

## **Advance Database System**

Course Project Report

# **Smart Parking Management System**

Submitted by:

**Sheena Patel**

## Table of Contents

### 1. Requirements Collection & Analysis

#### 1.1 Database Project Proposal / Choice Summary

- Overview of chosen project
- Justification for selection

#### 1.2 Primary Actors

- Identification of user groups (actors)
- Geographic and demographic characteristics

#### 1.3 System Requirements

- Functional requirements
- Data usage within the system
- Types of transactions performed
- Data flow from application to database (input/output)

#### 1.4 Entities, Relationships, and Constraints

- Entity types and attributes
- Relationships between entities
- Identified database constraints

#### 1.5 Resulting Entities and Attributes (Schema Design)

- Formatted entity list with primary and foreign keys
- Example schema notation

#### 1.6 UI Prototype

- Wireframe
- Description

### 2. Conceptual & Logical Design

#### 2.1 Introduction

#### 2.2 Conceptual Design (ER Diagram)

- Entities and Attributes
- Relationships and Constraints

- Conceptual ER Diagram

### **2.3 Logical Design (Relational Mapping)**

- 7 Step Mapping Algorithm
- Final Relational Schema
- Logical Design Diagram

### **2.4 Normalization**

- First Normal Form (1<sup>st</sup> NF)
- Second Normal Form (2nd NF)
- Third Normal Form (3rd NF)

### **2.5 Reflective Report**

## **3. Physical Design & Performance**

### **3.1 Physical Model Design**

- Physical ER Diagram (MySQL Workbench)
- Table Structures (Columns, Data Types, Constraints)
- Relationship Cardinality

### **3.2 Forward Engineering**

- Forward Engineering SQL Script
- Screenshot – Execution in Elvis

### **3.3 Data Population**

- Screenshot – USER table
- Screenshot – VEHICLE table
- Screenshot – PARKINGLOT table
- Screenshot – PARKINGSLOT table
- Screenshot – RESERVATION table
- Screenshot – PAYMENT table
- Screenshot – SENSOR table
- Screenshot – FEEDBACK table
- Screenshot – REPORT table

### **3.4 SQL JOIN Queries**

- JOIN Query 1
- JOIN Query 2

### **3.5 Database Objects**

- Views (2) Code + Results
- Functions (2) – Code + Output
- Stored Procedures (2) – Code + Execution
- Triggers (2) – Code + Validation Output

### **3.6 Final Reflection**

- Summary of Experience
- Challenges & Resolutions

## **4. Frontend Design & Implementation**

### **4.1 Project Description**

### **4.2 CRUD Operations Screenshots**

### **4.3 Views, Functions, Stored Procedures, Triggers Output**

### **4.4 Project URL**

## **1. Requirements Collection & Analysis**

### **1.1 Database Project Proposal / Choice Summary**

#### **• Overview of chosen project**

Parking has become one of the issues in urban areas. Drivers will get an opportunity to search for empty parking spaces, reserve them in advance, and pay online. An ecosystem will empower administrators and parking lot managers with the ability to monitor usage, assign slots, manage prices, and generate reports. IoT sensor-based systems will offer real-time data regarding the availability of parking slots.

This database will be at the core of handling users, vehicles, slots, bookings, payments, and reviews. Frontend applications (web and mobile) will utilize the database in giving comfort of transaction usage and real-time updating.

#### **• Justification for selection**

I have selected this project because it:

- Addresses a genuine urban problem: traffic congestion and time wasted searching for a parking space.
- Caters to different categories of users such as drivers, managers, administrators, and payment processors, allowing a complete and realistic design of the database.
- Has enough broad scope to define over 20 functional requirements ranging from transaction handling and reporting to monitoring and payment processing.
- Allows UI prototyping at this stage similar to those real-world applications that are out there, i.e., ParkMobile, ParkWhiz, or SpotHero.
- Applies more advanced database design and functionality concepts to a real-world application instead of going down the traditional path of classroom systems such as payroll or library database.

## 1.2 Primary Actors

### • Identification of user groups (actors)

#### Overview

The Smart Parking Management System has already pre-prepared certain sets of users who are engaging with the system in different ways. These user actors are interested in activities such as searching for and booking parking spaces, monitoring parking lots, or maybe maintaining those parking lots. The technical counterpart may include payment gateways and IoT sensors talking to the backend database.

#### Actors and Their Roles

- **Drivers / Car Owners** – Find vacant parking spots, book them, pay for them, and inspect them.
- **Parking Lot Administrators / Attendants** – Monitor parking lot usage, update slot statuses, and handle exceptional events like cancellations or defective slots.
- **System Administrator** – Handles all users, parking lots, configurations, price rules, and reports pertaining to the overall system.
- **Payment Gateway / Banking System** – Process secure online payments for parking booking and refunding.
- **Sensors / IoT Devices** – Automatically and in real-time update the database on slot availability and occupancy.

### • Geographic and demographic characteristics

#### Geographic Scope:

Primarily cities with peak congestion: city centers, business complexes, malls, stadiums, and airports.

#### Demographics:

- **Drivers:** daily commuters, shoppers, business persons, event attenders.

- **Managers/Attendants:** People who assist in the management of parking lots.
- **Administrator:** City or organization officials managing several parking lots.
- **Payment Services:** Financial institutions or digital wallet providers integrated with the system.

## 1.3 System Requirements

The Smart Parking Management System is able to handle multiple user interactions, real-time slot availability, payment security, and reporting. The system needs to have a clear set of functional requirements with a good understanding of what data will be used, types of transactions established, and structured data flow from the application to the backend database.

### • Functional requirements

#### Functional Requirements:

- Secure registration and user login.
- Search slots available by location.
- Book slots in advance.
- Real-time update of slot availability.
- Slot status can be overridden by car park manager in case of errors manually.
- There are several lots at various locations to be managed.
- Facilitate digital payment (bank, wallet, card).
- Generate electronic receipts.
- Users can be allowed to cancel booking.
- Car park managers can see the real-time occupancy.
- Calculate parking fees automatically.
- Rules for pricing are set by admin (hourly, daily, penalties).
- Send reminders/notifications on things (e.g., session expiring).

- Store car info (plate number, type, color).
- Admin can delete or add parking lots and slots.
- Keep record of all transactions for auditing.
- Collection of user ratings and feedback.
- Generate reports on usage and revenue.
- Sensors automatically change slot status.
- Provide emergency overrides (fire lane, blocked slot).
- Support reserved and walk-in parking.
- Show booking history to users.
- Process refunds for cancelled bookings.
- Make mobile and web applications accessible.

#### • Data usage within the system

- **User Data**-Authentication, contact information, and account role.
- **Vehicle Data**-License number and vehicle type information related to the user.
- **Parking Data**-Parking lots, parking slots, and occupancy status.
- **Reservation Data**-Slot reservation with start time and end time and status.
- **Payment Data**-Transaction amount and mode, receipts, and refund records.
- **Feedback Data**-Rating and comments on the quality of service.
- **Report Data**-Usage statistics, revenue trends, and violation records.

#### • Types of transactions performed

- **User Transactions** – Account registration, logging into the system, and adding a new vehicle.
- **Parking Transactions** – Location searching for availability, booking/cancellation of a parking space.
- **Payment Transactions** – Payment processing, refund issuance.
- **Management Transactions** – Slot updation and report generation by managers/admins.
- **Feedback Transactions** – Posting ratings/comments on successful booking.
- **System Transactions** – Automatic updation through IoT sensors, sending alerts, and maintaining system logs.

- **Data flow from application to database (input/output)**

Input Transactions:

- User registration → Saving new user records.
- Reservation creation → Saving the booking in the Reservation table.
- Payment confirmation → Saving transactions in the Payment table.
- Sensor update → Updating ParkingSlot status.

Output Transactions:

- Search availability → Queries parking slots and returns free slots.
- View reservations → Retrieves reservations of a given user.
- Generate reports → Retrieves aggregated data for the management dashboard.
- Notifications → Pulls reservation expiration information from the system and pushes notifications to users.

## 1.4 Entities, Relationships, and Constraints

Since the Smart Parking Management System operates on data manipulation and storage, there should be a database model with structure. This stage defines the entities (real things stored in tables), their associations (how entities are connected), and the respective constraints (regulations to keep things together and consistent).

- **Entity types and attributes**

### 1) User

- UserID (PK), Name, Email, PhoneNumber, Password, UserType

**2) Vehicle**

- VehicleID (PK), UserID (FK), LicensePlate, VehicleType, Color

**3) ParkingLot**

- LotID (PK), Name, Location, TotalSlots, ManagerID (FK)

**4) ParkingSlot**

- SlotID (PK), LotID (FK), SlotNumber, Status, SlotType

**5) Reservation**

- ReservationID (PK), SlotID (FK), UserID (FK), VehicleID (FK), StartTime, EndTime, Status

**6) Payment**

- PaymentID (PK), ReservationID (FK), Amount, PaymentMethod, PaymentStatus, TransactionDate

**7) Sensor**

- SensorID (PK), SlotID (FK), Status, LastUpdate

**8) Feedback**

- FeedbackID (PK), UserID (FK), LotID (FK), Rating, Comment, FeedbackDate

**9) Report**

- ReportID (PK), LotID (FK), ReportType, GeneratedDate

**• Relationships between entities**

- A User owns multiple Vehicles (1:N).
- A User owns multiple Reservations (1:N).
- A ParkingLot owns multiple ParkingSlots (1:N).

- One ParkingSlot can host many Reservations over a time span but only one activeReservation at any instant in time (1:N with constraint).
- One Reservation produces one Payment (1:1).
- One ParkingSlot can own one Sensor to monitor occupancy (1:1).
- A User gives multiple Feedbacks on multiple ParkingLots (M:N resolved by means of Feedback).
- An Admin generates multiple Reports for one ParkingLot (1:N).

### • Identified database constraints

1) **Primary Keys:** Every entity must have a unique identifier

- User: UserID
- Vehicle: VehicleID
- ParkingLot: LotID
- ParkingSlot: SlotID
- Reservation: ReservationID
- Payment: PaymentID
- Sensor: SensorID
- Feedback: FeedbackID
- Report: ReportID

2) **Foreign Keys:** Always reference valid records

- Vehicle.UserID → User(UserID)
- ParkingLot.ManagerID → User(UserID)
- ParkingSlot.LotID → ParkingLot(LotID)
- Reservation.SlotID → ParkingSlot(SlotID)
- Reservation.UserID → User(UserID)
- Reservation.VehicleID → Vehicle(VehicleID)
- Payment.ReservationID → Reservation(ReservationID)
- Sensor.SlotID → ParkingSlot(SlotID)
- Feedback.UserID → User(UserID)
- Feedback.LotID → ParkingLot(LotID)
- Report.LotID → ParkingLot(LotID)

3) **Reservation Constraints:**

- A ParkingSlot must not be reserved twice for the same overlapping time periods.

- The Reservation StartTime must be before the EndTime.

**4) Payment Constraints:**

- Every Reservation must be linked to exactly one Payment before confirmation.

**5) Feedback Constraints:**

- Feedback is possible only from a user upon making a Reservation.

**6) Slot Status Constraints:**

- ParkingSlot status changes automatically on reservation start or end.

**7) Sensor Constraints:**

- IoT sensor updates will override manual manager updates to maintain the system in check accurately.

## 1.5 Resulting Entities and Attributes (Schema Design)

**• Formatted entity list with primary and foreign keys**

### 1. User

Primary Key (PK): UserID

Attributes: Name, Email, PhoneNumber, Password, UserType

UserType defines role (Driver, Manager, Admin)

### 2. Vehicle

Primary Key (PK): VehicleID

Foreign Key (FK): UserID → User(UserID)

Attributes: LicensePlate, VehicleType (Car, Bike, Truck, etc.), Color

### **3. ParkingLot**

Primary Key (PK): LotID

Foreign Key (FK): ManagerID → User(UserID)

Attributes: Name, Location (Address/GPS), TotalSlots

### **4. ParkingSlot**

Primary Key (PK): SlotID

Foreign Key (FK): LotID → ParkingLot(LotID)

Attributes: SlotNumber, Status (Available, Reserved, Occupied, OutOfService), SlotType (Compact, Large, EV, Handicapped)

### **5. Reservation**

Primary Key (PK): ReservationID

Foreign Keys (FKs):

SlotID → ParkingSlot(SlotID)

UserID → User(UserID)

VehicleID → Vehicle(VehicleID)

Attributes: StartTime, EndTime, Status (Active, Completed, Cancelled)

### **6. Payment**

Primary Key (PK): PaymentID

Foreign Key (FK): ReservationID → Reservation(ReservationID)

Attributes: Amount, PaymentMethod (Card, Wallet, Cash), PaymentStatus (Paid, Pending, Refunded), TransactionDate

### **7. Sensor**

Primary Key (PK): SensorID

Foreign Key (FK): SlotID → ParkingSlot(SlotID)

Attributes: Status (Free/Occupied), LastUpdate

## 8. Feedback

Primary Key (PK): FeedbackID

Foreign Keys (FKs):

UserID → User(UserID)

LotID → ParkingLot(LotID)

Attributes: Rating (1–5), Comment, FeedbackDate

## 9. Report

Primary Key (PK): ReportID

Foreign Key (FK): LotID → ParkingLot(LotID)

Attributes: ReportType (Usage, Revenue, Violations), GeneratedDate

### • Example schema notation

**User**(UserID, Name, Email, PhoneNumber, Password, UserType)

PK = UserID

**Vehicle**(VehicleID, UserID, LicensePlate, VehicleType, Color)

PK = VehicleID

FK = UserID → User(UserID)

**ParkingLot**(LotID, Name, Location, TotalSlots, ManagerID)

PK = LotID

FK = ManagerID → User(UserID)

**ParkingSlot**(SlotID, LotID, SlotNumber, Status, SlotType)

PK = SlotID

FK = LotID → ParkingLot(LotID)

**Reservation**(ReservationID, SlotID, UserID, VehicleID, StartTime, EndTime, Status)

PK = ReservationID

FK = SlotID → ParkingSlot(SlotID)

FK = UserID → User(UserID)

FK = VehicleID → Vehicle(VehicleID)

**Payment**(PaymentID, ReservationID, Amount, PaymentMethod, PaymentStatus, TransactionDate)

PK = PaymentID

FK = ReservationID → Reservation(ReservationID)

**Sensor**(SensorID, SlotID, Status, LastUpdate)

PK = SensorID

FK = SlotID → ParkingSlot(SlotID)

**Feedback**(FeedbackID, UserID, LotID, Rating, Comment, FeedbackDate)

PK = FeedbackID

FK = UserID → User(UserID)

FK = LotID → ParkingLot(LotID)

**Report**(ReportID, LotID, ReportType, GeneratedDate)

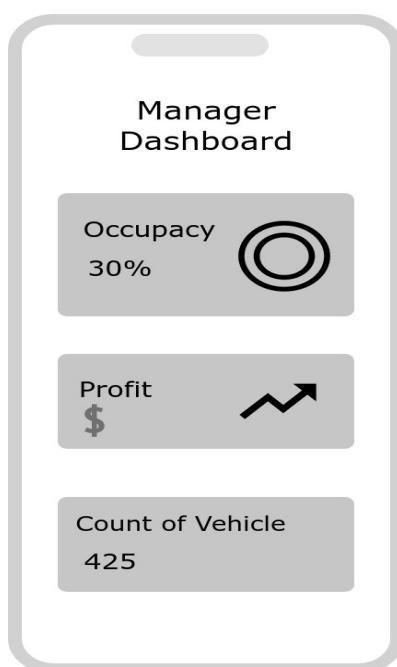
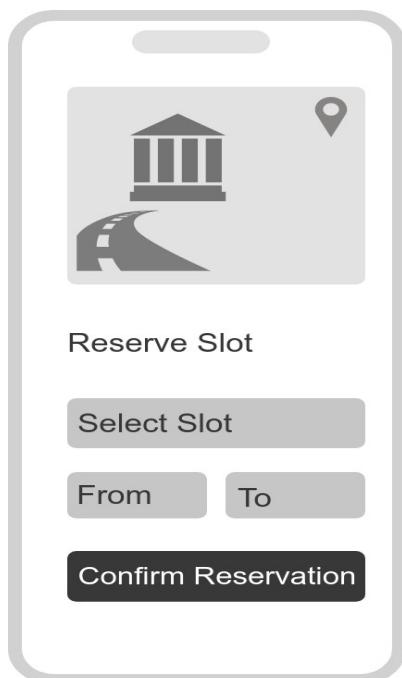
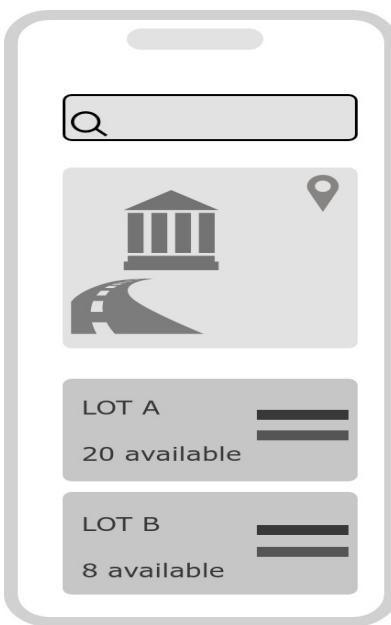
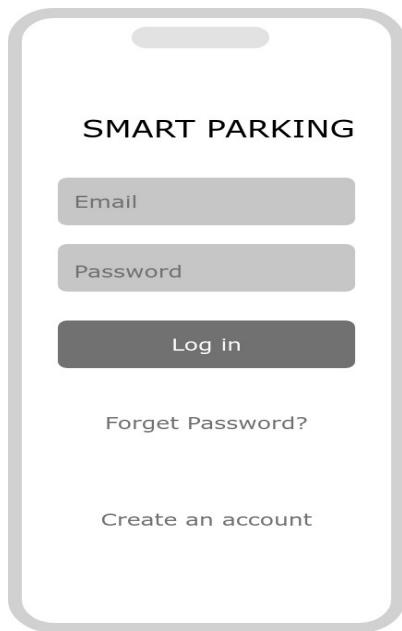
PK = ReportID

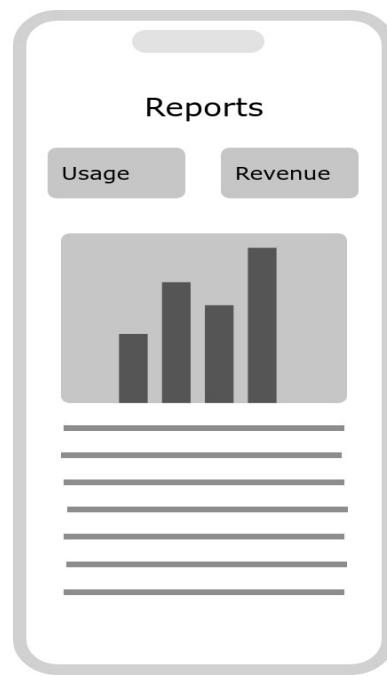
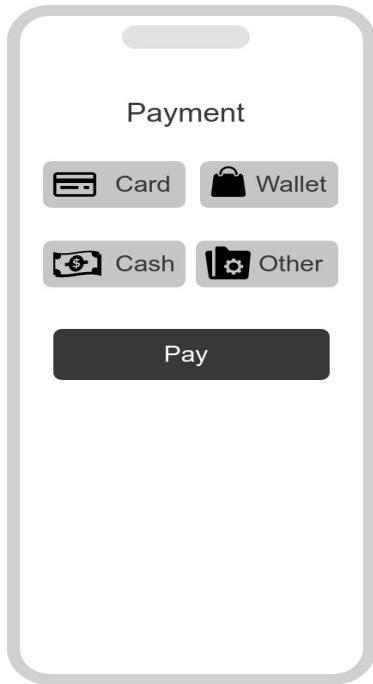
FK = LotID → ParkingLot(LotID)

## 1.6 UI Prototype

- Wireframe

<https://pr.to/TPS347/>



A mobile phone interface showing a feedback screen. At the top, it says "Feedback". Below that is a row of five stars, with the first four filled and the last one outlined. Below the stars is a text input field labeled "Comment". At the bottom is a large, dark grey "Submit" button.

<b>Screen</b>	<b>Purpose / Data Interaction</b>	<b>Backend Entities Involved</b>
1. Login / Registration Screen	User authentication and account creation. Collects user details and credentials.	User
2. Home Dashboard / Map View	Displays nearby parking lots with real-time slot availability (status fetched from Sensors table).	ParkingLot, ParkingSlot, Sensor
3. Slot Selection and Reservation Form	Allows driver to select a slot, choose time duration, and confirm reservation.	Reservation, Vehicle, User
4. Payment Interface	Processes booking payment through integrated gateway (simulated in prototype).	Payment, Reservation
5. Manager Dashboard	Lets lot managers monitor occupancy, update slot status, and view feedback.	ParkingSlot, Reservation, Feedback
6. Reports / Analytics Screen	Visualizes usage statistics, revenue, and feedback trends.	Report, Feedback
7. Feedback Form	Enables drivers to rate parking experience after completed reservation.	Feedback, User, ParkingLot

## **2. Conceptual & Logical Design**

### **2.1 Introduction**

The Smart Parking Management System (SPMS) database is created to handle the different aspects such as users, parking lots, slots, vehicles, reservations, sensors, and payments in an orderly and efficient way. The objective of this project is to come up with a relational database model that is normalized and presents all data interactions in a smart parking setting.

The whole design process is detailed in the report starting from conceptual modeling to normalization, adhering to the systematic method given by Elmasri & Navathe (2017), Fundamentals of Database Systems (7th Edition).

### **2.2 Conceptual Design (ER Diagram)**

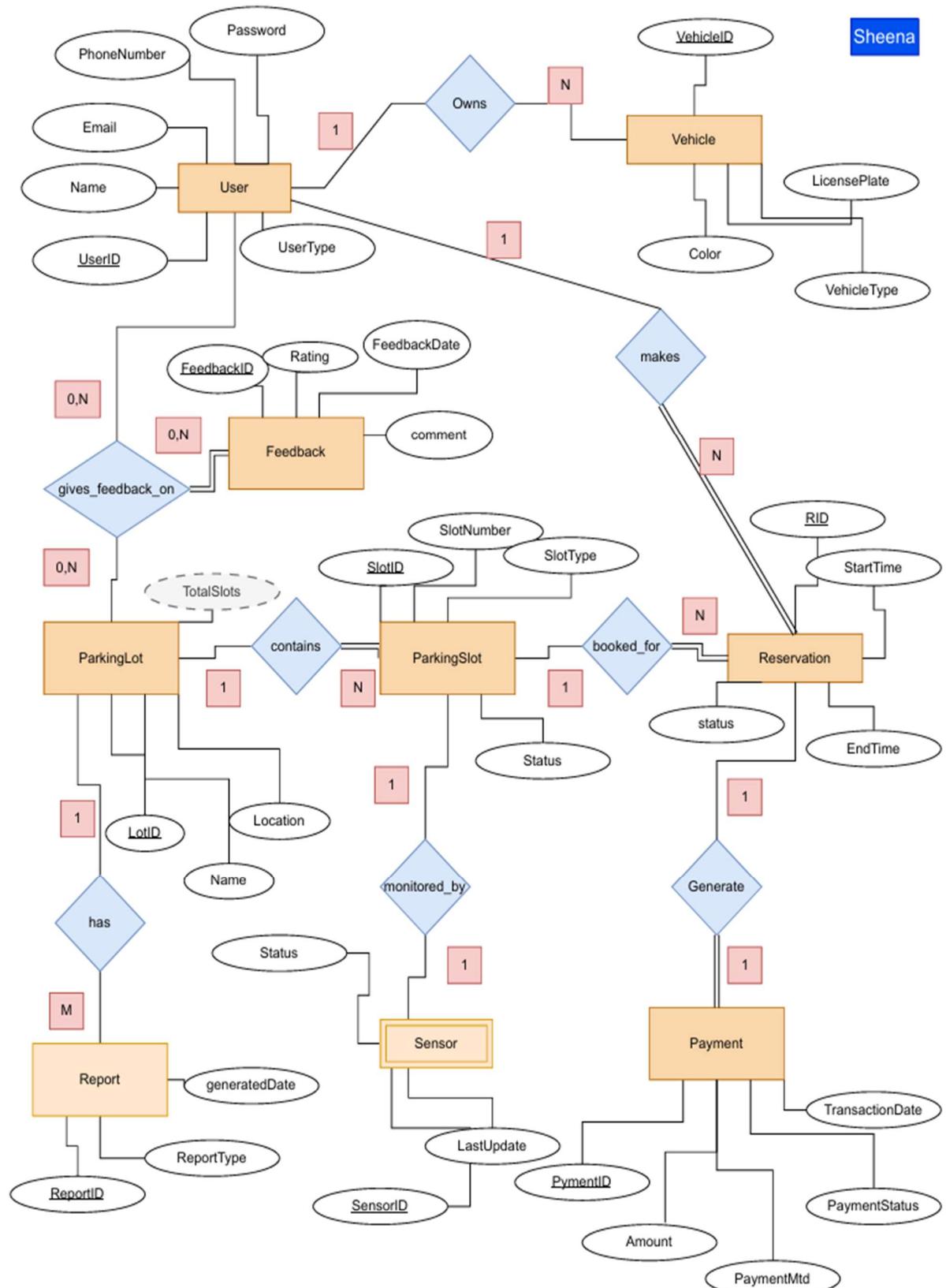
#### **• Entities and Attributes**

Entity	Key Attribute	Other Attributes
User	UserID	Name, Email, PhoneNumber, Password, UserType
Vehicle	VehicleID	LicensePlate, VehicleType, Color
ParkingLot	LotID	Name, Location, TotalSlots, ManagerID
ParkingSlot	SlotID	SlotNumber, SlotType, Status
Reservation	ReservationID	StartTime, EndTime, Status
Payment	PaymentID	Amount, PaymentMethod, PaymentStatus, TransactionDate
Sensor	SensorID	Status, LastUpdate
Feedback	FeedbackID	Rating, Comment, FeedbackDate
Report	ReportID	ReportType, GeneratedDate

- Relationships and Constraints**

Relationship	Description	Cardinality	Participation
<b>owns</b>	A User owns multiple Vehicles	1:N	Total on Vehicle side
<b>makes</b>	A User can make multiple Reservations	1:N	Partial on User, Total on Reservation
<b>contains</b>	A ParkingLot contains multiple ParkingSlots	1:N	Total on ParkingSlot side
<b>booked_for</b>	A Reservation is made for one ParkingSlot	1:1	Total on both sides
<b>generates</b>	A Reservation generates one Payment	1:1	Total on Payment side
<b>monitored_by</b>	A ParkingSlot is monitored by a Sensor	1:1	Total on Sensor side
<b>gives/about</b>	A User gives Feedback about a ParkingLot	M:N (resolved via Feedback)	Total on Feedback side
<b>has</b>	A ParkingLot has multiple Reports	1:N	Total on Report side

## • Conceptual ER Diagram



## 2.3 Logical Design Diagram

- **7 Step Mapping Algorithm**

### Step 1: Mapping of Regular(Strong) Entity Types

- Each strong entity in the ER diagram becomes a relation.
- All simple attributes are included.
- The key attribute becomes the primary key (PK) of the relation.

Resulting Relations:

USER(UserID, Name, Email, PhoneNumber, Password, UserType)  
 VEHICLE(VehicleID, LicensePlate, VehicleType, Color)  
 PARKINGLOT(LotID, Name, Location, TotalSlots, ManagerID)  
 PARKINGSLOT(SlotID, SlotNumber, Status, SlotType)  
 RESERVATION(ReservationID, StartTime, EndTime, Status)  
 PAYMENT(PaymentID, Amount, PaymentMethod, PaymentStatus, TransactionDate)  
 SENSOR(SensorID, Status, LastUpdate)  
 FEEDBACK(FeedbackID, Rating, Comment, FeedbackDate)  
 REPORT(ReportID, ReportType, GeneratedDate)

### Step 2: Mapping of Weak Entity Types

If a weak entity depends on an owner entity, include the owner's PK as a foreign key(FK) and use a composite key if necessary

- Sensor is dependent on ParkingSlot → Adding SlotID as a foreign key.

SENSOR(SensorID, SlotID, Status, LastUpdate)

### Step 3: Mapping of Binary 1:1 Relationships

Choose one relation (usually the one with total participation) and include the PK of the other as a foreign key.

- Reservation – Payment is 1:1 → Add ReservationID as FK in PAYMENT.

PAYMENT(PaymentID, ReservationID, Amount, PaymentMethod, PaymentStatus, TransactionDat

### Step 4: Mapping of Binary 1:N Relationships

Place the PK of the “1-side” as a foreign key on the “N-side”.

Relationship	1-side	N-side	Foreign Key Added
User – Vehicle	User	Vehicle	UserID
ParkingLot – ParkingSlot	ParkingLot	ParkingSlot	LotID
User – Reservation	User	Reservation	UserID
Vehicle – Reservation	Vehicle	Reservation	VehicleID
ParkingSlot – Reservation	ParkingSlot	Reservation	SlotID

Updated relations:

VEHICLE(VehicleID, UserID, LicensePlate, VehicleType, Color)

PARKINGSLOT(SlotID, LotID, SlotNumber, Status, SlotType)

RESERVATION(ReservationID, SlotID, UserID, VehicleID, StartTime, EndTime, Status)

### Step 5: Mapping of Binary M:N Relationships

Create a new relation whose PK is the combination of the PKs of the participating entities. Include any attributes of the relationship.

- User – ParkingLot (M:N) resolved by Feedback associative entity → UserID and LotID are foreign keys referencing USER and PARKINGLOT.

FEEDBACK(FeedbackID, UserID, LotID, Rating, Comment, FeedbackDate)

### **Step 6: Mapping of Multivalued Attributes**

Create a new relation if an attribute can have multiple values, with a composite key of (PK, attribute).

My model has no composite key

### **Step 7: Mapping of Derived Attributes**

Derived attributes are normally not stored; they can be computed when needed.

For example:

- TotalSlots in ParkingLot can be derived by counting related ParkingSlot rows.

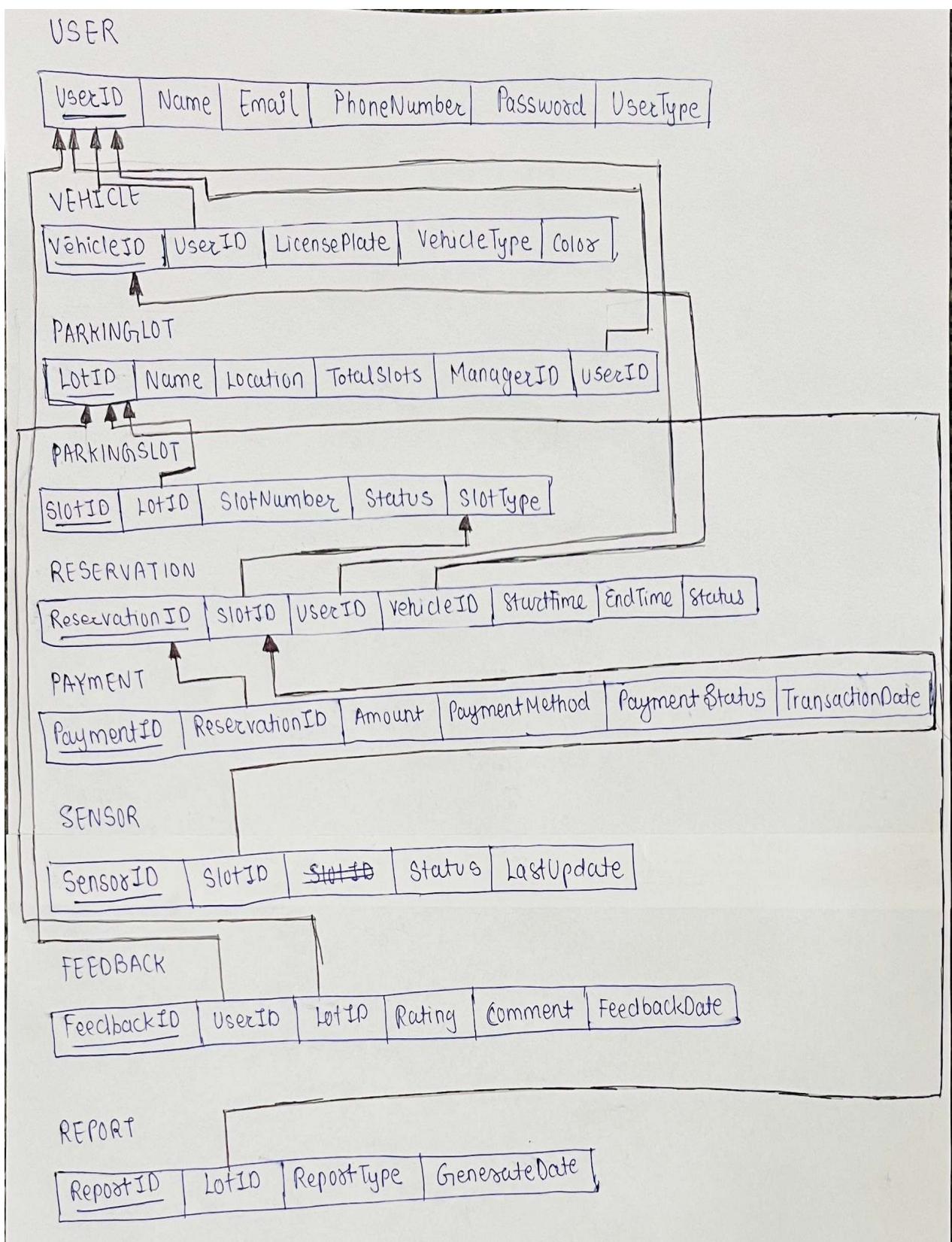
Hence, no relation added.

### **• Final Relational Schema**

- USER(UserID PK, Name, Email, PhoneNumber, Password, UserType)
- VEHICLE(VehicleID PK, UserID FK→USER(UserID), LicensePlate, VehicleType, Color)
- PARKINGLOT(LotID PK, Name, Location, TotalSlots, ManagerID FK→USER(UserID))

- PARKINGSLOT(SlotID PK, LotID FK→PARKINGLOT(LotID), SlotNumber, Status, SlotType)
- RESERVATION(ReservationID PK, SlotID FK→PARKINGSLOT(SlotID), UserID FK→USER(UserID), VehicleID FK→VEHICLE(VehicleID), StartTime, EndTime, Status)
- PAYMENT(PaymentID PK, ReservationID FK→RESERVATION(ReservationID) UNIQUE, Amount, PaymentMethod, PaymentStatus, TransactionDate)
- SENSOR(SensorID PK, SlotID FK→PARKINGSLOT(SlotID), Status, LastUpdate)
- FEEDBACK(FeedbackID PK, UserID FK→USER(UserID), LotID FK→PARKINGLOT(LotID), Rating, Comment, FeedbackDate)
- REPORT(ReportID PK, LotID FK→PARKINGLOT(LotID), ReportType, GeneratedDate)

• Logical Design Diagram



## 2.4 Normalization

### • First Normal Form (1<sup>st</sup> NF)

**Rule:** Each attribute must contain atomic (single-valued) data, and every record must be unique.

- All attributes in relations are already atomic, there is no multi values attributes.
- Therefore, all relations satisfy 1<sup>st</sup> NF

### • Second Normal Form (2nd NF)

**Rule:** A relation in 1NF is in 2NF if every non-key attribute depends on the whole primary key, not just part of a composite key.

- There are no composite primary key, thus there is no partial dependency
- Therefore, all relations satisfy 2nd NF

### • Third Normal Form (3rd NF)

**Rule:** A relation in 2NF is in 3NF if every non-key attribute depends directly on the primary key — not on another non-key attribute.

Relation	Check for Transitive Dependencies
USER	All depend directly on UserID.
VEHICLE	UserID FK, all other attributes depend on VehicleID.
PARKINGLOT	All attribute depends on LotID. Fks are legit.
PARKINGSLOT	All attributes depend on SlotID.
RESERVATION	All depend on ReservationID. Fks are legit.
PAYMENT	All depend on PaymentID.

SENSOR	All depend on SensorID.
FEEDBACK	All depend on FeedbackID.
REPORT	All depend on ReportID.

- No transitive dependencies exists , therefore all relations satisfy 3<sup>rd</sup> NF.

## 2.5 Reflective Report

Developing the database for the Smart Parking Management System required going through conceptual, logical, and normalization stages, as described by Elmasri & Navathe (2017).

Conceptual design mainly revolved around the challenges of defining participation constraints and cardinality ratios. The decision of feedback being modeled as an associative entity between USER and PARKINGLOT resolved the M:N relationship nicely.

During the logical design step, the 7-step mapping algorithm was employed and it turned out to be a very clear and systematic way to get from conceptual to relational schema. Each relationship was examined very carefully in order to place foreign keys correctly and thus, to ensure referential integrity (e.g., LotID in PARKINGSLOT, ReservationID in PAYMENT).

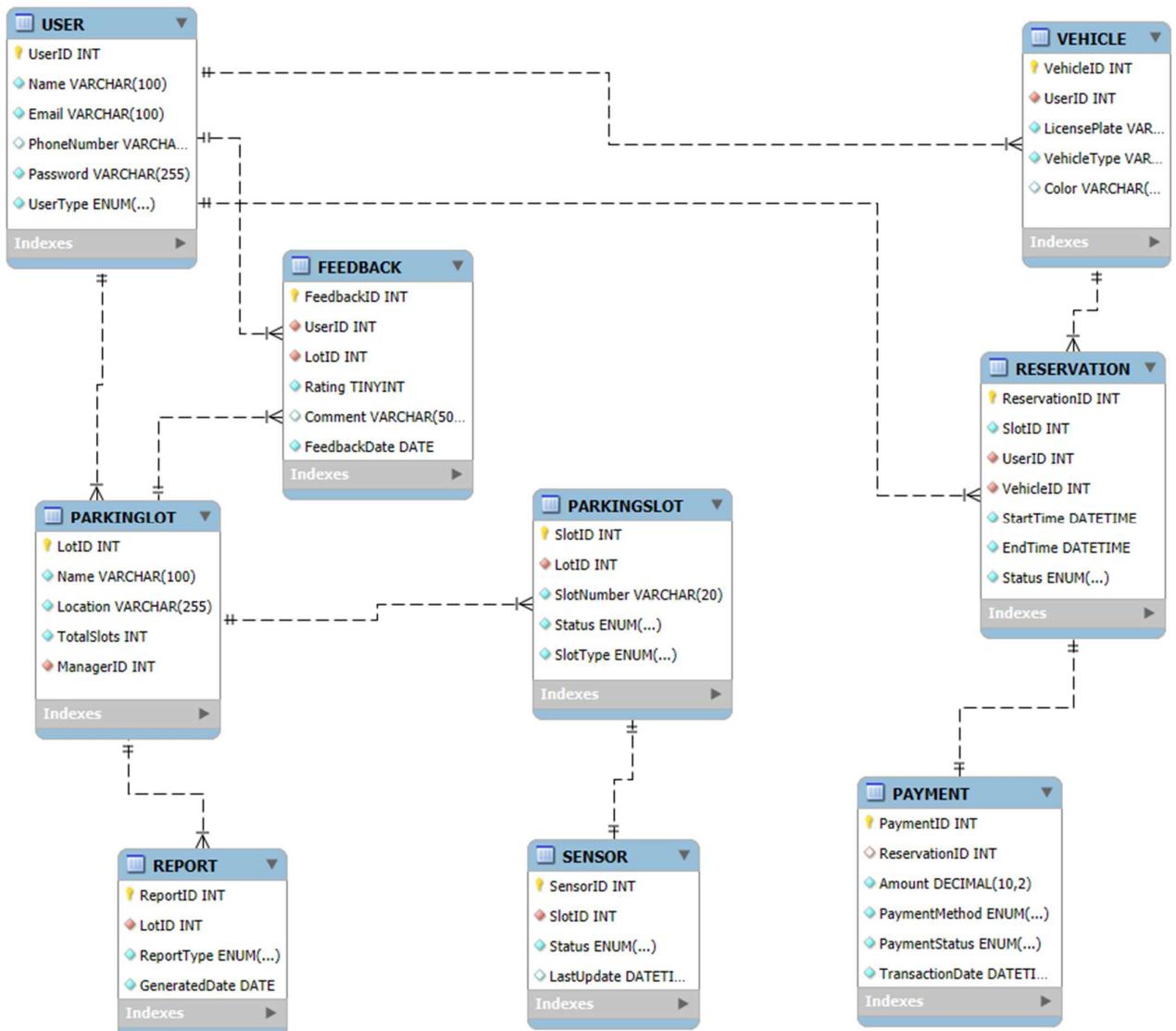
The normalization phase clarified once again the need for doing dependency analysis in order to avoid data anomalies. Discovering possible transitive dependencies (for example, ManagerName → ManagerID → PARKINGLOT) increased the efficiency of the design and ensured data integrity.

In summary, the process applied to design data integrity, clarity, and structured methodology. The end normalized scheme satisfies all functional requirements of the Smart Parking Management System and is in complete agreement with relational database theory.

### 3. Physical Design & Performance

#### 3.1 Physical Model Design

- Physical ER Diagram (MySQL Workbench)



- Table Structures (Columns, Data Types, Constraints)

**USER - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
UserID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Name	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Email	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
PhoneNumber	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Password	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
UserType	ENUM('Driver', 'Mana...')	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name:  Data Type:

**VEHICLE - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
VehicleID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
UserID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
LicensePlate	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
VehicleType	VARCHAR(30)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Color	VARCHAR(30)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name:  Data Type:

**PARKINGLOT - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
LotID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
Name	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
Location	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
TotalSlots	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
ManagerID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					

Column Name:  Data Type:



**SENSOR - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
SensorID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
SlotID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Status	ENUM('Free', 'Occupied')	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'Free'
LastUpdate	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

**REPORT - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ReportID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
LotID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
ReportType	ENUM('Usage', 'Revenue')	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
GeneratedDate	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					

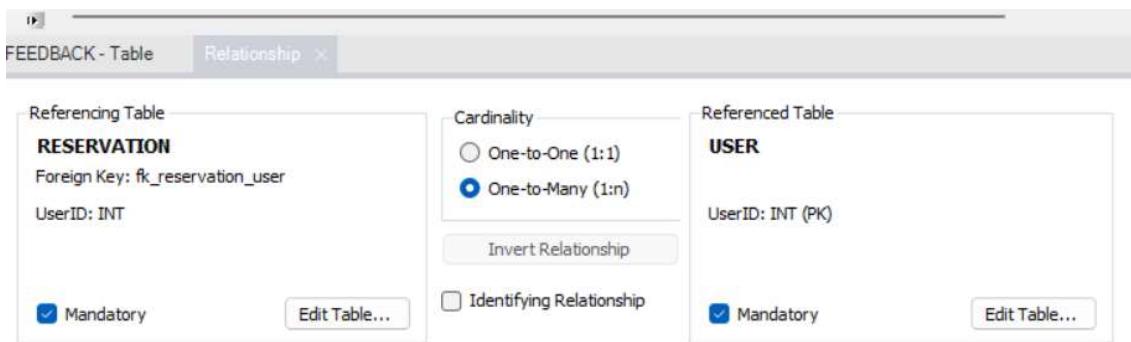
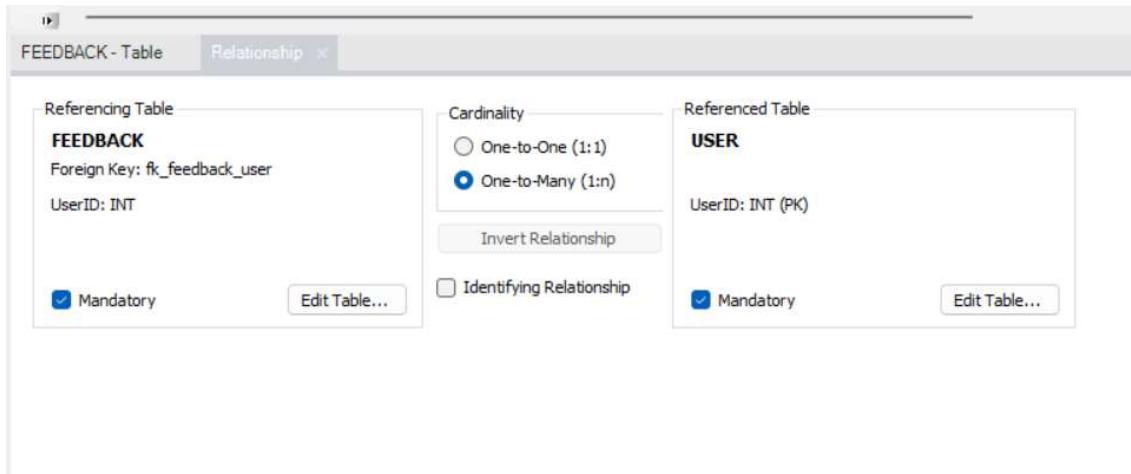
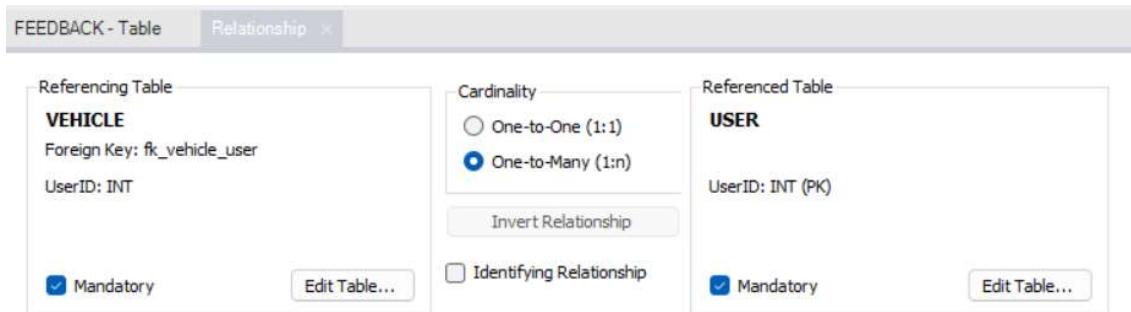
**Catalog Tree**

- patels184
  - Tables
    - FEEDBACK
    - PARKINGLOT
    - PARKINGSLOT
    - PAYMENT
    - REPORT
    - RESERVATION
    - SENSOR
    - USER
    - VEHICLE
  - Views

**FEEDBACK - Table**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
FeedbackID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
UserID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
LotID	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
Rating	TINYINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
Comment	VARCHAR(500)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
FeedbackDate	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					

- Relationship Cardinality



**FEEDBACK - Table Relationship**

Referencing Table <b>PARKINGLOT</b> Foreign Key: fk_parkinglot_manager ManagerID: INT	Cardinality <input type="radio"/> One-to-One (1:1) <input checked="" type="radio"/> One-to-Many (1:n)  <input type="checkbox"/> Invert Relationship  <input checked="" type="checkbox"/> Mandatory <a href="#">Edit Table...</a>	Referenced Table <b>USER</b> UserID: INT (PK)
--	--	---

**FEEDBACK - Table Relationship**

Referencing Table <b>RESERVATION</b> Foreign Key: fk_reservation_vehicle VehicleID: INT	Cardinality <input type="radio"/> One-to-One (1:1) <input checked="" type="radio"/> One-to-Many (1:n)  <input type="checkbox"/> Invert Relationship  <input checked="" type="checkbox"/> Mandatory <a href="#">Edit Table...</a>	Referenced Table <b>VEHICLE</b> VehicleID: INT (PK)
--	--	---

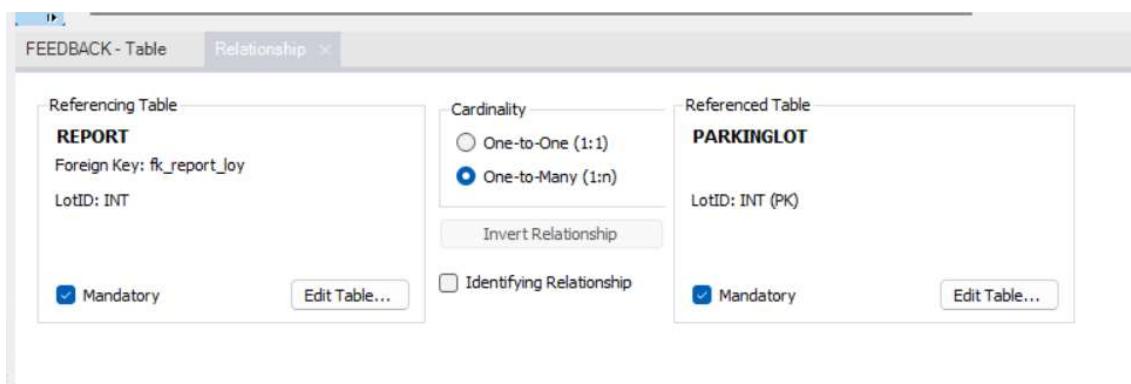
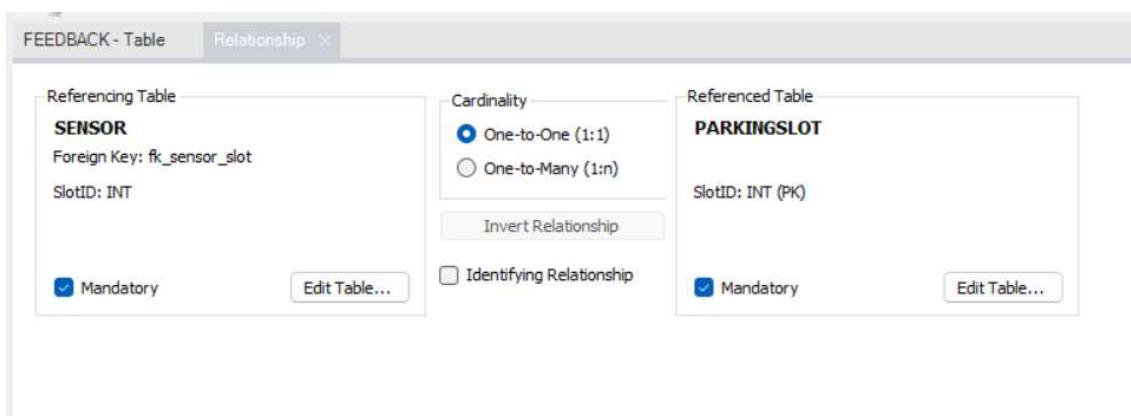
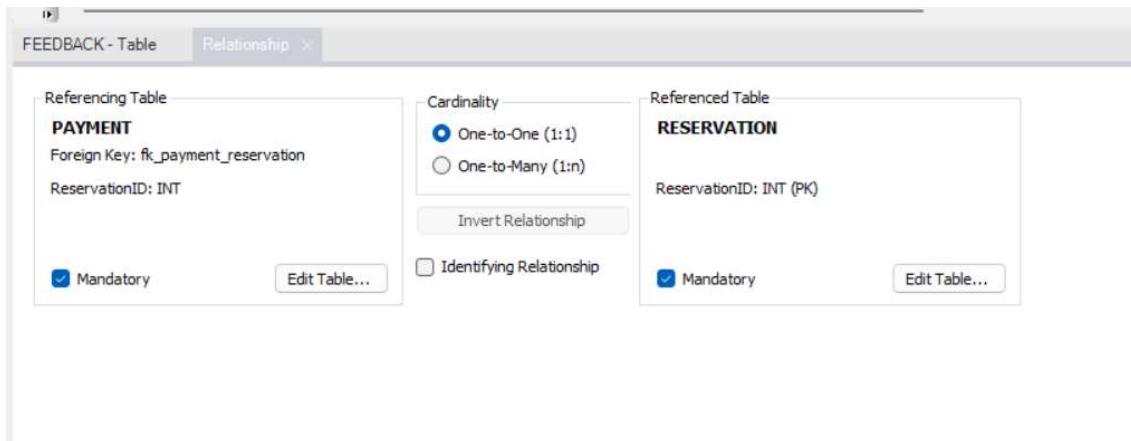
**FEEDBACK - Table Relationship**

Referencing Table <b>FEEDBACK</b> Foreign Key: fk_feedback_lot LotID: INT	Cardinality <input type="radio"/> One-to-One (1:1) <input checked="" type="radio"/> One-to-Many (1:n)  <input type="checkbox"/> Invert Relationship  <input checked="" type="checkbox"/> Mandatory <a href="#">Edit Table...</a>	Referenced Table <b>PARKINGLOT</b> LotID: INT (PK)
--	--	--

**FEEDBACK - Table Relationship**

Referencing Table <b>PARKINGSLOT</b> Foreign Key: fk_parkingslot_lot LotID: INT	Cardinality <input type="radio"/> One-to-One (1:1) <input checked="" type="radio"/> One-to-Many (1:n)  <input type="checkbox"/> Invert Relationship  <input checked="" type="checkbox"/> Mandatory <a href="#">Edit Table...</a>	Referenced Table <b>PARKINGLOT</b> LotID: INT (PK)
--	--	--



## 3.2 Forward Engineering

- Forward Engineering SQL Script

-- MySQL Workbench Forward Engineering

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_
DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBST
ITUTION';
```

---

-- Schema patels184

---

---

-- Schema patels184

---

```
CREATE SCHEMA IF NOT EXISTS `patels184` DEFAULT CHARACTER SET utf8 ;
USE `patels184` ;
```

---

-- Table `patels184`.`USER`

---

```
CREATE TABLE IF NOT EXISTS `patels184`.`USER` (
  `UserID` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(100) NOT NULL,
  `Email` VARCHAR(100) NOT NULL,
  `PhoneNumber` VARCHAR(20) NULL,
  `Password` VARCHAR(255) NOT NULL,
  `UserType` ENUM('Driver', 'Manager', 'Admin') NOT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE INDEX `USERcol_UNIQUE` (`Email` ASC) VISIBLE)
```

```
ENGINE = InnoDB;
```

```
-- Table `patels184`.`VEHICLE`
```

```
CREATE TABLE IF NOT EXISTS `patels184`.`VEHICLE` (
  `VehicleID` INT NOT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `LicensePlate` VARCHAR(20) NOT NULL,
  `VehicleType` VARCHAR(30) NOT NULL,
  `Color` VARCHAR(30) NULL,
  PRIMARY KEY (`VehicleID`),
  UNIQUE INDEX `LicensePlate_UNIQUE` (`LicensePlate` ASC) VISIBLE,
  INDEX `fk_vehicle_user_idx` (`UserID` ASC) VISIBLE,
  CONSTRAINT `fk_vehicle_user`
    FOREIGN KEY (`UserID`)
    REFERENCES `patels184`.`USER` (`UserID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
-- Table `patels184`.`PARKINGLOT`
```

```
CREATE TABLE IF NOT EXISTS `patels184`.`PARKINGLOT` (
  `LotID` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(100) NOT NULL,
  `Location` VARCHAR(255) NOT NULL,
```

```

`TotalSlots` INT NOT NULL,
`ManagerID` INT NOT NULL,
PRIMARY KEY (`LotID`),
INDEX `fk_parkinglot_manager_idx` (`ManagerID` ASC) VISIBLE,
CONSTRAINT `fk_parkinglot_manager`
FOREIGN KEY (`ManagerID`)
REFERENCES `patels184`.`USER` (`UserID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

-- Table `patels184`.`PARKINGSLOT`

```

CREATE TABLE IF NOT EXISTS `patels184`.`PARKINGSLOT` (
`SlotID` INT NOT NULL AUTO_INCREMENT,
`LotID` INT NOT NULL,
`SlotNumber` VARCHAR(20) NOT NULL,
`Status` ENUM('Available', 'Reserved', 'Occupied', 'OutOfService') NOT NULL
DEFAULT 'Available',
`SlotType` ENUM('Compact', 'Large', 'EV', 'Handicapped') NOT NULL,
PRIMARY KEY (`SlotID`),
INDEX `fk_parkingnslot_lot_idx` (`LotID` ASC) VISIBLE,
CONSTRAINT `fk_parkingnslot_lot`
FOREIGN KEY (`LotID`)
REFERENCES `patels184`.`PARKINGLOT` (`LotID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

---

-- Table `patels184`.`RESERVATION`

---

```
CREATE TABLE IF NOT EXISTS `patels184`.`RESERVATION` (
    `ReservationID` INT NOT NULL AUTO_INCREMENT,
    `SlotID` INT NOT NULL,
    `UserID` INT NOT NULL,
    `VehicleID` INT NOT NULL,
    `StartTime` DATETIME NOT NULL,
    `EndTime` DATETIME NOT NULL,
    `Status` ENUM('Active', 'Completed', 'Cancelled') NOT NULL DEFAULT 'Active',
    PRIMARY KEY (`ReservationID`),
    INDEX `fk_reservation_user_idx` (`UserID` ASC) VISIBLE,
    INDEX `fk_reservation_vehicle_idx` (`VehicleID` ASC) VISIBLE,
    CONSTRAINT `fk_reservation_user`
        FOREIGN KEY (`UserID`)
        REFERENCES `patels184`.`USER` (`UserID`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT `fk_reservation_vehicle`
        FOREIGN KEY (`VehicleID`)
        REFERENCES `patels184`.`VEHICLE` (`VehicleID`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

---

-- Table `patels184`.`PAYMENT`

---

```

CREATE TABLE IF NOT EXISTS `patels184`.`PAYMENT` (
    `PaymentID` INT NOT NULL AUTO_INCREMENT,
    `ReservationID` INT NULL,
    `Amount` DECIMAL(10,2) NOT NULL,
    `PaymentMethod` ENUM('Card', 'Wallet', 'Cash') NOT NULL,
    `PaymentStatus` ENUM('Paid', 'Pending', 'Refunded') NOT NULL DEFAULT 'Paid',
    `TransactionDate` DATETIME NOT NULL,
    PRIMARY KEY (`PaymentID`),
    UNIQUE INDEX `ReservationID_UNIQUE` (`ReservationID` ASC) VISIBLE,
    CONSTRAINT `fk_payment_reservation`
        FOREIGN KEY (`ReservationID`)
            REFERENCES `patels184`.`RESERVATION` (`ReservationID`)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

---

```
-- Table `patels184`.`SENSOR`
```

---

```

CREATE TABLE IF NOT EXISTS `patels184`.`SENSOR` (
    `SensorID` INT NOT NULL AUTO_INCREMENT,
    `SlotID` INT NOT NULL,
    `Status` ENUM('Free', 'Occupied') NOT NULL DEFAULT 'Free',
    `LastUpdate` DATETIME NULL,
    PRIMARY KEY (`SensorID`),
    UNIQUE INDEX `SlotID_UNIQUE` (`SlotID` ASC) VISIBLE,
    CONSTRAINT `fk_sensor_slot`
        FOREIGN KEY (`SlotID`)

```

```
REFERENCES `patels184`.`PARKINGSLOT` ('SlotID')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

---

```
-- Table `patels184`.`FEEDBACK`
```

---

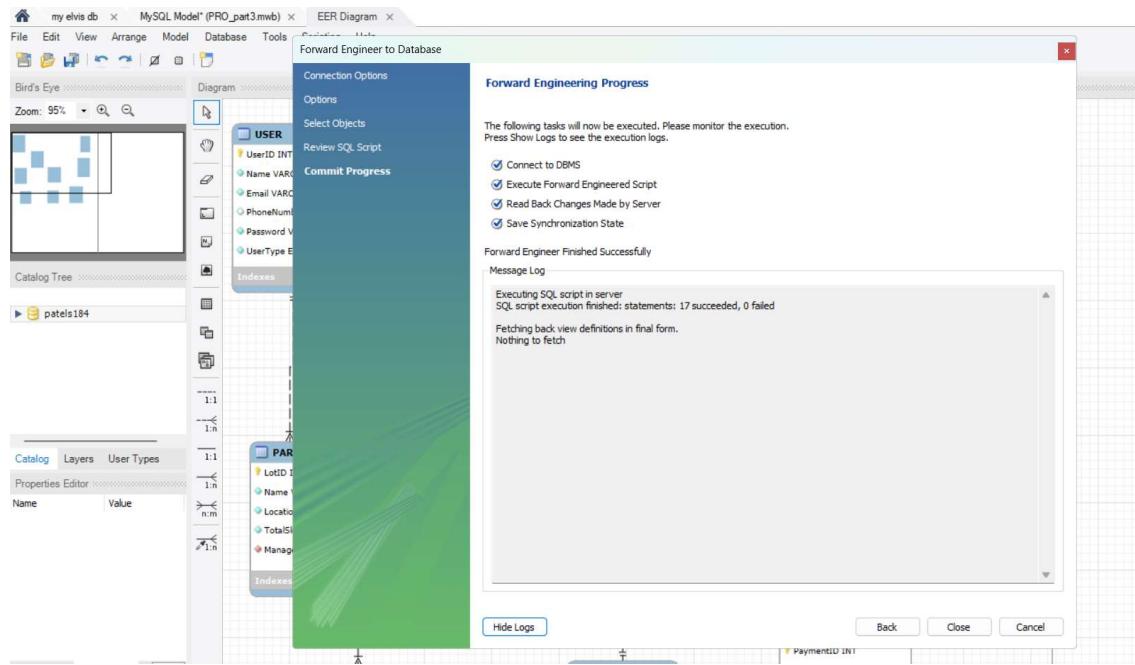
```
CREATE TABLE IF NOT EXISTS `patels184`.`FEEDBACK` (
  `FeedbackID` INT NOT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `LotID` INT NOT NULL,
  `Rating` TINYINT NOT NULL,
  `Comment` VARCHAR(500) NULL,
  `FeedbackDate` DATE NOT NULL,
  PRIMARY KEY (`FeedbackID`),
  INDEX `fk_feedback_lot_idx` (`LotID` ASC) VISIBLE,
  INDEX `fk_feedback_user_idx` (`UserID` ASC) VISIBLE,
  CONSTRAINT `fk_feedback_lot`
    FOREIGN KEY (`LotID`)
      REFERENCES `patels184`.`PARKINGLOT` (`LotID`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_feedback_user`
    FOREIGN KEY (`UserID`)
      REFERENCES `patels184`.`USER` (`UserID`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- Table `patels184`.`REPORT`

CREATE TABLE IF NOT EXISTS `patels184`.`REPORT` (
    `ReportID` INT NOT NULL AUTO_INCREMENT,
    `LotID` INT NOT NULL,
    `ReportType` ENUM('Usage', 'Revenue', 'Violations') NOT NULL,
    `GeneratedDate` DATE NOT NULL,
    PRIMARY KEY (`ReportID`),
    INDEX `fk_report_loy_idx` (`LotID` ASC) VISIBLE,
    CONSTRAINT `fk_report_loy`
        FOREIGN KEY (`LotID`)
        REFERENCES `patels184`.`PARKINGLOT` (`LotID`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

- Screenshot – Execution in Elvis



### 3.3 Data Population

- Screenshot – USER table (5 records)

The screenshot shows the MySQL Workbench interface. On the left, the database browser displays the schema for the 'patels184' database, including tables like FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, VEHICLE, and views like sakila, sampdb, and SPATIAL\_DB. The 'USER' table is selected.

In the center, a result grid shows the data for the 'USER' table:

	UserID	Name	Email	PhoneNumber	Password	UserType
1	Alice	Alice Driver	alice@example.com	555-111-1111	pwd1	Driver
2	Bob	Bob Manager	bob@example.com	555-222-2222	pwd2	Manager
3	Chris	Chris Admin	chris@example.com	555-333-3333	pwd3	Admin
4	David	David Driver	david@example.com	555-444-4444	pwd4	Driver
5	Eva	Eva Driver	eva@example.com	555-555-5555	pwd5	Driver
*	HULL	HULL	HULL	HULL	HULL	HULL

Below the grid, the "Output" tab shows the history of actions taken:

#	Time	Action	Message
58	20:31:21	INSERT INTO FEEDBACK (UserID, LotID, Rating, Comment, FeedbackDate) VALUES (1, 1, 5, 'Great service!', '2025-11-15')	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
59	20:31:21	INSERT INTO REPORT (LotID, ReportType, GeneratedDate) VALUES (1, 'Usage', '2025-11-15')	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
60	20:31:21	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected
61	20:31:45	select * from USER LIMIT 0, 1000	5 row(s) returned

- Screenshot – VEHICLE table (5 records)

4

5 • `select * from VEHICLE;`

The screenshot shows the MySQL Workbench interface. On the left, the database tree shows 'nwwind', 'patels184' (selected), 'Tables' containing 'FEEDBACK', 'PARKINGLOT', 'PARKINGSLOT', 'PAYMENT', 'REPORT', 'RESERVATION', 'SENSOR', 'USER', and 'VEHICLE', and other schemas like 'sakila', 'sampdb', and 'SPATIAL\_DB'. Below the tree are tabs for 'Administration' and 'Schemas'. The main area shows the 'VEHICLE' table results in a grid:

VehicleID	UserID	LicensePlate	VehicleType	Color
1	1	ABC1111	Car	Red
2	1	XYZ2222	SUV	Black
3	4	DRV3333	Car	Blue
4	5	EVA4444	Car	White
5	5	EV55555	Truck	Silver
NULL	NULL	NULL	NULL	NULL

Below the grid, there's an 'Information' section for the 'USER' table, showing columns: UserID (int AI PK), Name (varchar(100)), Email (varchar(100)), PhoneNumber (varchar(20)), Password (varchar(255)), and UserType (enum('Driver', 'Manager', 'Admin')). The 'Object Info' and 'Session' tabs are also visible.

USER 2    VEHICLE 3    X

Output

Action Output

#	Time	Action	Message
60	20:31:21	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected
61	20:31:45	select * from USER LIMIT 0, 1000	5 row(s) returned
62	20:32:51	select * from USER LIMIT 0, 1000	5 row(s) returned
63	20:32:52	select * from VEHICLE LIMIT 0, 1000	5 row(s) returned

• Screenshot – PARKINGLOT table (5 records)

8 • `select * from PARKINGLOT;`

The screenshot shows the MySQL Workbench interface. On the left, the database tree shows 'northwind', 'nwwind' (selected), 'patels184' (selected), 'Tables' containing 'FEEDBACK', 'PARKINGLOT', 'PARKINGSLOT', 'PAYMENT', 'REPORT', and 'DESCRIPTION', and other schemas like 'sakila', 'sampdb', and 'SPATIAL\_DB'. Below the tree are tabs for 'Administration' and 'Schemas'. The main area shows the 'PARKINGLOT' table results in a grid:

LotID	Name	Location	TotalSlots	ManagerID
1	Downtown Garage	123 Main St	10	2
2	Mall Parking	45 Shopping Ave	8	2
3	Airport Lot A	Airport Road	12	2
4	Stadium Lot	Arena Blvd	15	2
5	Business Center Lot	Corporate Plaza	6	2
NULL	NULL	NULL	NULL	NULL

Below the grid, there's an 'Information' section for the 'PARKINGLOT' table, showing columns: LotID (int AI PK), Name (varchar(45)), Location (varchar(45)), TotalSlots (int), and ManagerID (int). The 'Object Info' and 'Session' tabs are also visible.

USER 4    VEHICLE 5    PARKINGLOT 6    X

Output

Action Output

#	Time	Action	Message
65	20:34:01	select * from VEHICLE LIMIT 0, 1000	5 row(s) returned
66	20:34:02	select * from PARKINGLOT LIMIT 0, 1000	5 row(s) returned

• Screenshot – PARKINGSLOT table (5 records)

8 • `select * from PARKINGSLOT;`

The screenshot shows the MySQL Workbench interface with the database 'patels184' selected. The 'Tables' section lists FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, and RESERVATION. The 'Connection Details' and 'Server' sections provide connection information and server details. The main area displays the results of the query `select * from PARKINGSLOT;`. The Result Grid shows the following data:

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	Available	Compact
2	1	A2	Available	Compact
3	1	A3	Reserved	EV
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
*	NULL	NULL	NULL	NULL

The Action Output pane shows two log entries:

- # 68 20:34:50 select \* from VEHICLE LIMIT 0, 1000 5 row(s) returned
- # 69 20:34:50 select \* from PARKINGSLOT LIMIT 0, 1000 5 row(s) returned

- Screenshot – RESERVATION table (5 records)

8 • `select * from RESERVATION;`

The screenshot shows the MySQL Workbench interface with the database 'patels184' selected. The 'Tables' section lists FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, and RESERVATION. The 'Connection Details' and 'Server' sections provide connection information and server details. The main area displays the results of the query `select * from RESERVATION;`. The Result Grid shows the following data:

ReservationID	SlotID	UserID	VehicleID	StartTime	EndTime	Status
1	1	1	1	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
2	2	1	2	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
3	3	4	3	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
4	4	5	4	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
5	5	5	5	2025-11-18 18:00:00	2025-11-18 20:00:00	Active
*	NULL	NULL	NULL	NULL	NULL	NULL

The Action Output pane shows two log entries:

- # 71 20:35:27 select \* from VEHICLE LIMIT 0, 1000 5 row(s) returned
- # 72 20:35:27 select \* from RESERVATION LIMIT 0, 1000 5 row(s) returned

- Screenshot – PAYMENT table (5 records)

8 • `select * from PAYMENT;`

PaymentID	ReservationID	Amount	PaymentMethod	PaymentStatus	TransactionDate
1	1	12.00	Card	Paid	2025-11-15 10:05:00
2	2	18.00	Wallet	Paid	2025-11-16 09:05:00
3	3	10.00	Card	Paid	2025-11-15 12:05:00
4	4	0.00	Card	Refunded	2025-11-17 15:05:00
5	5	15.00	Cash	Pending	2025-11-18 18:05:00
*	HULL	HULL	HULL	HULL	HULL

USER 19 VEHICLE 20 PAYMENT 21 X Apply

Output

Action Output

#	Time	Action	Message
✓	80 20:36:49	select * from VEHICLE LIMIT 0, 1000	5 row(s) returned
✓	81 20:36:49	select * from PAYMENT LIMIT 0, 1000	5 row(s) returned

• Screenshot – SENSOR table (5 records)

8 • `select * from SENSOR;`

SensorID	SlotID	Status	LastUpdate
1	1	Free	2025-11-15 10:10:00
2	2	Occupied	2025-11-16 09:10:00
3	3	Free	2025-11-15 14:10:00
4	4	Free	2025-11-17 17:10:00
5	5	Occupied	2025-11-18 18:10:00
*	HULL	HULL	HULL

USER 16 VEHICLE 17 SENSOR 18 X Apply

Output

Action Output

#	Time	Action	Message
✓	77 20:36:22	select * from VEHICLE LIMIT 0, 1000	5 row(s) returned
✓	78 20:36:22	select * from SENSOR LIMIT 0, 1000	5 row(s) returned

• Screenshot – FEEDBACK table (5 records)

8 • `select * from FEEDBACK;`

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure with 'patels184' selected. The 'Tables' node under 'patels184' contains 'FEEDBACK', 'PARKINGLOT', 'PARKINGSLOT', 'PAYMENT', and 'REPORT'. The 'Connection Details' and 'Server' sections provide connection information. The main area shows the results of the query `select * from FEEDBACK;`. The Result Grid displays 5 rows of data:

FeedbackID	UserID	LotID	Rating	Comment	FeedbackDate
1	1	1	5	Great experience!	2025-11-15
2	1	2	4	Mall parking was ok	2025-11-16
3	4	1	3	Slots too full	2025-11-15
4	5	1	5	Loved the EV charger	2025-11-18
5	5	3	4	Airport lot was convenient	2025-11-18

The Output pane shows two log entries:

- Action Output: # 74 Time 20:35:50 Action select \* from VEHICLE LIMIT 0, 1000 Message 5 row(s) returned
- Action Output: # 75 Time 20:35:50 Action select \* from FEEDBACK LIMIT 0, 1000 Message 5 row(s) returned

• Screenshot – REPORT table (5 records)

8 • `select * from REPORT;`

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure with 'patels184' selected. The 'Tables' node under 'patels184' contains 'FEEDBACK', 'PARKINGLOT', 'PARKINGSLOT', 'PAYMENT', and 'REPORT'. The 'Connection Details' and 'Server' sections provide connection information. The main area shows the results of the query `select * from REPORT;`. The Result Grid displays 5 rows of data:

ReportID	LotID	ReportType	GeneratedDate
1	1	Usage	2025-11-15
2	1	Revenue	2025-11-15
3	2	Usage	2025-11-16
4	3	Revenue	2025-11-18
5	4	Violations	2025-11-15

The Output pane shows two log entries:

- Action Output: # 83 Time 20:37:14 Action select \* from VEHICLE LIMIT 0, 1000 Message 5 row(s) returned
- Action Output: # 84 Time 20:37:14 Action select \* from REPORT LIMIT 0, 1000 Message 5 row(s) returned

## 3.4 SQL JOIN Queries

- JOIN Query 1

SELECT

```
r.ReservationID,
u.Name AS UserName,
v.LicensePlate,
r.StartTime,
r.EndTime,
r.Status

FROM RESERVATION r
JOIN `USER` u ON r.UserID = u.UserID
JOIN VEHICLE v ON r.VehicleID = v.VehicleID;
```

The screenshot shows the MySQL Workbench interface. On the left, the Object Navigator displays a tree view of databases and tables, including 'pate184' which contains 'Tables' like FEEDBACK, PARKINGLOT, etc. Below the object navigator are tabs for 'Administration' and 'Schemas'. On the right, the main area shows the query editor with the following code:

```
1 • SELECT
2     r.ReservationID,
3     u.Name AS UserName,
4     v.LicensePlate,
5     r.StartTime,
6     r.EndTime,
7     r.Status
8 FROM RESERVATION r
9 JOIN `USER` u ON r.UserID = u.UserID
10 JOIN VEHICLE v ON r.VehicleID = v.VehicleID;
11
```

Below the query editor is a 'Result Grid' table with the following data:

	ReservationID	UserName	LicensePlate	StartTime	EndTime	Status
▶	1	Alice Driver	ABC1111	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
▶	2	Alice Driver	XYZ2222	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
▶	3	David Driver	DRV3333	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
▶	4	Eva Driver	EVA4444	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
▶	5	Eva Driver	EV55555	2025-11-18 18:00:00	2025-11-18 20:00:00	Active

At the bottom, the 'Output' tab shows the execution details:

#	Time	Action	Message
84	20:37:14	select * from REPORT LIMIT 0, 1000	5 row(s) returned
85	20:54:42	SELECT r.ReservationID, u.Name AS UserName, v.LicensePlate, r.StartTime, r.End...	5 row(s) returned

- JOIN Query 2

SELECT

```

pl.Name AS ParkingLot,
ps.SlotNumber,
ps.SlotType,
ps.Status AS SlotStatus,
s.Status AS SensorStatus,
s.LastUpdate

FROM PARKINGSLOT ps
JOIN PARKINGLOT pl ON ps.LotID = pl.LotID
LEFT JOIN SENSOR s ON ps.SlotID = s.SlotID;

```

The screenshot shows the SSMS interface with the following details:

- Object Explorer (Left):** Shows the database structure under the 'patels184' schema, including tables like FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, and USER.
- Query Editor (Right):**
  - Query 1:** Displays the T-SQL code for the JOIN query.
  - Result Grid:** Shows the output of the query, mapping parking slots to their respective parking lots and sensor status. The columns are: ParkingLot, SlotNumber, SlotType, SlotStatus, SensorStatus, and LastUpdate.
  - Output Window:** Shows the execution log with two entries. Entry 85 shows the SELECT statement being run at 20:54:42. Entry 86 shows the results being returned at 20:56:36, indicating 5 rows were returned.

ParkingLot	SlotNumber	SlotType	SlotStatus	SensorStatus	LastUpdate
Downtown Garage	A1	Compact	Available	Free	2025-11-15 10:10:00
Downtown Garage	A2	Compact	Available	Occupied	2025-11-16 09:10:00
Downtown Garage	A3	EV	Reserved	Free	2025-11-15 14:10:00
Downtown Garage	A4	Large	Occupied	Free	2025-11-17 17:10:00
Downtown Garage	A5	Handicapped	OutOfService	Occupied	2025-11-18 18:10:00

### 3.5 Database Objects

- Views (2) – Code + Results

**VIEW 1:**

CREATE VIEW vw\_reservation\_summary AS

SELECT

```
r.ReservationID,
u.Name AS UserName,
v.LicensePlate,
```

```
r.StartTime,
r.EndTime,
```

r.Status

FROM RESERVATION r

JOIN `USER` u ON r.UserID = u.UserID

JOIN VEHICLE v ON r.VehicleID = v.VehicleID;

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is set to 'sakila'.
- Tables:** A tree view of tables including FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, and VEHICLE.
- Views:** A list of views including 'vw\_reservation\_summary'.
- Stored Procedures:** A list of stored procedures.
- Functions:** A list of functions.
- Information:** A section showing connection details and server information.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Action Output:** Shows the execution history of the CREATE VIEW command and its SELECT statement.
- Output:** Shows the results of the SELECT query, which returned 5 rows.

```

45
46 • CREATE VIEW vw_reservation_summary AS
47   SELECT
48     r.ReservationID,
49     u.Name AS UserName,
50     v.LicensePlate,
51     r.StartTime,
52     r.EndTime,
53     r.Status
54   FROM RESERVATION r
55   JOIN `USER` u ON r.UserID = u.UserID
56   JOIN VEHICLE v ON r.VehicleID = v.VehicleID;
57

```

#	Time	Action	Message
84	20:37:14	select * from REPORT LIMIT 0, 1000	5 row(s) returned
85	20:54:42	SELECT r.ReservationID, u.Name AS UserName, v.LicensePlate, r.StartTime, r.End...	5 row(s) returned
86	20:56:36	SELECT pl.Name AS ParkingLot, ps.SlotNumber, ps.SlotType, ps.Status AS SlotStatus...	5 row(s) returned
87	21:33:48	CREATE VIEW vw_reservation_summary AS SELECT r.ReservationID, u.Name AS UserNa...	0 row(s) affected

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree displays the 'patels184' schema with tables like FEEDBACK, PARKINGLOT, PAYMENT, REPORT, RESERVATION, SENSOR, and USER. Below the schemas tree are sections for 'Administration' and 'Information'. Under 'Information', there are two tabs: 'Connection Details' and 'Server'. 'Connection Details' shows the connection name 'pro', host 'elvisdb.rowan.edu', port '3306', and SSL cipher 'TLS\_AES\_128\_GCM\_SHA256'. The 'Server' tab shows the product 'Source distribution' and version '8.0.43'. On the right, the main workspace shows the creation of a view:

```

61
62
63 •  SELECT * FROM vw_reservation_summary;
64
65

```

The 'Result Grid' shows the data returned by the query:

ReservationID	UserName	LicensePlate	StartTime	EndTime	Status
1	Alice Driver	ABC1111	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
2	Alice Driver	XYZ2222	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
3	David Driver	DRV3333	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
4	Eva Driver	EVA4444	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
5	Eva Driver	EV55555	2025-11-18 18:00:00	2025-11-18 20:00:00	Active

The 'Output' pane at the bottom shows the log of actions taken:

#	Time	Action	Message
85	20:54:42	SELECT r.ReservationID, u.Name AS UserName, v.LicensePlate, r.StartTime, r.EndTime, ps.Status AS Status	5 row(s) returned
86	20:56:36	SELECT pl.Name AS ParkingLot, ps.SlotNumber, ps.SlotType, ps.Status AS SlotStatus	5 row(s) returned
87	21:33:48	CREATE VIEW vw_reservation_summary AS SELECT r.ReservationID, u.Name AS UserName, v.LicensePlate, r.StartTime, r.EndTime, ps.Status AS Status, pl.Name AS ParkingLot, ps.SlotNumber, ps.SlotType, ps.Status AS SlotStatus	0 row(s) affected
88	21:36:05	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	5 row(s) returned

## VIEW 2:

```

CREATE VIEW vw_lot_usage AS
SELECT
    pl.Name AS ParkingLot,
    pl.TotalSlots,
    COUNT(ps.SlotID) AS SlotsConfigured,
    SUM(ps.Status = 'Occupied') AS OccupiedSlots,
    SUM(ps.Status = 'Available') AS AvailableSlots
FROM PARKINGLOT pl
JOIN PARKINGSLOT ps ON pl.LotID = ps.LotID
GROUP BY pl.LotID;

```

The screenshot shows the MySQL Workbench interface. On the left, the database schema for 'patels184' is visible, containing tables like FEEDBACK, PARKINGLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, VEHICLE, and views like sakila, sampdb, and SPATIAL\_DB. The 'Information' tab is selected.

**Connection Details:**

- Name: pro
- Host: elvisdb.rowan.edu
- Port: 3306
- Login User: patels184
- Current User: patels184@%
- SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server:**

- Product: Source distribution
- Version: 8.0.43

**Object Info** | **Session**

Query Completed

**Code (SQL):**

```

64
65
66
67 • CREATE VIEW vw_lot_usage AS
68   SELECT
69     pl.Name AS ParkingLot,
70     pl.TotalSlots,
71     COUNT(ps.SlotID) AS SlotsConfigured,
72     SUM(ps.Status = 'Occupied') AS OccupiedSlots,
73     SUM(ps.Status = 'Available') AS AvailableSlots
74   FROM PARKINGLOT pl
75   JOIN PARKINGSLOT ps ON pl.LotID = ps.LotID
76   GROUP BY pl.LotID;
77

```

**Action Output:**

#	Time	Action	Message
86	20:56:36	SELECT pl.Name AS ParkingLot, ps.SlotNumber, ps.SlotType, ps.Status AS SlotStatus...	5 row(s) returned
87	21:33:48	CREATE VIEW vw_reservation_summary AS SELECT r.ReservationID, u.Name AS UserName...	0 row(s) affected
88	21:36:05	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	5 row(s) returned
89	21:38:52	CREATE VIEW vw_lot_usage AS SELECT pl.Name AS ParkingLot, pl.TotalSlots, COUNT...	0 row(s) affected

The screenshot shows the MySQL Workbench interface. The schema for 'patels184' is visible on the left.

**Connection Details:**

- Name: pro
- Host: elvisdb.rowan.edu
- Port: 3306
- Login User: patels184
- Current User: patels184@%
- SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server:**

- Product: Source distribution
- Version: 8.0.43

**Object Info** | **Session**

Query Completed

**Code (SQL):**

```

79
80
81
82 • SELECT * FROM vw_lot_usage;
83

```

**Result Grid:**

ParkingLot	TotalSlots	SlotsConfigured	OccupiedSlots	AvailableSlots
Downtown Garage	10	5	1	2

**vw\_lot\_usage 28 x**

**Action Output:**

#	Time	Action	Message
87	21:33:48	CREATE VIEW vw_reservation_summary AS SELECT r.ReservationID, u.Name AS UserName...	0 row(s) affected
88	21:36:05	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	5 row(s) returned
89	21:38:52	CREATE VIEW vw_lot_usage AS SELECT pl.Name AS ParkingLot, pl.TotalSlots, COUNT...	0 row(s) affected
90	21:40:35	SELECT * FROM vw_lot_usage LIMIT 0, 1000	1 row(s) returned

### • Functions (2) – Code + Output

#### FUNCTION 1:

DELIMITER //

CREATE FUNCTION fn\_reservation\_payment(resID INT)

RETURNS DECIMAL(10,2)

## DETERMINISTIC

BEGIN

```
DECLARE amt DECIMAL(10,2);
```

```
SELECT Amount INTO amt
```

```
FROM PAYMENT
```

```
WHERE ReservationID = resID;
```

```
RETURN amt;
```

END//

DELIMITER ;

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is 'patels184'.
- Tables:** Shows tables like FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, and USER.
- Functions:** Shows functions like VEHICLE, Views, and Stored Procedures.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Object Info:** Shows the creation of a function.
- Session:** Displays the SQL code for creating the function:

```

88
89  DELIMITER //
90 • CREATE FUNCTION fn_reservation_payment(resID INT)
91   RETURNS DECIMAL(10,2)
92   DETERMINISTIC
93 BEGIN
94   DECLARE amt DECIMAL(10,2);
95   SELECT Amount INTO amt
96   FROM PAYMENT
97   WHERE ReservationID = resID;
98   RETURN amt;
99 END//|
100 DELIMITER ;
101

```

**Action Output:**

#	Time	Action	Message
88	21:36:05	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	5 row(s) returned
89	21:38:52	CREATE VIEW vw_lot_usage AS SELECT pl.Name AS ParkingLot, pl.TotalSlots, COUNT(*) AS UsedSlots, (COUNT(*) / pl.TotalSlots) * 100 AS Percentage FROM vw_reservation_summary AS rs JOIN vw_parkinglot AS pl ON rs.ParkingLotID = pl.ID GROUP BY pl.Name	0 row(s) affected
90	21:40:35	SELECT * FROM vw_lot_usage LIMIT 0, 1000	1 row(s) returned
91	21:43:16	CREATE FUNCTION fn_reservation_payment(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN     DECLARE amt DECIMAL(10,2);     SELECT Amount INTO amt     FROM PAYMENT     WHERE ReservationID = resID;     RETURN amt; END//	0 row(s) affected

Query Completed

The screenshot shows the MySQL Workbench interface. On the left, the schema browser displays tables like PARKINGLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, and VEHICLE, along with Views, Stored Procedures, and Functions. Below the schema browser, the 'Information' tab is selected. Under 'Connection Details', it shows the connection to 'pro' at 'elvisdb.rowan.edu' on port 3306, with the current user being 'patels184'. Under 'Server', it shows the product as 'Source distribution' and version as '8.0.43'. The 'Object Info' tab is also visible.

In the main area, a query editor window is open with the following code:

```

104
105
106 • SELECT fn_reservation_payment(3);
107
108

```

The result grid shows the output of the query:

fn_reservation_payment(3)
10.00

Below the result grid, the 'Result 29' tab is active, showing the action history:

#	Time	Action	Message
89	21:38:52	CREATE VIEW vw_lot_usage AS SELECT pl.Name AS ParkingLot, pl.TotalSlots, COUNT(*) AS UsedSlots, (COUNT(*) / pl.TotalSlots) * 100 AS UsagePercentage FROM Reservation r JOIN ParkingLot pl ON r.ParkingLotID = pl.ID GROUP BY pl.ID	0 row(s) affected
90	21:40:35	SELECT * FROM vw_lot_usage LIMIT 0, 1000	1 row(s) returned
91	21:43:16	CREATE FUNCTION fn_reservation_payment(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN     DECLARE hrs DECIMAL(10,2);     SELECT TIMESTAMPDIFF(MINUTE, StartTime, EndTime)/60 INTO hrs     FROM RESERVATION     WHERE ReservationID = resID;     RETURN hrs; END//	0 row(s) affected
92	21:44:39	SELECT fn_reservation_payment(3) LIMIT 0, 1000	1 row(s) returned

## FUNCTION 2:

DELIMITER //

```

CREATE FUNCTION fn_reservation_hours(resID INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE hrs DECIMAL(10,2);
    SELECT TIMESTAMPDIFF(MINUTE, StartTime, EndTime)/60 INTO hrs
    FROM RESERVATION
    WHERE ReservationID = resID;
    RETURN hrs;
END//
DELIMITER ;

```

The screenshot shows the MySQL Workbench interface with two main panes. The left pane displays the database schema for the 'patels184' database, listing tables like FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, VEHICLE, and views/functions. The right pane shows the SQL editor and results.

**Code Editor (Top Right):**

```

108
109
110  DELIMITER //
111 • CREATE FUNCTION fn_reservation_hours(resID INT)
112   RETURNS DECIMAL(10,2)
113   DETERMINISTIC
114 BEGIN
115   DECLARE hrs DECIMAL(10,2);
116   SELECT TIMESTAMPDIFF(MINUTE, StartTime, EndTime)/60 INTO hrs
117   FROM RESERVATION
118   WHERE ReservationID = resID;
119   RETURN hrs;
120 END//
121 DELIMITER ;
122
123

```

**Output (Top Right):**

#	Time	Action	Message
90	21:40:35	SELECT * FROM vw_lot_usage LIMIT 0, 1000	1 row(s) returned
91	21:43:16	CREATE FUNCTION fn_reservation_payment(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC	0 row(s) affected
92	21:44:39	SELECT fn_reservation_payment(3) LIMIT 0, 1000	1 row(s) returned
93	21:45:38	CREATE FUNCTION fn_reservation_hours(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC	0 row(s) affected

**Code Editor (Bottom Left):**

```

122
123
124 • SELECT fn_reservation_hours(1);

```

**Result Grid (Bottom Left):**

fn_reservation_hours(1)
2.00

**Output (Bottom Right):**

#	Time	Action	Message
91	21:43:16	CREATE FUNCTION fn_reservation_payment(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC	0 row(s) affected
92	21:44:39	SELECT fn_reservation_payment(3) LIMIT 0, 1000	1 row(s) returned
93	21:45:38	CREATE FUNCTION fn_reservation_hours(resID INT) RETURNS DECIMAL(10,2) DETERMINISTIC	0 row(s) affected
94	21:46:31	SELECT fn_reservation_hours(1) LIMIT 0, 1000	1 row(s) returned

### • Stored Procedures (2) – Code + Execution

#### STORED PROCEDURE 1:

DELIMITER //

CREATE PROCEDURE sp\_create\_reservation\_with\_payment(

IN pSlotID INT,

IN pUserID INT,

```
IN pVehicleID INT,  
IN pStart DATETIME,  
IN pEnd DATETIME,  
IN pAmount DECIMAL(10,2)  
)  
BEGIN  
DECLARE newID INT;  
START TRANSACTION;  
  
INSERT INTO RESERVATION (SlotID, UserID, VehicleID, StartTime,  
EndTime, Status)  
VALUES (pSlotID, pUserID, pVehicleID, pStart, pEnd, 'Active');  
  
SET newID = LAST_INSERT_ID();  
  
INSERT INTO PAYMENT (ReservationID, Amount, PaymentMethod,  
PaymentStatus, TransactionDate)  
VALUES (newID, pAmount, 'Card', 'Pending', NOW());  
  
COMMIT;  
END//  
DELIMITER ;
```

**Object Explorer:**

- Adventureworks
- Chinook
- Murach
- Myersjac
- Northwind
- Nwind
- patels184**
  - Tables: FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, VEHICLE
  - Views
  - Stored Procedures
  - Functions
- sakila
- sampdb
- SPATIAL\_DB

**Connection Details:**

- Name: pro
- Host: elvisdb.rowan.edu
- Port: 3306
- Login User: patels184
- Current User: patels184@%
- SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server:**

- Product: Source distribution
- Version: 8.0.43

**Action Output:**

```

127      -- stored procedure
128      DELIMITER //
129
130  • CREATE PROCEDURE sp_create_reservation_with_payment(
131      IN pSlotID INT,
132      IN pUserID INT,
133      IN pVehicleID INT,
134      IN pStart DATETIME,
135      IN pEnd DATETIME,
136      IN pAmount DECIMAL(10,2)
137  )
138
139  BEGIN
140      DECLARE newID INT;
141      START TRANSACTION;
142
143      INSERT INTO RESERVATION (SlotID, UserID, VehicleID, StartTime, EndTime, Status)
144      VALUES (pSlotID, pUserID, pVehicleID, pStart, pEnd, 'Active');
145
146      SET newID = LAST_INSERT_ID();
147
148      INSERT INTO PAYMENT (ReservationID, Amount, PaymentMethod, PaymentStatus, TransactionDate)
149      VALUES (newID, pAmount, 'Card', 'Pending', NOW());
150
151      COMMIT;
152
153  END//
154
155  DELIMITER ;

```

**Output:**

#	Time	Action	Message
92	21:44:39	SELECT fn_reservation_payment(3) LIMIT 0, 1000	1 row(s) returned
93	21:45:38	CREATE FUNCTION fn_reservation_hours(resID INT) RETURNS DECIMAL(10,2) DETERMINIS...	0 row(s) affected
94	21:46:31	SELECT fn_reservation_hours(1) LIMIT 0, 1000	1 row(s) returned
95	21:49:07	CREATE PROCEDURE sp_create_reservation_with_payment( IN pSlotID INT, IN pUserD ...	0 row(s) affected

**Result Grid:**

ReservationID	SlotID	UserID	VehicleID	StartTime	EndTime	Status
1	1	1	1	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
2	2	1	2	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
3	3	4	3	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
4	4	5	4	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
5	5	5	5	2025-11-18 18:00:00	2025-11-18 20:00:00	Active
6	1	1	1	2025-11-17 02:52:08	2025-11-17 03:52:08	Active
NULL	NULL	NULL	NULL	NULL	NULL	NULL

**RESERVATION 31:**

**Action Output:**

```

154 • CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW(), INTERVAL 1 HOUR), 10.00);
155 • SELECT * FROM RESERVATION;
156 • SELECT * FROM PAYMENT;

```

**Result Grid:**

PaymentID	ReservationID	Amount	PaymentMethod	PaymentStatus	TransactionDate
1	1	12.00	Card	Paid	2025-11-15 10:05:00
2	2	18.00	Wallet	Paid	2025-11-16 09:05:00
3	3	10.00	Card	Paid	2025-11-15 12:05:00
4	4	0.00	Card	Refunded	2025-11-17 15:05:00
5	5	15.00	Cash	Pending	2025-11-18 18:05:00
6	6	10.00	Card	Pending	2025-11-17 02:52:08
NULL	NULL	NULL	NULL	NULL	NULL

**PAYMENT 32:**

**Action Output:**

```

154 • CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW(), INTERVAL 1 HOUR), 10.00);
155 • SELECT * FROM RESERVATION;
156 • SELECT * FROM PAYMENT;

```

**Result Grid:**

PaymentID	ReservationID	Amount	PaymentMethod	PaymentStatus	TransactionDate
1	1	12.00	Card	Paid	2025-11-15 10:05:00
2	2	18.00	Wallet	Paid	2025-11-16 09:05:00
3	3	10.00	Card	Paid	2025-11-15 12:05:00
4	4	0.00	Card	Refunded	2025-11-17 15:05:00
5	5	15.00	Cash	Pending	2025-11-18 18:05:00
6	6	10.00	Card	Pending	2025-11-17 02:52:08
NULL	NULL	NULL	NULL	NULL	NULL

**Action Output:**

```

95 21:49:07 CREATE PROCEDURE sp_create_reservation_with_payment( IN pSlotID INT, IN pUserD ...
96 21:52:08 CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW()), INTERVAL 1 H...
97 21:52:13 SELECT * FROM RESERVATION LIMIT 0, 1000

```

**Output:**

#	Time	Action	Message
95	21:49:07	CREATE PROCEDURE sp_create_reservation_with_payment( IN pSlotID INT, IN pUserD ...	0 row(s) affected
96	21:52:08	CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW()), INTERVAL 1 H...	0 row(s) affected
97	21:52:13	SELECT * FROM RESERVATION LIMIT 0, 1000	6 row(s) returned
98	21:53:04	SELECT * FROM PAYMENT LIMIT 0, 1000	6 row(s) returned

## STORED PROCEDURE 2:

DELIMITER //

```
CREATE PROCEDURE sp_get_payments()
```

BEGIN

```
    SELECT PaymentID, ReservationID, Amount, PaymentStatus
```

```
    FROM PAYMENT;
```

END//

DELIMITER ;

```

Tables
  FEEDBACK
  PARKINGLOT
  PARKINGSLOT
  PAYMENT
  REPORT
  RESERVATION
  SENSOR
  USER
  VEHICLE
Views
Stored Procedures
Functions
sakila
sampdb
SPATIAL_DB

Administration Schemas Information

Connection Details
Name: pro
Host: elvisdb.rowan.edu
Port: 3306
Login User: patels184
Current User: patels184@%
SSL cipher: TLS_AES_128_GCM_SHA256

Server
Product: Source distribution
Version: 8.0.43

Object Info Session
Query Completed

157
158
159  DELIMITER //
160 • CREATE PROCEDURE sp_get_payments()
161 BEGIN
162     SELECT PaymentID, ReservationID, Amount, PaymentStatus
163     FROM PAYMENT;
164 END//
165 DELIMITER ;
166
167
168
169

Output
Action Output
# Time Action Message
96 21:52:08 CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW(), INTERVAL 1 H... 0 row(s) affected
97 21:52:13 SELECT * FROM RESERVATION LIMIT 0, 1000 6 row(s) returned
98 21:53:04 SELECT * FROM PAYMENT LIMIT 0, 1000 6 row(s) returned
99 21:54:37 CREATE PROCEDURE sp_get_payments() BEGIN SELECT PaymentID, ReservationID, Amo... 0 row(s) affected

Query Completed

```

```

Tables
  FEEDBACK
  PAYMENT
  REPORT
  RESERVATION
  SENSOR
  USER
  VEHICLE
Views
Stored Procedures
Functions
sakila
sampdb
SPATIAL_DB

Administration Schemas Information

Connection Details
Name: pro
Host: elvisdb.rowan.edu
Port: 3306
Login User: patels184
Current User: patels184@%
SSL cipher: TLS_AES_128_GCM_SHA256

Server
Product: Source distribution
Version: 8.0.43

Object Info Session
Query Completed

166
167
168 • CALL sp_get_payments();
169

Result Grid | Filter Rows: | Export: | Wrap Cell Content: 
| PaymentID | ReservationID | Amount | PaymentStatus |
| 1 | 1 | 12.00 | Paid |
| 2 | 2 | 18.00 | Paid |
| 3 | 3 | 10.00 | Paid |
| 4 | 4 | 0.00 | Refunded |
| 5 | 5 | 15.00 | Pending |
| 6 | 6 | 10.00 | Pending |

Result 33 ×
Output
Action Output
# Time Action Message
97 21:52:13 SELECT * FROM RESERVATION LIMIT 0, 1000 6 row(s) returned
98 21:53:04 SELECT * FROM PAYMENT LIMIT 0, 1000 6 row(s) returned
99 21:54:37 CREATE PROCEDURE sp_get_payments() BEGIN SELECT PaymentID, ReservationID, Amo... 0 row(s) affected
100 21:56:05 CALL sp_get_payments() 6 row(s) returned

Query Completed

```

- Triggers (2) – Code + Validation Output

### TRIGGER 1:

DELIMITER //

```
CREATE TRIGGER trg_update_sensor
AFTER UPDATE ON PARKINGSLOT
FOR EACH ROW
BEGIN
    UPDATE SENSOR
    SET Status = IF(NEW.Status = 'Occupied', 'Occupied', 'Free'),
        LastUpdate = NOW()
    WHERE SlotID = NEW.SlotID;
END//
```

DELIMITER ;

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is 'patels184'. The 'Tables' section lists: FEEDBACK, PARKINGLOT, PARKINGSLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, VEHICLE.
- Code Editor:** The code area contains the SQL script for creating the trigger:

```
171
172  DELIMITER //
173 • CREATE TRIGGER trg_update_sensor
174  AFTER UPDATE ON PARKINGSLOT
175  FOR EACH ROW
176  BEGIN
177      UPDATE SENSOR
178      SET Status = IF(NEW.Status = 'Occupied', 'Occupied', 'Free'),
179          LastUpdate = NOW()
180      WHERE SlotID = NEW.SlotID;
181  END//
182  DELIMITER ;
```

- Output Window:** The 'Output' tab displays the execution results of the commands:

#	Time	Action	Message
98	21:53:04	SELECT * FROM PAYMENT LIMIT 0, 1000	6 row(s) returned
99	21:54:37	CREATE PROCEDURE sp_get_payments() BEGIN SELECT PaymentID, ReservationID, Amo...	0 row(s) affected
100	21:56:05	CALL sp_get_payments()	6 row(s) returned
101	21:57:40	CREATE TRIGGER trg_update_sensor AFTER UPDATE ON PARKINGSLOT FOR EACH ROW...	0 row(s) affected

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view displays the 'ladies' schema with tables like FEEDBACK, PARKINGLOT, PAYMENT, REPORT, RESERVATION, SENSOR, USER, and VEHICLE. It also lists Views, Stored Procedures, Functions, and databases sakila, sampdb, and SPATIAL\_DB. Below the tree are tabs for Administration, Schemas, and Information.

In the center, a code editor window shows a script with two statements:

```

183
184
185 • UPDATE PARKINGLOT SET Status = 'Occupied' WHERE SlotID = 1;
186 • SELECT * FROM SENSOR WHERE SlotID = 1;
187
188

```

To the right of the code editor is a 'Result Grid' window showing the output of the SELECT statement:

SensorID	SlotID	Status	LastUpdate
1	1	Occupied	2025-11-17 02:59:02
NULL	NULL	NULL	NULL

Below the result grid is a 'Sensor 34' session window showing the history of actions:

#	Time	Action	Message
100	21:56:05	CALL sp_get_payments()	6 row(s) returned
101	21:57:40	CREATE TRIGGER trg_update_sensor AFTER UPDATE ON PARKINGLOT FOR EACH ROW...	0 row(s) affected
102	21:59:02	UPDATE PARKINGLOT SET Status = 'Occupied' WHERE SlotID = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
103	21:59:07	SELECT * FROM SENSOR WHERE SlotID = 1 LIMIT 0, 1000	1 row(s) returned

## TRIGGER 2:

```

DELIMITER //

CREATE TRIGGER trg_validate_rating
BEFORE INSERT ON FEEDBACK
FOR EACH ROW
BEGIN
    IF NEW.Rating < 1 OR NEW.Rating > 5 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Rating';
    END IF;
END//;

DELIMITER ;

```

```

187
188
189  DELIMITER //
190 • CREATE TRIGGER trg_validate_rating
191   BEFORE INSERT ON FEEDBACK
192   FOR EACH ROW
193   BEGIN
194     IF NEW.Rating < 1 OR NEW.Rating > 5 THEN
195       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Rating';
196     END IF;
197   END//
198
199  DELIMITER ;
200
201

```

**Session 1 (Top):**

```

187
188
189  DELIMITER //
190 • CREATE TRIGGER trg_validate_rating
191   BEFORE INSERT ON FEEDBACK
192   FOR EACH ROW
193   BEGIN
194     IF NEW.Rating < 1 OR NEW.Rating > 5 THEN
195       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Rating';
196     END IF;
197   END//
198
199  DELIMITER ;
200
201

```

**Session 2 (Bottom):**

```

187
188
189  DELIMITER //
190 • CREATE TRIGGER trg_validate_rating
191   BEFORE INSERT ON FEEDBACK
192   FOR EACH ROW
193   BEGIN
194     IF NEW.Rating < 1 OR NEW.Rating > 5 THEN
195       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Rating';
196     END IF;
197   END//
198
199  DELIMITER ;
200
201

```

### 3.6 Final Reflection

#### • Summary of Experience

Completion of Part 3 of the database design project enabled me to transition from a theoretical model to a fully functional physical implementation in MySQL. I designed a physical ER model, defining entities, attributes, and relationships using MySQL Workbench; after that, I forward-engineered the model into a live MySQL instance on Elvis. Populating the tables with sample data helped validate referential integrity and allowed me to experiment with SQL queries and database objects in a realistic environment.

SQL views, functions, stored procedures, and triggers showed how business rules could be embedded inside the database layer to enhance performance, constrain data integrity, and facilitate users' queries. In general, this project solidified my learning on relational database design, physical modeling, SQL scripting, and MySQL tools. It also helped me understand how a well-defined

schema fits into the real-world operations of workflows, such as reservations, payments, feedback collection, and reporting.

### • Challenges & Resolutions

One of the biggest hurdles was the foreign key and schema permission issues that were involved in the forward engineering step. Because the Elvis server does not allow schema creation by unauthorized users, the original script failed while trying to create a new database. To handle this, I edited the script to use my assigned schema, patels184, and went into MySQL Workbench and disabled the option to CREATE SCHEMA. This experience has greatly helped me understand database permissions and the constraints on deployment.

The other challenge arose with the correct order of sample data insertion. Due to the presence of foreign key dependencies, inserting into child tables before parent tables resulted in errors. I needed to resolve this by creating a structured insert sequence and temporarily disabling the foreign key checks during sample data loading.

Creating triggers and stored procedures required a lot of debugging, with the use of transactions. I learned how one can test procedures step-by-step, implement LAST\_INSERT\_ID(), and validate trigger behavior through controlled updates.

In the end, working through these challenges improved my understanding of relational integrity, error handling, and the importance of well-structured database logic. It also reinforced how database design decisions made early in the project affect implementation later.

## **4. Frontend Design & Implementation**

### **4.1 Project Description**

This final milestone implements the single-page frontend web application for the Smart Parking Management System. The page is written entirely in PHP + HTML and interacts directly with the physical MySQL schema deployed earlier on the Elvis server.

The objective of Part 4 is to demonstrate frontend interaction with the database by performing complete CRUD operations and executing SQL objects such as views, functions, stored procedures, and triggers.

The single page provides:

#### **Database Operations (CRUD)**

- Displaying records (SELECT): Shows all parking slots in a formatted table.
- Adding new records (INSERT): Allows the user to insert new parking slot data.
- Editing existing records (UPDATE): Updates slot status and shows trigger-based SENSOR synchronization.
- Deleting records (DELETE): Removes specific parking slot entries.

#### **SQL Objects Demonstrated**

- Views: vw\_reservation\_summary, vw\_lot\_usage
- Functions: fn\_reservation\_payment, fn\_reservation\_hours
- Stored Procedures: sp\_create\_reservation\_with\_payment, sp\_get\_payments
- Triggers:
  - trg\_update\_sensor (auto-updates SENSOR when a slot is updated)

- trg\_validate\_rating (rejects invalid ratings before insert)

## Frontend Implementation

- Fully integrated single-page layout with section navigation
- Clean and intuitive UI redesign: gradient background, card-style sections, styled buttons
- All actions display confirmation messages and database results

Together, these components verify that the Smart Parking Management System is fully functional from database design through deployment and user interaction.

The screenshot shows a web browser window with the URL [elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php](http://elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php). The page has a blue header bar with the title "Smart Parking Management" and the author "By: Sheena Patel". Below the header is a navigation bar with links: Quick Navigation, View Slots, Add Slot, Update Slot, Delete Slot, Views, Functions, Stored Procedures, and Rating Trigger. The main content area is divided into three cards:

- View All Parking Slots**: Contains a button labeled "Show Parking Slots".
- Add Parking Slot**: Contains fields for LotID (text input), Slot Number (text input), Slot Type (dropdown menu set to "Compact"), and an "Insert Slot" button.
- Update Slot Status (Trigger 1: trg\_update\_sensor)**: Contains fields for SlotID (text input), New Status (dropdown menu set to "Available"), and an "Update Status" button.

⚠ Not secure elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php

### Delete Parking Slot

SlotID:

[Delete Slot](#)

### Views Output

[Reservation Summary \(vw\\_reservation\\_summary\)](#) [Lot Usage \(vw\\_lot\\_usage\)](#)

### Functions Output

ReservationID:

[Get Payment Amount \(fn\\_reservation\\_payment\)](#) [Get Reservation Hours \(fn\\_reservation\\_hours\)](#)

⚠ Not secure elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php

### Stored Procedures

#### Create Reservation + Payment (sp\_create\_reservation\_with\_payment)

SlotID:

UserID:

VehicleID:

Start (YYYY-MM-DD HH:MM):  mm/dd/yyyy --:-- --

End (YYYY-MM-DD HH:MM):  mm/dd/yyyy --:-- --

Amount:

[Run SP – Create Reservation](#)

#### Show All Payments (sp\_get\_payments)

[Run SP – Get Payments](#)

Run SP – Get Payments

### Trigger 2: Validate Rating (trg\_validate\_rating)

This form inserts into FEEDBACK. Trigger `trg_validate_rating` ensures ratings are between 1 and 5.

UserID:

LotID:

Rating (1-5):

Comment:

**Insert Feedback**

## 4.2 CRUD Operations Screenshots

### Displaying Records (SELECT):

**View All Parking Slots**

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact

The screenshot shows a MySQL Workbench session with the following details:

- Schemas:** The current schema is 'formation'.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Connector:**
  - Version: C++ 9.3.0

In the SQL pane, the query `select * from PARKINGSLOT;` is run, resulting in the following output grid:

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
*	HULL	HULL	HULL	HULL

The Action Output pane shows two log entries:

- # 10 22-06-21 SELECT \* FROM PAYMENT LIMIT 0, 1000 Message 6 row(s) returned
- # 11 22-06-21 select \* from PARKINGSLOT LIMIT 0, 1000 Message 10 row(s) returned

This screenshot shows all parking slots retrieved from the PARKINGSLOT table, demonstrating successful SELECT operation.

### Insert Operation:

Add Parking Slot

LotID:

Slot Number:

Slot Type:  Compact

Add Parking Slot

LotID:  2

Slot Number:  A1

Slot Type:  Compact

Compact

**Insert Slot**

**Slot inserted successfully!**

**View All Parking Slots**

**Show Parking Slots**

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
18	2	A1	Available	Compact

6

7 • `select * from PARKINGSLOT;`

**Result Grid**

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
18	2	A1	Available	Compact

**PARKINGSLOT 2**

**Output**

#	Time	Action	Message
11	22:56:49	select * from PARKINGSLOT LIMIT 0, 1000	10 row(s) returned
12	23:10:07	select * from PARKINGSLOT LIMIT 0, 1000	11 row(s) returned

This screenshot confirms that a new parking slot was inserted successfully into the database using an INSERT statement.

## Update Operation:

**View All Parking Slots**

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	Available	Compact

▶ orderdetails  
▶ orders  
▶ PARKINGLOT  
▶ PARKINGSLOT  
▶ PAYMENT  
▶ payments  
▶ productlines  
▶ products  
▶ REPORT  
▶ RESERVATION  
▶ RTDataExtract  
▶ SENSOR  
▶ USER  
▶ VEHICLE

7 • `select * from PARKINGSLOT;`

**Result Grid**

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	Available	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
18	2	A1	Available	Compact
NULL	NULL	NULL	NULL	NULL

PARKINGSLOT 3 ×

Output

## Update Slot Status (Trigger 1: trg\_update\_sensor)

SlotID:

New Status:

**Update Status**

**View All Parking Slots**

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact

6  
7 • `select * from PARKINGSLOT;`

Result Grid | Filter Rows: Edit: Export/Import:

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
18	2	A1	Available	Compact
*	HULL	HULL	HULL	HULL

PARKINGSLOT 4 ×

Output

Action Output

#	Time	Action
13	23:12:13	select *from PARKINGSLOT LIMIT 0, 1000
14	23:14:02	select *from PARKINGSLOT LIMIT 0, 1000

This screenshot shows a slot status being updated.

### Delete Operation:

**View All Parking Slots**

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large

**Delete Parking Slot**

SlotID:

**Delete Slot**

**Delete Slot**

**Slot deleted successfully!**

**View All Parking Slots**

**Show Parking Slots**

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
4	1	A4	Occupied	Large

► PARKINGLOT  
► PARKINGSLOT  
► PAYMENT  
► payments  
► productlines  
► products  
► REPORT  
► RESERVATION  
► RTDataExtract  
► SENSOR  
► USER  
► VEHICLE

ministration Schemas

formation

nection Details

Name: pro  
Host: elvisdb.rowan.edu  
Port: 3306  
Login User: patels184  
Current User: patels184@%  
SSL cipher: TLS\_AES\_128\_GCM\_SHA256

erver

Product: Source distribution  
Version: 8.0.43

nector

Version: C++ 9.3.0

7 • `select * from PARKINGSLOT;`

Result Grid | Filter Rows: Edit: Export/Import: Wrap Cell Content:

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact
18	2	A1	Available	Compact
*	*	*	*	*

PARKINGSLOT 5 ×

Output:

Action Output

#	Time	Action	Message
14	23:14:02	select * from PARKINGSLOT LIMIT 0, 1000	11 row(s) returned
15	23:17:07	select * from PARKINGSLOT LIMIT 0, 1000	10 row(s) returned

This screenshot demonstrates the DELETE operation where the selected parking slot was removed from the system.

## 4.3 Views, Functions, Stored Procedures, Triggers Output

### Views Output

#### Views Output

Reservation Summary (vw\_reservation\_summary)

Lot Usage (vw\_lot\_usage)

#### View 1:

##### Views Output

Reservation Summary (vw\_reservation\_summary)

Lot Usage (vw\_lot\_usage)

ReservationID	UserName	LicensePlate	StartTime	EndTime	Status
1	Alice Driver	ABC1111	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
6	Alice Driver	ABC1111	2025-11-17 02:52:08	2025-11-17 03:52:08	Active
3	David Driver	DRV3333	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
5	Eva Driver	EV55555	2025-11-18 18:00:00	2025-11-18 20:00:00	Active
4	Eva Driver	EVA4444	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
17	David Driver	EVA4444	2025-12-07 22:15:00	2025-12-07 23:15:00	Active
2	Alice Driver	XYZ2222	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
14	Bob Manager	XYZ2222	2025-12-07 22:09:00	2025-12-07 22:30:00	Active

The screenshot shows a database interface with a sidebar containing connection details and a main area displaying the output of a query.

**Schemas:**

- orderdetails
- orders
- PARKINGLOT
- PARKINGSLOT
- PAYMENT
- payments
- productlines
- products
- REPORT
- RESERVATION
- RTDataExtract
- SENSOR
- USER
- VEHICLE

**Connection Details:**

Name: pro  
Host: elvisdb.rowan.edu  
Port: 3306  
Login User: patels184  
Current User: patels184@%  
SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server:**

Product: Source distribution  
Version: 8.0.43

**Connector:**

Version: C++ 9.3.0

**Query Output:**

```
63 • SELECT * FROM vw_reservation_summary;
64
65
66
```

**Result Grid:**

ReservationID	UserName	LicensePlate	StartTime	EndTime	Status
1	Alice Driver	ABC1111	2025-11-15 08:00:00	2025-11-15 10:00:00	Completed
6	Alice Driver	ABC1111	2025-11-17 02:52:08	2025-11-17 03:52:08	Active
3	David Driver	DRV3333	2025-11-15 12:00:00	2025-11-15 14:00:00	Completed
5	Eva Driver	EV55555	2025-11-18 18:00:00	2025-11-18 20:00:00	Active
4	Eva Driver	EVA4444	2025-11-17 15:00:00	2025-11-17 17:00:00	Cancelled
17	David Driver	EVA4444	2025-12-07 22:15:00	2025-12-07 23:15:00	Active
2	Alice Driver	XYZ2222	2025-11-16 09:00:00	2025-11-16 11:30:00	Active
14	Bob Manager	XYZ2222	2025-12-07 22:09:00	2025-12-07 22:30:00	Active

**vw\_reservation\_summary 5 ×**

**Action Output:**

#	Time	Action	Message
15	23:17:07	select * from PARKINGSLOT LIMIT 0, 1000	10 row(s) returned
16	23:18:25	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	8 row(s) returned

This screenshot displays the output of the reservation summary view, showing joined reservation and user data.

## View 2:

The screenshot shows a database interface with a sidebar containing connection details and a main area displaying the output of a query.

**Schemas:**

- REPORT
- RESERVATION
- RTDataExtract
- SENSOR
- USER
- VEHICLE

**Connection Details:**

Name: pro  
Host: elvisdb.rowan.edu  
Port: 3306  
Login User: patels184  
Current User: patels184@%  
SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server:**

Product: Source distribution  
Version: 8.0.43

**Connector:**

Version: C++ 9.3.0

**Query Output:**

```
86 • SELECT * FROM vw_lot_usage;
87
88
```

**Result Grid:**

ParkingLot	TotalSlots	SlotsConfigured	OccupiedSlots	AvailableSlots
Downtown Garage	10	7	1	3
Stadium Lot	15	1	0	1
Airport Lot A	12	1	0	1
Mall Parking	8	1	0	1

**vw\_lot\_usage 6 ×**

**Action Output:**

#	Time	Action	Message
16	23:18:25	SELECT * FROM vw_reservation_summary LIMIT 0, 1000	8 row(s) returned
17	23:20:11	SELECT * FROM vw_lot_usage LIMIT 0, 1000	4 row(s) returned

This screenshot shows the lot usage view, summarizing total, occupied, and available slots per parking lot.

## Functions Output

**Functions Output**

ReservationID:

**Get Payment Amount (fn\_reservation\_payment)**    **Get Reservation Hours (fn\_reservation\_hours)**

### Function 1:

**Functions Output**

ReservationID:

**Get Payment Amount (fn\_reservation\_payment)**    **Get Reservation Hours (fn\_reservation\_hours)**

**Payment Amount:** 18.00

The screenshot shows a database interface with the following details:

- Schemas:** production, products, REPORT, RESERVATION, RTDataExtract, SENSOR, USER, VEHICLE.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Connector:**
  - Version: C++ 9.3.0
- Query:** SELECT FN\_RESERVATION\_PAYMENT(2);
- Result Grid:** FN\_RESERVATION\_PAYMENT(2) | 18.00
- Action Output:**

#	Time	Action	Message
17	23:20:11	SELECT * FROM vw_lot_usage LIMIT 0, 1000	4 row(s) returned
18	23:22:06	SELECT FN_RESERVATION_PAYMENT(2) LIMIT 0, 1000	1 row(s) returned

This screenshot demonstrates retrieving the payment amount for a reservation using the reservation ID.

## Function 2:

**Functions Output**

ReservationID:

**Get Payment Amount (fn\_reservation\_payment)**    **Get Reservation Hours (fn\_reservation\_hours)**

**Total Hours:** 2.50

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** A tree view showing schemas: `productions`, `products`, `REPORT`, `RESERVATION`, `RTDdataExtract`, `SENSOR`, `USER`, and `VEHICLE`.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Connector:**
  - Version: C++ 9.3.0
- Query Results:**

```

127
128 •  SELECT fn_reservation_hours(2);
129
130
  
```

The result grid shows the output of the query: `fn_reservation_hours(2)` with a value of `2.50`.

- Action Output:**

#	Time	Action	Message
18	23:22:06	SELECT FN_RESERVATION_PAYMENT(2) LIMIT 0, 1000	1 row(s) returned
19	23:22:56	SELECT fn_reservation_hours(2) LIMIT 0, 1000	1 row(s) returned

This screenshot shows the function calculating total reserved hours based on start and end time.

## Stored Procedures

### Stored Procedures

#### Create Reservation + Payment (sp\_create\_reservation\_with\_payment)

SlotID:

UserID:

VehicleID:

Start (YYYY-MM-DD HH:MM):

 mm/dd/yyyy --:-- -- 

End (YYYY-MM-DD HH:MM):

 mm/dd/yyyy --:-- -- 

Amount:

[Run SP – Create Reservation](#)

#### Show All Payments (sp\_get\_payments)

[Run SP – Get Payments](#)

## Stored Procedures 1:

**Stored Procedures**

**Create Reservation + Payment (sp\_create\_reservation\_with\_payment)**

SlotID:

UserID:

VehicleID:

Start (YYYY-MM-DD HH:MM):

End (YYYY-MM-DD HH:MM):

Amount:

**Run SP – Create Reservation**

**Run SP – Create Reservation**

**Reservation & Payment created successfully by stored procedure!**

**New Reservation Created:**

ReservationID	SlotID	UserID	VehicleID	StartTime	EndTime	Status
17	4	4	4	2025-12-07 22:15:00	2025-12-07 23:15:00	Active

**Payment Created:**

PaymentID	ReservationID	Amount	PaymentMethod	PaymentStatus	TransactionDate
8	17	2.00	Card	Pending	2025-12-08 03:15:50

Administration Schemas

**Connection Details**

- Name: pro
- Host: elvisdb.rowan.edu
- Port: 3306
- Login User: patels184
- Current User: patels184@%
- SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server**

- Product: Source distribution
- Version: 8.0.43

**Connector**

- Version: C++ 9.3.0

RESERVATION 11 ×

Output:

#	Time	Action	Message
21	23:23:53	SELECT * FROM PAYMENT LIMIT 0, 1000	8 row(s) returned
22	23:24:44	SELECT * FROM RESERVATION LIMIT 0, 1000	8 row(s) returned

Administration Schemas

**Connection Details**

- Name: pro
- Host: elvisdb.rowan.edu
- Port: 3306
- Login User: patels184
- Current User: patels184@%
- SSL cipher: TLS\_AES\_128\_GCM\_SHA256

**Server**

- Product: Source distribution
- Version: 8.0.43

**Connector**

- Version: C++ 9.3.0

PAYMENT 12 ×

Output:

#	Time	Action	Message
22	23:24:44	SELECT * FROM RESERVATION LIMIT 0, 1000	8 row(s) returned
23	23:25:36	SELECT * FROM PAYMENT LIMIT 0, 1000	8 row(s) returned

This screenshot shows successful execution of the stored procedure which creates a reservation and its corresponding payment.

## Stored Procedures 2:

Run SP – Get Payments

PaymentID	ReservationID	Amount	PaymentStatus
1	1	12.00	Paid
2	2	18.00	Paid
3	3	10.00	Paid
4	4	0.00	Refunded
5	5	15.00	Pending
6	6	10.00	Pending
7	14	2.25	Pending
8	17	2.00	Pending

The screenshot shows a database interface with the following details:

- Schemas:** A tree view of schemas including orders, PARKINGLOT, PAYMENT, payments, productlines, products, REPORT, RESERVATION, RTDataExtract, SENSOR, USER, and VEHICLE.
- Connection Details:**
  - Name: pro
  - Host: elvisdb.rowan.edu
  - Port: 3306
  - Login User: patels184
  - Current User: patels184@%
  - SSL cipher: TLS\_AES\_128\_GCM\_SHA256
- Server:**
  - Product: Source distribution
  - Version: 8.0.43
- Connector:**
  - Version: C++ 9.3.0

In the main pane, the command `CALL sp_get_payments();` is run, resulting in the following output:

PaymentID	ReservationID	Amount	PaymentStatus
1	1	12.00	Paid
2	2	18.00	Paid
3	3	10.00	Paid
4	4	0.00	Refunded
5	5	15.00	Pending
6	6	10.00	Pending
7	14	2.25	Pending
8	17	2.00	Pending

The Action Output pane shows the following log entries:

#	Time	Action	Message
23	23:25:36	SELECT * FROM PAYMENT LIMIT 0, 1000	8 row(s) returned
24	23:26:19	CALL sp_get_payments()	8 row(s) returned

This screenshot displays all payment records returned by the stored procedure `sp_get_payments`.

## Triggers Demonstration

### Trigger 1:

#### View All Parking Slots

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Reserved	Compact
4	1	A4	Occupied	Large

#### Update Slot Status (Trigger 1: trg\_update\_sensor)

SlotID:

New Status:

Update Status

Update Status

Status updated successfully! Trigger `trg_update_sensor` auto-updated SENSOR.

#### Sensor Row After Update (Trigger Output)

SensorID	SlotID	Status	LastUpdate
2	2	Free	2025-12-08 03:23:16

**Update Slot Status (Trigger 1: trg\_update\_sensor)**

SlotID:

New Status:

---

Status updated successfully! Trigger `trg_update_sensor` auto-updated SENSOR.

**Sensor Row After Update (Trigger Output)**

SensorID	SlotID	Status	LastUpdate
2	2	Occupied	2025-12-08 03:24:43

---

This screenshot shows Trigger 1 updating the SENSOR table automatically when a slot status changes.

**Trigger 2:****Trigger 2: Validate Rating (trg\_validate\_rating)**

This form inserts into FEEDBACK. Trigger `trg_validate_rating` ensures ratings are between 1 and 5.

UserID:

LotID:

Rating (1-5):

Comment:

Insert Feedback

**Thank you for your feedback!**

### Trigger 2: Validate Rating (trg\_validate\_rating)

This form inserts into FEEDBACK. Trigger trg\_validate\_rating ensures ratings are between 1 and 5.

UserID:

LotID:

Rating (1-5):

Comment:

Insert Feedback

Insert Feedback

**Rating invalid. Please rate from 1 to 5.**

This screenshot displays validation preventing invalid ratings and allowing only values between 1 and 5.

### 4.4 Project URL

<http://elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php>