

CIS 552- DATABASE DESIGN

(Fall 2023)

Project Report

Designing Database for Vehicle Parking Management System

Professor:

Name: Yukui Luo, PhD

Office: Dion 302C

Email: yluo2@umassd.edu

Group Members: (Group 27)

S.No	Name	Student ID
65	Naga Venkata Sai Pavan Kumar Papireddy Padala	02079194
66	Sasi kiran Mullapudi	02084115

Introduction

In today's fast-paced urban environments, efficient management of vehicle parking spaces has become increasingly crucial. The objective of our project is to design and implement a sophisticated database system dedicated to managing and monitoring the inflow, outflow, and occupancy of vehicles in parking lots and garages. This initiative aims to address the growing need for streamlined parking solutions in crowded urban areas, commercial complexes, and public spaces.

Our system, while currently focused on the backend database infrastructure, lays the foundation for a comprehensive parking management solution. Without the development of front-end screens at this stage, our primary emphasis is on creating a robust and scalable database system that can accurately track and manage parking space utilization.

The core of our project revolves around a well-structured SQL database that encompasses detailed tables for parking spaces, users, vehicles, and entry/exit logs. This design facilitates the efficient recording and retrieval of crucial information such as parking space availability, vehicle details, user information, and precise timestamps of vehicle movements.

Problem Statement

In urban and suburban areas, managing and monitoring parking spaces efficiently has become a significant challenge due to the increasing number of vehicles and limited parking availability. Traditional parking systems often lead to inefficient space usage, traffic congestion, and difficulty in locating available spots. There is a lack of real-time information on space availability, leading to increased pollution and time wastage. Furthermore, the absence of a systemized way to track vehicle movements and user information raises security concerns and operational inefficiencies.

Our project aims to develop a robust database system for a Vehicle Parking Management System, designed to efficiently manage parking space occupancy, track vehicle inflow and outflow, and maintain comprehensive vehicle and user records. This solution is intended to optimize space utilization, reduce congestion, enhance security, and lay the foundation for future integration with user-facing applications, ultimately transforming parking management in urban settings.

Database Design

i) Schema Description

The database schema for the Vehicle Parking Management System consists of four main tables:

a) Parking Space

SpaceID (INT, Primary Key): Unique identifier for each parking space.

Status (VARCHAR(15)): Indicates whether the space is available or occupied.

SizeType (VARCHAR(50)): Describes the size of the parking space (e.g., Compact, Standard, Large).

b) User

UserID (INT, Primary Key): Unique identifier for each user.

Name (VARCHAR(100)): Name of the user.

ContactInformation (VARCHAR(255)): Contact details of the user.

c) Vehicle

RegistrationNumber (VARCHAR(20), Primary Key): Unique identifier for each vehicle.

VehicleType (VARCHAR(50)): Type of vehicle (e.g., Car, Truck, SUV).

OwnerUserID (INT, Foreign Key): References UserID in the User table.

d) EntryExitLog

LogID (INT, Primary Key): Unique identifier for each entry/exit log.

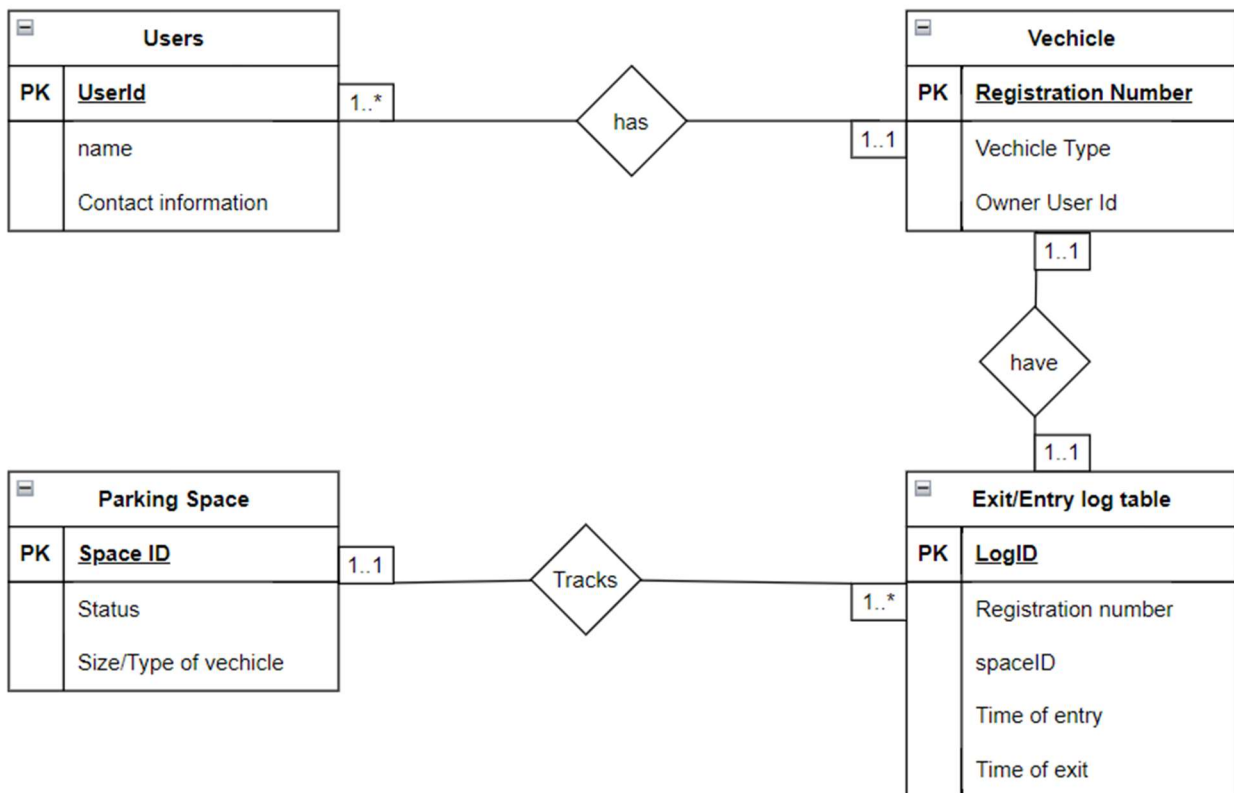
RegistrationNumber (VARCHAR(20), Foreign Key): References RegistrationNumber in the Vehicle table.

SpaceID (INT, Foreign Key): References SpaceID in the ParkingSpace table.

TimeOfEntry (TIMESTAMP): Time when the vehicle entered the parking space.

TimeOfExit (TIMESTAMP): Time when the vehicle left the parking space.

ii) UML Diagram



iii) Normalization

The database design adheres to the principles of normalization:

1st Normal Form (1NF): Each table has a primary key, and all fields contain atomic values, ensuring that there are no repeating groups.

2nd Normal Form (2NF): The design meets all the requirements of 1NF, and all non-key attributes are fully functionally dependent on the primary key. This is evident in tables like Vehicle, where VehicleType is dependent on RegistrationNumber.

3rd Normal Form (3NF): The schema satisfies 2NF, and all the fields in each table are only dependent on the primary key. For instance, OwnerUserID in Vehicle is directly dependent on RegistrationNumber, not on any other field.

This level of normalization ensures data integrity, minimizes redundancy, and enhances data consistency, which is crucial for maintaining accurate and reliable parking management data.

Data Collection

Data collection in the Vehicle Parking Management System is a critical process that involves gathering relevant information required to effectively manage parking spaces. The data is collected and stored in the database schema as outlined previously. Here's an overview of how data is collected for each table:

Usually, the data is to be sourced used as per the below mentioned Process, but here in this Project we have created the raw data randomly as we haven't found any sources for the data on the web

ParkingSpace Table

Source: The data for parking spaces typically comes from the physical layout and capacity assessments of the parking lot or garage.

Method: Parking space details like size and location are recorded during the initial setup of the system. The status (Available/Occupied) is dynamically updated based on real-time parking activity.

User Table

Source: User data is obtained when individuals register for access to the parking facility. This could be through a registration form or an online portal (if integrated in the future).

Method: Collection involves capturing essential user details like name and contact information. In a more advanced setup, this could also include capturing user preferences and vehicle details.

Vehicle Table

Source: Vehicle data is collected when a user registers a vehicle with the parking management system, which could be part of the user registration process.

Method: Key information like the vehicle's registration number, type, and owner's details are recorded. This data is crucial for linking vehicles to their respective owners and managing parking permissions.

EntryExitLog Table

Source: This data comes from the real-time monitoring of vehicles entering and exiting the parking lot.

Method: Automated systems such as RFID readers, license plate recognition cameras, or manual entry at a parking booth can be used to log the time of entry and exit of vehicles. This information is essential for tracking the usage of parking spaces and for billing purposes (if applicable).

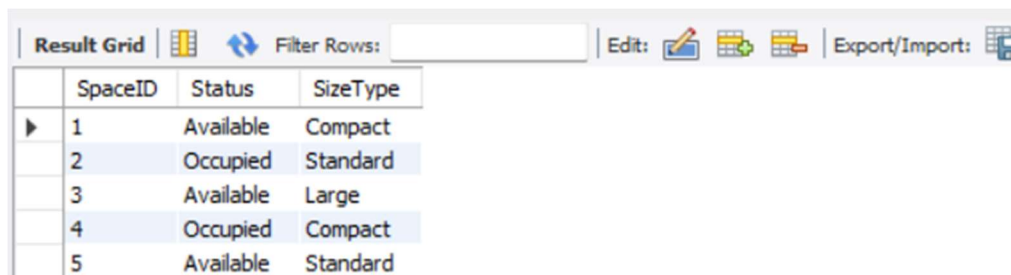
SQL query Development

Creating the tables and inserting the values, below are the queries and output.

1.

```
CREATE TABLE ParkingSpace (  
    SpaceID INT PRIMARY KEY,  
    Status VARCHAR(15) NOT NULL,  
    SizeType VARCHAR(50) NOT NULL  
);  
INSERT INTO ParkingSpace (SpaceID, Status, SizeType) VALUES  
(1, 'Available', 'Compact'),  
(2, 'Occupied', 'Standard'),  
(4, 'Occupied', 'Compact'),  
(5, 'Available', 'Standard');  
select * from parkingSpace
```

Output:

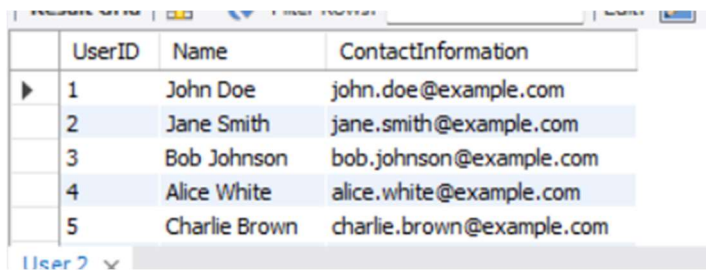


	SpaceID	Status	SizeType
▶	1	Available	Compact
	2	Occupied	Standard
	3	Available	Large
	4	Occupied	Compact
	5	Available	Standard

2.

```
CREATE TABLE User (  
    UserID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    ContactInformation VARCHAR(255) NOT NULL  
);  
  
INSERT INTO User (UserID, Name, ContactInformation) VALUES  
(1, 'John Doe', 'john.doe@example.com'),  
(2, 'Jane Smith', 'jane.smith@example.com'),  
(3, 'Bob Johnson', 'bob.johnson@example.com'),  
(4, 'Alice White', 'alice.white@example.com'),  
(5, 'Charlie Brown', 'charlie.brown@example.com'),
```

Output:



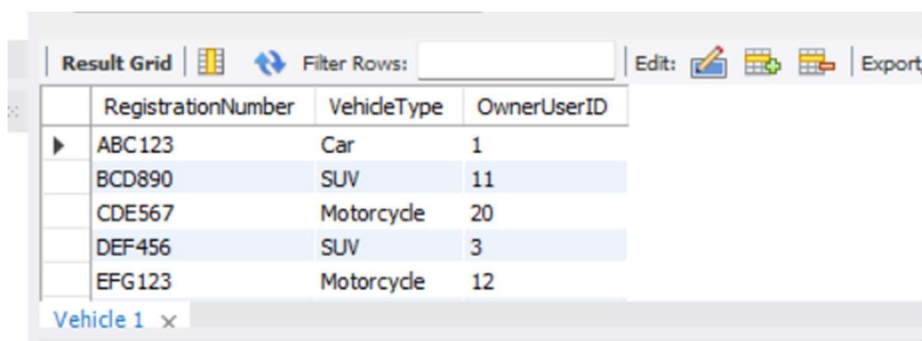
	UserID	Name	ContactInformation
▶	1	John Doe	john.doe@example.com
	2	Jane Smith	jane.smith@example.com
	3	Bob Johnson	bob.johnson@example.com
	4	Alice White	alice.white@example.com
	5	Charlie Brown	charlie.brown@example.com

User 2 ▼

3.

```
CREATE TABLE Vehicle (  
    RegistrationNumber VARCHAR(20) PRIMARY KEY,  
    VehicleType VARCHAR(50) NOT NULL,  
    OwnerUserID INT,  
    FOREIGN KEY (OwnerUserID) REFERENCES User(UserID)  
);  
INSERT INTO Vehicle (RegistrationNumber, VehicleType, OwnerUserID) VALUES  
( 'ABC123', 'Car', 1),  
( 'XYZ789', 'Truck', 2),  
( 'DEF456', 'SUV', 3),  
( 'GHI789', 'Motorcycle', 4),  
( 'JKL012', 'Car', 5);
```

Output:



	RegistrationNumber	VehicleType	OwnerUserID
▶	ABC123	Car	1
	BCD890	SUV	11
	CDE567	Motorcycle	20
	DEF456	SUV	3
	EFG123	Motorcycle	12

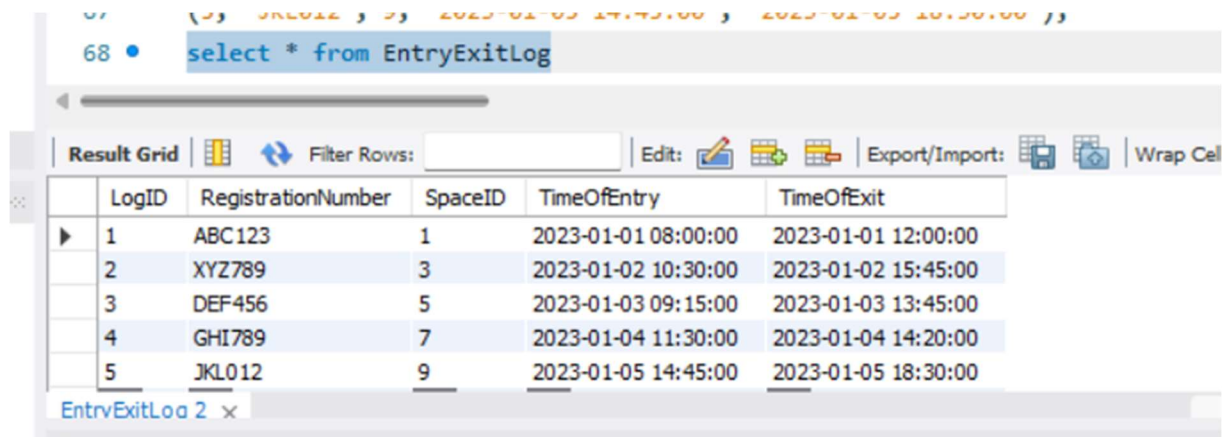
Vehicle 1 x

4.

```
CREATE TABLE EntryExitLog (  
    LogID INT PRIMARY KEY,  
    RegistrationNumber VARCHAR(20),  
    SpaceID INT,  
    TimeOfEntry TIMESTAMP NOT NULL,  
    TimeOfExit TIMESTAMP,  
    FOREIGN KEY (RegistrationNumber) REFERENCES Vehicle(RegistrationNumber),  
    FOREIGN KEY (SpaceID) REFERENCES ParkingSpace(SpaceID)  
);  
INSERT INTO EntryExitLog (LogID, RegistrationNumber, SpaceID, TimeOfEntry, TimeOfExit) VALUES  
(1, 'ABC123', 1, '2023-01-01 08:00:00', '2023-01-01 12:00:00'),  
(2, 'XYZ789', 3, '2023-01-02 10:30:00', '2023-01-02 15:45:00'),  
(3, 'DEF456', 5, '2023-01-03 09:15:00', '2023-01-03 13:45:00'),  
(4, 'GHI789', 7, '2023-01-04 11:30:00', '2023-01-04 14:20:00'),  
(5, 'JKL012', 9, '2023-01-05 14:45:00', '2023-01-05 18:30:00');
```

```
select * from EntryExitLog
```

Output:



The screenshot shows a SQL query editor with the query `select * from EntryExitLog` and its result grid. The result grid has columns: LogID, RegistrationNumber, SpaceID, TimeOfEntry, and TimeOfExit. It contains 5 rows of data.

LogID	RegistrationNumber	SpaceID	TimeOfEntry	TimeOfExit
1	ABC123	1	2023-01-01 08:00:00	2023-01-01 12:00:00
2	XYZ789	3	2023-01-02 10:30:00	2023-01-02 15:45:00
3	DEF456	5	2023-01-03 09:15:00	2023-01-03 13:45:00
4	GHI789	7	2023-01-04 11:30:00	2023-01-04 14:20:00
5	JKL012	9	2023-01-05 14:45:00	2023-01-05 18:30:00

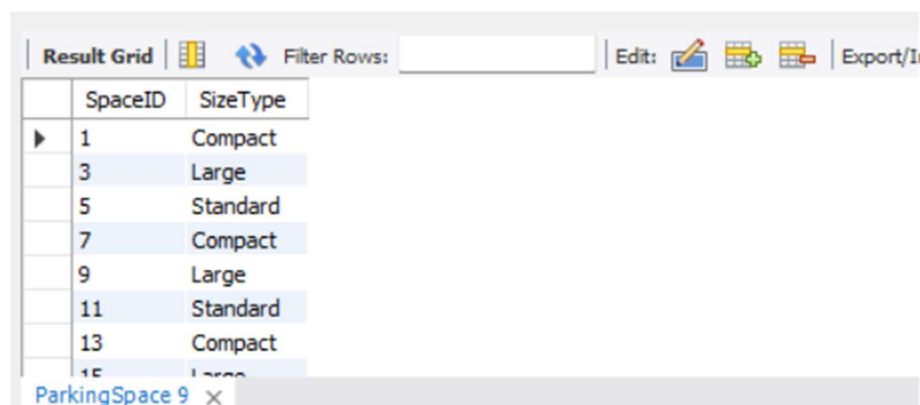
Data Analysis and Manipulation

For the Vehicle Parking Management System, several key SQL queries can be formulated to extract valuable information and insights from the database. Below are some examples of such queries along with their purposes and hypothetical results or insights that could be gained from them.

1) Query to retrieve available parking spaces

```
SELECT SpaceID, SizeType  
FROM ParkingSpace  
WHERE Status = 'Available';
```

Output:



The screenshot shows a SQL query result grid with columns: SpaceID and SizeType. It contains 8 rows of data.

SpaceID	SizeType
1	Compact
3	Large
5	Standard
7	Compact
9	Large
11	Standard
13	Compact
15	Large

2) Query for Retrieving Entry/Exit Logs for a Specific Vehicle

```
SELECT LogID, RegistrationNumber, TimeOfEntry, TimeOfExit  
FROM EntryExitLog  
WHERE RegistrationNumber = 'ABC123';
```

Output:

Result Grid	Filter Rows:	Edit:	Export/Import:
LogID	RegistrationNumber	TimeOfEntry	TimeOfExit
1	ABC123	2023-01-01 08:00:00	2023-01-01 12:00:00
21	ABC123	2023-02-01 08:30:00	2023-02-01 12:45:00
NULL	NULL	NULL	NULL

3. Calculate Average Parking Duration for Each Vehicle Type

```
SELECT VehicleType, AVG(TIMESTAMPDIFF(SECOND, TimeOfEntry, TimeOfExit) / 3600) AS AvgParkingDurationHours
```

```
FROM EntryExitLog
```

```
JOIN Vehicle ON EntryExitLog.RegistrationNumber = Vehicle.RegistrationNumber
```

```
GROUP BY VehicleType;
```

Output:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
VehideType	AvgParkingDurationHours		
Car	4.20605455		
SUV	4.29168000		
Motorcycle	3.38333000		
Truck	4.46363636		

4. Query to retrieve vehicle information and owner details

```
SELECT Vehicle.RegistrationNumber, Vehicle.VehicleType, User.Name AS OwnerName, User.ContactInformation AS OwnerContact
```

```
FROM Vehicle
```

```
LEFT JOIN User ON Vehicle.OwnerUserID = User.UserID;
```

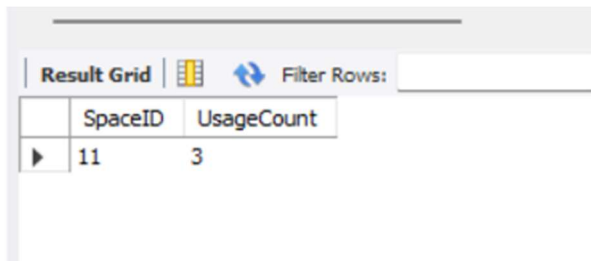
Output:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
RegistrationNumber	VehideType	OwnerName	OwnerContact
ABC123	Car	John Doe	john.doe@example.com
BCD890	SUV	Jack Wilson	jack.wilson@example.com
CDE567	Motorcycle	Tina Turner	tina.turner@example.com
DEF456	SUV	Bob Johnson	bob.johnson@example.com
EFG123	Motorcycle	Kelly Taylor	kelly.taylor@example.com
GHI789	Motorcycle	Alice White	alice.white@example.com
HIJ456	Car	Mike Evans	mike.evans@example.com
JKL012	Car	Charlie Brown	charlie.brown@example.com
KLM789	Truck	Nancy Reed	nancy.reed@example.com

5. Query to identify the most frequently used parking space

```
SELECT SpaceID, COUNT(*) AS UsageCount
FROM EntryExitLog
GROUP BY SpaceID
ORDER BY UsageCount DESC
LIMIT 1;
```

Output:



The screenshot shows a database interface with a 'Result Grid' tab. It displays a single row of data with two columns: 'SpaceID' and 'UsageCount'. The values are 11 and 3 respectively. Above the grid, there are icons for a grid, a refresh, and a filter, along with a 'Filter Rows:' input field.

	SpaceID	UsageCount
▶	11	3

Performance Tuning

Performance tuning in a database system like the Vehicle Parking Management System is crucial for ensuring quick and efficient data retrieval, especially as the volume of data grows over time. A key strategy in performance tuning is the use of indexing. Indexes are special lookup tables that the database search engine can use to speed up data retrieval.

```
#query optimization using indexing
.
.
-- Example index creation
• CREATE INDEX idx_parking_space_status ON ParkingSpace(Status);
• CREATE INDEX idx_vehicle_registration ON Vehicle(RegistrationNumber);
• CREATE INDEX idx_entry_exit_log_registration ON EntryExitLog(RegistrationNumber);
.
```

SQL Queries and Solutions

For the Vehicle Parking Management System, let's outline some key SQL queries, their purposes, and the types of insights or results these queries can provide.

1) Determine Parking Space Utilization Rates by Size/Type

Understanding how different types of parking spaces are utilized can inform decisions about layout changes or expansions.

```
SELECT ParkingSpace.SizeType, COUNT(EntryExitLog.LogID) AS TotalUsage, AVG(TIMESTAMPDIFF(HOUR,
TimeOfEntry, TimeOfExit)) AS AverageDuration
FROM ParkingSpace
LEFT JOIN EntryExitLog ON ParkingSpace.SpaceID = EntryExitLog.SpaceID
```

GROUP BY ParkingSpace.SizeType

ORDER BY TotalUsage DESC;

Output:

Result Grid			
		Filter Rows:	
		Export:	Wrap Cell Content:
	SizeType	TotalUsage	AverageDuration
▶	Standard	16	4.0625
	Compact	14	3.5714
	Large	12	3.2500

Tuple Relational Calculus

P as a tuple in the ParkingSpace table.

E as a tuple in the EntryExitLog table.

T as the resulting tuple with attributes SizeType, TotalUsage, and AverageDuration.

$$\{T | P \in \text{ParkingSpace} \wedge T.\text{SizeType} = P.\text{SizeType} \wedge (E \in \text{EntryExitLog} \Rightarrow P.\text{SpaceID} = E.\text{SpaceID}) \wedge T.\text{TotalUsage} = \text{COUNT}(E.\text{LogID}) \wedge T.\text{AverageDuration} = \text{AVG}(\text{TIMESTAMPDIFF}(\text{HOUR}, E.\text{TimeOfEntry}, E.\text{TimeOfExit})) \wedge (\forall P' \in \text{ParkingSpace}) [P'.\text{SizeType} = P.\text{SizeType} \Rightarrow T.\text{TotalUsage} \geq \text{COUNT}(E'.\text{LogID} \text{ where } P'.\text{SpaceID} = E'.\text{SpaceID})]\}$$

2) Find Total Parking Hours for Each User

```
DELIMITER //
CREATE PROCEDURE TotalParkingHoursPerUser()
BEGIN
    SELECT User.UserID, User.Name, SUM(ParkingDuration) AS TotalHours
    FROM (
        SELECT OwnerUserID, TIMESTAMPDIFF(HOUR, TimeOfEntry, TimeOfExit) AS ParkingDuration
        FROM EntryExitLog
        JOIN Vehicle ON EntryExitLog.RegistrationNumber = Vehicle.RegistrationNumber
    ) AS ParkingTimes
    JOIN User ON ParkingTimes.OwnerUserID = User.UserID
    GROUP BY User.UserID;
END //
DELIMITER ;
```

Call TotalParkingHoursPerUser;

Output:

Result Grid	Filter Rows:	Export:
UserID	Name	TotalHours
1	John Doe	8
2	Jane Smith	9
3	Bob Johnson	9
4	Alice White	6
5	Charlie Brown	7

Tuple Relational Calculus

$\{T \mid U \in \text{User} \wedge T.UserID = U.UserID \wedge T.Name = U.Name \wedge T.TotalHours = \sum(\{E.TimeStampDiff \mid E \in \text{EntryExitLog} \wedge V \in \text{Vehicle} \wedge E.RegistrationNumber = V.RegistrationNumber \wedge V.OwnerUserID = U.UserID\})\}$

U as a tuple variable for the User table.

E as a tuple variable for the EntryExitLog table.

V as a tuple variable for the Vehicle table.

T as the resulting tuple with attributes UserID, Name, and TotalHours.

3) Find Users Who Have Parked More Than Once on the Same Day

```
SELECT DISTINCT U.UserID, U.Name
```

```
FROM User U
```

```
INNER JOIN Vehicle V ON U.UserID = V.OwnerUserID
```

```
INNER JOIN (
```

```
    SELECT RegistrationNumber, Date(TimeOfEntry) AS EntryDate
```

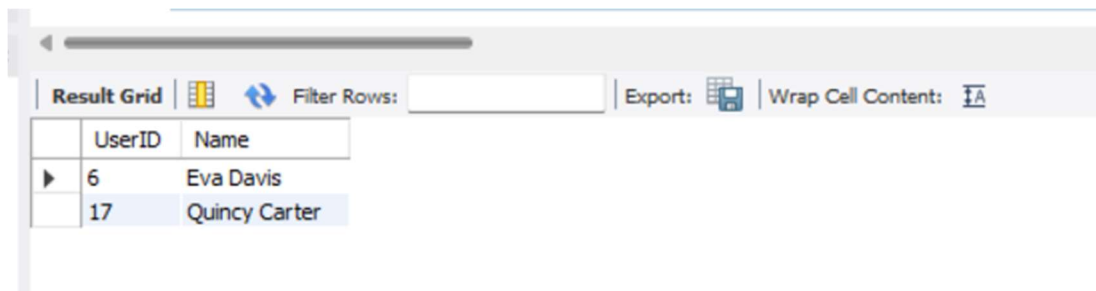
```
    FROM EntryExitLog
```

```
    GROUP BY RegistrationNumber, EntryDate
```

```
    HAVING COUNT(*) > 1
```

```
) AS MultipleEntries ON V.RegistrationNumber = MultipleEntries.RegistrationNumber;
```

Output:



	UserID	Name
▶	6	Eva Davis
	17	Quincy Carter

Tuple Relational Calculus

U for tuples in the User table.

V for tuples in the Vehicle table.

E for tuples in the EntryExitLog table.

T for tuples in the resulting set.

$$\{T | U \in \text{User} \wedge V \in \text{Vehicle} \wedge U.UserID = V.OwnerUserID \wedge (\exists E1 \in \text{EntryExitLog})(\exists E2 \in \text{EntryExitLog})(E1.RegistrationNumber = E2.RegistrationNumber \wedge E1.RegistrationNumber = V.RegistrationNumber \wedge Date(E1.TimeOfEntry) = Date(E2.TimeOfEntry) \wedge T.UserID = U.UserID \wedge T.Name = U.Name)\}$$

Limitations and future work

Current Limitations

- 1.Lack of Real-Time Data Integration:** The current system primarily relies on historical data. It may not incorporate real-time data, which is essential for dynamic parking management.
- 2.Limited Predictive Analytics:** Without advanced analytics, the system may not effectively predict future parking trends or user behaviors.
- 3.Front-End User Interface Absence:** Currently, the system lacks a user-friendly front-end interface, which limits direct interaction with users.
- 4.Scalability Concerns:** As the number of users and vehicles increases, the system might face challenges in efficiently handling large volumes of data.
- 5.Integration with External Systems:** There's limited integration with external systems like payment gateways or traffic management systems.
- 6.Limited Customization for Different Parking Scenarios:** The system may not be fully adaptable to different types of parking facilities, such as street parking vs. multi-level garages.

Scope for future work

- 1.Real-Time Data Processing:** Incorporating IoT devices for real-time space monitoring and updating the system in real-time to improve parking space allocation.
- 2.Advanced Analytics and Machine Learning:** Implementing predictive analytics to forecast parking availability and demand, thus aiding in strategic planning and dynamic pricing models.
- 3.Development of a User-Friendly Front-End:** Creating a mobile app or web interface for users to find, reserve, and pay for parking spaces.
- 4.Enhanced Scalability:** Upgrading the database infrastructure to handle larger datasets and more complex queries without compromising performance.
- 5.Integration with Payment and Traffic Systems:** Building interfaces for seamless integration with payment gateways for easy billing and with city traffic systems for coordinated traffic management.

References:

1. https://www.w3schools.com/sql/sql_stored_procedures.asp
2. <https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/>
3. <https://stackoverflow.com/questions/24718701/mysql-database-design-for-a-car-parking-garage>

Github Repository :

<https://github.com/kiranm7753/Databasedesign-Project>