Advance Database System

Course Project Report

Smart Parking Management System

Submitted by:

Sheena Patel

Table of Contents

- 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 (1st 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)

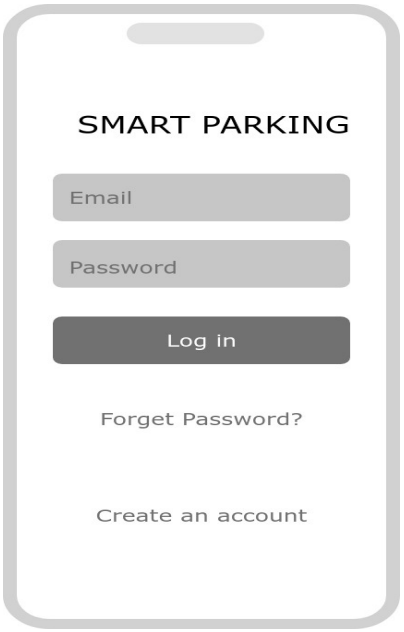
PK = ReportID

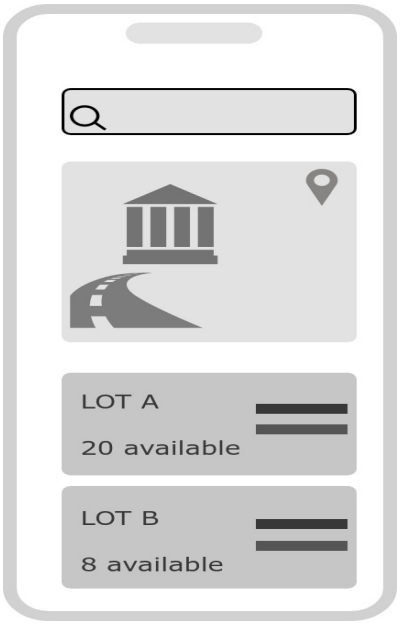
FK = LotID → ParkingLot(LotID)

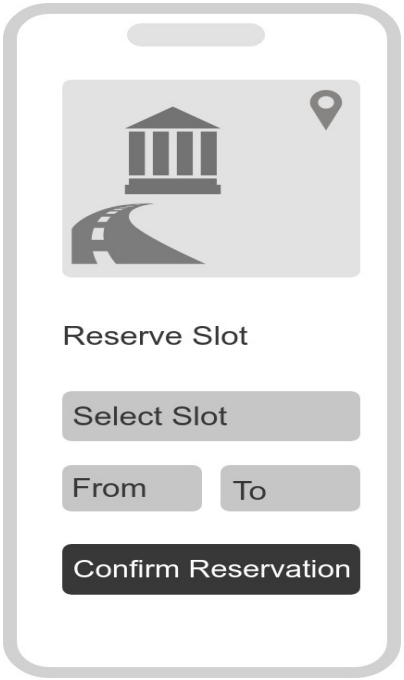
1.6 UI Prototype

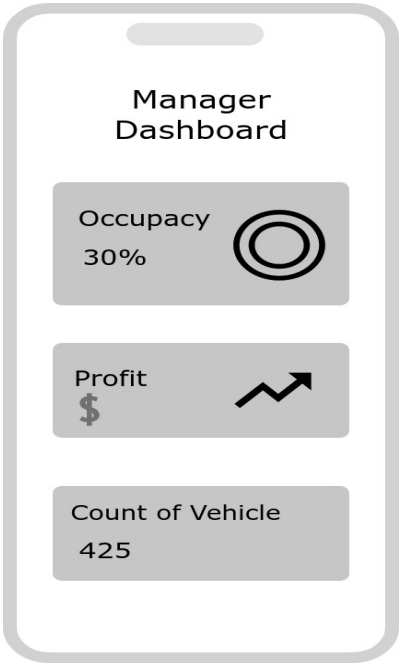
• Wireframe

https://pr.to/TPS347/

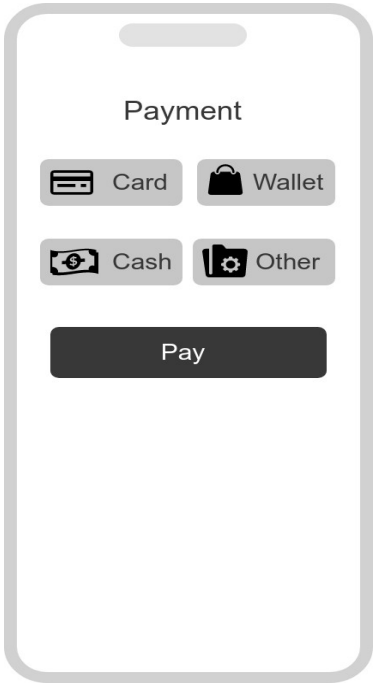








Sheena Patel



Screen	Purpose / Data Interaction	Backend Entities Involved
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)

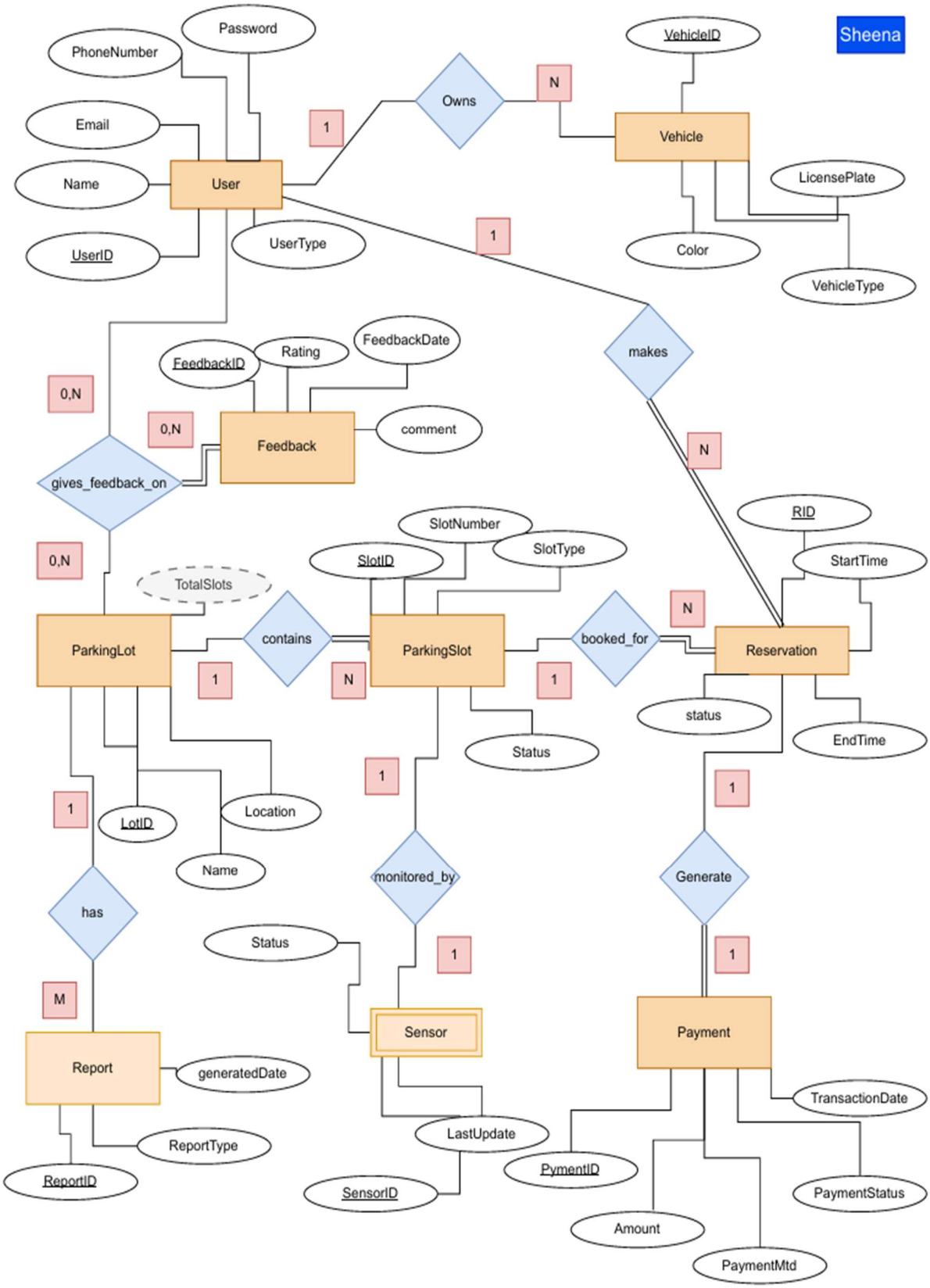
• Entities and Attributes

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

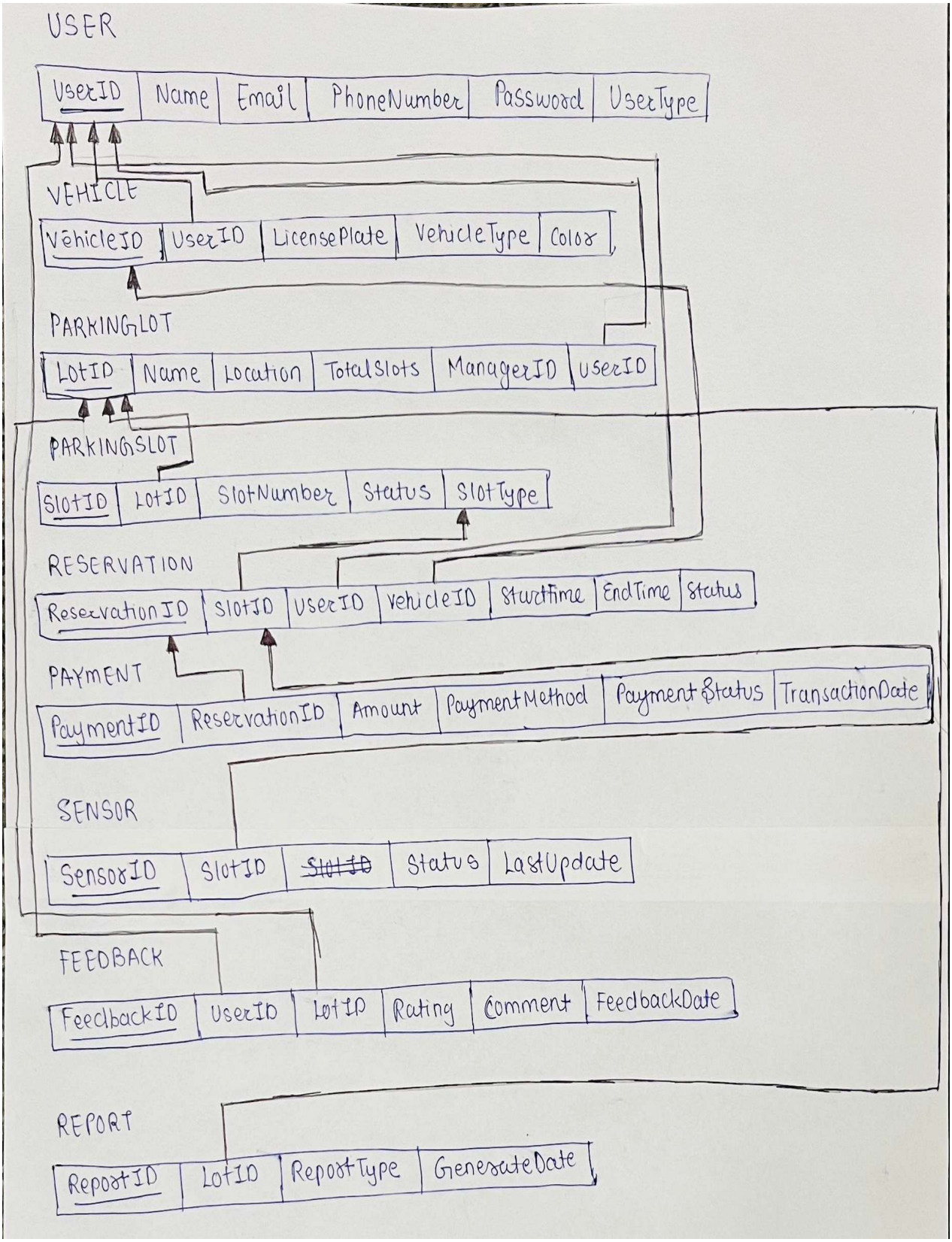
- PAYMENT(PaymentID PK, ReservationID FK→RESERVATION(ReservationID) UNIQUE, Amount, PaymentMethod, PaymentStatus, TransactionDate)

- 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 Normaliztion

• First Normal Form (1st 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 1st 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 3rd 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).

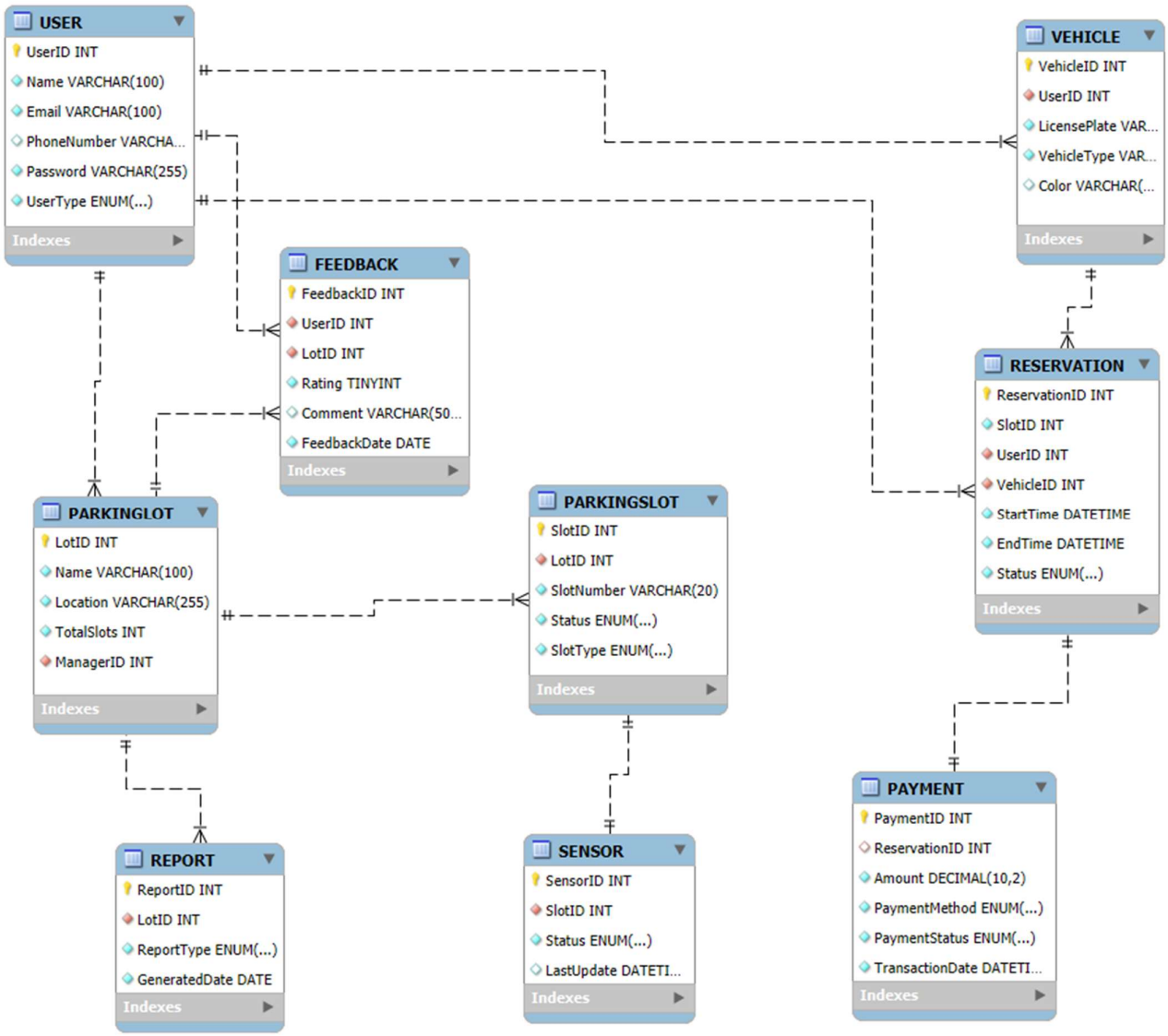
The normalization phase clarified once again the need for doing dependency analysis in order to avoid data anomalies. Discovering possible transitive dependencies (for example, ManagerName → ManagerID → PARKINGLOT) increased the efficiency of the design and ensured data integrity.

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.

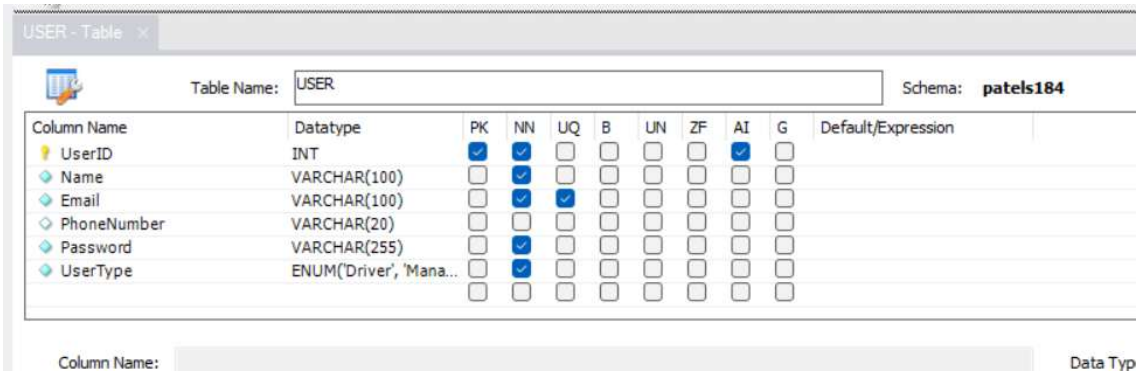
3. Physical Design & Performance

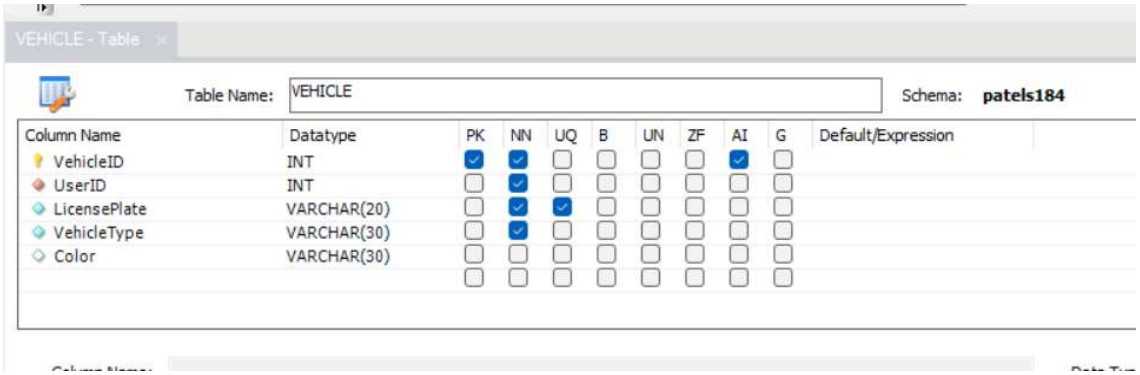
3.1 Physical Model Design

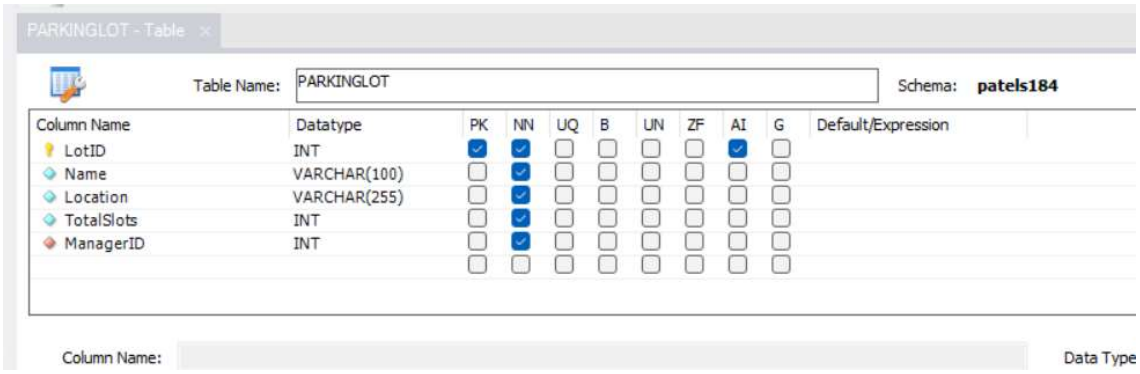
- Physical ER Diagram (MySQL Workbench)



- Table Structures (Columns, Data Types, Constraints)







PARKINGSLOT - Table

Table Name: PARKINGSLOT Schema: **patels184**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
SlotID	INT	☑	☑	☐	☐	☐	☐	☑	☐	
LotID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
SlotNumber	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	☐	
Status	ENUM('Available', 'Re...	☐	☑	☐	☐	☐	☐	☐	☐	'Available'
SlotType	ENUM('Compact', 'La...	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

RESERVATION - Table

Table Name: RESERVATION Schema: **patels184**

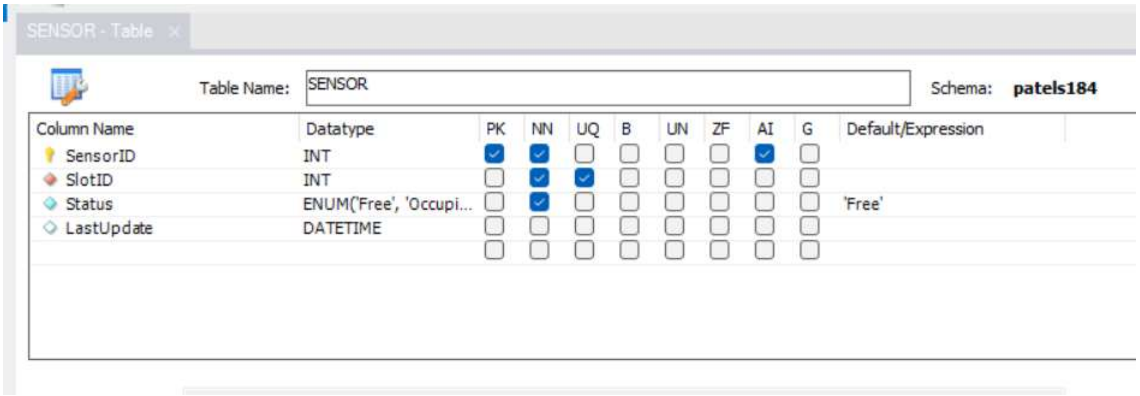
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ReservationID	INT	☑	☑	☐	☐	☐	☐	☑	☐	
SlotID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
UserID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
VehicleID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
StartTime	DATETIME	☐	☑	☐	☐	☐	☐	☐	☐	
EndTime	DATETIME	☐	☑	☐	☐	☐	☐	☐	☐	
Status	ENUM('Active', 'Comp...	☐	☑	☐	☐	☐	☐	☐	☐	'Active'
		☐	☐	☐	☐	☐	☐	☐	☐	

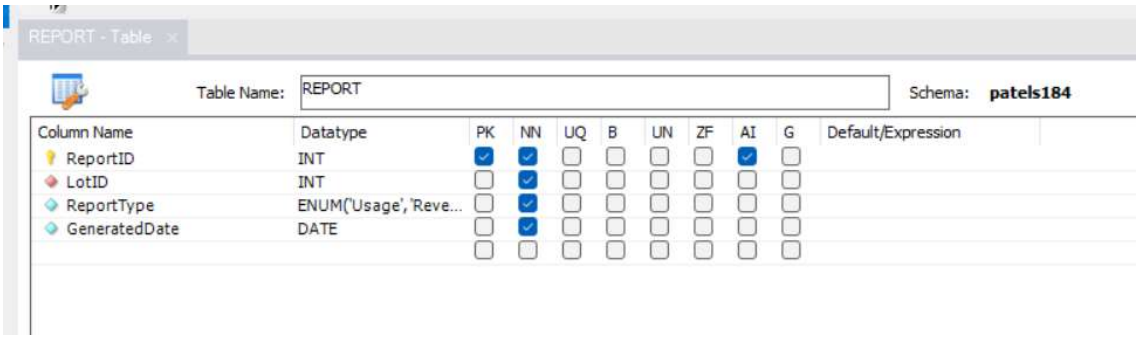
Column Name:

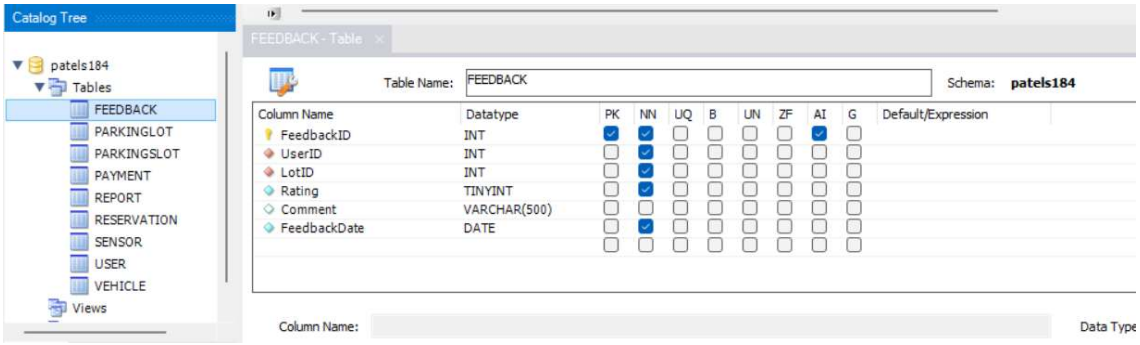
PAYMENT - Table

Table Name: PAYMENT Schema: **patels184**

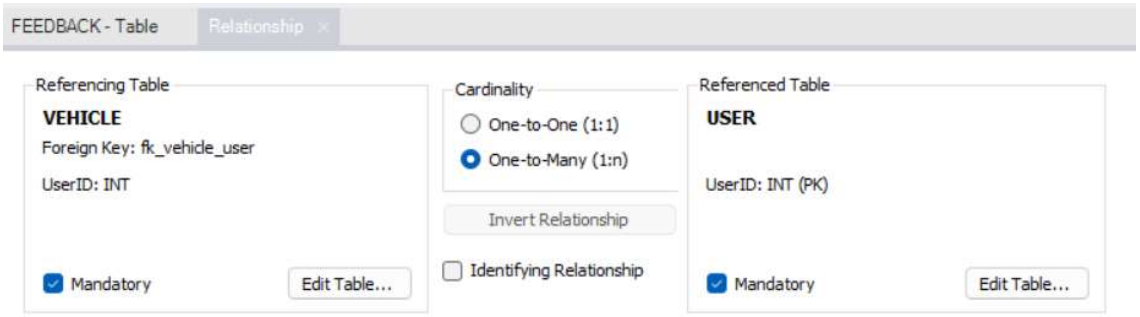
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
PaymentID	INT	☑	☑	☐	☐	☐	☐	☑	☐	
ReservationID	INT	☐	☐	☑	☐	☐	☐	☐	☐	
Amount	DECIMAL(10,2)	☐	☑	☐	☐	☐	☐	☐	☐	
PaymentMethod	ENUM('Card', 'Wallet'...	☐	☑	☐	☐	☐	☐	☐	☐	
PaymentStatus	ENUM('Paid', 'Pendin...	☐	☑	☐	☐	☐	☐	☐	☐	'Paid'
TransactionDate	DATETIME	☐	☑	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

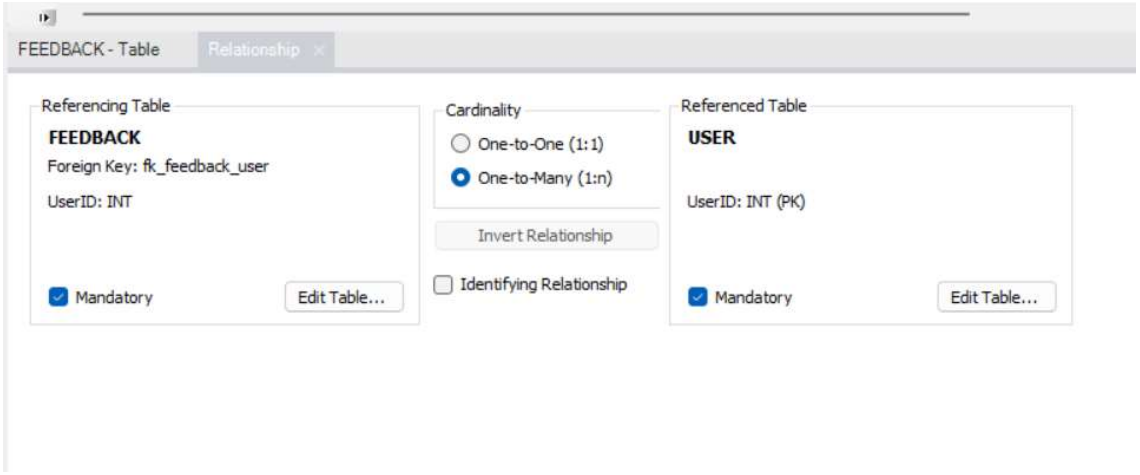


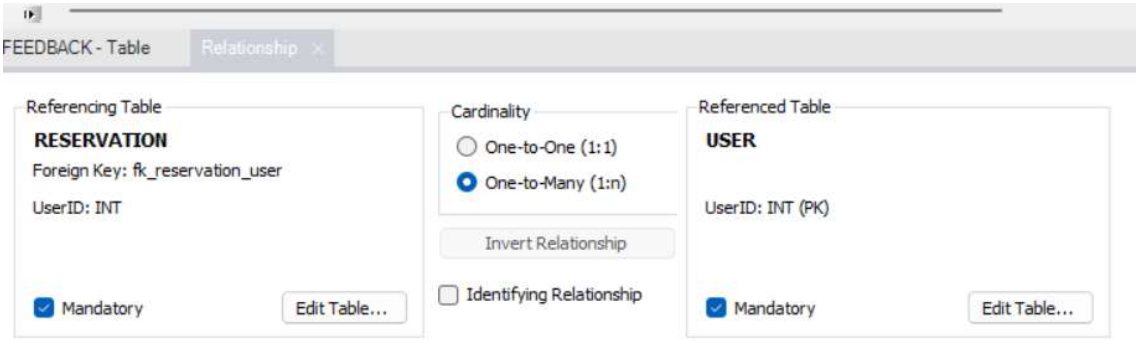


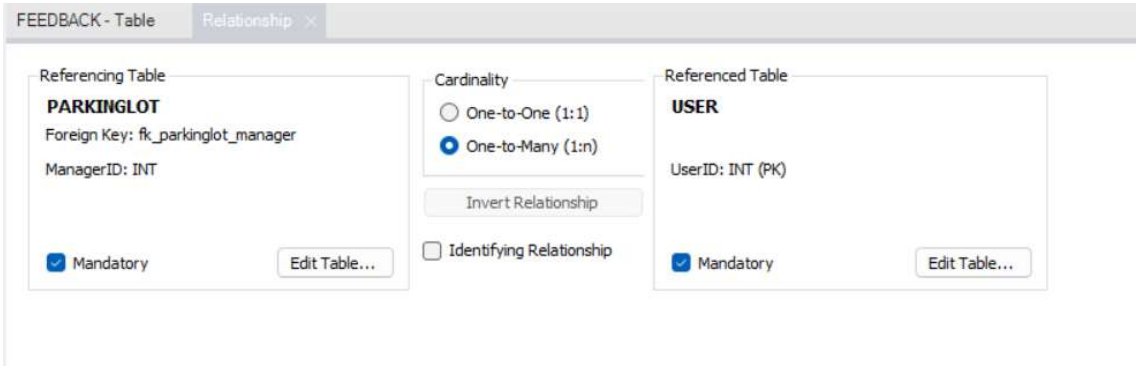


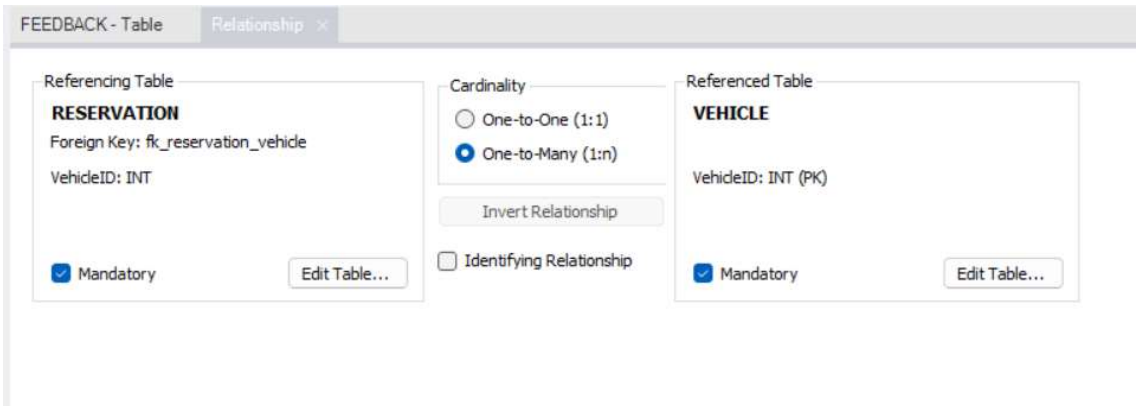
- Relationship Cardinality

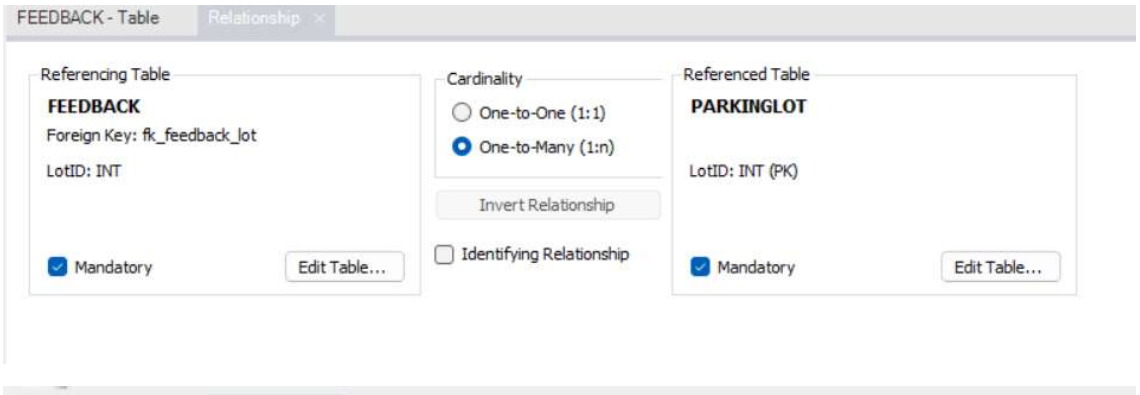

FEEDBACK - Table
Relationship
Referencing Table
VEHICLE
Foreign Key: fk_vehicle_user
UserID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
USER
UserID: INT (PK)
Mandatory
Edit Table...

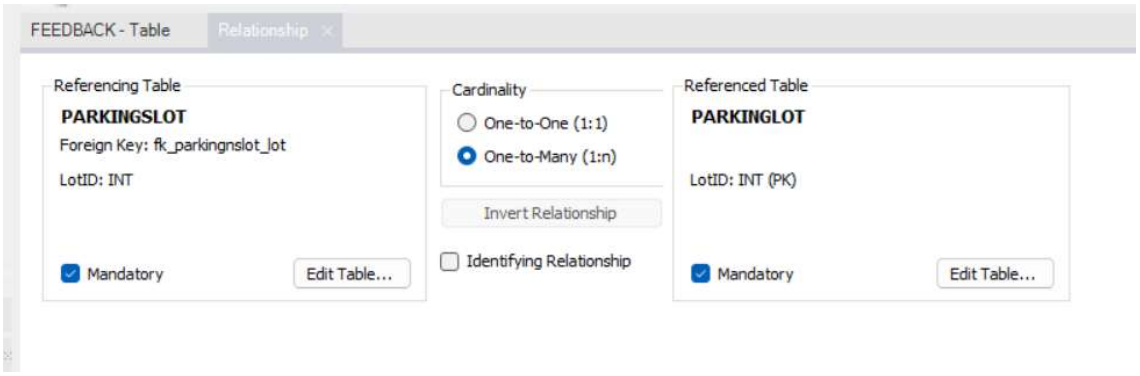

FEEDBACK - Table
Relationship
Referencing Table
FEEDBACK
Foreign Key: fk_feedback_user
UserID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
USER
UserID: INT (PK)
Mandatory
Edit Table...

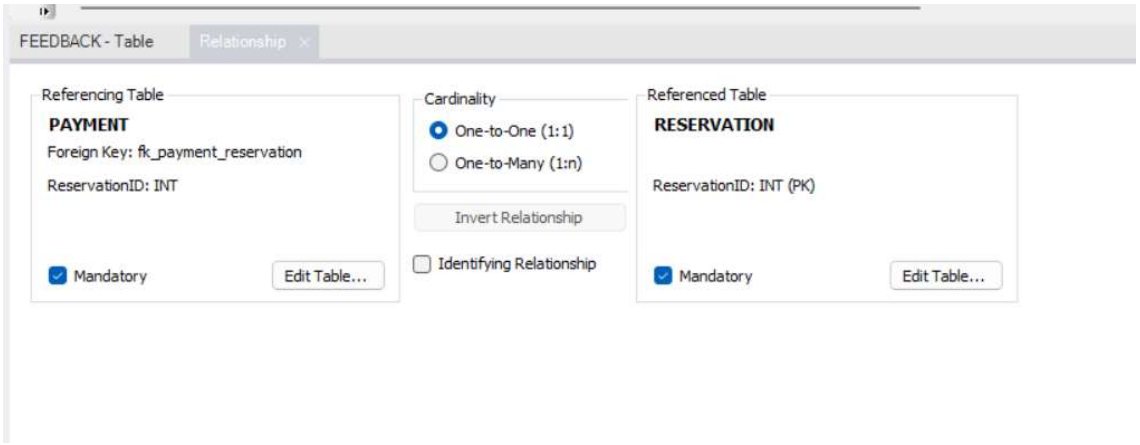

FEEDBACK - Table
Relationship
Referencing Table
RESERVATION
Foreign Key: fk_reservation_user
UserID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
USER
UserID: INT (PK)
Mandatory
Edit Table...

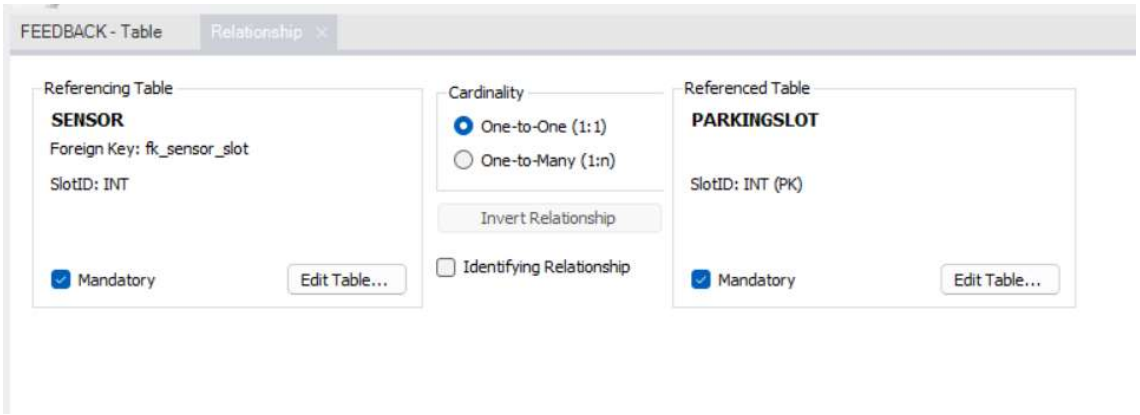

FEEDBACK - Table
Relationship
Referencing Table
PARKINGLOT
Foreign Key: fk_parkinglot_manager
ManagerID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
USER
UserID: INT (PK)
Mandatory
Edit Table...

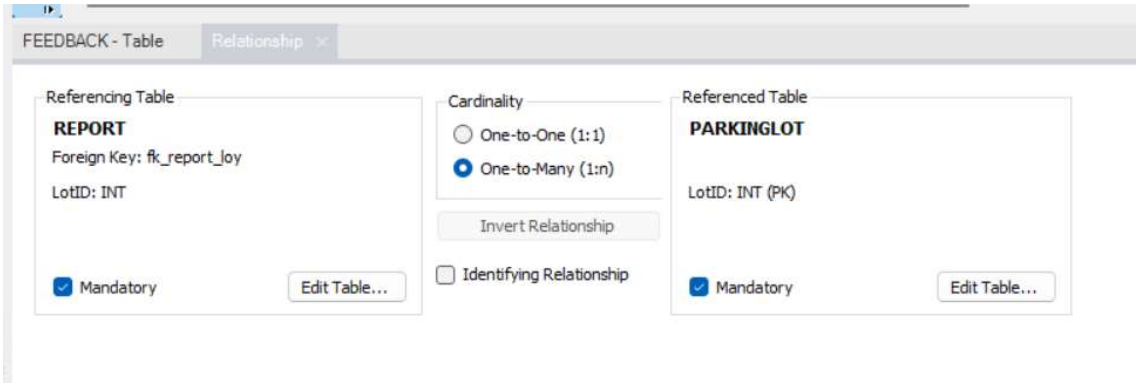

FEEDBACK - Table
Relationship
Referencing Table
RESERVATION
Foreign Key: fk_reservation_vehicle
VehicleID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
VEHICLE
VehicleID: INT (PK)
Mandatory
Edit Table...


FEEDBACK - Table
Relationship
Referencing Table
FEEDBACK
Foreign Key: fk_feedback_lot
LotID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
PARKINGLOT
LotID: INT (PK)
Mandatory
Edit Table...


FEEDBACK - Table
Relationship
Referencing Table
PARKINGSLOT
Foreign Key: fk_parkingnslot_lot
LotID: INT
Mandatory
Edit Table...
Cardinality
One-to-One (1:1)
One-to-Many (1:n)
Invert Relationship
Identifying Relationship
Referenced Table
PARKINGLOT
LotID: INT (PK)
Mandatory
Edit Table...







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;
```

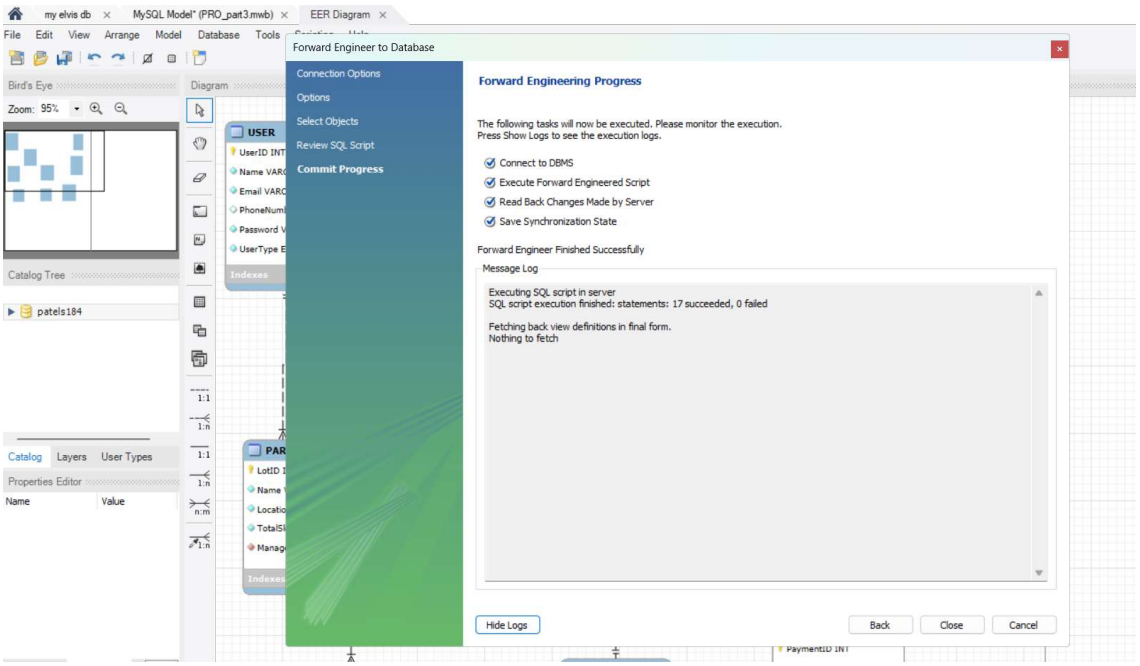
```
-- -----------------------------------------------------
-- Table `patels184`.`RESERVATION`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `patels184`.`RESERVATION` (
  `ReservationID` INT NOT NULL AUTO_INCREMENT,
  `SlotID` INT NOT NULL,
  `UserID` INT NOT NULL,
  `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;
```

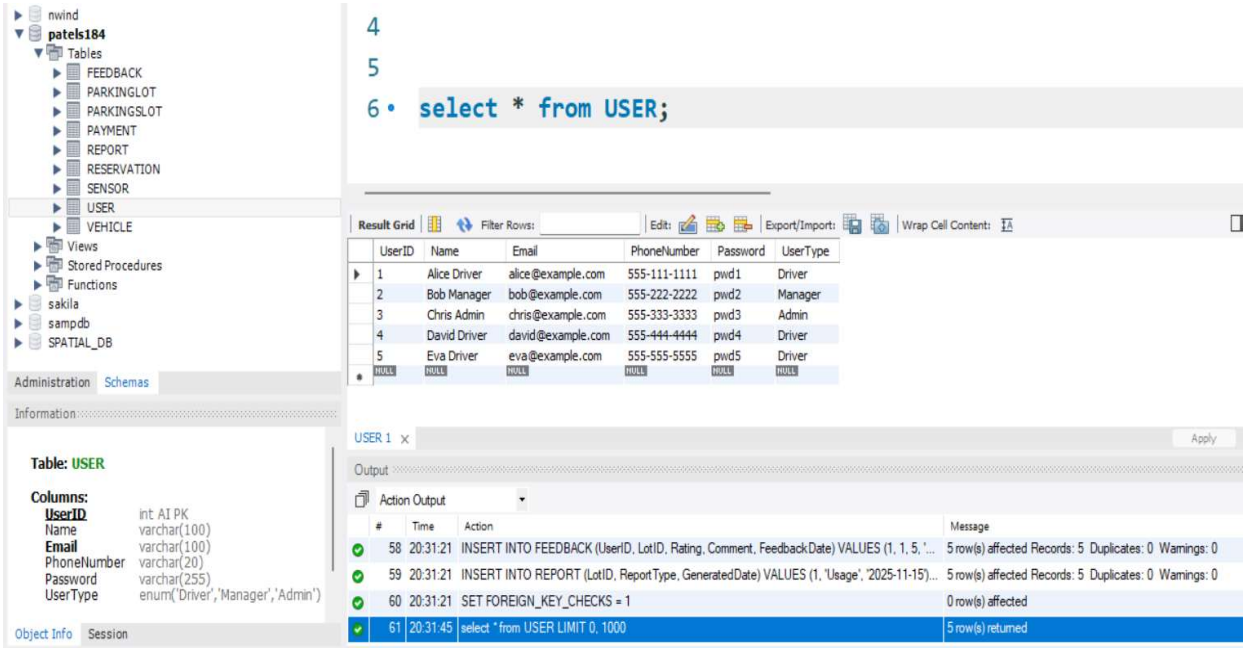
```
-- -----------------------------------------------------
-- Table `patels184`.`REPORT`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `patels184`.`REPORT` (
  `ReportID` INT NOT NULL AUTO_INCREMENT,
  `LotID` INT NOT NULL,
  `ReportType` ENUM('Usage', 'Revenue', '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

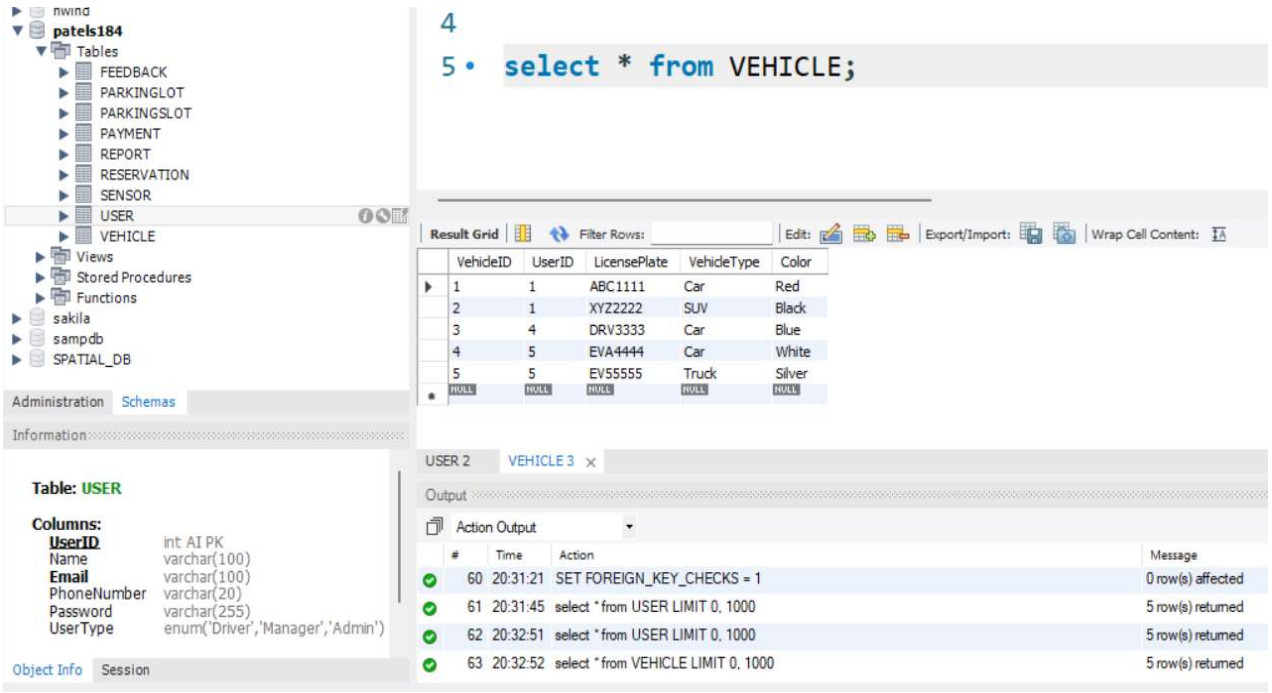


3.3 Data Population

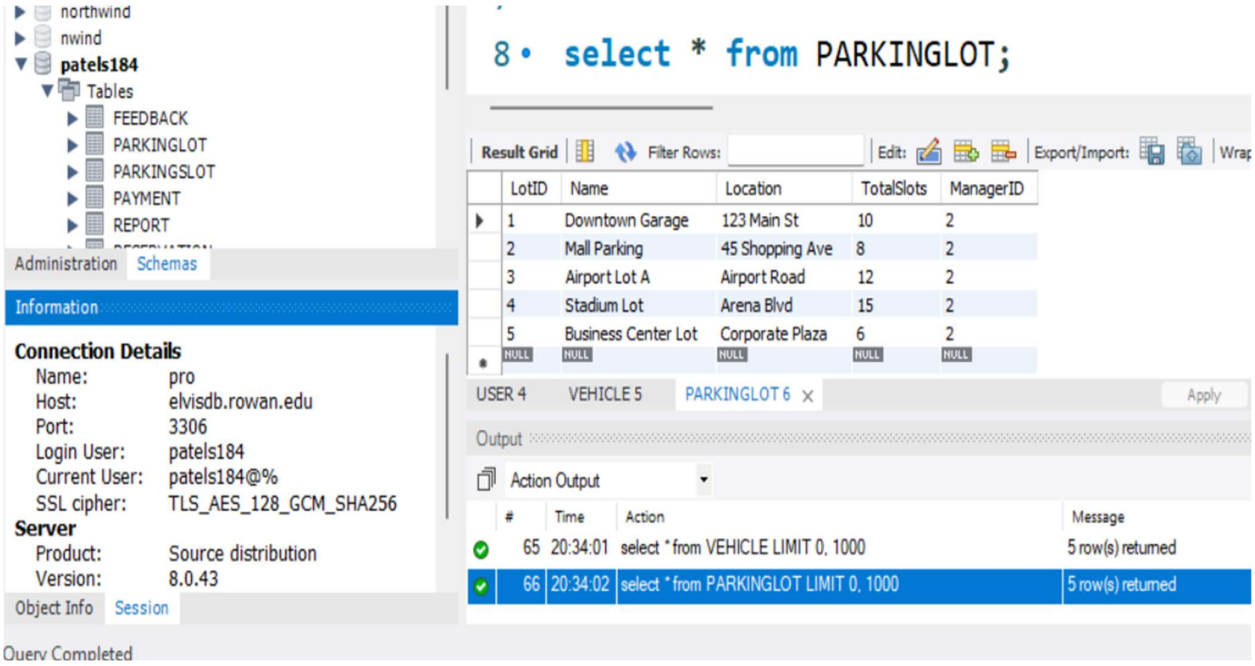
- Screenshot – USER table (5 records)



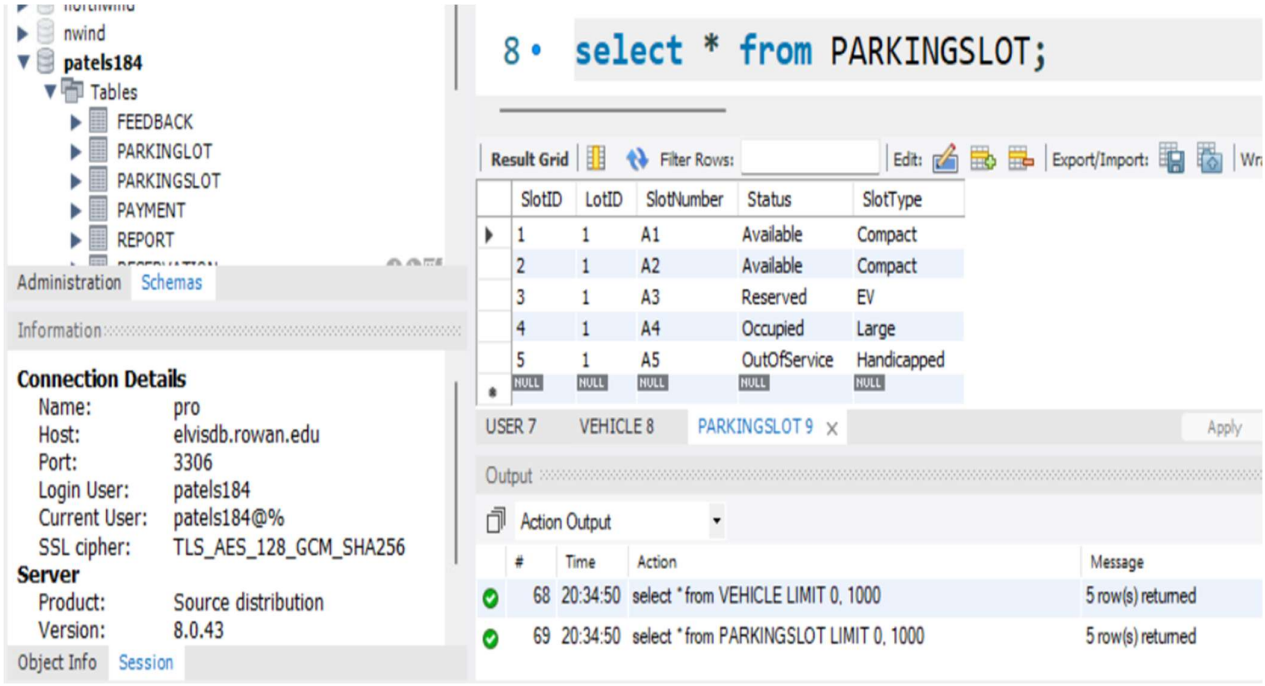
- Screenshot – VEHICLE table (5 records)



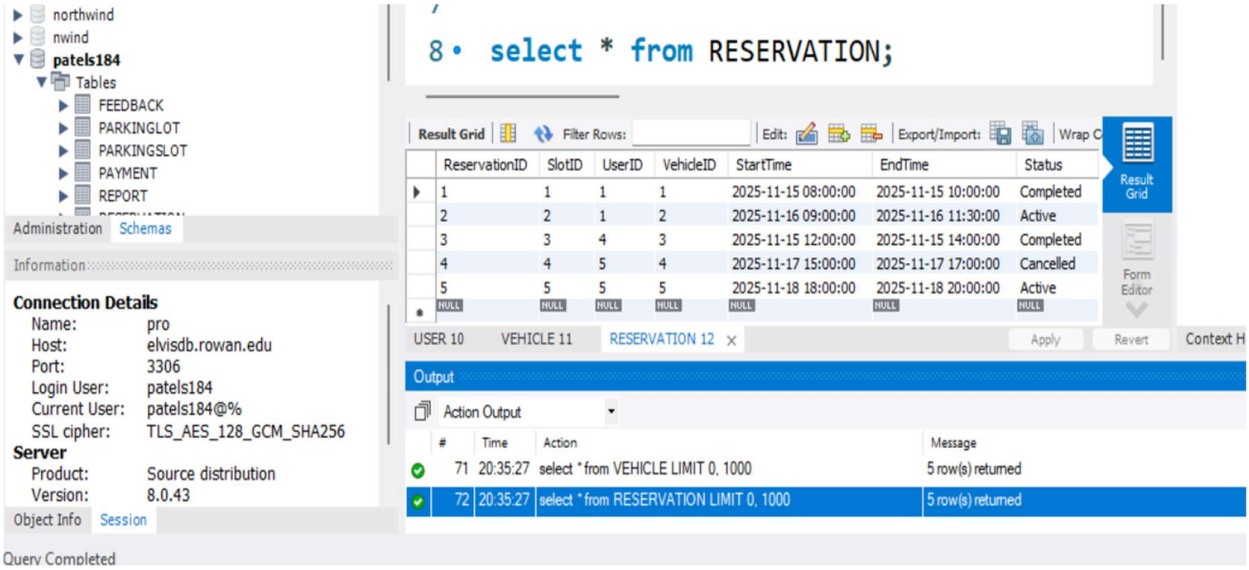
- Screenshot – PARKINGLOT table (5 records)



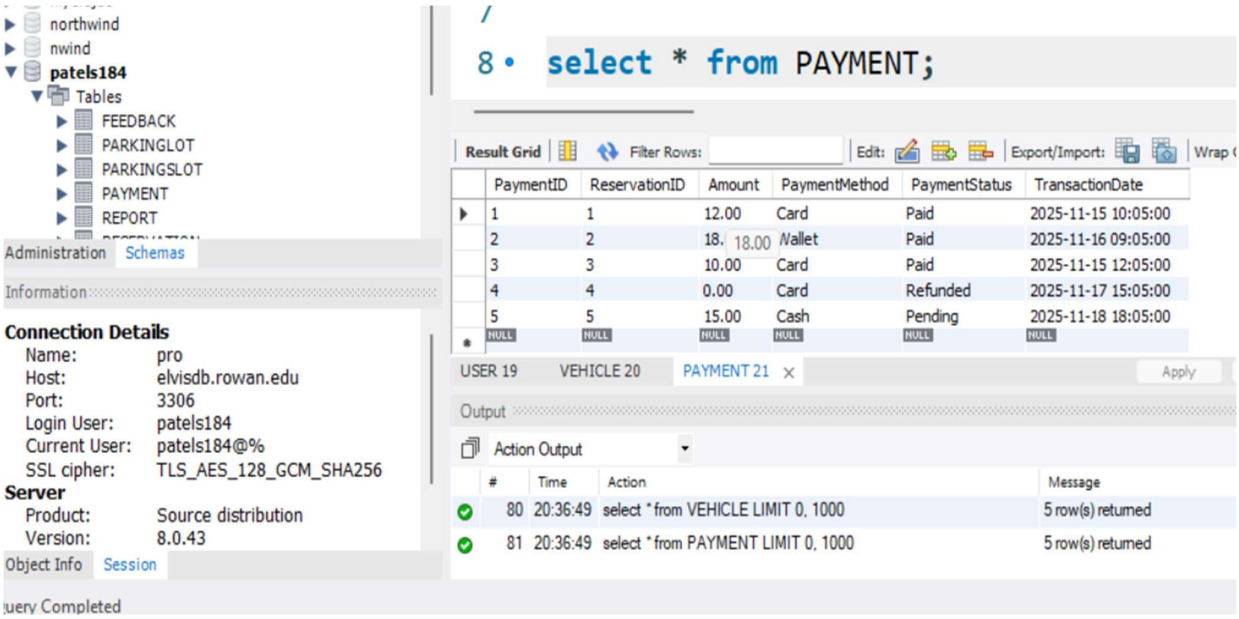
- Screenshot – PARKINGSLOT table (5 records)



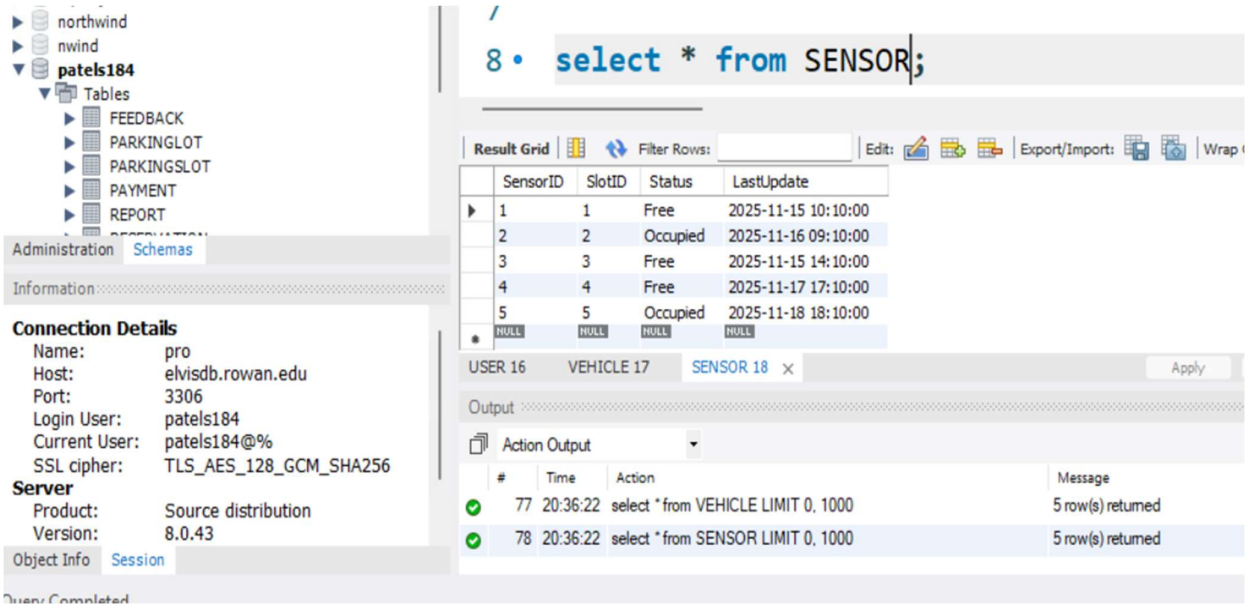
- Screenshot – RESERVATION table (5 records)



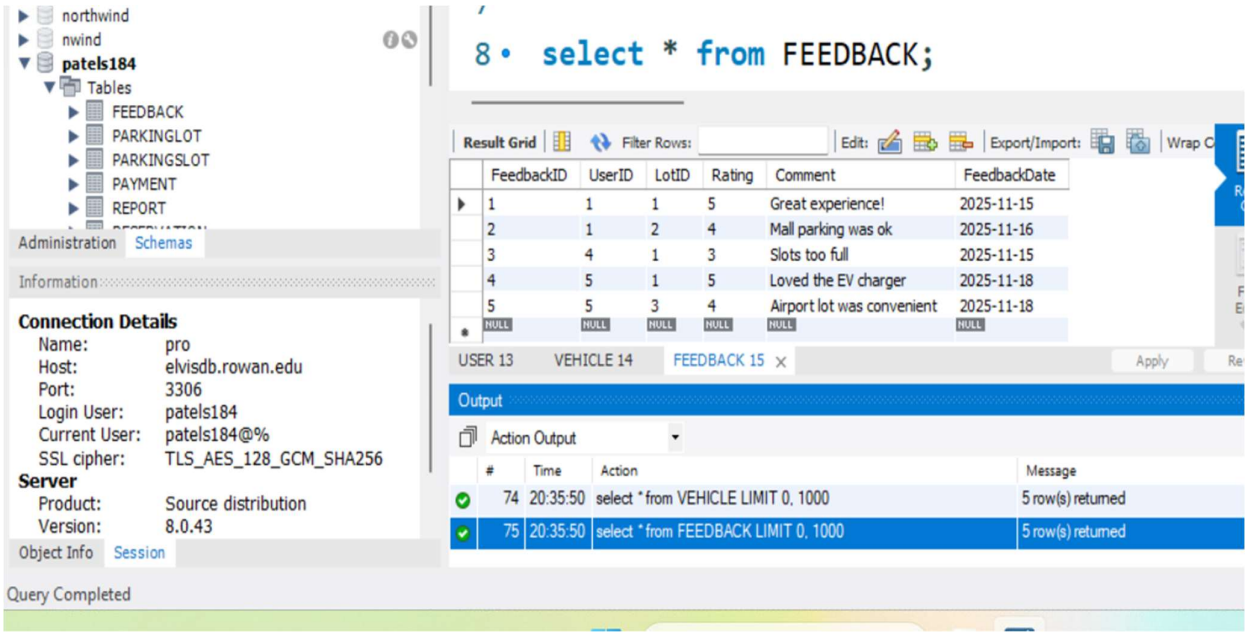
- Screenshot – PAYMENT table (5 records)



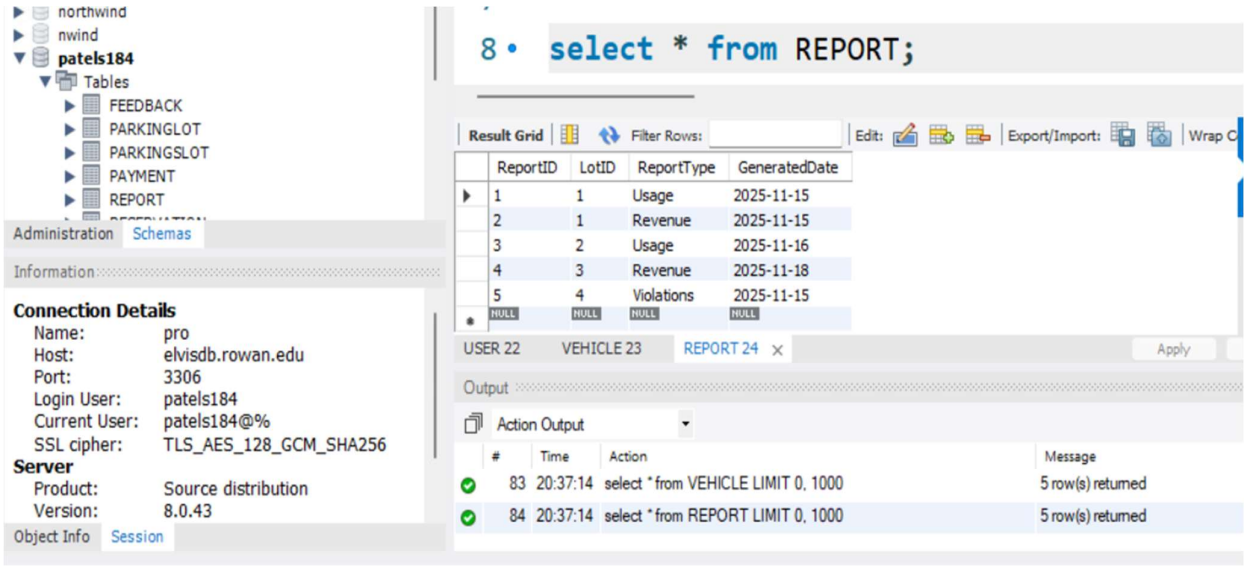
- Screenshot – SENSOR table (5 records)



- Screenshot – FEEDBACK table (5 records)



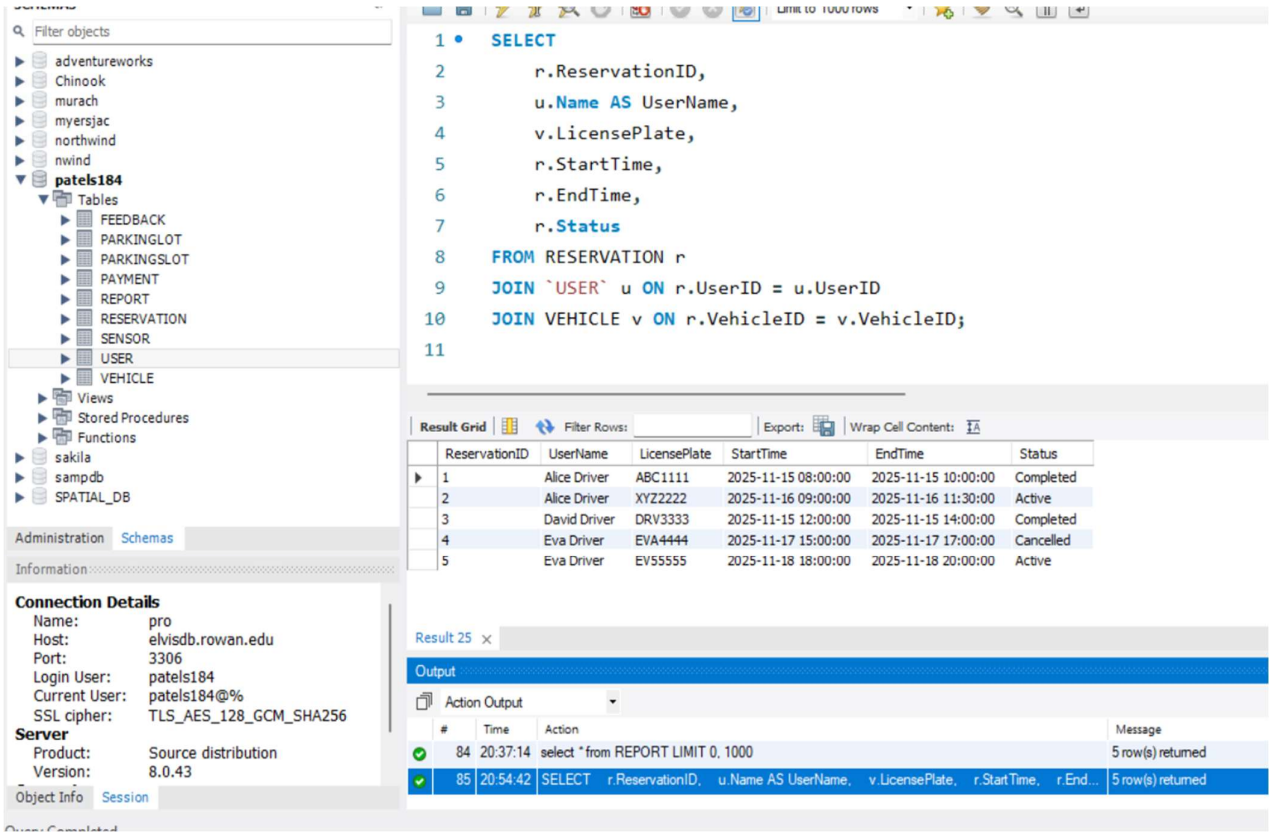
- Screenshot – REPORT table (5 records)



3.4 SQL JOIN Queries

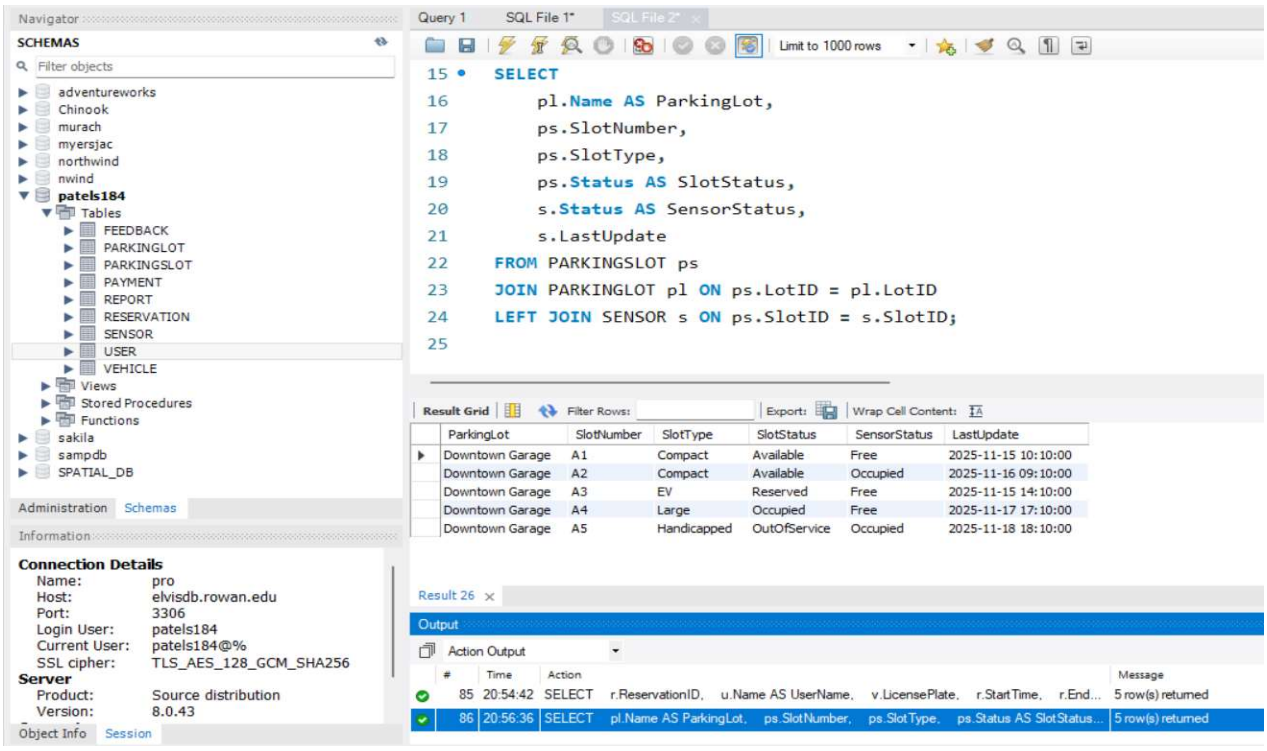
- JOIN Query 1

```
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;
```



- 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;
```

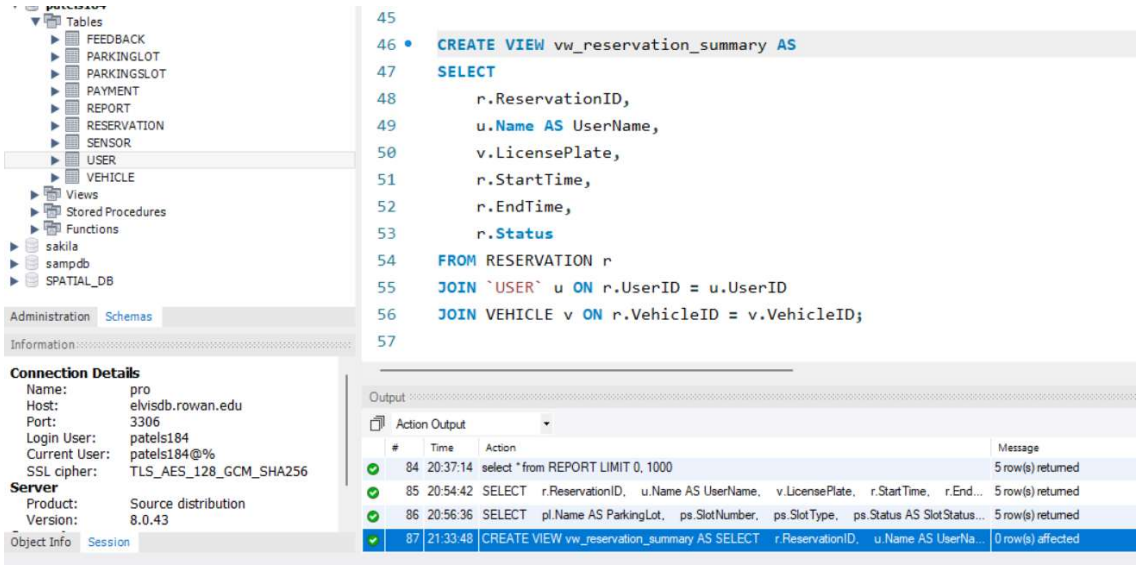


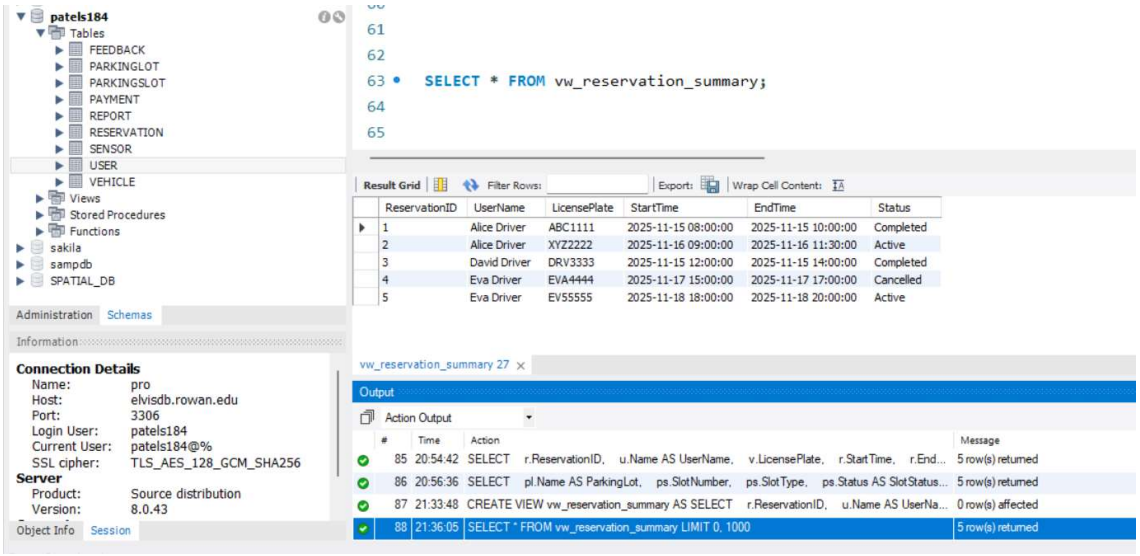
3.5 Database Objects

- Views (2) – Code + Results

VIEW 1:

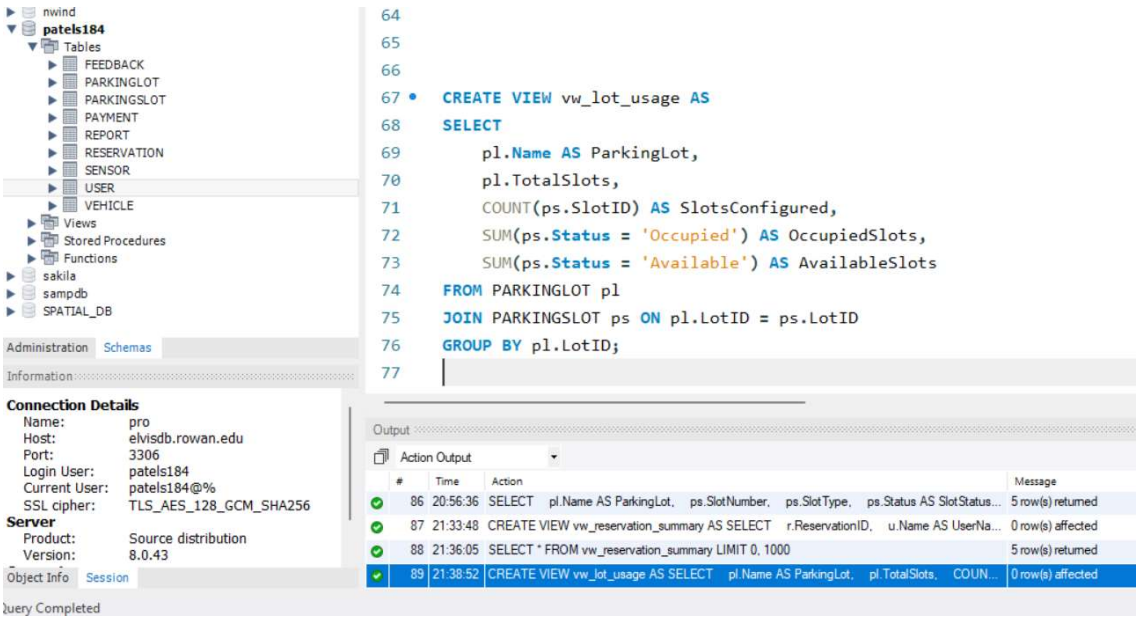
```
CREATE VIEW vw_reservation_summary AS
SELECT
    r.ReservationID,
    u.Name AS UserName,
    v.LicensePlate,
    r.StartTime,
    r.EndTime,
    r.Status
FROM RESERVATION r
JOIN `USER` u ON r.UserID = u.UserID
JOIN VEHICLE v ON r.VehicleID = v.VehicleID;
```

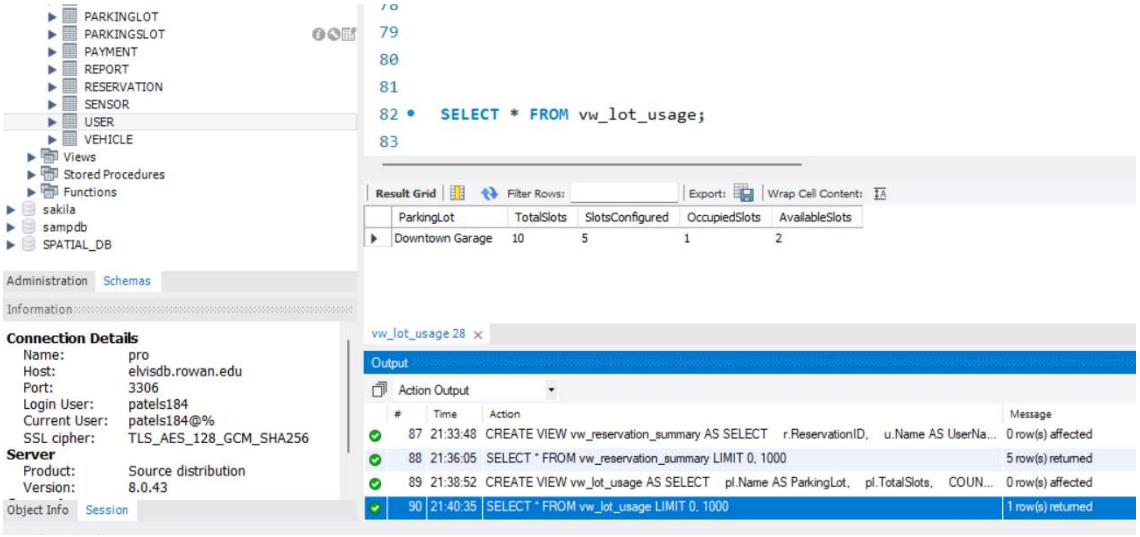




VIEW 2:

```
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;
```



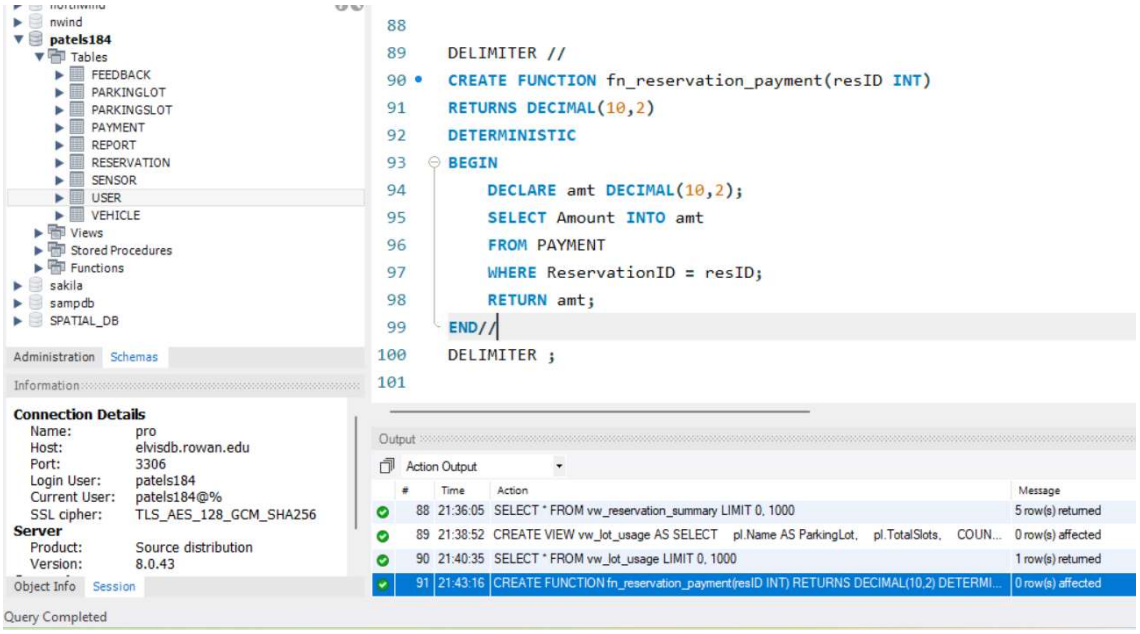


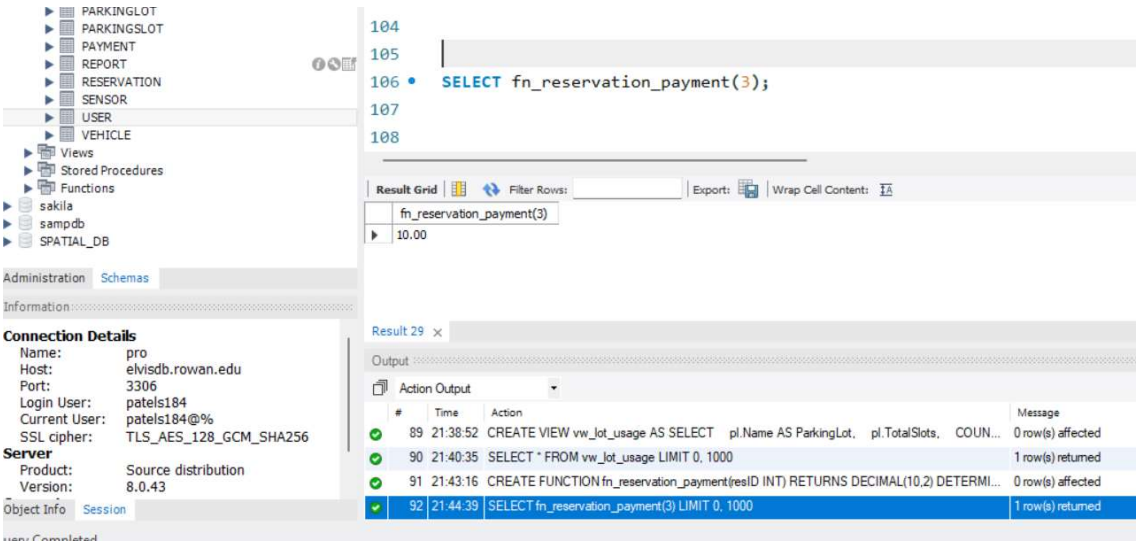
- Functions (2) – Code + Output

FUNCTION 1:

```
DELIMITER //
CREATE FUNCTION fn_reservation_payment(resID INT)
RETURNS DECIMAL(10,2)
```

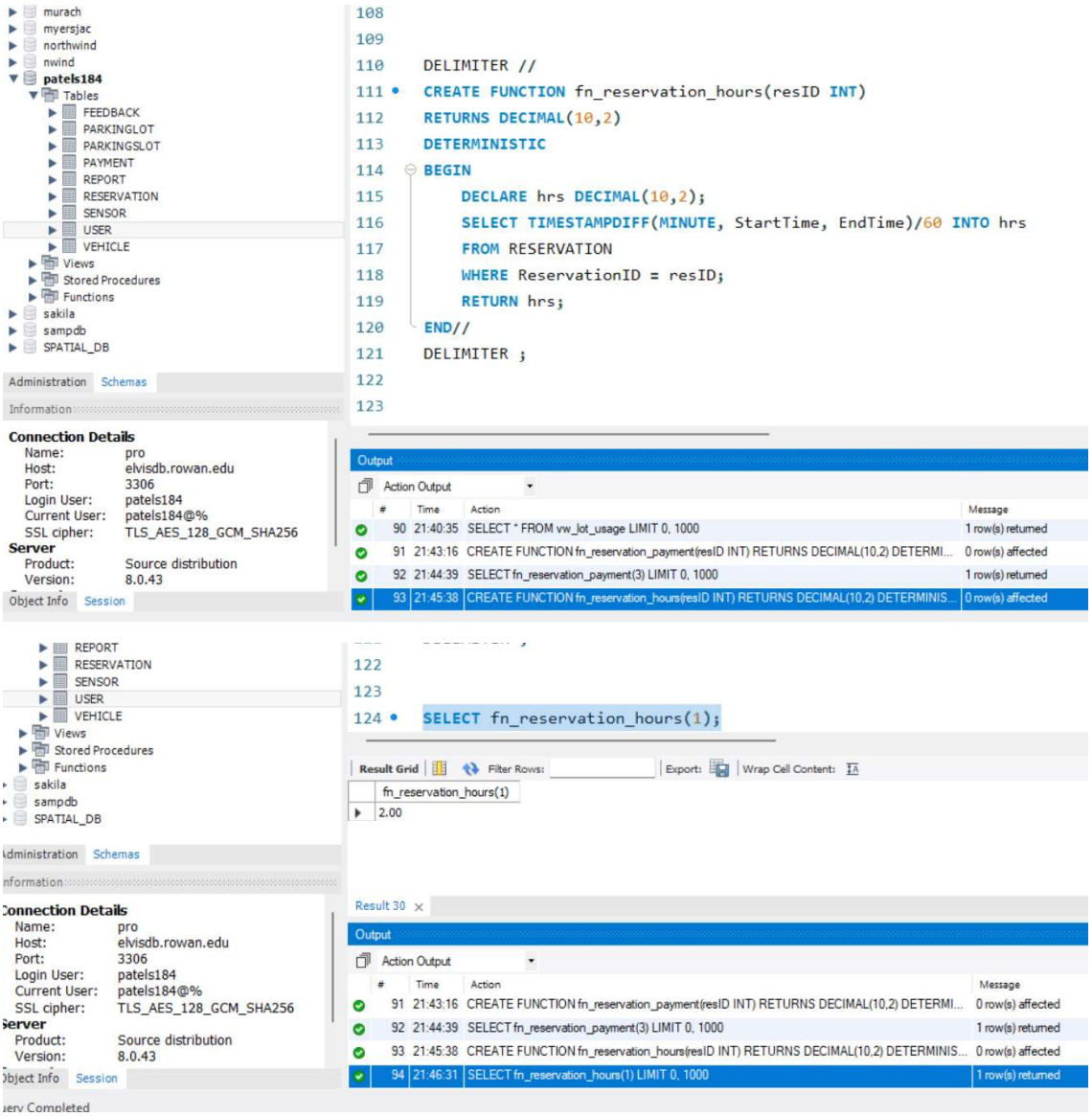
```
DETERMINISTIC
BEGIN
    DECLARE amt DECIMAL(10,2);
    SELECT Amount INTO amt
    FROM PAYMENT
    WHERE ReservationID = resID;
    RETURN amt;
END//
DELIMITER ;
```





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 ;
```

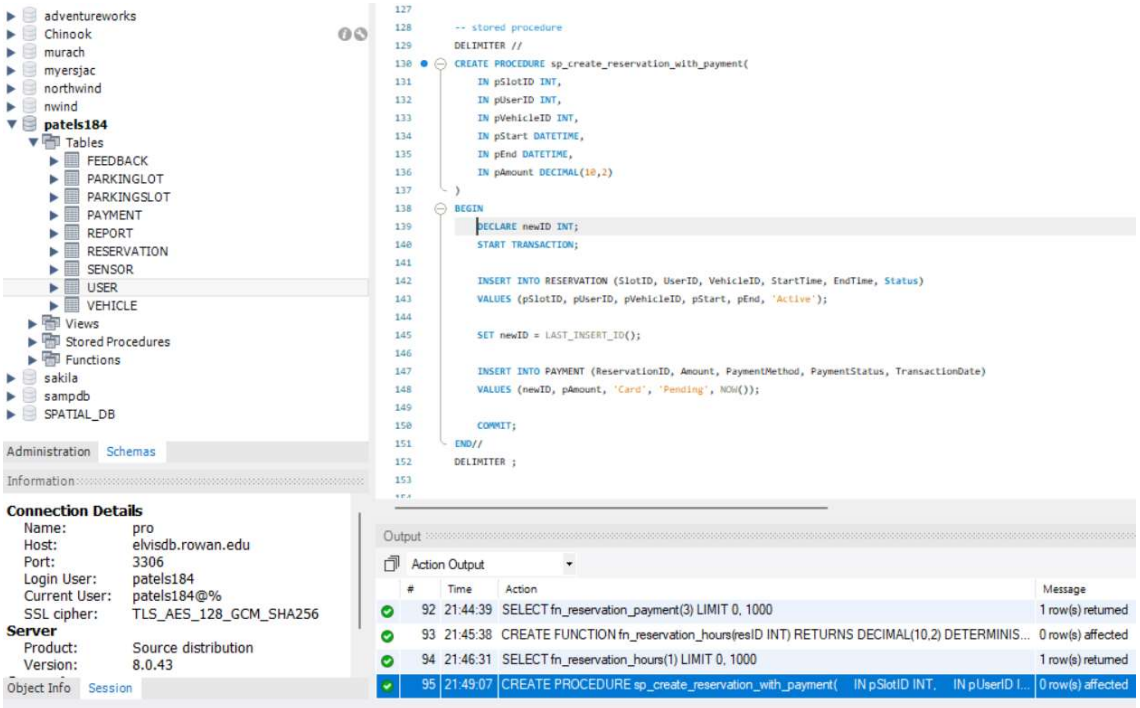


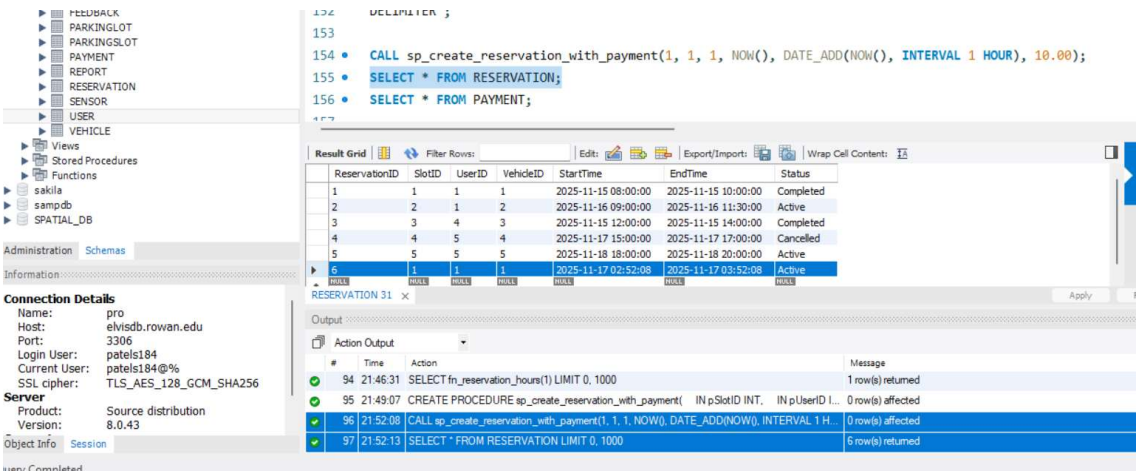
- Stored Procedures (2) – Code + Execution

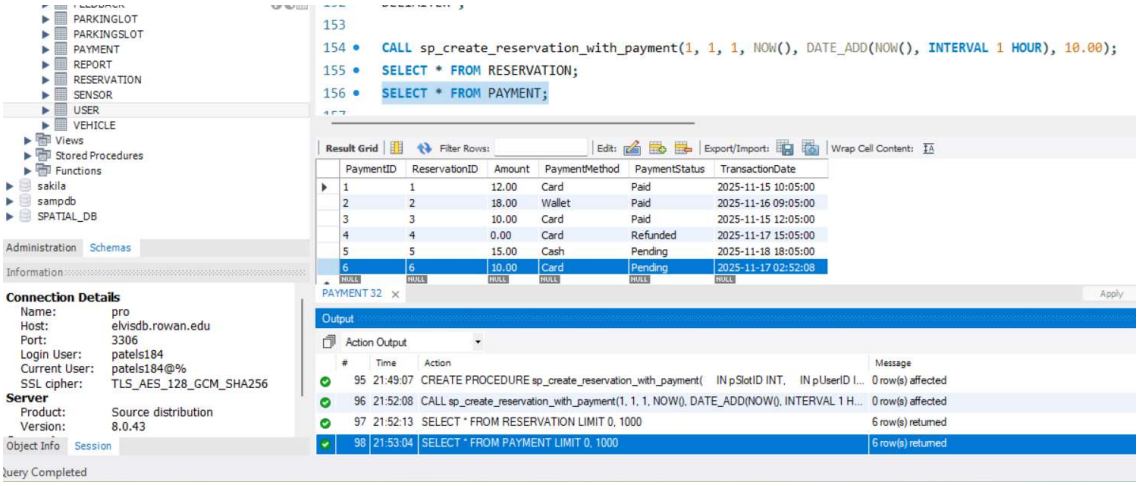
STORED PROCEDURE 1:

```
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 ;
```

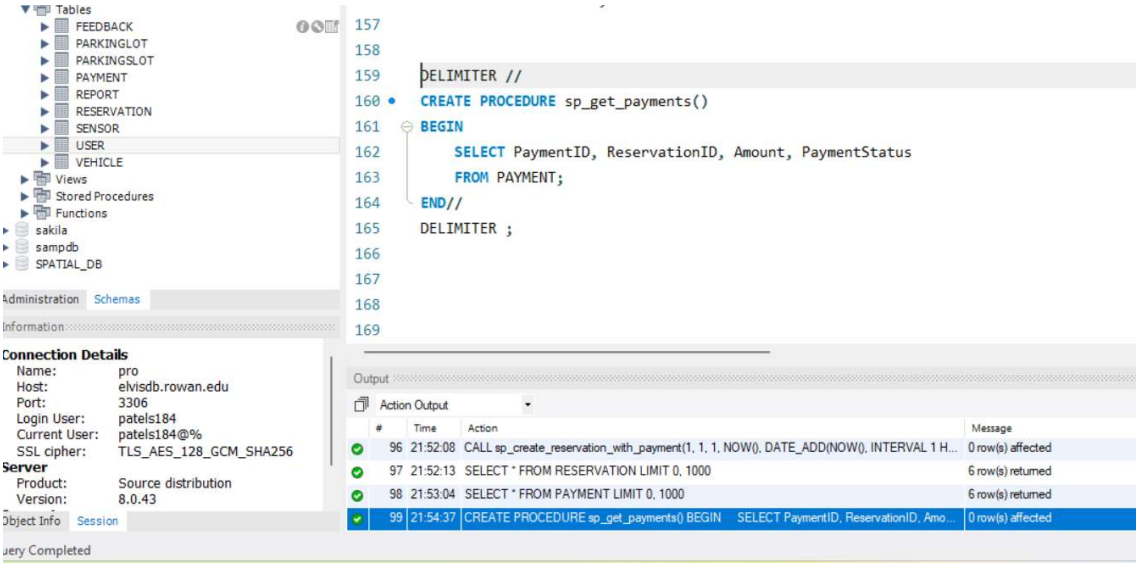

adventureworks
Chinook
murach
myersjac
northwind
nwind
patels184
Tables
FEEDBACK
PARKINGLOT
PARKINGSLOT
PAYMENT
REPORT
RESERVATION
SENSOR
USER
VEHICLE
Views
Stored Procedures
Functions
sakila
sampdb
SPATIAL_DB
Administration Schemas
Information
Connection Details
Name: pro
Host: elvisdb.rowan.edu
Port: 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
-- stored procedure
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 ;
Output
Action 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 pUserID I... 0 row(s) affected

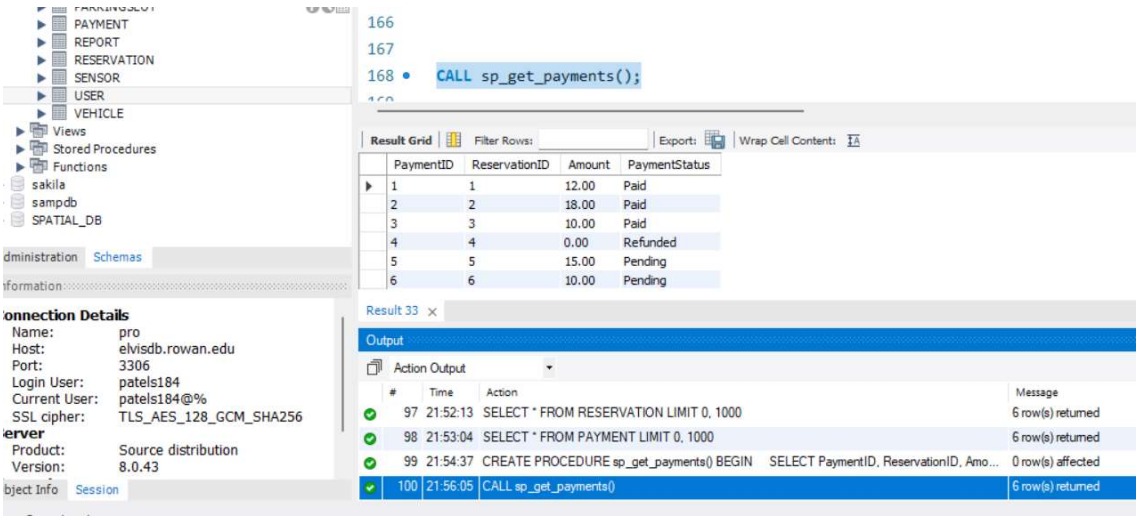

CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW(), INTERVAL 1 HOUR), 10.00);
SELECT * FROM RESERVATION;
SELECT * FROM PAYMENT;
Result Grid
ReservationID SlotID UserID VehicleID StartTime EndTime Status
1 1 1 1 2025-11-15 08:00:00 2025-11-15 10:00:00 Completed
2 2 1 2 2025-11-16 09:00:00 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
RESERVATION 31
Output
Action Output
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 pUserID I... 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


CALL sp_create_reservation_with_payment(1, 1, 1, NOW(), DATE_ADD(NOW(), INTERVAL 1 HOUR), 10.00);
SELECT * FROM RESERVATION;
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
PAYMENT 32
Output
Action Output
95 21:49:07 CREATE PROCEDURE sp_create_reservation_with_payment(IN pSlotID INT, IN pUserID I... 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 ;
```

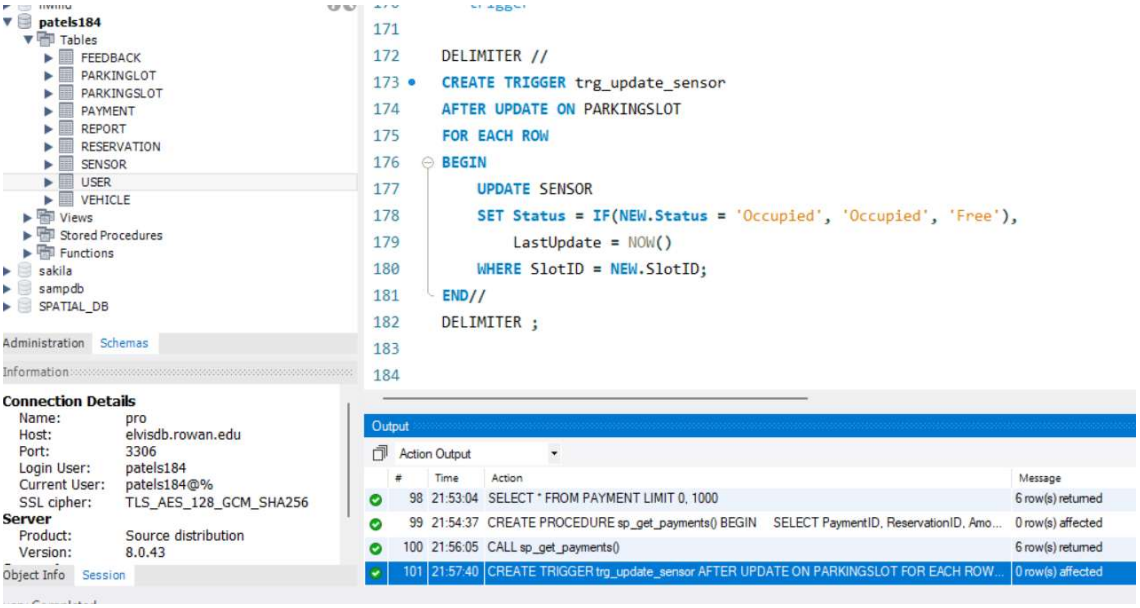


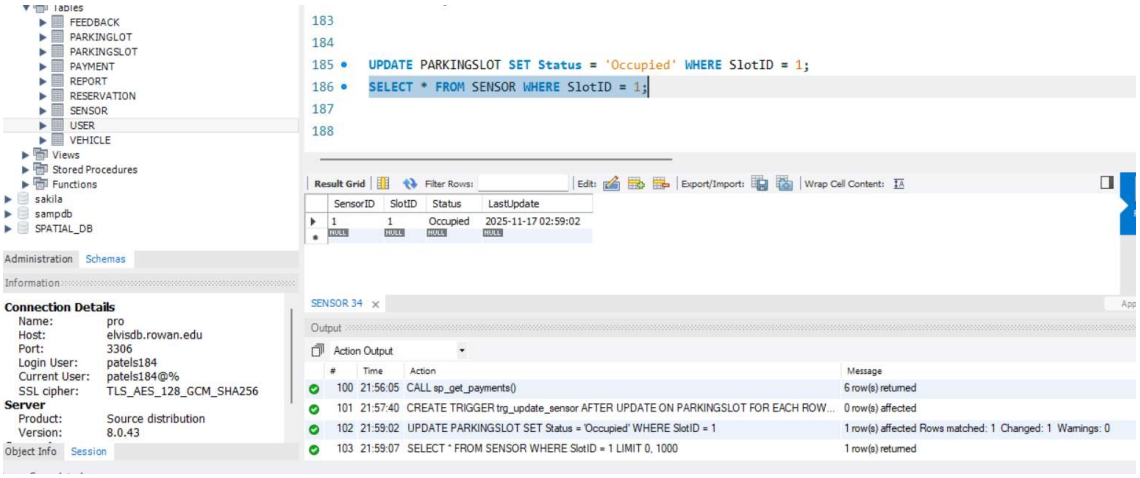


- Triggers (2) – Code + Validation Output

TRIGGER 1:

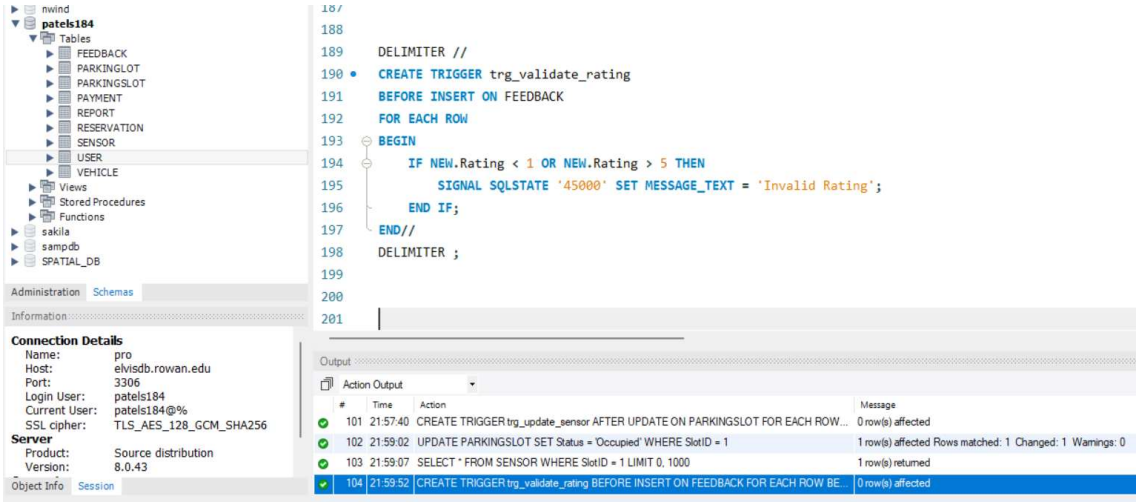
```
DELIMITER //
CREATE TRIGGER trg_update_sensor
AFTER UPDATE ON PARKINGSLOT
FOR EACH ROW
BEGIN
    UPDATE SENSOR
    SET Status = IF(NEW.Status = 'Occupied', 'Occupied', 'Free'),
        LastUpdate = NOW()
    WHERE SlotID = NEW.SlotID;
END//
DELIMITER ;
```

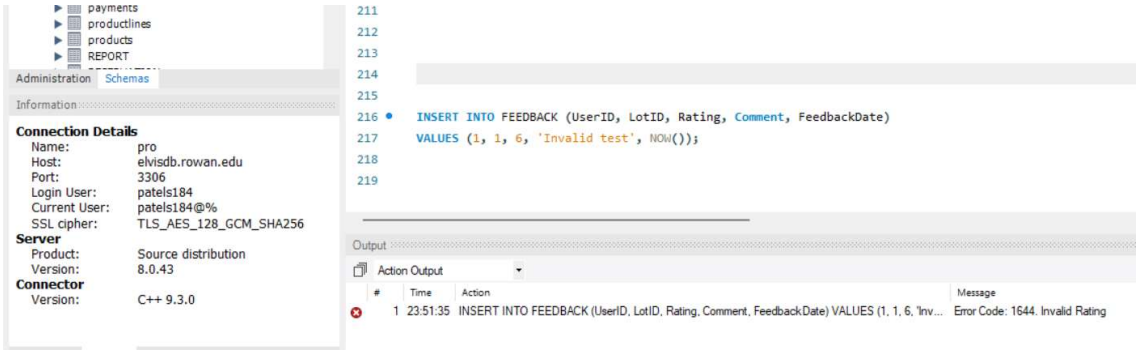




TRIGGER 2:

```
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 ;
```





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

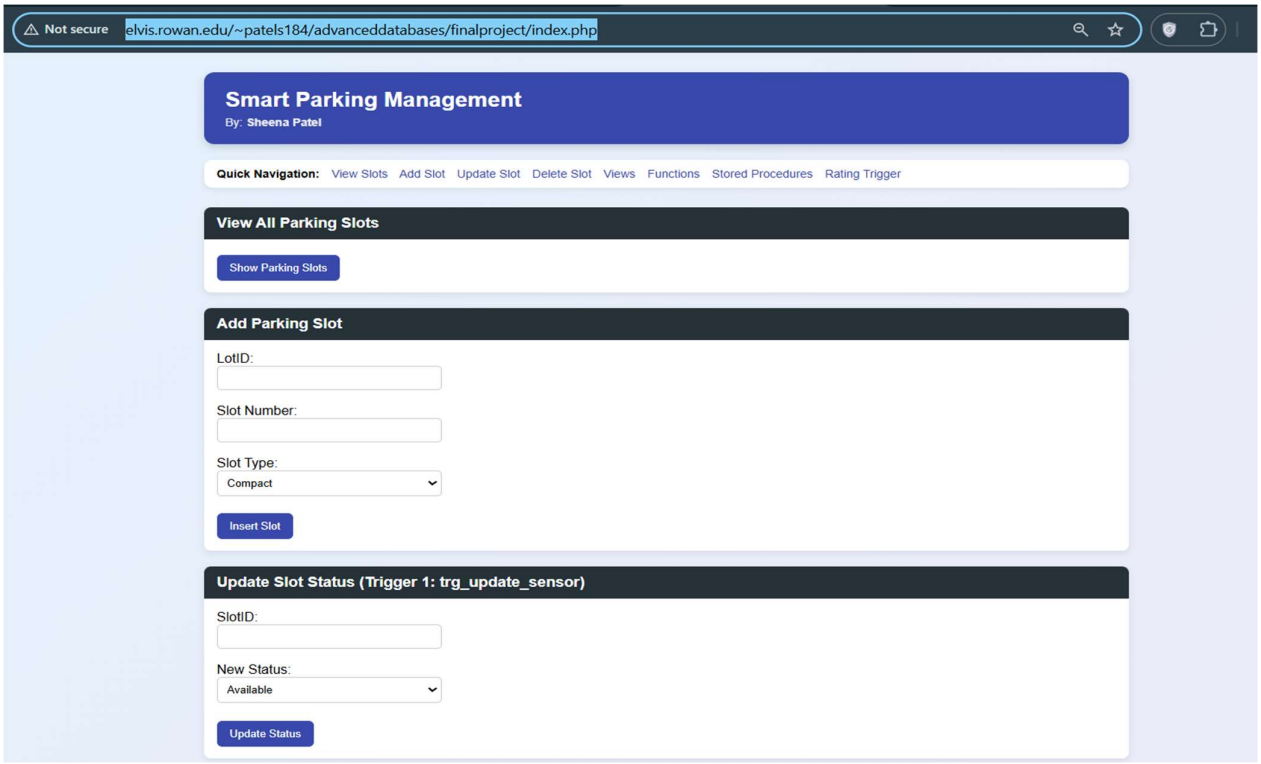
- Views: vw_reservation_summary, vw_lot_usage
- Functions: fn_reservation_payment, fn_reservation_hours
- Stored Procedures: sp_create_reservation_with_payment, sp_get_payments
- Triggers:
    - trg_update_sensor (auto-updates SENSOR when a slot is updated)

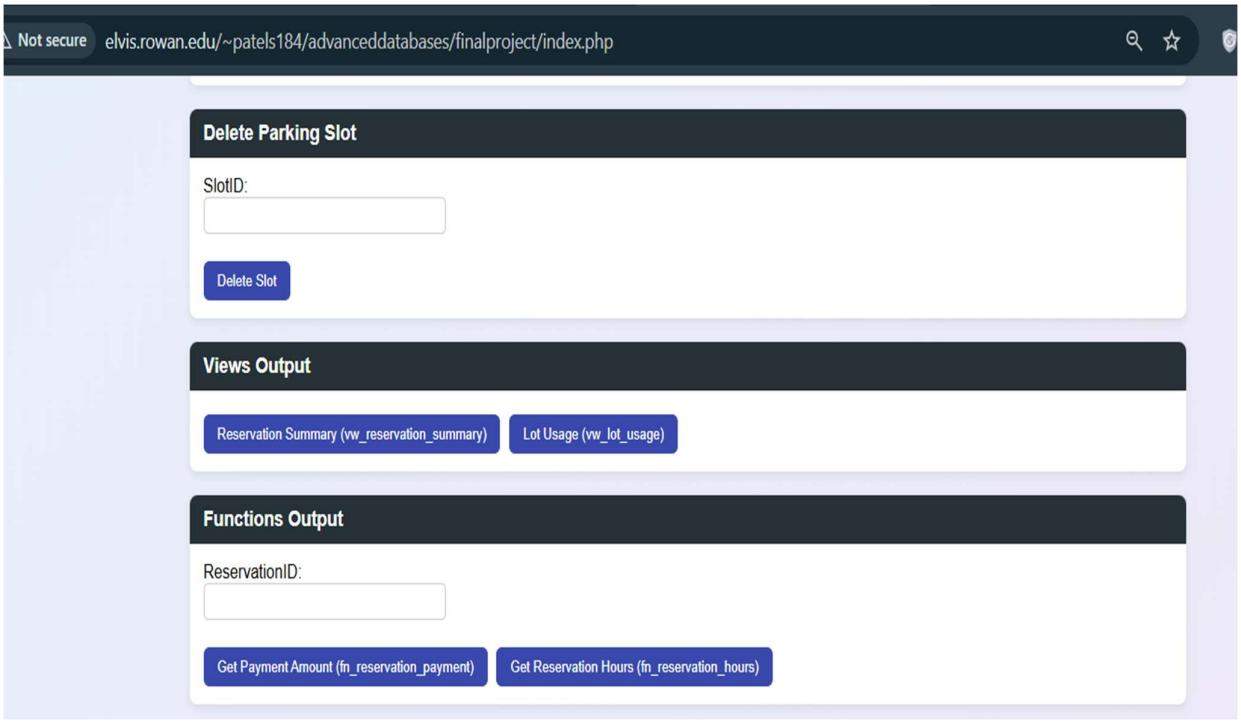
- trg_validate_rating (rejects invalid ratings before insert)

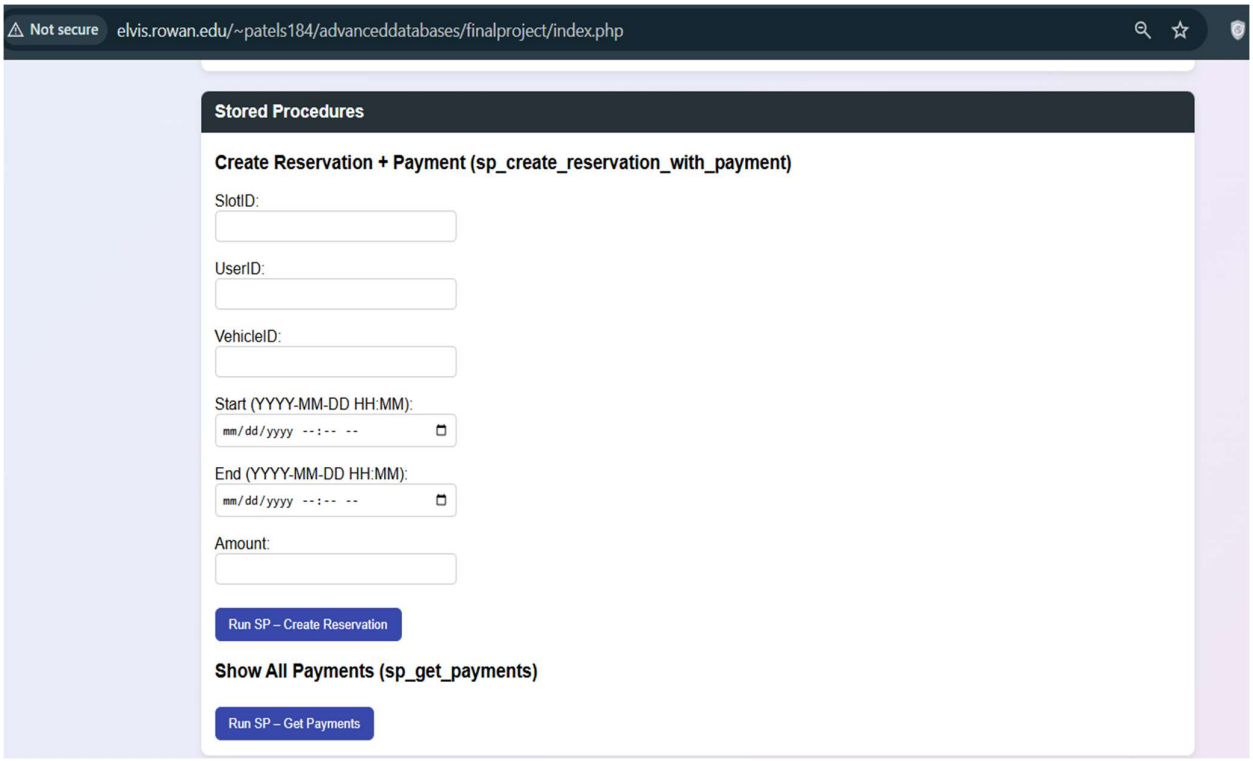
Frontend Implementation

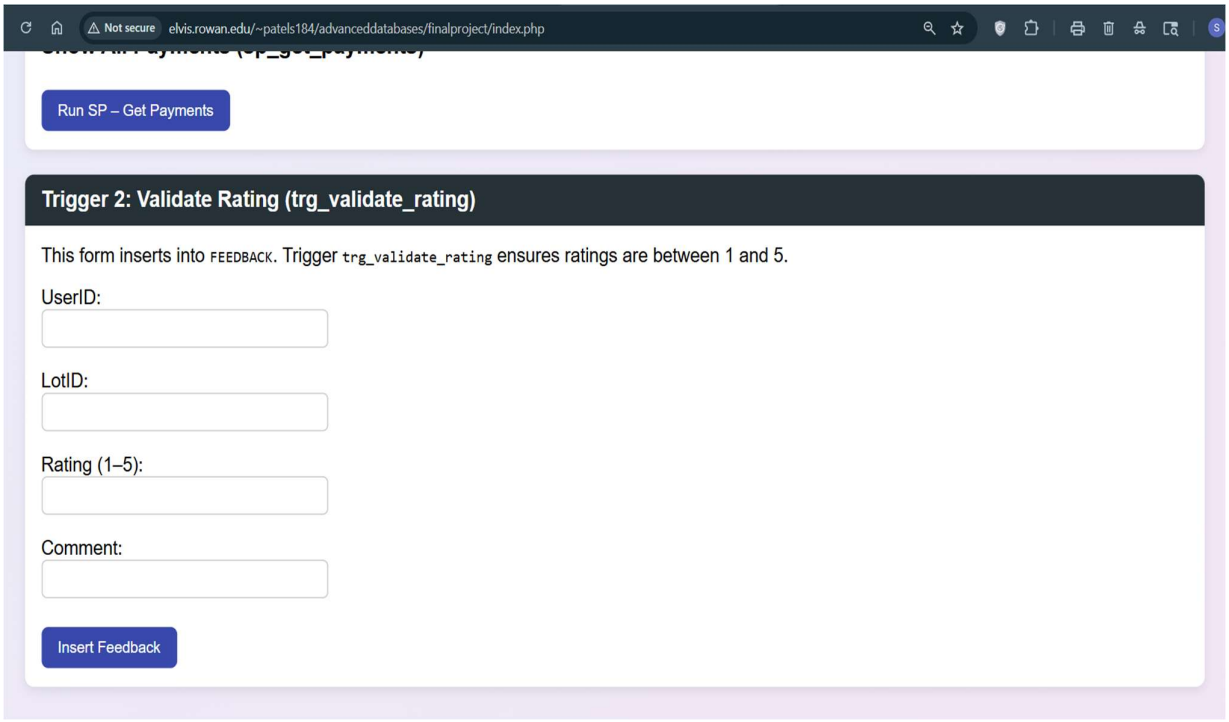
- Fully integrated single-page layout with section navigation
- Clean and intuitive UI redesign: gradient background, card-style sections, styled buttons
- All actions display confirmation messages and database results

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



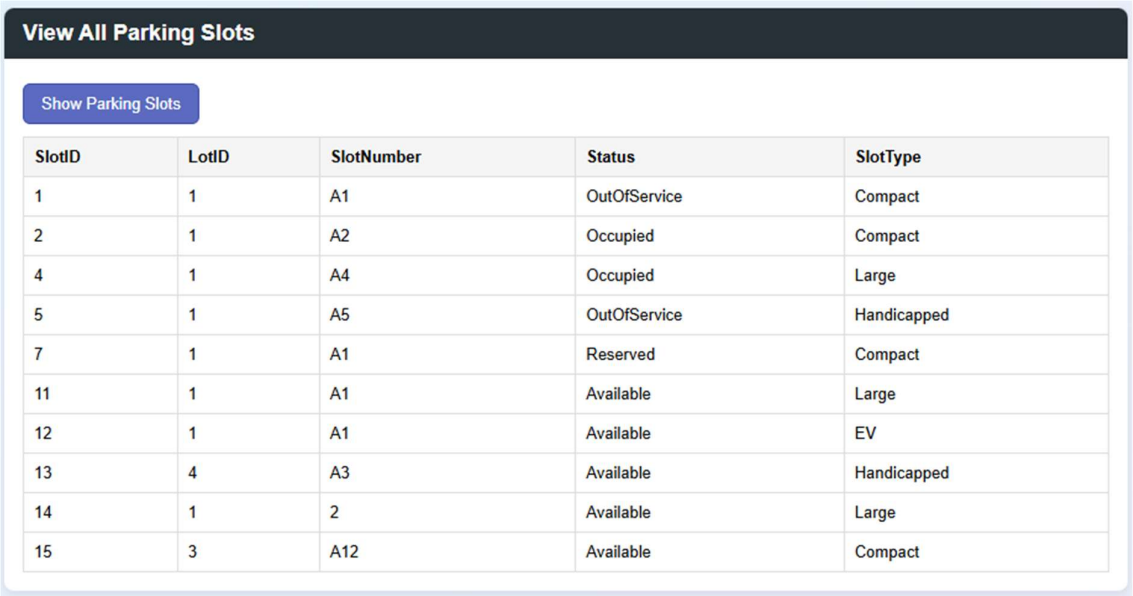

Not secure
elvis.rowan.edu/~patels184/advanceddatabases/finalproject/index.php
Delete Parking Slot
SlotID:
Delete Slot
Views Output
Reservation Summary (vw_reservation_summary)
Lot Usage (vw_lot_usage)
Functions Output
ReservationID:
Get Payment Amount (fn_reservation_payment)
Get Reservation Hours (fn_reservation_hours)


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


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

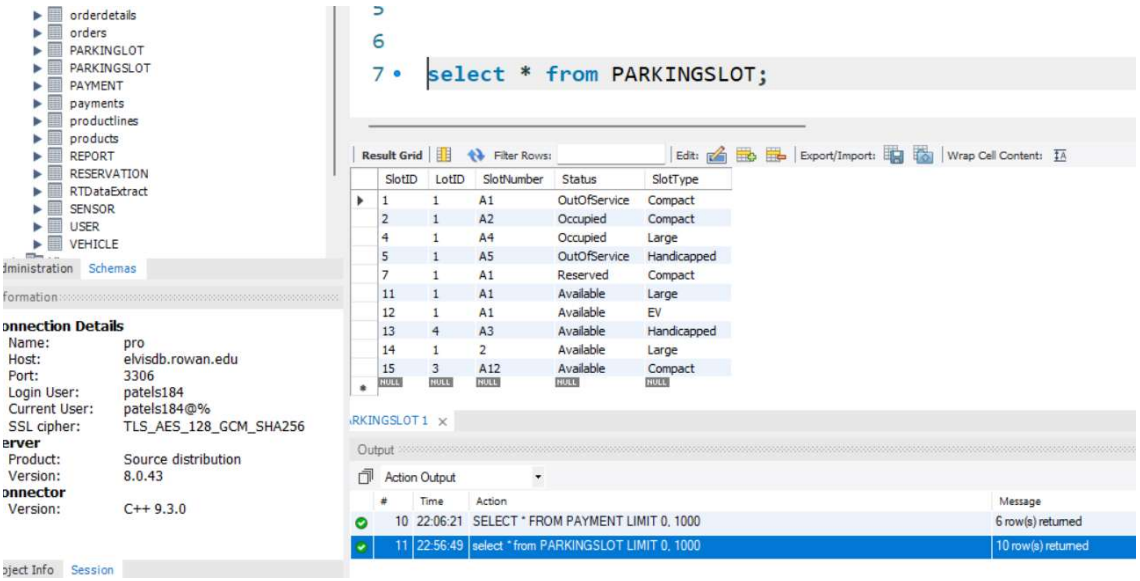
Displaying Records (SELECT):



View All Parking Slots

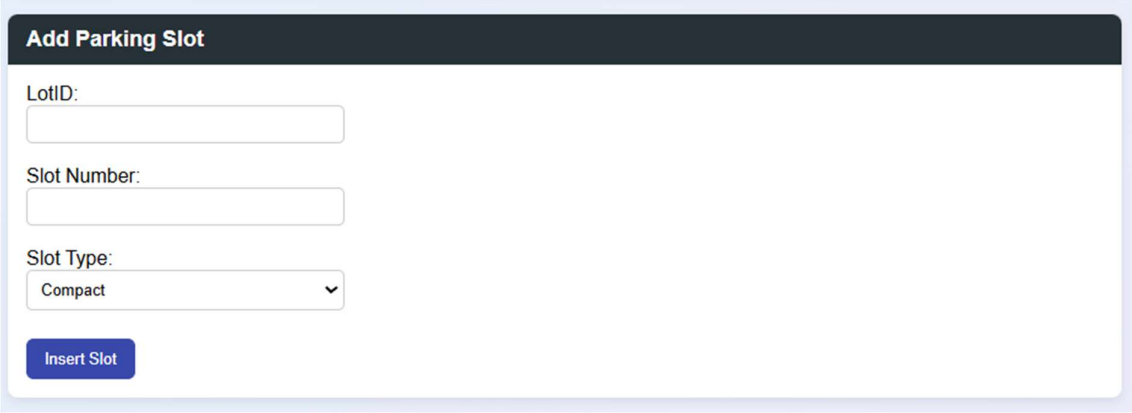
Show Parking Slots

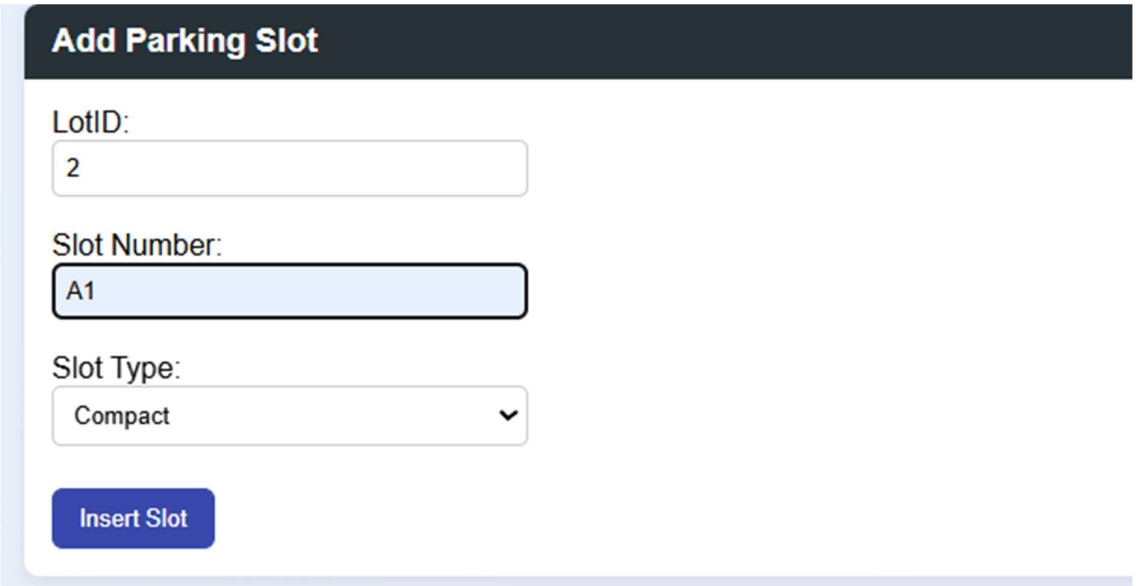
SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	OutOfService	Compact
2	1	A2	Occupied	Compact
4	1	A4	Occupied	Large
5	1	A5	OutOfService	Handicapped
7	1	A1	Reserved	Compact
11	1	A1	Available	Large
12	1	A1	Available	EV
13	4	A3	Available	Handicapped
14	1	2	Available	Large
15	3	A12	Available	Compact

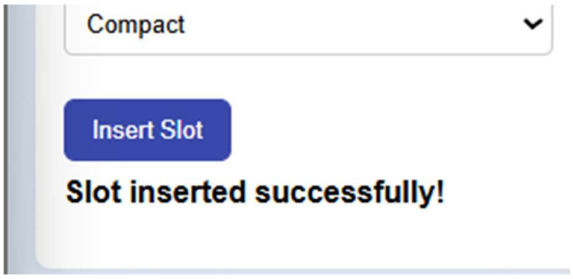


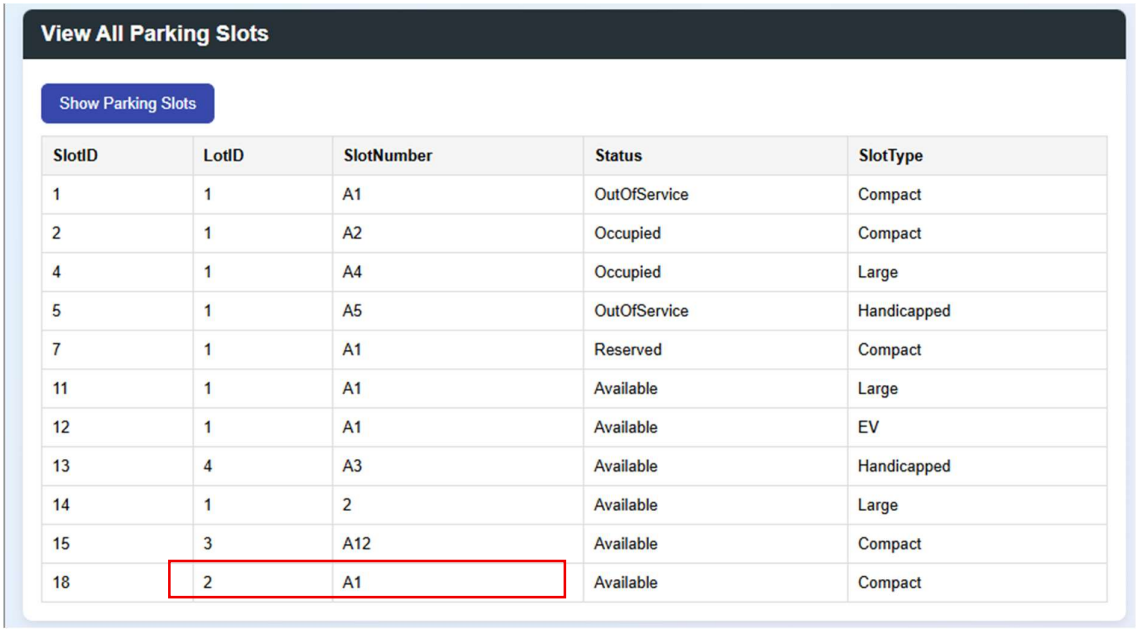
This screenshot shows all parking slots retrieved from the PARKINGSLOT table, demonstrating successful SELECT operation.

Insert Operation:





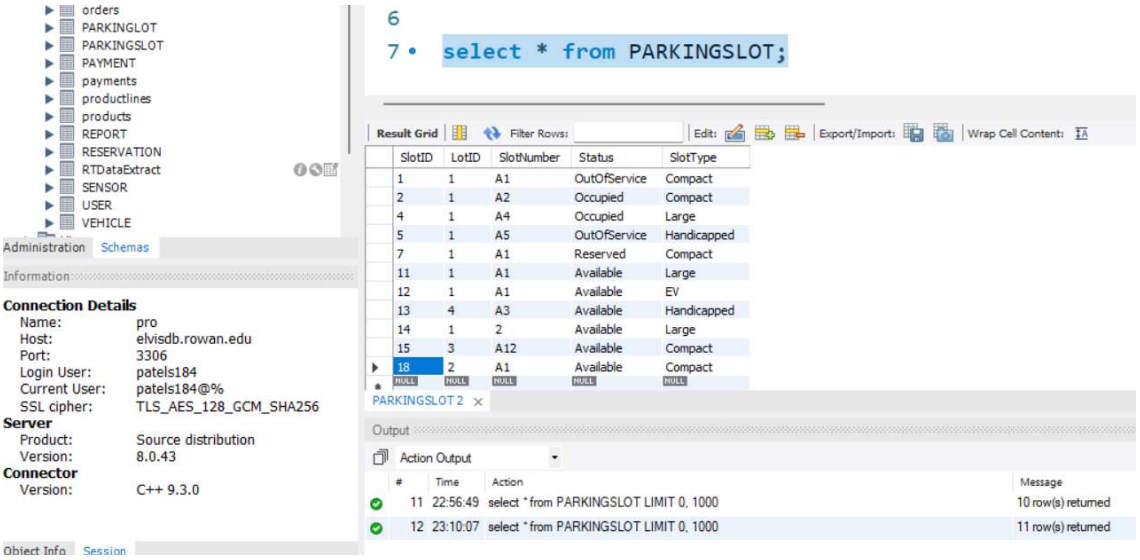




View All Parking Slots

Show Parking Slots

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



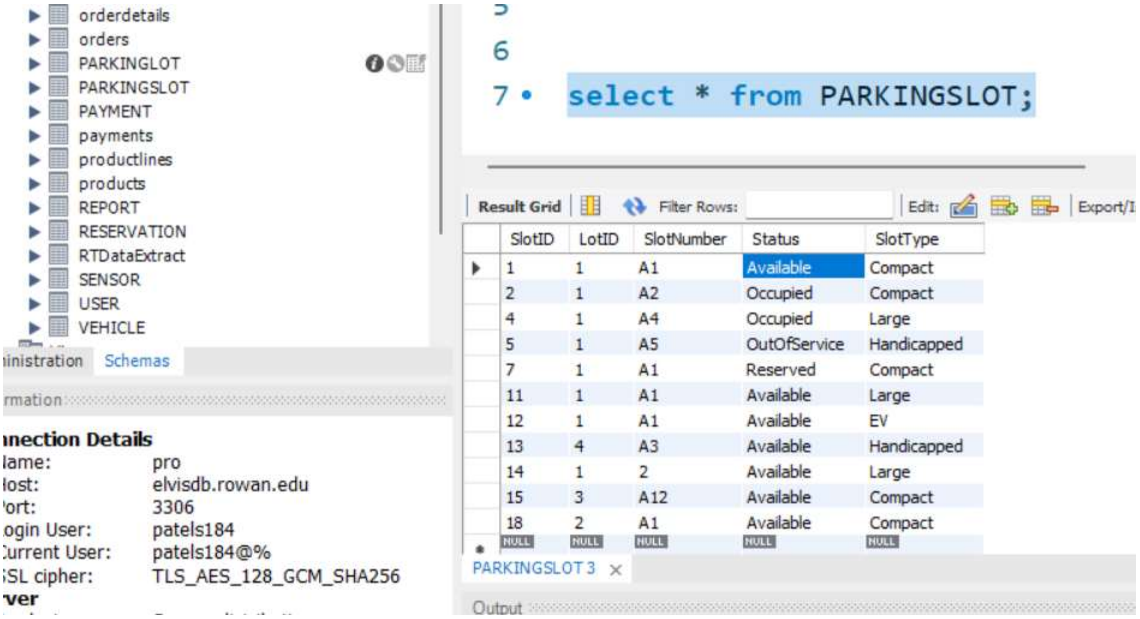
This screenshot confirms that a new parking slot was inserted successfully into the database using an INSERT statement.

Update Operation:

View All Parking Slots

Show Parking Slots

SlotID	LotID	SlotNumber	Status	SlotType
1	1	A1	Available	Compact



Update Slot Status (Trigger 1: trg_update_sensor)

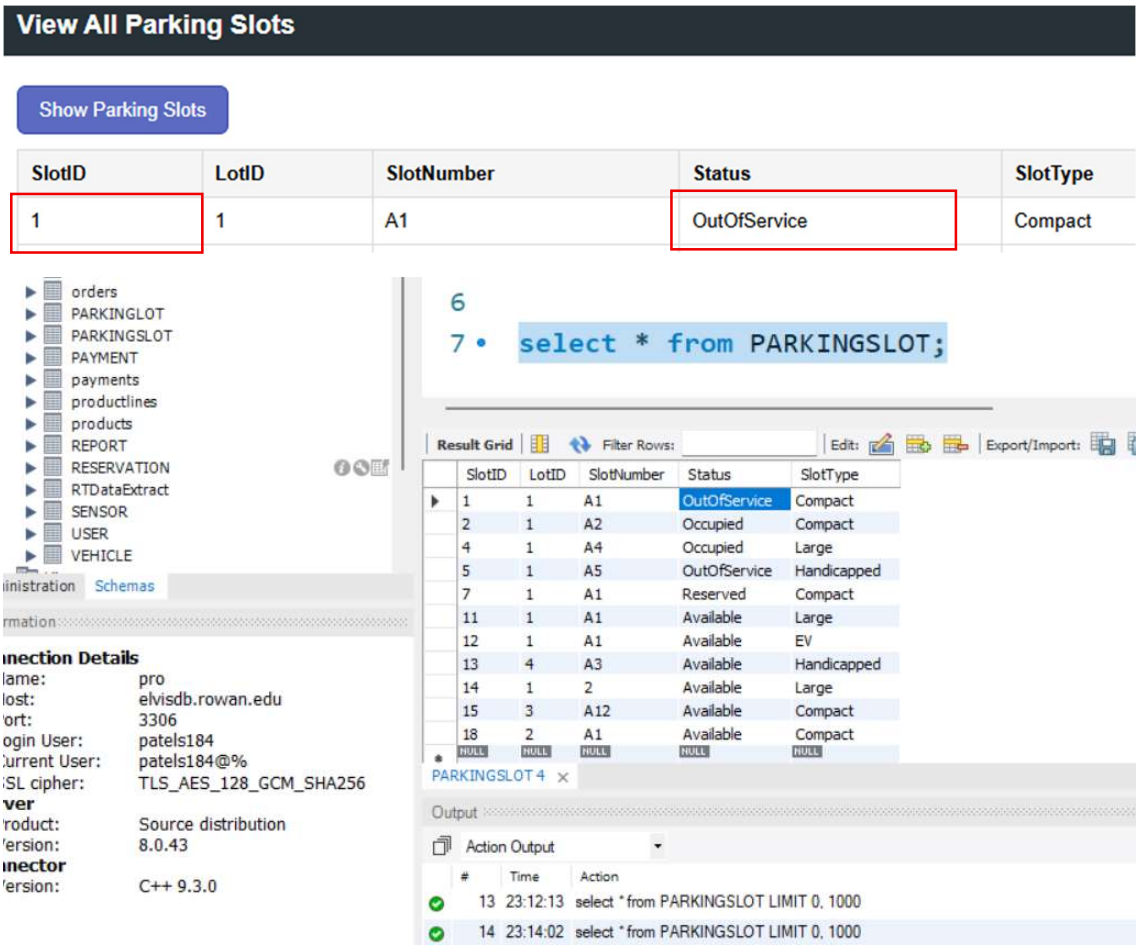
SlotID:

1

New Status:

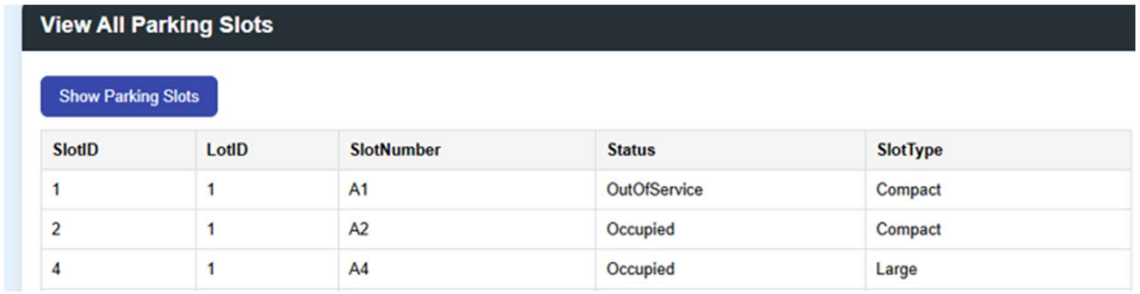
OutOfService

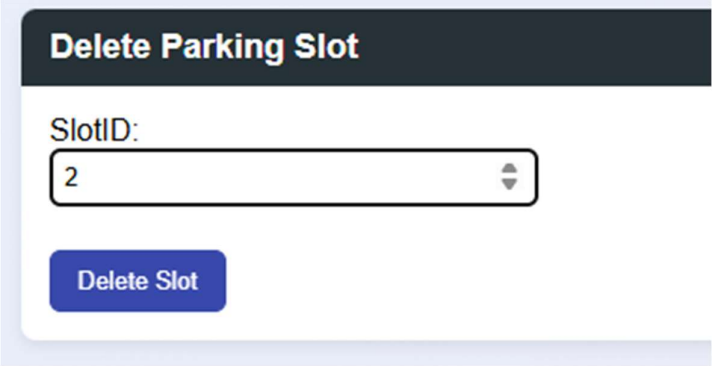
Update Status

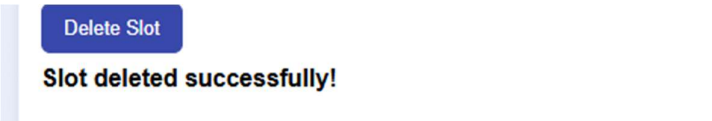


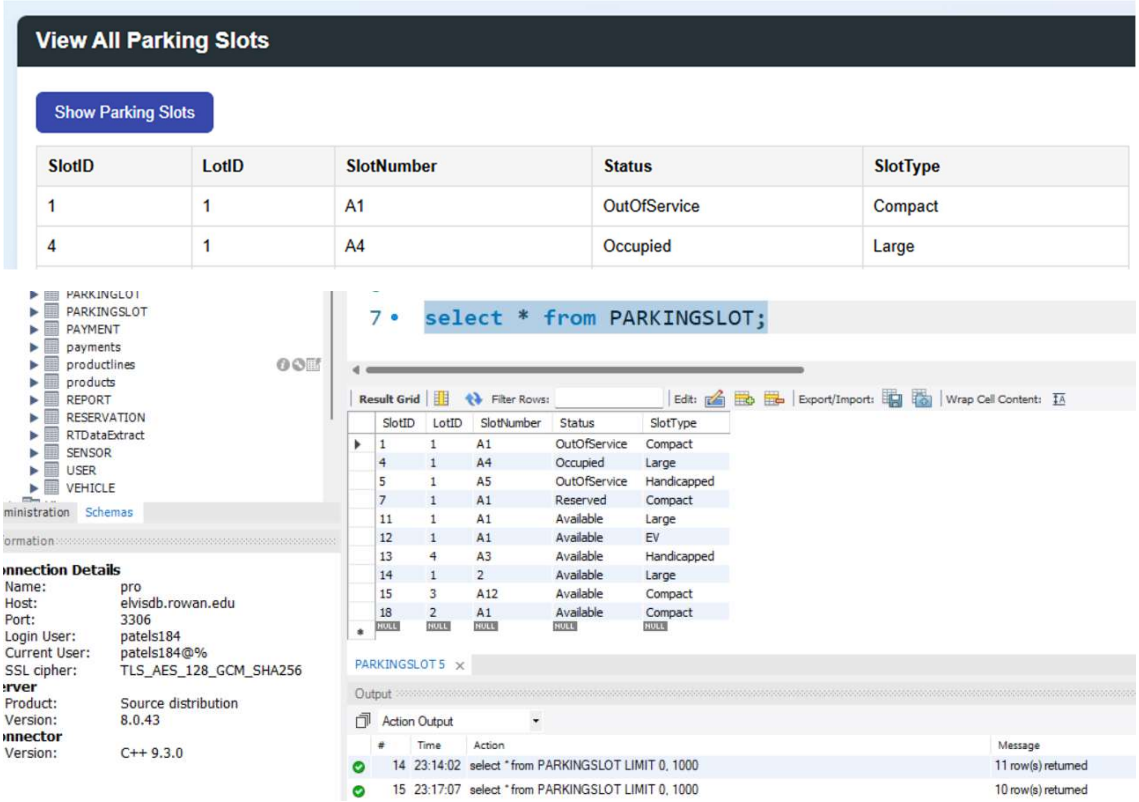
This screenshot shows a slot status being updated.

Delete Operation:





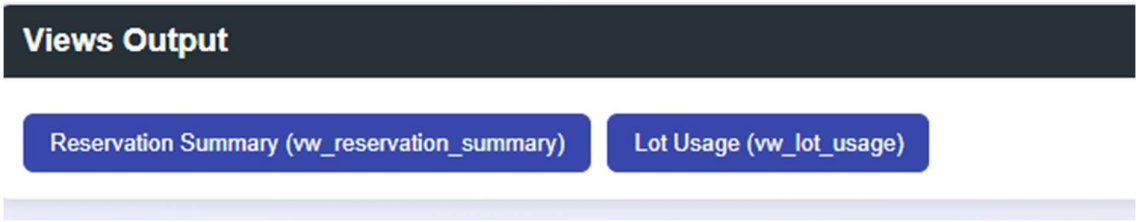




This screenshot demonstrates the DELETE operation where the selected parking slot was removed from the system.

4.3 Views, Functions, Stored Procedures, Triggers Output

Views Output

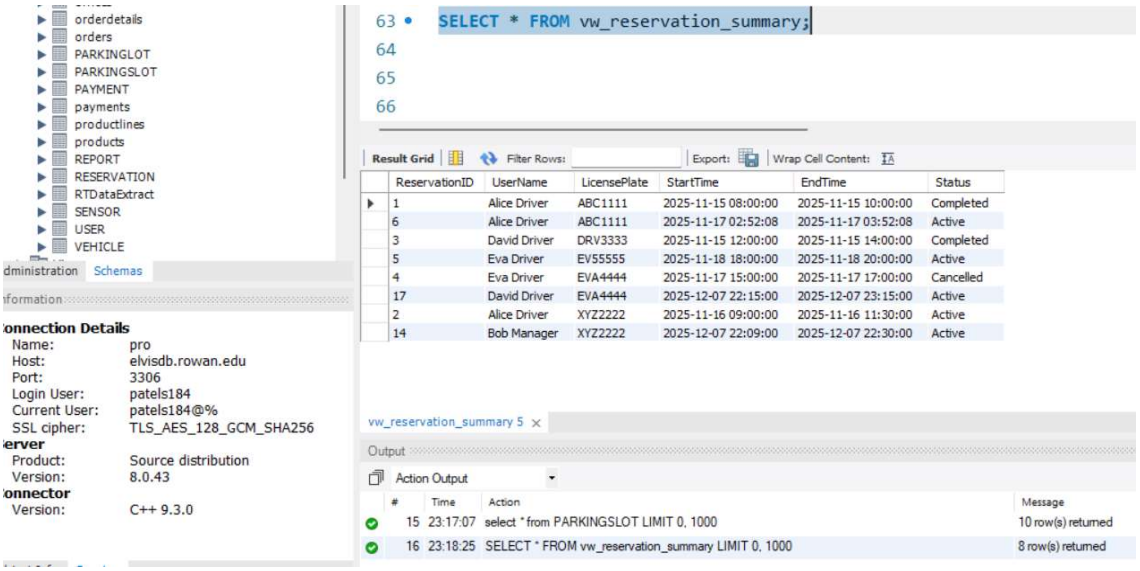


View 1:

Views Output

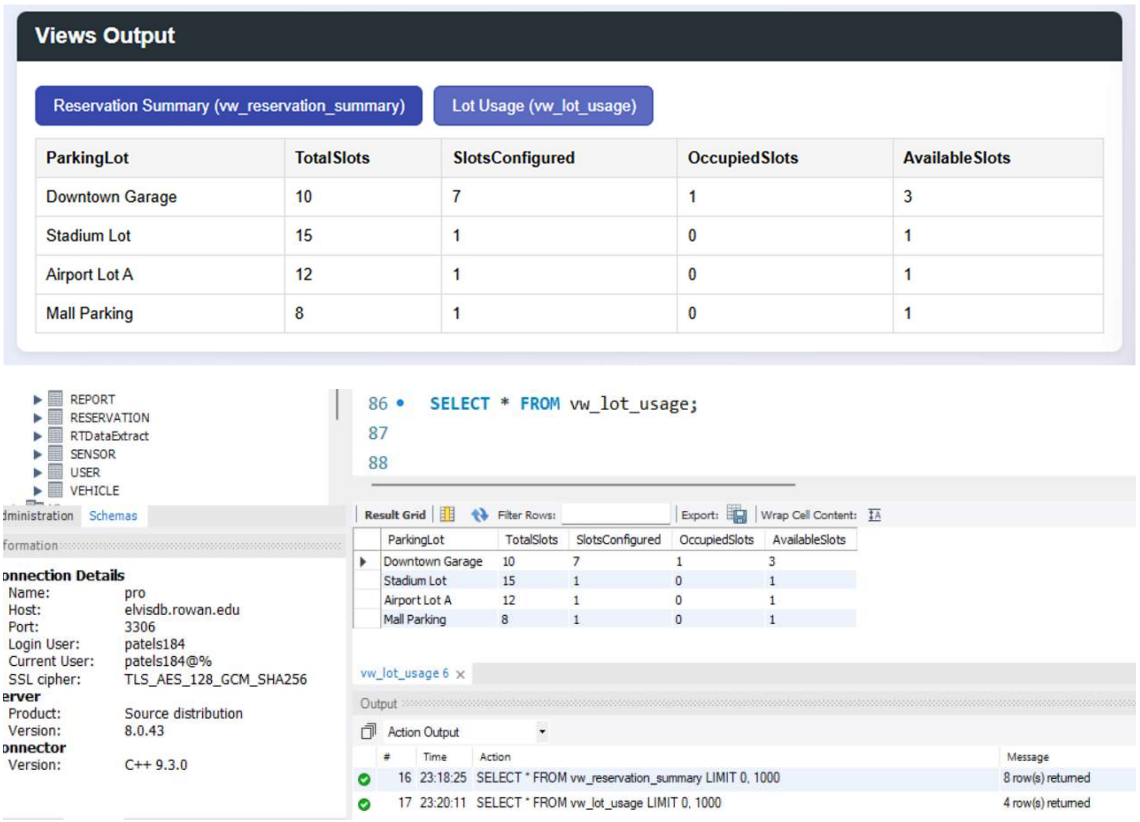
Reservation Summary (vw_reservation_summary) | Lot Usage (vw_lot_usage)

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



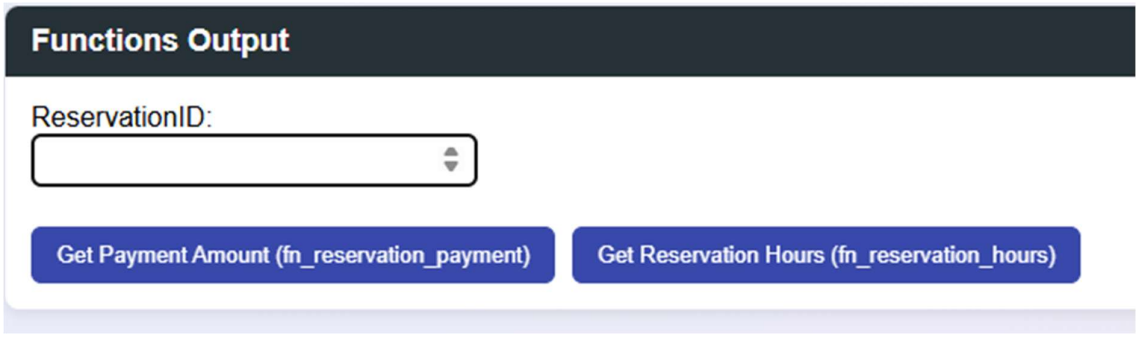
This screenshot displays the output of the reservation summary view, showing joined reservation and user data.

View 2:

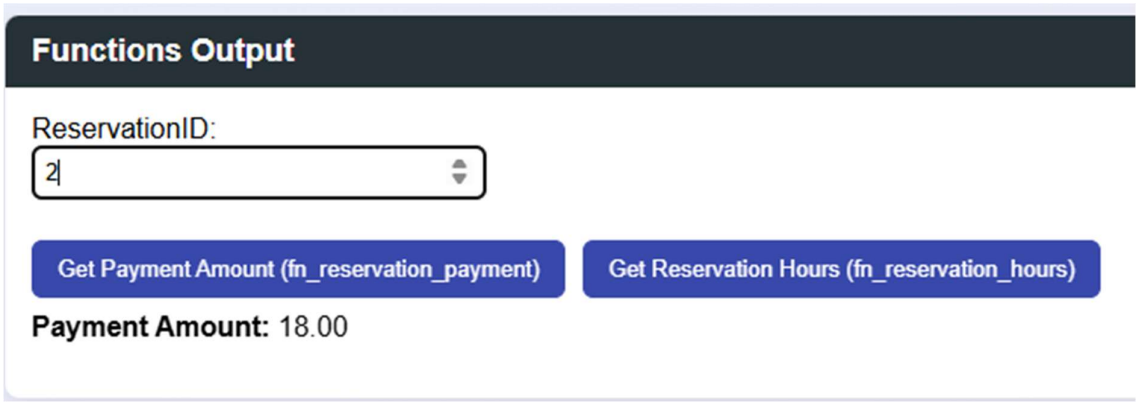


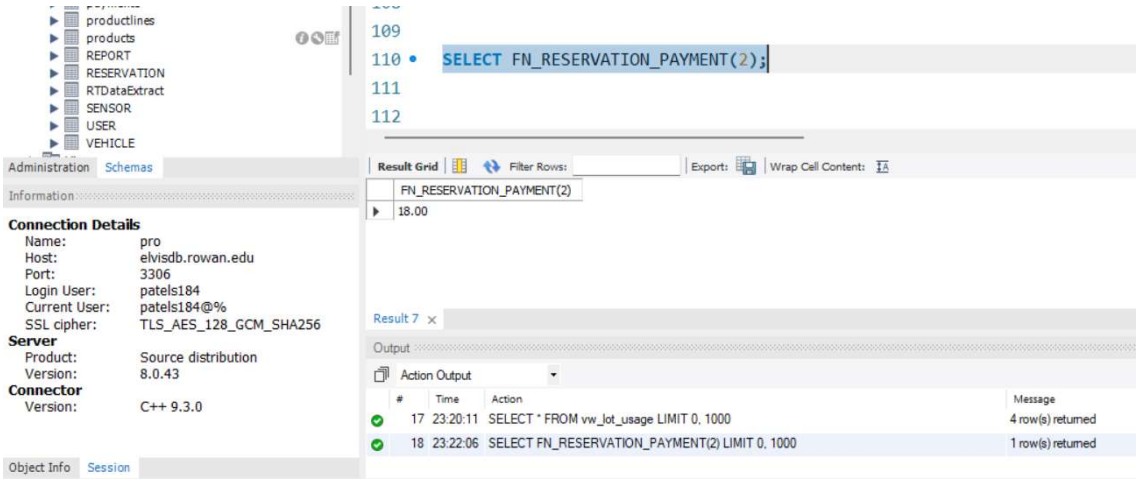
This screenshot shows the lot usage view, summarizing total, occupied, and available slots per parking lot.

Functions Output



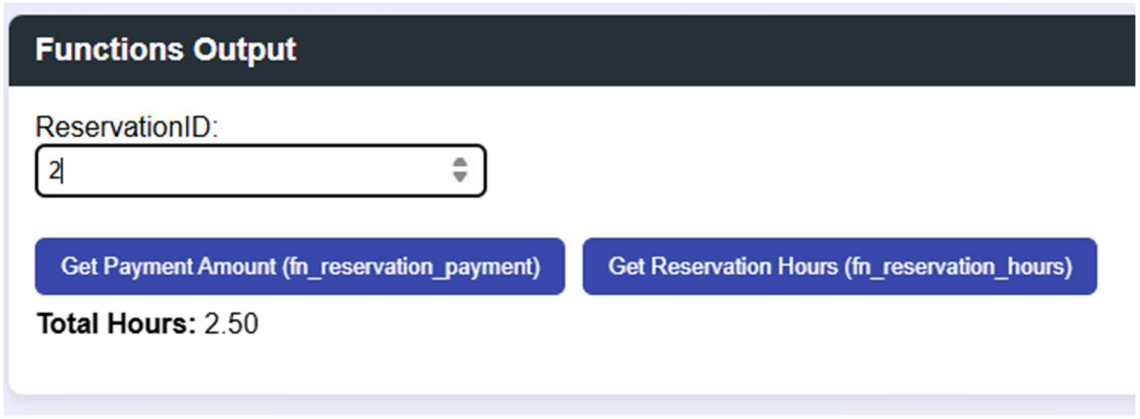
Function 1:

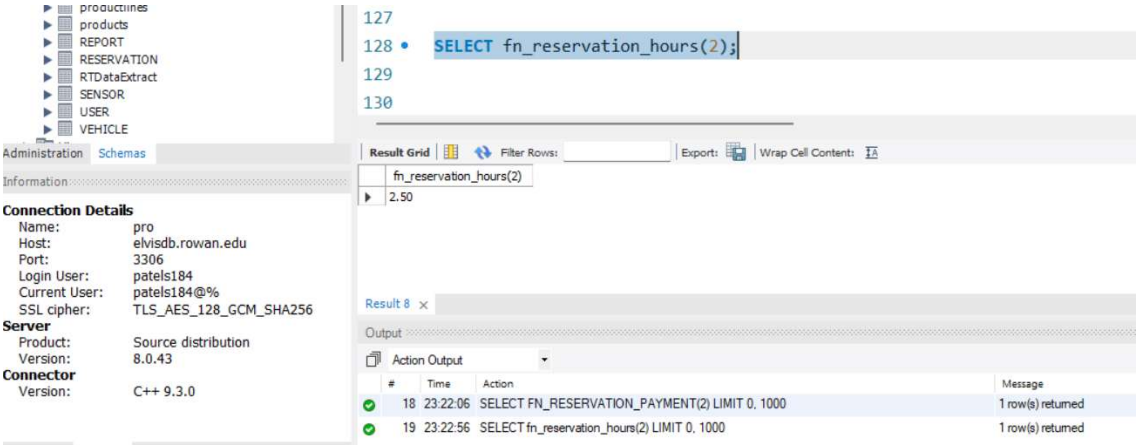




This screenshot demonstrates retrieving the payment amount for a reservation using the reservation ID.

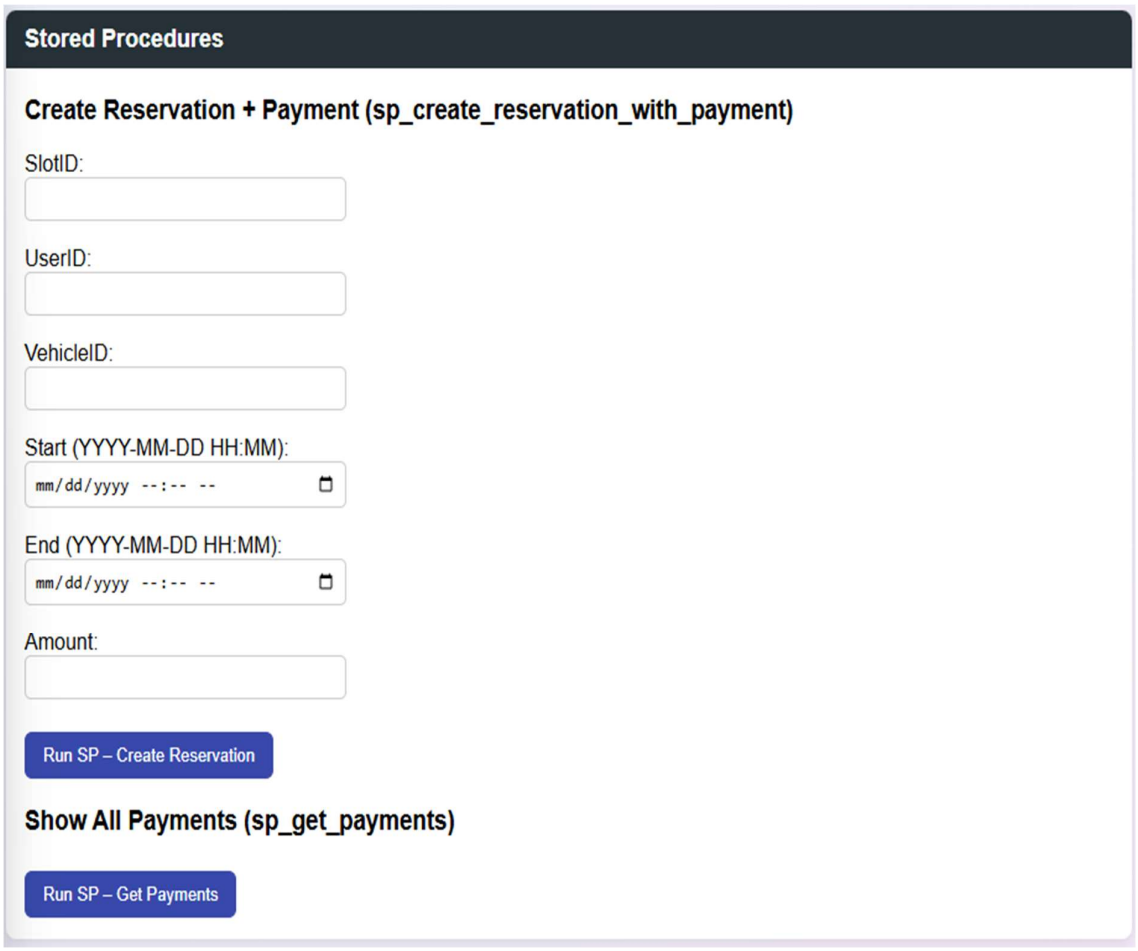
Function 2:



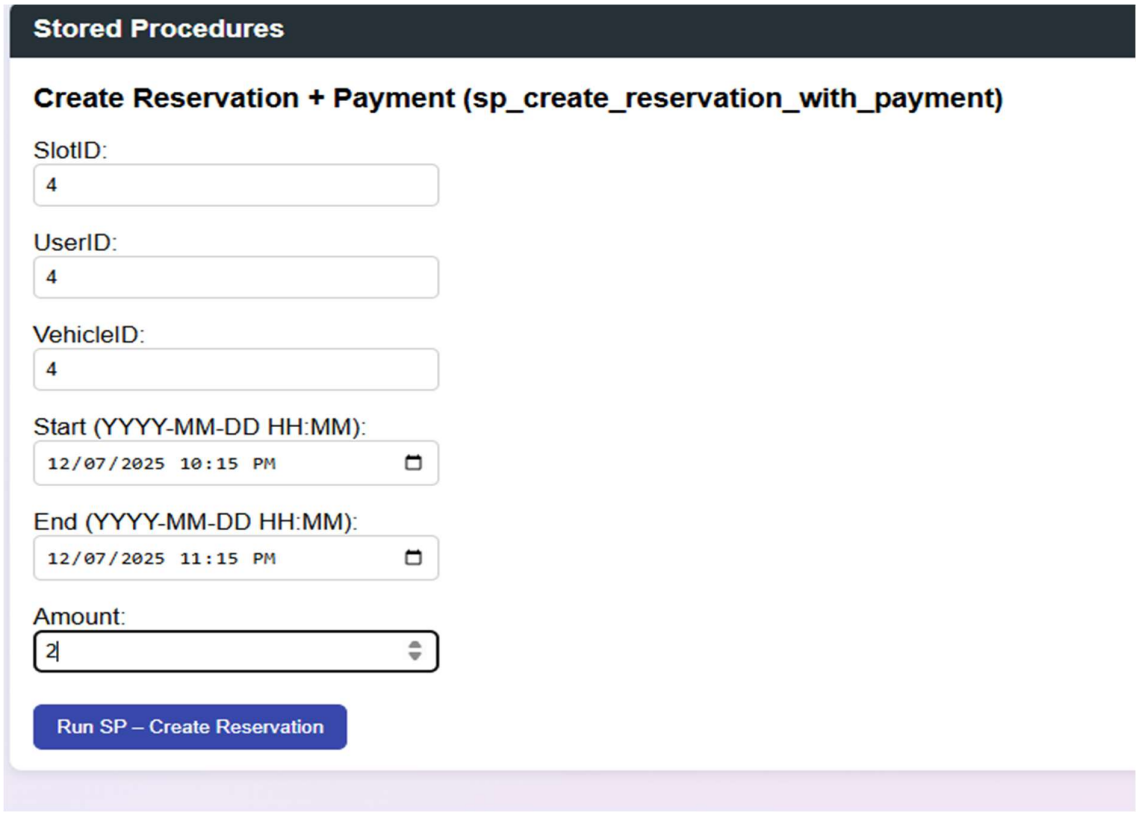


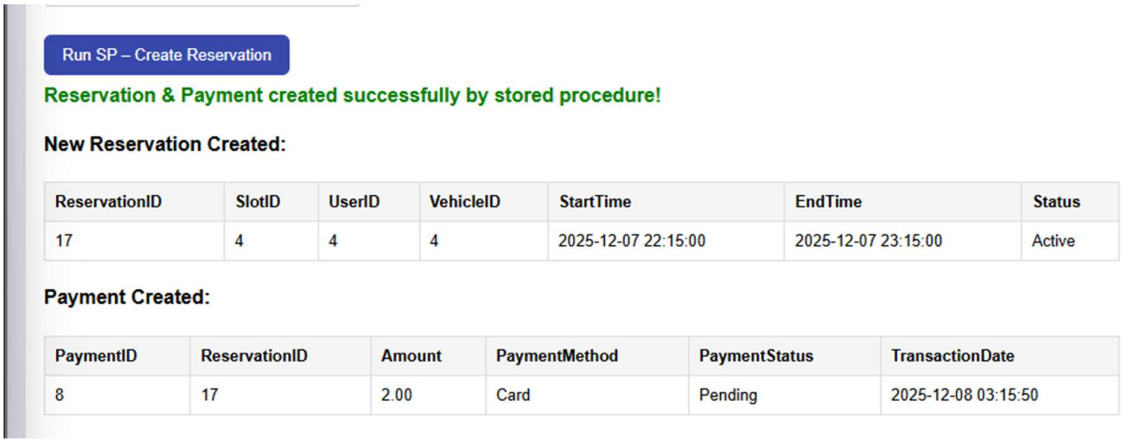
This screenshot shows the function calculating total reserved hours based on start and end time.

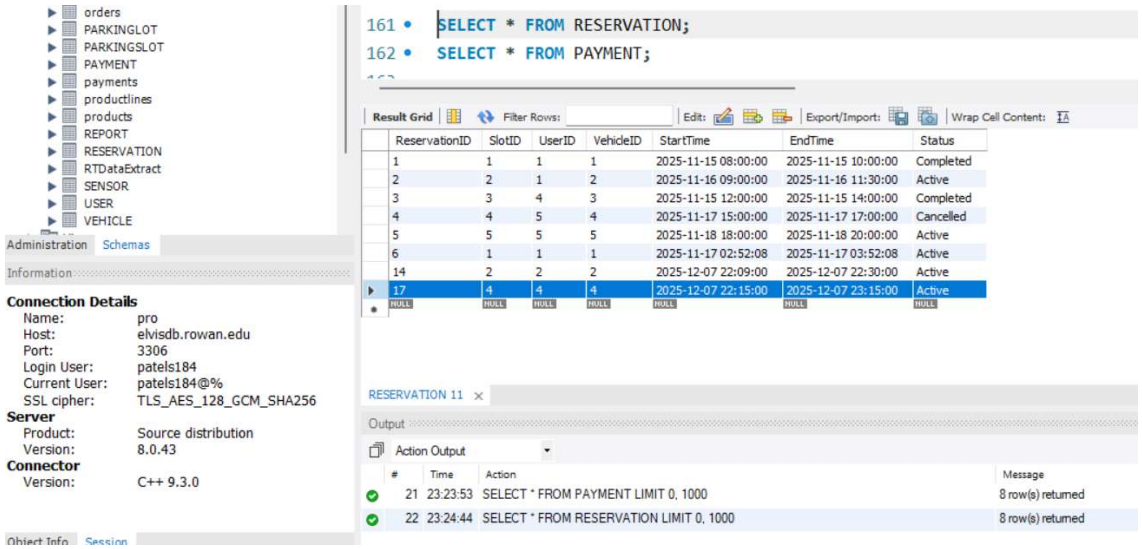
Stored Procedures

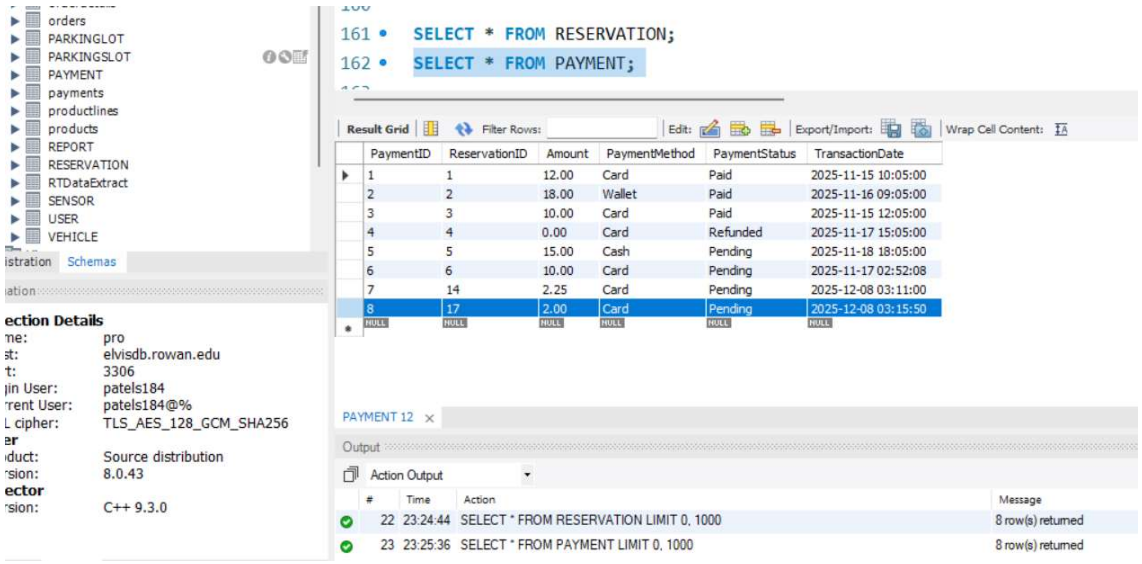


Stored Procedures 1:







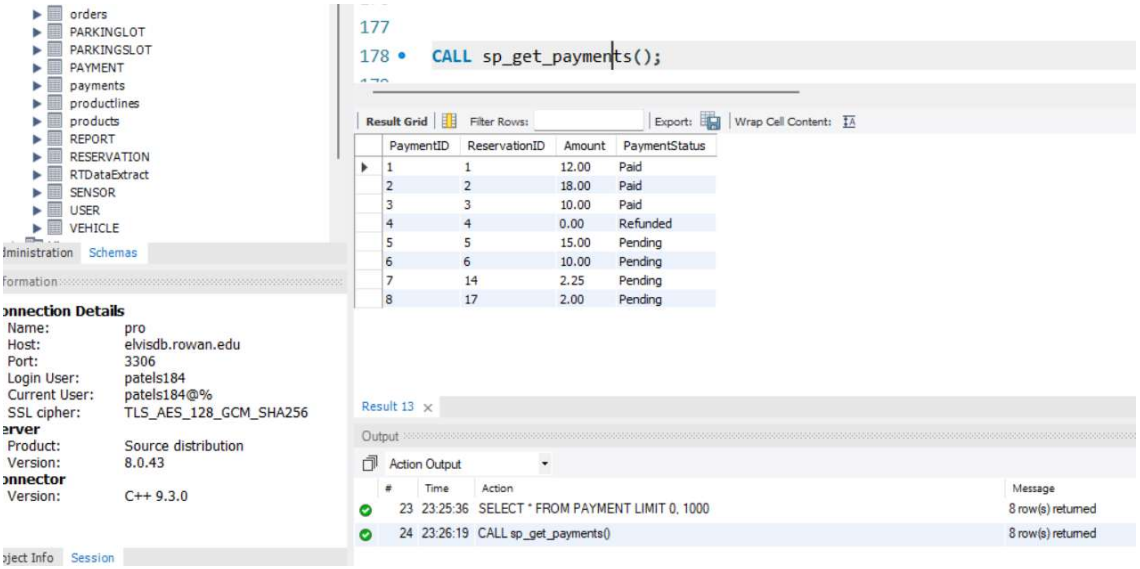


This screenshot shows successful execution of the stored procedure which creates a reservation and its corresponding payment.

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



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:

2

New Status:

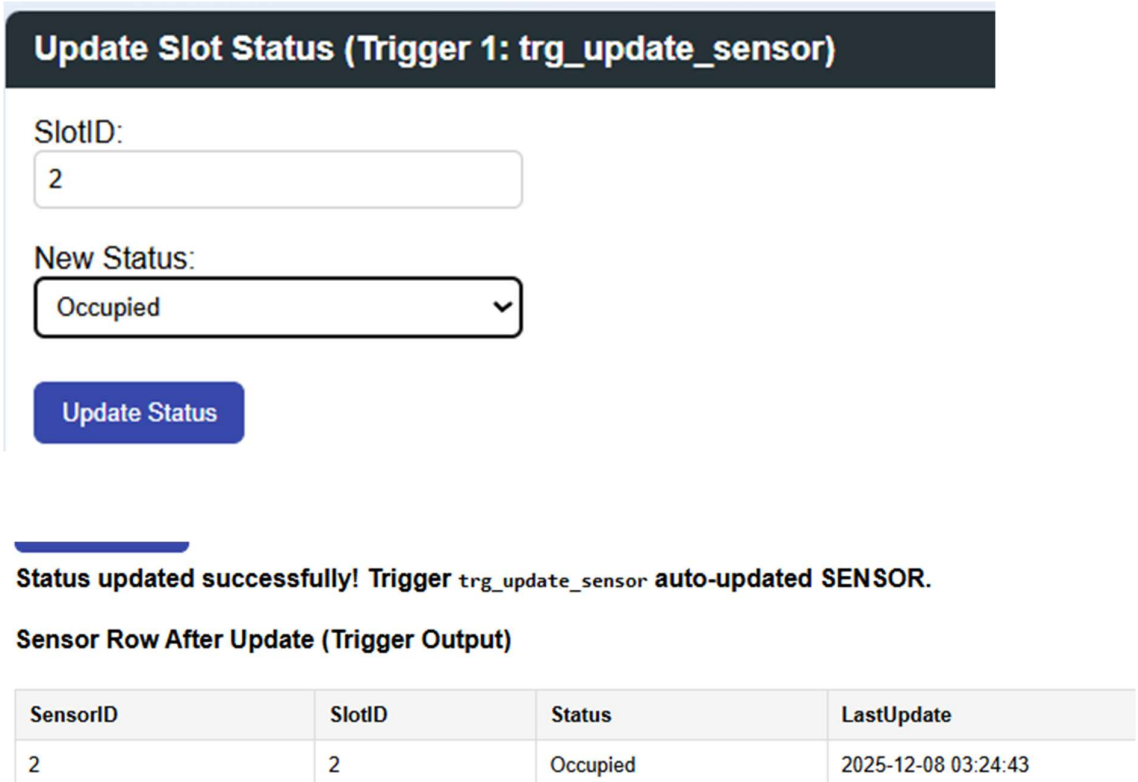
Available

Update Status

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

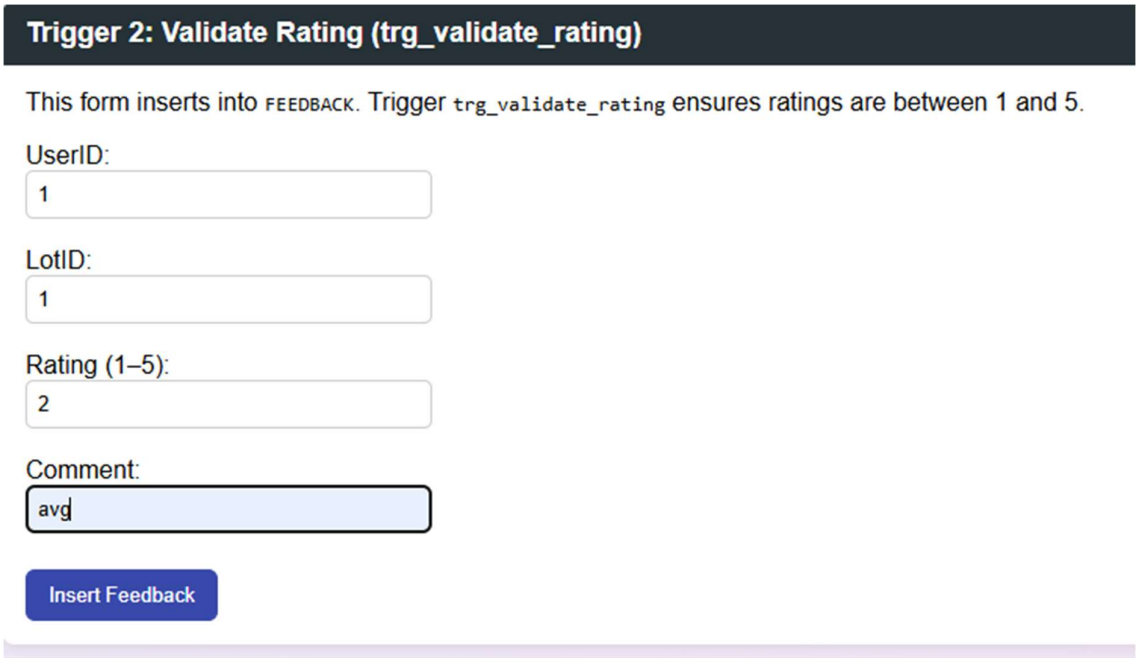
Sensor Row After Update (Trigger Output)

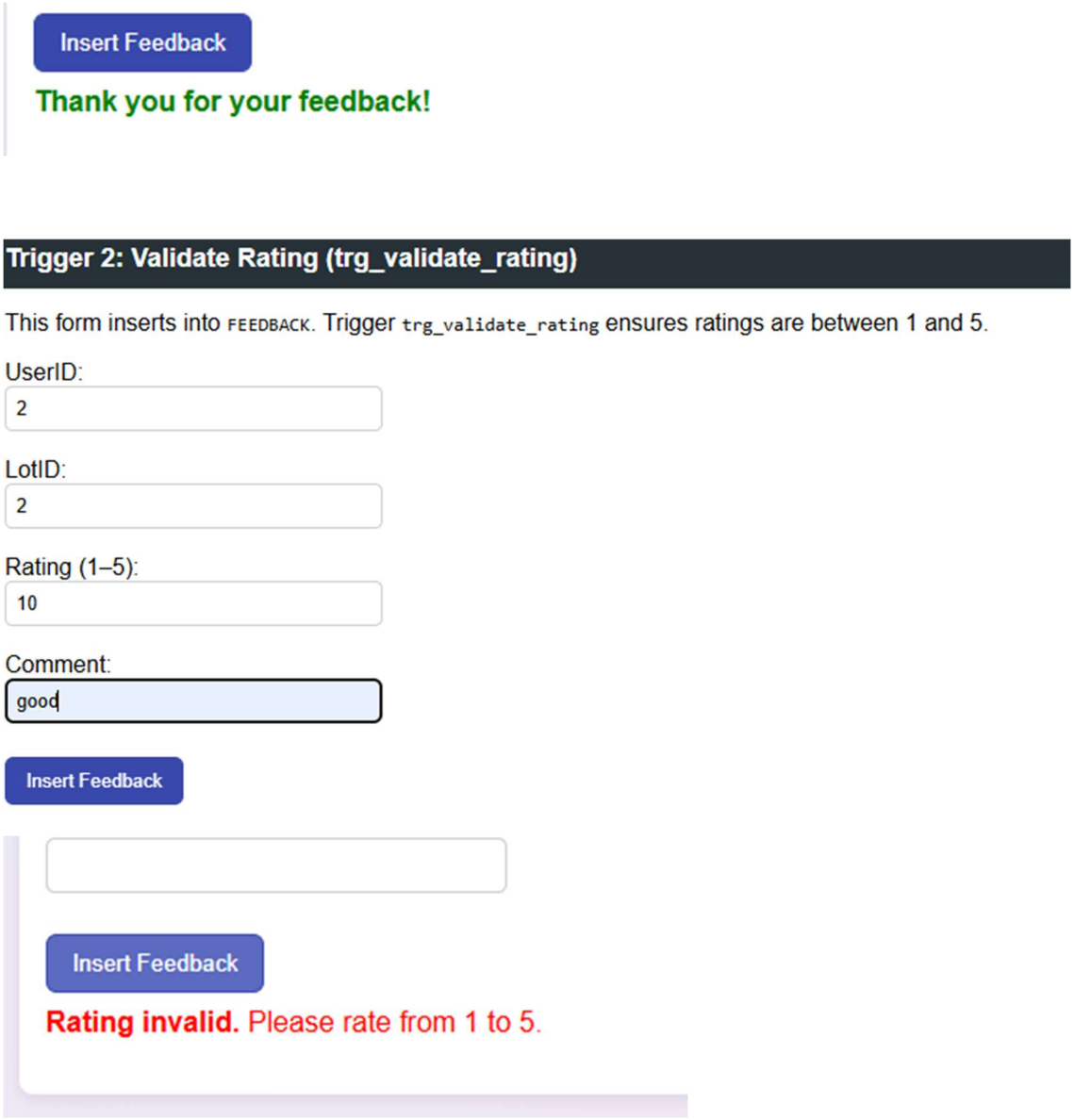
SensorID	SlotID	Status	LastUpdate
2	2	Free	2025-12-08 03:23:16



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

Trigger 2:





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