entity and its subtype entities, aiming to streamline the model and enhance query performance. Within this framework, two types of constraints govern the supertype/subtype relationships.

Firstly, the completeness constraint dictates whether there must be a distinct subtype for every conceivable instance of the supertype. Secondly, the Disjointness constraint determines whether a given instance of the supertype could potentially belong to multiple subtypes simultaneously.

For instance, in the context of a **Recipient** entity, the method must be either categorized as "**customer**" or "**company**," or in both - using a representation of double lines descending from the supertype entity type, which indicates Disjoint.

3. LOGICAL MODEL

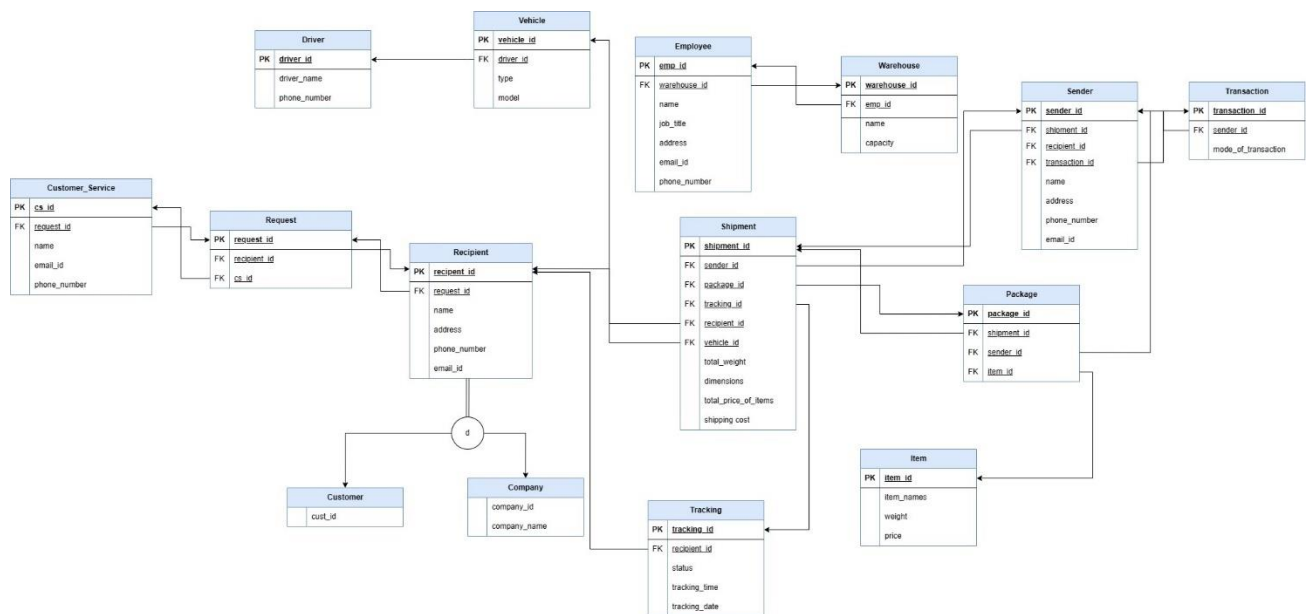
A logical model describes the conceptual model developed above in as much detail as possible. Each entity's primary keys, foreign keys, and attributes are listed in the logical model for the UPS delivery business.

Entities, Attribute, Primary keys & Foreign key relationships:

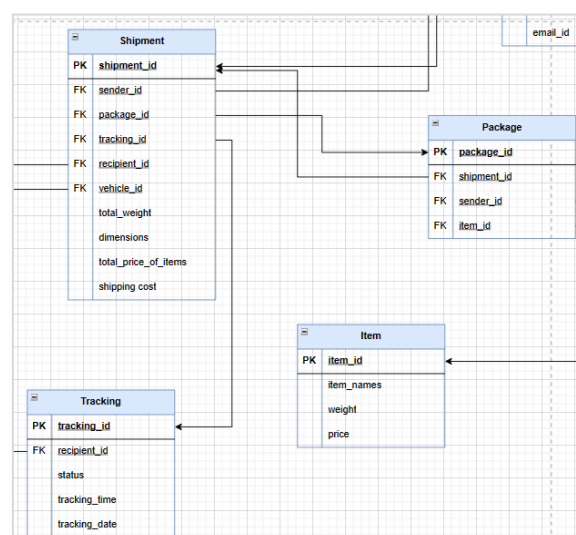
<ul style="list-style-type: none"> • Packages Entity <ul style="list-style-type: none"> ➤ Package_ID (Primary Key) ➤ Shipment_ID (Foreign Key) ➤ Sender_ID (Foreign Key) ➤ Item_ID (Foreign Key) 	<ul style="list-style-type: none"> • Warehouse Entity <ul style="list-style-type: none"> ➤ WarehouseID (Primary Key) ➤ Emp_ID (Foreign Key) ➤ Name ➤ Capacity
<ul style="list-style-type: none"> • Shipment Entity <ul style="list-style-type: none"> ➤ Shipment_ID (Primary Key) ➤ Sender_ID (Foreign Key) ➤ Package_ID (Foreign Key) ➤ Tracking_ID (Foreign Key) ➤ Recipient_ID (Foreign Key) ➤ Vehicle_ID (Foreign Key) ➤ Total_weight ➤ Dimensions ➤ total_price_of_items ➤ Shipping cost 	<ul style="list-style-type: none"> • Recipient Entity <ul style="list-style-type: none"> ➤ Recipient_ID (Primary Key) ➤ Request_ID (Foreign Key) ➤ Name ➤ Address ➤ Phone_number ➤ Email_ID
<ul style="list-style-type: none"> • Customer Entity <ul style="list-style-type: none"> ➤ Cust_ID 	<ul style="list-style-type: none"> • Company Entity <ul style="list-style-type: none"> ➤ Company_ID ➤ Company_Name
<ul style="list-style-type: none"> • Sender Entity <ul style="list-style-type: none"> ➤ Sender_ID (Primary Key) ➤ Shipment_ID (Foreign Key) 	<ul style="list-style-type: none"> • Customer Service Entity <ul style="list-style-type: none"> ➤ cs_ID (Primary key) ➤ Request_ID (Foreign Key)

<ul style="list-style-type: none"> ➤ Recipient_ID (Foreign Key) ➤ Name ➤ Address ➤ Phone_number ➤ Email_ID 	<ul style="list-style-type: none"> ➤ name ➤ email_ID ➤ phone_number
<ul style="list-style-type: none"> • Employee Entity <ul style="list-style-type: none"> ➤ Emp_ID (Primary Key) ➤ WarehouseID (Foreign Key) ➤ Name ➤ Job_title ➤ Address ➤ Email_ID ➤ Phone_number 	<ul style="list-style-type: none"> • Vehicle Entity <ul style="list-style-type: none"> ➤ Vehical_ID (Primary Key) ➤ Employee_ID (Foreign Key) ➤ Type ➤ model
<ul style="list-style-type: none"> • Driver Entity <ul style="list-style-type: none"> ➤ driver_id (PRIMARY KEY) ➤ driver_name ➤ phone_number 	<ul style="list-style-type: none"> • Tracking Entity <ul style="list-style-type: none"> ➤ Tracking_ID (Primary Key) ➤ Recipient_ID (Foreign Key) ➤ Status ➤ Tracking_Time ➤ Tracking_Date
<ul style="list-style-type: none"> • Transaction Entity <ul style="list-style-type: none"> ➤ Transaction_ID (Primary Key) ➤ Recipient_ID (Foreign Key) ➤ Mode_of_transaction 	<ul style="list-style-type: none"> • Request Entity <ul style="list-style-type: none"> ➤ request_id (Primary Key) ➤ recipient_id (Foreign Key) ➤ cs_id (Foreign Key)

This is our fully developed logical model. As you can see, though, when there are an excessive number of entities and links, EER diagrams become challenging to understand. We chose to organize the entities and relationships into entity clusters in order to correct this issue. An entity cluster is a collection of one or more entity types clustered together under a single abstract entity type along with related relationships. We ultimately divided the fifteen entities into groupings.

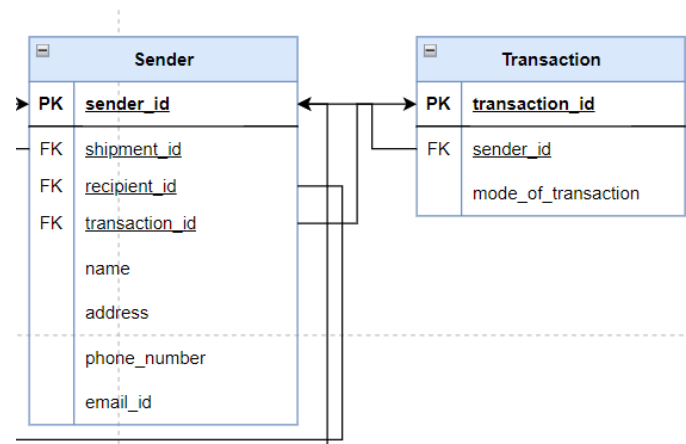


In the logical model, we've delineated the primary keys and foreign keys, elements absent in the conceptual model. Additionally, we've incorporated attributes not originally outlined in the conceptual data model. The **Sender** Entity, includes information about the customer requesting the order, and the **Shipment** entity includes information about the details fulfilling the order. The **Package** entity includes information about the specific order, such as the package details, item details. The **Transaction** entity includes information about the payment method used for the order, status. The **Driver** entity includes information related to the delivery process, such as the ID of the driver assigned, name and contact details of the driver.



Shipment Entity Clustering provides comprehensive information regarding shipment details such as parcel dimensions, weight, total price of items inside the parcel, and shipping costs, which are ultimately used to calculate the sender's transaction amount. This entity is further

divided into two sub-entities: **Package** and **Item**. These entities delve into more specific details about the items within the parcel, including weight, price, name, and quantity of items. The **Tracking** entity is associated with the shipment to provide status updates, tracking dates, and times, enhancing the delivery experience by allowing users to monitor the current status of their package.



The **Sender** Entity Clustering will cover the area of **transaction**. The sender will create the account to enter the credit/debit card information. This will determine how the customer will pay when he or she makes the order. For record purposes, store transactions essential information when they ordered to send some packages.

4. DATA DICTIONARY

Entity Name: Packages					
Column	Datatype	Description	Range	Required	PK/FK
Package_ID	INT	Unique identifier for each package	00001-99999	Y	PK
Shipment_ID	INT	Identifier for the shipment associated with the package	00001-99999	Y	FK
Sender_ID	INT	Identifier for the sender associated with the package	00001-99999	Y	FK
Item_ID	INT	Identifier for the item associated with the package	00001-99999	Y	FK

Entity Name: Warehouse					
Column	Datatype	Description	Range	Required	PK/FK
Warehouse_ID	INT	Unique identifier for each warehouse	00001-99999	Y	PK
Emp_ID	INT	Identifier for the employee of the warehouse	00001-99999	Y	FK
Name	VARCHAR	Name of the warehouse employees		Y	
Capacity	INT	Maximum capacity of the warehouse	00001-99999	Y	

Entity Name: Shipment					
Column	Datatype	Description	Range	Required	PK/FK
Shipment_ID	INT	Unique identifier for each shipment	00001-99999	Y	PK
Sender_ID	INT	Identifier for the sender of the shipment	00001-99999	Y	FK
Package_ID	INT	Identifier for the package associated with the shipment			FK
Tracking_ID	INT	Identifier for tracking the shipment	00001-99999	Y	FK
Recipient_ID	INT	Identifier for the recipient of the shipment	00001-99999	Y	FK
Vehicle_ID	INT	Identifier for the vehicle used for the shipment	00001-99999	Y	FK
Item_names	VARCHAR	Names of the items included in the shipment			
Total_weight	INT	Total weight of the shipment	00001-99999	Y	
Dimensions	INT	Dimensions of the shipment	00001-99999	Y	
total_price_of_items	INT	Declared value or price of the items	00001-99999		
Shipping_cost	INT	Cost of shipping the shipment	00001-99999	Y	

Entity Name: Recipient					
Column	Datatype	Description	Range	Required	PK/FK
Recipient_ID	INT	Unique identifier for each recipient	00001-99999	Y	PK
Address	VARCHAR	address of the recipient		Y	VARCHAR

Request_ID	INT	Identifier for the request associated with the recipient	00001-99999	Y	FK
Name	VARCHAR	Name of the recipient		Y	
Phone_number	INT	Phone number of the recipient	00001-99999	Y	
Email_ID	VARCHAR	Email address of the recipient		Y	

Entity Name: Customer					
Column	Datatype	Description	Range	Required	PK/FK
cust_ID	INT	Unique identifier for each customer	00001-99999	Y	PK

Entity Name: Company					
Column	Datatype	Description	Range	Required	PK/FK
Company_ID	INT	Unique identifier for each company	00001-99999	Y	PK
Company_name	VARCHAR	Name of the company		Y	FK

Entity Name: Sender					
Column	Datatype	Description	Range	Required	PK/FK
Sender_ID	INT	Unique identifier for each sender	00001-99999	Y	PK
Shipment_ID	INT	Identifier for the shipment associated with the sender	00001-99999	Y	FK
Name	VARCHAR	Name of the sender		Y	
Transaction_ID	INT	Identifier for transaction associated with the sender	00001-99999	Y	FK
Address	VARCHAR	Address of the sender		Y	
Phone_number	INT	Phone number of the sender	00001-99999	Y	
Email_ID	VARCHAR	Email address of the sender		Y	

Entity Name: customer Service					
Column	Datatype	Description	Range	Required	PK/FK
CS_ID	INT	Unique identifier for each customer service request	00001-99999	Y	PK
Recipient_ID	INT	Identifier for the recipient associated with the request	00001-99999	Y	FK

name	VARCHAR	Name of the customer service representative		Y	
Email_id	VARCHAR	Email address of the customer service representative		Y	
phone_number	INT	Identifier for contact details	00001-99999	Y	

Entity Name: Request					
Column	Datatype	Description	Range	Required	PK/FK
request_id	INT	Unique identifier for each customer service request	00001-99999	Y	PK
Recipient_ID	INT	Identifier for the recipient associated with the request	00001-99999	Y	FK
cs_id	INT	Identifier for the customer representative associated with the request	00001-99999	Y	FK

Entity Name: Employee					
Column	Datatype	Description	Range	Required	PK/FK
Emp_ID	INT	Unique identifier for each employee, serving as the primary key (PK).	00001-99999	Y	PK
WarehouseID	INT	Identifier for the warehouse to which the employee is assigned, acting as a foreign key (FK).	00001-99999	Y	FK
Name	VARCHAR	Name of the employee		Y	
Job_title	VARCHAR	Job title or role of the employee.		Y	
Address	VARCHAR	Address of the employee.		Y	
Email_ID	VARCHAR	Email address of the employee.		Y	
Phone_no	INT	Phone number of the employee	00001-99999		

Entity Name: Vehicle					
Column	Datatype	Description	Range	Required	PK/FK
Vehical_ID	INT	Unique identifier for each vehicle, serving as the primary key (PK).	00001-99999	Y	PK
Employee_ID	INT	Identifier for the employee associated with the vehicle, acting as a foreign key (FK).	00001-99999	Y	FK
Type	VARCHAR	Description of the vehicle type		Y	
Model	INT	Description of model type of vehicle.	00001-99999	Y	

Entity Name: Driver					
Column	Datatype	Description	Range	Required	PK/FK
Driver_ID	INT	Unique identifier for each driver	00001-99999	Y	PK
Phone_number	INT	Identifier for each driver's contact	00001-99999	Y	
driver_name	VARCHAR	Identifier for each driver's name		Y	

Entity Name: Tracking					
Column	Datatype	Description	Range	Required	PK/FK
Tracking_ID	INT	Unique identifier for tracking each shipment.	00001-99999	Y	PK
Recipient_ID	INT	Identifier for the recipient	00001-99999	Y	FK

		associated with the tracking.			
Status	VARCHAR	Current status of the shipment.		Y	
Tracking_Time	DateTime	Date and time of the tracking event.		Y	
Tracking_Date	Date	Date of the tracking event.		Y	

Entity Name: Transaction					
Column	Datatype	Description	Range	Required	PK/FK
Transaction_ID	INT	Unique identifier for each transaction	00001-99999	Y	PK
Sender_ID	INT	Identifier for the sender associated with the transaction.	00001-99999	Y	FK
mode_of_transaction	INT	Mode of the transaction (e.g., online, in-person).	00001-99999	Y	

Entity Name: Item					
Column	Datatype	Description	Range	Required	PK/FK
item_id	INT	Unique identifier for each item	00001-99999	Y	PK
weight	INT	Identifier for the weight associated with the item.	00001-99999	Y	FK
price	INT	Identifier for the price associated with the item.	00001-99999	Y	
item_names	VARCHAR	Describes name of items to be packed.		Y	

The purpose of the database:

As mentioned in the introduction, the escalating volume of packages and the expanding scope of UPS's delivery operations present a substantial challenge. With an increasing number of customers relying on UPS for their shipping needs, manual handling of packages and logistics becomes increasingly cumbersome and inefficient. To address this challenge, UPS needs to develop a robust database solution tailored to meet the evolving demands of modern logistics.

The primary objective of creating this database solution is to streamline UPS's package handling and delivery logistics by automating and expediting numerous day-to-day tasks. By leveraging efficient data storage and retrieval mechanisms, the database will empower UPS to manage the growing influx of packages with greater precision and efficiency.

Furthermore, ensuring the accuracy and efficiency of data storage and retrieval processes is paramount to the success of the database solution. It is essential to design the database meticulously to meet UPS's unique business requirements and operational workflows effectively. This will enable UPS to navigate the complexities of modern logistics with agility and precision, enhancing the level of service provided to customers and stakeholders alike.

5. PHYSICAL DESIGN - Partition and De-Normalization

No tables have been partitioned or denormalized. Partitioning might not be extremely helpful if the queries do not frequently use the partitioning key to filter data. When queries frequently concentrate on a particular range of data that matches the partition boundaries, partitioning works best. Partitioning also increases the intricacy of management tasks and database schema. It might be more effective to maintain a simple and manageable database structure if the needs of your application do not warrant the additional complexity. Partitioning might not be required if the data is naturally well-distributed among the tables and does not call for it for better distribution or scalability.

The introduction of redundancy through denormalization of the database raises the possibility of inconsistent data and update anomalies. Update anomalies, in which changing data in one location necessitates updating multiple copies of the same data, can arise from denormalization and raise the possibility of inconsistent results. Normalized databases store data in a structured, normalized format, reducing these risks. Since normalized databases do not store redundant data, they usually make better use of available storage space. There is a chance that de-normalization the database will require more storage.

We are designing a database for courier services, so we will need few entities to have access to few functions like read, update, sort and delete entries from the data table. We need to make sure that the database is always has up to date information.

For example, we can sort the price in the item table in a certain order for understanding data by using the following query,

Sort item table by price in descending order

Select * from item order by price;

We can also perform function such as filtering the address of the customer. Example,

Select customerID that are having address in California, Ohio, New York.

Select cust_id from Customer where address in ['California', 'Ohio', 'New York'];

It is necessary to consider the expected values, the type of data being stored, and the operations that will be carried out on the data to determine which data types are suitable for each attribute in a database table. Our case requires that we use 4 different data types. Since attributes are composed of numbers, we use the INT data type for attribute (such as Shipment_ID, Sender_ID, Recipient_ID and Phone_number) that requires a number. To preserve the organization and readability of our data, we utilized the DATE and DATETIME data types for attributes pertaining to date and time.

We used the VARCHAR data type for any attributes used storing text data such as Address, Email_ID and name.

We have chosen to go with the Indexed File method for our database. An indexed file is a file in which each record includes a primary key. To distinguish one record from another, the value of the primary key must be unique for each record. Records can then be accessed randomly by specifying the value of the record's primary key. Indexed file records can also be accessed sequentially. Indexes improve query performance for point queries and selective retrieval by offering quick access to records based on key values.

6. DATABASE RECOVERY POLICY

A database recovery policy is a set of rules that an organization adheres to to guarantee that a process or mechanism is in place for recovering lost databases or fixing database damage. Having backup facilities is one way of doing it.

It includes components like,

Storage Infrastructure: This refers to systems or storage devices that are used to hold backup data. It could include cloud-based storage options as well as on-site storage servers, tape libraries, and disk arrays.

Backup Software: Backup software controls data transfer, schedules backups, oversees the backup procedure, and aids in recovery efforts. Oracle RMAN (Recovery Manager) is one example.

7. DATABASE SECURITY POLICY

Data Encryption and Security: Encryption techniques can be used by backup facilities to safeguard backup data while it is being transmitted and stored, shielding it from breaches or unwanted access.

Disaster Recovery Planning: A comprehensive disaster recovery plan, which includes backup facilities, usually includes procedures for restoring data and systems in the event of a significant outage or disaster.

8. SQL QUERIES

Having finalized the design of our database, the next crucial step is to implement it into SQL. For this application, we have opted for MySQL as our SQL Flavors of choice. The initial phase involves creating tables in MySQL based on the entities delineated in our design, ensuring that each column is assigned the appropriate data type.

Our database design encompasses 15 entities or classes, necessitating the creation of 15 tables in MySQL, with each column populated with the correct data types. During the table creation process, it's imperative to establish relationships between entities using primary and foreign keys.

Subsequently, we proceed to insert dummy data into the database. This data, although devoid of real information, serves the purpose of testing the functionality of our MySQL database for this project. While accuracy and timeliness aren't paramount for this test data, they would be critical considerations for a real-world application.

With the dummy data inserted, we can now formulate a diverse range of queries to retrieve specific information and execute common tasks that users might perform. These queries form the backbone of our database's functionality, addressing various user requirements.

A couple of these queries are the following:

- 1) CRUD represents the four fundamental database operations: Create, Read, Update, Delete (These operations respectively involve adding new data, retrieving existing data, modifying data, and removing data from a database)

CREATE table Vehicle :

```

1 CREATE TABLE Vehicle (
2     vehicle_id INT AUTO_INCREMENT PRIMARY KEY,
3     driver_id INT,
4     type VARCHAR(100) NOT NULL,
5     model VARCHAR(100) NOT NULL,
6     FOREIGN KEY (driver_id) REFERENCES Driver(driver_id)
7 );

```

Output

#	Time	Action
1	20:16:46	create database dbproj
2	20:17:47	use dbproj
3	20:22:12	CREATE TABLE Driver (driver_id INT AUTO_INCREMENT PRIMARY KEY, driver_name VARCHAR(2...
4	20:22:24	CREATE TABLE Vehicle (vehicle_id INT AUTO_INCREMENT PRIMARY KEY, driver_id INT, type V...

Delete : Delete from employee table where employee id is 1

```

1 use proj;
2 DELETE FROM employee WHERE employee_id=1;

```

Select : We used "Select*" query for reading/retrieving all the data present in the employee table

Query 1 SQL File 1

```
1 Select * from employees;
```

Result Grid

Employee_ID	WarehouseID	name	job_title	address	email_id	pl
1	1	Alexander Smith	Manager	New York	alexander.smith@example.com	22
2	2	Emma Johnson	Worker	California	emma.johnson@example.com	35
3	3	Ethan Brown	Worker	Illinois	ethan.brown@example.com	95
4	4	Sophia Williams	Worker	Texas	sophia.williams@example.com	53
5	5	Benjamin Jones	Worker	Arizona	benjamin.jones@example.com	58
6	6	Mia Garcia	Worker	Pennsylvania	mia.garcia@example.com	34
7	7	Samuel Martinez	Worker	Texas	samuel.martinez@example.com	70

Update : Update" query is used for modifying the existing table data example. Updating name to "Hello" where WarehouseID = 1

Query 1 SQL File 1

```

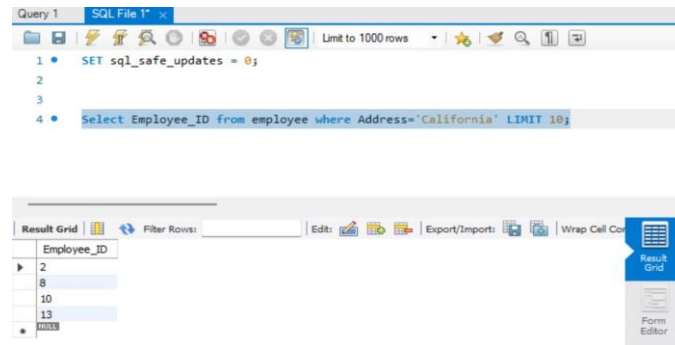
1 SET sql_safe_updates = 0;
2
3
4 Update employee SET name= "Hello" where WarehouseID=1;
5 Select * from employee

```

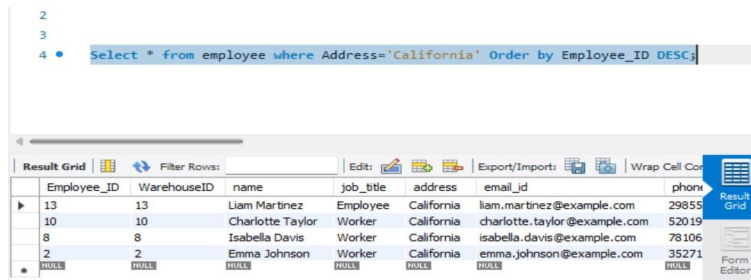
Result Grid

Employee_ID	WarehouseID	name	job_title	address	email_id	pl
1	1	Hello	Manager	New York	alexander.smith@example.com	22
2	2	Emma Johnson	Worker	California	emma.johnson@example.com	35
3	3	Ethan Brown	Worker	Illinois	ethan.brown@example.com	95
4	4	Sophia Williams	Worker	Texas	sophia.williams@example.com	53
5	5	Benjamin Jones	Worker	Arizona	benjamin.jones@example.com	58
6	6	Mia Garcia	Worker	Pennsylvania	mia.garcia@example.com	34
7	7	Samuel Martinez	Worker	Texas	samuel.martinez@example.com	70
8	8	Isabella Davis	Worker	California	isabella.davis@example.com	78
9	9	James Anderson	Worker	Texas	james.anderson@example.com	90
10	10	Charlotte Taylor	Worker	California	charlotte.taylor@example.com	52
11	11	William Rodriguez	Employee	Texas	william.rodriguez@example.com	72
12	12	Amelia Hernandez	Employee	Florida	amelia.hernandez@example.com	53
13	13	Liam Martinez	Employee	California	liam.martinez@example.com	29
14	14	Harper Wilson	Employee	Indiana	harper.wilson@example.com	26
15	15	Oliver Thompson	Employee	Ohio	oliver.thompson@example.com	99

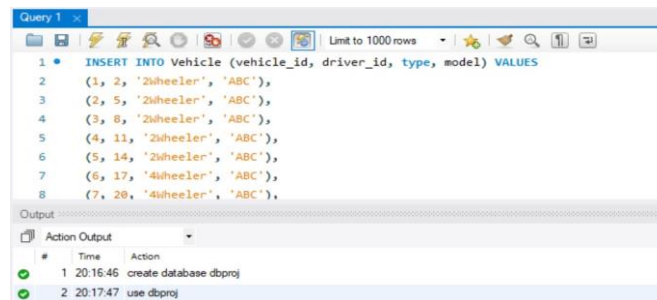
Limit : Limiting the number of record from the table employee and LIMIT to 10



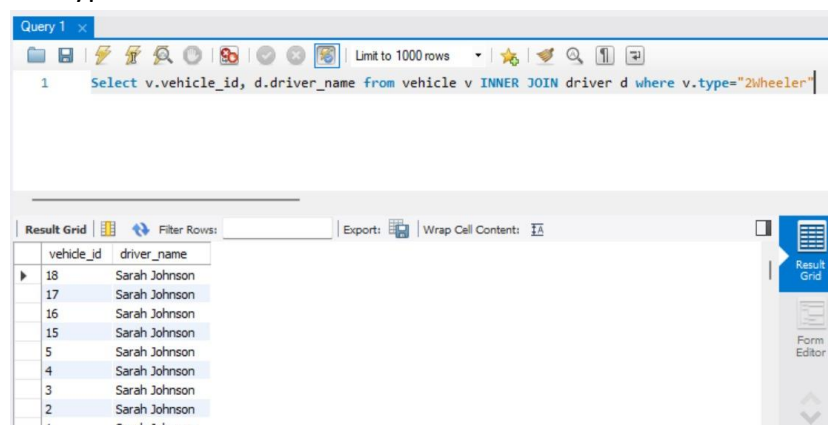
Order : Ordering the table employee in a descending manner where Address = "California"



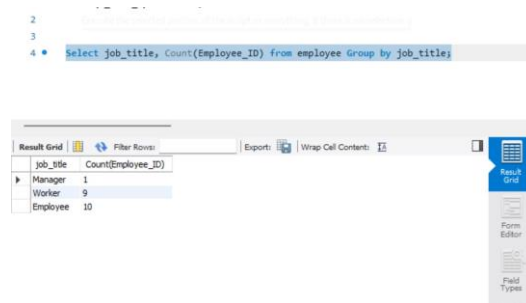
Insert : Insert data into the table using INSERT command adding vehicle_id, driver_id, model details.



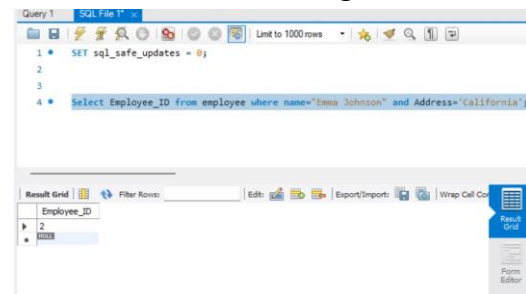
JOIN : Inner Join Command used to display vehicle ID and driver name from vehicle table where the vehicle type is 2wheeler.



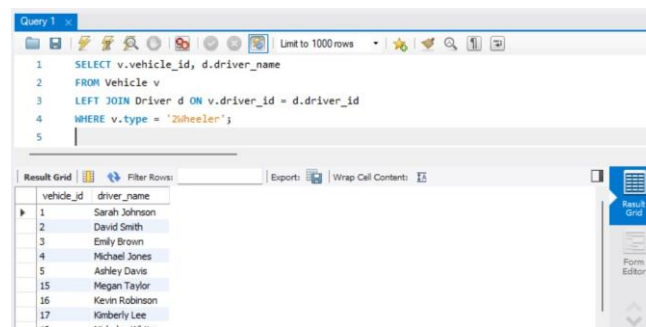
Group : Grouping the table and providing the count in each grp



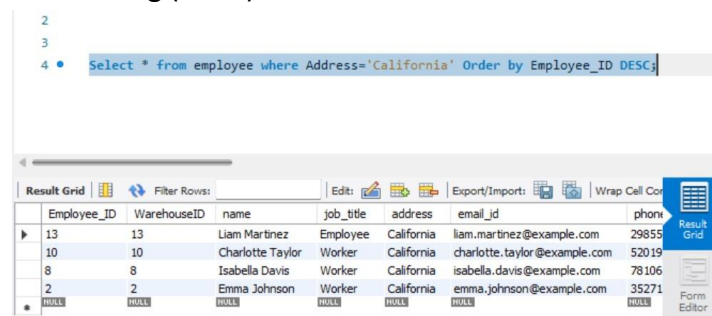
Filter : Filtering the column based on the condition given in Name and Address



Left : The LEFT JOIN command is used in SQL to retrieve data from two or more database tables based on a related column between them

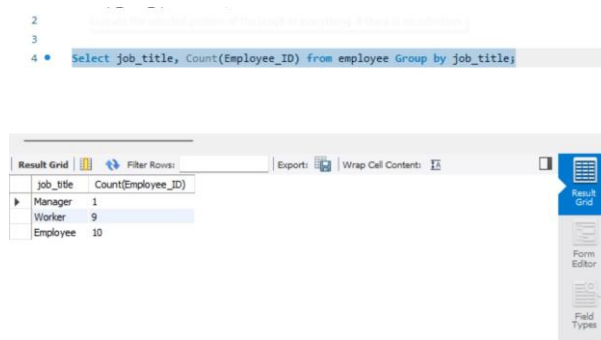


ORDER : The ORDER BY command in SQL is used to sort the result set of a query in either ascending (ASC) or descending (DESC) order based on one or more columns.



Group By :The GROUP BY command in SQL is used to group rows that have the same values into summary rows.

```
1  -- Create the employee table
2  -- Create the employee table
3
4  * Select job_title, Count(Employee_ID) from employee Group by job_title;
```

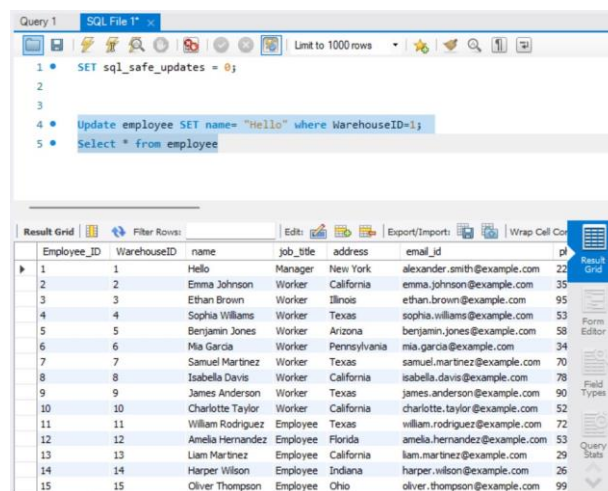


The screenshot shows a SQL query editor with a query that groups employees by job title and counts the number of employees for each job title. The result grid shows the following data:

job_title	Count(Employee_ID)
Manager	1
Worker	9
Employee	10

Update : The UPDATE command in SQL is used to modify existing records in a table.

```
1  SET sql_safe_updates = 0;
2
3
4  Update employee SET name= "Hello" where WarehouseID=1;
5  Select * from employee
```



The screenshot shows a SQL query editor with a query that updates the name of the employee with WarehouseID=1 to 'Hello'. The result grid shows the following data:

Employee_ID	WarehouseID	name	job_title	address	email_id	pt
1	1	Hello	Manager	New York	alexander.smith@example.com	22
2	2	Emma Johnson	Worker	California	emma.johnson@example.com	35
3	3	Ethan Brown	Worker	Illinois	ethan.brown@example.com	95
4	4	Sophia Williams	Worker	Texas	sophia.williams@example.com	53
5	5	Benjamin Jones	Worker	Arizona	benjamin.jones@example.com	58
6	6	Mia Garcia	Worker	Pennsylvania	mia.garcia@example.com	34
7	7	Samuel Martinez	Worker	Texas	samuel.martinez@example.com	70
8	8	Isabella Davis	Worker	California	isabella.davis@example.com	78
9	9	James Anderson	Worker	Texas	james.anderson@example.com	90
10	10	Charlotte Taylor	Worker	California	charlotte.taylor@example.com	52
11	11	William Rodriguez	Employee	Texas	william.rodriguez@example.com	72
12	12	Amelia Hernandez	Employee	Florida	amelia.hernandez@example.com	53
13	13	Liam Martinez	Employee	California	liam.martinez@example.com	29
14	14	Harper Wilson	Employee	Indiana	harper.wilson@example.com	26
15	15	Oliver Thompson	Employee	Ohio	oliver.thompson@example.com	99