Fleet Management System

Abhishek Jadhav
UBID: ajadhav8

Aishwarya Salvi
UBID: aishvary

MILESTONE I

I. INTRODUCTION:

Fleet management systems are crucial for maximizing vehicle use, maintenance, and operation, particularly when considering contemporary mobility options like autonomous vehicles. The main goal of this project is to create a scalable and reliable database system specifically designed for overseeing a fleet of autonomous vehicles. The system is made to efficiently plan maintenance, track vehicle data, expedite customer ride bookings, and handle payments. The solution guarantees real-time data processing, reduces operational delays, and improves fleet management's overall effectiveness by utilizing a well-structured database.

II. PROBLEM STATEMENT:

The quick integration of driverless cars into ride-hailing services has made fleet management much more difficult. The dynamic and complicated nature of the data produced by autonomous ride-booking services, such as real-time ride requests, vehicle telemetry, maintenance schedules, and payment processing, is frequently too much for current systems to manage. Delays in decision-making, fragmented data management, and inefficient fleet operations are caused by the absence of a centralized and scalable database system. The goal is to build a scalable and organized database that can manage vehicle operations, process payments, handle user ride requests, and keep track of previous trip information. This system seeks to offer a dependable and effective solution for modern transportation needs by tackling the intricacies of fleet management through an efficient data model.

The system will:

- **Centralize Data Storage**: Organize data related to customers, vehicles, trips, payments, and maintenance in a well- structured relational database to ensure data integrity and accessibility.
- **Enable Real-Time Updates**: Support dynamic updates for ride requests, vehicle statuses, payment processing, and maintenance alerts to ensure seamless operations.
- **Facilitate Querying**: Allow stakeholders to retrieve critical information, such as a user's ride history, available vehicles near a location, total revenue from completed trips, and vehicles requiring maintenance.
- **Provide Analytical Insights**: Enable trend analysis, such as historical ride patterns, fare calculations, and vehicle utilization rates, to support data-driven decision-making.

III. REASONS FOR CHOOSING DATABASE OVER EXCEL

Large amounts of structured and relational data, including ride reservations, vehicle telemetry, maintenance logs, and payment information, must be handled while managing an autonomous car fleet. Excel and similar programs are good for managing data on a local scale, but they can't handle the complexity and scalability needs of a fleet management system. However, a database system has several benefits that make it the ideal option for this project:

A. **Scalability**: Handles large volumes of structured data efficiently, unlike Excel, which struggles with large datasets.
B. **Data Integrity**: Ensures accuracy through constraints, relationships, and validation rules, minimizing errors.
C. **Concurrent Access**: Supports multiple users accessing and modifying data simultaneously without conflicts.
D. **Security**: Provides role-based access control, ensuring only authorized personnel can access or modify data.
E. **Advanced Querying**: Enables complex data retrieval and analysis using SQL, which is cumbersome in Excel.
F. **Automated Backups**: Ensures data recovery in case of failures, preventing data loss and enhancing reliability.

IV. INTENDED USERS OF THE DATABASE

The fleet management database system will serve the following user.

A. **Fleet Administrators:** Fleet administrators will use the system to monitor vehicle operations, schedule maintenance, and manage ride assignments. They rely on the database to ensure efficient fleet utilization and timely decision-making.
B. Customers: Customers will interact with the system to book rides, track trip statuses, and make payments. The database ensures a seamless and reliable experience for users by providing real-time updates and accurate trip information.
C. **Maintenance Teams**: Maintenance teams will use the system to identify vehicles requiring servicing, track maintenance schedules, and log repair activities. The database helps ensure vehicle safety and minimizes downtime.
D. **Data Analysts**: Data analysts will leverage the database to analyze historical ride trends, calculate revenue, and optimize fleet operations. The system

provides valuable insights for improving efficiency and profitability.

E. **Payment Processor:** Payment processors will use the database to handle transaction records, verify payments, and generate financial reports. The system ensures secure and accurate payment processing.

V. WHO WILL ADMINISTER THE DATABASE?

Data engineers and database administrators (DBAs) will work together to maintain the fleet management database's performance, security, and integrity. They will be in charge of crucial duties like data entry, schema design, optimization, backups, and security to guarantee the system's scalability and stability. According to their needs, data analysts will also communicate with the database to produce reports and insights for different stakeholders.

Based on the analysis, the company implemented the following measures:

- **Predictive Maintenance**: Scheduled maintenance based on vehicle telemetry data to prevent unexpected breakdowns.
- **Dynamic Ride Allocation**: Optimized ride assignments to reduce idle time and improve fleet utilization.
- **Customer Feedback Integration**: Used trip data to improve service quality and customer satisfaction.

VI. RELATIONS AND ATTRIBUTES

To gain a deeper understanding of the dataset and develop an optimized database for fleet management, we examined the data to identify essential entities, attributes, and relationships. The system was then modeled using the following tables:

Table 1: Vehicles Table

Attributes: vehicle_id, model, status, latitude, longitude, battery_id,last_updated
Relations: The Vehicles table is part of the Fleet schema and represents individual vehicles. Each vehicle is uniquely identified by the vehicle_id primary key. It has a one-to-many relationship with the Batteries table via battery_id, which serves as a foreign key. It also connects to other tables such as Telemetry, Alerts, Maintenance, and Trips through vehicle_id.

Table 2: Batteries Table:

Attributes: battery_id, capacity_kwh, health_status, charge_level, last_replacement_date, last_updated
Relations: The Batteries table stores information about vehicle batteries. Each battery is uniquely identified by battery_id (primary key). It has a one-to-many relationship with the BatteryUpdates table through battery_id. Additionally, it connects to the Vehicles table via a foreign key relationship.

Table 3: BatteryUpdates Table

Attributes: update_id, battery_id, vehicle_id, charge_level_before, charge_level_after, timestamp
Relations: This table logs updates to battery charge levels. Each record is uniquely identified by update_id. It has a many-to-one relationship with the Batteries table via battery_id and with the Vehicles table through the vehicle_id column.

Table 4: Telemetry Table

Attributes: telemetry_id, vehicle_id, timestamp, speed_kmh, charge_level, battery_temp, latitude, longitude, errors
Relations: The Telemetry table records real-time data for vehicles. Each record is uniquely identified by telemetry_id. It has a many-to-one relationship with the Vehicles table via vehicle_id.

Table 5: Alerts Table

Attributes: alert_id, vehicle_id, issue, severity, timestamp
Relations: The Alerts table logs issues or warnings related to vehicles. Each alert is uniquely identified by alert_id. It has a many-to-one relationship with the Vehicles table via vehicle_id.

Table 6: Maintenance Table

Attributes: maintenance_id, vehicle_id, maintenance_type, scheduled_date, cost
Relations: The Maintenance table tracks scheduled or completed maintenance for vehicles. Each record is uniquely identified by maintenance_id. It has a many-to-one relationship with the Vehicles table via vehicle_id.

Table 7: Trips Table

Attributes: trip_id, vehicle_id, start_time, end_time, start_latitude,start_longitude,end_latitude, end_longitude, distance_km
Relations: The Trips table logs details of trips taken by vehicles. Each trip is uniquely identified by trip_id. It has a many-to-one relationship with the Vehicles table via vehicle_id.

Table 8: Customers Table

Attributes: customer_id, name, phone_number, email, default_payment_method
Relations: The Customers table stores customer information. Each customer is uniquely identified by customer_id. It connects to the RideRequests and Payments tables via foreign keys.

Table 9: RideRequests Table

Attributes: ride_id, customer_id, vehicle_id (optional), request_time, start_latitude, start_longitude, end_latitude,

end_longitude,estimated_fare,status.
Relations: This table logs ride requests made by customers. Each request is uniquely identified by ride_id. It has foreign key relationships with the Customers and Vehicles tables.

Table 10: Payments Table

Attributes: payment_id, ride_id (optional), customer_id (optional), amount, payment_method, payment_status
Relations: The Payments table tracks payments made for rides. Each payment is uniquely identified by payment_id. It connects to RideRequests and Customers tables via foreign keys.

This ER diagram represents a system managing an electric vehicle fleet and its operations (e.g., telemetry data collection), customer ride requests/payments integration with fleet management functionalities like alerts and maintenance tracking.

VII. ER Diagram

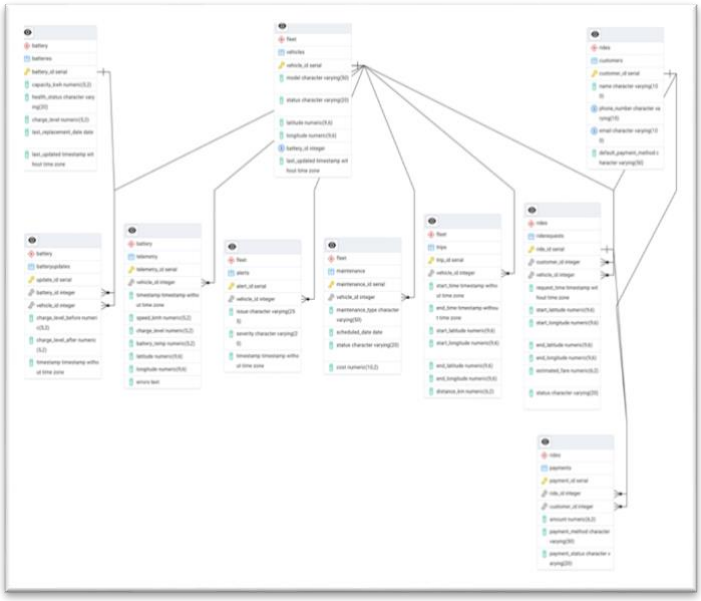


Fig 1: ER Diagram

VIII. DESCRIPTION OF THE ATTRIBUTES

The following table represents the attributes of our Dataset. And its details description and attribute relations for the different tables. We can identify different use cases of keys and datatype in bellow details

Attribute Name	Data Type	Description
vehicle_id	serial	Unique identifier for each vehicle.
model	character varying(50)	Model name or type of the vehicle.
status	character varying(50)	Current operational status of the vehicle (e.g., active, inactive).
latitude	numeric(9, 6)	Current latitude coordinate of the vehicle's location.
longitude	numeric(9, 6)	Current longitude coordinate of the vehicle's location.
battery_id	integer	Foreign key linking to the Batteries table.
last_updated	timestamp without time zone	Timestamp of the last update for this vehicle.
battery_id	serial	Unique identifier for each battery.
capacity_kwh	numeric(5, 2)	Maximum energy capacity of the battery in kilowatt-hours.
health_status	character varying(20)	Health condition or status of the battery (e.g., good, degraded).
charge_level	numeric(5, 2)	Current charge level as a percentage of full capacity.
last_replacement_date	date	Date when the battery was last replaced.
update_id	serial	Unique identifier for

		each battery update record.
battery_id	integer	Foreign key linking to the Batteries table.
vehicle_id	integer	Foreign key linking to the Vehicles table.
charge_level_before	numeric(5, 2)	Battery charge level before the update occurred.
charge_level_after	numeric(5, 2)	Battery charge level after the update occurred.
timestamp	timestamp without time zone	Timestamp when the update occurred.
telemetry_id	serial	Unique identifier for each telemetry record.
vehicle_id	integer	Foreign key linking to the Vehicles table.
timestamp	timestamp without time zone	Timestamp of telemetry data collection.
speed_kmh	numeric(5, 2)	Current speed of the vehicle in kilometers per hour.
charge_level	numeric(5, 2)	Current battery charge level as a percentage of full capacity.
battery_temp	numeric(5, 2)	Current temperature of the battery in degrees Celsius.
latitude	numeric(9, 6)	Latitude coordinate of the vehicle's location during telemetry collection.
longitude	numeric(9, 6)	Longitude coordinate of the vehicle's location during telemetry collection.
errors	text	Description of any errors or issues detected during telemetry data collection.
alert_id	serial	Unique identifier for each alert record.
vehicle_id	integer	Foreign key linking to the Vehicles table associated with this alert.
issue	character varying(255)	Description of the issue causing the alert to be generated.
severity	character varying(20)	Severity level of the alert (e.g., low, medium, high).
timestamp	timestamp without time zone	Timestamp when the alert was generated.
maintenance_id	serial	Unique identifier for each maintenance record.
vehicle_id	integer	Foreign key linking to the Vehicles table associated with this maintenance task.
maintenance_type	character varying(20)	Type or category of maintenance performed or scheduled.
scheduled_date	date	Date when maintenance is scheduled to occur.
cost	numeric(10,2)	Cost associated with this

		maintenance task.
trip_id	serial	Unique identifier for each trip record.
vehicle_id	integer	Foreign key linking to the Vehicles table associated with this trip.
start_time	timestamp without time zone	Timestamp when the trip started.
end_time	timestamp without time zone	Timestamp when the trip ended.
start_latitude	numeric(9, 6)	Latitude coordinate where the trip started.
start_longitude	numeric(9, 6)	Longitude coordinate where the trip started.
end_latitude	numeric(9, 6)	Latitude coordinate where the trip ended.
end_longitude	numeric(9, 6)	Longitude coordinate where the trip ended.
distance_km	numeric(10 ,2)	Total distance traveled during this trip in kilometers.
customer_id	serial	Unique identifier for each customer record.
name	character varying(100)	Full name of the customer.
phone_number	character varying(15)	Contact phone number for the customer.
email	character varying(100)	Email address of the customer.
default_payment_m ethod	character varying(50)	Default payment method

		selected by the customer.
ride_id	serial	Unique identifier for each ride request record.
customer_id	integer	Foreign key linking to Customers table associated with this ride request.
vehicle_id	integer	Foreign key linking to Vehicles table assigned to fulfill this ride request.
request_time	timestamp without time zone	Timestamp when this ride request was made.
start_latitude	numeric(9, 6)	Latitude coordinate where the ride starts.
start_longitude	numeric(9, 6)	Longitude coordinate where the ride starts.
end_latitude	numeric(9, 6)	Latitude coordinate where the ride ends.
end_longitude	numeric(9, 6)	Longitude coordinate where the ride ends.
estimated_fare	numeric(6, 2)	Estimated fare for fulfilling this ride request in currency units.
status	character varying(20)	Status of this ride request.
payment_id	serial	Unique identifier for each payment record.
ride_id	integer	Foreign key linking to the RideRequests table for the ride associated

		with this payment.
customer_id	integer	Foreign key linking to the Customers table for the customer making the payment.
amount	numeric(6, 2)	Total amount paid for the ride in currency units.
payment_method	character varying(50)	Method used for payment (e.g., credit card, PayPal).
payment_status	character varying(20)	Status of the payment (e.g., completed, pending).

IX. WHAT HAPPENS WHEN THE PRIMARY KEY IS DELETED

With Foreign Key Constraints:

On delete cascade: Related records in child tables are automatically deleted.

On delete set null: Foreign key fields in child tables are set to NULL.

Restrict/no action: Deletion is blocked if dependent records exist.

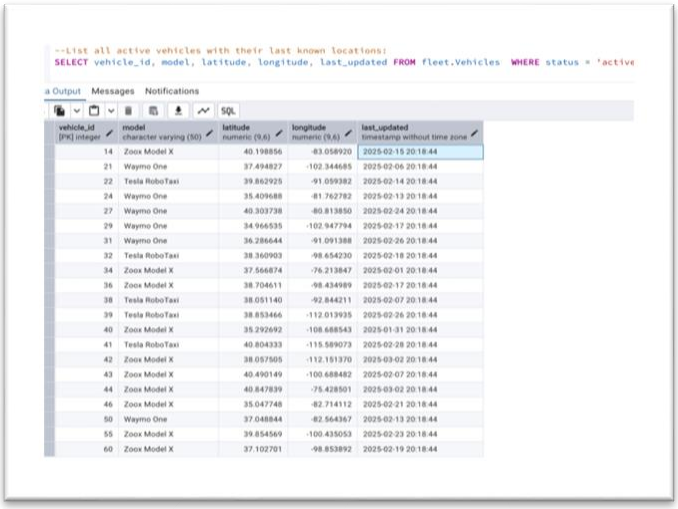
Without Foreign Key Constraints:

The primary key record is deleted, but orphaned records (child records referencing non-existent keys) may remain, causing data integrity issues.

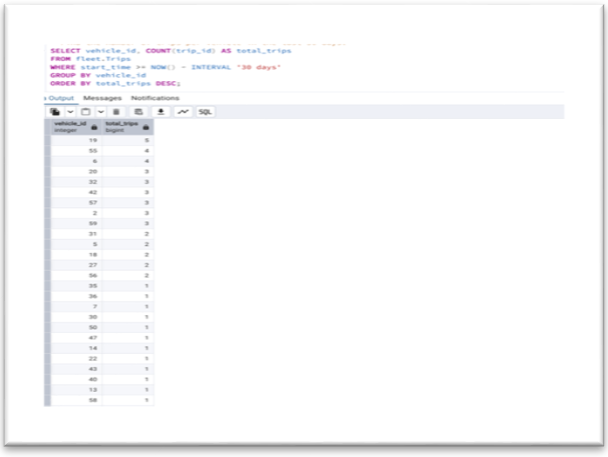
X. QUERY APPLICATION

We tested a few basic queries to ensure our system operates as expected. And we have represented bellow sample queries and its results in cumulative form. We tried to implement complex queries for better understanding of database and its attributes.

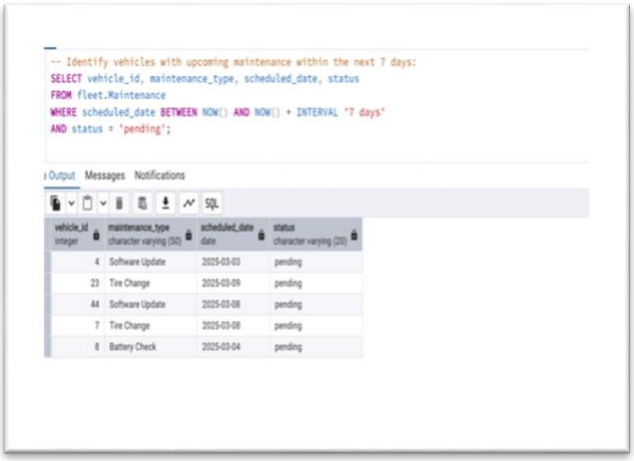
1) All Active Vehicles with their last known locations.



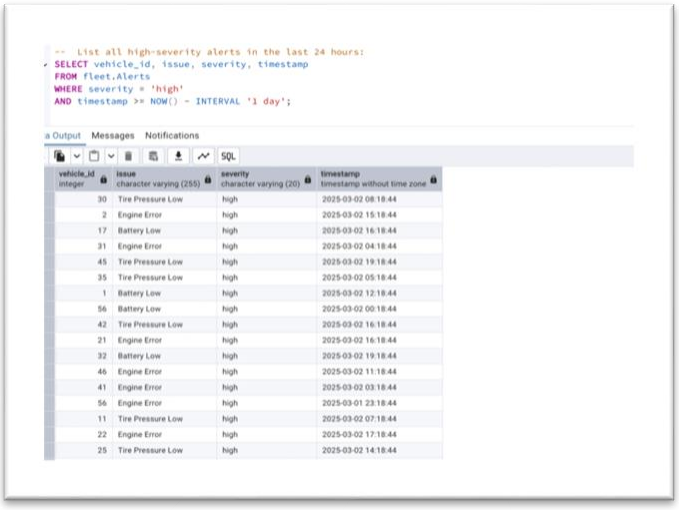
2)No of vehicles usage in last 30 days



3) Identify vehicles with upcoming maintenance within next 7 days.



4) List all High- severity alerts in the last 24 hours.

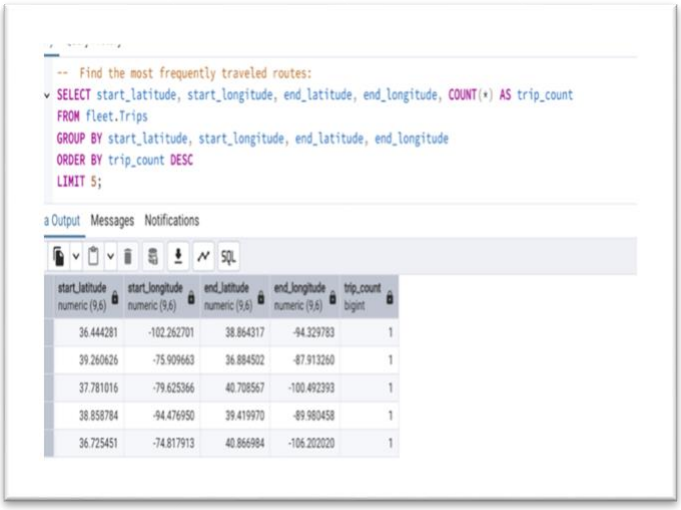


```
-- List all high-severity alerts in the last 24 hours:
SELECT vehicle_id, issue, severity, timestamp
FROM fleet.Alerts
WHERE severity = 'high'
AND timestamp >= NOW() - INTERVAL '1 day';
```

a Output Messages Notifications

vehicle_id integer	issue character varying (255)	severity character varying (20)	timestamp timestamp without time zone
30	Tire Pressure Low	high	2025-03-02 08:18:44
2	Engine Error	high	2025-03-02 15:18:44
17	Battery Low	high	2025-03-02 16:18:44
31	Engine Error	high	2025-03-02 04:18:44
45	Tire Pressure Low	high	2025-03-02 19:18:44
35	Tire Pressure Low	high	2025-03-02 05:18:44
1	Battery Low	high	2025-03-02 12:18:44
56	Battery Low	high	2025-03-02 00:18:44
42	Tire Pressure Low	high	2025-03-02 16:18:44
21	Engine Error	high	2025-03-02 16:18:44
32	Battery Low	high	2025-03-02 19:18:44
46	Engine Error	high	2025-03-02 11:18:44
41	Engine Error	high	2025-03-02 03:18:44
56	Engine Error	high	2025-03-01 23:18:44
11	Tire Pressure Low	high	2025-03-02 07:18:44
22	Engine Error	high	2025-03-02 17:18:44
25	Tire Pressure Low	high	2025-03-02 14:18:44

5)Most Frequent traveled routes

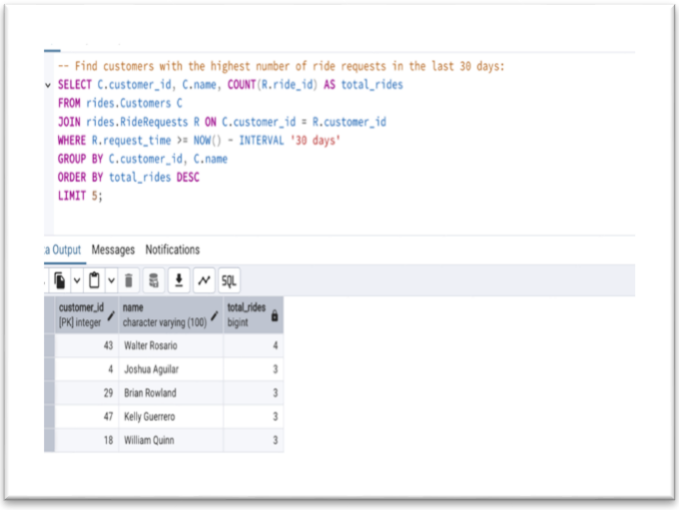


```
-- Find the most frequently traveled routes:
SELECT start_latitude, start_longitude, end_latitude, end_longitude, COUNT(*) AS trip_count
FROM fleet.Trips
GROUP BY start_latitude, start_longitude, end_latitude, end_longitude
ORDER BY trip_count DESC
LIMIT 5;
```

a Output Messages Notifications

start_latitude numeric (9,6)	start_longitude numeric (9,6)	end_latitude numeric (9,6)	end_longitude numeric (9,6)	trip_count bigint
36.444281	-102.262701	38.864317	-94.329783	1
39.260626	-75.909663	36.884502	-87.913260	1
37.781016	-79.625366	40.708567	-100.492393	1
38.858784	-94.476950	39.419970	-89.980458	1
36.725451	-74.817913	40.866984	-106.202020	1

6)Find Customer with highest number of ride requests in the last 30 days.

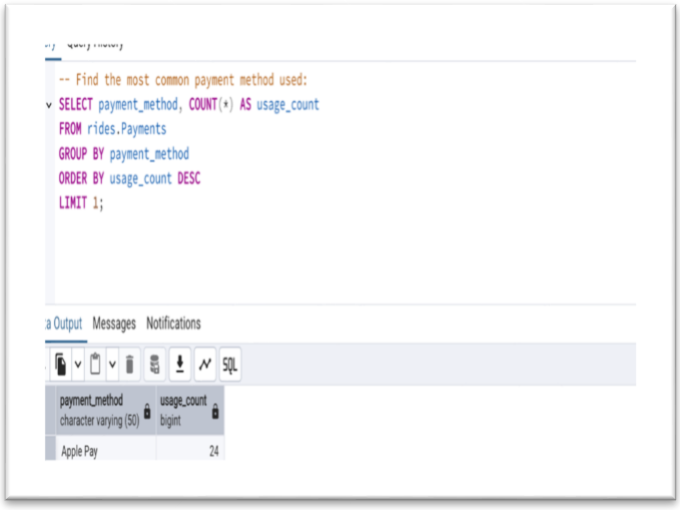


```
-- Find customers with the highest number of ride requests in the last 30 days:
SELECT C.customer_id, C.name, COUNT(R.ride_id) AS total_rides
FROM rides.Customers C
JOIN rides.RideRequests R ON C.customer_id = R.customer_id
WHERE R.request_time >= NOW() - INTERVAL '30 days'
GROUP BY C.customer_id, C.name
ORDER BY total_rides DESC
LIMIT 5;
```

a Output Messages Notifications

customer_id [PK] integer	name character varying (100)	total_rides bigint
43	Walter Rosario	4
4	Joshua Aguilar	3
29	Brian Rowland	3
47	Kelly Guerrero	3
18	William Quinn	3

7) Most Common Payment Method Used-



```
-- Find the most common payment method used:
SELECT payment_method, COUNT(*) AS usage_count
FROM rides.Payments
GROUP BY payment_method
ORDER BY usage_count DESC
LIMIT 1;
```

a Output Messages Notifications

payment_method character varying (50)	usage_count bigint
Apple Pay	24

Conclusion -

This database model is created to quickly access the data and have a whole systems view to understand where the data resides. ER – diagram helps you understand the relationship between tables. Data catalogues help you understand the meaning of it. Having the data model in 3nf along with each component of the system separated into different schemas helps a lot even for providing access related to different teams.