PARKING MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

SRIMAN E [RA2211003011568]

KEERTHIVARSHA J [RA2211003011598]

ABIDEEPADARSAN S K [RA2211003011607]

Under the Guidance of

Dr. ROBERT P

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Register no: [RA2211003011568], [RA2211003011598],[RA2211003011607]

Certified to be the Bonafede work done **Sriman E, Keerthivarsha J, Abideepadarsan S K** of II year/IV Sem B. Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date:

Faculty in Charge
Dr. Robert P
Assistant professor
C Tech
SRMIST -KTR

HEAD OF THE DEPARTMENT
Dr. M. Pushpalatha
C Tech
SRMIST - KTR

ABSTRACT

Our project aims to develop a user-friendly graphical interface for a comprehensive parking management system using Python. The GUI will serve as the primary interaction point for both administrators and users, providing intuitive controls for parking space allocation, vehicle entry/exit tracking, payment processing, and administrative tasks.

The system will feature a visually appealing layout designed to enhance user experience and streamline parking operations. Through seamless integration with backend database operations, the GUI will facilitate efficient management of parking spaces, real-time monitoring of vehicle movement, and generation of insightful analytics for informed decision-making.

PROBLEM STATEMENT

In urban areas, efficient management of parking spaces is crucial for alleviating traffic congestion and providing convenience to drivers. Traditional parking management systems often rely on manual processes, leading to inefficiencies, long waiting times, and frustration among users. To address these challenges, a modern parking management system is needed that automates parking space allocation, facilitates seamless vehicle entry/exit, and provides real-time monitoring and analytics capabilities.

Develop a comprehensive parking management system that incorporates a user-friendly graphical interface for administrators and users. The system should enable efficient management of parking spaces, streamline vehicle entry/exit processes, facilitate payment processing, and provide insights through analytics.

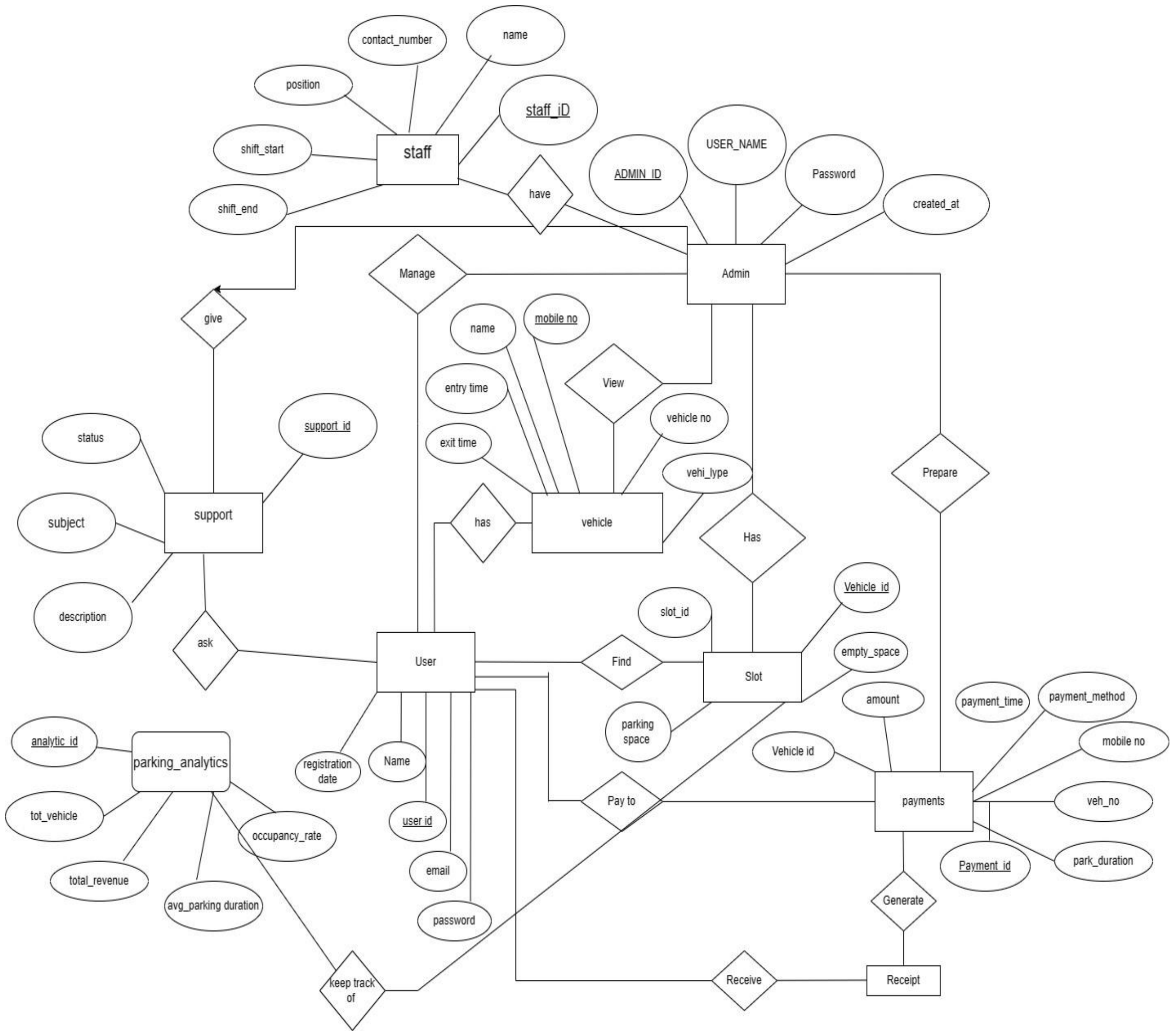
TABLE OF CONTENTS

Problem Understanding:

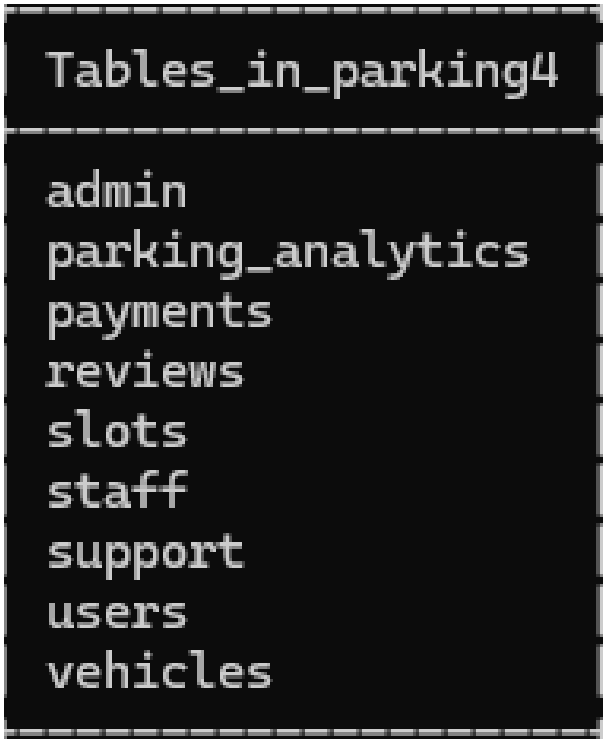
Understanding the problem statement is crucial for any project. In the case of your parking management system, it involves grasping the requirements and constraints of the system you're building. This includes understanding the data that needs to be stored, the actions users can perform, and any external factors affecting the system's behavior. Once you have a clear understanding of the problem, you can effectively design and implement solutions using SQLPlus with Oracle and Python. If you're struggling with understanding the project requirements, breaking them down into smaller, manageable parts and seeking clarification from stakeholders can help.

Understanding the problem for your parking management system involves identifying key components such as user roles, system functionalities, data storage requirements, and potential challenges like concurrency and security. It requires careful analysis and communication with stakeholders to ensure that the final solution meets the needs of all users and adheres to project constraints. By breaking down the problem into manageable chunks and addressing each aspect systematically, you can gain a comprehensive understanding and proceed with confidence in your project development.

Identification of Entity and Relationships:



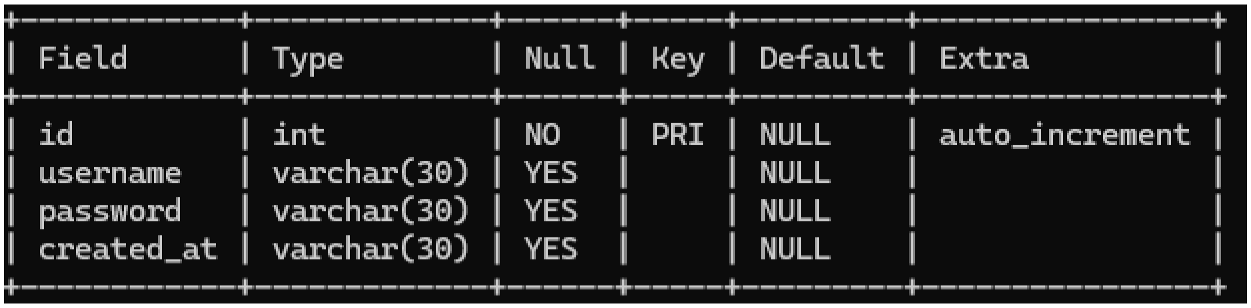
Construction of DB using ER Model:



```
+----------------------+
| Tables_in_parking4   |
+----------------------+
| admin                |
| parking_analytics    |
| payments             |
| reviews              |
| slots                |
| staff                |
| support              |
| users                |
| vehicles             |
+----------------------+
```

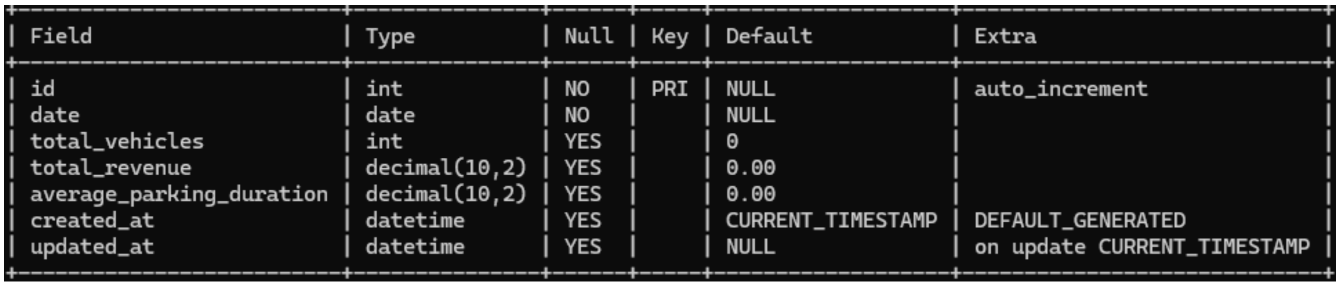
Creation of Database Tables

Table-1 Admin



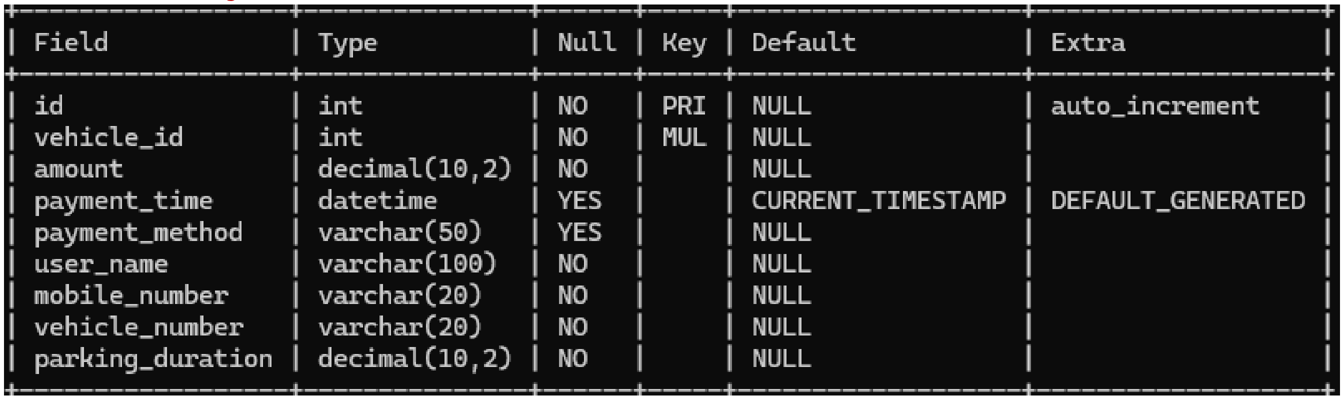
```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int         | NO   | PRI | NULL    | auto_increment |
| username   | varchar(30) | YES  |     | NULL    |                |
| password   | varchar(30) | YES  |     | NULL    |                |
| created_at | varchar(30) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Table-2 Parking_analytics



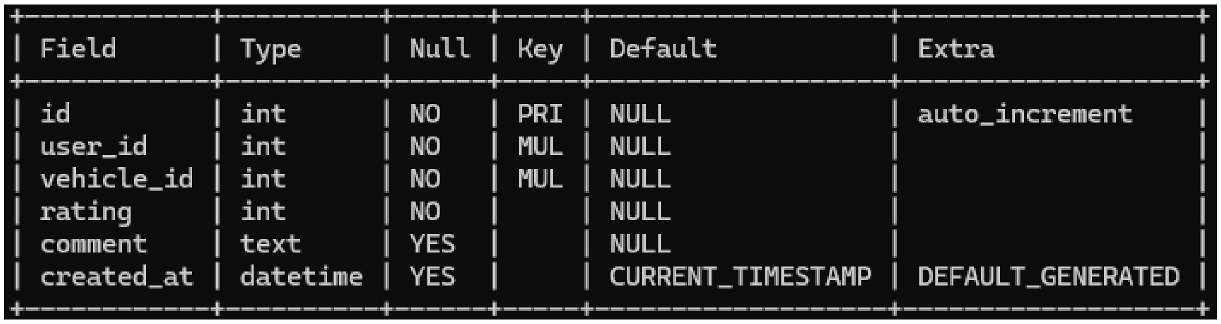
```
+--------------------------+---------------+------+-----+-------------------+-----------------------------+
| Field                    | Type          | Null | Key | Default           | Extra                       |
+--------------------------+---------------+------+-----+-------------------+-----------------------------+
| id                       | int           | NO   | PRI | NULL              | auto_increment              |
| date                     | date          | NO   |     | NULL              |                             |
| total_vehicles           | int           | YES  |     | 0                 |                             |
| total_revenue            | decimal(10,2) | YES  |     | 0.00              |                             |
| average_parking_duration | decimal(10,2) | YES  |     | 0.00              |                             |
| created_at               | datetime      | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED           |
| updated_at               | datetime      | YES  |     | NULL              | on update CURRENT_TIMESTAMP |
+--------------------------+---------------+------+-----+-------------------+-----------------------------+
```

Table-3 Payments



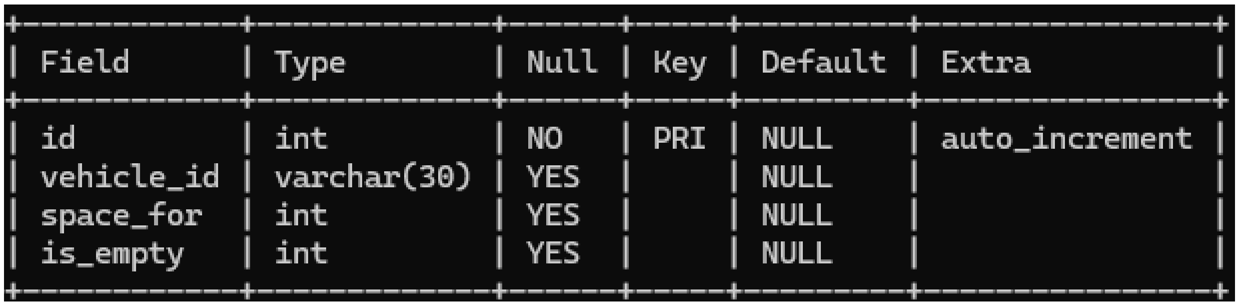
```
+------------------+---------------+------+-----+-------------------+-------------------+
| Field            | Type          | Null | Key | Default           | Extra             |
+------------------+---------------+------+-----+-------------------+-------------------+
| id               | int           | NO   | PRI | NULL              | auto_increment    |
| vehicle_id       | int           | NO   | MUL | NULL              |                   |
| amount           | decimal(10,2) | NO   |     | NULL              |                   |
| payment_time     | datetime      | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
| payment_method   | varchar(50)   | YES  |     | NULL              |                   |
| user_name        | varchar(100)  | NO   |     | NULL              |                   |
| mobile_number    | varchar(20)   | NO   |     | NULL              |                   |
| vehicle_number   | varchar(20)   | NO   |     | NULL              |                   |
| parking_duration | decimal(10,2) | NO   |     | NULL              |                   |
+------------------+---------------+------+-----+-------------------+-------------------+
```

Table-4 Reviews



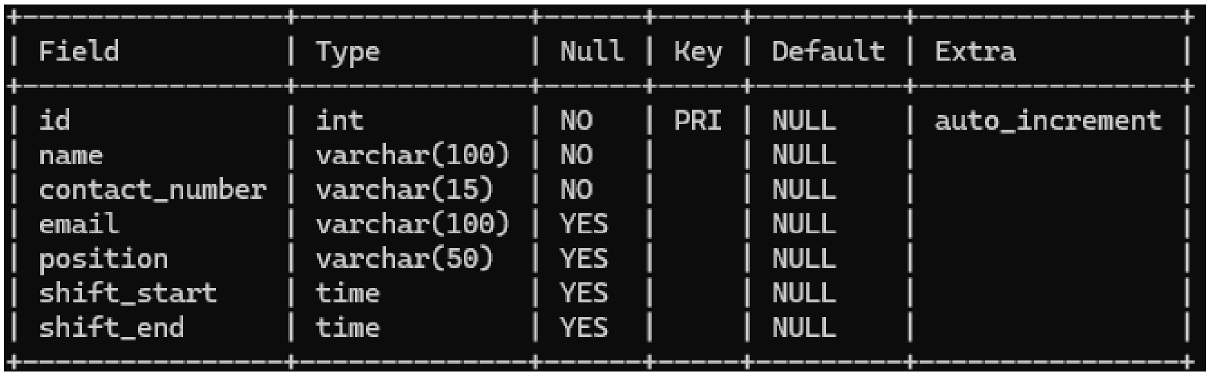
```
+------------+----------+------+-----+-------------------+-------------------+
| Field      | Type     | Null | Key | Default           | Extra             |
+------------+----------+------+-----+-------------------+-------------------+
| id         | int      | NO   | PRI | NULL              | auto_increment    |
| user_id    | int      | NO   | MUL | NULL              |                   |
| vehicle_id | int      | NO   | MUL | NULL              |                   |
| rating     | int      | NO   |     | NULL              |                   |
| comment    | text     | YES  |     | NULL              |                   |
| created_at | datetime | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+----------+------+-----+-------------------+-------------------+
```

Table-5 Slots



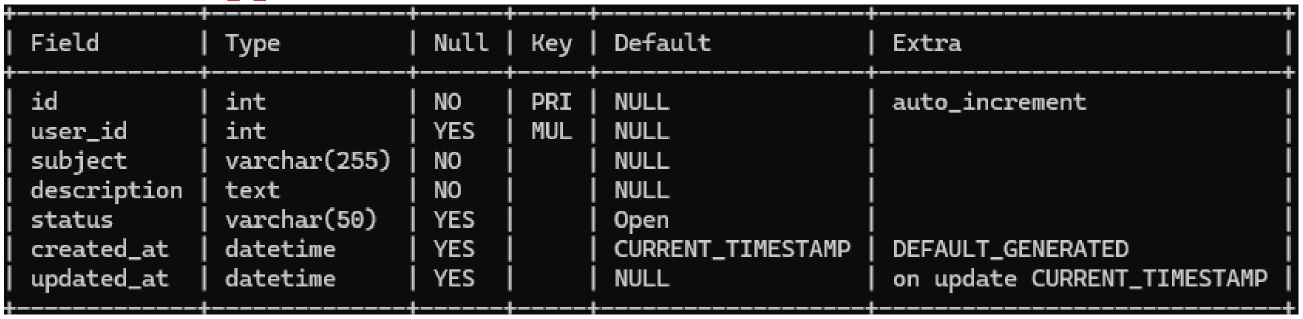
```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int         | NO   | PRI | NULL    | auto_increment |
| vehicle_id | varchar(30) | YES  |     | NULL    |                |
| space_for  | int         | YES  |     | NULL    |                |
| is_empty   | int         | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Table-6 Staff



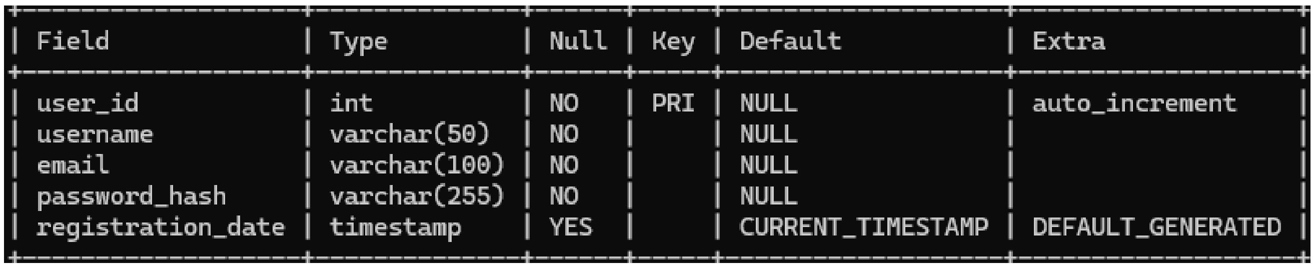
```
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| name           | varchar(100) | NO   |     | NULL    |                |
| contact_number | varchar(15)  | NO   |     | NULL    |                |
| email          | varchar(100) | YES  |     | NULL    |                |
| position       | varchar(50)  | YES  |     | NULL    |                |
| shift_start    | time         | YES  |     | NULL    |                |
| shift_end      | time         | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
```

Table-7 Support



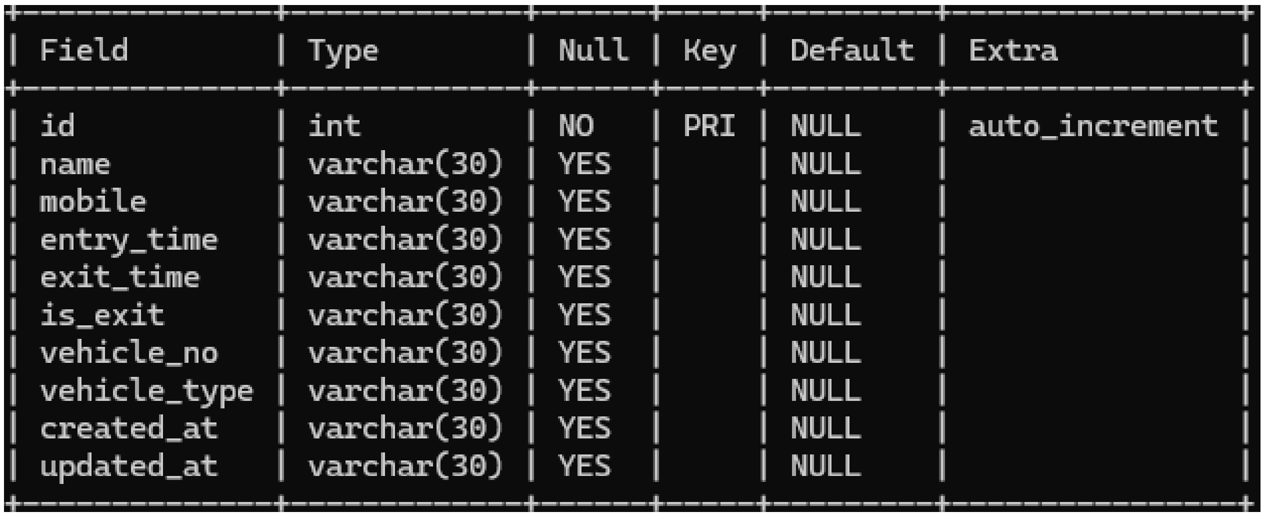
```
+-------------+--------------+------+-----+-------------------+-----------------------------+
| Field       | Type         | Null | Key | Default           | Extra                       |
+-------------+--------------+------+-----+-------------------+-----------------------------+
| id          | int          | NO   | PRI | NULL              | auto_increment              |
| user_id     | int          | YES  | MUL | NULL              |                             |
| subject     | varchar(255) | NO   |     | NULL              |                             |
| description | text         | NO   |     | NULL              |                             |
| status      | varchar(50)  | YES  |     | Open              |                             |
| created_at  | datetime     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED           |
| updated_at  | datetime     | YES  |     | NULL              | on update CURRENT_TIMESTAMP |
+-------------+--------------+------+-----+-------------------+-----------------------------+
```

Table-8 Users



```
+-------------------+--------------+------+-----+-------------------+-------------------+
| Field             | Type         | Null | Key | Default           | Extra             |
+-------------------+--------------+------+-----+-------------------+-------------------+
| user_id           | int          | NO   | PRI | NULL              | auto_increment    |
| username          | varchar(50)  | NO   |     | NULL              |                   |
| email             | varchar(100) | NO   |     | NULL              |                   |
| password_hash     | varchar(255) | NO   |     | NULL              |                   |
| registration_date | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------------+--------------+------+-----+-------------------+-------------------+
```

Table-9 Vehicles



```
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| id           | int         | NO   | PRI | NULL    | auto_increment |
| name         | varchar(30) | YES  |     | NULL    |                |
| mobile       | varchar(30) | YES  |     | NULL    |                |
| entry_time   | varchar(30) | YES  |     | NULL    |                |
| exit_time    | varchar(30) | YES  |     | NULL    |                |
| is_exit      | varchar(30) | YES  |     | NULL    |                |
| vehicle_no   | varchar(30) | YES  |     | NULL    |                |
| vehicle_type | varchar(30) | YES  |     | NULL    |                |
| created_at   | varchar(30) | YES  |     | NULL    |                |
| updated_at   | varchar(30) | YES  |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
```

Design of Relational Schemas :

Designing relational schemas involves organizing data into tables and establishing relationships between them. First, identify entities and their attributes. Normalize data to reduce redundancy and anomalies. Define relationships and choose keys to represent them. Establish constraints for data integrity. Denormalization may be necessary for performance. Review and refine the design, documenting it for reference. Implement the schema in your database system and test thoroughly. This structured approach ensures effective data modeling and supports application requirements.

Queries For Creation Of Tables:

TABLE USER:

```
CREATE TABLE users (
user_id INT NOT NULL AUTO_INCREMENT,
username VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL,
password_hash VARCHAR(255) NOT NULL,
registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (user_id)
);
```

TABLE PAYMENTS;

```
CREATE TABLE payments (
id int(255) NOT NULL AUTO_INCREMENT,
vehicle_id int(255) NOT NULL,
amount decimal(10,2) NOT NULL,
payment_time datetime DEFAULT CURRENT_TIMESTAMP,
payment_method varchar(50) DEFAULT NULL,
user_name varchar(100) NOT NULL,
mobile_number varchar(20) NOT NULL,
vehicle_number varchar(20) NOT NULL,
parking_duration decimal(10,2) NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (vehicle_id) REFERENCES vehicles (id)
ALTER TABLE payments
MODIFY id int(255) NOT NULL AUTO_INCREMENT;
```

TABLE PARKING ANALYTICS;

```
CREATE TABLE parking_analytics (
id int(255) NOT NULL,
date date NOT NULL,
total_vehicles int(255) DEFAULT 0,
total_revenue decimal(10,2) DEFAULT 0.00,
average_parking_duration decimal(10,2) DEFAULT 0.00, -- in hours
occupancy_rate decimal(5,2) DEFAULT 0.00, -- percentage of occupied slots
created_at datetime DEFAULT CURRENT_TIMESTAMP,
updated_at datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP
)
ALTER TABLE parking_analytics
ADD PRIMARY KEY (id);
```

TABLE SUPPORT;

```
CREATE TABLE support (
id int(255) NOT NULL,
user_id int(255) DEFAULT NULL,
subject varchar(255) NOT NULL,
description text NOT NULL,
status varchar(50) DEFAULT 'Open', -- Example statuses: Open, In Progress, Resolved, Closed
created_at datetime DEFAULT CURRENT_TIMESTAMP,
updated_at datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP
)
ALTER TABLE support
ADD PRIMARY KEY (id),
ADD KEY user_id (user_id);
```

TABLE STAFF;

```
CREATE TABLE staff (
id int(255) NOT NULL,
name varchar(100) NOT NULL,
contact_number varchar(15) NOT NULL,
email varchar(100) DEFAULT NULL,
position varchar(50) DEFAULT NULL,
shift_start time DEFAULT NULL,
shift_end time DEFAULT NULL)
ALTER TABLE payments
ADD PRIMARY KEY (id),
ADD KEY vehicle_id (vehicle_id);
```

TABLE VEHICLES;

```
CREATE TABLE vehicles (
id int(255) NOT NULL,
name varchar(30) DEFAULT NULL,
mobile varchar(30) DEFAULT NULL,
entry_time varchar(30) DEFAULT NULL,
exit_time varchar(30) DEFAULT NULL,
is_exit varchar(30) DEFAULT NULL,
vehicle_no varchar(30) DEFAULT NULL,
vehicle_type varchar(30) DEFAULT NULL,
created_at varchar(30) DEFAULT NULL,
updated_at varchar(30) DEFAULT NULL)
ALTER TABLE vehicles
ADD PRIMARY KEY (id);
```

TABLE SLOTS;

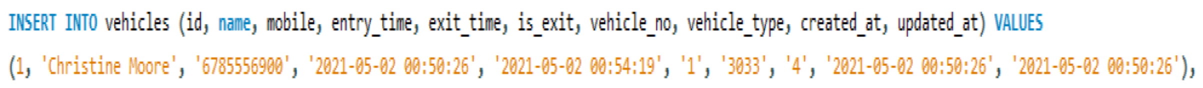
```
CREATE TABLE slots (
  id int(255) NOT NULL,
  vehicle_id varchar(30) DEFAULT NULL,
  space_for int(25) DEFAULT NULL,
  is_empty int(25) DEFAULT NULL
) ALTER TABLE slots
  ADD PRIMARY KEY (id);
```

TABLE ADMIN;

```
CREATE TABLE admin (
  id int(255) NOT NULL,
  username varchar(30) DEFAULT NULL,
  password varchar(30) DEFAULT NULL,
  created_at varchar(30) DEFAULT NULL
) ALTER TABLE admin
  ADD PRIMARY KEY (id);
```

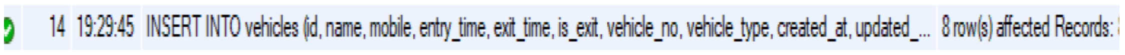
Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors:

Inserting Data:



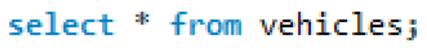
```
INSERT INTO vehicles (id, name, mobile, entry_time, exit_time, is_exit, vehicle_no, vehicle_type, created_at, updated_at) VALUES
(1, 'Christine Moore', '6785556900', '2021-05-02 00:50:26', '2021-05-02 00:54:19', '1', '3033', '4', '2021-05-02 00:50:26', '2021-05-02 00:50:26'),
```

-DML (Data Manipulation Language) - Represents a collection of programming languages explicitly used to make changes to the database, such as: CRUD operations to create, read, update and delete data. Using INSERT, SELECT, UPDATE, and DELETE commands.
-Operation : Insert operation.
-Work: It inserts the given data to the table.
Output



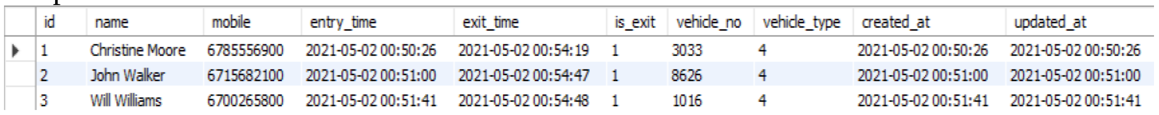
```
14  19:29:45  INSERT INTO vehicles (id, name, mobile, entry_time, exit_time, is_exit, vehicle_no, vehicle_type, created_at, updated_...  8 row(s) affected Records:
```

Viewing Data:



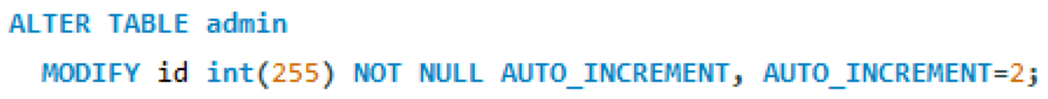
```
select * from vehicles;
```

-DDL (Data Definition Language) - It retrieves data from the database without modifying it.
-Operation : Select operation.
-Work: Retrieves all categories from the "category" table to populate a dropdown menu of vehicles.
Output



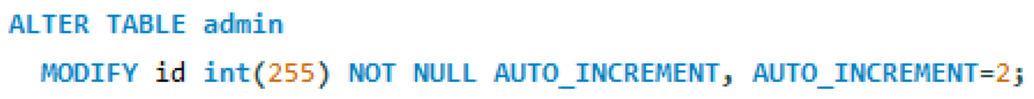
id	name	mobile	entry_time	exit_time	is_exit	vehicle_no	vehicle_type	created_at	updated_at
1	Christine Moore	6785556900	2021-05-02 00:50:26	2021-05-02 00:54:19	1	3033	4	2021-05-02 00:50:26	2021-05-02 00:50:26
2	John Walker	6715682100	2021-05-02 00:51:00	2021-05-02 00:54:47	1	8626	4	2021-05-02 00:51:00	2021-05-02 00:51:00
3	Will Williams	6700265800	2021-05-02 00:51:41	2021-05-02 00:54:48	1	1016	4	2021-05-02 00:51:41	2021-05-02 00:51:41

Alter table Admin:



```
ALTER TABLE admin
  MODIFY id int(255) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
```

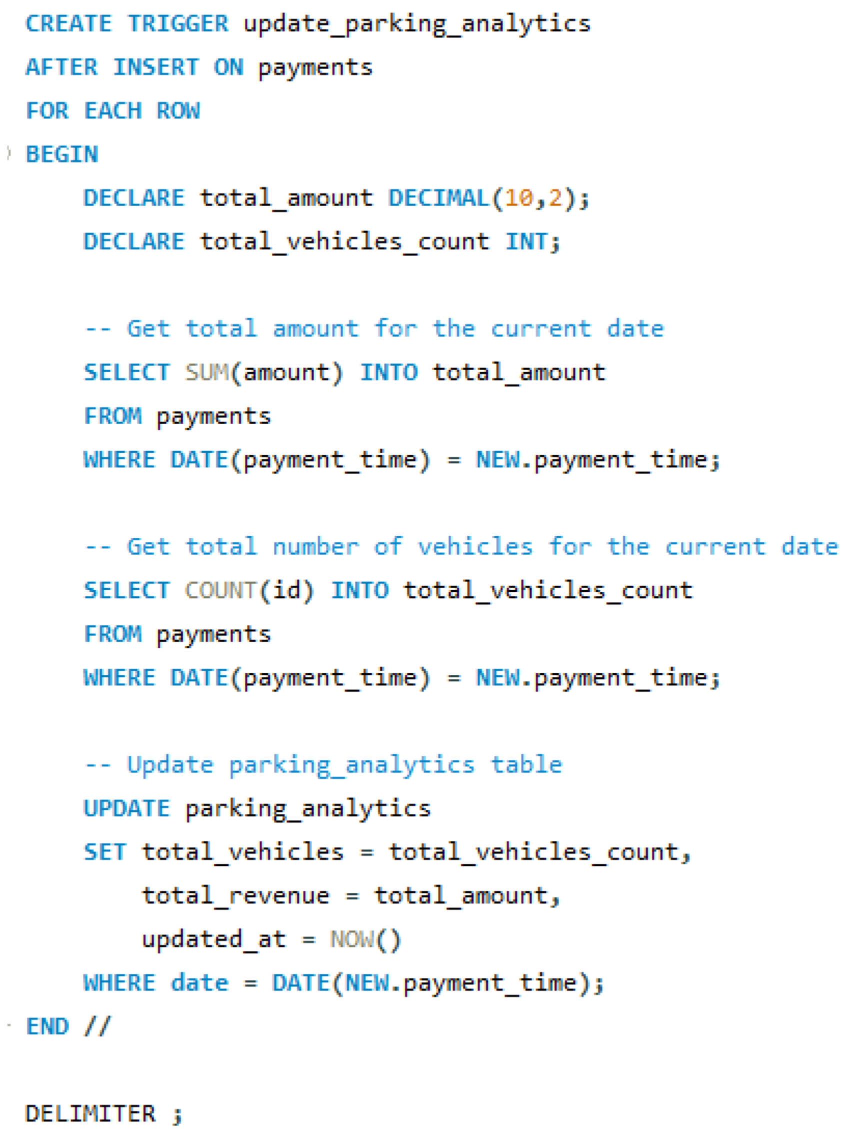
-DML (Data Manipulation Language) – It modifies data in the database.
-Operation : Alter operation.
-Work: To modify table admin and add auto increment function.



```
ALTER TABLE admin
  MODIFY id int(255) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
```

Trigger Operation:

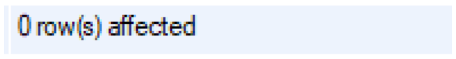
Trigger Creation – Update Parking Analytics:



```
CREATE TRIGGER update_parking_analytics
AFTER INSERT ON payments
FOR EACH ROW
BEGIN
    DECLARE total_amount DECIMAL(10,2);
    DECLARE total_vehicles_count INT;

    -- Get total amount for the current date
    SELECT SUM(amount) INTO total_amount
    FROM payments
    WHERE DATE(payment_time) = NEW.payment_time;

    -- Get total number of vehicles for the current date
    SELECT COUNT(id) INTO total_vehicles_count
    FROM payments
    WHERE DATE(payment_time) = NEW.payment_time;

    -- Update parking_analytics table
    UPDATE parking_analytics
    SET total_vehicles = total_vehicles_count,
        total_revenue = total_amount,
        updated_at = NOW()
    WHERE date = DATE(NEW.payment_time);
END //

DELIMITER ;
```

- Operation: Trigger Creation
- Work: Establishes a trigger to find total count of cars and total amount collected.
- DDL/DML: DDL (Data Definition Language)

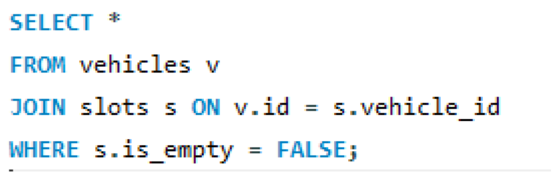
Output



```
0 row(s) affected
```

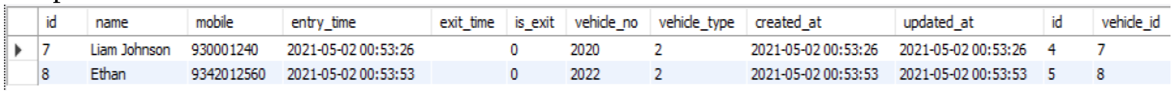
Joins and Unions:

Join Query



```
SELECT *
FROM vehicles v
JOIN slots s ON v.id = s.vehicle_id
WHERE s.is_empty = FALSE;
```

Operation: Join Query
Work: Retrieves all vehicles currently parked.
DDL/DML: DML (Data Manipulation Language)
Output

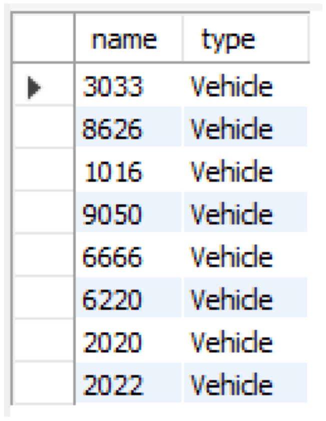


id	name	mobile	entry_time	exit_time	is_exit	vehicle_no	vehicle_type	created_at	updated_at	id	vehicle_id
7	Liam Johnson	930001240	2021-05-02 00:53:26		0	2020	2	2021-05-02 00:53:26	2021-05-02 00:53:26	4	7
8	Ethan	9342012560	2021-05-02 00:53:53		0	2022	2	2021-05-02 00:53:53	2021-05-02 00:53:53	5	8

Union Query:

```
SELECT
    u.username AS name,
    'User' AS type
FROM
    Users u

UNION

SELECT
    v.vehicle_no AS name,
    'Vehicle' AS type
FROM
    Vehicles v;
```

Operation: Union Query
Work: Combines Usename and vehicle numbers.
DDL/DML: DML (Data Manipulation Language)
Output



name	type
3033	Vehicle
8626	Vehicle
1016	Vehicle
9050	Vehicle
6666	Vehicle
6220	Vehicle
2020	Vehicle
2022	Vehicle

PITFALLS AND NORMALIZATION:

1. Table USERS:

This table seems to be in the first normal form (1NF) already since it doesn't contain repeating groups.

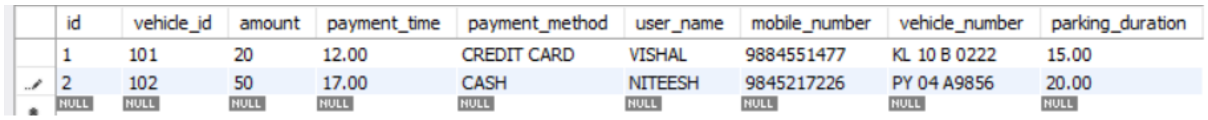
2. Table PAYMENTS:

This table has some normalization issues:

Partial Dependency: Columns user_name, mobile_number, vehicle_number are functionally dependent on id, amount, payment_time, payment_method, and parking_duration. They should be moved to another table.

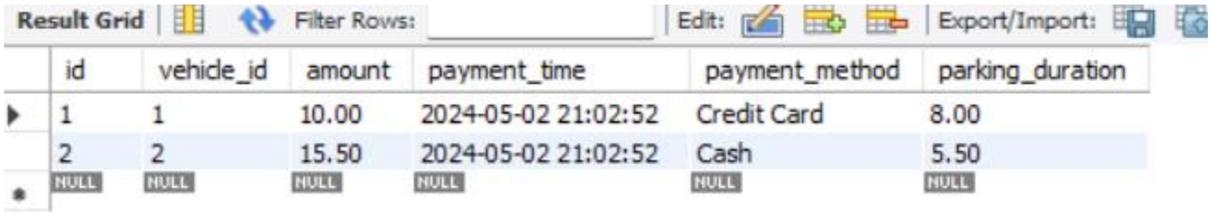
Transitive Dependency: vehicle_id determines payment_time, payment_method, and parking_duration, which are not directly dependent on the primary key.

BEFORE NORMALIZATION;

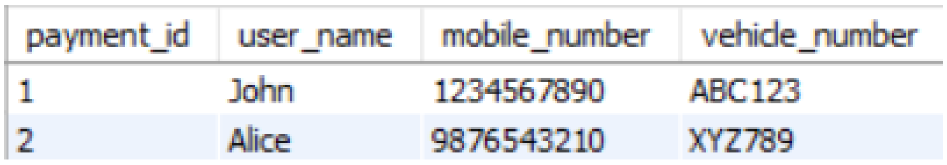


id	vehicle_id	amount	payment_time	payment_method	user_name	mobile_number	vehicle_number	parking_duration
1	101	20	12.00	CREDIT CARD	VISHAL	9884551477	KL 10 B 0222	15.00
2	102	50	17.00	CASH	NITEESH	9845217226	PY 04 A9856	20.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

AFTER NORMALIZATION;



id	vehicle_id	amount	payment_time	payment_method	parking_duration
1	1	10.00	2024-05-02 21:02:52	Credit Card	8.00
2	2	15.50	2024-05-02 21:02:52	Cash	5.50
NULL	NULL	NULL	NULL	NULL	NULL



payment_id	user_name	mobile_number	vehicle_number
1	John	1234567890	ABC123
2	Alice	9876543210	XYZ789

3. Table PARKING_ANALYTICS:

This table appears to be in the first normal form (1NF) already.

4. Table SUPPORT:

This table seems to be in the first normal form (1NF) already.

5. Table STAFF:

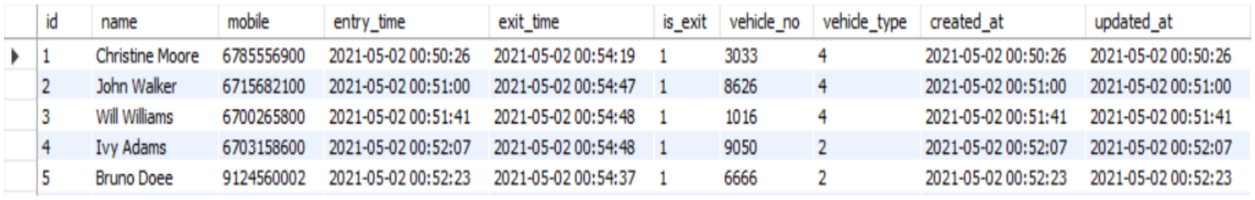
This table appears to be in the first normal form (1NF) already.

6. **Table VEHICLES:**

This table appears to have normalization issues:

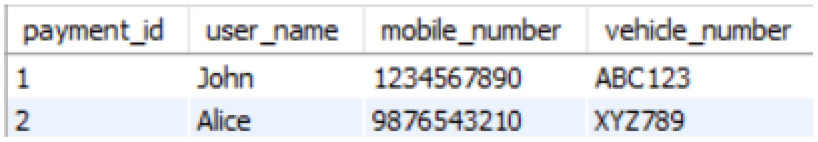
The entry_time, exit_time, created_at, and updated_at columns are storing datetime values as varchar. These should be converted to proper datetime data types.

BEFORE NORMALIZATION;

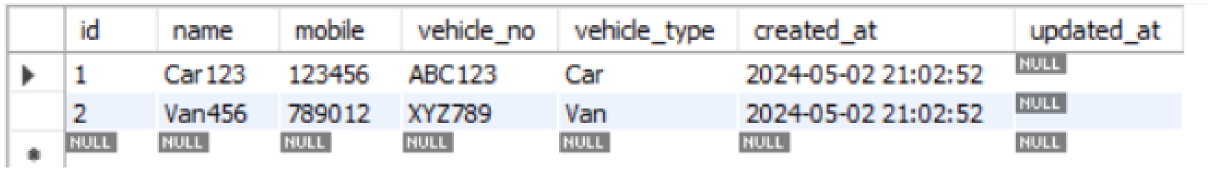


id	name	mobile	entry_time	exit_time	is_exit	vehicle_no	vehicle_type	created_at	updated_at
1	Christine Moore	6785556900	2021-05-02 00:50:26	2021-05-02 00:54:19	1	3033	4	2021-05-02 00:50:26	2021-05-02 00:50:26
2	John Walker	6715682100	2021-05-02 00:51:00	2021-05-02 00:54:47	1	8626	4	2021-05-02 00:51:00	2021-05-02 00:51:00
3	Will Williams	6700265800	2021-05-02 00:51:41	2021-05-02 00:54:48	1	1016	4	2021-05-02 00:51:41	2021-05-02 00:51:41
4	Ivy Adams	6703158600	2021-05-02 00:52:07	2021-05-02 00:54:48	1	9050	2	2021-05-02 00:52:07	2021-05-02 00:52:07
5	Bruno Doee	9124560002	2021-05-02 00:52:23	2021-05-02 00:54:37	1	6666	2	2021-05-02 00:52:23	2021-05-02 00:52:23

AFTER NORMALIZATION;



payment_id	user_name	mobile_number	vehicle_number
1	John	1234567890	ABC123
2	Alice	9876543210	XYZ789



id	name	mobile	vehicle_no	vehicle_type	created_at	updated_at
1	Car123	123456	ABC123	Car	2024-05-02 21:02:52	NULL
2	Van456	789012	XYZ789	Van	2024-05-02 21:02:52	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. **Table SLOTS:**

This table appears to be in the first normal form (1NF) already.

8. Table ADMIN:

This table appears to be in the first normal form (1NF) already.

FUNCTIONAL DEPENDENCIES:

Functional Dependencies:

users:

user_id -> {username, email, password_hash, registration_date}

email -> {username, user_id, password_hash, registration_date}

payments:

id -> {vehicle_id, amount, payment_time, payment_method, user_name, mobile_number, vehicle_number, parking_duration}

vehicle_id -> {id, amount, payment_time, payment_method, user_name, mobile_number, vehicle_number, parking_duration}

parking_analytics:

id -> {date, total_vehicles, total_revenue, average_parking_duration, occupancy_rate}

date -> {id, total_vehicles, total_revenue, average_parking_duration, occupancy_rate}

support:

id -> {user_id, subject, description, status, created_at, updated_at}

user_id -> {id, subject, description, status, created_at, updated_at}

staff:

id -> {name, contact_number, email, position, shift_start, shift_end}

vehicles:

id -> {name, mobile, entry_time, exit_time, is_exit, vehicle_no, vehicle_type, created_at, updated_at}

slots:

id -> {vehicle_id, space_for, is_empty}

admin:

id -> {username, password, created_at}

Implementation of Concurrency Control and Recovery Mechanisms:

COMMIT:

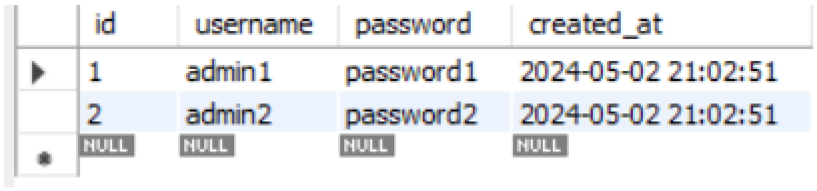
1) Begin a transaction and insert data into the Admin table:

INSERT INTO admin (username, password) VALUES

('admin1', 'password1'),

('admin2', 'password2');

OUTPUT:



id	username	password	created_at
1	admin1	password1	2024-05-02 21:02:51
2	admin2	password2	2024-05-02 21:02:51
NULL	NULL	NULL	NULL

This SQL script is inserting a new record into the Admin table. It adds a new admin with the username 'admin1', password 'Password1'. Finally, it commits the transaction to make the changes permanent.

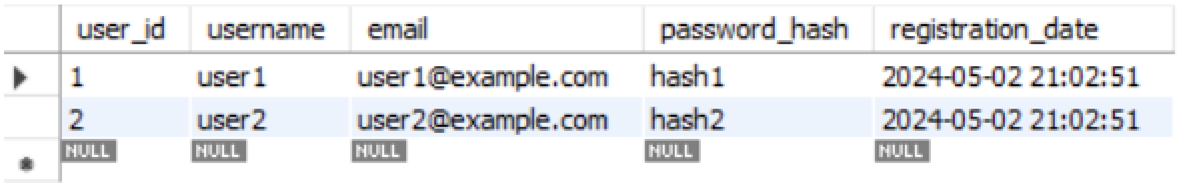
2) Begin a transaction and update data in fines table:

INSERT INTO users (username, email, password_hash) VALUES

('user1', 'user1@example.com', 'hash1'),

('user2', 'user2@example.com', 'hash2');

Output



user_id	username	email	password_hash	registration_date
1	user1	user1@example.com	hash1	2024-05-02 21:02:51
2	user2	user2@example.com	hash2	2024-05-02 21:02:51
NULL	NULL	NULL	NULL	NULL

This SQL script begins a transaction, inserts a record into the `user` table, specifying details such as the user ID, user name, email, and password_hash. It then commits the transaction to make the changes permanent.

InnoDB Tables:

->InnoDB is a storage engine that supports transactions and row-level locking. to perform transactions on these InnoDB tables, you can use SQL statements wrapped within the START TRANSACTION, COMMIT, and ROLLBACK statements. Here's an example of how you can perform transactions:

START TRANSACTION;

INSERT INTO users (username, email, password_hash) VALUES ('JohnDoe', 'john@example.com', 'password123');

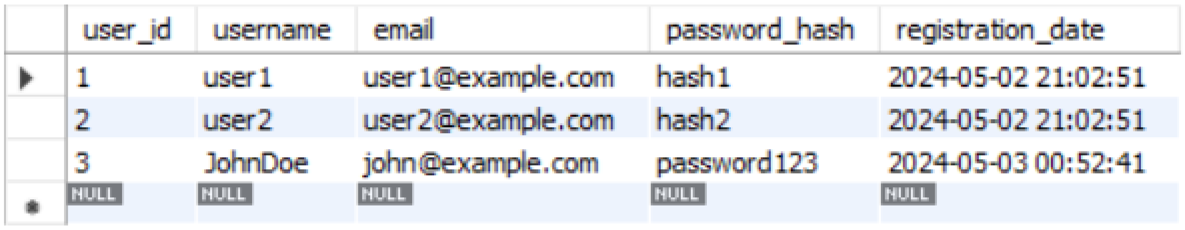
INSERT INTO payments (vehicle_id, amount, payment_method) VALUES (1, 50.00, 'Credit Card');

UPDATE vehicles SET is_exit = 'Yes' WHERE id = 1;

COMMIT;

In the above example, we started a transaction using START TRANSACTION. Then, we performed some example transactions such as inserting data into the users and payments tables, and updating data in the vehicles table. Finally, if all transactions are successful, we commit the changes using COMMIT.

Output;



user_id	username	email	password_hash	registration_date
1	user1	user1@example.com	hash1	2024-05-02 21:02:51
2	user2	user2@example.com	hash2	2024-05-02 21:02:51
3	JohnDoe	john@example.com	password123	2024-05-03 00:52:41
NULL	NULL	NULL	NULL	NULL

ROLLBACK

If any error occurs during the transactions or if you want to discard all changes made within the transaction block, you can use ROLLBACK:

START TRANSACTION;

INSERT INTO users (username, email, password_hash) VALUES ('JaneDoe', 'jane@example.com', 'password456');

INSERT INTO payments (vehicle_id, amount, payment_method) VALUES (2, 75.00, 'PayPal');

ROLLBACK;

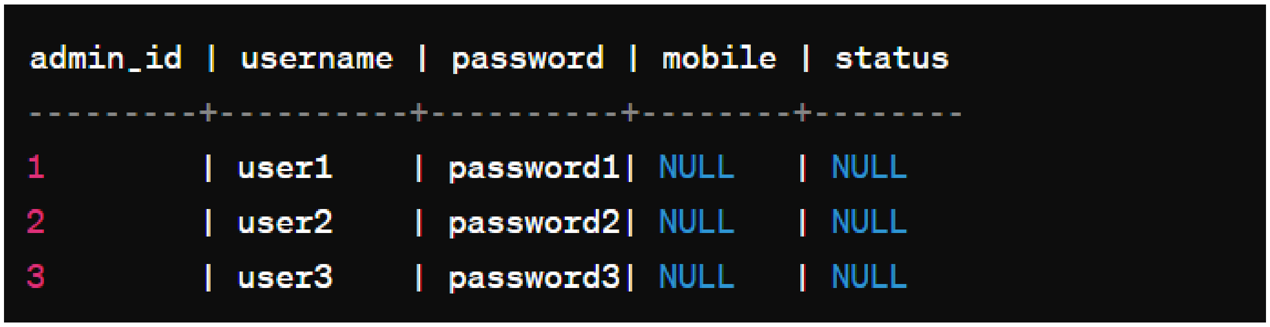
Using transactions ensures that all operations are either completed successfully or completely rolled back in case of errors or failures, maintaining data integrity.

BEGIN TRANSACTION; -- Start a transaction

UPDATE login SET status = 'active' WHERE username = 'user1'; -- Update the status of 'user1'

Simulate an error or system crash before committing

ROLLBACK; -- Rollback the transaction, reverting any changes made within it SELECT * FROM login; -- Check the current state of the 'login' table



```
admin_id | username | password | mobile | status
---------+----------+----------+--------+--------
1        | user1    | password1| NULL   | NULL
2        | user2    | password2| NULL   | NULL
3        | user3    | password3| NULL   | NULL
```

Query: Attempt to update the delivery person's payment, but before committing, simulate an error causing a rollback.

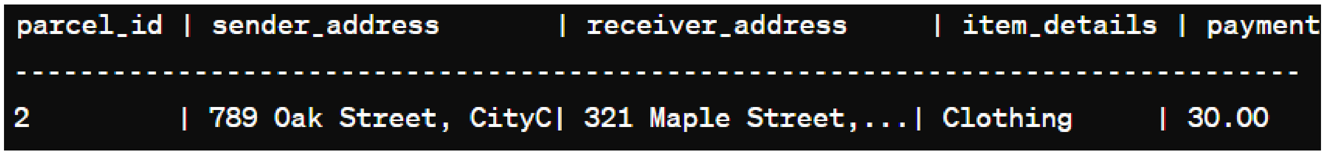
BEGIN TRANSACTION;

UPDATE Delivery_person SET payment = 40.00 WHERE parcel_id = 2;

-- Simulate an error or system crash before committing

ROLLBACK;

SELECT * FROM Delivery_person WHERE parcel_id = 2;



```
parcel_id | sender_address       | receiver_address     | item_details | payment
-------------------------------------------------------------------------------
2         | 789 Oak Street, CityC| 321 Maple Street,...| Clothing     | 30.00
```

Query: Start a transaction to update the status of an admin and then select the updated status.

Atomicity, Consistency, Isolation, Durability (ACID PROPERTIES)

1. Atomicity:

- Atomicity ensures that each transaction is treated as a single, indivisible unit of work.
- In our database, when inserting data into multiple tables, such as Organizer, Attendee, Event, etc., either all the inserts are successful, or none of them are. This ensures atomicity.
- For example, if an error occurs during the insertion of data into one table, the entire transaction will be rolled back, ensuring that no partial changes are made.

2. Consistency:

- Consistency ensures that the database remains in a consistent state before and after the transaction.
- In our database, constraints such as primary key constraints, foreign key constraints, and data types are enforced to maintain data integrity.
- For example, the foreign key constraint between tables ensures that an Event cannot exist without a valid OrganizerID, maintaining consistency between related tables.

3. Isolation:

- Isolation ensures that transactions are executed independently of each other, without interference.
- In your database transactions are executed in isolation from each other, preventing concurrent transactions from affecting each other's results.
- For example, if two transactions attempt to update the same record simultaneously, the database management system ensures that they are executed sequentially, avoiding conflicts.

4. Durability:

- Durability ensures that once a transaction is committed, its changes persist even in the event of system failure. In your database:
- In our database, the database management system ensures that committed transactions are permanently saved to disk.
- For example, if a transaction successfully inserts data into the database and then the system crashes, upon recovery, the changes made by the committed transaction will still be intact.

Our database adheres to the ACID properties by ensuring that transactions are atomic, maintain consistency, execute in isolation, and guarantee durability of committed transactions.

Code:

1. Install Window:

```
from PyQt5.QtWidgets import QWidget,QPushButton,QVBoxLayout,QLabel,QLineEdit
from LoginWindow import LoginScreen
import json
from DataBaseOperation import DBOperation

class InstallWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Install Vehical Parking System")
        self.resize(400,200)

        layout=QVBoxLayout()

        label_db_name=QLabel("Database Name : ")
        label_db_name.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_db_username=QLabel("Database Username : ")
        label_db_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_db_password=QLabel("Database Password : ")
        label_db_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_admin_username=QLabel("Admin Username : ")
        label_admin_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_admin_password=QLabel("Admin Password : ")
        label_admin_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_no_of_two_seater=QLabel("No of Two Wheeler Space : ")
        label_no_of_two_seater.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        label_no_of_four_seater=QLabel("No. of Four Wheeler Space : ")
        label_no_of_four_seater.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")


        self.input_db_name=QLineEdit()
        self.input_db_name.setText("vehicle_parking")
        self.input_db_name.setStyleSheet("padding:5px;font-size:17px")

        self.input_db_username=QLineEdit()
        self.input_db_username.setText("vehicle")
        self.input_db_username.setStyleSheet("padding:5px;font-size:17px")

        self.input_db_password=QLineEdit()
        self.input_db_password.setText("vehicle_password")
        self.input_db_password.setStyleSheet("padding:5px;font-size:17px")

        self.input_admin_username=QLineEdit()
        self.input_admin_username.setStyleSheet("padding:5px;font-size:17px")
        self.input_admin_password=QLineEdit()
        self.input_admin_password.setStyleSheet("padding:5px;font-size:17px")
        self.input_two_wheeler=QLineEdit()
        self.input_two_wheeler.setStyleSheet("padding:5px;font-size:17px")
        self.input_four_wheeler=QLineEdit()
        self.input_four_wheeler.setStyleSheet("padding:5px;font-size:17px")

        buttonsave=QPushButton("save config")
        buttonsave.setStyleSheet("padding:5px;font-size:17px;background:green;color:#fff")

        self.error_label=QLabel()
        self.error_label.setStyleSheet("color:red")

        layout.addWidget(label_db_name)
        layout.addWidget(self.input_db_name)
        layout.addWidget(label_db_username)
        layout.addWidget(self.input_db_username)
        layout.addWidget(label_db_password)
        layout.addWidget(self.input_db_password)
        layout.addWidget(label_admin_username)
        layout.addWidget(self.input_admin_username)
        layout.addWidget(label_admin_password)
        layout.addWidget(self.input_admin_password)
        layout.addWidget(label_no_of_two_seater)
        layout.addWidget(self.input_two_wheeler)
        layout.addWidget(label_no_of_four_seater)
        layout.addWidget(self.input_four_wheeler)
        layout.addWidget(buttonsave)
        layout.addWidget(self.error_label)

        buttonsave.clicked.connect(self.showStepInfo)

        self.setLayout(layout)

    def showStepInfo(self):
        if self.input_db_name.text()=="":
            self.error_label.setText("Please Enter DB Name")
            return

        if self.input_db_username.text()=="":
            self.error_label.setText("Please Enter DB Username")
            return

        if self.input_db_password.text()=="":
            self.error_label.setText("Please Enter DB Password")
            return

        if self.input_admin_username.text()=="":
            self.error_label.setText("Please Enter Admin Username")
            return

        if self.input_admin_password.text()=="":
            self.error_label.setText("Please Enter Admin Password")
            return

        if self.input_two_wheeler.text()=="":
            self.error_label.setText("Please Enter Two Wheeler Space")
            return

        if self.input_four_wheeler.text()=="":
            self.error_label.setText("Please Enter Four Wheeler Space")
            return



data={"username":self.input_db_username.text(),"database":self.input_db_name.text(),"password":self.input
_db_password.text()}
        file=open("./config.json","w")
        file.write(json.dumps(data))
        file.close()
        dbOperation=DBOperation()
        dbOperation.CreateTables()
        dbOperation.InsertAdmin(self.input_admin_username.text(),self.input_admin_password.text())
        dbOperation.InsertOneTimeData(int(self.input_two_wheeler.text()),int(self.input_four_wheeler.text()))

        self.close()
        self.login=LoginScreen()
        self.login.showLoginScreen()
        print("Save")
```

2. Login Window:

```
from PyQt5.QtWidgets import QWidget,QVBoxLayout,QPushButton,QLabel,QLineEdit,QApplication
import sys
from DataBaseOperation import DBOperation
from HomeWindow import HomeScreen

class LoginScreen(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Admin Login")
        self.resize(300,100)
        layout=QVBoxLayout()

        label_username=QLabel("Username : ")
        label_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        self.input_username=QLineEdit()
        self.input_username.setStyleSheet("padding:5px;font-size:17px")
        label_password=QLabel("Password : ")
        label_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
        self.error_msg=QLabel()
        self.error_msg.setStyleSheet("color:red;padding:8px 0px;font-size:18px;text-align:center")
        self.input_password=QLineEdit()
        self.input_password.setStyleSheet("padding:5px;font-size:17px")

        btn_login=QPushButton("Login")
        btn_login.setStyleSheet("padding:5px;font-size:20px;background:green;color:#fff")
        layout.addWidget(label_username)
        layout.addWidget(self.input_username)
        layout.addWidget(label_password)
        layout.addWidget(self.input_password)
        layout.addWidget(btn_login)
        layout.addWidget(self.error_msg)
        layout.addStretch()
        btn_login.clicked.connect(self.showHome)
        self.setLayout(layout)

    def showLoginScreen(self):
        self.show()

    def showHome(self):
        if self.input_username.text()=="":
            self.error_msg.setText("Please Enter Username")
            return

        if self.input_password.text()=="":
            self.error_msg.setText("Please Enter Password")
            return
        dboperation=DBOperation()
        result=dboperation.doAdminLogin(self.input_username.text(),self.input_password.text())
        if result:
            self.error_msg.setText("Login Successful")
            self.close()
            self.home = HomeScreen()
            self.home.show()
        else:
            self.error_msg.setText("Invalid Login Details")
```

3. Home Window:

```
from PyQt5.QtWidgets import
QWidget,QMainWindow,QPushButton,QLineEdit,QLabel,QVBoxLayout,QHBoxLayout,QFrame,QGridLa
yout,QComboBox,QTableWidget,QTableWidgetItem
from DataBaseOperation import DBOperation
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtWidgets import QHeaderView,qApp
import PyQt5.QtGui
from collections import defaultdict
from datetime import datetime
import matplotlib.pyplot as plt
from PyQt5.QtWidgets import QDialog
import mysql.connector
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QDialog
from PyQt5.QtWidgets import QLineEdit
from PyQt5.QtWidgets import QInputDialog
import uuid




class AdminPasswordDialog(QDialog):
    def __init__(self):
        super().__init__()

        # Create widgets for the dialog
        self.password_label = QLabel("Enter Admin Password:")
        self.password_input = QLineEdit()
        self.submit_button = QPushButton("Submit")
        self.submit_button.clicked.connect(self.accept)

        # Create layout for the dialog
        layout = QVBoxLayout()
        layout.addWidget(self.password_label)
        layout.addWidget(self.password_input)
        layout.addWidget(self.submit_button)

        self.setLayout(layout)

    def getPassword(self):
        return self.password_input.text()

class HomeScreen(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Home")
        self.dbOperation=DBOperation()
        self.setGeometry(100, 100, 600, 600)
        widget=QWidget()
        widget.setStyleSheet("background-color: #D3D3D3;")
        layout_horizontal=QHBoxLayout()
        menu_vertical_layout=QVBoxLayout()
        self.user_name_input = QLineEdit()
        self.mobile_number_input = QLineEdit()
        self.vehicle_number_input = QLineEdit()
        self.payment_method_dropdown = QComboBox()


        self.vtype = QComboBox()
        self.btn_home=QPushButton("Home")
        self.btn_add = QPushButton("Add Vehicle")
        self.btn_manage = QPushButton("Manage Vehicle")
        self.btn_history = QPushButton("History")
        self.btn_payments = QPushButton("Payments")
        self.btn_analytics = QPushButton("Analytics")
        self.btn_support = QPushButton("Tickets")
        self.btn_users = QPushButton("Users")
        self.btn_staff = QPushButton("staff")

        menu_vertical_layout.setContentsMargins(0,0,0,0)
        menu_vertical_layout.setSpacing(0)
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_home.clicked.connect(self.showHome)
        self.btn_add.clicked.connect(self.showAdd)
        self.btn_manage.clicked.connect(self.showManage)
        self.btn_history.clicked.connect(self.showHistory)
        self.btn_payments.clicked.connect(self.showPayments)
        self.btn_analytics.clicked.connect(self.showAnalytics)
        self.btn_support.clicked.connect(self.showSupport)
        self.btn_users.clicked.connect(self.showUsers)
        self.btn_staff.clicked.connect(self.showStaff)


        menu_frame=QFrame()
        menu_vertical_layout.addWidget(self.btn_home)
        menu_vertical_layout.addWidget(self.btn_add)
        menu_vertical_layout.addWidget(self.btn_manage)
        menu_vertical_layout.addWidget(self.btn_history)
        menu_vertical_layout.addWidget(self.btn_payments)
        menu_vertical_layout.addWidget(self.btn_analytics)
        menu_vertical_layout.addWidget(self.btn_support)
        menu_vertical_layout.addWidget(self.btn_users)
        menu_vertical_layout.addWidget(self.btn_staff)
        menu_vertical_layout.addStretch()
        menu_frame.setLayout(menu_vertical_layout)
        #menu_frame.setMinimumWidth(200)
        #menu_frame.setMaximumHeight(200)
```

```
parent_vertical=QVBoxLayout()
parent_vertical.setContentsMargins(0,0,0,0)
self.vertical_1=QVBoxLayout()
self.addHomePageData()


self.vertical_2=QVBoxLayout()
self.vertical_2.setContentsMargins(0,0,0,0)
self.addAddStudentPage()

self.vertical_3=QVBoxLayout()
self.vertical_3.setContentsMargins(0,0,0,0)
self.addManagePage()

self.vertical_4=QVBoxLayout()
self.vertical_4.setContentsMargins(0,0,0,0)
self.addHistoryPage()

self.vertical_5=QVBoxLayout()
self.vertical_5.setContentsMargins(0, 0, 0, 0)
self.addPaymentsPage()

self.vertical_6=QVBoxLayout()
self.vertical_6.setContentsMargins(0,0,0,0)
self.addAnalyticsPage()

self.vertical_7=QVBoxLayout()
self.vertical_7.setContentsMargins(0,0,0,0)
self.addSupportPage()

self.vertical_8=QVBoxLayout()
self.vertical_8.setContentsMargins(0,0,0,0)
self.addUsersPage()

self.vertical_8.addWidget(self.users_table)
self.vertical_8.setContentsMargins(0, 0, 0, 0)  # Set a fixed size for the layout

# Populate the users table
self.populateUsersTable()

self.vertical_9 = QVBoxLayout()
self.addStaffPage()
self.vertical_9.setContentsMargins(0, 0, 0, 0)  # Set a fixed size for the layout

self.frame_1=QFrame()
self.frame_1.setMinimumWidth(self.width())
self.frame_1.setMaximumWidth(self.width())
self.frame_1.setMaximumHeight(self.width())
self.frame_1.setMaximumHeight(self.width())
self.frame_1.setLayout(self.vertical_1)
self.frame_2=QFrame()
self.frame_2.setLayout(self.vertical_2)
self.frame_3=QFrame()
self.frame_3.setLayout(self.vertical_3)
self.frame_4=QFrame()
self.frame_4.setLayout(self.vertical_4)
self.frame_5=QFrame()
self.frame_5.setLayout(self.vertical_5)
self.frame_6=QFrame()
self.frame_6.setLayout(self.vertical_6)
self.frame_7=QFrame()
self.frame_7.setLayout(self.vertical_7)
self.frame_8=QFrame()
self.frame_8.setLayout(self.vertical_8)
self.frame_9=QFrame()
self.frame_9.setLayout(self.vertical_9)



parent_vertical.addWidget(self.frame_1)
parent_vertical.addWidget(self.frame_2)
parent_vertical.addWidget(self.frame_3)
parent_vertical.addWidget(self.frame_4)
parent_vertical.addWidget(self.frame_5)
parent_vertical.addWidget(self.frame_6)
parent_vertical.addWidget(self.frame_7)
parent_vertical.addWidget(self.frame_8)
parent_vertical.addWidget(self.frame_9)



layout_horizontal.addWidget(menu_frame)
layout_horizontal.addLayout(parent_vertical)
layout_horizontal.setContentsMargins(0,0,0,0)
parent_vertical.setContentsMargins(0,0,0,0)
parent_vertical.addStretch()
#menu_vertical_layout.addStretch()
layout_horizontal.addStretch()
widget.setLayout(layout_horizontal)

self.frame_1.show()
self.frame_2.hide()
self.frame_3.hide()
self.frame_4.hide()
self.frame_5.hide()
self.frame_6.hide()
self.frame_7.hide()
self.frame_8.hide()
self.frame_9.hide()

self.setCentralWidget(widget)
```

```
    def showStaff(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.show()


    def showUsers(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_9.hide()
        self.frame_8.show()


    def showSupport(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_7.show()


    def showAnalytics(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_6.show()


    def showPayments(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_5.show()

    def showHistory(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_4.show()

    def showManage(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_2.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_3.show()

    def showAdd(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_1.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_2.show()

    def showHome(self):
        self.btn_home.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_payments.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_analytics.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_support.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
```

```
        self.btn_users.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_staff.setStyleSheet("width:200px;height:80px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.frame_2.hide()
        self.frame_3.hide()
        self.frame_4.hide()
        self.frame_5.hide()
        self.frame_6.hide()
        self.frame_7.hide()
        self.frame_8.hide()
        self.frame_9.hide()
        self.frame_1.show()

    def refreshHome(self):
        while self.gridLayout.count():
            child=self.gridLayout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()
        row=0
        i=0
        alldata=self.dbOperation.getSlotSpace()
        for data in alldata:
            label=QPushButton("Slot "+str(data[0])+" \n "+str(data[1]))

            if data[3]==1:
                label.setStyleSheet("background-
color:green;color:white;padding:5px;width:100px;height:100px;border:1px solid white;text-
align:center;font-weight:bold")
            else:
                label.setStyleSheet("background-
color:red;color:white;padding:5px;width:100px;height:100px;border:1px solid white;text-align:center;font-
weight:bold")

            if i%5==0:
                i=0
                row=row+1

            self.gridLayout.addWidget(label,row,i)
            i=i+1


    def addHomePageData(self):
        self.vertical_1.setContentsMargins(0,0,0,0)

        button=QPushButton("Refresh Slot")
        button.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px;background:#696969;border:1px solid
white")
        button.clicked.connect(self.refreshHome)

        vertical_layout=QVBoxLayout()
        vertical_layout.setContentsMargins(0,0,0,0)
        frame=QFrame()

        horizontal=QHBoxLayout()
        horizontal.setContentsMargins(0,0,0,0)
        vertical_layout.addLayout(horizontal)

        alldata=self.dbOperation.getSlotSpace()
        self.gridLayout=QGridLayout()
        self.gridLayout.setContentsMargins(0,0,0,0)
        self.gridLayout.setHorizontalSpacing(0)
        self.gridLayout.setVerticalSpacing(0)
        vertical_layout.addWidget(button)
        vertical_layout.addLayout(self.gridLayout)

        row=0
        i=0
        for data in alldata:
            label=QPushButton("Slot "+str(data[0])+" \n "+str(data[1]))

            if data[3]==1:
                label.setStyleSheet("background-
color:green;color:white;padding:5px;width:100px;height:100px;border:1px solid white;text-
align:center;font-weight:bold")
            else:
                label.setStyleSheet("background-
color:red;color:white;padding:5px;width:100px;height:100px;border:1px solid white;text-align:center;font-
weight:bold")

            if i%5==0:
                i=0
                row=row+1

            self.gridLayout.addWidget(label,row,i)
            i=i+1

        frame.setLayout(vertical_layout)
        self.vertical_1.addWidget(frame)
        self.vertical_1.addStretch()

    def addAddStudentPage(self):
        layout=QVBoxLayout()
        frame=QFrame()


        name_label=QLabel("Name : ")
        name_label.setStyleSheet("color:black; background-color:white; padding:8px 0px;font-
size:30px,border:5px black")
        name_label.setMinimumHeight(40)
        mobile_label=QLabel("Mobile : ")
        mobile_label.setStyleSheet("color:black;background-color:white;padding:8px 0px;font-
size:30px.border:5px black")
```

```
        mobile_label.setMinimumHeight(40)
        vechicle_label=QLabel("Vehicle No : ")
        vechicle_label.setStyleSheet("color:black;background-color:white;padding:8px 0px;font-
size:30px,border:5px black")
        vechicle_label.setMinimumHeight(40)
        vechicle_type=QLabel("Vehicle Type : ")
        vechicle_type.setStyleSheet("color:black;background-color:white; padding:8px 0px;font-
size:30px,border:5px black")
        vechicle_type.setMinimumHeight(40)
        error_label=QLabel("")
        error_label.setStyleSheet("color:black;background-color:white;padding:8px 0px;font-
size:30px,border:5px black")
        error_label.setMinimumHeight(40)

        name_input=QLineEdit()
        name_input.setStyleSheet("color:black;background-color: white;")
        name_input.setMinimumHeight(60)
        mobile_input=QLineEdit()
        mobile_input.setStyleSheet("color:black;background-color: white;")
        mobile_input.setMinimumHeight(60)
        vehicle_input=QLineEdit()
        vehicle_input.setStyleSheet("color:black;background-color: white;")
        vehicle_input.setMinimumHeight(60)
        vtype=QComboBox()
        vtype.setMinimumHeight(40)
        vtype.setStyleSheet("color:black;background-color: white;")
        vtype.addItem("2 Wheeler")
        vtype.addItem("4 Wheeler")
        vtype.setMinimumHeight(40)

        button=QPushButton("Add Vehicle")
        button.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px;background:green;border:1px solid
white")

        layout.addWidget(name_label)
        layout.addWidget(name_input)
        layout.addWidget(mobile_label)
        layout.addWidget(mobile_input)
        layout.addWidget(vechicle_label)
        layout.addWidget(vehicle_input)
        layout.addWidget(vechicle_type)
        layout.addWidget(vtype)
        layout.addWidget(button)
        layout.addWidget(error_label)

        layout.setContentsMargins(0,0,0,0)
        frame.setMinimumHeight(self.height())
        frame.setMinimumWidth(self.width())
        frame.setMaximumHeight(self.width())
        frame.setMaximumWidth(self.width())

        layout.addStretch()
        frame.setLayout(layout)

button.clicked.connect(lambda:self.addVehicles(name_input.text(),vehicle_input.text(),mobile_input.text(),v
type.currentIndex(),error_label))
        self.vertical_2.addWidget(frame)

    def addVehicles(self,name,vehicleno,mobile,index,error_label):
        vtp=2
        if index==0:
            vtp=2
        else:
            vtp=4


        data=self.dbOperation.AddVehicles(name,vehicleno,mobile,str(vtp))
        if data==True:
            error_label.setText("Added Successfully")
        elif data==False:
            error_label.setText("Failed to Add Vehicle")
        else:
            error_label.setText(str(data))



    def addManagePage(self):
        data=self.dbOperation.getCurrentVehicle()
        self.table=QTableWidget()
        self.table.setStyleSheet("background:#fff")
        self.table.resize(self.width(),self.height())
        self.table.setRowCount(len(data))
        self.table.setColumnCount(7)

        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)
        self.table.setHorizontalHeaderItem(0,QTableWidgetItem("ID"))
        self.table.setHorizontalHeaderItem(1,QTableWidgetItem("Name"))
        self.table.setHorizontalHeaderItem(2,QTableWidgetItem("VEHICLE No"))
        self.table.setHorizontalHeaderItem(3,QTableWidgetItem("MOBILE"))
        self.table.setHorizontalHeaderItem(4,QTableWidgetItem("VEHICLE TYPE"))
        self.table.setHorizontalHeaderItem(5,QTableWidgetItem("ENTRY TIME"))
        self.table.setHorizontalHeaderItem(6,QTableWidgetItem("ACTION"))

        loop=0
        for smalldata in data:
            self.table.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
            self.table.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
            self.table.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
            self.table.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
            self.table.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
            self.table.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
            self.button_exit=QPushButton("Exit")
            self.button_exit.setStyleSheet("color:#fff;padding:8px 0px;font-
size:20px;background:green;border:1px solid white")
```

```
            self.table.setCellWidget(loop,6,self.button_exit)
            self.button_exit.clicked.connect(self.exitCall)
            loop=loop+1

        frame=QFrame()
        layout=QVBoxLayout()
        button=QPushButton("Refresh")
        button.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px;background:green;border:1px solid
white")
        button.clicked.connect(self.refreshManage)
        layout.setContentsMargins(0,0,0,0)
        layout.setSpacing(0)
        layout.addWidget(button)
        layout.addWidget(self.table)
        frame.setLayout(layout)
        frame.setContentsMargins(0,0,0,0)
        frame.setMaximumWidth(self.width())
        frame.setMinimumWidth(self.width())
        frame.setMaximumHeight(self.height())
        frame.setMinimumHeight(self.height())
        self.vertical_3.addWidget(frame)
        self.vertical_3.addStretch()

    def refreshManage(self):
        data=self.dbOperation.getCurrentVehicle()
        self.table.setRowCount(len(data))
        self.table.setColumnCount(7)
        loop=0
        for smalldata in data:
            self.table.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
            self.table.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
            self.table.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
            self.table.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
            self.table.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
            self.table.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
            self.button_exit=QPushButton("Exit")
            self.table.setCellWidget(loop,6,self.button_exit)
            self.button_exit.clicked.connect(self.exitCall)
            loop=loop+1


    def refreshHistory(self):
        self.table1.clearContents()
        data=self.dbOperation.getAllVehicle()
        loop=0
        self.table1.setRowCount(len(data))
        self.table1.setColumnCount(7)
        for smalldata in data:
            self.table1.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
            self.table1.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
            self.table1.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
            self.table1.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
            self.table1.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
            self.table1.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
            self.table1.setItem(loop,6,QTableWidgetItem(str(smalldata[4])))
            loop=loop+1

    def addHistoryPage(self):
        data=self.dbOperation.getAllVehicle()
        self.table1=QTableWidget()
        self.table1.resize(self.width(),self.height())
        self.table1.setRowCount(len(data))
        self.table1.setStyleSheet("background:#fff")
        self.table1.setColumnCount(7)

        button=QPushButton("Refresh")
        button.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px;background:green;border:1px solid
white")
        button.clicked.connect(self.refreshHistory)


        self.table1.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)
        self.table1.setHorizontalHeaderItem(0,QTableWidgetItem("ID"))
        self.table1.setHorizontalHeaderItem(1,QTableWidgetItem("Name"))
        self.table1.setHorizontalHeaderItem(2,QTableWidgetItem("VEHICLE No"))
        self.table1.setHorizontalHeaderItem(3,QTableWidgetItem("MOBILE"))
        self.table1.setHorizontalHeaderItem(4,QTableWidgetItem("VEHICLE TYPE"))
        self.table1.setHorizontalHeaderItem(5,QTableWidgetItem("ENTRY TIME"))
        self.table1.setHorizontalHeaderItem(6,QTableWidgetItem("EXIT TIME"))

        loop=0
        for smalldata in data:
            self.table1.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
            self.table1.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
            self.table1.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
            self.table1.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
            self.table1.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
            self.table1.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
            self.table1.setItem(loop,6,QTableWidgetItem(str(smalldata[4])))
            loop=loop+1

        self.frame5=QFrame()
        self.layout1=QVBoxLayout()
        self.layout1.setContentsMargins(0,0,0,0)
        self.layout1.setSpacing(0)
        self.layout1.addWidget(button)
        self.layout1.addWidget(self.table1)
        self.frame5.setLayout(self.layout1)
        self.frame5.setContentsMargins(0,0,0,0)
        self.frame5.setMaximumWidth(self.width())
        self.frame5.setMinimumWidth(self.width())
        self.frame5.setMaximumHeight(self.height())
        self.frame5.setMinimumHeight(self.height())
        self.vertical_4.addWidget(self.frame5)
        self.vertical_4.addStretch()


    def addPaymentsPage(self):
    # Clear any existing widgets in vertical layout
        while self.vertical_5.count():
            child = self.vertical_5.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        label_stylesheet = "color: black; font-size: 20px;"

    # Fetch vehicle IDs from the database
        vehicle_ids = self.dbOperation.getCurrentVehicle()

    # Create a label and dropdown menu for selecting vehicle ID
        vehicle_id_label = QLabel("Select Vehicle ID:")
        vehicle_id_label.setStyleSheet("color: black; font-size: 30")
        self.vehicle_id_dropdown = QComboBox()
        for vehicle in vehicle_ids:
            self.vehicle_id_dropdown.addItem(str(vehicle[0]))

        self.vehicle_id_dropdown.setStyleSheet("color: black; background-color: white;font-size: 30;")
        self.vehicle_id_dropdown.setMinimumHeight(40)

        self.refreshData()

    # Create labels and input fields for other payment details

        payment_method_label = QLabel("Payment Method:")
        payment_method_label.setStyleSheet("color: black; font-size: 30")
        self.payment_method_dropdown = QComboBox()
        self.payment_method_dropdown.addItems(["UPI", "Cash", "Credit/Debit Card"])
        self.payment_method_dropdown.setStyleSheet("color: black; background-color: white;font-size: 30;")
        payment_method_label.setMinimumHeight(40)
        self.payment_method_dropdown.setMinimumHeight(40)

        user_name_label = QLabel("User Name:")
        user_name_label.setStyleSheet("color: black; font-size: 30")
        self.user_name_input = QLineEdit()
        user_name_label.setMinimumHeight(40)
        self.user_name_input.setMinimumHeight(40)

        mobile_number_label = QLabel("Mobile Number:")
        mobile_number_label.setStyleSheet("color: black; font-size: 30")
        self.mobile_number_input = QLineEdit()
        mobile_number_label.setMinimumHeight(40)
        self.mobile_number_input.setMinimumHeight(40)

        vehicle_number_label = QLabel("Vehicle Number:")
        vehicle_number_label.setStyleSheet("color: black; font-size: 30")
        self.vehicle_number_input = QLineEdit()
        vehicle_number_label.setMinimumHeight(40)
        self.vehicle_number_input.setMinimumHeight(40)

        amount_label = QLabel("Amount:")
        amount_label.setStyleSheet("color: black; font-size: 30")
        self.amount_input = QLineEdit()
        self.amount_input.setStyleSheet("color: black; background-color: white;")
        self.amount_input.setMinimumHeight(40)
        amount_label.setMinimumHeight(40)

    # Calculate parking duration
        selected_vehicle_id = int(self.vehicle_id_dropdown.currentText())
        entry_time_str = self.dbOperation.getEntryTime(selected_vehicle_id)
        entry_time = datetime.strptime(entry_time_str, "%Y-%m-%d %H:%M:%S")

        current_time = datetime.now()
        parking_duration = (current_time - entry_time).total_seconds() / 3600


    # Display parking duration
        parking_duration_label = QLabel("Parking Duration:")
        self.parking_duration_input = QLineEdit(str(round(parking_duration, 2)))
        self.parking_duration_input.setReadOnly(True)
        self.parking_duration_input.setStyleSheet("color: black; background-color: white;")
        self.parking_duration_input.setMinimumHeight(40)
        parking_duration_label.setMinimumHeight(40)


    # Create a submit button
        submit_button = QPushButton("SUBMIT")
        submit_button.clicked.connect(self.submitPayment)
        submit_button.setStyleSheet("color: white; background-color: green;")

    # Create a refresh button
        refresh_button = QPushButton("REFRESH")
        refresh_button.clicked.connect(self.refreshData)
        refresh_button.setStyleSheet("color: white; background-color: green;")

    # Set stylesheets for labels and input fields
        labels = [vehicle_id_label, amount_label, payment_method_label, user_name_label,
mobile_number_label, vehicle_number_label, parking_duration_label]
        inputs = [self.user_name_input, self.mobile_number_input, self.vehicle_number_input,
self.amount_input]

        for label in labels:
            label.setStyleSheet("color: white;")  # White text color
        for input_field in inputs:
            input_field.setStyleSheet("color: black; background-color: white;")  # Black text on white
background

    # Create a vertical layout to hold the widgets
        vertical_layout = QVBoxLayout()
    vertical_layout.addWidget(vehicle_id_label)
    vertical_layout.addWidget(self.vehicle_id_dropdown)
    vertical_layout.addWidget(amount_label)
    vertical_layout.addWidget(self.amount_input)
    vertical_layout.addWidget(payment_method_label)
    vertical_layout.addWidget(self.payment_method_dropdown)
    vertical_layout.addWidget(user_name_label)
    vertical_layout.addWidget(self.user_name_input)
    vertical_layout.addWidget(mobile_number_label)
    vertical_layout.addWidget(self.mobile_number_input)
    vertical_layout.addWidget(vehicle_number_label)
    vertical_layout.addWidget(self.vehicle_number_input)
    vertical_layout.addWidget(parking_duration_label)
    vertical_layout.addWidget(self.parking_duration_input)
    vertical_layout.addWidget(submit_button)
    vertical_layout.addWidget(refresh_button)

# Add the vertical layout to the main vertical layout
    self.vertical_5.addLayout(vertical_layout)
    self.vertical_5.addStretch()

    vehicle_id_label.setStyleSheet(label_stylesheet)
    payment_method_label.setStyleSheet(label_stylesheet)
    user_name_label.setStyleSheet(label_stylesheet)
    mobile_number_label.setStyleSheet(label_stylesheet)
    vehicle_number_label.setStyleSheet(label_stylesheet)
    amount_label.setStyleSheet(label_stylesheet)
    parking_duration_label.setStyleSheet(label_stylesheet)


def refreshData(self):
# Get the selected vehicle ID
    selected_vehicle_id = int(self.vehicle_id_dropdown.currentText())

# Fetch vehicle details from the database based on the selected vehicle ID
    vehicle_details = self.dbOperation.getCurrentVehicle()  # Remove the argument here
    if vehicle_details:
    # Find the details for the selected vehicle ID
        for vehicle in vehicle_details:
            if vehicle[0] == selected_vehicle_id:
            # Fill in the input fields with the fetched vehicle details
                self.user_name_input.setText(vehicle[1])  # Assuming name is at index 1 in the tuple
                self.mobile_number_input.setText(vehicle[2])  # Assuming mobile number is at index 2
                self.vehicle_number_input.setText(vehicle[6])  # Assuming vehicle number is at index 6
                break


def submitPayment(self):
# Retrieve input values from UI elements
    vehicle_id = int(self.vehicle_id_dropdown.currentText())
    payment_method = self.payment_method_dropdown.currentText()
    user_name = self.user_name_input.text()
```

```
        mobile_number = self.mobile_number_input.text()
        vehicle_number = self.vehicle_number_input.text()
        parking_duration = float(self.parking_duration_input.text())
        amount = self.amount_input.text()


        # Insert payment into database
        if self.dbOperation.addPayment(vehicle_id, amount, payment_method, user_name, mobile_number,
vehicle_number, parking_duration):
            QMessageBox.information(self, "Success", "Payment added successfully.")
        else:
            QMessageBox.warning(self, "Error", "Failed to add payment.")

    def addAnalyticsPage(self):
    # Clear any existing widgets in vertical layout
        while self.vertical_6.count():
            child = self.vertical_6.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        try:
            # Fetch analytics data from the database for the current date
            today_date = datetime.now().date().strftime("%Y-%m-%d")
            analytics_data = self.dbOperation.get_daily_analytics(today_date)

            if analytics_data:
                # Create labels to display analytics details
                total_vehicles_label = QLabel(f"Total Vehicles: {analytics_data['total_vehicles']}")
                total_vehicles_label.setStyleSheet("color: white;")  # Change text color to white
                total_vehicles_label.setMinimumHeight(30)

                total_revenue_label = QLabel(f"Total Revenue: ${analytics_data['total_revenue']}")
                total_revenue_label.setStyleSheet("color: white;")  # Change text color to white
                total_revenue_label.setMinimumHeight(30)

                average_duration_label = QLabel(f"Average Parking Duration:
{analytics_data['average_parking_duration']} hours")
                average_duration_label.setStyleSheet("color: white;")  # Change text color to white
                average_duration_label.setMinimumHeight(30)


                # Create a button to visualize analytics data
                visualize_button = QPushButton("Visualize Analytics")
                visualize_button.clicked.connect(self.visualizeAnalytics)
                visualize_button.setStyleSheet("background-color = white;")

                # Create a vertical layout to hold the widgets
                vertical_layout = QVBoxLayout()
                vertical_layout.addWidget(total_vehicles_label)
                vertical_layout.addWidget(total_revenue_label)
                vertical_layout.addWidget(average_duration_label)
                vertical_layout.addWidget(visualize_button)
                # Add the vertical layout to the main vertical layout
                self.vertical_layout.addLayout(vertical_layout)
                self.vertical_layout.addStretch()
            else:
                # Display a message if no analytics data is available
                no_data_label = QLabel("No analytics data available for the selected date.")
                self.vertical_6.addWidget(no_data_label)

        except mysql.connector.Error as error:
            print("Error:", error)


    def visualizeAnalytics(self):
        try:
            # Fetch analytics data from the database for the current date
            today_date = datetime.now().date().strftime("%Y-%m-%d")
            analytics_data = self.dbOperation.get_daily_analytics(today_date)

            if analytics_data:
                # Extract data for visualization
                labels = ['Total Vehicles', 'Total Revenue', 'Average Parking Duration']
                values = [analytics_data['total_vehicles'], analytics_data['total_revenue'],
analytics_data['average_parking_duration']]

                # Create a bar plot
                plt.bar(labels, values)
                plt.title('Daily Analytics')
                plt.xlabel('Metrics')
                plt.ylabel('Values')
                plt.show()
            else:
                # Display a message if no analytics data is available
                print("No analytics data available for visualization.")

        except mysql.connector.Error as error:
            print("Error:", error)



    def addSupportPage(self):
        # Clear any existing widgets in vertical layout
        while self.vertical_7.count():
            child = self.vertical_7.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        # Create labels and input fields for support ticket details
        user_id_label = QLabel("User ID:")
        user_id_label.setStyleSheet("color: black;")
        self.user_id_input = QLineEdit()
        self.user_id_input.setStyleSheet("color: black; background-color: white;")
```

```
user_id_label.setMinimumHeight(40)
self.user_id_input.setMinimumHeight(40)

subject_label = QLabel("Subject:")
subject_label.setStyleSheet("color: black;")
self.subject_input = QLineEdit()
self.subject_input.setStyleSheet("color: black; background-color: white;")
subject_label.setMinimumHeight(40)
self.subject_input.setMinimumHeight(40)

description_label = QLabel("Description:")
description_label.setStyleSheet("color: black;")
self.description_input = QLineEdit()
self.description_input.setStyleSheet("color: black; background-color: white;")
description_label.setMinimumHeight(40)
self.description_input.setMinimumHeight(40)

# Create a submit button
submit_button = QPushButton("Submit")
submit_button.setStyleSheet("color: white; background-color: green;")
submit_button.clicked.connect(self.submitSupportTicket)

# Create a table to display support tickets
self.support_table = QTableWidget()
self.support_table.setColumnCount(4)  # User ID, Subject, Description, Status
self.support_table.setHorizontalHeaderLabels(["User ID", "Subject", "Description", "Status"])
self.support_table.setStyleSheet("background-color: white;")


self.support_table.setMinimumWidth(550)  # Set minimum width
self.support_table.setMinimumHeight(450)  # Set minimum height

# Fetch support tickets and populate the table
self.populateSupportTable()

# Create a vertical layout to hold the widgets
vertical_layout = QVBoxLayout()
vertical_layout.addWidget(user_id_label)
vertical_layout.addWidget(self.user_id_input)
vertical_layout.addWidget(subject_label)
vertical_layout.addWidget(self.subject_input)
vertical_layout.addWidget(description_label)
vertical_layout.addWidget(self.description_input)
vertical_layout.addWidget(submit_button)
vertical_layout.addWidget(self.support_table)

# Set background color for the main layout
self.setStyleSheet("background-color: white;")

# Add the vertical layout to the main vertical layout
self.vertical_7.addLayout(vertical_layout)
self.vertical_7.addStretch()
```

```
def submitSupportTicket(self):
# Generate a unique ticket ID
    existing_tickets_count = len(self.dbOperation.getAllSupportTickets())
    ticket_id = str(existing_tickets_count + 1)

    # Get input values from the input fields
    user_id = self.user_id_input.text()
    subject = self.subject_input.text()
    description = self.description_input.text()
    status = "Open"  # Default status

    # Call the addSupportTicket method from DBOperation to add the support ticket to the database
    success = self.dbOperation.addSupportTicket(user_id, subject, description, status)

    if success:
        print("Support ticket added successfully!")
        # Update the table after adding the ticket
        self.populateSupportTable()
    else:
        print("Failed to add support ticket.")


def populateSupportTable(self):
# Fetch support tickets from the database
    support_tickets = self.dbOperation.getAllSupportTickets()

    # Clear existing rows in the table
    self.support_table.setRowCount(0)

    # Populate the table with support ticket data
    for row_num, ticket in enumerate(support_tickets):
        self.support_table.insertRow(row_num)
        for col_num, data in enumerate(ticket):
            item = QTableWidgetItem(str(data))
            # Set columns "User ID", "Subject", and "Description" non-editable
            if col_num in [0, 1, 2]:
                item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.support_table.setItem(row_num, col_num, item)

    # Connect slot for editing "Status" column
    self.support_table.itemDoubleClicked.connect(self.editSupportTicketStatus)

def editSupportTicketStatus(self, item):
    # Check if the double-clicked column is "Status"
    if item.column() == 3:  # Assuming "Status" column is at index 3
        # Create the admin password dialog
        dialog = AdminPasswordDialog()
        if dialog.exec_() == QDialog.Accepted:
            # Get the entered password
            admin_password = dialog.getPassword()
            # Check if the admin password is correct
            if self.dbOperation.checkAdminPassword(admin_password):
                # Prompt for new status
                new_status, ok = QInputDialog.getText(self, "Enter New Status", "New Status:")
                if ok:
                    # Get the row index
                    row_index = item.row()
                    # Get the ticket ID from the "User ID" column (assuming it's in the first column)
                    ticket_id = self.support_table.item(row_index, 0).text()
                    # Update the status in the database
                    success = self.dbOperation.changeSupportTicketStatus(ticket_id, new_status)
                    if success:
                        QMessageBox.information(self, "Success", "Support ticket status changed successfully.")
                        # Update the table after changing status
                        self.populateSupportTable()
                    else:
                        QMessageBox.warning(self, "Error", "Failed to change support ticket status.")
            else:
                QMessageBox.warning(self, "Access Denied", "Incorrect admin password.")
    def addUsersPage(self):
        # Clear any existing widgets in vertical layout
        while self.vertical_8.count():
            child = self.vertical_8.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

        # Create labels and input fields for user details
        username_label = QLabel("Username:")
        username_label.setStyleSheet("color: black;")
        self.username_input = QLineEdit()
        self.username_input.setStyleSheet("color: black; background-color: white;")
        username_label.setMinimumHeight(40)
        self.username_input.setMinimumHeight(40)

        email_label = QLabel("Email:")
        email_label.setStyleSheet("color: black;")
        self.email_input = QLineEdit()
        self.email_input.setStyleSheet("color: black; background-color: white;")
        email_label.setMinimumHeight(40)
        self.email_input.setMinimumHeight(40)

        password_label = QLabel("Password:")
        password_label.setStyleSheet("color: black;")
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)
        self.password_input.setStyleSheet("color:black; background-color: white;")
        password_label.setMinimumHeight(40)
        self.password_input.setMinimumHeight(40)
        # Create a button to add user
        add_user_button = QPushButton("Add User")
        add_user_button.clicked.connect(self.addUser)
        add_user_button.setStyleSheet("color:white; background-color: green;")
        add_user_button.setMinimumHeight(40)

        # Create a table to display existing users
        self.users_table = QTableWidget()
        self.users_table.setStyleSheet("color: black; background-color: white;")
        self.users_table.setColumnCount(4)
        self.users_table.setHorizontalHeaderLabels(["User ID", "Username", "Email", "Registration Date"])
        self.populateUsersTable()

        self.users_table.setMinimumWidth(550)  # Set minimum width
        self.users_table.setMinimumHeight(400)  # Set minimum height


        # Create a button to remove user
        remove_user_button = QPushButton("Remove User")
        remove_user_button.clicked.connect(self.removeUser)
        remove_user_button.setStyleSheet("background-color: red;")

        # Create a vertical layout to hold the widgets
        vertical_layout = QVBoxLayout()
        vertical_layout.addWidget(username_label)
        vertical_layout.addWidget(self.username_input)
        vertical_layout.addWidget(email_label)
        vertical_layout.addWidget(self.email_input)
        vertical_layout.addWidget(password_label)
        vertical_layout.addWidget(self.password_input)
        vertical_layout.addWidget(add_user_button)
        vertical_layout.addWidget(self.users_table)
        vertical_layout.addWidget(remove_user_button)

        self.setStyleSheet("background-color: white;")

        # Add the vertical layout to the main vertical layout
        self.vertical_8.addLayout(vertical_layout)
        self.vertical_8.addStretch()
    def setWidgetStyles(self, widget):
        widget.setStyleSheet("color: black; background-color: white; border: 5px solid grey; margin: 5px;")

    def populateUsersTable(self):
        # Fetch existing users from the database
        users = self.dbOperation.getAllUsers()

        # Clear existing rows in the table
        self.users_table.setRowCount(0)
        self.users_table.setStyleSheet("background-color: white;")

        # Get the current date
        current_date = datetime.now().strftime("%Y-%m-%d")

        # Populate the table with user data
        for row_num, user_data in enumerate(users):
            self.users_table.insertRow(row_num)
            for col_num, data in enumerate(user_data):
                # Display email in the "Email" column
                if col_num == 2:
                    item = QTableWidgetItem(str(data))
                    self.users_table.setItem(row_num, col_num, item)
                # Display registration date in the "Registration Date" column
                elif col_num == 3:
                    item = QTableWidgetItem(current_date)
                    self.users_table.setItem(row_num, col_num, item)
                # Hide password from the table
                elif col_num == 4:
                    continue
                else:
                    item = QTableWidgetItem(str(data))
                    self.users_table.setItem(row_num, col_num, item)

        # Resize the table horizontally and vertically
        table_width = 550  # Set the desired width in pixels
        table_height = 450  # Set the desired height in pixels



        # Resize columns to fit content
        self.users_table.resizeColumnsToContents()

        # Set email column width to a fixed size
        self.users_table.setColumnWidth(2, 250)  # Adjust the width as needed (200 pixels in this example)




    def addUser(self):
        # Get user details from input fields
        username = self.username_input.text()
        email = self.email_input.text()
        password = self.password_input.text()



        # Add user to the database
        if self.dbOperation.addUser(username, email, password):
            QMessageBox.information(self, "Success", "User added successfully.")
            self.populateUsersTable()
        else:
            QMessageBox.information(self, "error")
    def removeUser(self):
        # Get the selected row from the table
        selected_row = self.users_table.currentRow()
        if selected_row != -1:
            # Get the user_id of the selected user
            user_id = int(self.users_table.item(selected_row, 0).text())

            # Remove user from the database
            if self.dbOperation.removeUser(user_id):
                QMessageBox.information(self, "Success", "User removed successfully.")
            self.populateUsersTable()
        else:
            QMessageBox.information(self, "error", "User could not be removed.")
    else:
        QMessageBox.information(self, "error", "No user selected.")

def addStaffPage(self):
    # Clear any existing widgets in vertical layout
    while self.vertical_9.count():
        child = self.vertical_9.takeAt(0)
        if child.widget():
            child.widget().deleteLater()

    # Create labels and input fields for staff details
    name_label = QLabel("Name:")
    name_label.setStyleSheet("color: black;")
    self.name_input = QLineEdit()
    self.name_input.setStyleSheet("color: black; background-color: white;")
    name_label.setMinimumHeight(40)
    self.name_input.setMinimumHeight(40)

    contact_number_label = QLabel("Contact Number:")
    self.contact_number_input = QLineEdit()
    contact_number_label.setMinimumHeight(40)
    self.contact_number_input.setStyleSheet("color: black; background-color: white;")
    contact_number_label.setStyleSheet("color: black;")
    self.contact_number_input.setMinimumHeight(40)

    email_label = QLabel("Email:")
    self.email_input = QLineEdit()
    email_label.setMinimumHeight(40)
    self.email_input.setStyleSheet("color: black; background-color: white;")
    email_label.setStyleSheet("color: black;")
    self.email_input.setMinimumHeight(40)

    position_label = QLabel("Position:")
    self.position_input = QLineEdit()
    position_label.setMinimumHeight(40)
    self.position_input.setStyleSheet("color: black; background-color: white;")
    position_label.setStyleSheet("color: black;")
    self.position_input.setMinimumHeight(40)


    # Create a submit button
    submit_button = QPushButton("Add Staff")
    submit_button.clicked.connect(self.addStaff)
    submit_button.setStyleSheet("color: white; background-color: green;")
    submit_button.setMinimumHeight(40)

    # Create a vertical layout to hold the widgets
    vertical_layout = QVBoxLayout()
    vertical_layout.addWidget(name_label)
```

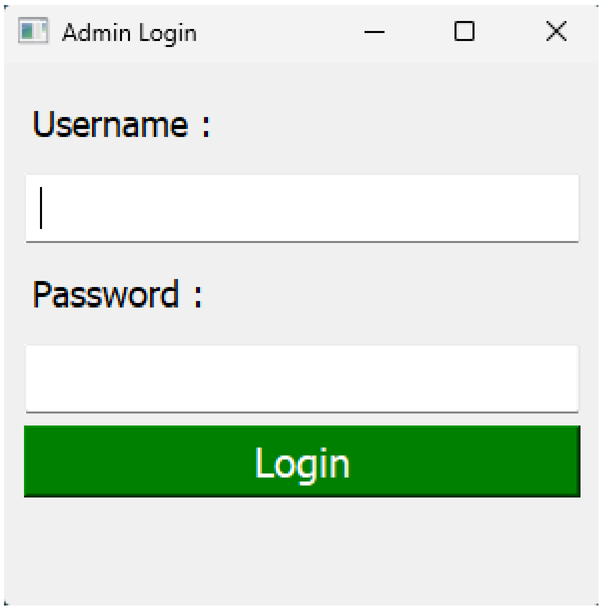
```
        vertical_layout.addWidget(self.name_input)
        vertical_layout.addWidget(contact_number_label)
        vertical_layout.addWidget(self.contact_number_input)
        vertical_layout.addWidget(email_label)
        vertical_layout.addWidget(self.email_input)
        vertical_layout.addWidget(position_label)
        vertical_layout.addWidget(self.position_input)
        vertical_layout.addWidget(submit_button)

        # Add the vertical layout to the main vertical layout
        self.vertical_9.addLayout(vertical_layout)
        self.vertical_9.addStretch()

    def addStaff(self):
        name = self.name_input.text()
        contact_number = self.contact_number_input.text()
        email = self.email_input.text()
        position = self.position_input.text()

        # Validate input fields
        if not name or not contact_number:
            QMessageBox.warning(self, "Warning", "Name and Contact Number are required fields.")
            return

        # Call DBOperation method to add staff
        if self.dbOperation.addStaff(name, contact_number, email, position):
            QMessageBox.information(self, "Success", "Staff member added successfully.")
            # Clear input fields
            self.name_input.clear()
            self.contact_number_input.clear()
            self.email_input.clear()
            self.position_input.clear()
            self.shift_start_input.clear()
            self.shift_end_input.clear()
        else:
            QMessageBox.critical(self, "Error", "Failed to add staff member. Please try again.")

    def exitCall(self):
        btton=self.sender()
        if btton:
            row=self.table.indexAt(btton.pos()).row()
            id =str(self.table.item(row,0).text())
            self.dbOperation.exitVehicle(id)
            self.table.removeRow(row)
```

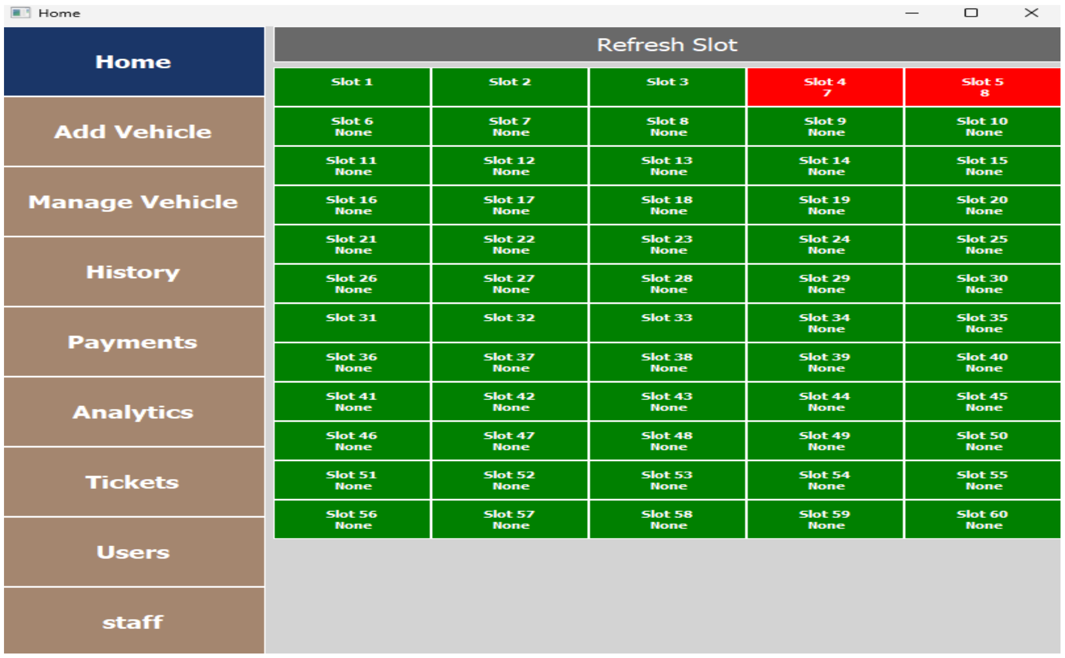
4. Main Program:

```
import sys
import os
from InstallWindow import InstallWindow
from LoginWindow import LoginScreen
from PyQt5.QtWidgets import QApplication,QSplashScreen,QLabel
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt
from PyQt5.QtCore import QTimer

class MainScreen():
    def showSplashScreen(self):
        self.pix=QPixmap("slash_img.jpg")
        self.splassh=QSplashScreen(self.pix,Qt.WindowStaysOnTopHint)
        self.splassh.show()


def showSetupWindow():
    mainScreen.splassh.close()
    installWindow.show()


def showLoginWindow():
    mainScreen.splassh.close()
    login.showLoginScreen()



app=QApplication(sys.argv)
login=LoginScreen()
mainScreen=MainScreen()
mainScreen.showSplashScreen()
installWindow=InstallWindow()

if os.path.exists("./config.json"):
    QTimer.singleShot(3000,showLoginWindow)
else:
    QTimer.singleShot(3000,showSetupWindow)


sys.exit(app.exec_())
```

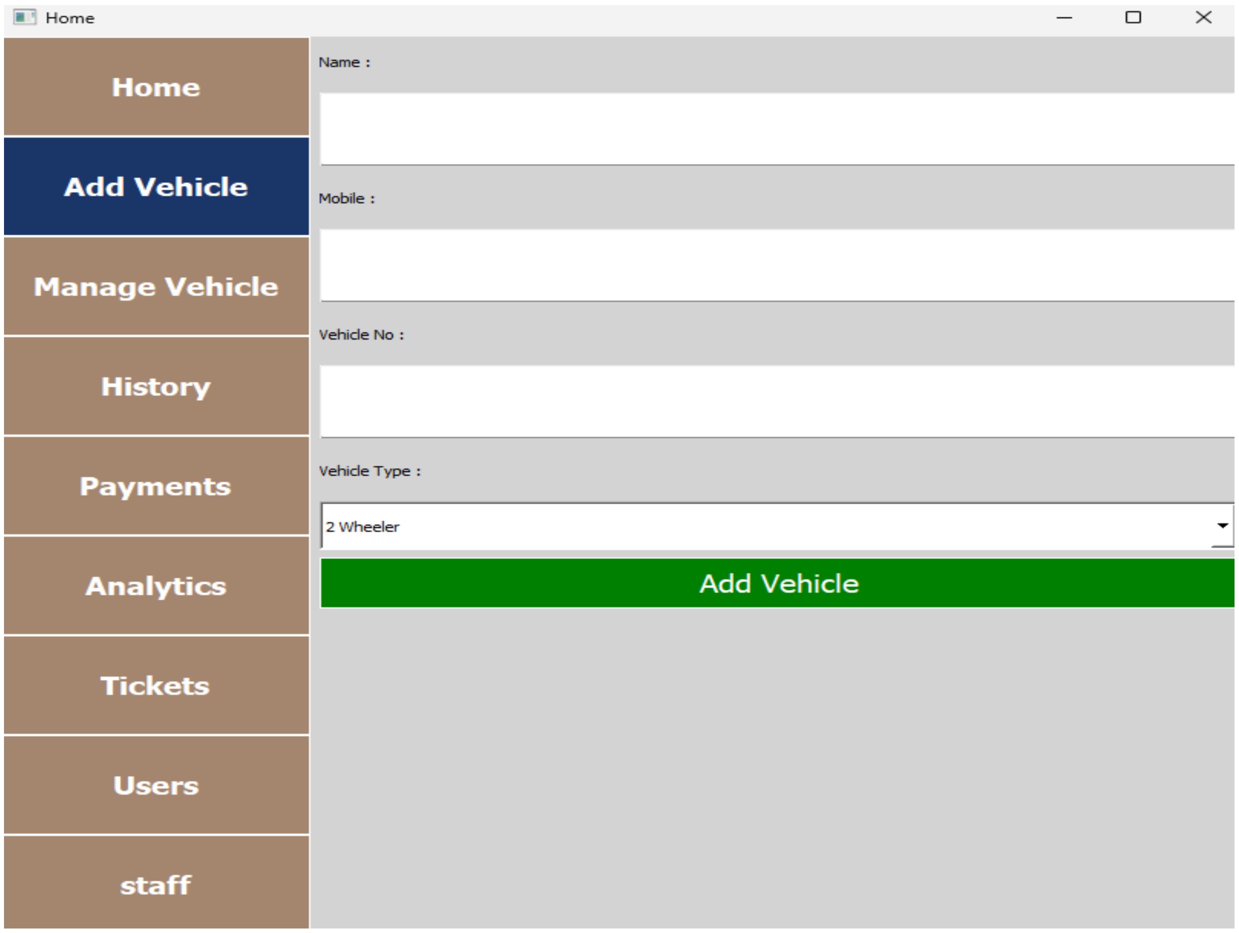
Result and Discussion :

The implementation of a parking management system featuring a Python GUI interface has yielded notable successes. Through the integration of a user-centric graphical interface, the project has significantly enhanced the overall user experience, simplifying tasks such as vehicle registration and fee payment. This enhancement has led to streamlined parking operations, notably reducing congestion and wait times. Furthermore, the system's robust reporting capabilities have provided invaluable insights into parking usage and revenue generation, empowering stakeholders with actionable data for informed decision-making. The implementation of efficient fee collection processes has resulted in a tangible increase in revenue. Favorable feedback from stakeholders and users alike underscores the system's efficacy in addressing pertinent parking management challenges. As the project progresses, continued refinement based on user feedback and evolving requirements will ensure its sustained relevance and impact





Slot 1	Slot 2	Slot 3	Slot 4 7	Slot 5 8
Slot 6 None	Slot 7 None	Slot 8 None	Slot 9 None	Slot 10 None
Slot 11 None	Slot 12 None	Slot 13 None	Slot 14 None	Slot 15 None
Slot 16 None	Slot 17 None	Slot 18 None	Slot 19 None	Slot 20 None
Slot 21 None	Slot 22 None	Slot 23 None	Slot 24 None	Slot 25 None
Slot 26 None	Slot 27 None	Slot 28 None	Slot 29 None	Slot 30 None
Slot 31	Slot 32	Slot 33	Slot 34 None	Slot 35 None
Slot 36 None	Slot 37 None	Slot 38 None	Slot 39 None	Slot 40 None
Slot 41 None	Slot 42 None	Slot 43 None	Slot 44 None	Slot 45 None
Slot 46 None	Slot 47 None	Slot 48 None	Slot 49 None	Slot 50 None
Slot 51 None	Slot 52 None	Slot 53 None	Slot 54 None	Slot 55 None
Slot 56 None	Slot 57 None	Slot 58 None	Slot 59 None	Slot 60 None



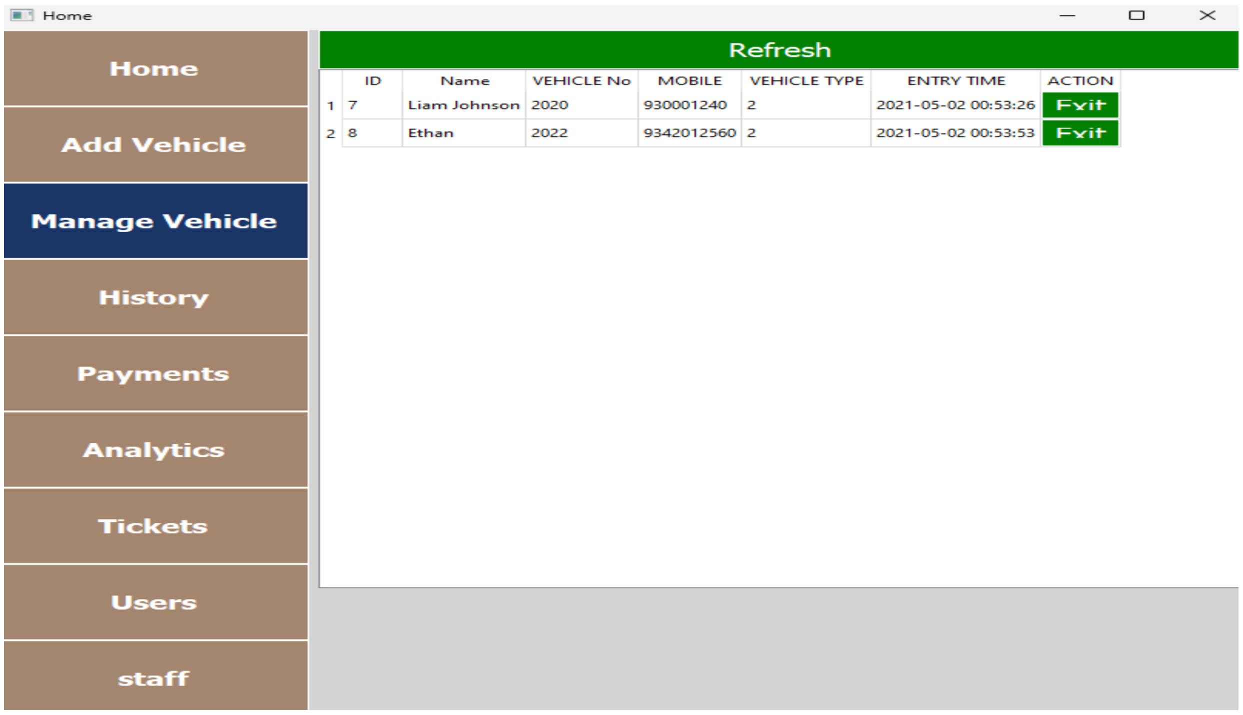
Name :

Mobile :

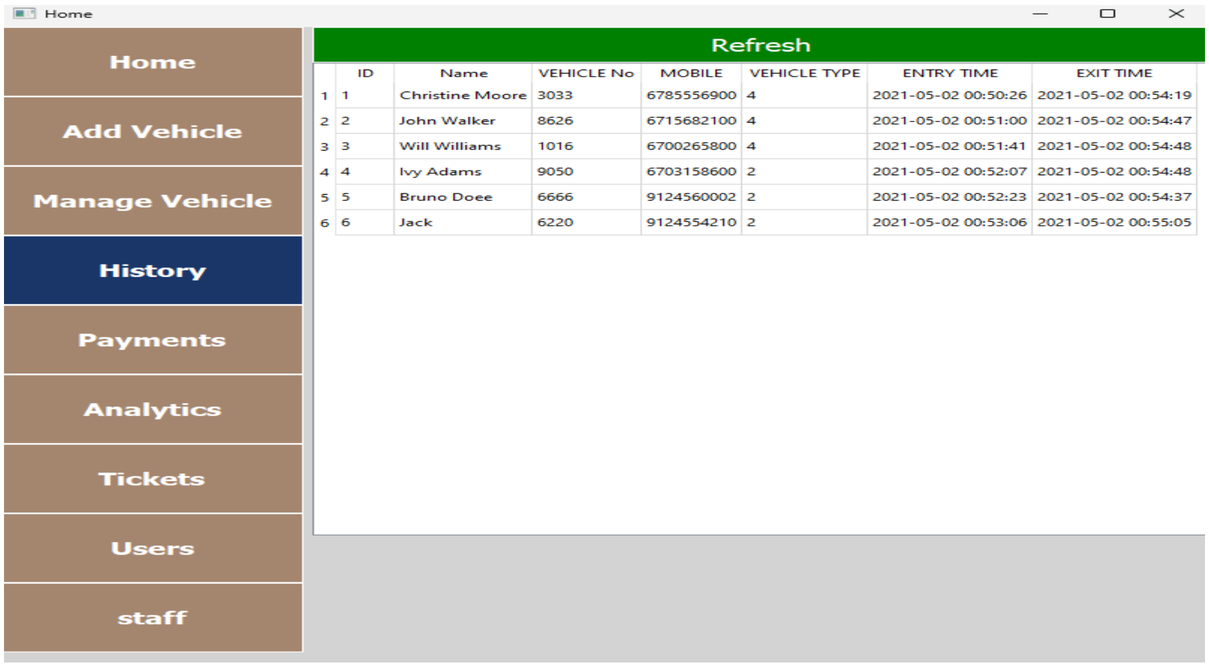
Vehicle No :

Vehicle Type :

2 Wheeler



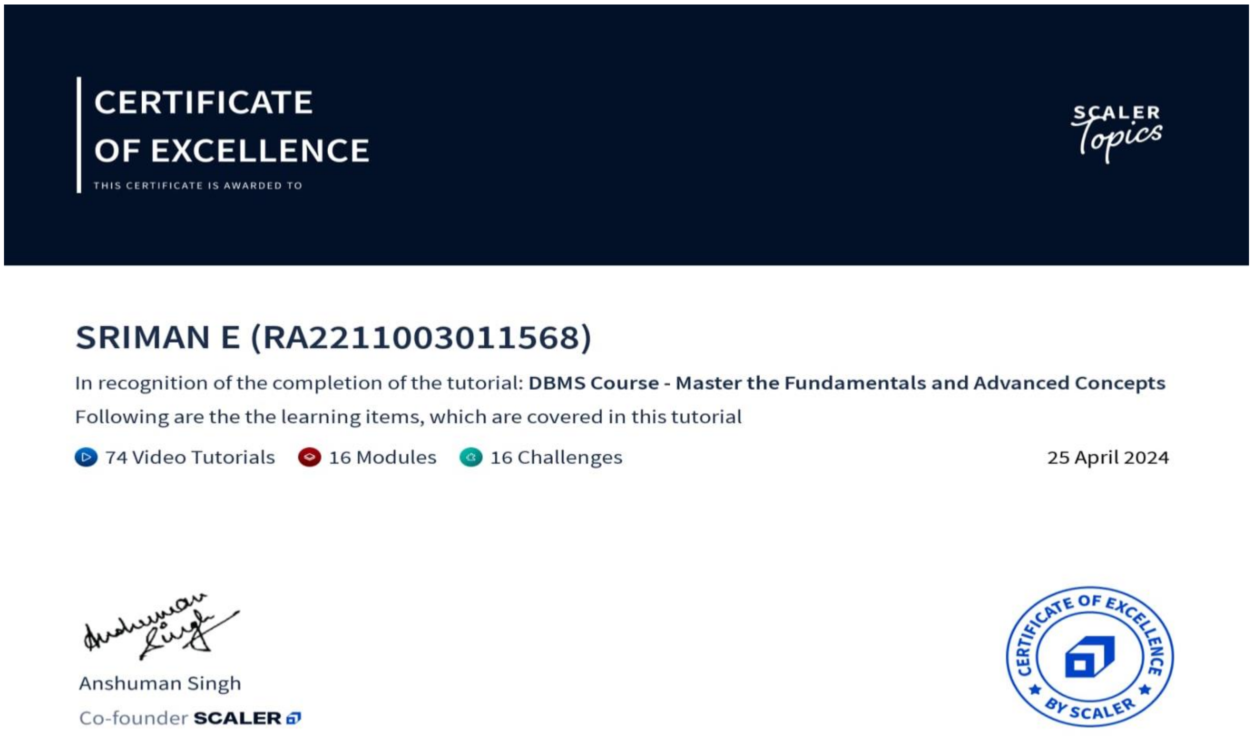
	ID	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	ACTION
1	7	Liam Johnson	2020	930001240	2	2021-05-02 00:53:26	Exit
2	8	Ethan	2022	9342012560	2	2021-05-02 00:53:53	Exit



	ID	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	EXIT TIME
1	1	Christine Moore	3033	6785556900	4	2021-05-02 00:50:26	2021-05-02 00:54:19
2	2	John Walker	8626	6715682100	4	2021-05-02 00:51:00	2021-05-02 00:54:47
3	3	Will Williams	1016	6700265800	4	2021-05-02 00:51:41	2021-05-02 00:54:48
4	4	Ivy Adams	9050	6703158600	2	2021-05-02 00:52:07	2021-05-02 00:54:48
5	5	Bruno Doee	6666	9124560002	2	2021-05-02 00:52:23	2021-05-02 00:54:37
6	6	Jack	6220	9124554210	2	2021-05-02 00:53:06	2021-05-02 00:55:05

REAL TIME PROJECT CERTIFICATE:

1. **Sriman E**



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

SRIMAN E (RA2211003011568)

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

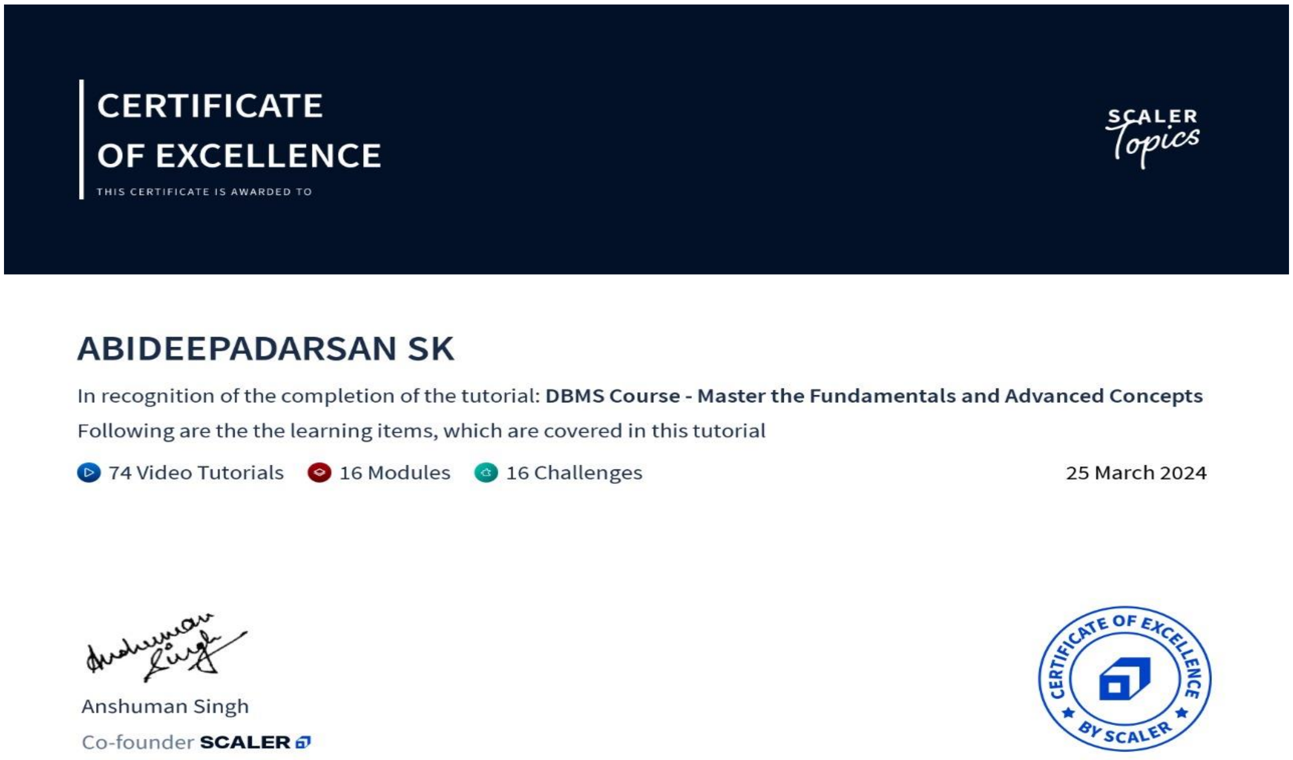
74 Video Tutorials   16 Modules   16 Challenges   25 April 2024

Anshuman Singh

Co-founder SCALER

CERTIFICATE OF EXCELLENCE BY SCALER

2. Abideepadarsan S K



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER
Topics

ABIDEEPADARSAN SK

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

25 March 2024

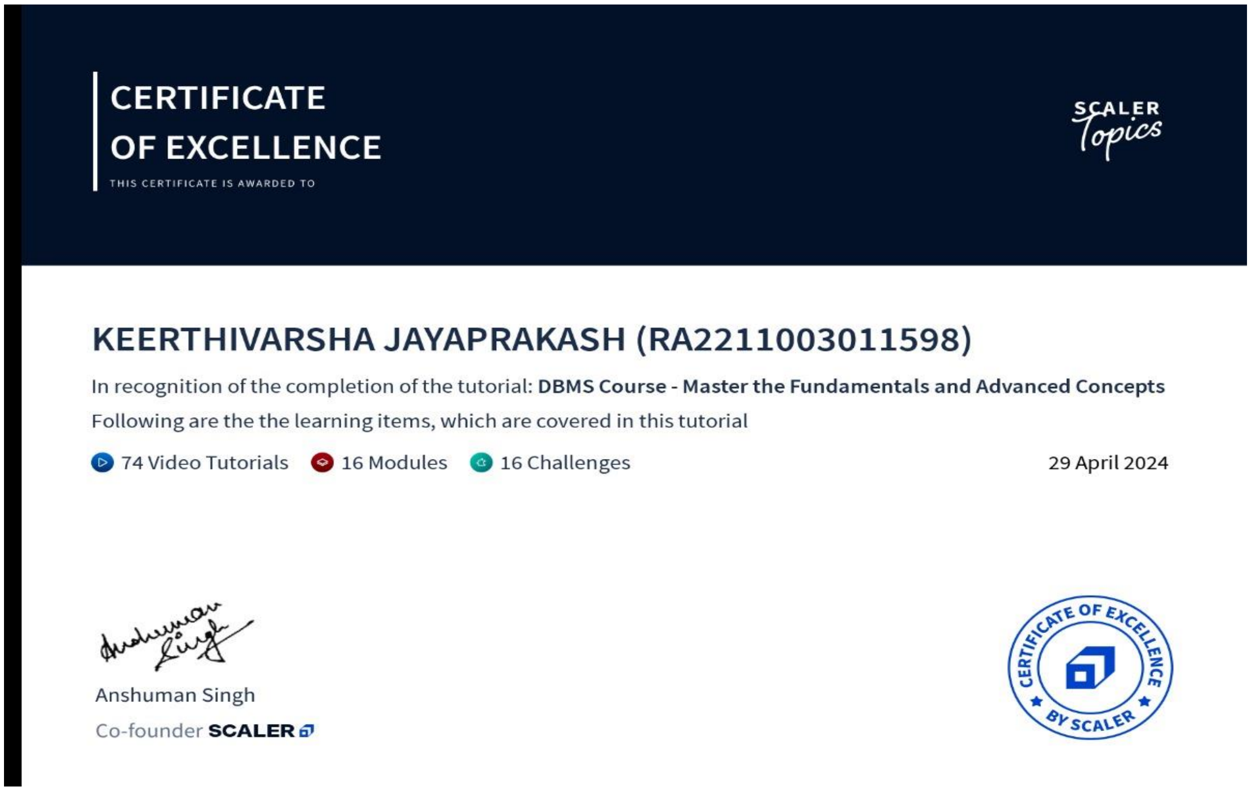
[illegible]

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER

3. Keerthivarsha J



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

KEERTHIVARSHA JAYAPRAKASH (RA2211003011598)

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

29 April 2024

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER