Parking Management System

Name	ID	Contribution Percentage	Topic Names
Md Rased Hasan Rokon	21-44574-1	35%	ER Diagram, User Interface Planning, Table Creation, Future Work.
Prachurjo Nath	21-44545-1	35%	ER Diagram, Table Creation, Data Insertion, Query Writing,
Parboti Das Puja	21-44648-1	30%	Introduction, Normalization, Query Writing, Future Work, Conclusion

Course Name: Advance Database Management System

Section: A

Contents

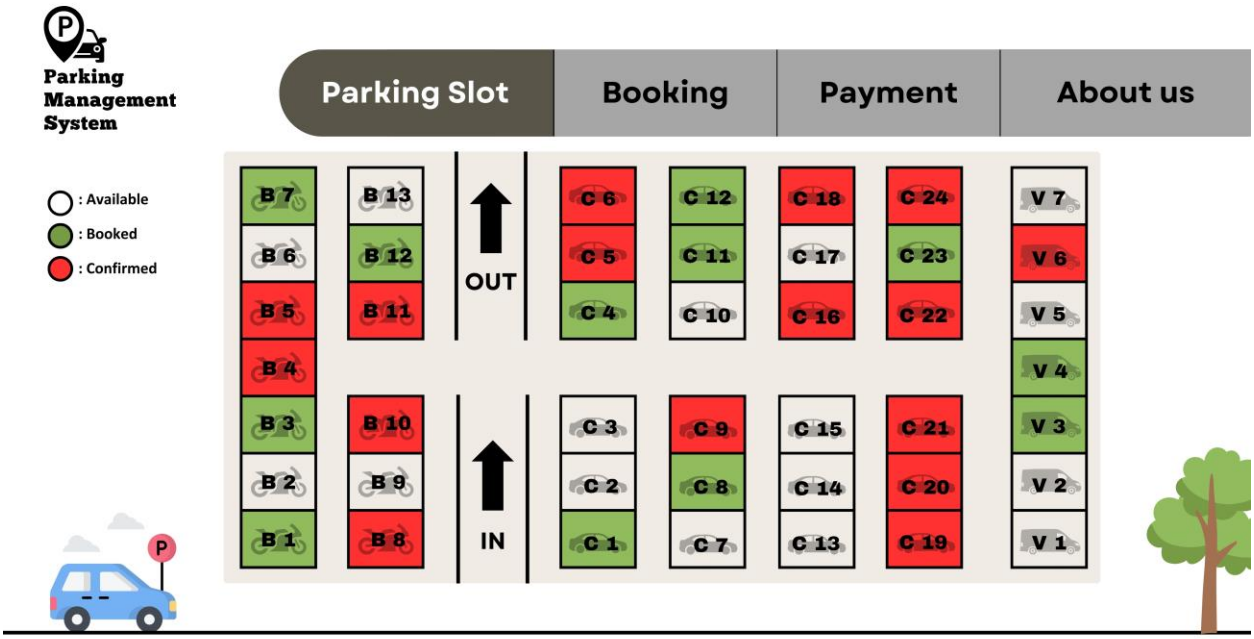
Introduction

Parking Management System is a user-friendly solution designed to make finding and booking parking spaces simple and convenient. It helps vehicle owners easily locate available parking slots, reserve them in advance, and complete payments securely.

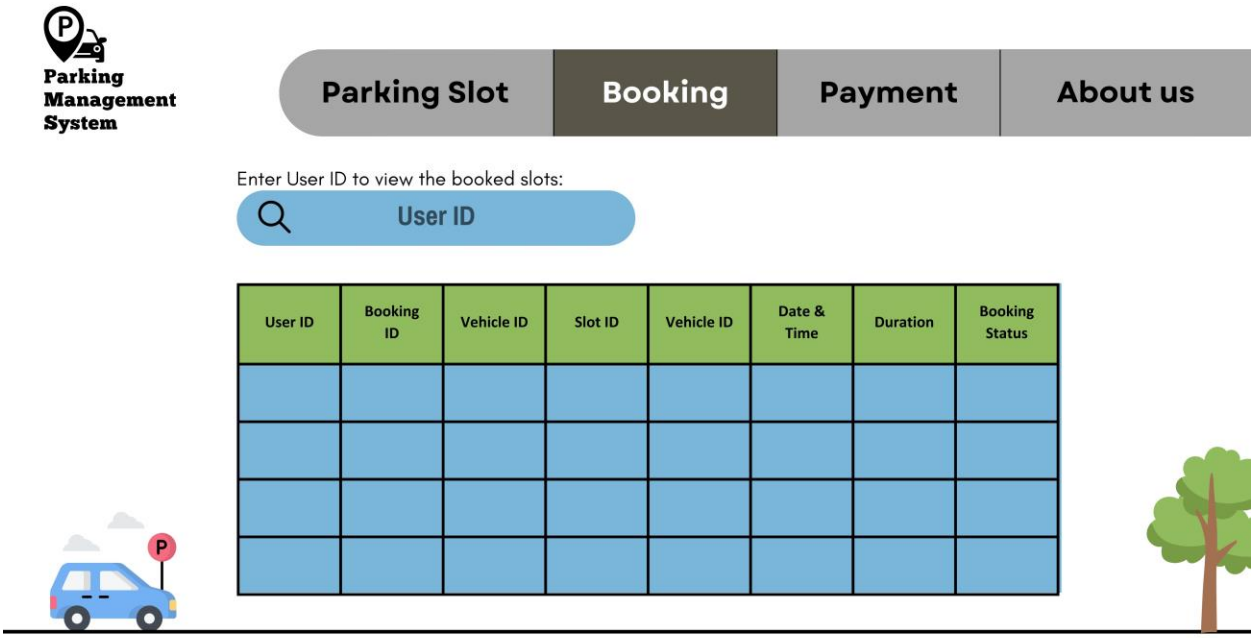
By organizing parking processes digitally, the system saves time, reduces hassle, and ensures that parking spaces are used efficiently. Whether you're parking a car, bike, or microbus, this system provides a seamless experience, allowing you to focus on your day without worrying about parking issues.

This project aims to simplify parking for everyone while ensuring fairness and ease of use for both users and parking operators.

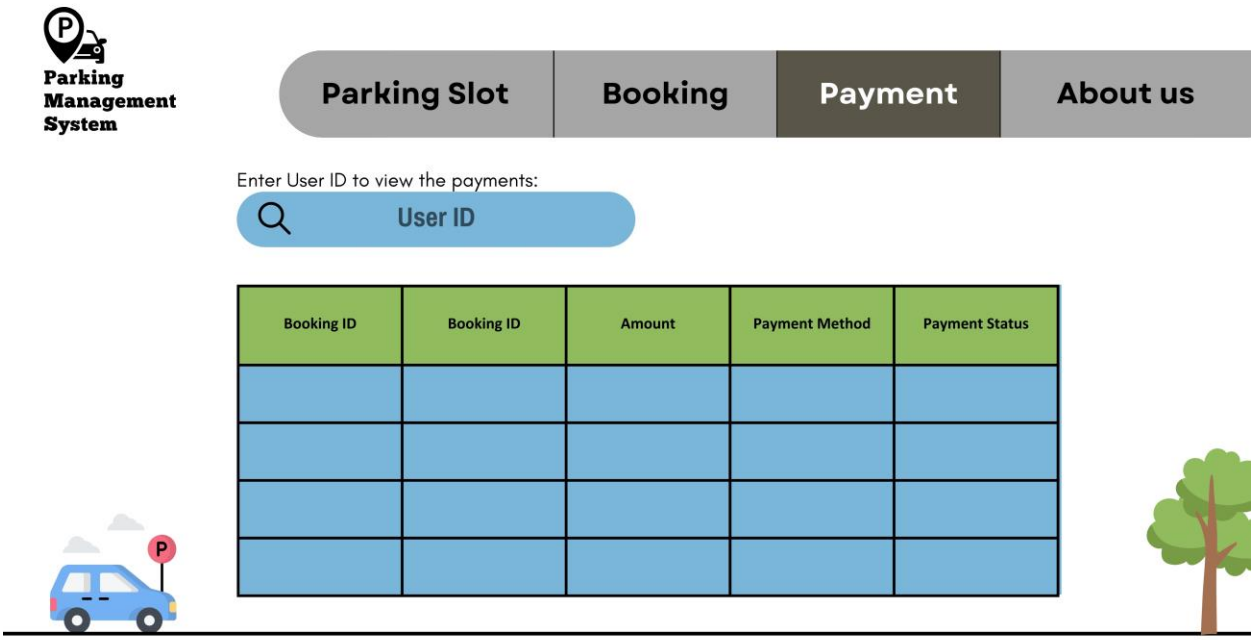
User Interface Planning



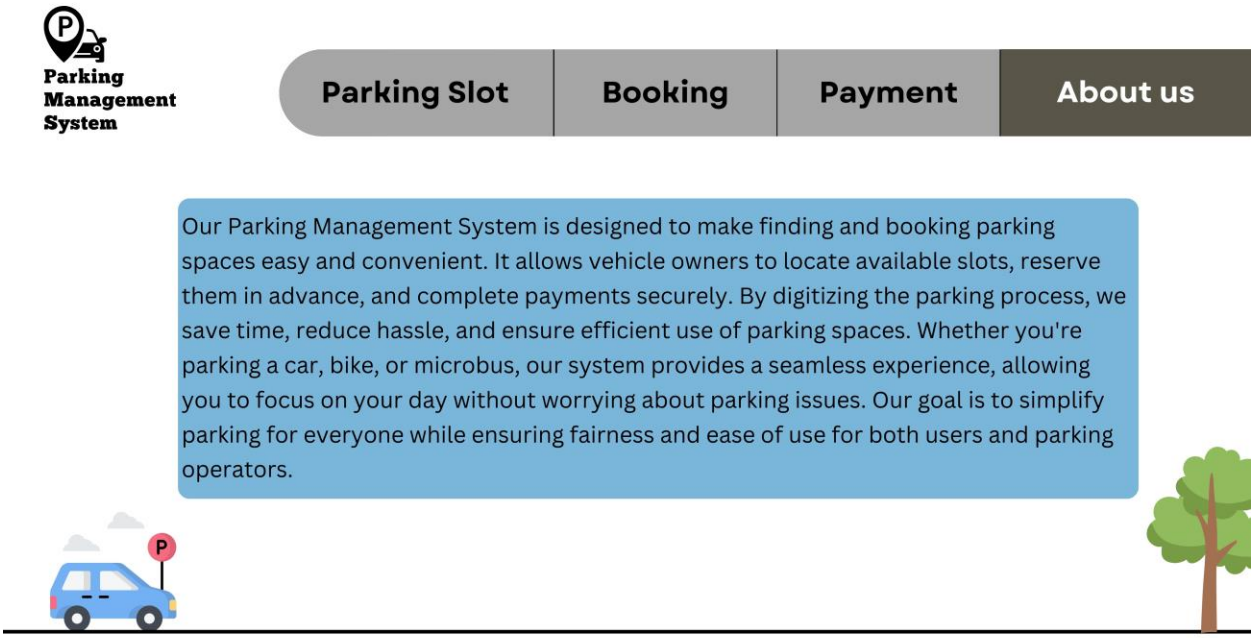
In this page, user can view available parking slots and select them for booking. Users have to submit their data and vehicle data for booking.



In this page, user can view their booked parking slots and payment status.



On this page, users can view their pending payments and can confirm the booked slots by completing payment.



This is about us page of the system

Scenario Description

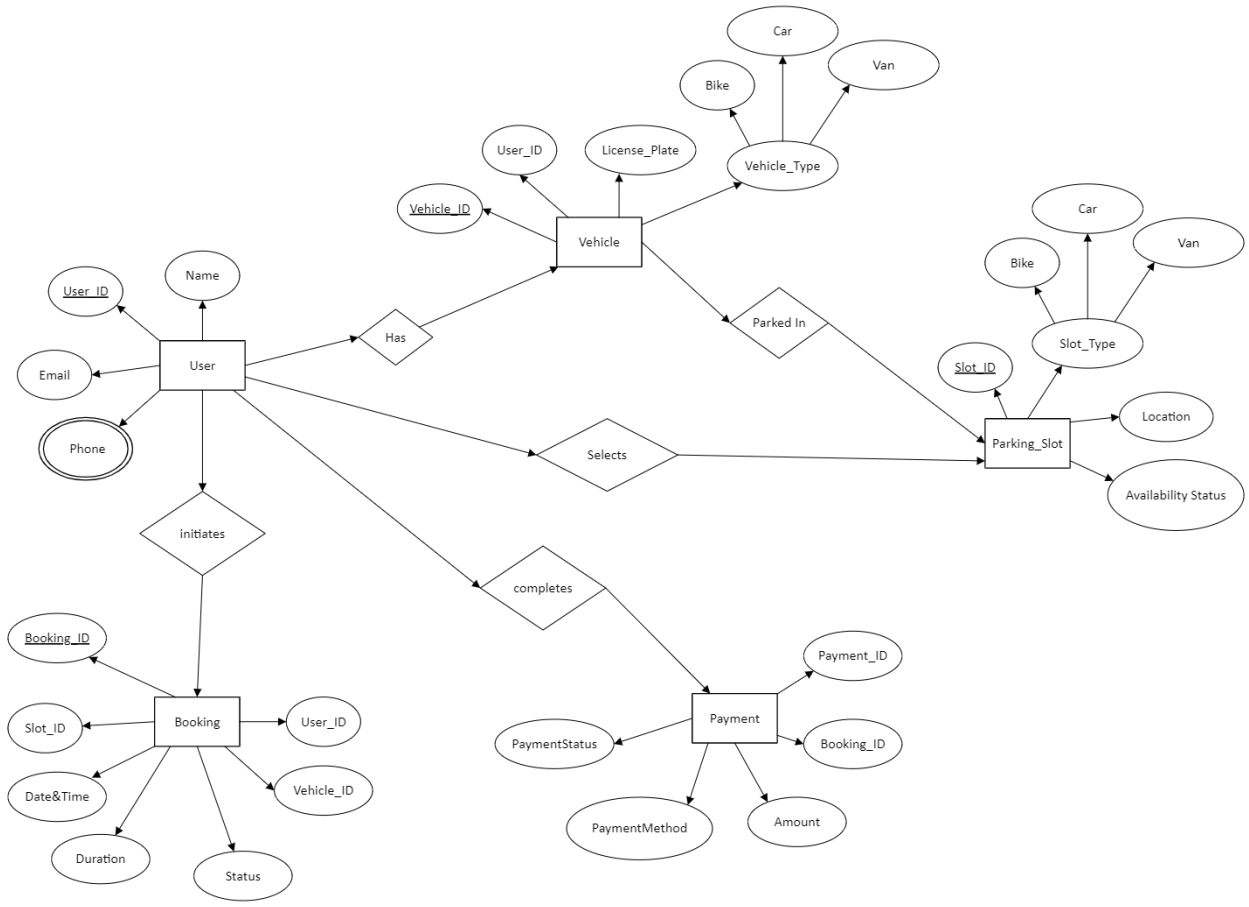
In a Parking Management System, a vehicle owner seeks to find available parking slots for their vehicle. Each owner is identified by a unique User ID along with their Name, Phone, and Email.

The user's vehicle is associated with a Vehicle ID and characterized by its License Plate and Vehicle Type (e.g., bike, car, or microbus). The system enables users to search for parking slots based on availability. Each parking slot is defined by a Slot ID, Slot Type, Location, and its Availability Status.

Once the user selects an available parking slot, they initiate a booking. The booking captures details such as Booking ID, Slot ID, User ID, Vehicle ID, Date and Time, Duration, and Booking Status (active or canceled).

To confirm the booking, the user must complete a payment. The payment record includes a Payment ID, Booking ID, Amount, Payment Method (e.g., cash, credit card), and Payment Status (completed or pending). Upon successful payment, the system secures the parking slot and notifies the user via email and text, including the confirmation of the booking details.

ER Diagram



Normalization

1st Normal Form:

UserID	VehicleID	VehicleType	SlotID	SlotType	BookingID	Date&Time	Duration	PaymentID	Amount	PaymentMeethod
1	1	Car	1	Car	1	10/12/2024	1hour	1	50	Cash
2	2	Bike	2	Bike	2	10/12/2024	2hour	2	100	Bkash

2nd Normal Form:

User-Parking Slot table

UserID	VehicleID	VehicleType	SlotID	SlotType

1	1	Car	1	Car
2	2	Bike	2	Bike

User-Booking table

UserID	SlotID	BookingID	Date&Time	Duration
1	1	1	10/12/2024	1hour
2	2	2	10/12/2024	2hour

Booking-Payment table

BookingID	PaymentID	Amount	PaymentMeethod
1	1	50	Cash
2	2	100	Bkash

3rd Normal Form:

User-Vehicle table

UserID	VehicleID	VehicleType
1	1	Car
2	2	Bike

Parking Slot table

SlotID	SlotType
1	Car
2	Bike

Booking table

BookingID	Date&Time	Duration
1	10/12/2024	1hour
2	10/12/2024	2hour

Payment table

PaymentID	Amount	PaymentMeethod
1	50	Cash
2	100	Bkash

Table Creation

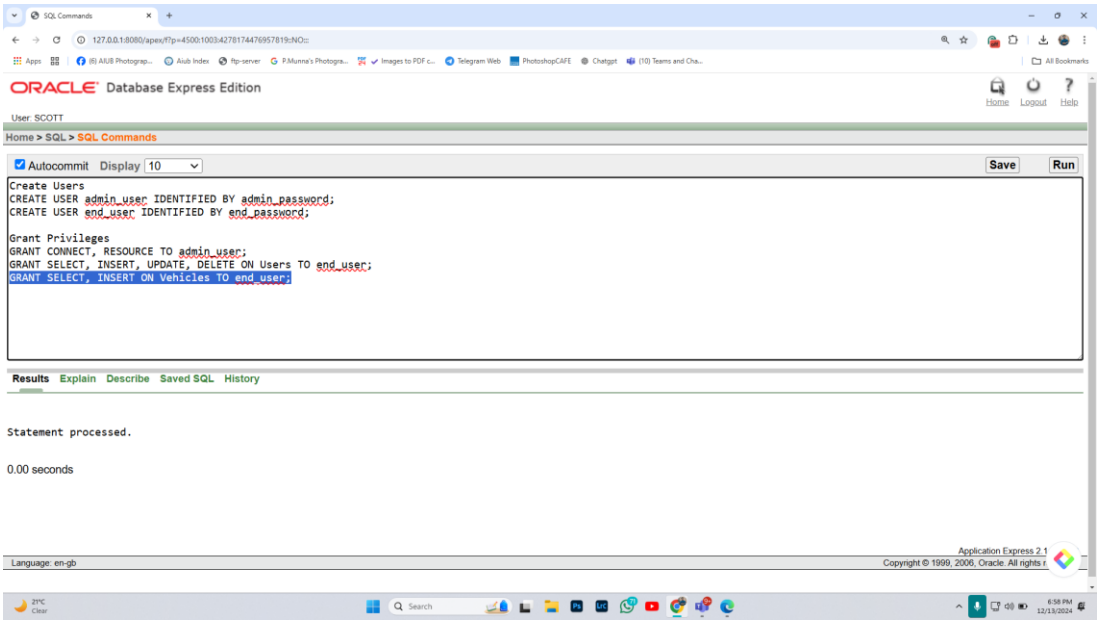
User creation:

CREATE USER admin_user IDENTIFIED BY admin_password;

CREATE USER end_user IDENTIFIED BY end_password;

GRANT CONNECT, RESOURCE TO admin_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON Users TO end_user; GRANT SELECT, INSERT ON Vehicles TO end_user;



Sequence creation:

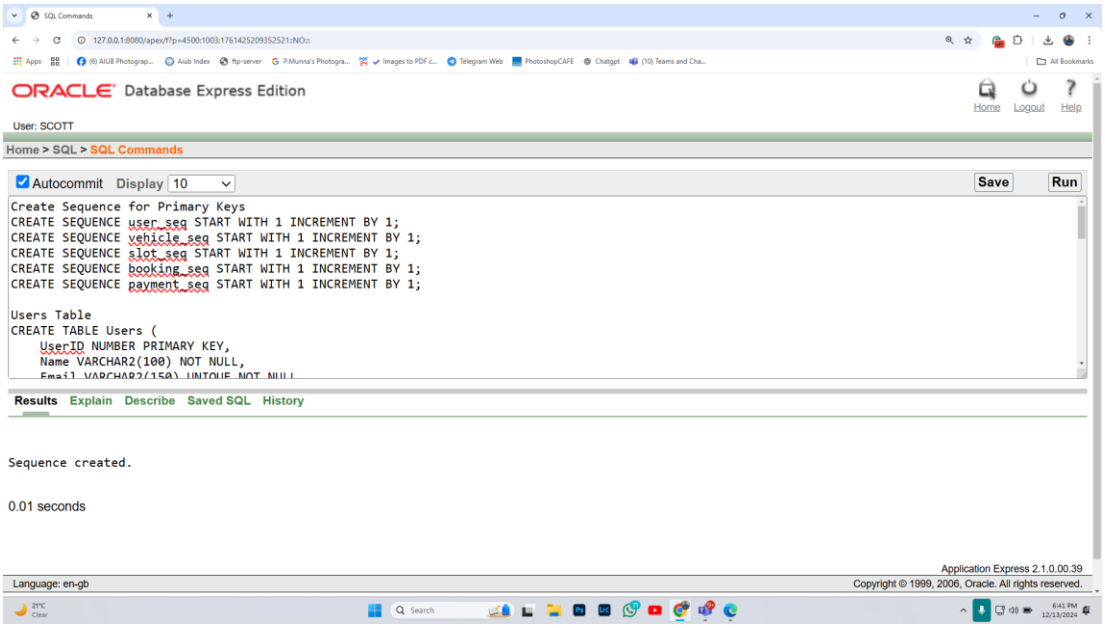
CREATE SEQUENCE user_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE vehicle_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE slot_seq START WITH 1 INCREMENT BY 1;

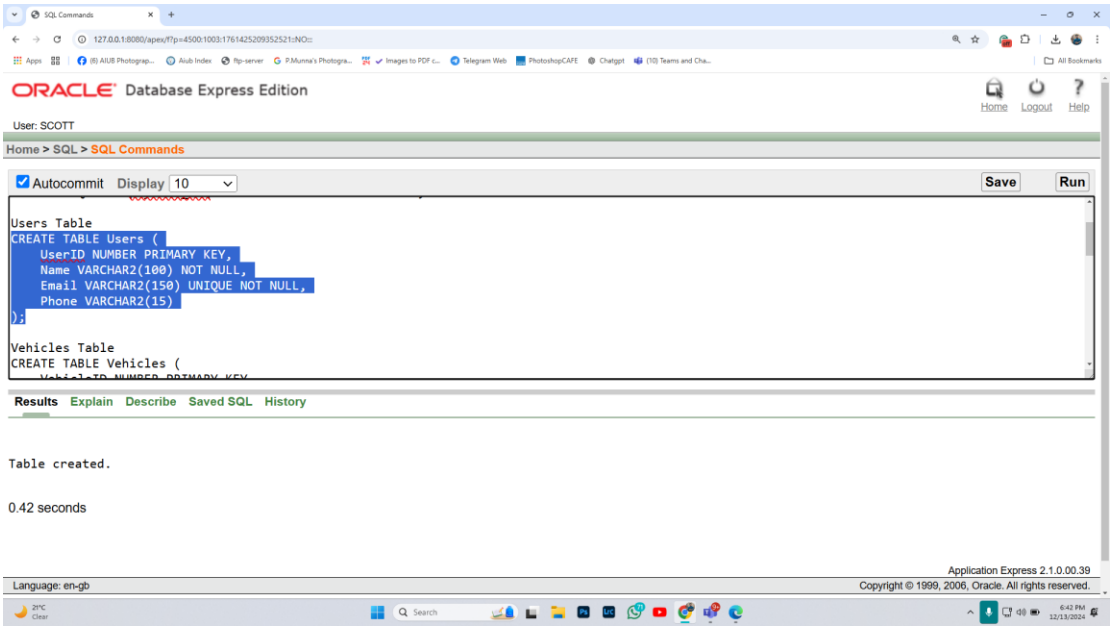
CREATE SEQUENCE booking_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE payment_seq START WITH 1 INCREMENT BY 1;



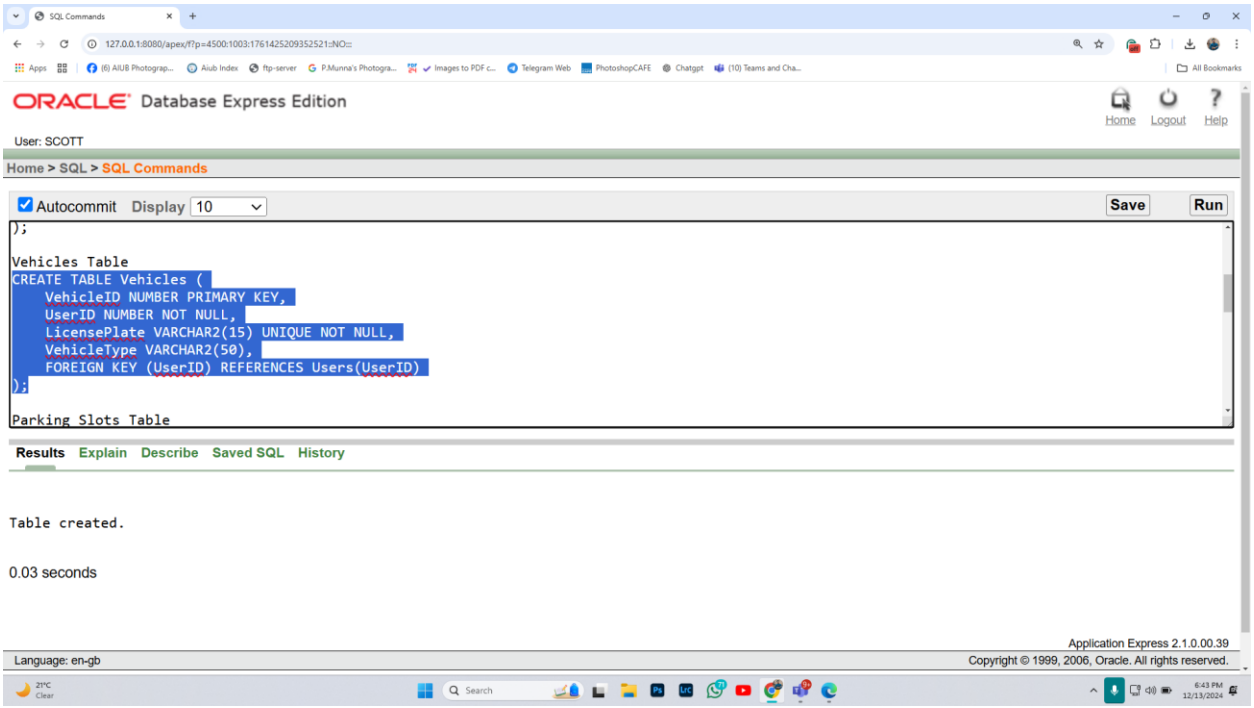
User Table:

```
CREATE TABLE Users (
        UserID NUMBER PRIMARY KEY,
        Name VARCHAR2(100) NOT NULL,
        Email VARCHAR2(150) UNIQUE NOT NULL,
        Phone VARCHAR2(15)
);
```



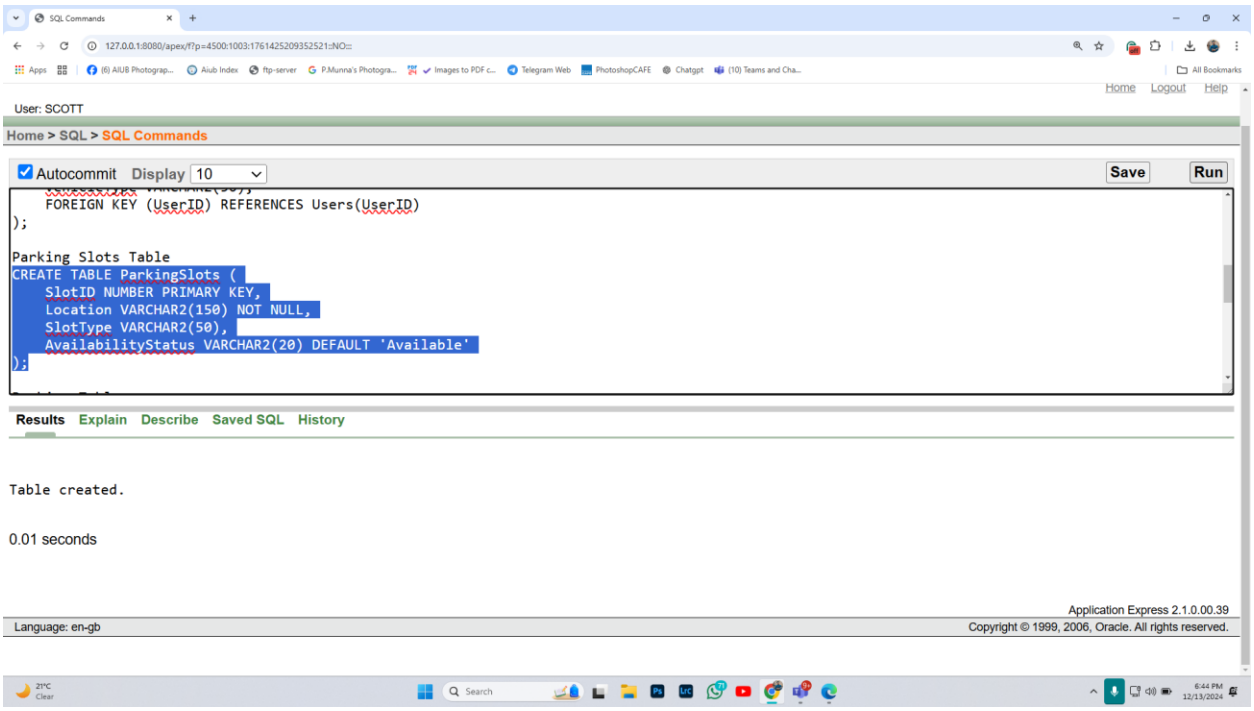
Vehicles Table:

```
CREATE TABLE Vehicles (
        VehicleID NUMBER PRIMARY KEY,
        UserID NUMBER NOT NULL,
        LicensePlate VARCHAR2(15) UNIQUE NOT NULL,
        VehicleType VARCHAR2(50),
        FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```



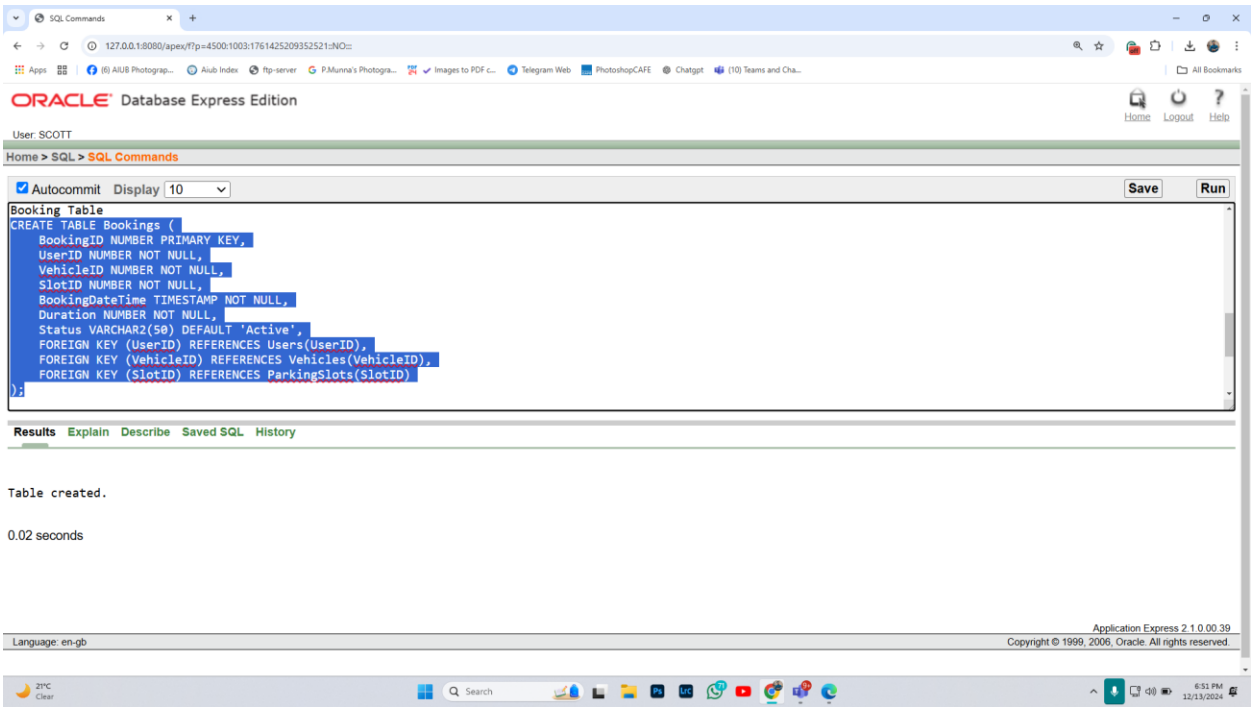
Parking Slots Table:

```
CREATE TABLE ParkingSlots (
        SlotID NUMBER PRIMARY KEY,
        Location VARCHAR2(150) NOT NULL,
        SlotType VARCHAR2(50),
        AvailabilityStatus VARCHAR2(20) DEFAULT 'Available'
 );
```



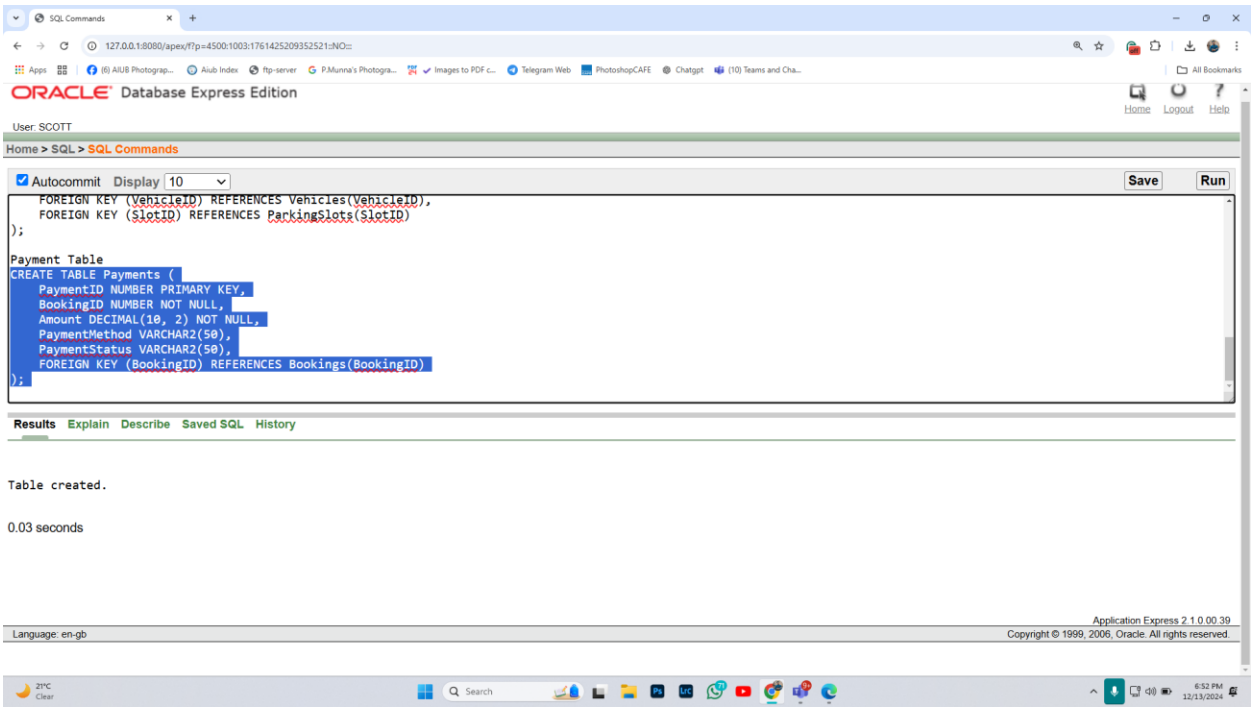
Booking Table:

```
CREATE TABLE Bookings (
        BookingID NUMBER PRIMARY KEY,
        UserID NUMBER NOT NULL,
        SlotID NUMBER NOT NULL,
        BookingDateTime TIMESTAMP NOT NULL,
        Duration NUMBER NOT NULL,
        Status VARCHAR2(50) DEFAULT 'Active',
        FOREIGN KEY (UserID) REFERENCES Users(UserID),
        FOREIGN KEY (SlotID) REFERENCES ParkingSlots(SlotID)
);
```



Payment Table:

```
CREATE TABLE Payments (
        PaymentID NUMBER PRIMARY KEY,
        BookingID NUMBER NOT NULL,
        Amount DECIMAL(10, 2) NOT NULL,
        PaymentMethod VARCHAR2(50),
        PaymentStatus VARCHAR2(50),
        FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)
);
```



Data Insertion

Insert Data into Users:

INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Abdul Karim', 'abdul.karim@example.com', '01712345678');

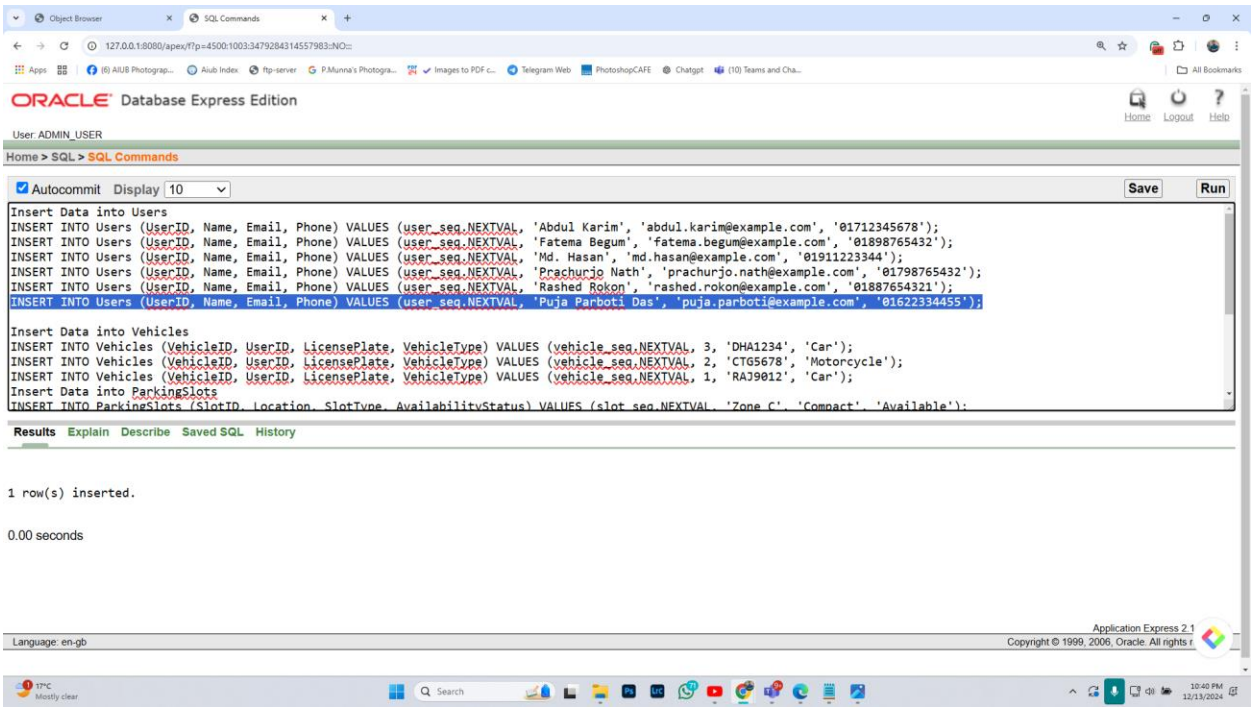
INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Fatema Begum', 'fatema.begum@example.com', '01898765432');

INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Md. Hasan', 'md.hasan@example.com', '01911223344');

INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Prachurjo Nath', 'prachurjo.nath@example.com', '01798765432');

INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Rashed Rokon', 'rashed.rokon@example.com', '01887654321');

INSERT INTO Users (UserID, Name, Email, Phone, Password) VALUES (user_seq.NEXTVAL, 'Puja Parboti Das', 'puja.parboti@example.com', '01622334455');



Insert Data into Vehicles:

INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 3, 'DHA1234', 'Car');

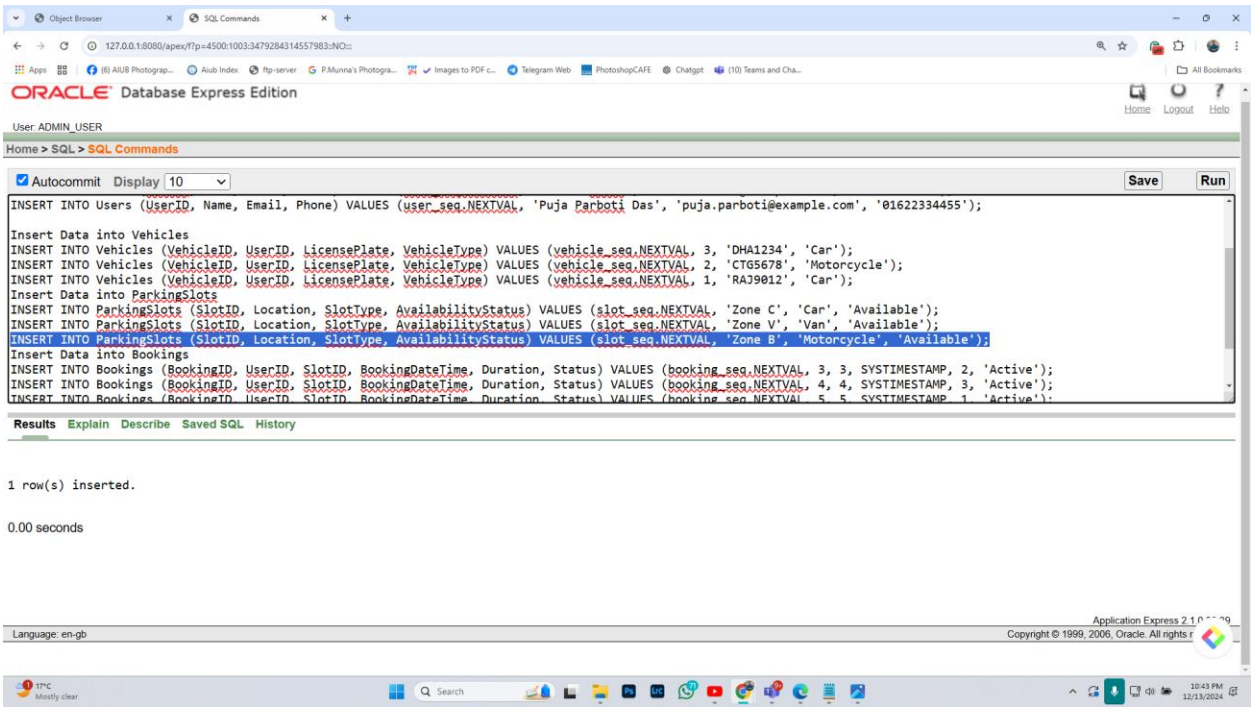
INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 4, 'CTG5678', 'Motorcycle');

INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 5, 'RAJ9012', 'Car');

INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 4, 'SYL3456', 'Car');

INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 5, 'KHU7890', 'Truck');

INSERT INTO Vehicles (VehicleID, UserID, LicensePlate, VehicleType) VALUES (vehicle_seq.NEXTVAL, 6, 'BAR1122', 'Motorcycle');



Insert Data into ParkingSlots:

INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone C', 'Compact', 'Available');

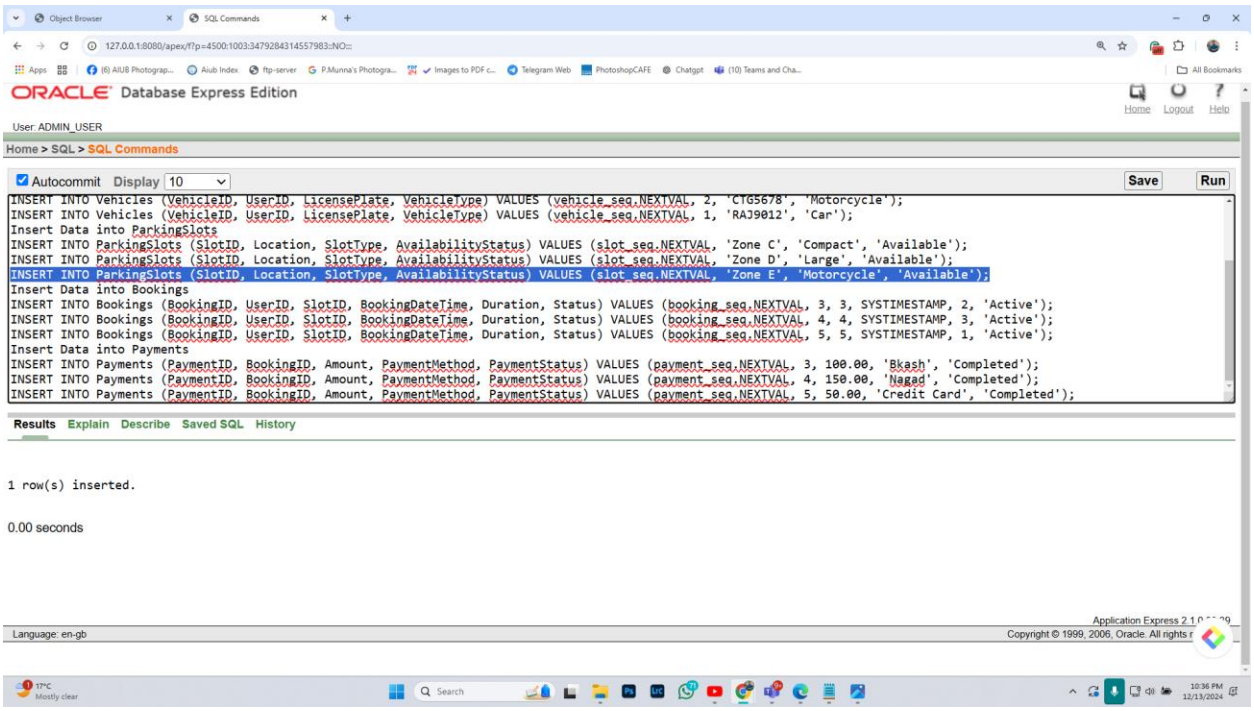
INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone D', 'Large', 'Available');

INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone E', 'Motorcycle', 'Available');

INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone F', 'Compact', 'Completed');

INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone G', 'Truck', 'Available');

INSERT INTO ParkingSlots (SlotID, Location, SlotType, AvailabilityStatus) VALUES (slot_seq.NEXTVAL, 'Zone H', 'Car', 'Completed');



Insert Data into Bookings:

NSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 3, 3, SYSTIMESTAMP, 2, 'Active');

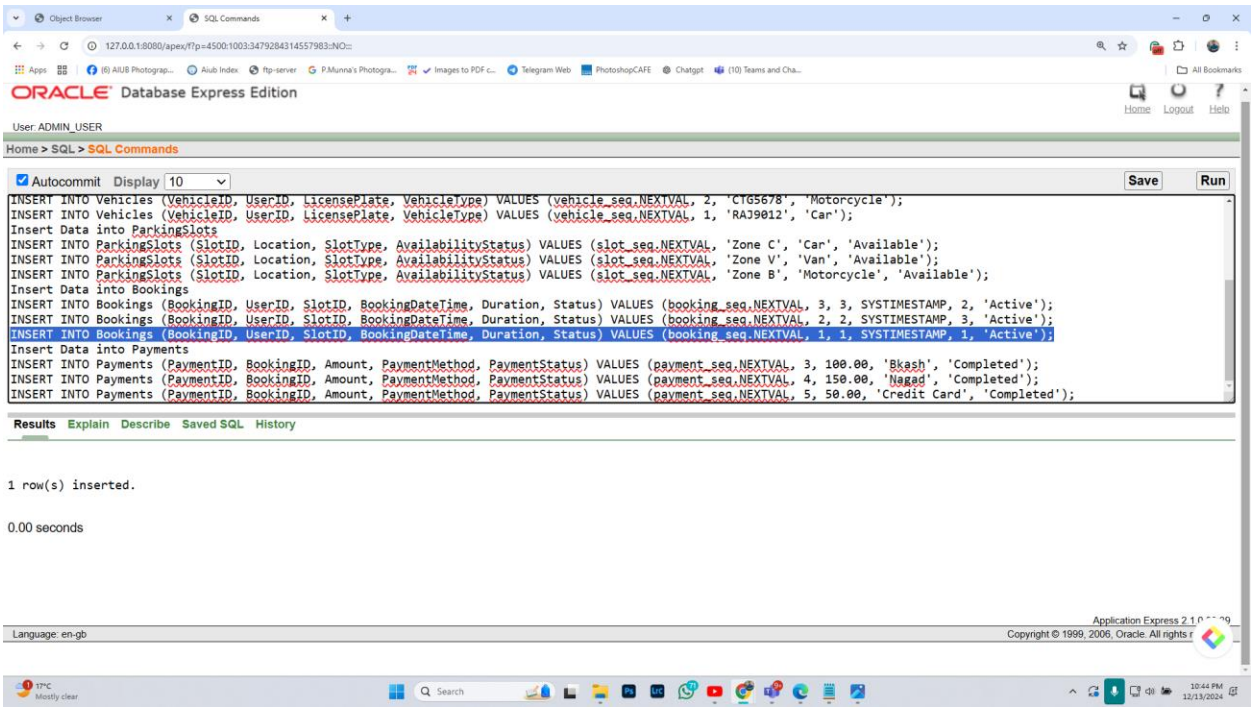
INSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 4, 4, SYSTIMESTAMP, 3, 'Active');

INSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 5, 5, SYSTIMESTAMP, 1, 'Active');

INSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 6, 6, SYSTIMESTAMP, 4, 'Completed');

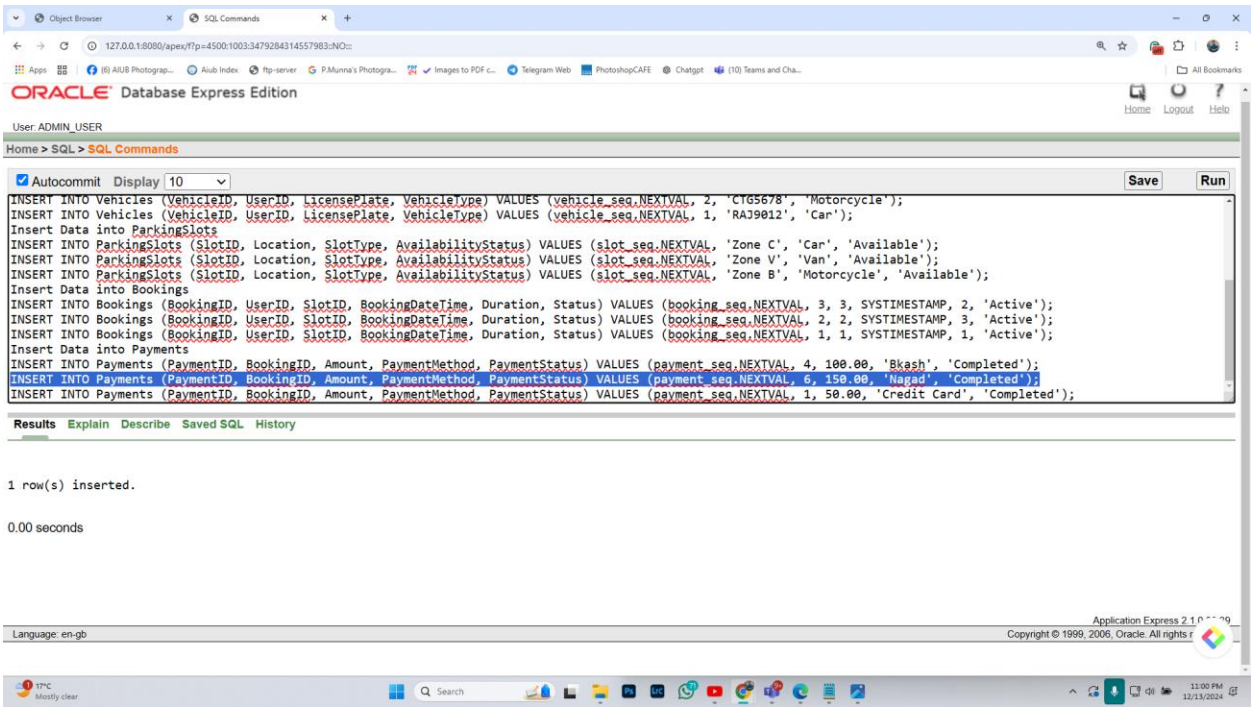
INSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 2, 2, SYSTIMESTAMP, 3, 'Cancelled');

INSERT INTO Bookings (BookingID, UserID, SlotID, BookingDateTime, Duration, Status) VALUES (booking_seq.NEXTVAL, 1, 1, SYSTIMESTAMP, 5, 'Active');



Insert Data into Payments:

INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 3, 100.00, 'Bkash', 'Completed');
INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 4, 150.00, 'Nagad', 'Completed');
INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 5, 50.00, 'Credit Card', 'Completed');
INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 6, 200.00, 'Bkash', 'Completed');
INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 2, 75.00, 'Nagad', 'Pending');
INSERT INTO Payments (PaymentID, BookingID, Amount, PaymentMethod, PaymentStatus) VALUES (payment_seq.NEXTVAL, 1, 120.00, 'Cash', 'Refunded');



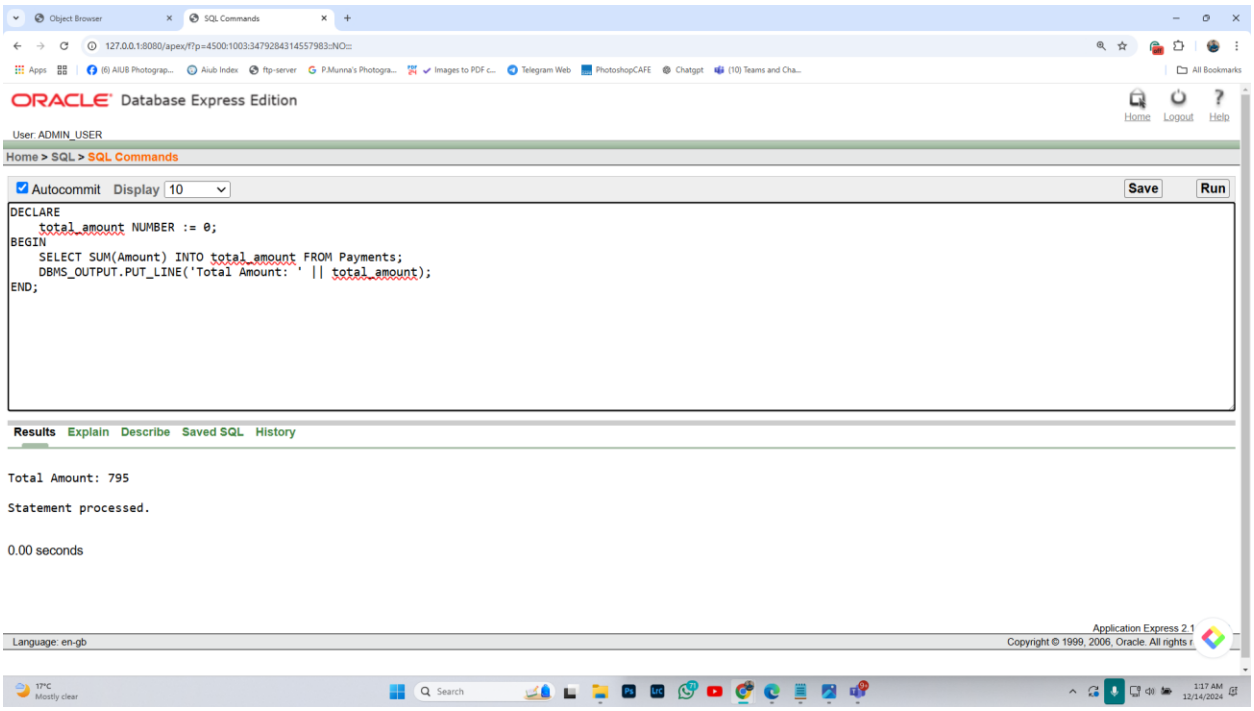
Query Writing

Basic PL/SQL Questions

Variables

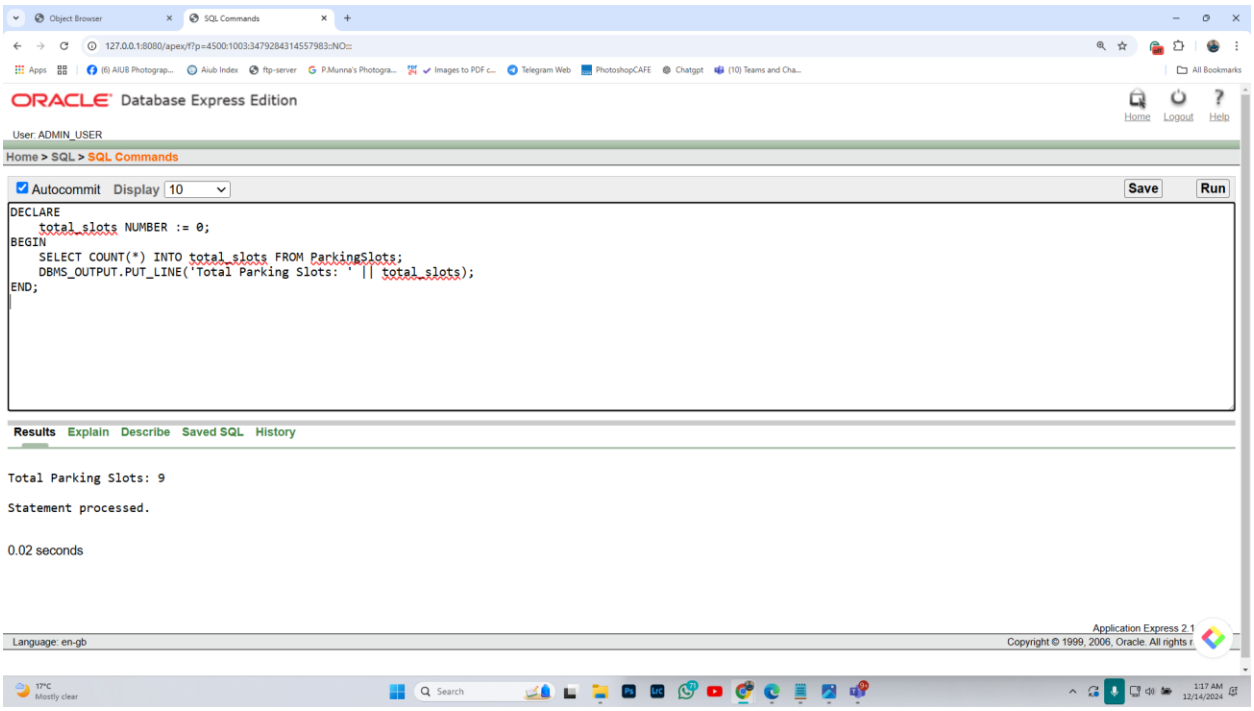
1. Calculate the total amount from the `Payments` table using a variable.

```
DECLARE
total_amount NUMBER := 0;
BEGIN
SELECT SUM(Amount) INTO total_amount FROM Payments;
DBMS_OUTPUT.PUT_LINE('Total Amount: ' || total_amount);
END;
```



2. Count the total number of parking slots and store the result in a variable.

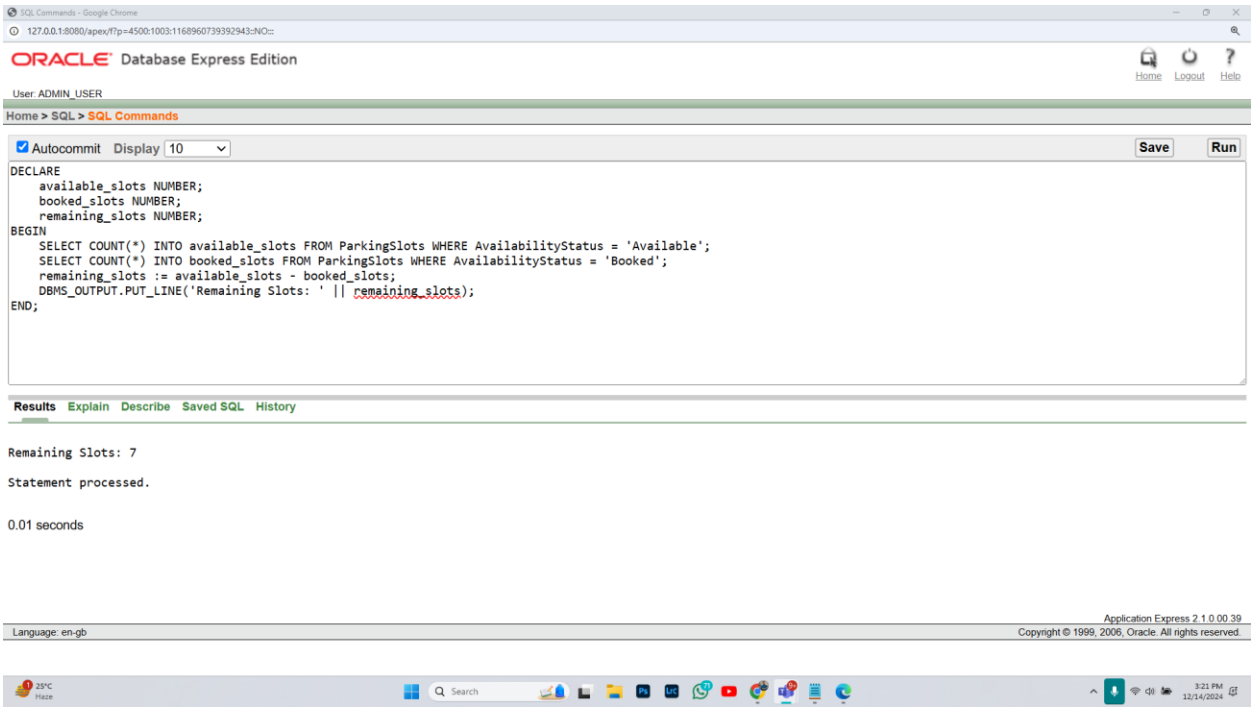
```
DECLARE
total_slots NUMBER := 0;
BEGIN
SELECT COUNT(*) INTO total_slots FROM ParkingSlots;
 DBMS_OUTPUT.PUT_LINE('Total Parking Slots: ' || total_slots);
END;
```



Operators

1. Compute the difference between available and booked slots using arithmetic operators.
2. Display the result of an arithmetic operation between two counts from different tables.

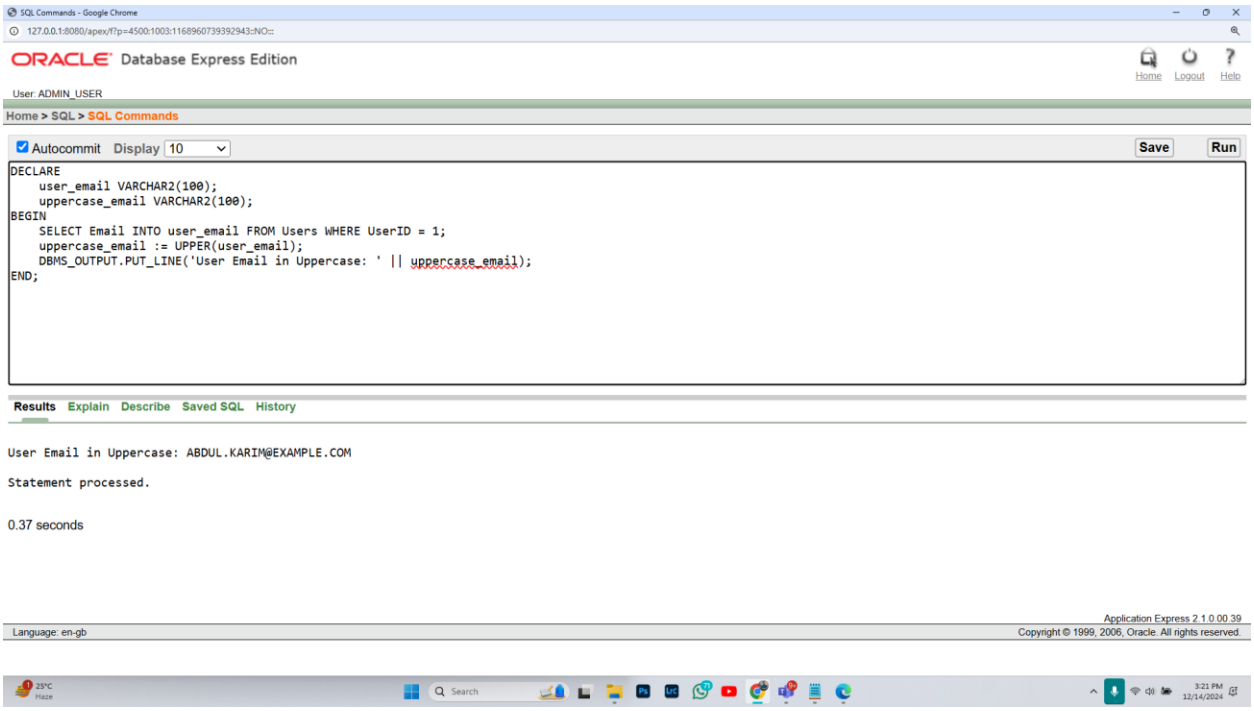
```
DECLARE
available_slots NUMBER;
booked_slots NUMBER;
remaining_slots NUMBER;
BEGIN
SELECT COUNT() INTO available_slots FROM ParkingSlots WHERE AvailabilityStatus =
'Available';
SELECT COUNT() INTO booked_slots FROM ParkingSlots WHERE AvailabilityStatus =
'Booked';
remaining_slots := available_slots - booked_slots;
DBMS_OUTPUT.PUT_LINE('Remaining Slots: ' || remaining_slots);
END;
```



Single-Row Functions

1. Convert a user's email into uppercase using PL/SQL.
2. How would you fetch the email of a specific user and perform string manipulation on it.

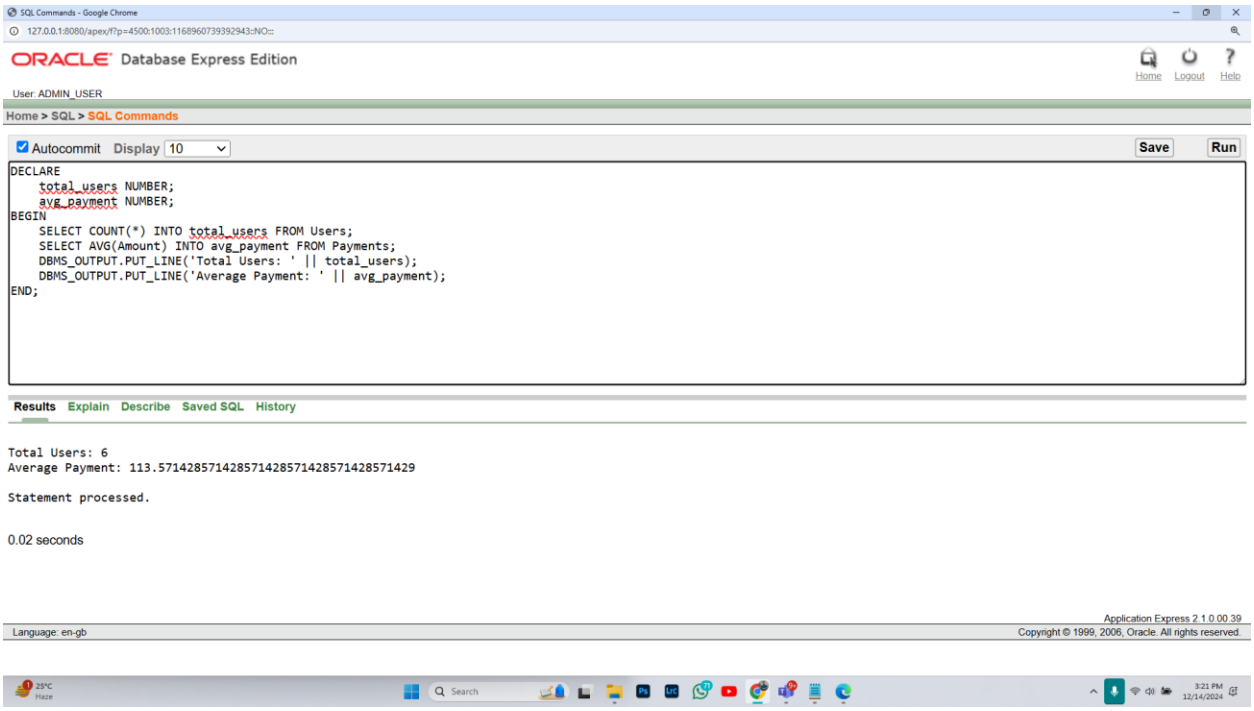
```
DECLARE
user_email VARCHAR2(100);
uppercase_email VARCHAR2(100);
BEGIN
SELECT Email INTO user_email FROM Users WHERE UserID = 1;
uppercase_email := UPPER(user_email);
DBMS_OUTPUT.PUT_LINE('User Email in Uppercase: ' || uppercase_email);
END;
```



Group Functions

1. How can you calculate the total number of users in the `Users` table using a group function.
2. How would you find the average payment amount from the `Payments` table.

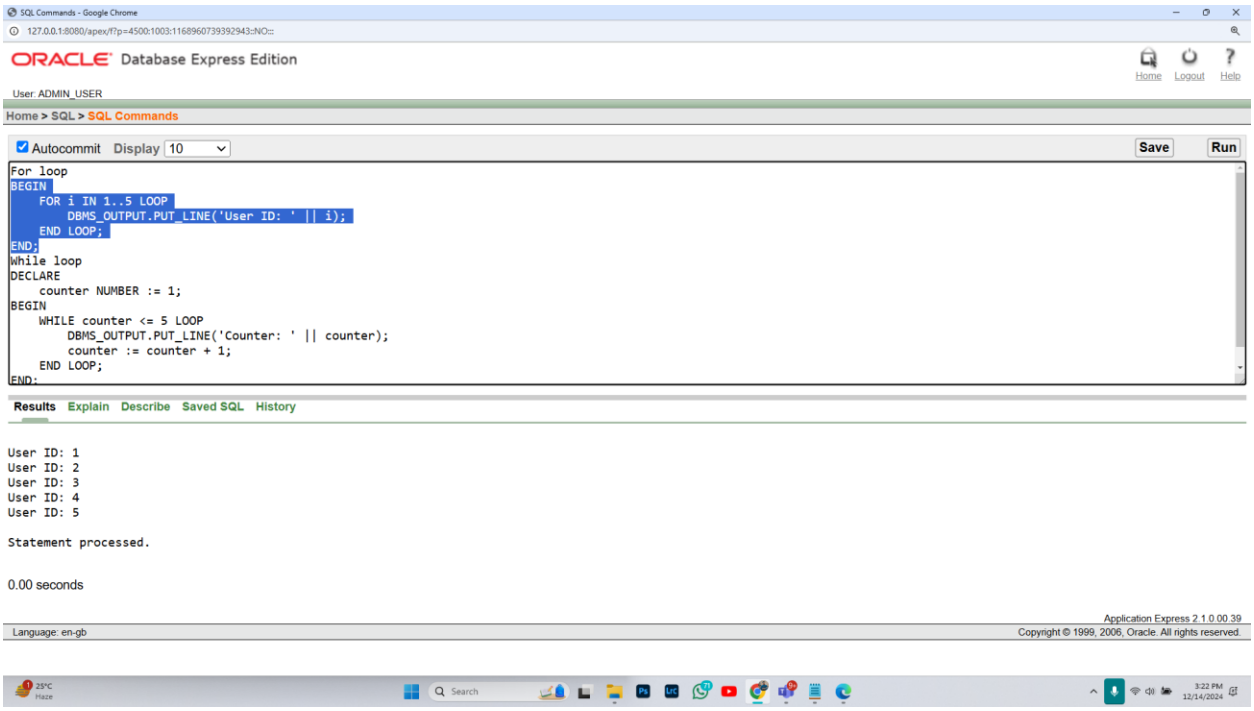
```
DECLARE
total_users NUMBER;
avg_payment NUMBER;
BEGIN
SELECT COUNT(*) INTO total_users FROM Users;
SELECT AVG(Amount) INTO avg_payment FROM Payments;
DBMS_OUTPUT.PUT_LINE('Total Users: ' || total_users);
DBMS_OUTPUT.PUT_LINE('Average Payment: ' || avg_payment);
END;
```



Loops

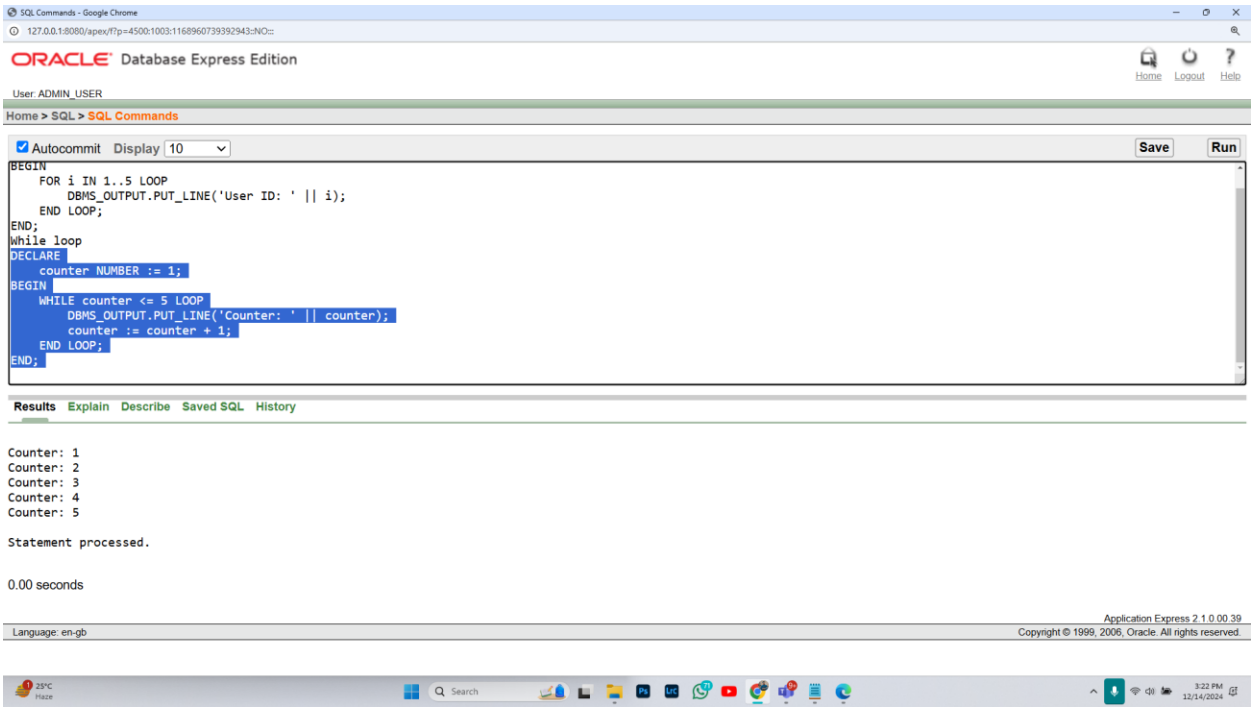
1. Use a FOR loop to iterate through numbers and display a sequence in PL/SQL.

```
BEGIN

FOR i IN 1..5 LOOP

DBMS_OUTPUT.PUT_LINE('User ID: ' || i);

END LOOP;

END;
```



2. How can you use a `WHILE` loop to execute repetitive tasks until a condition is met?

```
DECLARE
counter NUMBER := 1;
BEGIN
WHILE counter <= 5 LOOP
DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
counter := counter + 1;
END LOOP;
END;
```



Conditional Statements

1. Check the `PaymentStatus` and display a message if it is "Completed" using an `IF-THEN-ELSE` statement.

DECLARE

payment_status VARCHAR2(20);

BEGIN

SELECT PaymentStatus INTO payment_status FROM Payments WHERE PaymentID = 2;

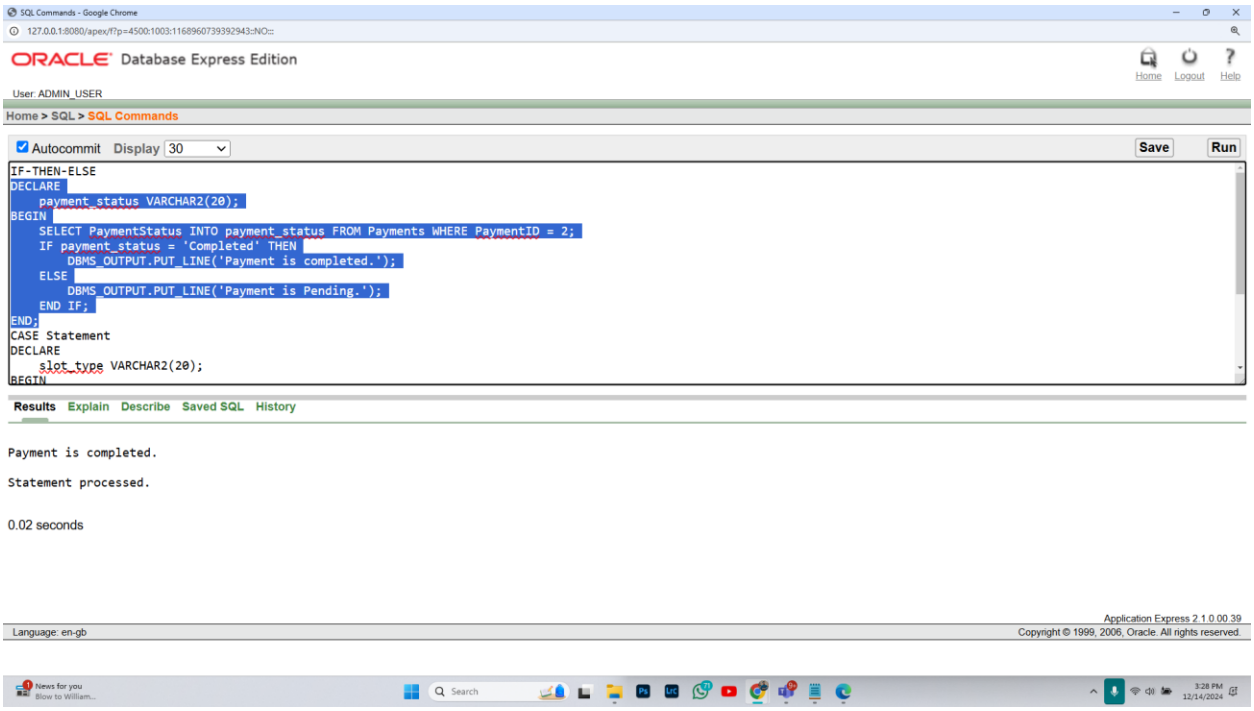
IF payment_status = 'Completed' THEN

DBMS_OUTPUT.PUT_LINE('Payment is completed.');

ELSE DBMS_OUTPUT.PUT_LINE('Payment is pending.');

END IF;

END;



2. Use a CASE statement to display messages based on the slot type in the `ParkingSlots` table.

DECLARE

slot_type VARCHAR2(20);

BEGIN

SELECT SlotType INTO slot_type FROM ParkingSlots WHERE SlotID = 1;

CASE slot_type WHEN 'Car' THEN

DBMS_OUTPUT.PUT_LINE('Car Slot Selected');

WHEN 'Large'

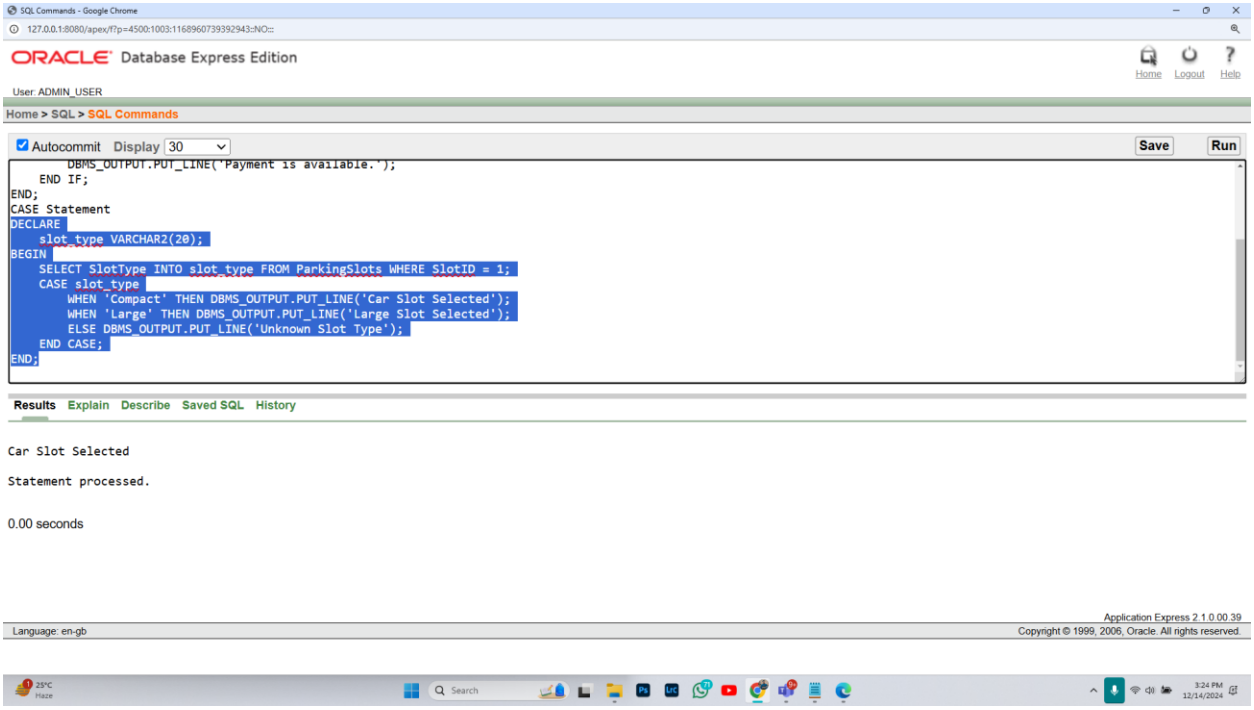
THEN DBMS_OUTPUT.PUT_LINE('Large Slot Selected');

ELSE

DBMS_OUTPUT.PUT_LINE('Unknown Slot Type');

END CASE;

END;



Subqueries

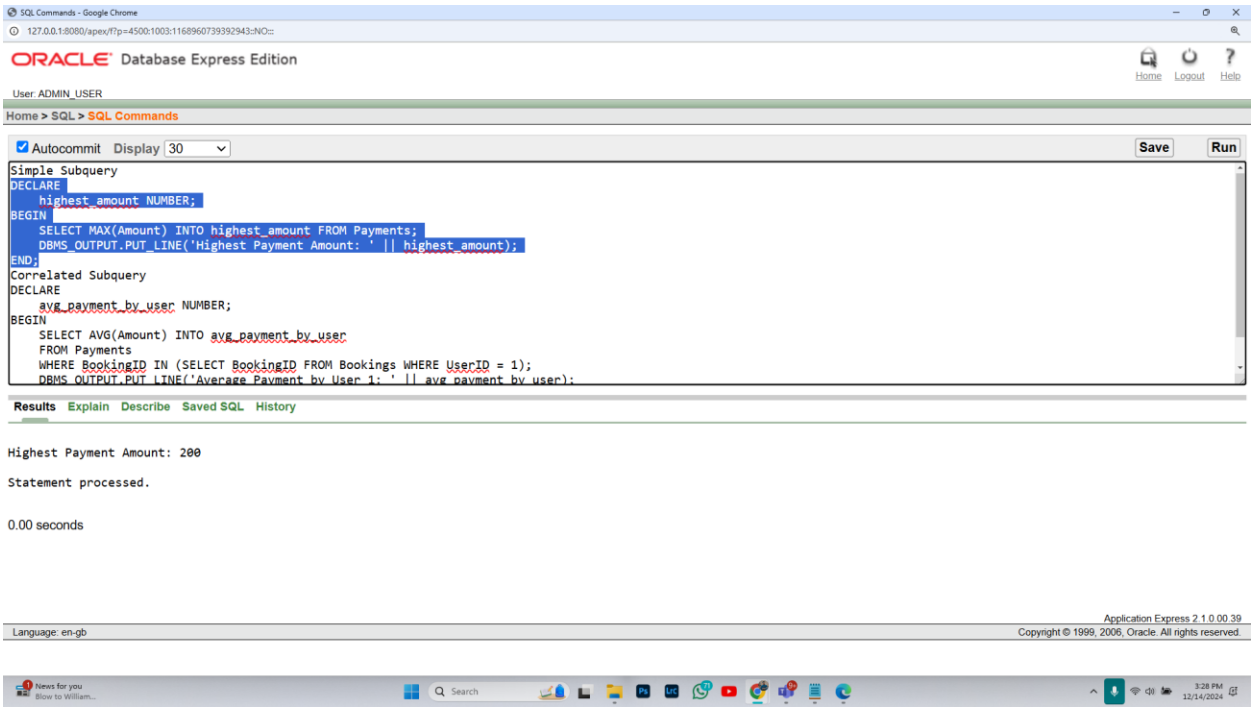
1. Find the highest payment amount in the `Payments` table using a subquery.

DECLARE

highest_amount NUMBER;

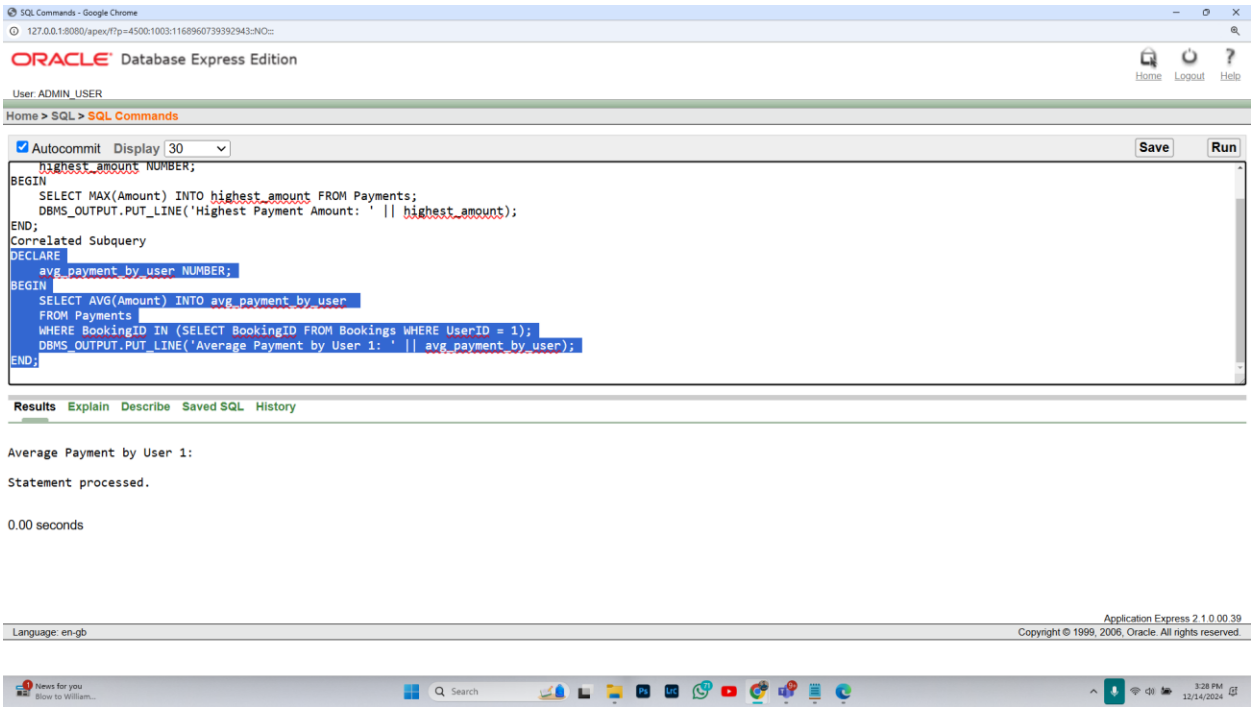
BEGIN SELECT MAX(Amount) INTO highest_amount FROM Payments;
DBMS_OUTPUT.PUT_LINE('Highest Payment Amount: ' || highest_amount);

END;



2. Calculate the average payment for a specific user using a correlated subquery.

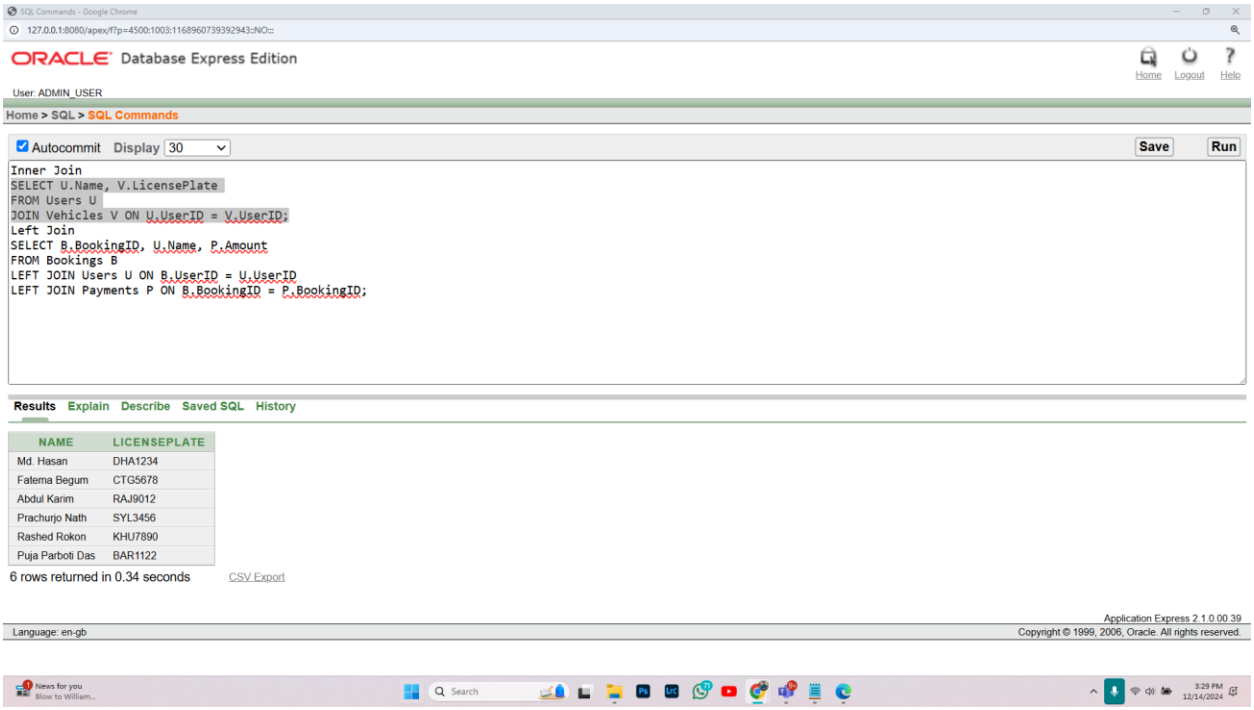
```
DECLARE avg_payment_by_user NUMBER;

BEGIN

SELECT AVG(Amount) INTO avg_payment_by_user FROM Payments WHERE BookingID IN
(SELECT BookingID FROM Bookings WHERE UserID = 1);
DBMS_OUTPUT.PUT_LINE('Average Payment by User 1: ' || avg_payment_by_user);

END;
```



Joining

1. Join the `Users` and `Vehicles` tables to display user names and their associated vehicles.

```
SELECT U.Name, V.LicensePlate
FROM Users U
JOIN Vehicles V ON U.UserID = V.UserID;
```



2. Use a `LEFT JOIN` to retrieve all bookings along with payment details, even if no payment exists.

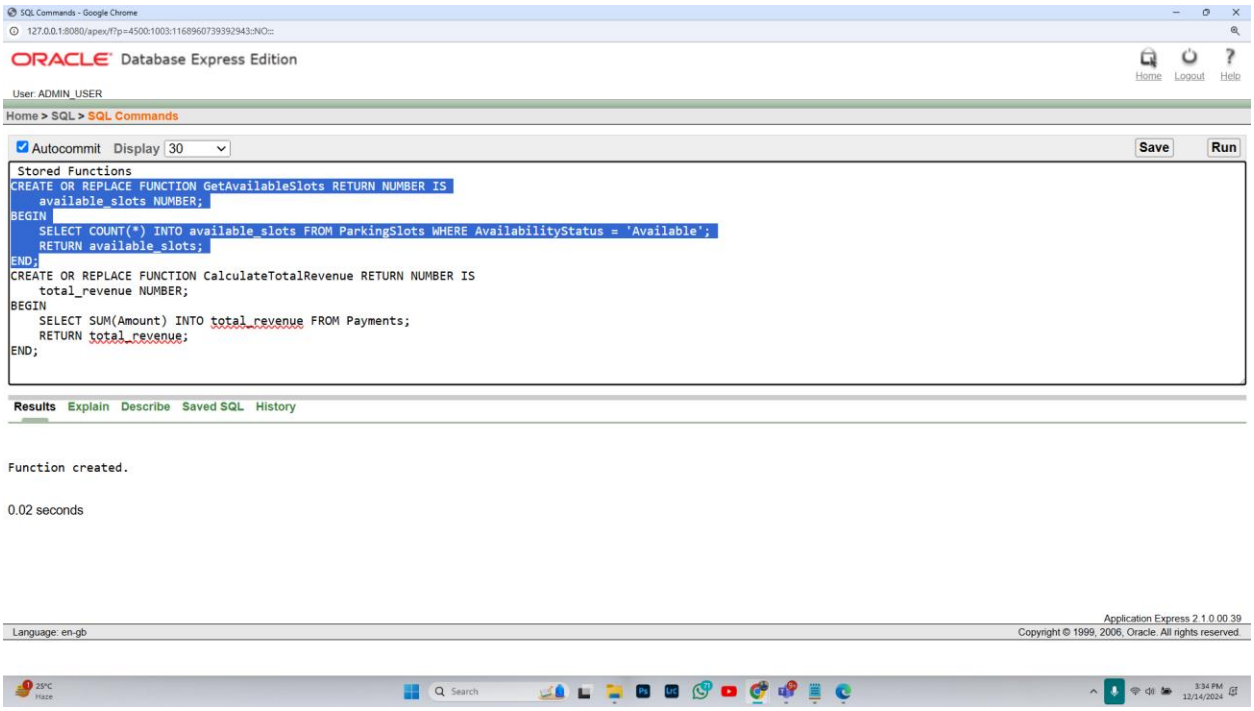
```
SELECT B.BookingID, U.Name, P.Amount
FROM Bookings B
LEFT JOIN Users U ON B.UserID = U.UserID
LEFT JOIN Payments P ON B.BookingID = P.BookingID;
```

Advanced PL/SQL Questions

Stored Functions

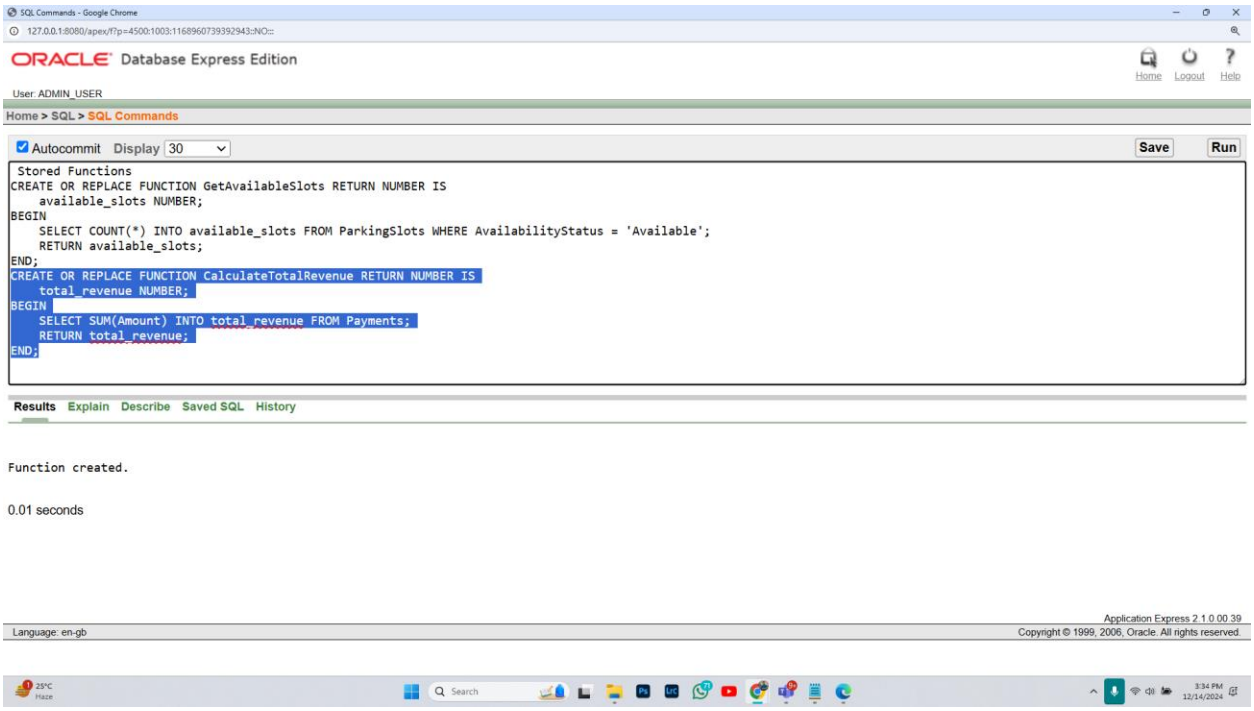
1. Create a stored function to calculate the total available slots in the `ParkingSlots` table.

```
CREATE OR REPLACE FUNCTION GetAvailableSlots RETURN NUMBER IS
   available_slots NUMBER;
BEGIN
   SELECT COUNT(*) INTO available_slots FROM ParkingSlots WHERE AvailabilityStatus =
'Available';
   RETURN available_slots;
END;
```



2. Design a stored function to calculate the total revenue from the `Payments` table.

```
CREATE OR REPLACE FUNCTION CalculateTotalRevenue RETURN NUMBER IS
   total_revenue NUMBER;
BEGIN
   SELECT SUM(Amount) INTO total_revenue FROM Payments;
   RETURN total_revenue;
END;
```



Stored Procedures

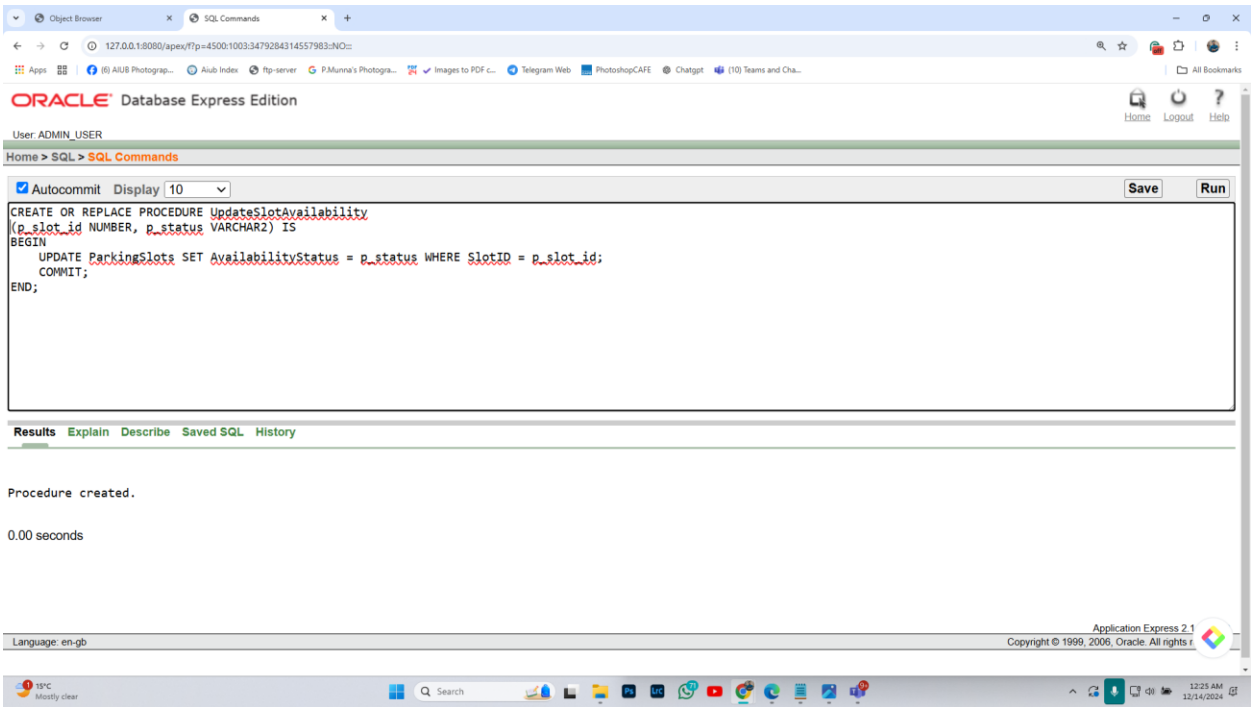
1. Create a stored procedure to update the availability status of a parking slot.

CREATE OR REPLACE PROCEDURE UpdateSlotAvailability(p_slotID IN NUMBER, p_status IN VARCHAR2) AS

BEGIN

UPDATE ParkingSlots SET AvailabilityStatus = p_status WHERE SlotID = p_slotID;

END;



2. Design a stored procedure to delete bookings older than 30 days.

```
CREATE OR REPLACE PROCEDURE DeleteOldBookings IS
BEGIN
    DELETE FROM Bookings WHERE BookingDateTime < SYSDATE - 30;
END;
```

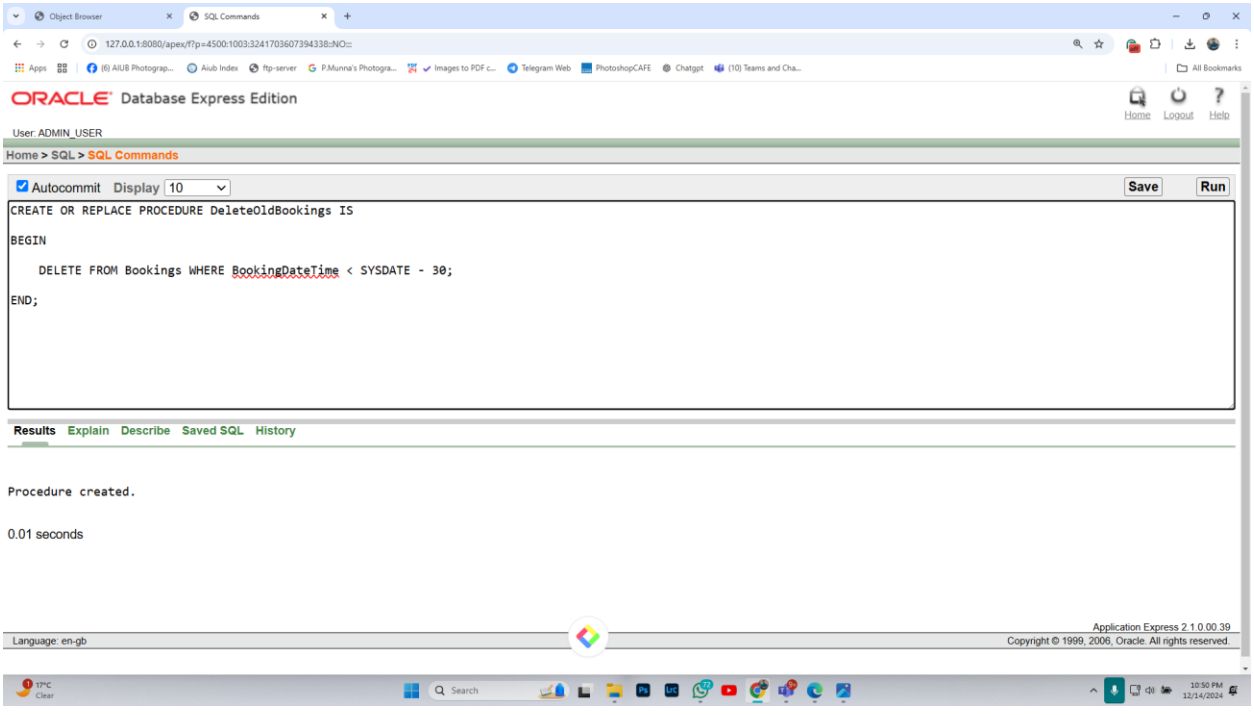


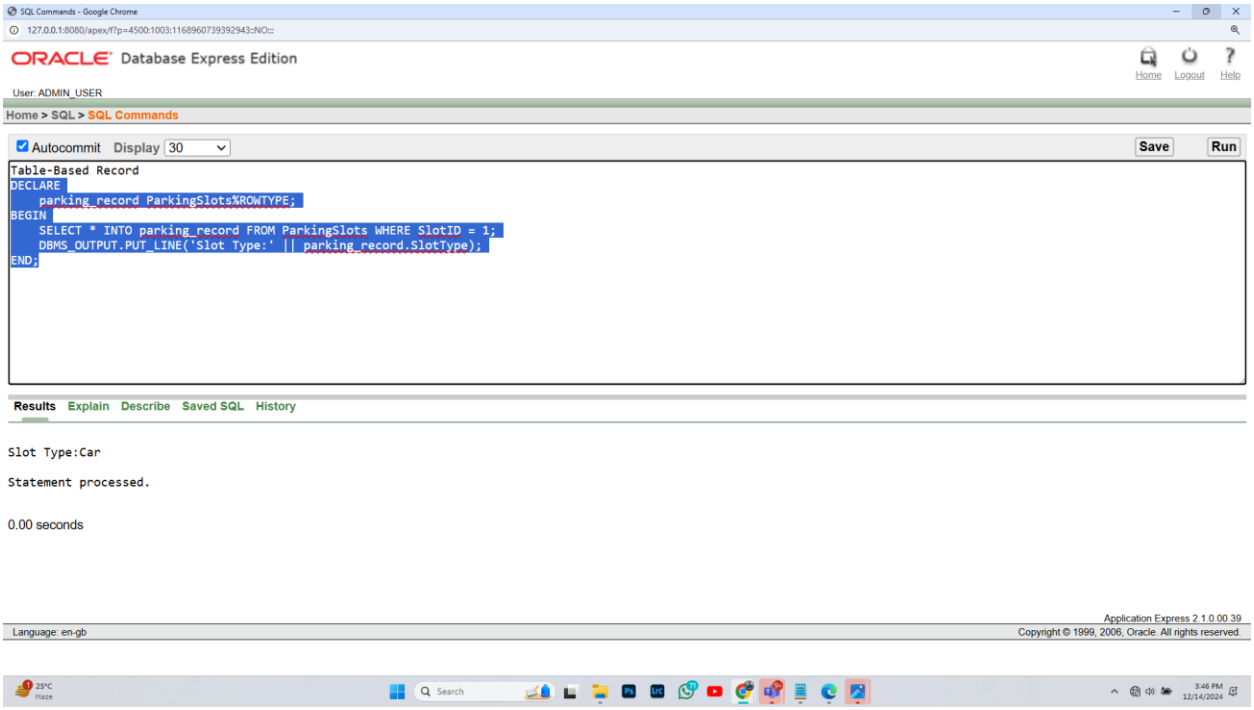
Table-Based Record

1. Declare a table-based record for the `ParkingSlots` table and retrieve all fields for a specific slot.

DECLARE parking_record ParkingSlots%ROWTYPE;

BEGIN SELECT * INTO parking_record FROM ParkingSlots WHERE SlotID = 1;
DBMS_OUTPUT.PUT_LINE('Slot Type: ' || parking_record.SlotType);

END;



2. Use a table-based record to fetch and display the details of a parking slot.

DECLARE parking_record ParkingSlots%ROWTYPE;

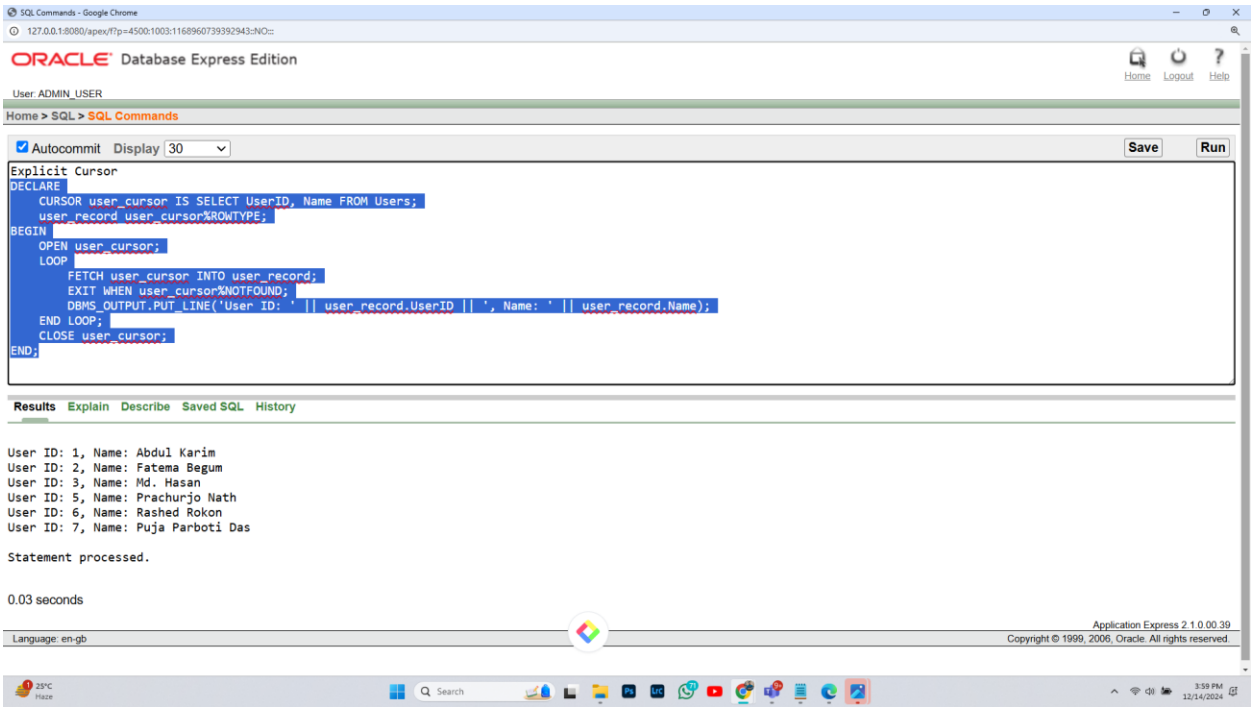
BEGIN SELECT * INTO parking_record FROM ParkingSlots WHERE SlotID = 1;
DBMS_OUTPUT.PUT_LINE('Slot Type: ' || parking_record.SlotType);

END; /

Explicit Cursor

1. Use an explicit cursor to fetch and display all user IDs and names from the `Users` table.
2. Loop through an explicit cursor to display all records in the `Bookings` table.

```
DECLARE CURSOR user_cursor IS

SELECT UserID, Name FROM Users;

user_record user_cursor%ROWTYPE;

BEGIN OPEN user_cursor;

LOOP

FETCH user_cursor INTO user_record;

EXIT WHEN user_cursor%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('User ID: ' || user_record.UserID || ', Name: ' ||
user_record.Name);

END LOOP;

CLOSE user_cursor;

END;
```



Cursor-Based Record

1. Declare a cursor-based record for the `Bookings` table to fetch and display its details.
2. Use a cursor-based record to retrieve booking IDs and their statuses.

DECLARE CURSOR booking_cursor IS

SELECT BookingID, Status FROM Bookings;

booking_record booking_cursor%ROWTYPE;

BEGIN

OPEN booking_cursor;

LOOP

FETCH booking_cursor INTO booking_record;

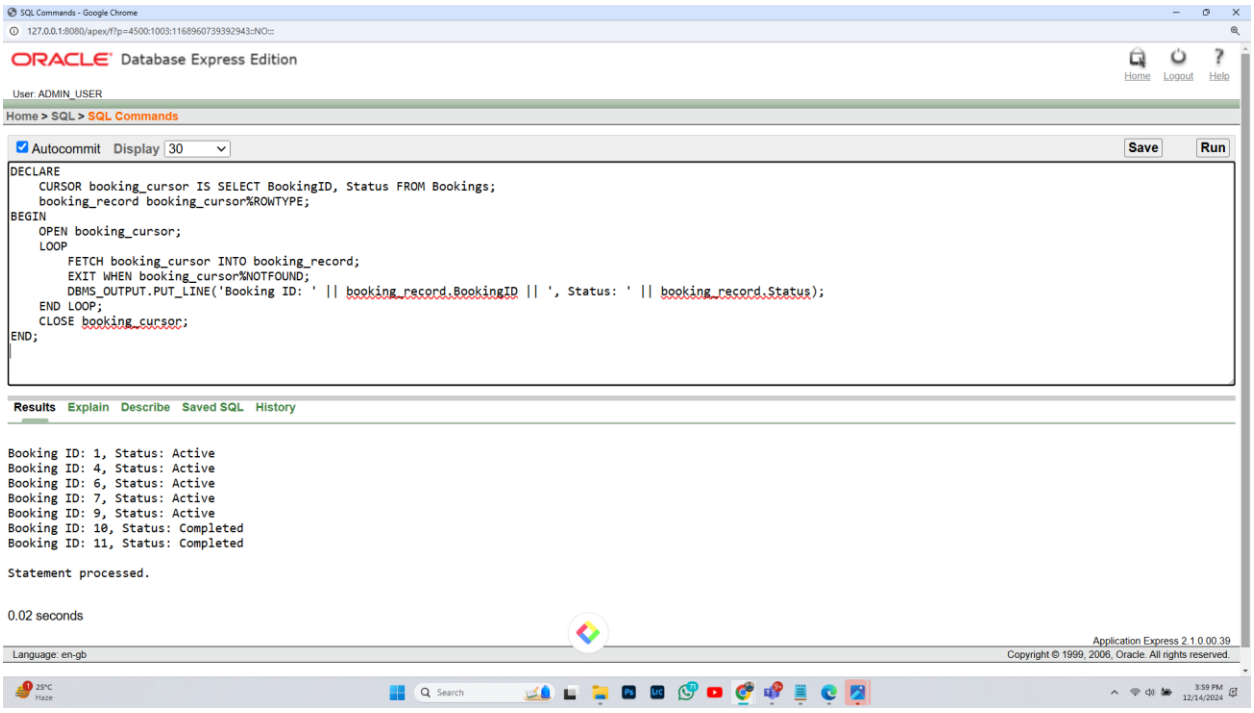
EXIT WHEN booking_cursor%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('Booking ID: ' || booking_record.BookingID || ', Status: ' || booking_record.Status);

END LOOP;

CLOSE booking_cursor;

END;



Row-Level Trigger

1. Create a row-level trigger to update the slot availability status after a new booking is inserted.
2. Create a trigger executes only for a specific slot ID.

CREATE OR REPLACE TRIGGER UpdateSlotOnBooking

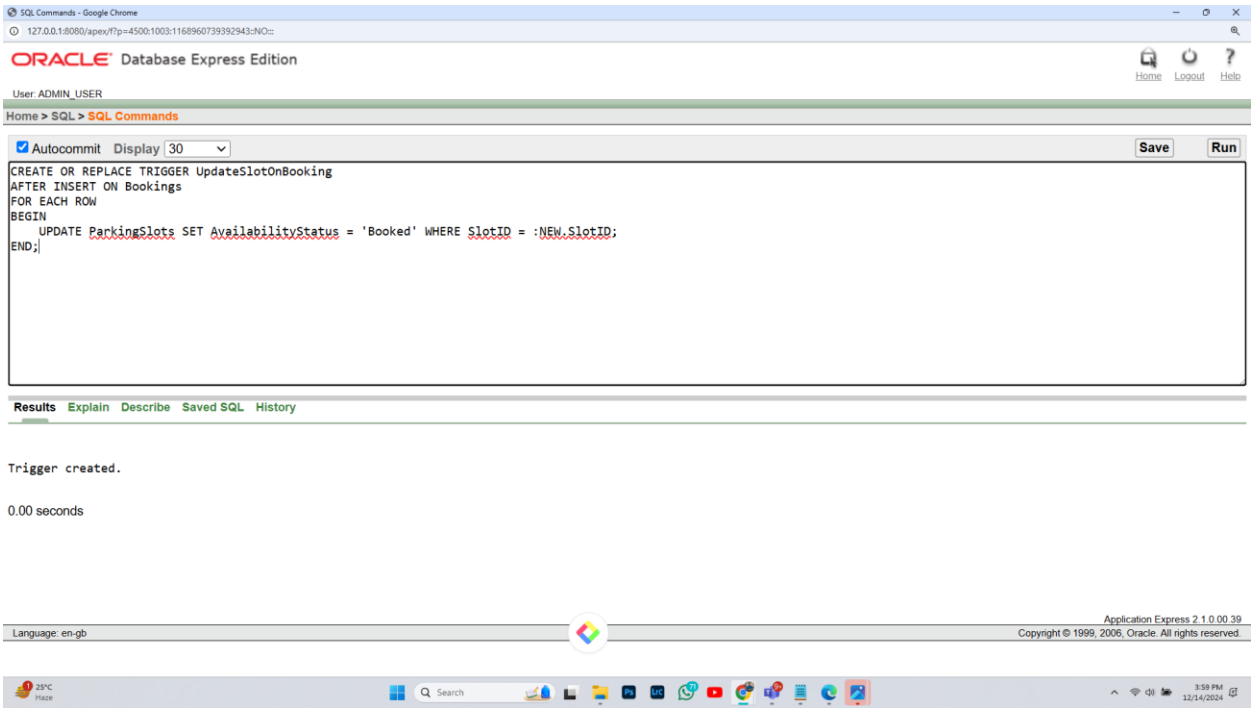
AFTER INSERT ON Bookings

FOR EACH ROW

BEGIN

UPDATE ParkingSlots SET AvailabilityStatus = 'Booked' WHERE SlotID = :NEW.SlotID;

END;



Statement-Level Trigger

1. Create a statement-level trigger to log all new bookings in a separate log table.
2. Design a trigger that captures the timestamp of a new booking record.

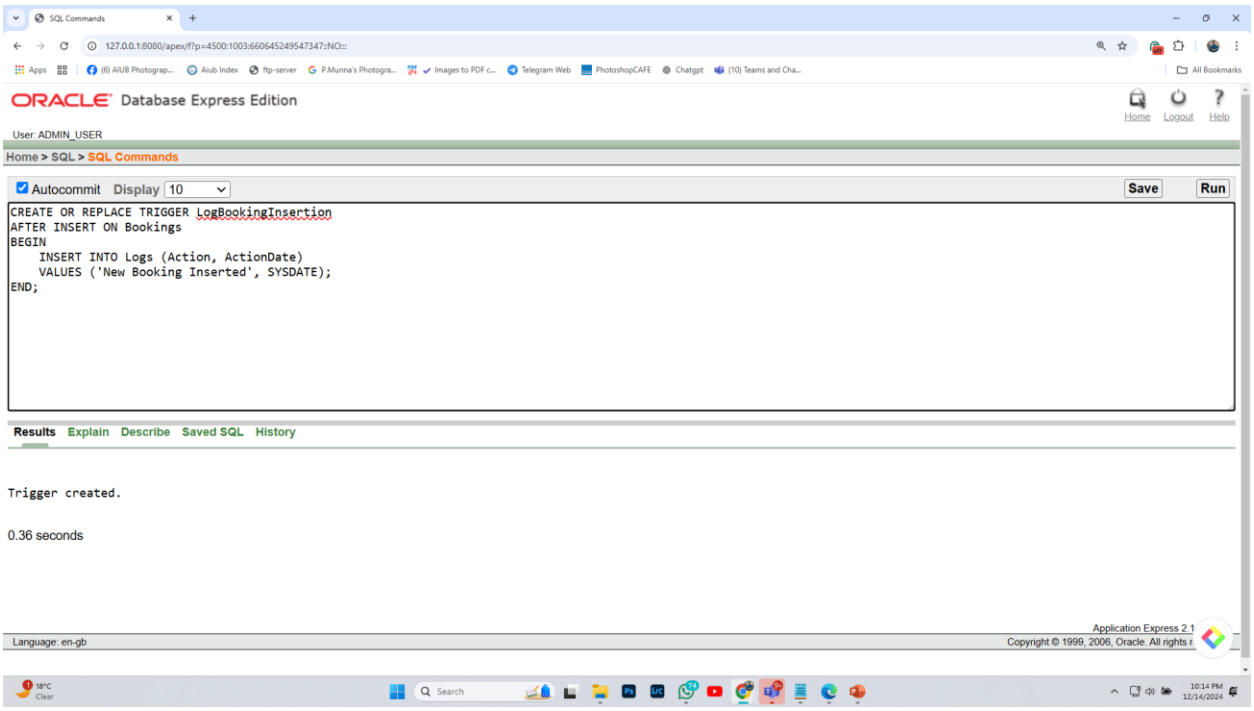
CREATE OR REPLACE TRIGGER LogBookingInsertion

AFTER INSERT ON Bookings

BEGIN

INSERT INTO Logs (Action, ActionDate) VALUES ('New Booking Inserted', SYSDATE);

END;



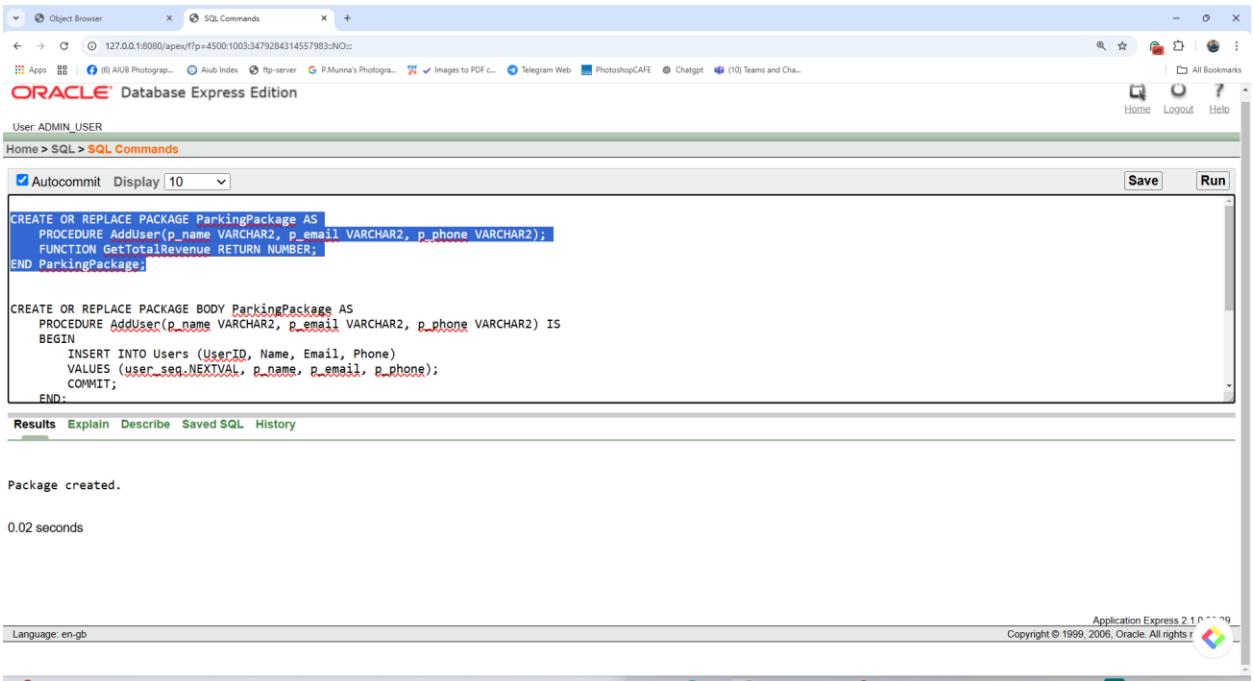
Package

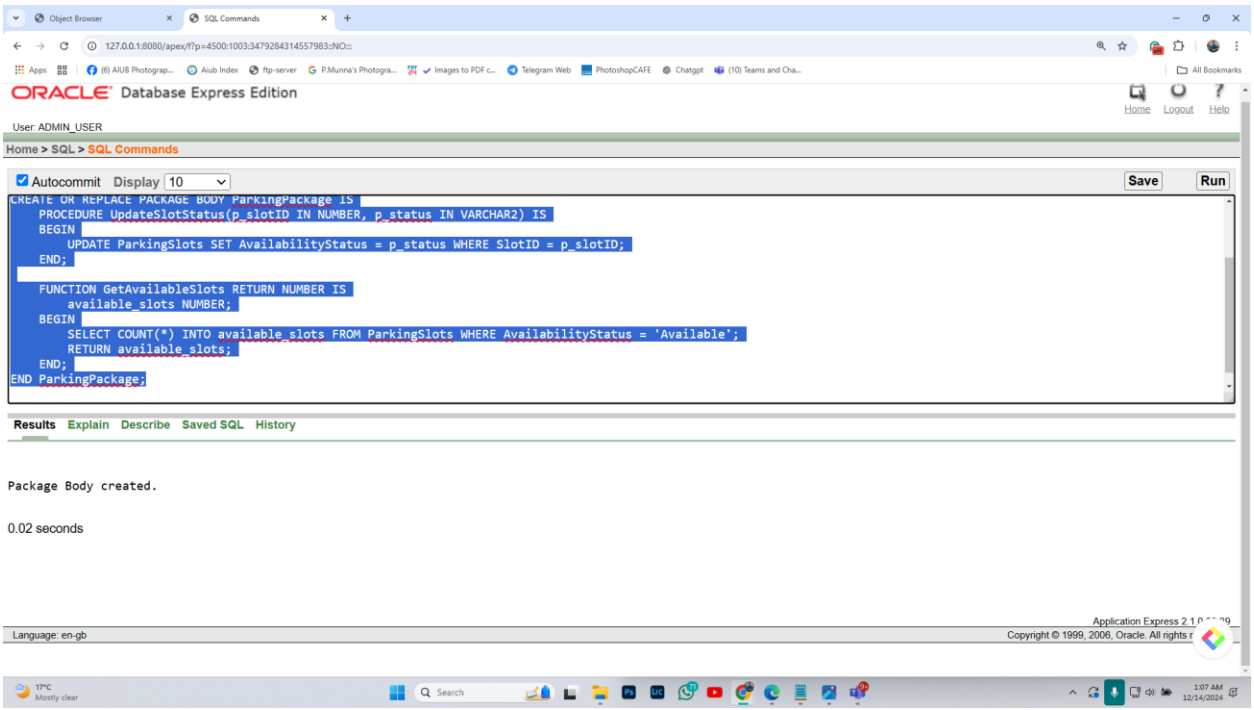
1. Define a package to include procedures and functions for managing parking slots.
2. Use a package to encapsulate the logic for calculating available slots and updating their. status.

CREATE OR REPLACE PACKAGE ParkingPackage IS PROCEDURE UpdateSlotStatus(p_slotID IN NUMBER, p_status IN VARCHAR2); FUNCTION GetAvailableSlots RETURN NUMBER; END ParkingPackage; /

CREATE OR REPLACE PACKAGE BODY ParkingPackage IS PROCEDURE UpdateSlotStatus(p_slotID IN NUMBER, p_status IN VARCHAR2) IS BEGIN UPDATE ParkingSlots SET AvailabilityStatus = p_status WHERE SlotID = p_slotID; END;

```
FUNCTION GetAvailableSlots RETURN NUMBER IS
    available_slots NUMBER;
BEGIN
    SELECT COUNT(*) INTO available_slots FROM ParkingSlots WHERE
AvailabilityStatus = 'Available';
    RETURN available_slots;
END;


END ParkingPackage;
```


ORACLE Database Express Edition
User: ADMIN_USER
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
CREATE OR REPLACE PACKAGE ParkingPackage AS
PROCEDURE AddUser(p_name VARCHAR2, p_email VARCHAR2, p_phone VARCHAR2);
FUNCTION GetTotalRevenue RETURN NUMBER;
END ParkingPackage;
CREATE OR REPLACE PACKAGE BODY ParkingPackage AS
PROCEDURE AddUser(p_name VARCHAR2, p_email VARCHAR2, p_phone VARCHAR2) IS
BEGIN
INSERT INTO Users (UserID, Name, Email, Phone)
VALUES (user_seq.NEXTVAL, p_name, p_email, p_phone);
COMMIT;
END;
Results Explain Describe Saved SQL History
Package created.
0.02 seconds
Language: en-gb
Copyright © 1999, 2006, Oracle. All rights r


ORACLE Database Express Edition
User: ADMIN_USER
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
CREATE OR REPLACE PACKAGE BODY ParkingPackage IS
PROCEDURE UpdateSlotStatus(p_slotID IN NUMBER, p_status IN VARCHAR2) IS
BEGIN
UPDATE ParkingSlots SET AvailabilityStatus = p_status WHERE SlotID = p_slotID;
END;
FUNCTION GetAvailableSlots RETURN NUMBER IS
available_slots NUMBER;
BEGIN
SELECT COUNT(*) INTO available_slots FROM ParkingSlots WHERE AvailabilityStatus = 'Available';
RETURN available_slots;
END;
END ParkingPackage;
Results Explain Describe Saved SQL History
Package Body created.
0.02 seconds
Language: en-gb
Copyright © 1999, 2006, Oracle. All rights r

Future Work

To improve the existing project, the following enhancements are proposed:

1. **Integration with IoT and Real-Time Monitoring:**

Add IoT sensors for real-time detection of occupied or available parking slots. This integration can update the database dynamically and provide real-time feedback to users.

2. **Predictive Analytics and Optimization:**

Implement machine learning algorithms to predict parking demand based on historical data. This can optimize slot allocation and help users find slots faster during peak hours.

3. **Enhanced Security Measures:**

Add features like two-factor authentication for user accounts and encryption for sensitive data (e.g., payment details).

4. **Dynamic Pricing Model:**

Introduce dynamic pricing based on factors such as location, demand, and time of day. This model can be integrated into the payment system for more efficient revenue generation.

5. **Comprehensive Reporting System:**

Create detailed analytics dashboards for parking lot administrators to monitor usage patterns, revenue trends, and maintenance schedules.

6. **Automated Conflict Resolution:**Develop a conflict resolution mechanism to handle overbooking or erroneous slot allocation automatically, minimizing user complaints.

Conclusion

The Parking Management System database project was successfully designed and implemented to meet the requirements of efficient parking slot allocation, user and vehicle management, booking and payment processing, and real-time slot availability tracking. Through the use of advanced PL/SQL features such as stored procedures, triggers, cursors, and packages, the system was able to handle complex database operations while maintaining data integrity and security.

Key findings of the project include:

- **Efficiency in Slot Management:** The system automates the booking and slot allocation process, reducing manual errors.
- **Data Integrity:** Constraints like primary and foreign keys ensure consistent and reliable data.
- **Enhanced Security:** User roles and privileges were implemented to restrict unauthorized access.
- **Scalability:** The normalized database structure can handle additional tables and features with minimal modification.