Database Management System

Project Report
On

BIKE RENTAL SYSTEM

By : Nilay Singh

INDEX

1. Introduction

With the growing popularity of shared mobility services and the increasing demand for sustainable urban transportation solutions, the development of an efficient, user-friendly, and secure Bike Rental System has become more essential than ever. The rapid urbanisation in cities across the globe has led to congestion, pollution, and transportation inefficiencies, prompting the need for alternative mobility options. This project aims to address these challenges by offering a robust digital solution that enables users to conveniently locate, reserve, and rent bikes through an integrated online platform.

The Bike Rental System is designed to streamline the process of renting bikes for a wide range of users, including urban commuters seeking eco-friendly alternatives to traditional vehicles, tourists exploring cities at their own pace, and delivery riders in need of reliable short-term transport. Through a responsive and intuitive interface, users can register an account, browse available bikes based on location, check real-time availability, select rental durations, and complete secure payments using various methods such as credit/debit cards, digital wallets, or UPI.

In addition to the user-facing features, the system provides a comprehensive administrative panel that empowers system managers to monitor and control operations efficiently. Admins have access to functionalities such as inventory management, booking oversight, user management, system analytics, and maintenance scheduling. This dual-layered approach ensures a balanced experience for both users and system operators.

Technically, the system’s backend is structured around a relational database model, utilizing a detailed Entity-Relationship (ER) diagram to represent and manage core entities such as Users, Bikes, Bookings, Locations, Payments, and Admins. This architecture ensures scalability, consistency, and ease of maintenance, making it suitable for deployment in both small-scale communities and large urban environments. Furthermore, the modular design facilitates future enhancements, such as integration with GPS tracking, mobile app support, or dynamic pricing models.

Ultimately, the Bike Rental System not only enhances urban mobility but also contributes to broader goals such as reducing traffic congestion, minimising carbon footprints, and promoting healthier, more active lifestyles. By digitising and

automating the rental process, this system paves the way for a smarter and more sustainable future in urban transportation.

Project Resource Requirements:

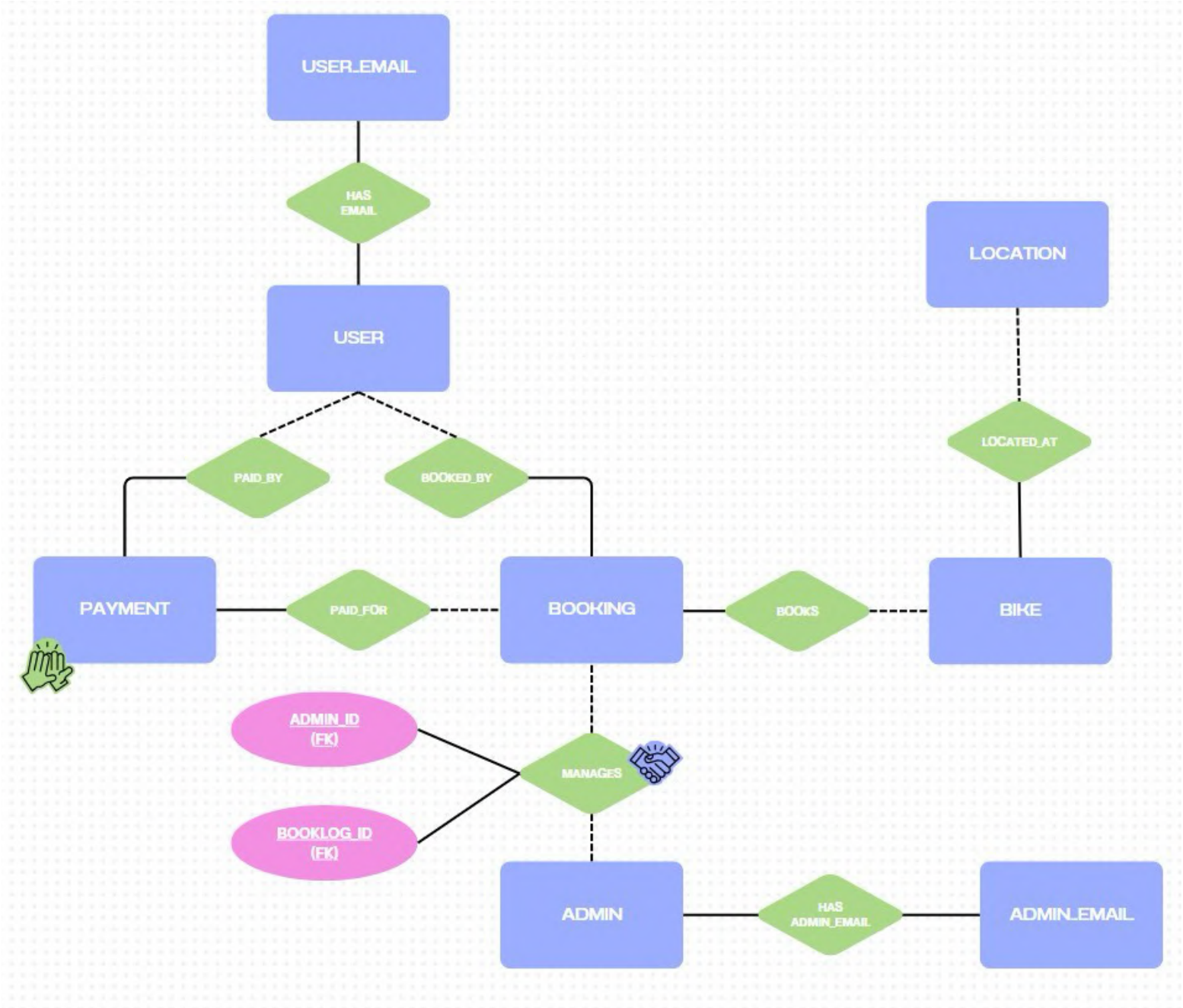
A database is an application that stores the organised collection of records. It can be accessed and manage by the user very easily. Today, many databases available like MySQL, Sybase , Oracle, MongoDB, PostgreSQL, SQL Server, etc. We used MySQL server for managing data.

MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. MySQL is a widely used relational database management system (RDBMS). It is free and open-source. MySQL is ideal for both small and large applications.

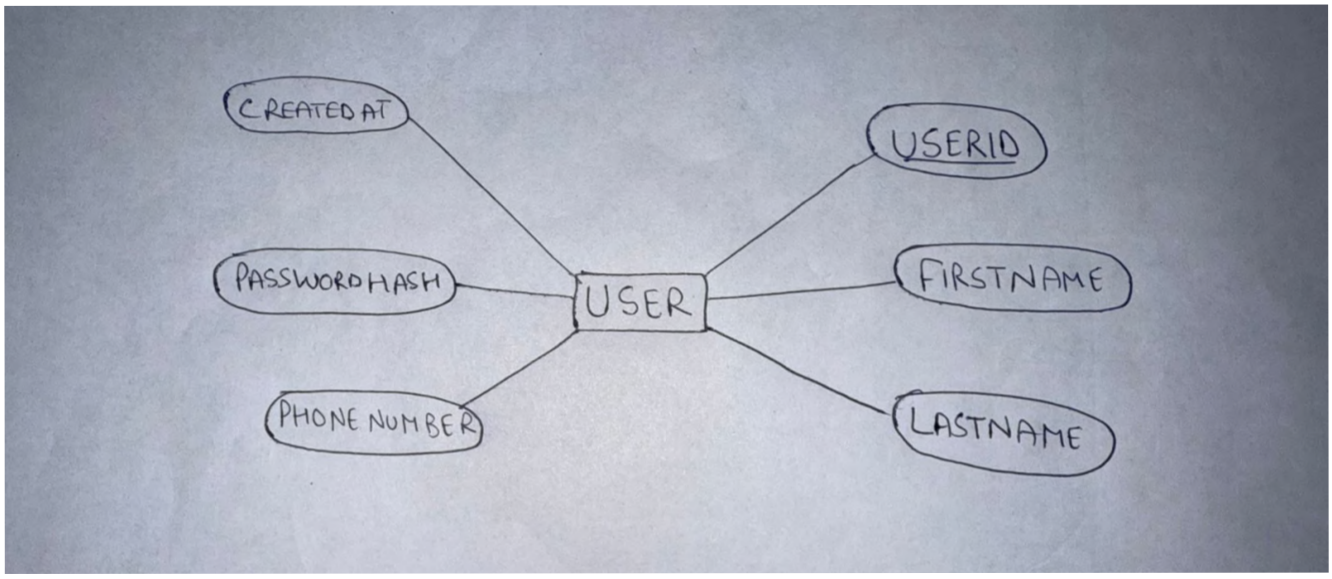
MySQL follows the working of Client-Server Architecture. This model is designed for the end-users called clients to access the resources from a central computer known as a server using network services. Here, the clients make requests through a graphical user interface (GUI), and the server will give the desired output as soon as the instructions are matched. The process of MySQL environment is the same as the client-server model.

MySQL is a very powerful program that can handle a large set of functionality of the most expensive and powerful database packages. It supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

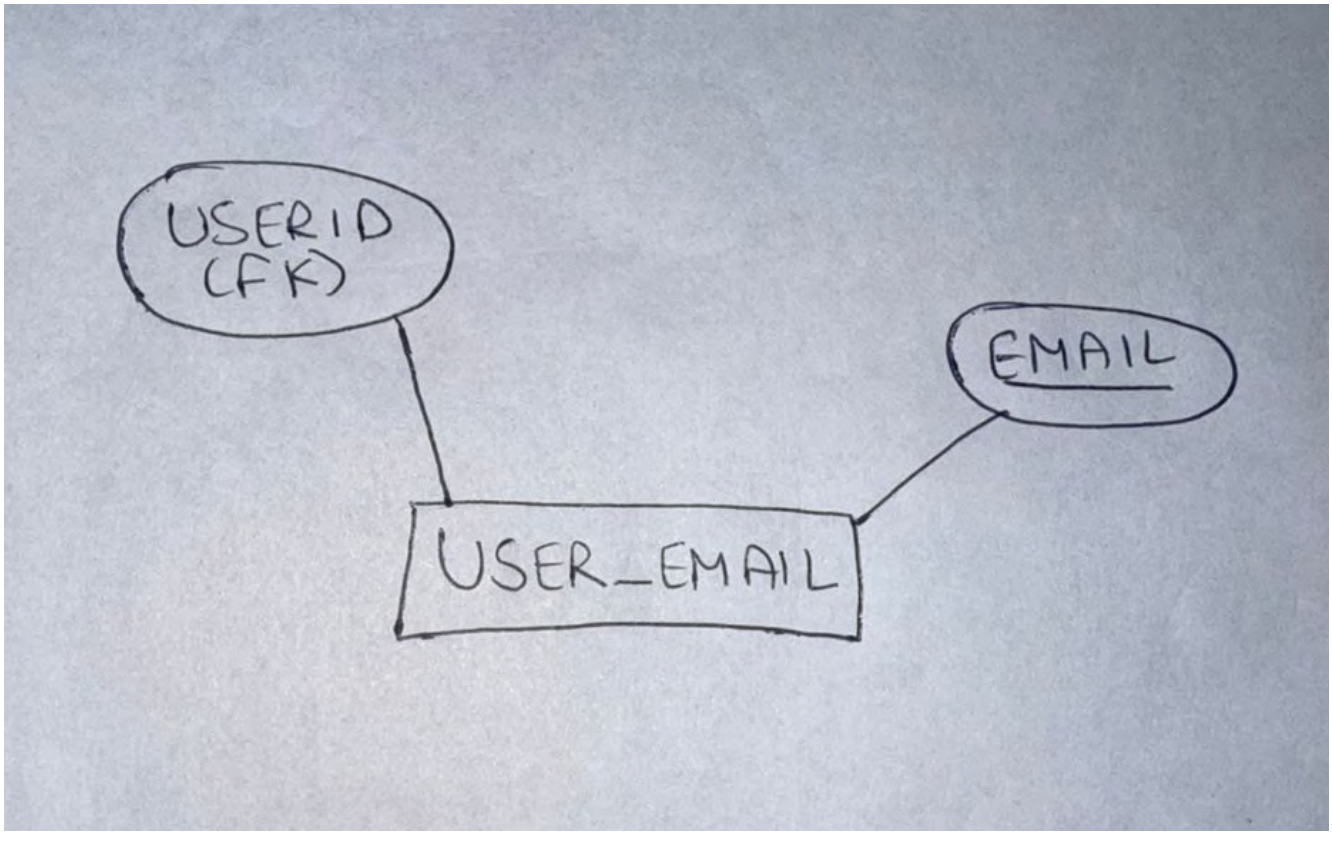
2. ER-DIAGRAM


USER_EMAIL
HAS EMAIL
USER
LOCATION
LOCATED_AT
PAID_BY
BOOKED_BY
PAYMENT
PAID_FOR
BOOKING
BOOKS
BIKE
ADMIN_ID (FK)
MANAGES
BOOKLOG_ID (FK)
ADMIN
HAS ADMIN_EMAIL
ADMIN_EMAIL

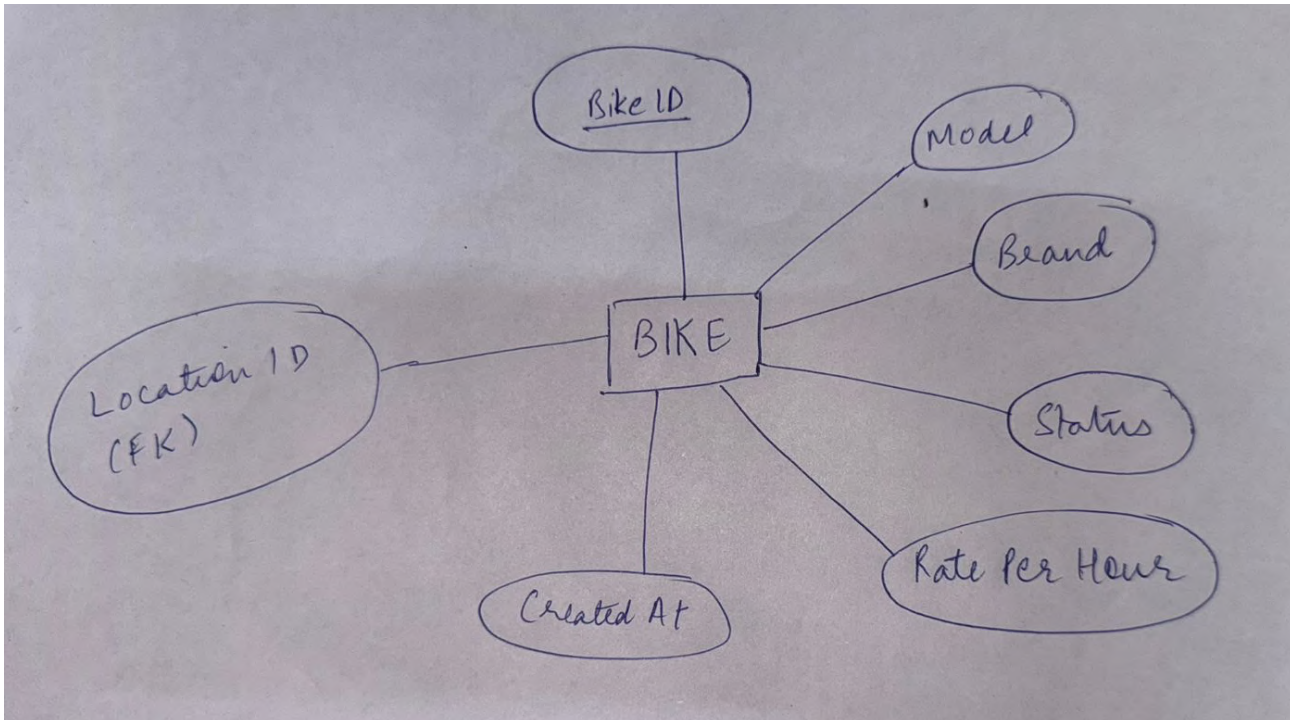
1. USER



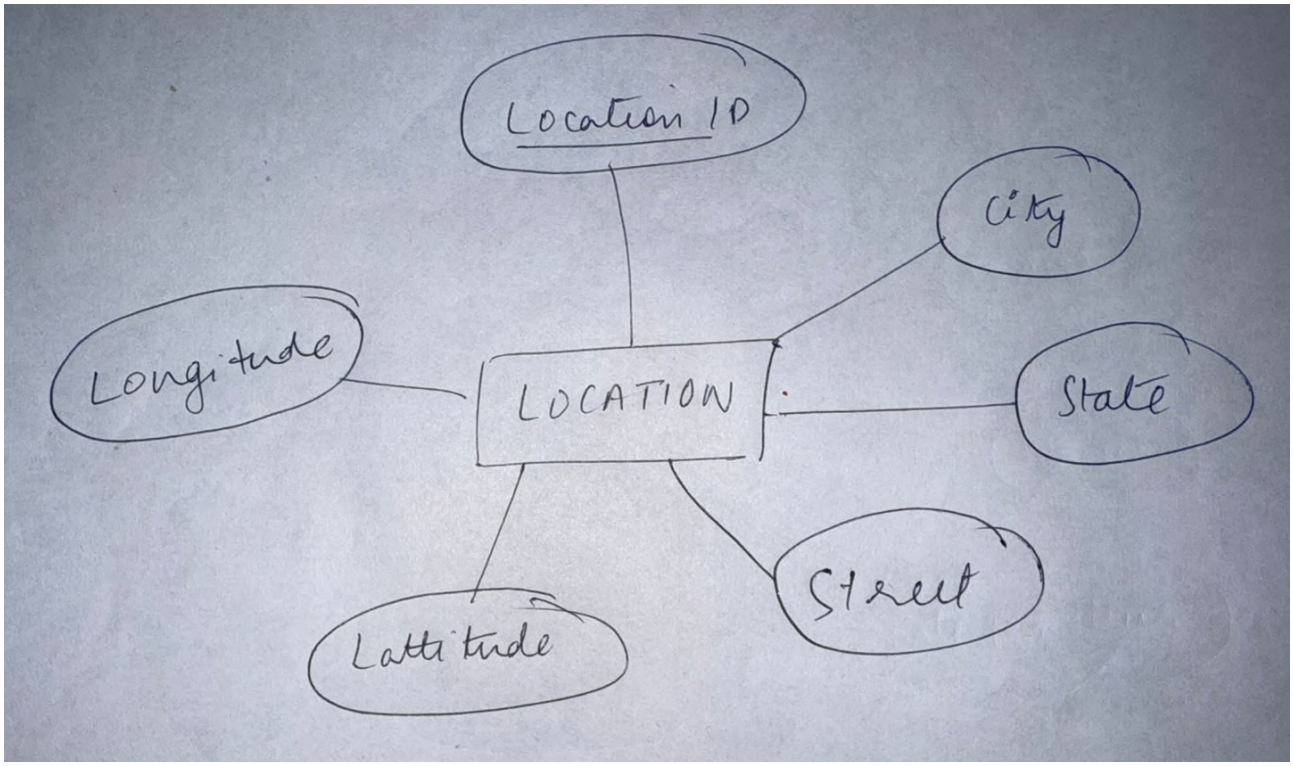
2. USER_EMAIL



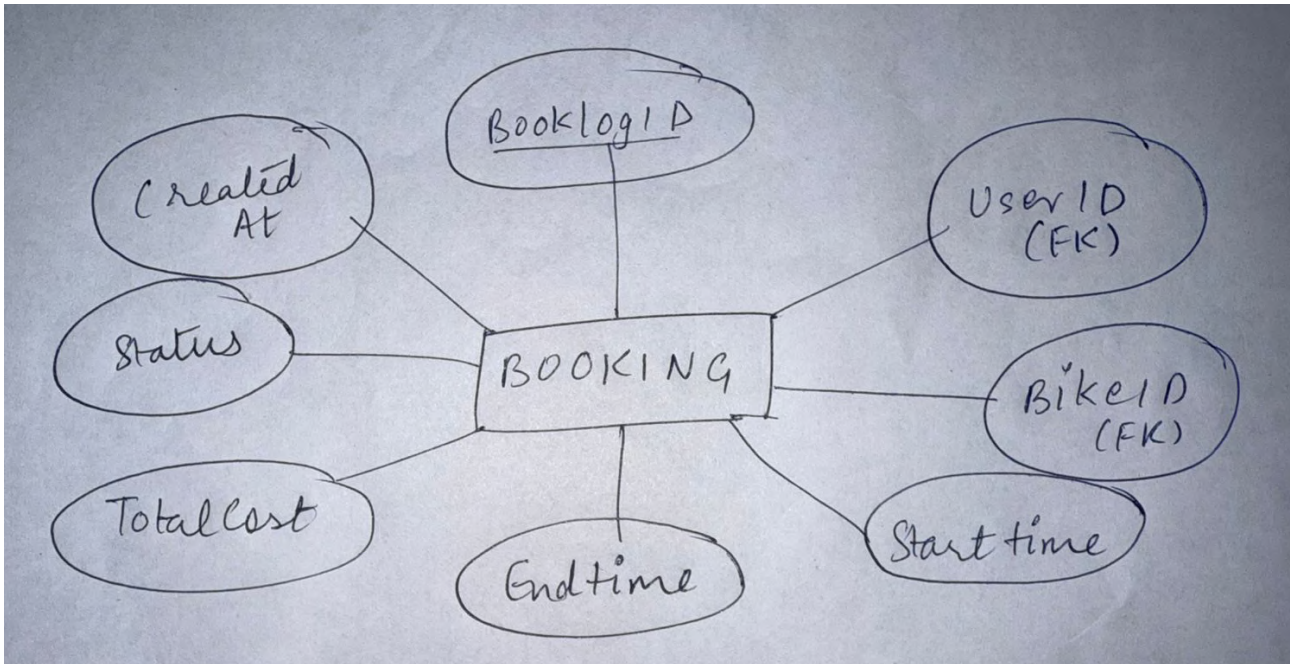
3. BIKE



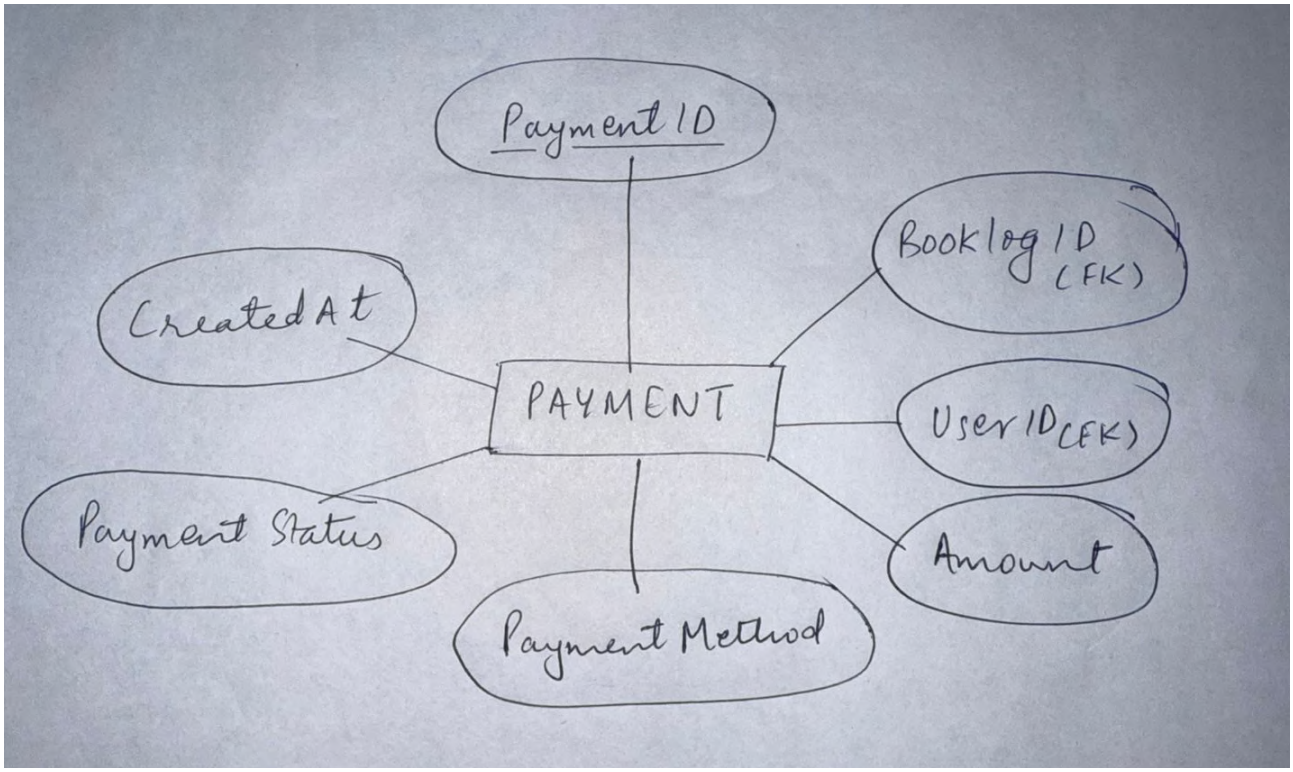
4. LOCATION



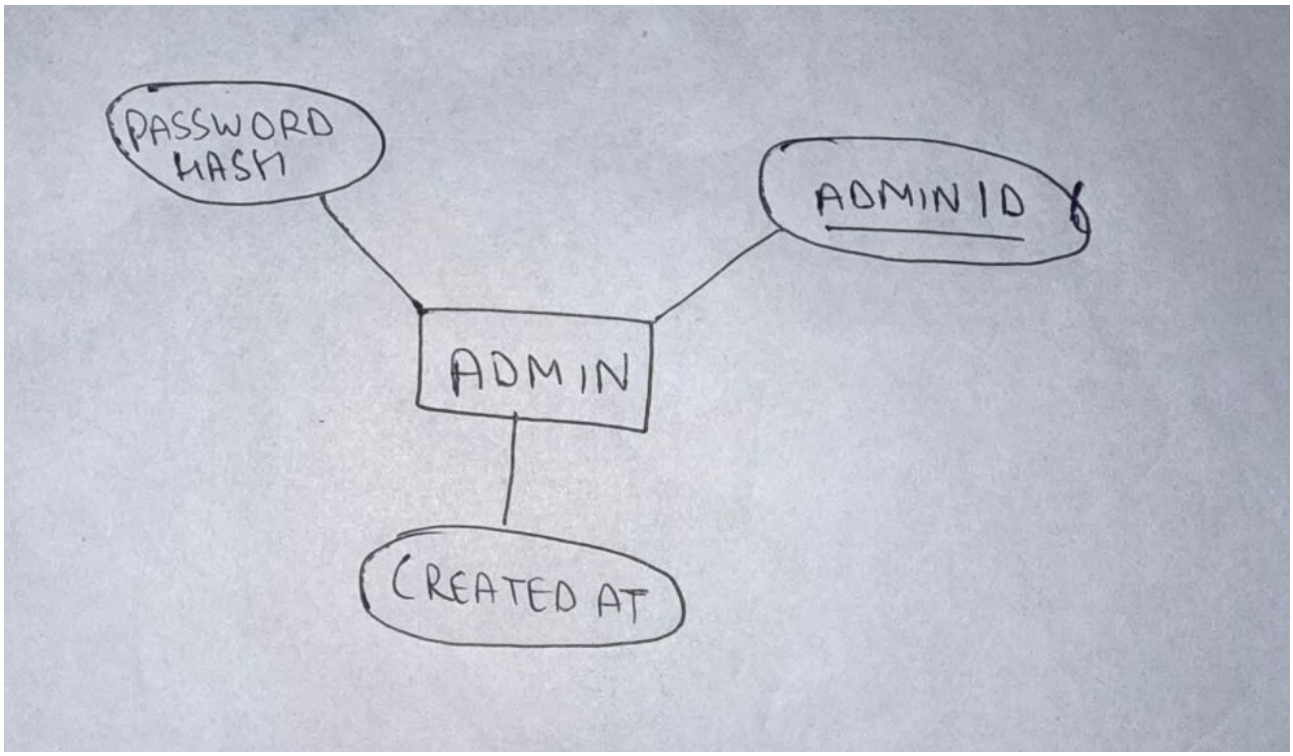
5. BOOKING



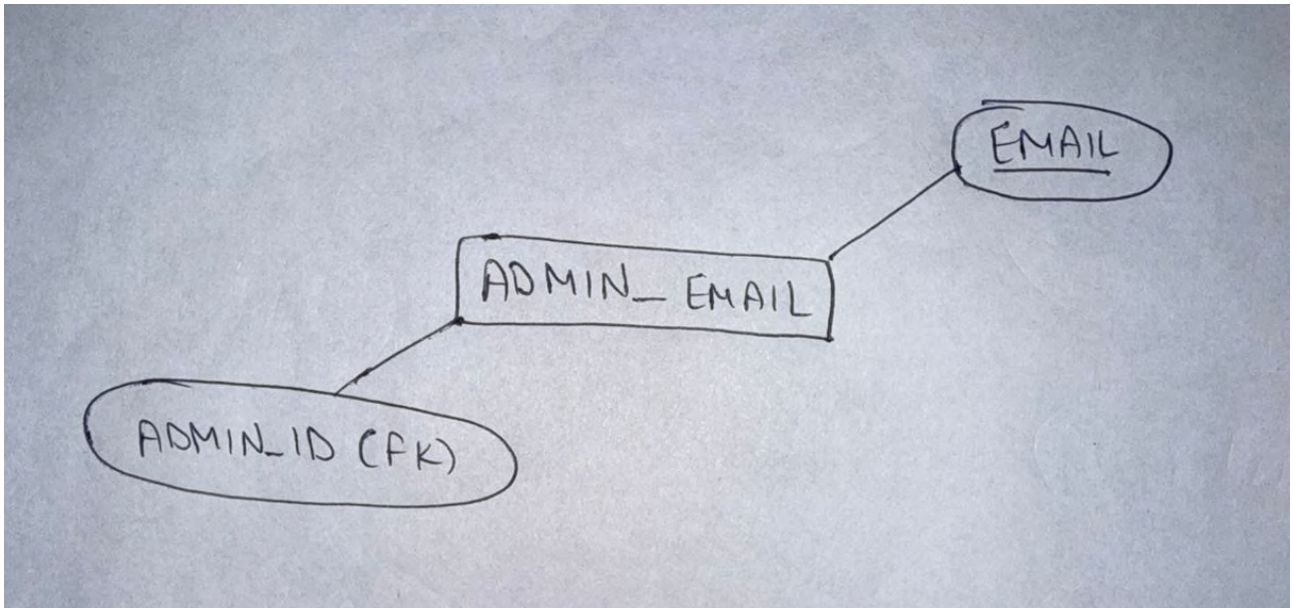
6. PAYMENT



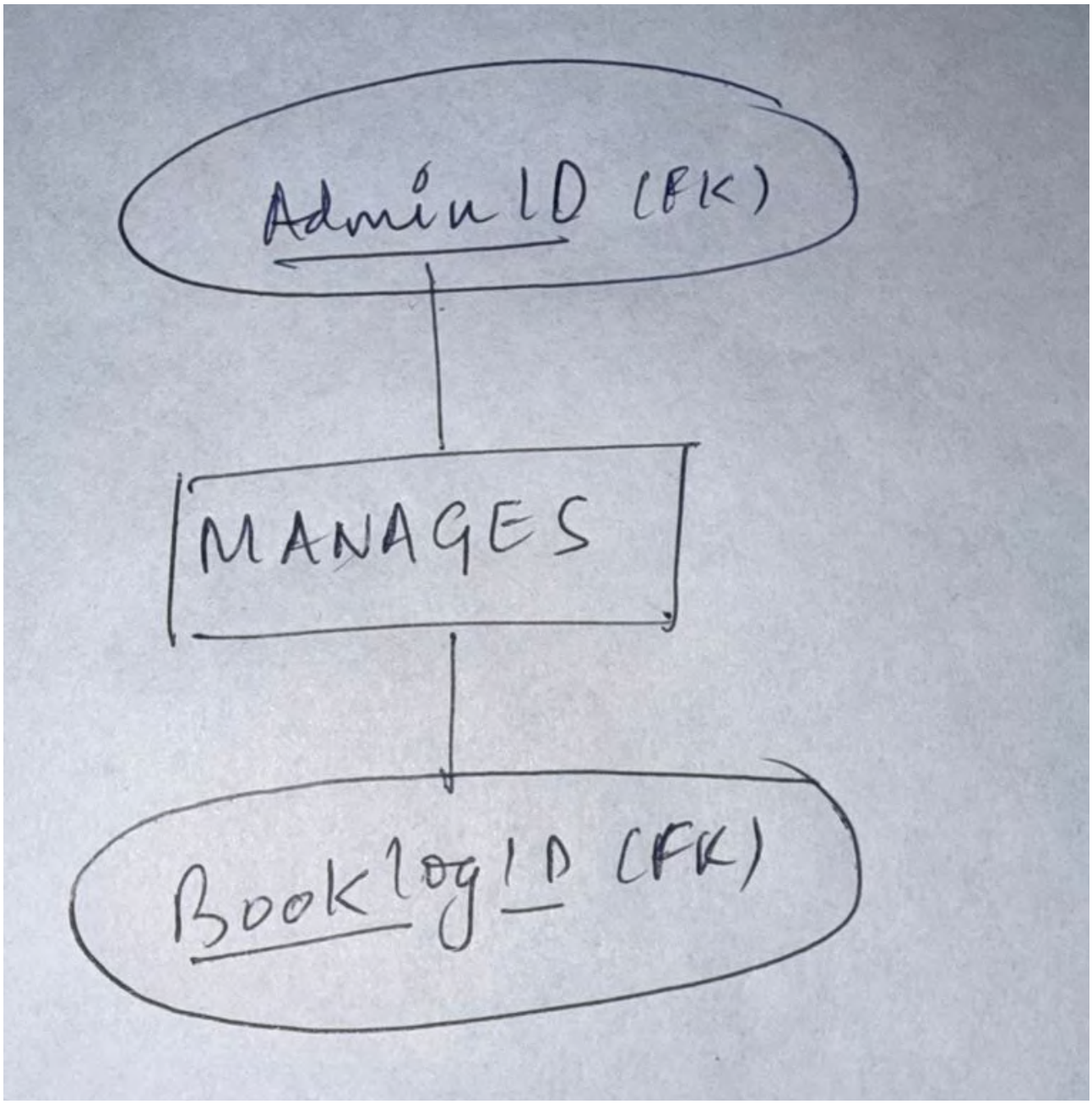
7. ADMIN



8. ADMIN_EMAIL



9. MANAGES (Junction Table)



ER ANALYSIS

Identifying Entity Sets and Relationship Sets:

Entity Sets:

1. ***ADMIN:***

   AdminID (PK, INT, Auto-Increment)
   Email (VARCHAR, UNIQUE)
   PasswordHash (VARCHAR, 255)
   CreatedAt (DATETIME)

2. ***USER***

   UserID (PK, INT, Auto-Increment)
   FirstName (VARCHAR, 50)
   LastName (VARCHAR, 50)
   Email (VARCHAR, UNIQUE)
   PhoneNumber (VARCHAR, 20)
   PasswordHash (VARCHAR, 255)
   CreatedAt (DATETIME)

3. ***BIKE***

   BikeID (PK, INT, Auto-Increment)
   Model (VARCHAR, 50)
   Brand (VARCHAR, 50)
   Status (ENUM: "Available", "Rented", “Maintenance")
   LocationID (FK to LOCATION)
   RatePerHour (DECIMAL, 5.2)
   CreatedAt (DATETIME)

4. ***LOCATION***

   LocationID (PK, INT, Auto-Increment)
   City (VARCHAR, 50)
   State (VARCHAR, 50)
   Street (VARCHAR, 100)
   Latitude (DECIMAL, 10, 8)
   Longitude (DECIMAL, 10, 8)

5. ***BOOKING***
   - ***BooklogID (PK, INT, Auto-Increment)***
   - ***UserID (FK to USER)***
   - ***BikeID (FK to BIKE)***
   - ***StartTime (DATETIME)***
   - ***EndTime (DATETIME)***
   - ***TotalCost (DECIMAL, 6.2)***
   - ***Status (VARCHAR, e.g., "Confirmed", “Cancelled")***
   - ***CreatedAt (DATETIME)***

6. ***PAYMENT***
   - ***PaymentID (PK, INT, Auto-Increment)***
   - ***BooklogID (FK to BOOKING)***
   - ***UserID (FK to USER)***
   - ***Amount (DECIMAL, 6.2)***
   - ***PaymentMethod (ENUM: "Cash", "Wallet", “Card")***
   - ***PaymentStatus (ENUM: "Pending", "Processing", "Failed", “Completed")***
   - ***CreatedAt (DATETIME)***

Relationship Sets

1. ***1. MANAGES***
   - ***Admin (0,1) ↔ Booking (1,N)***
   - ***An admin can manage multiple bookings, but a booking is managed by at most one admin (optional).***
2. ***2. BOOKS***
   - ***User (1,N) ↔ Booking (1,1)***
   - ***A user can create multiple bookings, but each booking is made by exactly one user.***
3. ***3 . ASSIGNED_TO (between BIKE and BOOKING)***
   - ***Bike (1,1) ↔ Booking (0,N)***
   - ***A bike can be booked multiple times (or not at all), but each booking is for exactly one bike.***
4. ***4. LOCATED_AT***
   - ***Bike (0,N) ↔ Location (1,1)***
   - ***A bike is assigned to exactly one location, and a location can have multiple bikes (or none).***
5. ***5. PAYS_FOR***
   - ***Booking (1,1) ↔ Payment (1,N)***
   - ***A booking can have multiple payments (e.g., installments), but each payment is tied to one booking.***
6. ***6. MAKES_PAYMENT***
   - ***User (1,1) ↔ Payment (0,N)***
   - ***A user can make multiple payments, but each payment is made by one user.***

Conversion of ER to Tables

1. USER

UserID → {FirstName, LastName, Email, PhoneNumber, PasswordHash, CreatedAt}

2. BIKE

BikeID → {Model, Brand, Status, LocationID, RatePerHour, CreatedAt}

3. LOCATION

LocationID → {City, State, Street, Latitude, Longitude}

4. BOOKING

BooklogID → {UserID, BikeID, StartTime, EndTime, TotalCost, Status, CreatedAt}

5. PAYMENT

PaymentID → {BooklogID, UserID, Amount, PaymentMethod, PaymentStatus, CreatedAt}

6. ADMIN

AdminID → {Email, PasswordHash, CreatedAt}

7. MANAGES (Admin-Booking Relationship)

AdminID → BooklogID

NORMALISATION

Introduction

The database for the Bike Rental Management System is designed to handle users, bikes, bookings, payments, locations, and admin activities efficiently. Normalization ensures minimal redundancy, data integrity, and optimized performance. This report documents the normalization process from UNF → 1NF → 2NF → 3NF → BCNF.

First Normal Form (1NF)

Transformation Actions

1. ***Ensure atomic attributes (no repeating groups).***
2. ***Assign primary keys for uniqueness.***

Normalized Tables

1. ***USER (UserID, FirstName, LastName, Email, PhoneNumber, PasswordHash, CreatedAt)***

2. ***BIKE (BikeID, Model, Brand, Status, LocationID, RatePerHour, CreatedAt)***

3. ***LOCATION (LocationID, City, State, Street, Latitude, Longitude)***

4. ***BOOKING (BooklogID, UserID, BikeID, StartTime, EndTime, TotalCost, Status, CreatedAt)***

5. ***PAYMENT (PaymentID, BooklogID, UserID, Amount, PaymentMethod, PaymentStatus, CreatedAt)***

6. ***ADMIN (AdminID, Email, PasswordHash, CreatedAt)***

7. ***MANAGES (AdminID, BooklogID) (Composite key for Admin-Booking relationship)***

Second Normal Form (2NF)

Transformation Actions

1. ***Remove partial dependencies (non-key attributes must depend on the entire PK).***
2. ***Split composite-key dependencies.***

Changes

1. ***BOOKING already satisfies 2NF (all non-key attributes depend on BooklogID).***

2. ***PAYMENT has a dependency on BooklogID (not just PaymentID), but since PaymentID is the PK, no changes needed.***

Third Normal Form (3NF)

Transformation Actions

1. ***Remove transitive dependencies (non-key attributes must depend only on the PK).***

Changes

1. ***USER has Email which is unique and could be a candidate key → Extract to a separate table.***
2. ***ADMIN has Email which is unique → Extract to a separate table.***

Updated Tables

1. *USER (UserID, FirstName, LastName, PhoneNumber, PasswordHash, CreatedAt)*
2. *USER_EMAIL (Email, UserID) (Ensures uniqueness and removes transitive dependency)*
3. *ADMIN (AdminID, PasswordHash, CreatedAt)*
4. *ADMIN_EMAIL (Email, AdminID) (Ensures uniqueness and removes transitive dependency)*

Boyce-Codd Normal Form (BCNF)

Transformation Actions

1. ***Ensure every determinant is a candidate key.***

Changes

1. ***BOOKING depends on UserID and BikeID (both are candidate keys).***
2. ***PAYMENT depends on BooklogID (candidate key).***

Final Tables (BCNF-Compliant)

1. *USER (UserID, FirstName, LastName, PhoneNumber, PasswordHash, CreatedAt)*
2. *USER_EMAIL (Email, UserID)*
3. *BIKE (BikeID, Model, Brand, Status, LocationID, RatePerHour, CreatedAt)*
4. *LOCATION (LocationID, City, State, Street, Latitude, Longitude)*
5. *BOOKING (BooklogID, UserID, BikeID, StartTime, EndTime, TotalCost, Status, CreatedAt)*
6. *PAYMENT (PaymentID, BooklogID, UserID, Amount, PaymentMethod, PaymentStatus, CreatedAt)*
7. *ADMIN (AdminID, PasswordHash, CreatedAt)*
8. *ADMIN_EMAIL (Email, AdminID)*
9. *MANAGES (AdminID, BooklogID)*

Functional Dependencies After Normalisation

1. USER
- *UserID → {FirstName, LastName, PhoneNumber, PasswordHash, CreatedAt}*

2. USER_EMAIL
- *Email → UserID*
- *UserID → Email (1:1 relationship, since each user has one email and vice versa)*

3. BIKE
- *BikeID → {Model, Brand, Status, LocationID, RatePerHour, CreatedAt}*
- *LocationID → {City, State, Street, Latitude, Longitude} (via LOCATION table)*

4. LOCATION
- *LocationID → {City, State, Street, Latitude, Longitude}*

5. BOOKING
- *BooklogID → {UserID, BikeID, StartTime, EndTime, TotalCost, Status, CreatedAt}*
- *{UserID, BikeID, StartTime} → BooklogID (Candidate key for uniqueness)*

6. PAYMENT
- *PaymentID → {BooklogID, UserID, Amount, PaymentMethod, PaymentStatus, CreatedAt}*
- *BooklogID → {UserID, BikeID, ...} (Partial dependency resolved via foreign key)*

7. ADMIN
- *AdminID → {PasswordHash, CreatedAt}*

8. ADMIN_EMAIL
- *Email → AdminID*
- *AdminID → Email (1:1 relationship)*

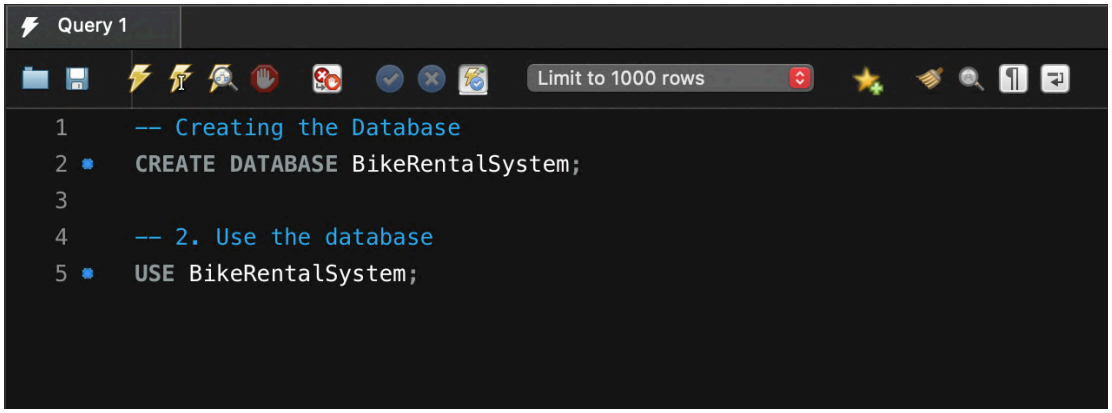
9. MANAGES (Admin-Booking Relationship)
- *{AdminID, BooklogID} → {} (Pure junction table with no additional attributes)*

SQL & PL/SQL

(Stored Procedure and Functions, Cursors, Triggers) Snapshots with output.

1. Create DBMS Database

CREATE DATABASE BikeRentalSystem;
USE BikeRentalSystem;



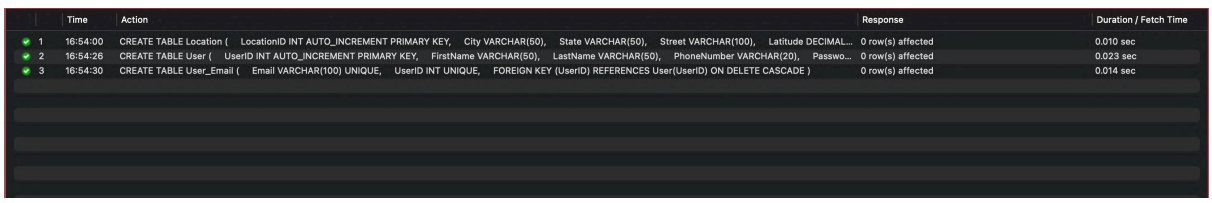
2. Create Location Table

```
CREATE TABLE Location (
        LocationID INT AUTO_INCREMENT PRIMARY KEY,
        City VARCHAR(50),
        State VARCHAR(50),
        Street VARCHAR(100),
        Latitude DECIMAL(10, 8),
        Longitude DECIMAL(10, 8)
);
```

3. Creating User and User_Email Table

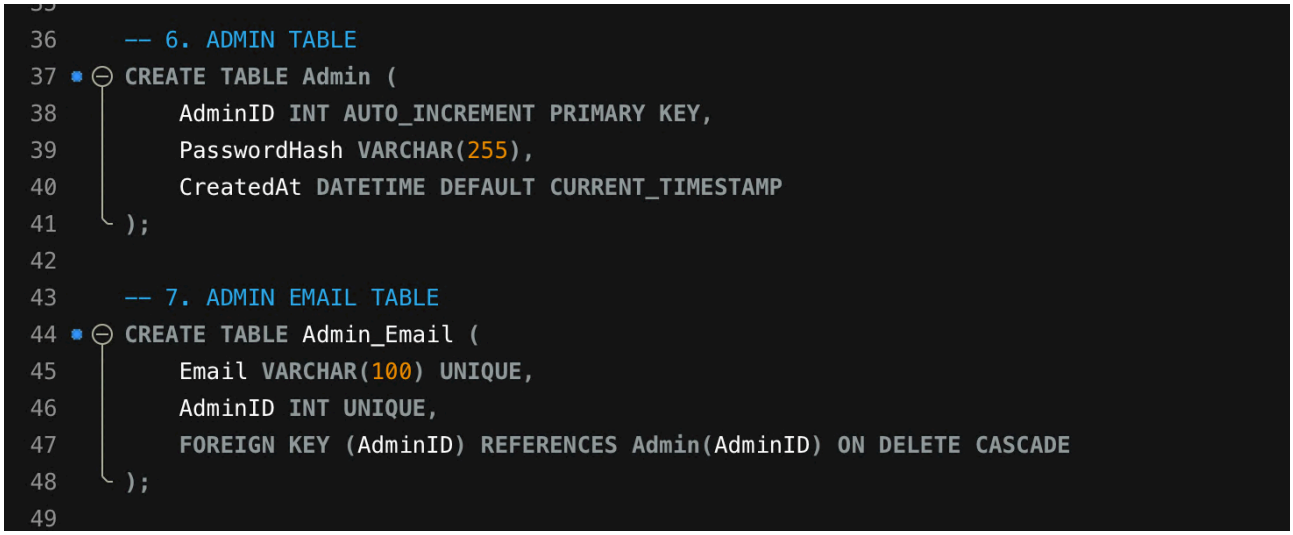
```
CREATE TABLE User (
        UserID INT AUTO_INCREMENT PRIMARY KEY,
        FirstName VARCHAR(50),
        LastName VARCHAR(50),
        PhoneNumber VARCHAR(20),
        PasswordHash VARCHAR(255),
        CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE User_Email (
        Email VARCHAR(100) UNIQUE,
        UserID INT UNIQUE,
        FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE
        CASCADE
);
```

```sql
-- 4. USER TABLE
CREATE TABLE User (
    UserID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    PhoneNumber VARCHAR(20),
    PasswordHash VARCHAR(255),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- 5. USER EMAIL TABLE
CREATE TABLE User_Email (
    Email VARCHAR(100) UNIQUE,
    UserID INT UNIQUE,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);
```



4. Creating Admin and Admin_Email Tables

```
CREATE TABLE Admin (
    AdminID INT AUTO_INCREMENT PRIMARY KEY,
    PasswordHash VARCHAR(255),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE Admin_Email (
    Email VARCHAR(100) UNIQUE,
    AdminID INT UNIQUE,
    FOREIGN KEY (AdminID) REFERENCES Admin(AdminID) ON DELETE
CASCADE
);
```



5. Creating the Bike Table

CREATE TABLE Bike (
BikeID INT AUTO_INCREMENT PRIMARY KEY,
Model VARCHAR(50),
Brand VARCHAR(50),
Status ENUM('Available', 'Rented', 'Maintenance'),
LocationID INT,
RatePerHour DECIMAL(5,2),
CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

```
50      -- 8. BIKE TABLE
51      CREATE TABLE Bike (
52          BikeID INT AUTO_INCREMENT PRIMARY KEY,
53          Model VARCHAR(50),
54          Brand VARCHAR(50),
55          Status ENUM('Available', 'Rented', 'Maintenance'),
56          LocationID INT,
57          RatePerHour DECIMAL(5,2),
58          CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
59          FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
60      );
```

Action Output

	Time	Action	Response	Duration / Fetch Time
1	16:54:00	CREATE TABLE Location (LocationID INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL(10, 8),...	0 row(s) affected	0.010 sec
2	16:54:26	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), PasswordHash...	0 row(s) affected	0.023 sec
3	16:54:30	CREATE TABLE User_Email (Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE)	0 row(s) affected	0.014 sec
4	16:55:20	CREATE TABLE Admin (AdminID INT AUTO_INCREMENT PRIMARY KEY, PasswordHash VARCHAR(255), CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP)	0 row(s) affected	0.014 sec
5	16:55:25	CREATE TABLE Admin_Email (Email VARCHAR(100) UNIQUE, AdminID INT UNIQUE, FOREIGN KEY (AdminID) REFERENCES Admin(AdminID) ON DELETE CASCADE)	0 row(s) affected	0.014 sec
6	16:56:37	CREATE TABLE Bike (BikeID INT AUTO_INCREMENT PRIMARY KEY, Model VARCHAR(50), Brand VARCHAR(50), Status ENUM('Available', 'Rented', 'Maintenance'), Loca...	0 row(s) affected	0.029 sec

6. Creating the Booking , Manages and Payments Table

```
CREATE TABLE Booking (
    BooklogID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT,
    BikeID INT,
    StartTime DATETIME,
    EndTime DATETIME,
    TotalCost DECIMAL(6,2),
    Status VARCHAR(20),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (BikeID) REFERENCES Bike(BikeID)
);


CREATE TABLE Manages (
    AdminID INT,
    BooklogID INT,
    PRIMARY KEY (AdminID, BooklogID),
    FOREIGN KEY (AdminID) REFERENCES Admin(AdminID),
    FOREIGN KEY (BooklogID) REFERENCES Booking(BooklogID)
);


CREATE TABLE Payment (
    PaymentID INT AUTO_INCREMENT PRIMARY KEY,
    BooklogID INT,
    UserID INT,
    Amount DECIMAL(6,2),
    PaymentMethod ENUM('Cash', 'Wallet', 'Card'),
    PaymentStatus ENUM('Pending', 'Processing', 'Failed', 'Completed'),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (BooklogID) REFERENCES Booking(BooklogID),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Code Snippet:

```sql
-- 9. BOOKING TABLE
CREATE TABLE Booking (
    BooklogID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT,
    BikeID INT,
    StartTime DATETIME,
    EndTime DATETIME,
    TotalCost DECIMAL(6,2),
    Status VARCHAR(20),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (BikeID) REFERENCES Bike(BikeID)
);

-- 10. MANAGES TABLE (Admin ↔ Booking)
CREATE TABLE Manages (
    AdminID INT,
    BooklogID INT,
    PRIMARY KEY (AdminID, BooklogID),
    FOREIGN KEY (AdminID) REFERENCES Admin(AdminID),
    FOREIGN KEY (BooklogID) REFERENCES Booking(BooklogID)
);

-- 11. PAYMENT TABLE
CREATE TABLE Payment (
    PaymentID INT AUTO_INCREMENT PRIMARY KEY,
    BooklogID INT,
    UserID INT,
    Amount DECIMAL(6,2),
    PaymentMethod ENUM('Cash', 'Wallet', 'Card'),
    PaymentStatus ENUM('Pending', 'Processing', 'Failed', 'Completed'),
    CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (BooklogID) REFERENCES Booking(BooklogID),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Action Output

	Time	Action	Response	Duration / Fetch Time
1	16:54:00	CREATE TABLE Location (LocationID INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL(10, 8),...	0 row(s) affected	0.010 sec
2	16:54:26	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), PasswordHash...	0 row(s) affected	0.023 sec
3	16:54:30	CREATE TABLE User_Email (Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE)	0 row(s) affected	0.014 sec
4	16:55:20	CREATE TABLE Admin (AdminID INT AUTO_INCREMENT PRIMARY KEY, PasswordHash VARCHAR(255), CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP)	0 row(s) affected	0.014 sec
5	16:55:25	CREATE TABLE Admin_Email (Email VARCHAR(100) UNIQUE, AdminID INT UNIQUE, FOREIGN KEY (AdminID) REFERENCES Admin(AdminID) ON DELETE CASCADE)	0 row(s) affected	0.014 sec
6	16:56:37	CREATE TABLE Bike (BikeID INT AUTO_INCREMENT PRIMARY KEY, Model VARCHAR(50), Brand VARCHAR(50), Status ENUM('Available', 'Rented', 'Maintenance'), Loca...	0 row(s) affected	0.029 sec
7	16:56:51	CREATE TABLE Booking (BooklogID INT AUTO_INCREMENT PRIMARY KEY, UserID INT, BikeID INT, StartTime DATETIME, EndTime DATETIME, TotalCost DECIMAL(6,...	0 row(s) affected	0.027 sec
8	16:57:05	CREATE TABLE Manages (AdminID INT, BooklogID INT, PRIMARY KEY (AdminID, BooklogID), FOREIGN KEY (AdminID) REFERENCES Admin(AdminID), FOREIGN KEY (Bo...	0 row(s) affected	0.019 sec
9	16:57:08	CREATE TABLE Payment (PaymentID INT AUTO_INCREMENT PRIMARY KEY, BooklogID INT, UserID INT, Amount DECIMAL(6,2), PaymentMethod ENUM('Cash', 'Wallet',...	0 row(s) affected	0.017 sec

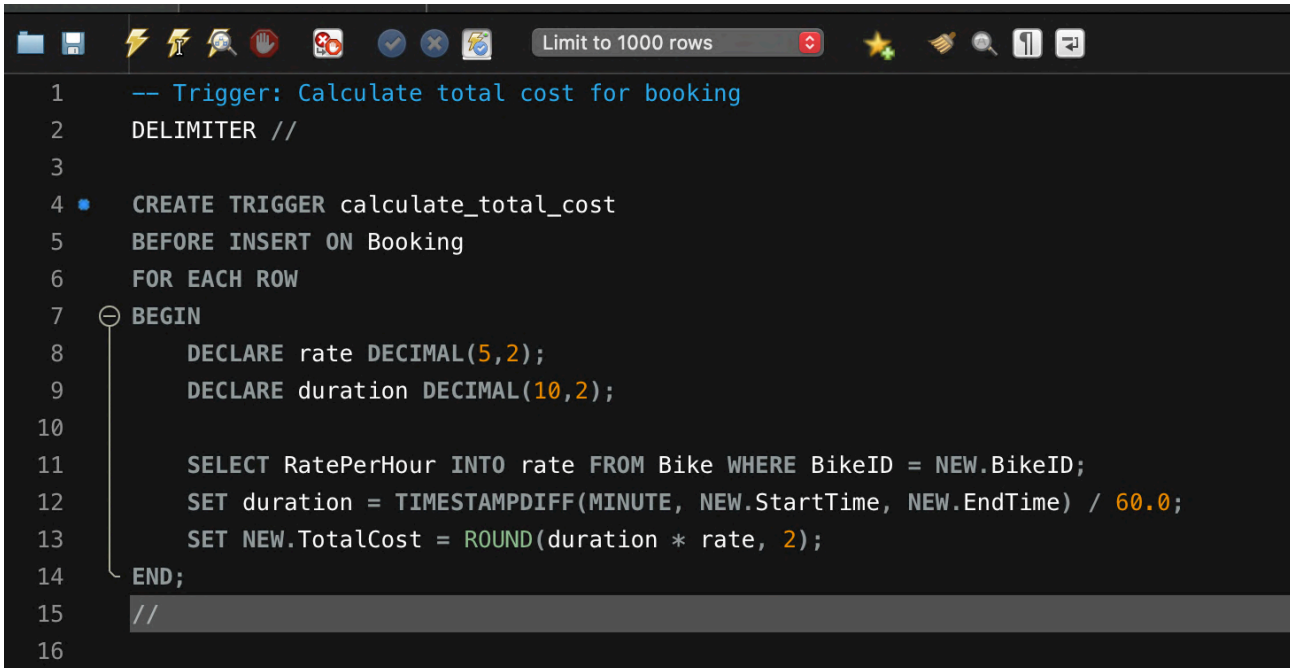
7. Creating a Trigger to Calculate the total cost of booking

DELIMITER //

CREATE TRIGGER calculate_total_cost
BEFORE INSERT ON Booking
FOR EACH ROW
BEGIN
DECLARE rate DECIMAL(5,2);
DECLARE duration DECIMAL(10,2);

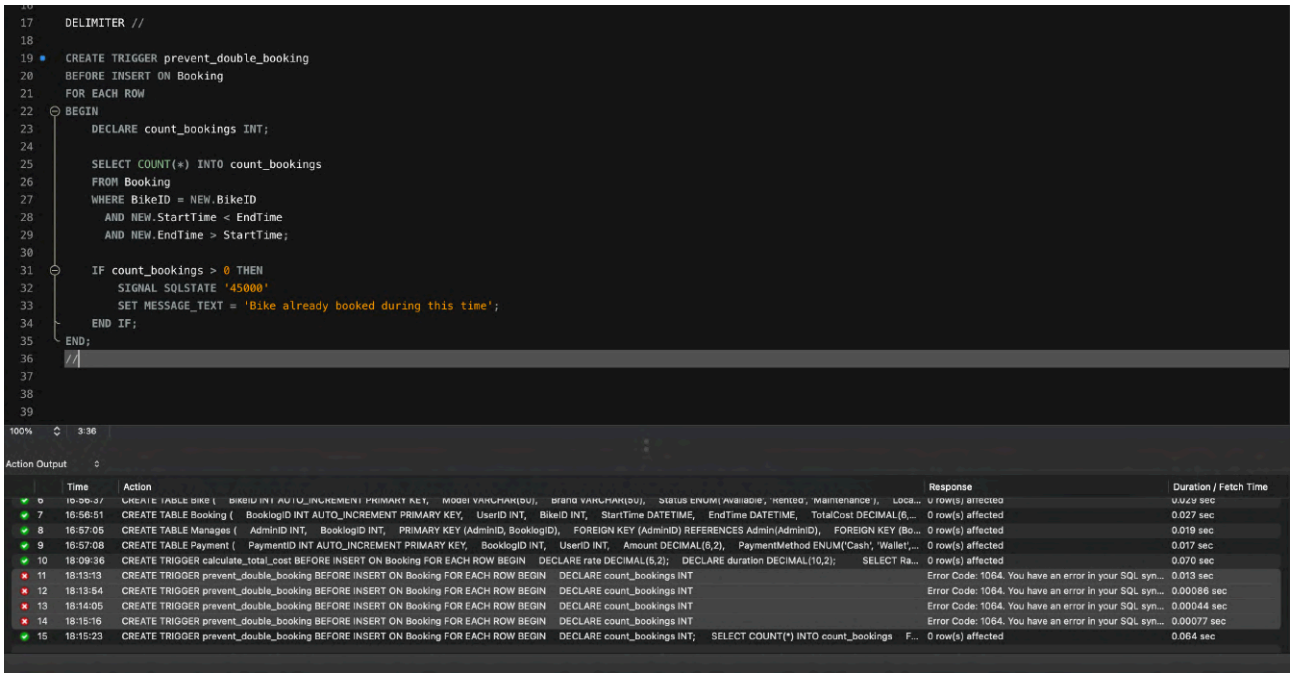
SELECT RatePerHour INTO rate FROM Bike WHERE BikeID = NEW.BikeID;
SET duration = TIMESTAMPDIFF(MINUTE, NEW.StartTime, NEW.EndTime) / 60.0;
*SET NEW.TotalCost = ROUND(duration * rate, 2);*
END;
//

This trigger calculate_total_cost automatically calculates the total cost of a bike booking before it's inserted into the Booking table.



8. Create a Trigger that prevents double-booking of a bike

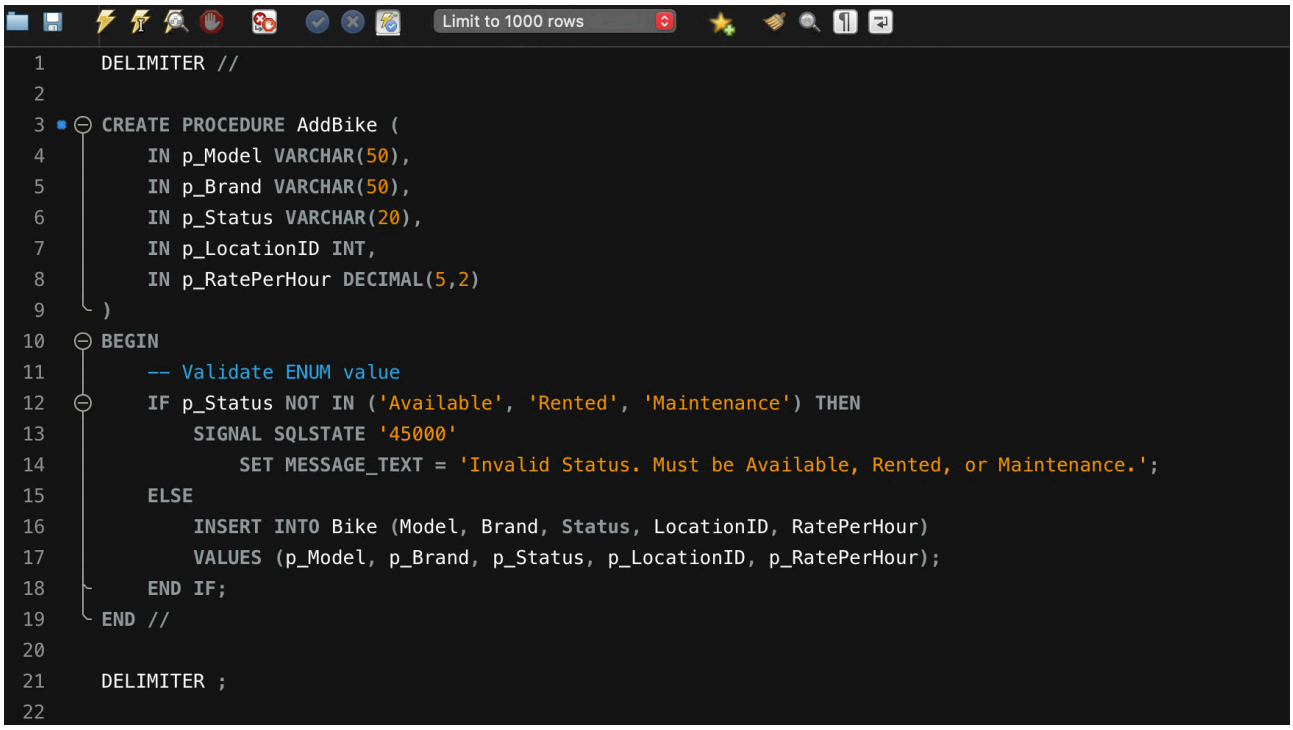
```
CREATE TRIGGER prevent_double_booking
BEFORE INSERT ON Booking
FOR EACH ROW
BEGIN
    DECLARE count_bookings INT;
    SELECT COUNT(*) INTO count_bookings
    FROM Booking
    WHERE BikeID = NEW.BikeID
      AND NEW.StartTime < EndTime
      AND NEW.EndTime > StartTime;

    IF count_bookings > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Bike already booked during this time';
    END IF;
END;
//
```



9. Creating a procedure to add a bike in the bike table

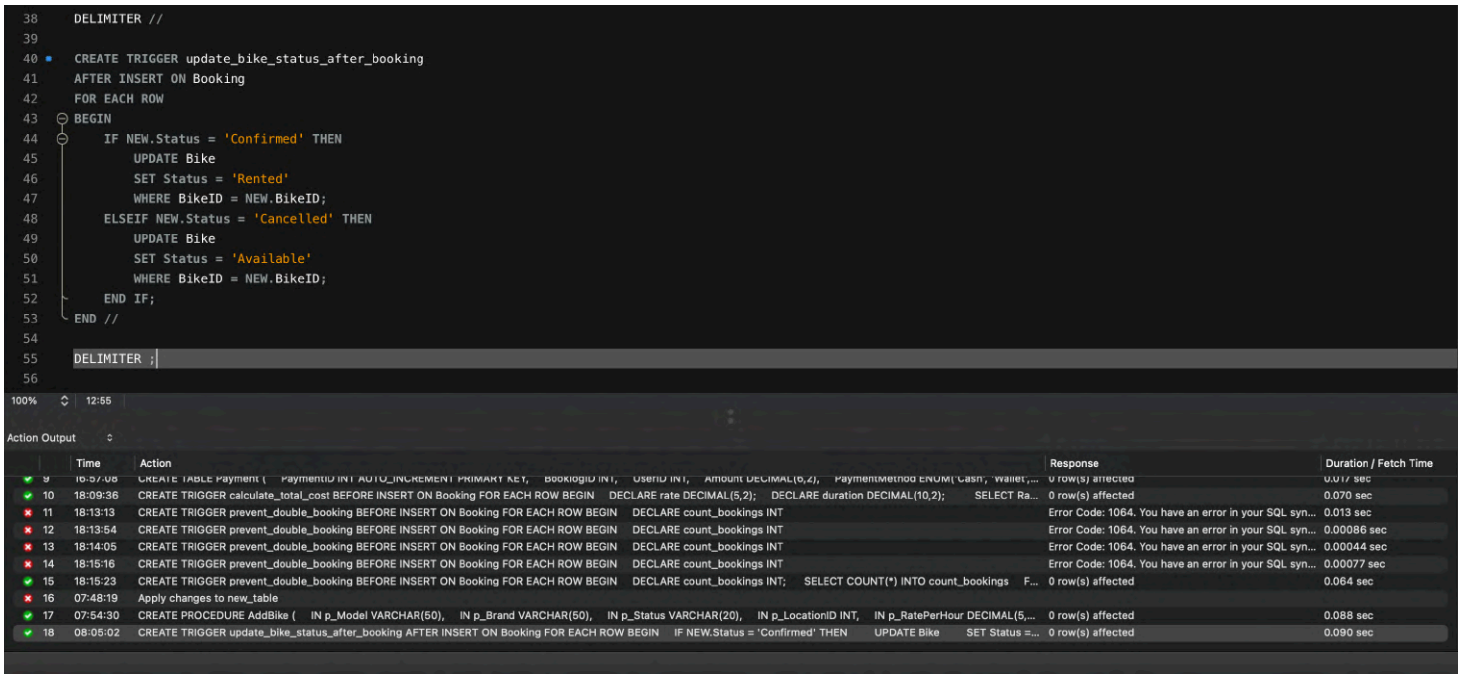
DELIMITER //

CREATE PROCEDURE AddBike (
IN p_Model VARCHAR(50),
IN p_Brand VARCHAR(50),
IN p_Status VARCHAR(20),
IN p_LocationID INT,
IN p_RatePerHour DECIMAL(5,2)
)
BEGIN
-- Validate ENUM value
IF p_Status NOT IN ('Available', 'Rented', 'Maintenance') THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Invalid Status. Must be Available, Rented, or Maintenance.';
ELSE
INSERT INTO Bike (Model, Brand, Status, LocationID, RatePerHour)
VALUES (p_Model, p_Brand, p_Status, p_LocationID, p_RatePerHour);
END IF;
END //



10. Creating a Trigger to Update Bike status after booking

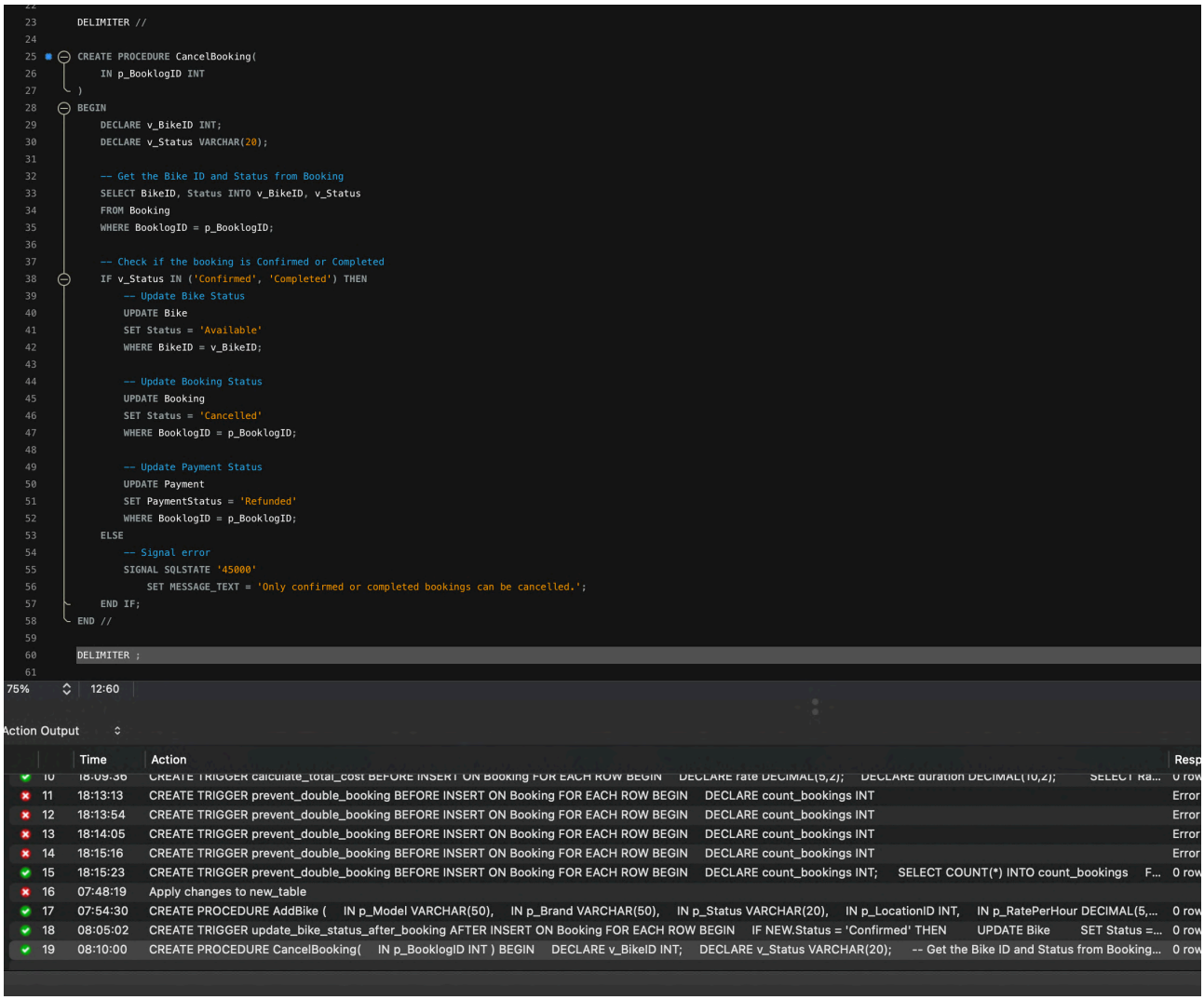
```
DELIMITER //

CREATE TRIGGER update_bike_status_after_booking
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN
    IF NEW.Status = 'Confirmed' THEN
        UPDATE Bike
        SET Status = 'Rented'
        WHERE BikeID = NEW.BikeID;
    ELSEIF NEW.Status = 'Cancelled' THEN
        UPDATE Bike
        SET Status = 'Available'
        WHERE BikeID = NEW.BikeID;
    END IF;
END //
```



11. Creating a Stored Procedure for Canceling Booking

```
DELIMITER //

CREATE PROCEDURE CancelBooking(
    IN p_BooklogID INT
)
BEGIN
    DECLARE v_BikeID INT;
    DECLARE v_Status VARCHAR(20);

    -- Get the Bike ID and Status from Booking
    SELECT BikeID, Status INTO v_BikeID, v_Status
    FROM Booking
    WHERE BooklogID = p_BooklogID;

    -- Check if the booking is Confirmed or Completed
    IF v_Status IN ('Confirmed', 'Completed') THEN
        -- Update Bike Status
        UPDATE Bike
        SET Status = 'Available'
        WHERE BikeID = v_BikeID;

        -- Update Booking Status
        UPDATE Booking
        SET Status = 'Cancelled'
        WHERE BooklogID = p_BooklogID;

        -- Update Payment Status
        UPDATE Payment
        SET PaymentStatus = 'Refunded'
        WHERE BooklogID = p_BooklogID;
    ELSE
        -- Signal error
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Only confirmed or completed bookings can be
cancelled.';
    END IF;
END //
```

This procedure allows users or admins to cancel a booking, making sure the status of the bike is updated, and it refunds the payment if needed.



```
DELIMITER //

CREATE PROCEDURE CancelBooking(
    IN p_BooklogID INT
)
BEGIN
    DECLARE v_BikeID INT;
    DECLARE v_Status VARCHAR(20);

    -- Get the Bike ID and Status from Booking
    SELECT BikeID, Status INTO v_BikeID, v_Status
    FROM Booking
    WHERE BooklogID = p_BooklogID;

    -- Check if the booking is Confirmed or Completed
    IF v_Status IN ('Confirmed', 'Completed') THEN
        -- Update Bike Status
        UPDATE Bike
        SET Status = 'Available'
        WHERE BikeID = v_BikeID;

        -- Update Booking Status
        UPDATE Booking
        SET Status = 'Cancelled'
        WHERE BooklogID = p_BooklogID;

        -- Update Payment Status
        UPDATE Payment
        SET PaymentStatus = 'Refunded'
        WHERE BooklogID = p_BooklogID;
    ELSE
        -- Signal error
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Only confirmed or completed bookings can be cancelled.';
    END IF;
END //

DELIMITER ;
```

75% 12:60

Action Output

	Time	Action	Resp
11	18:13:13	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error
12	18:13:54	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error
13	18:14:05	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error
14	18:15:16	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error
15	18:15:23	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings F...	0 row
16	07:48:19	Apply changes to new_table	
17	07:54:30	CREATE PROCEDURE AddBike (IN p_Model VARCHAR(50), IN p_Brand VARCHAR(50), IN p_Status VARCHAR(20), IN p_LocationID INT, IN p_RatePerHour DECIMAL(5,...	0 row
18	08:05:02	CREATE TRIGGER update_bike_status_after_booking AFTER INSERT ON Booking FOR EACH ROW BEGIN IF NEW.Status = 'Confirmed' THEN UPDATE Bike SET Status =...	0 row
19	08:10:00	CREATE PROCEDURE CancelBooking(IN p_BooklogID INT) BEGIN DECLARE v_BikeID INT; DECLARE v_Status VARCHAR(20); -- Get the Bike ID and Status from Booking...	0 row

12. Creating a Procedure to get available bikes on the basis of location

DELIMITER //

CREATE PROCEDURE GetAvailableBikesByLocation (
IN p_LocationID INT
)
BEGIN
SELECT BikeID, Model, Brand, RatePerHour
FROM Bike
WHERE LocationID = p_LocationID AND Status = 'Available';
END //

DELIMITER ;

```
62  DELIMITER //
63
64  CREATE PROCEDURE GetAvailableBikesByLocation (
65      IN p_LocationID INT
66  )
67  BEGIN
68      SELECT BikeID, Model, Brand, RatePerHour
69      FROM Bike
70      WHERE LocationID = p_LocationID AND Status = 'Available';
71  END //
72
73  DELIMITER ;
74
75
76
77
78
79
```

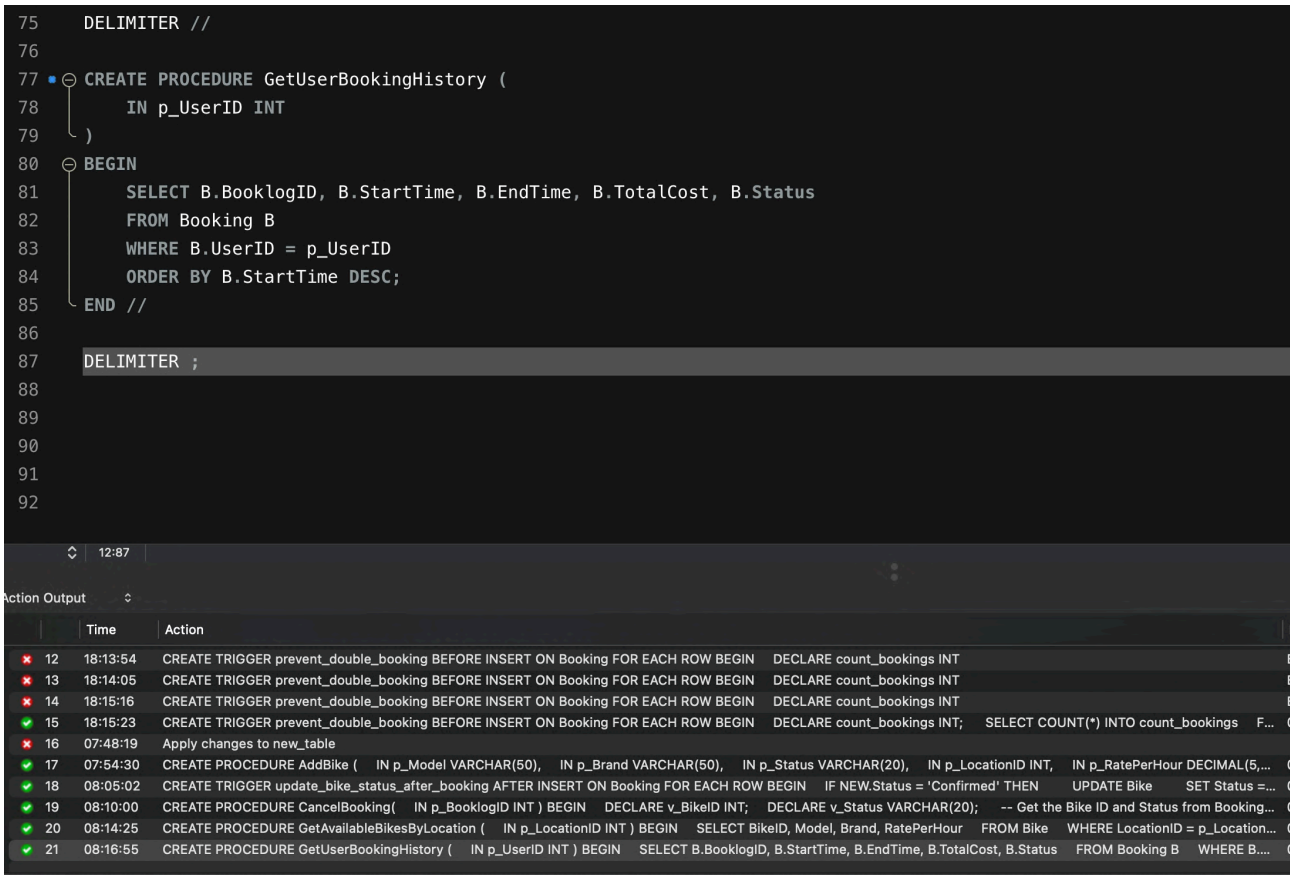
1:75

ction Output

	Time	Action	Respo
1	16:54:00	CREATE TABLE Location (LocationID INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL(10, 8),...	0 row(
2	16:54:26	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), PasswordHash...	0 row(
3	16:54:30	CREATE TABLE User_Email (Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE)	0 row(
4	16:55:20	CREATE TABLE Admin (AdminID INT AUTO_INCREMENT PRIMARY KEY, PasswordHash VARCHAR(255), CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP)	0 row(
5	16:55:25	CREATE TABLE Admin_Email (Email VARCHAR(100) UNIQUE, AdminID INT UNIQUE, FOREIGN KEY (AdminID) REFERENCES Admin(AdminID) ON DELETE CASCADE)	0 row(
6	16:56:37	CREATE TABLE Bike (BikeID INT AUTO_INCREMENT PRIMARY KEY, Model VARCHAR(50), Brand VARCHAR(50), Status ENUM('Available', 'Rented', 'Maintenance'), Loca...	0 row(
7	16:56:51	CREATE TABLE Booking (BooklogID INT AUTO_INCREMENT PRIMARY KEY, UserID INT, BikeID INT, StartTime DATETIME, EndTime DATETIME, TotalCost DECIMAL(6,...	0 row(
8	16:57:05	CREATE TABLE Manages (AdminID INT, BooklogID INT, PRIMARY KEY (AdminID, BooklogID), FOREIGN KEY (AdminID) REFERENCES Admin(AdminID), FOREIGN KEY (Bo...	0 row(
9	16:57:08	CREATE TABLE Payment (PaymentID INT AUTO_INCREMENT PRIMARY KEY, BooklogID INT, UserID INT, Amount DECIMAL(6,2), PaymentMethod ENUM('Cash', 'Wallet',...	0 row(
10	18:09:36	CREATE TRIGGER calculate_total_cost BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE rate DECIMAL(5,2); DECLARE duration DECIMAL(10,2); SELECT Ra...	0 row(
11	18:13:13	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error C
12	18:13:54	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error C
13	18:14:05	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error C
14	18:15:16	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error C
15	18:15:23	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings F...	0 row(
16	07:48:19	Apply changes to new_table	
17	07:54:30	CREATE PROCEDURE AddBike (IN p_Model VARCHAR(50), IN p_Brand VARCHAR(50), IN p_Status VARCHAR(20), IN p_LocationID INT, IN p_RatePerHour DECIMAL(5,...	0 row(
18	08:05:02	CREATE TRIGGER update_bike_status_after_booking AFTER INSERT ON Booking FOR EACH ROW BEGIN IF NEW.Status = 'Confirmed' THEN UPDATE Bike SET Status =...	0 row(
19	08:10:00	CREATE PROCEDURE CancelBooking(IN p_BooklogID INT) BEGIN DECLARE v_BikeID INT; DECLARE v_Status VARCHAR(20); -- Get the Bike ID and Status from Booking...	0 row(
20	08:14:25	CREATE PROCEDURE GetAvailableBikesByLocation (IN p_LocationID INT) BEGIN SELECT BikeID, Model, Brand, RatePerHour FROM Bike WHERE LocationID = p_Location...	0 row(

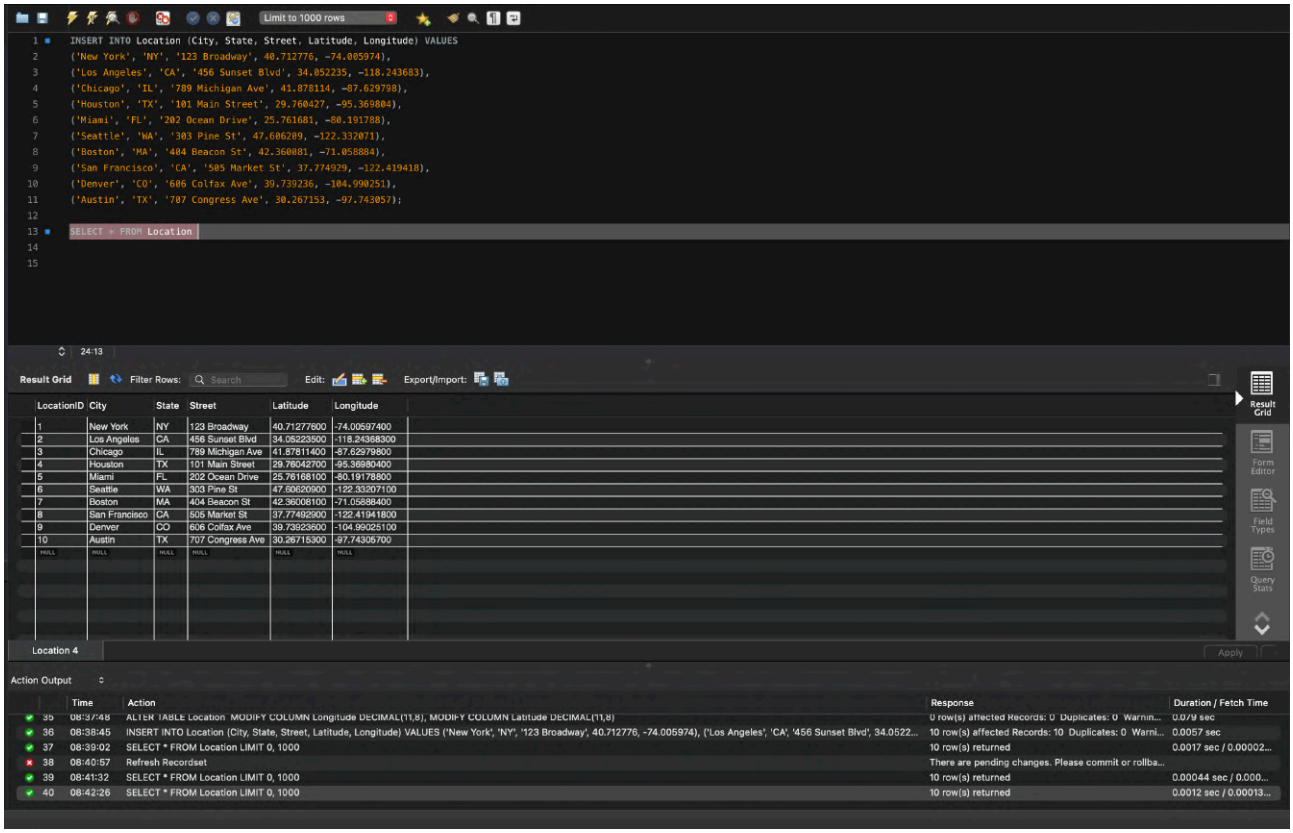
13. Create a Procedure to Get User Booking History

```
DELIMITER //

CREATE PROCEDURE GetUserBookingHistory (
    IN p_UserID INT
)
BEGIN
    SELECT B.BooklogID, B.StartTime, B.EndTime, B.TotalCost, B.Status
    FROM Booking B
    WHERE B.UserID = p_UserID
    ORDER BY B.StartTime DESC;
END //

DELIMITER ;
```

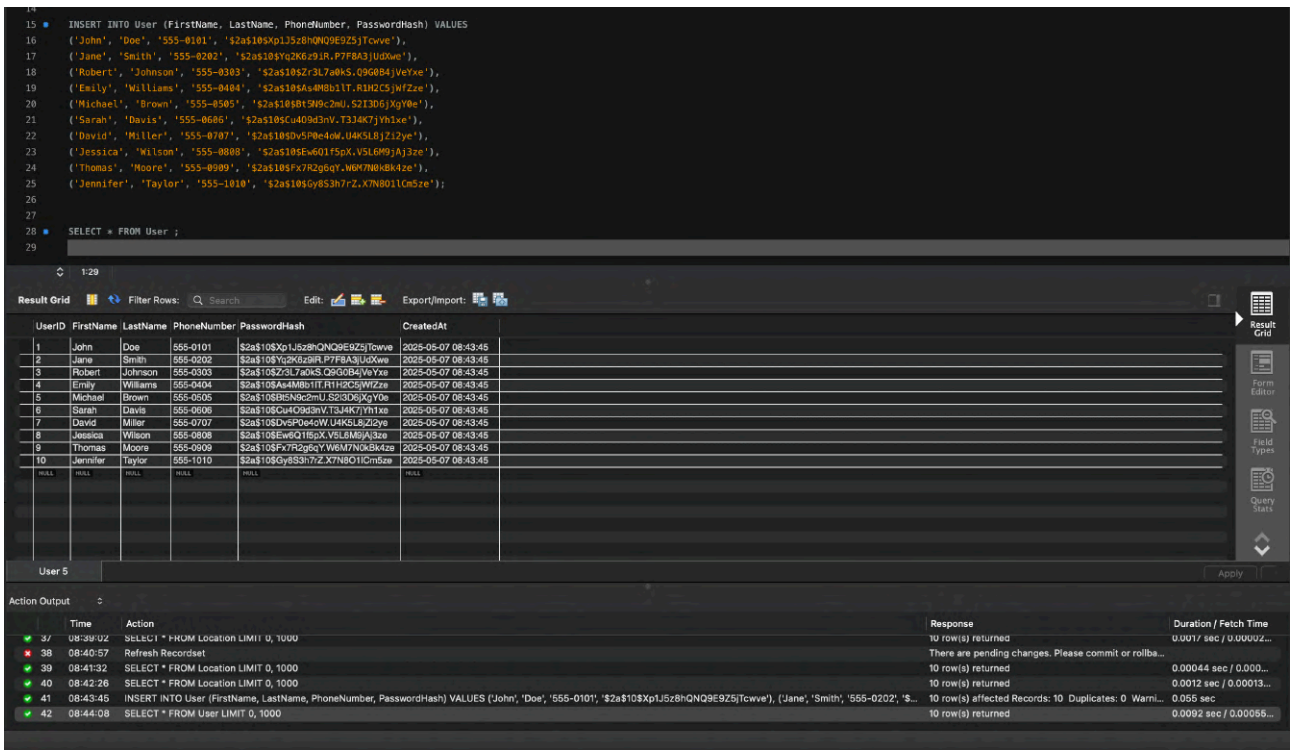


14. Entering information

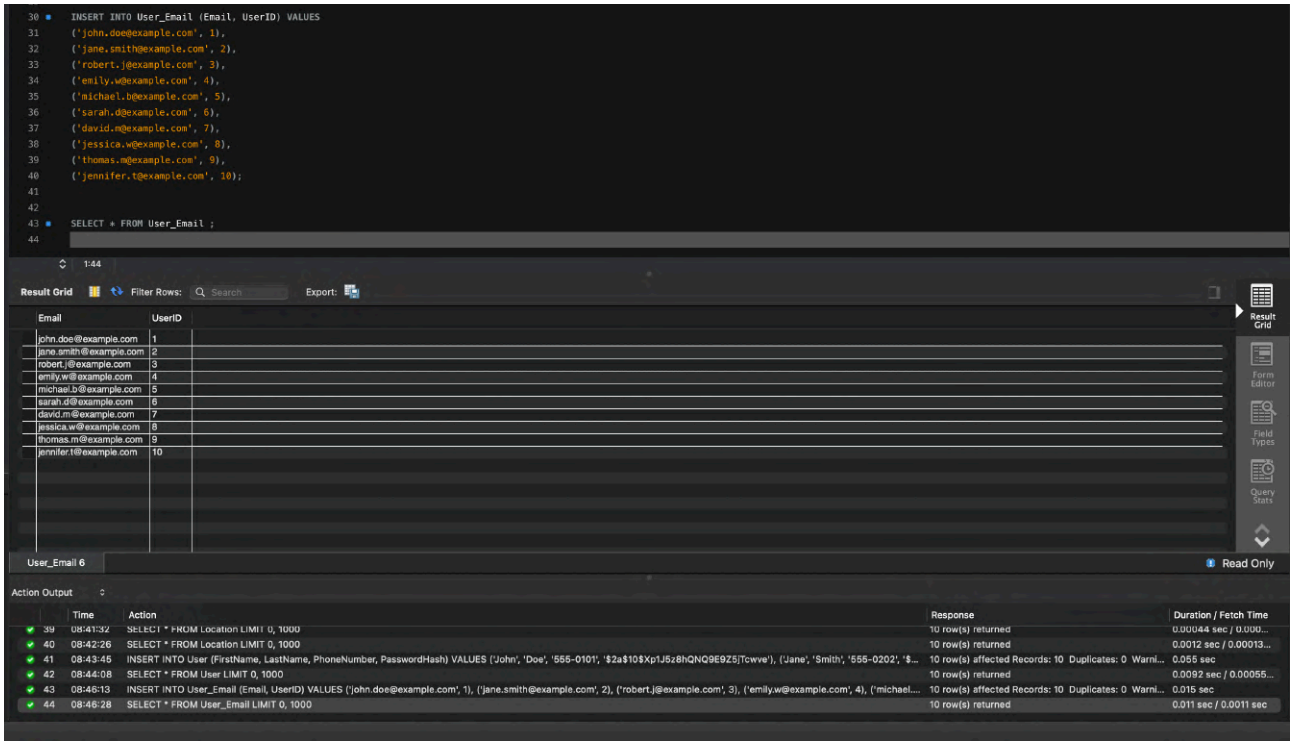
1. Location Table



2. User Table



3. User_Email Table



```
INSERT INTO User_Email (Email, UserID) VALUES
('john.doe@example.com', 1),
('jane.smith@example.com', 2),
('robert.j@example.com', 3),
('emily.w@example.com', 4),
('michael.b@example.com', 5),
('sarah.d@example.com', 6),
('david.m@example.com', 7),
('jessica.w@example.com', 8),
('thomas.m@example.com', 9),
('jennifer.t@example.com', 10);


SELECT * FROM User_Email ;
```

Result Grid — Filter Rows: Search — Export:

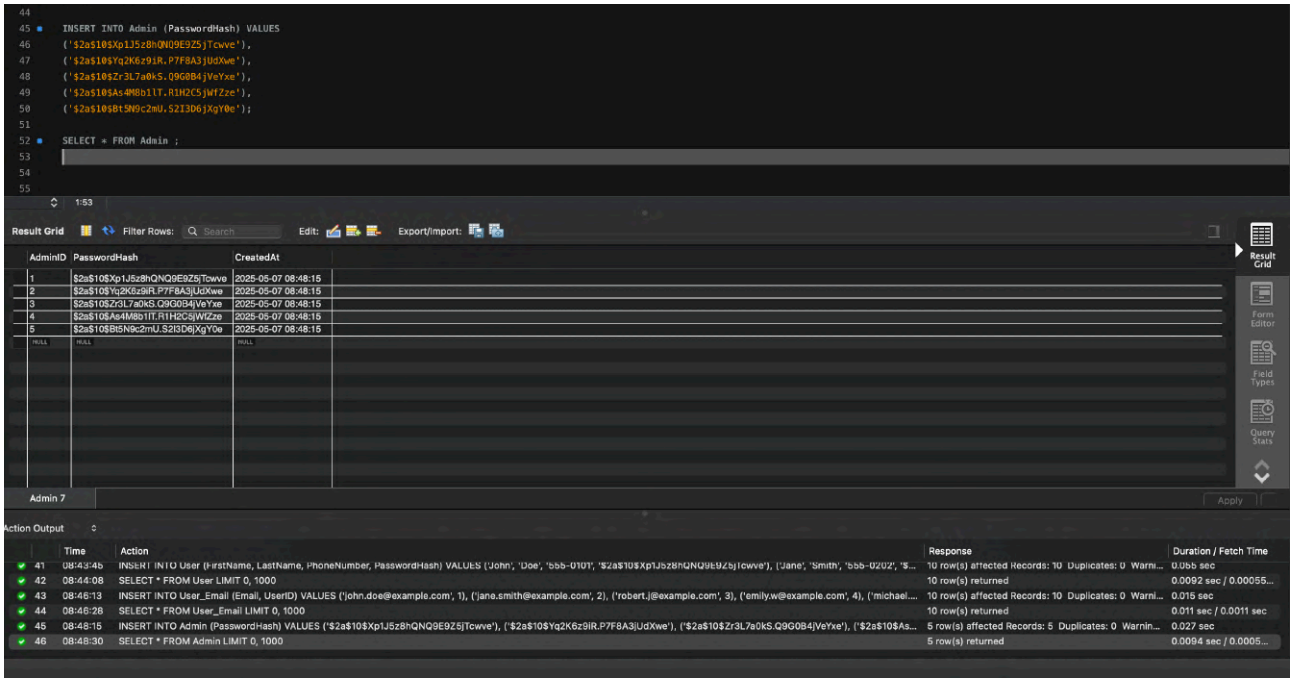
Email	UserID
john.doe@example.com	1
jane.smith@example.com	2
robert.j@example.com	3
emily.w@example.com	4
michael.b@example.com	5
sarah.d@example.com	6
david.m@example.com	7
jessica.w@example.com	8
thomas.m@example.com	9
jennifer.t@example.com	10

User_Email 6 — Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
39	08:41:32	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.00044 sec / 0.000...
40	08:42:26	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0012 sec / 0.00013...
41	08:43:45	INSERT INTO User (FirstName, LastName, PhoneNumber, PasswordHash) VALUES ('John', 'Doe', '555-0101', '$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve'), ('Jane', 'Smith', '555-0202', '$...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.055 sec
42	08:44:08	SELECT * FROM User LIMIT 0, 1000	10 row(s) returned	0.0092 sec / 0.00055...
43	08:46:13	INSERT INTO User_Email (Email, UserID) VALUES ('john.doe@example.com', 1), ('jane.smith@example.com', 2), ('robert.j@example.com', 3), ('emily.w@example.com', 4), ('michael....	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.015 sec
44	08:46:28	SELECT * FROM User_Email LIMIT 0, 1000	10 row(s) returned	0.011 sec / 0.0011 sec

4. Admin Table



```
INSERT INTO Admin (PasswordHash) VALUES
('$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve'),
('$2a$10$Yq2K6z9iR.P7F8A3jUdXwe'),
('$2a$10$Zr3L7a0kS.Q9G0B4jVeYxe'),
('$2a$10$As4M8b1lT.R1H2C5jWfZze'),
('$2a$10$BtSN9c2mU.S2I3D6jXgY0e');

SELECT * FROM Admin ;
```

Result Grid — Filter Rows: Search — Edit: — Export/Import:

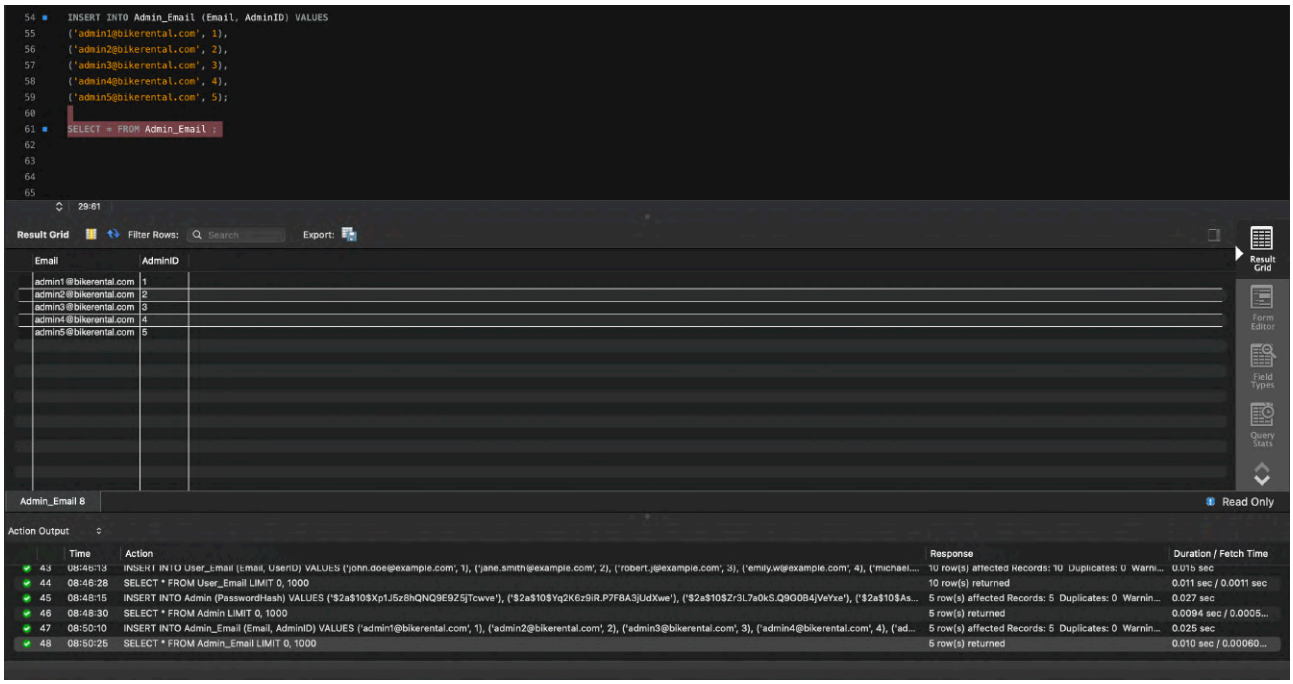
AdminID	PasswordHash	CreatedAt
1	$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve	2025-05-07 08:48:15
2	$2a$10$Yq2K6z9iR.P7F8A3jUdXwe	2025-05-07 08:48:15
3	$2a$10$Zr3L7a0kS.Q9G0B4jVeYxe	2025-05-07 08:48:15
4	$2a$10$As4M8b1lT.R1H2C5jWfZze	2025-05-07 08:48:15
5	$2a$10$BtSN9c2mU.S2I3D6jXgY0e	2025-05-07 08:48:15
NULL	NULL	NULL

Admin 7 — Apply

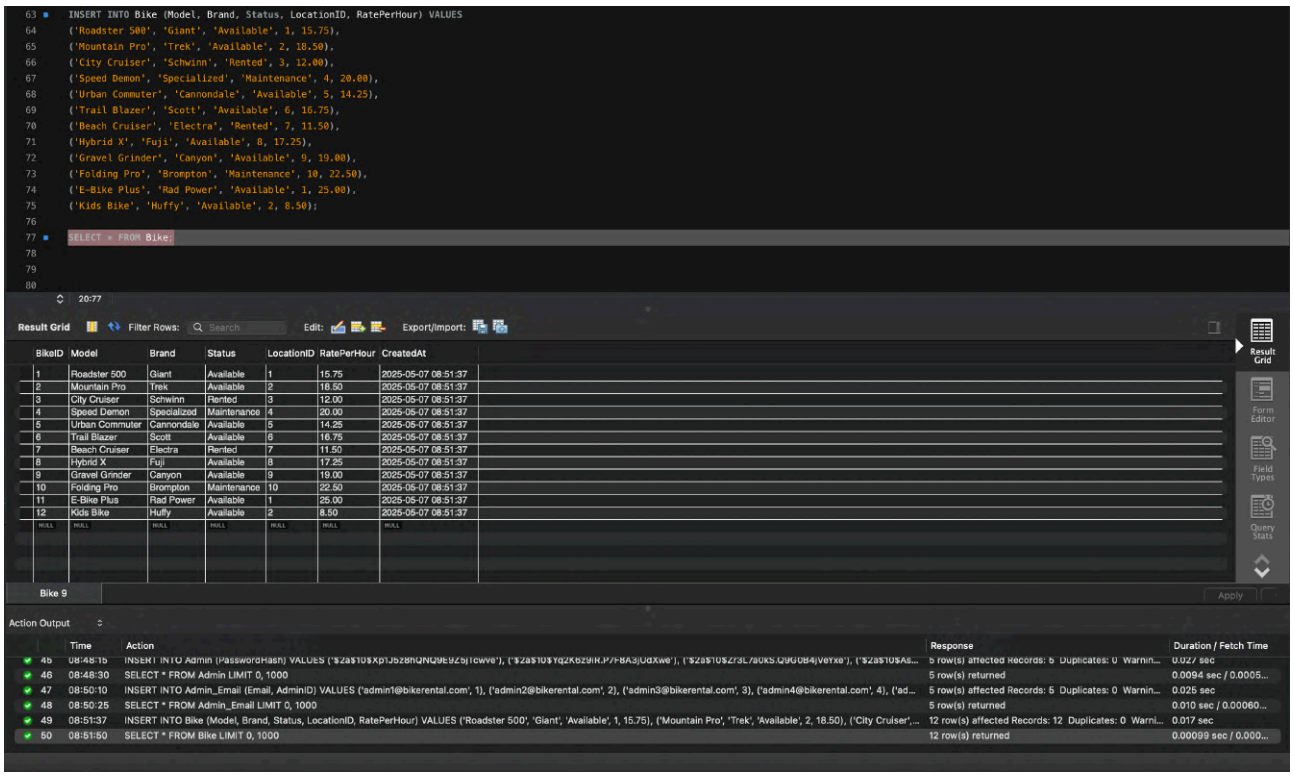
Action Output

	Time	Action	Response	Duration / Fetch Time
41	08:43:45	INSERT INTO User (FirstName, LastName, PhoneNumber, PasswordHash) VALUES ('John', 'Doe', '555-0101', '$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve'), ('Jane', 'Smith', '555-0202', '$...	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.055 sec
42	08:44:08	SELECT * FROM User LIMIT 0, 1000	10 row(s) returned	0.0092 sec / 0.00055...
43	08:46:13	INSERT INTO User_Email (Email, UserID) VALUES ('john.doe@example.com', 1), ('jane.smith@example.com', 2), ('robert.j@example.com', 3), ('emily.w@example.com', 4), ('michael....	10 row(s) affected Records: 10 Duplicates: 0 Warni...	0.015 sec
44	08:46:28	SELECT * FROM User_Email LIMIT 0, 1000	10 row(s) returned	0.011 sec / 0.0011 sec
45	08:48:15	INSERT INTO Admin (PasswordHash) VALUES ('$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve'), ('$2a$10$Yq2K6z9iR.P7F8A3jUdXwe'), ('$2a$10$Zr3L7a0kS.Q9G0B4jVeYxe'), ('$2a$10$As...	5 row(s) affected Records: 5 Duplicates: 0 Warnin...	0.027 sec
46	08:48:30	SELECT * FROM Admin LIMIT 0, 1000	5 row(s) returned	0.0094 sec / 0.0005...

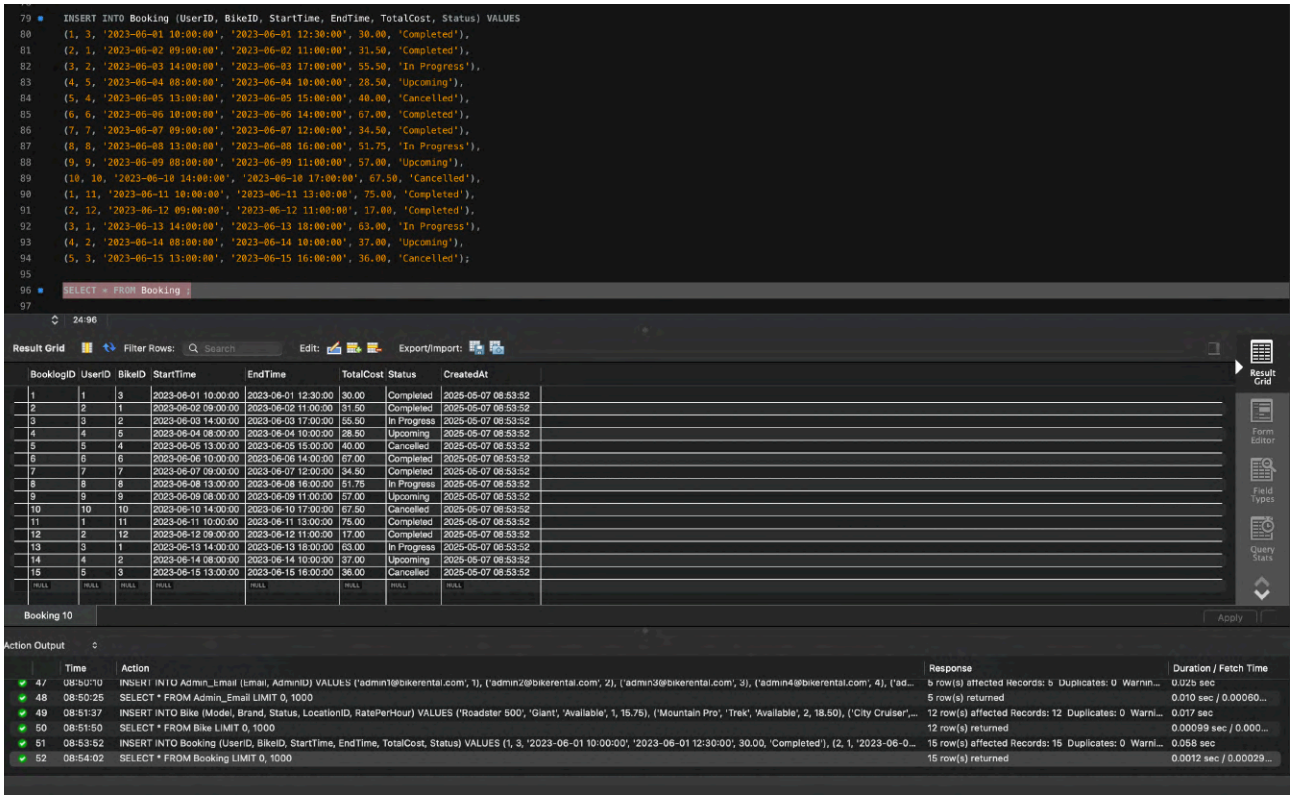
5. Admin_Email Table



6. Bike Table



7.Booking Table



```
INSERT INTO Booking (UserID, BikeID, StartTime, EndTime, TotalCost, Status) VALUES
(1, 3, '2023-06-01 10:00:00', '2023-06-01 12:30:00', 30.00, 'Completed'),
(2, 1, '2023-06-02 09:00:00', '2023-06-02 11:00:00', 31.50, 'Completed'),
(3, 2, '2023-06-03 14:00:00', '2023-06-03 17:00:00', 55.50, 'In Progress'),
(4, 5, '2023-06-04 08:00:00', '2023-06-04 10:00:00', 28.50, 'Upcoming'),
(5, 4, '2023-06-05 13:00:00', '2023-06-05 15:00:00', 40.00, 'Cancelled'),
(6, 6, '2023-06-06 10:00:00', '2023-06-06 14:00:00', 67.00, 'Completed'),
(7, 7, '2023-06-07 09:00:00', '2023-06-07 12:00:00', 34.50, 'Completed'),
(8, 8, '2023-06-08 13:00:00', '2023-06-08 16:00:00', 51.75, 'In Progress'),
(9, 9, '2023-06-09 08:00:00', '2023-06-09 11:00:00', 57.00, 'Upcoming'),
(10, 10, '2023-06-10 14:00:00', '2023-06-10 17:00:00', 67.50, 'Cancelled'),
(1, 11, '2023-06-11 10:00:00', '2023-06-11 13:00:00', 75.00, 'Completed'),
(2, 12, '2023-06-12 09:00:00', '2023-06-12 11:00:00', 17.00, 'Completed'),
(3, 1, '2023-06-13 14:00:00', '2023-06-13 18:00:00', 63.00, 'In Progress'),
(4, 2, '2023-06-14 08:00:00', '2023-06-14 10:00:00', 37.00, 'Upcoming'),
(5, 3, '2023-06-15 13:00:00', '2023-06-15 16:00:00', 36.00, 'Cancelled');

SELECT * FROM Booking ;
```

Result Grid

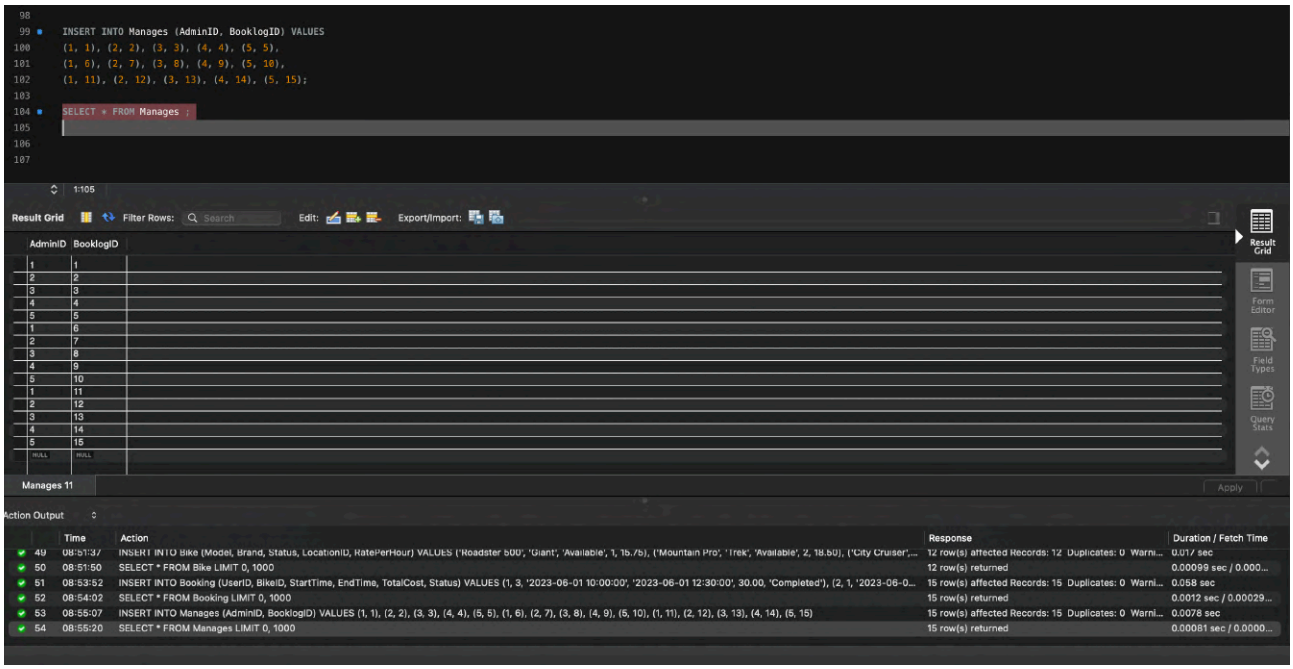
BookingID	UserID	BikeID	StartTime	EndTime	TotalCost	Status	CreatedAt
1	1	3	2023-06-01 10:00:00	2023-06-01 12:30:00	30.00	Completed	2025-05-07 08:53:52
2	2	1	2023-06-02 09:00:00	2023-06-02 11:00:00	31.50	Completed	2025-05-07 08:53:52
3	3	2	2023-06-03 14:00:00	2023-06-03 17:00:00	55.50	In Progress	2025-05-07 08:53:52
4	4	5	2023-06-04 08:00:00	2023-06-04 10:00:00	28.50	Upcoming	2025-05-07 08:53:52
5	5	4	2023-06-05 13:00:00	2023-06-05 15:00:00	40.00	Cancelled	2025-05-07 08:53:52
6	6	6	2023-06-06 10:00:00	2023-06-06 14:00:00	67.00	Completed	2025-05-07 08:53:52
7	7	7	2023-06-07 09:00:00	2023-06-07 12:00:00	34.50	Completed	2025-05-07 08:53:52
8	8	8	2023-06-08 13:00:00	2023-06-08 16:00:00	51.75	In Progress	2025-05-07 08:53:52
9	9	9	2023-06-09 08:00:00	2023-06-09 11:00:00	57.00	Upcoming	2025-05-07 08:53:52
10	10	10	2023-06-10 14:00:00	2023-06-10 17:00:00	67.50	Cancelled	2025-05-07 08:53:52
11	1	11	2023-06-11 10:00:00	2023-06-11 13:00:00	75.00	Completed	2025-05-07 08:53:52
12	2	12	2023-06-12 09:00:00	2023-06-12 11:00:00	17.00	Completed	2025-05-07 08:53:52
13	3	1	2023-06-13 14:00:00	2023-06-13 18:00:00	63.00	In Progress	2025-05-07 08:53:52
14	4	2	2023-06-14 08:00:00	2023-06-14 10:00:00	37.00	Upcoming	2025-05-07 08:53:52
15	5	3	2023-06-15 13:00:00	2023-06-15 16:00:00	36.00	Cancelled	2025-05-07 08:53:52

Booking 10

Action Output

	Time	Action	Response	Duration / Fetch Time
47	08:50:10	INSERT INTO Admin_Email (Email, AdminID) VALUES ('admin1@bikerental.com', 1), ('admin2@bikerental.com', 2), ('admin3@bikerental.com', 3), ('admin4@bikerental.com', 4), ('ad...	5 row(s) affected Records: 5 Duplicates: 0 Warnin...	0.026 sec
48	08:50:25	SELECT * FROM Admin_Email LIMIT 0, 1000	5 row(s) returned	0.010 sec / 0.00060...
49	08:51:37	INSERT INTO Bike (Model, Brand, Status, LocationID, RatePerHour) VALUES ('Roadster 500', 'Giant', 'Available', 1, 15.75), ('Mountain Pro', 'Trek', 'Available', 2, 18.50), ('City Cruiser',...	12 row(s) affected Records: 12 Duplicates: 0 Warni...	0.017 sec
50	08:51:50	SELECT * FROM Bike LIMIT 0, 1000	12 row(s) returned	0.00099 sec / 0.000...
51	08:53:52	INSERT INTO Booking (UserID, BikeID, StartTime, EndTime, TotalCost, Status) VALUES (1, 3, '2023-06-01 10:00:00', '2023-06-01 12:30:00', 30.00, 'Completed'), (2, 1, '2023-06-0...	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.058 sec
52	08:54:02	SELECT * FROM Booking LIMIT 0, 1000	15 row(s) returned	0.0012 sec / 0.00029...

8. Manages Table



```
INSERT INTO Manages (AdminID, BookingID) VALUES
(1, 1), (2, 2), (3, 3), (4, 4), (5, 5),
(1, 6), (2, 7), (3, 8), (4, 9), (5, 10),
(1, 11), (2, 12), (3, 13), (4, 14), (5, 15);

SELECT * FROM Manages ;
```

Result Grid

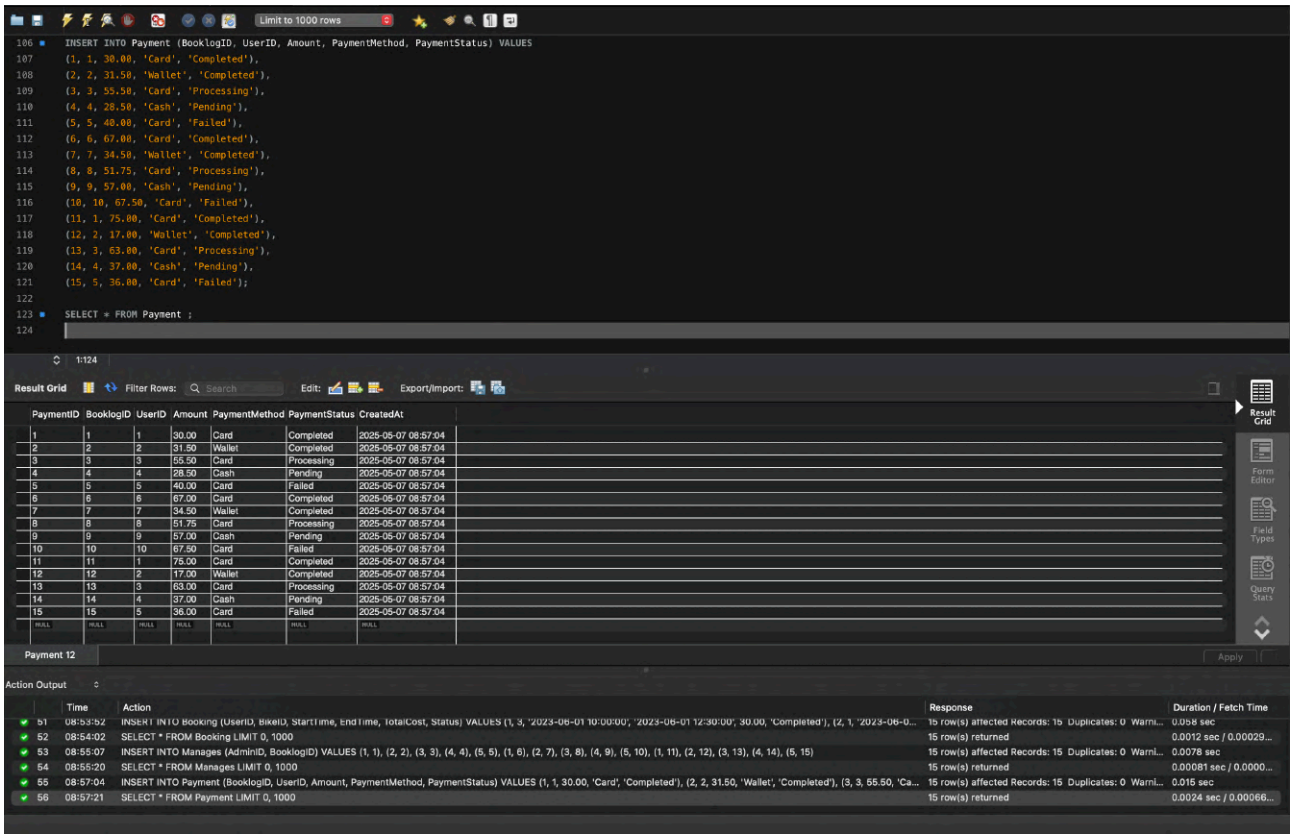
AdminID	BookingID
1	1
2	2
3	3
4	4
5	5
1	6
2	7
3	8
4	9
5	10
1	11
2	12
3	13
4	14
5	15

Manages 11

Action Output

	Time	Action	Response	Duration / Fetch Time
49	08:51:37	INSERT INTO Bike (Model, Brand, Status, LocationID, RatePerHour) VALUES ('Roadster 500', 'Giant', 'Available', 1, 15.75), ('Mountain Pro', 'Trek', 'Available', 2, 18.50), ('City Cruiser',...	12 row(s) affected Records: 12 Duplicates: 0 Warni...	0.017 sec
50	08:51:50	SELECT * FROM Bike LIMIT 0, 1000	12 row(s) returned	0.00099 sec / 0.000...
51	08:53:52	INSERT INTO Booking (UserID, BikeID, StartTime, EndTime, TotalCost, Status) VALUES (1, 3, '2023-06-01 10:00:00', '2023-06-01 12:30:00', 30.00, 'Completed'), (2, 1, '2023-06-0...	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.058 sec
52	08:54:02	SELECT * FROM Booking LIMIT 0, 1000	15 row(s) returned	0.0012 sec / 0.00029...
53	08:55:07	INSERT INTO Manages (AdminID, BookingID) VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 6), (2, 7), (3, 8), (4, 9), (5, 10), (1, 11), (2, 12), (3, 13), (4, 14), (5, 15)	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.0078 sec
54	08:55:20	SELECT * FROM Manages LIMIT 0, 1000	15 row(s) returned	0.00081 sec / 0.0000...

9.Payment Table



```
INSERT INTO Payment (BooklogID, UserID, Amount, PaymentMethod, PaymentStatus) VALUES
(1, 1, 30.00, 'Card', 'Completed'),
(2, 2, 31.50, 'Wallet', 'Completed'),
(3, 3, 55.50, 'Card', 'Processing'),
(4, 4, 28.50, 'Cash', 'Pending'),
(5, 5, 40.00, 'Card', 'Failed'),
(6, 6, 67.00, 'Card', 'Completed'),
(7, 7, 34.50, 'Wallet', 'Completed'),
(8, 8, 51.75, 'Card', 'Processing'),
(9, 9, 57.00, 'Cash', 'Pending'),
(10, 10, 67.50, 'Card', 'Failed'),
(11, 1, 75.00, 'Card', 'Completed'),
(12, 2, 17.00, 'Wallet', 'Completed'),
(13, 3, 63.00, 'Card', 'Processing'),
(14, 4, 37.00, 'Cash', 'Pending'),
(15, 5, 36.00, 'Card', 'Failed');

SELECT * FROM Payment ;
```

PaymentID	BooklogID	UserID	Amount	PaymentMethod	PaymentStatus	CreatedAt
1	1	1	30.00	Card	Completed	2025-05-07 08:57:04
2	2	2	31.50	Wallet	Completed	2025-05-07 08:57:04
3	3	3	55.50	Card	Processing	2025-05-07 08:57:04
4	4	4	28.50	Cash	Pending	2025-05-07 08:57:04
5	5	5	40.00	Card	Failed	2025-05-07 08:57:04
6	6	6	67.00	Card	Completed	2025-05-07 08:57:04
7	7	7	34.50	Wallet	Completed	2025-05-07 08:57:04
8	8	8	51.75	Card	Processing	2025-05-07 08:57:04
9	9	9	57.00	Cash	Pending	2025-05-07 08:57:04
10	10	10	67.50	Card	Failed	2025-05-07 08:57:04
11	11	1	75.00	Card	Completed	2025-05-07 08:57:04
12	12	2	17.00	Wallet	Completed	2025-05-07 08:57:04
13	13	3	63.00	Card	Processing	2025-05-07 08:57:04
14	14	4	37.00	Cash	Pending	2025-05-07 08:57:04
15	15	5	36.00	Card	Failed	2025-05-07 08:57:04

Time	Action	Response	Duration / Fetch Time
08:54:02	SELECT * FROM Booking LIMIT 0, 1000	15 row(s) returned	0.0012 sec / 0.00029...
08:55:07	INSERT INTO Manages (AdminID, BooklogID) VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 6), (2, 7), (3, 8), (4, 9), (5, 10), (1, 11), (2, 12), (3, 13), (4, 14), (5, 15)	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.0078 sec
08:55:20	SELECT * FROM Manages LIMIT 0, 1000	15 row(s) returned	0.00081 sec / 0.0000...
08:57:04	INSERT INTO Payment (BooklogID, UserID, Amount, PaymentMethod, PaymentStatus) VALUES (1, 1, 30.00, 'Card', 'Completed'), (2, 2, 31.50, 'Wallet', 'Completed'), (3, 3, 55.50, 'Ca...	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.015 sec
08:57:21	SELECT * FROM Payment LIMIT 0, 1000	15 row(s) returned	0.0024 sec / 0.00066...

Working of Stored Procedures in the database

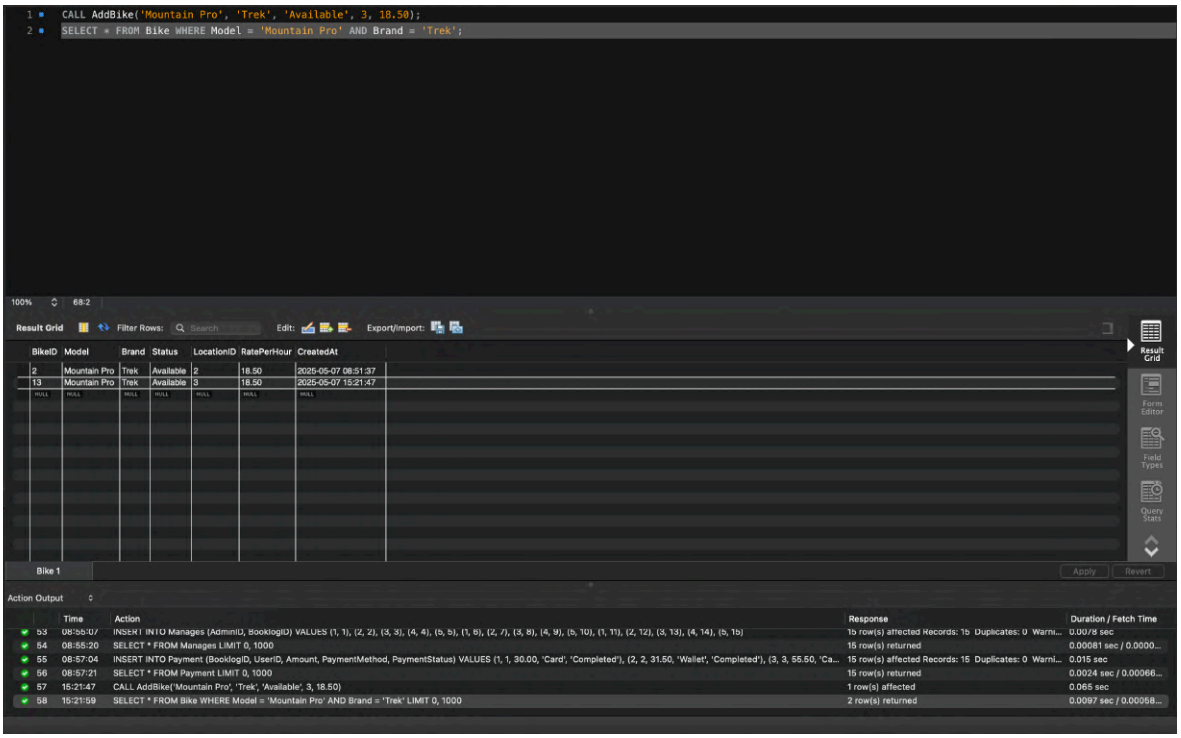
1. Add Bike Procedure

Command to call the Procedure :

CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50);

Verifying the procedure worked

*SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek';*



```
CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50);
SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek';
```

BikeID	Model	Brand	Status	LocationID	RatePerHour	CreatedAt
2	Mountain Pro	Trek	Available	2	18.50	2025-05-07 08:51:37
13	Mountain Pro	Trek	Available	3	18.50	2025-05-07 15:21:47

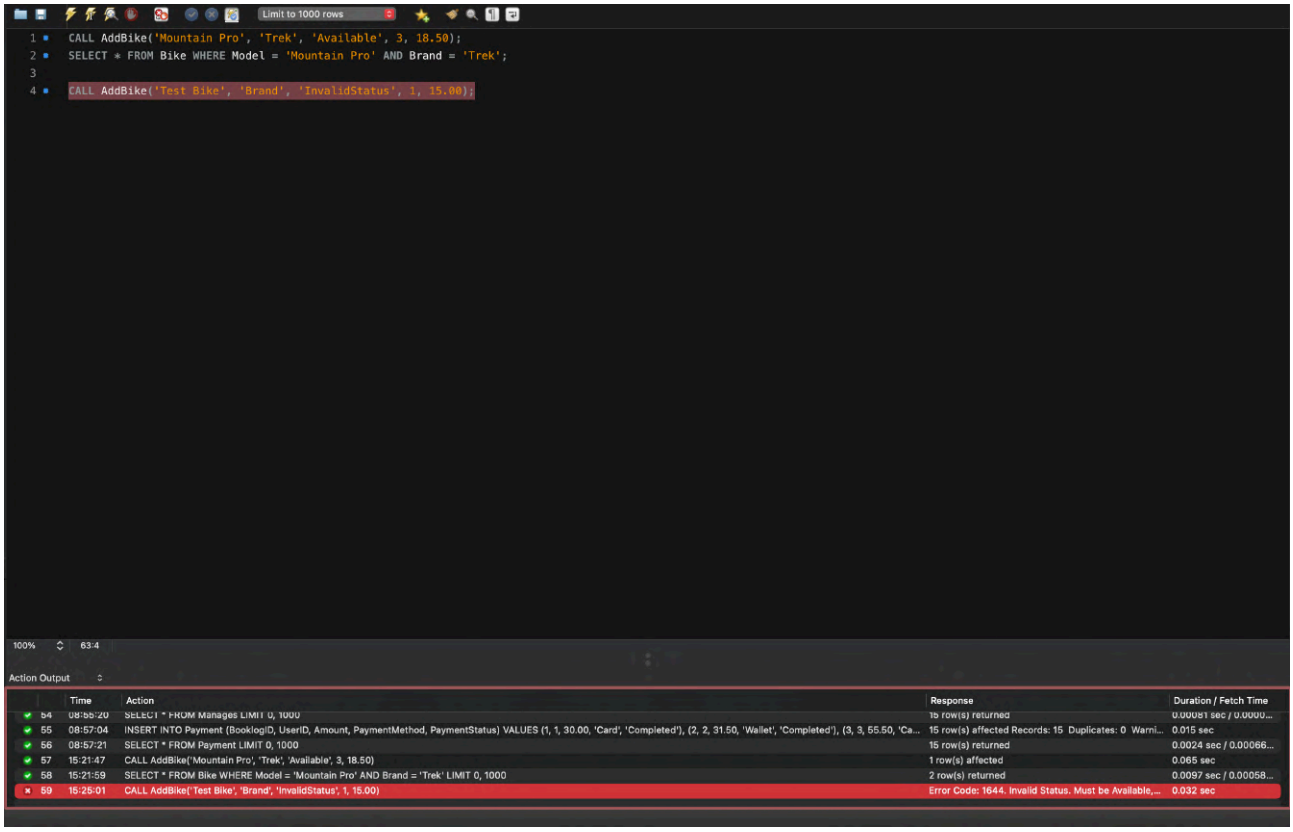
Time	Action	Response	Duration / Fetch Time
08:55:07	INSERT INTO Manages (AdminID, BooklogID) VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 6), (2, 7), (3, 8), (4, 9), (5, 10), (1, 11), (2, 12), (3, 13), (4, 14), (5, 15)	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.0078 sec
08:55:20	SELECT * FROM Manages LIMIT 0, 1000	15 row(s) returned	0.00081 sec / 0.0000...
08:57:04	INSERT INTO Payment (BooklogID, UserID, Amount, PaymentMethod, PaymentStatus) VALUES (1, 1, 30.00, 'Card', 'Completed'), (2, 2, 31.50, 'Wallet', 'Completed'), (3, 3, 55.50, 'Ca...	15 row(s) affected Records: 15 Duplicates: 0 Warni...	0.015 sec
08:57:21	SELECT * FROM Payment LIMIT 0, 1000	15 row(s) returned	0.0024 sec / 0.00066...
15:21:47	CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50)	1 row(s) affected	0.065 sec
15:21:59	SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek' LIMIT 0, 1000	2 row(s) returned	0.0097 sec / 0.00058...

Testing Error Cases :

CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00);

Returns:

Error Code: 1644. Invalid Status. Must be Available, Rented, or Maintenance.



2. Cancel booking Procedure

From Booking table we cancel BookLogID = 2:

*SELECT * FROM Booking WHERE BookLogID = 2;*

Call the cancel Procedure:

CALL CancelBooking(2);

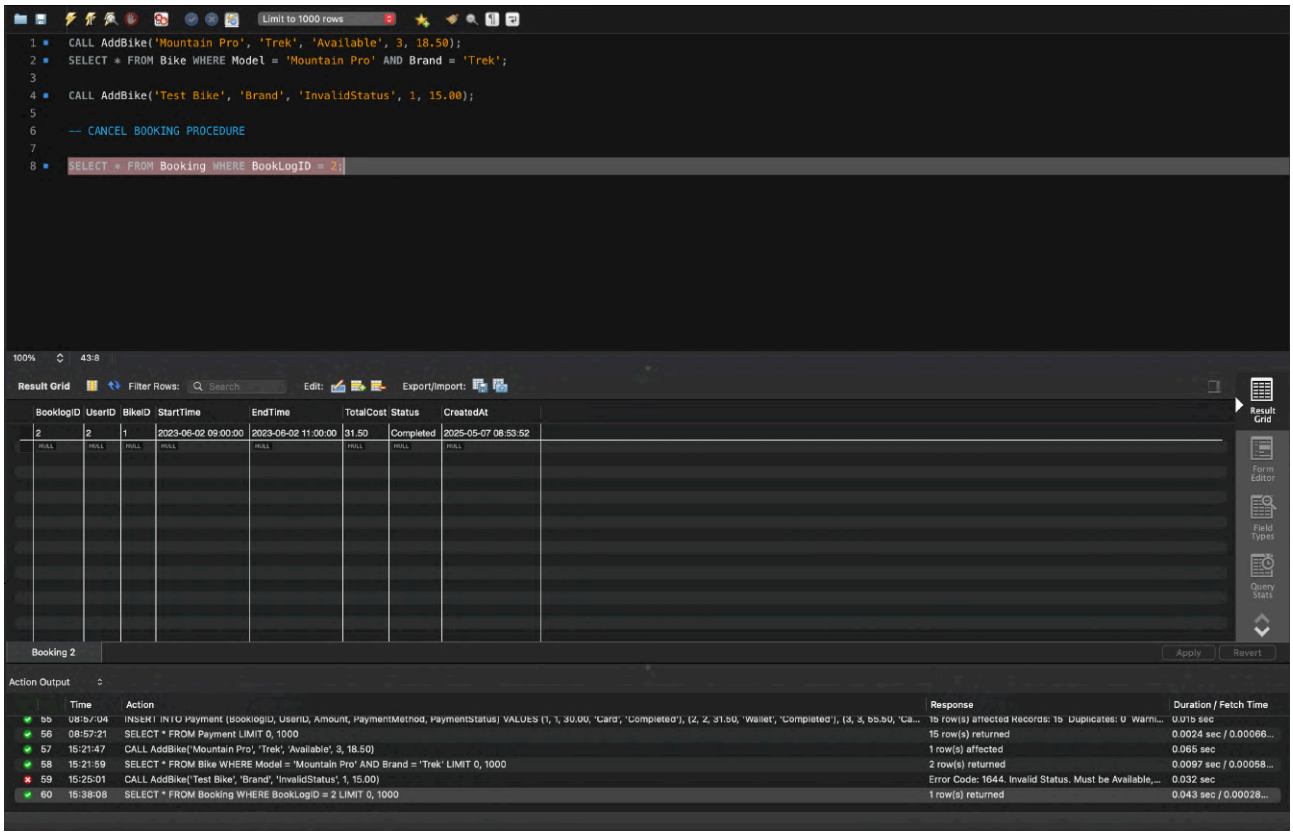
Verify the Updates:

SELECT BooklogID, Status FROM Booking WHERE BooklogID = 2;

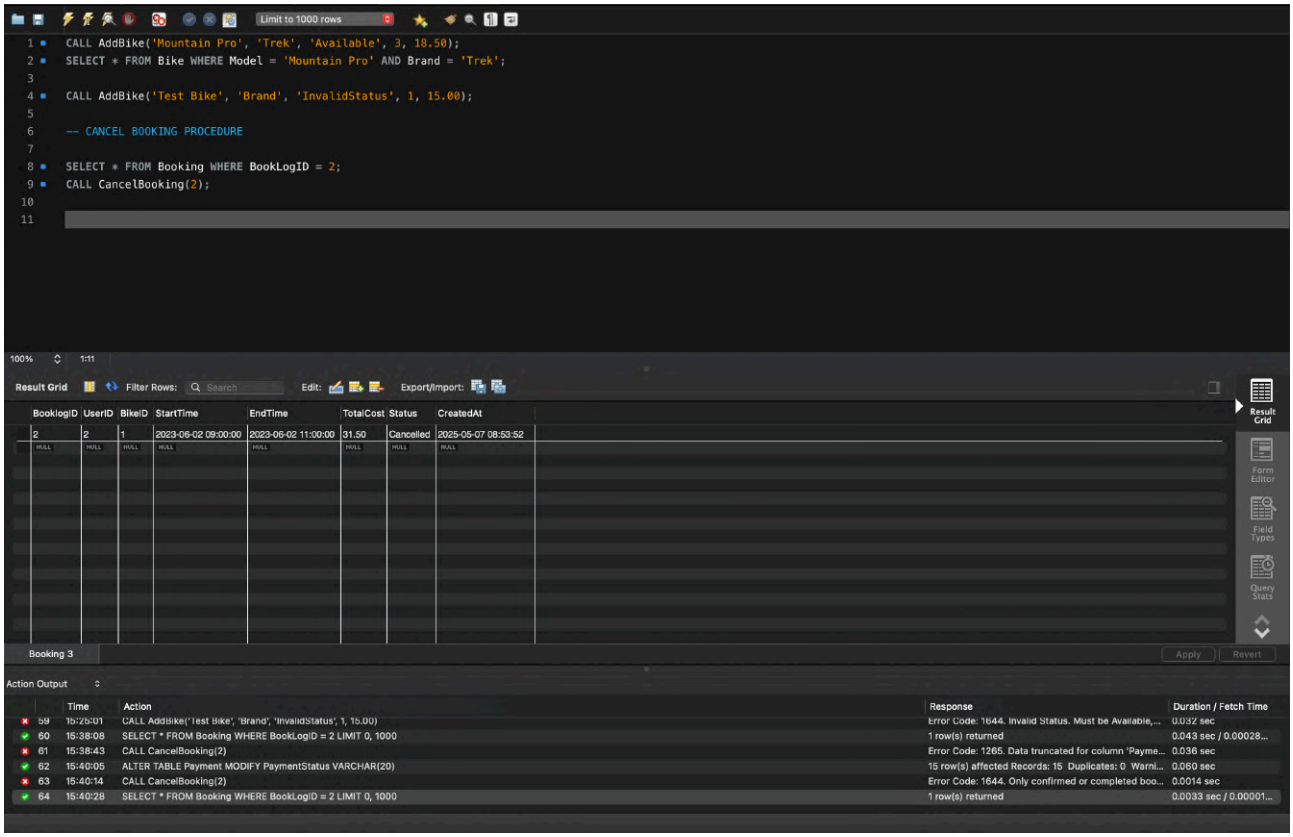
SELECT BikeID, Status FROM Bike WHERE BikeID = 1;

SELECT BooklogID, PaymentStatus FROM Payment WHERE BooklogID = 2;

Before Calling the procedure:



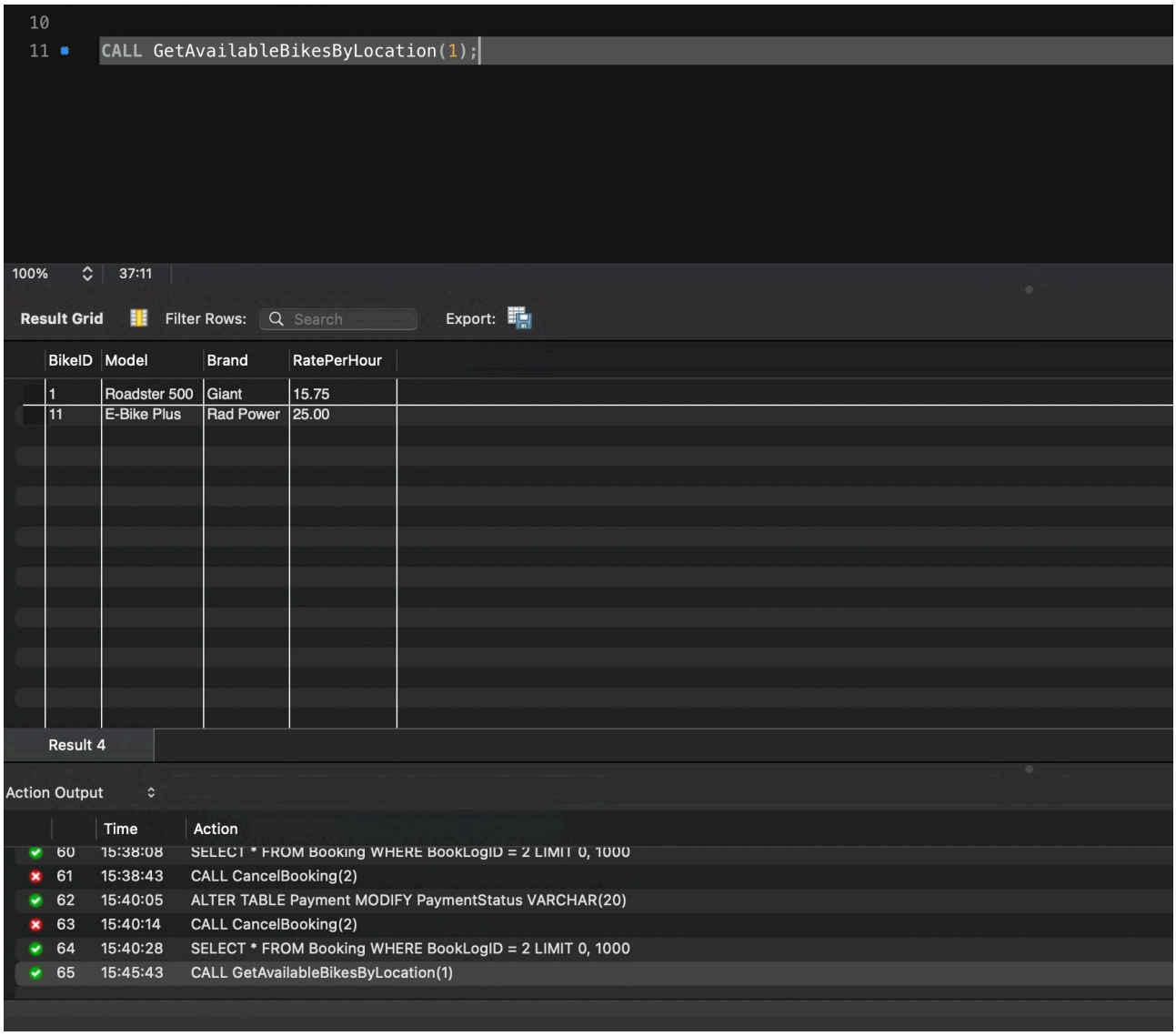
After Calling the Procedure :



3. Get Available Bikes at a location Procedure

Calling the Procedure :

CALL GetAvailableBikesByLocation(1);



4. Get User Booking History

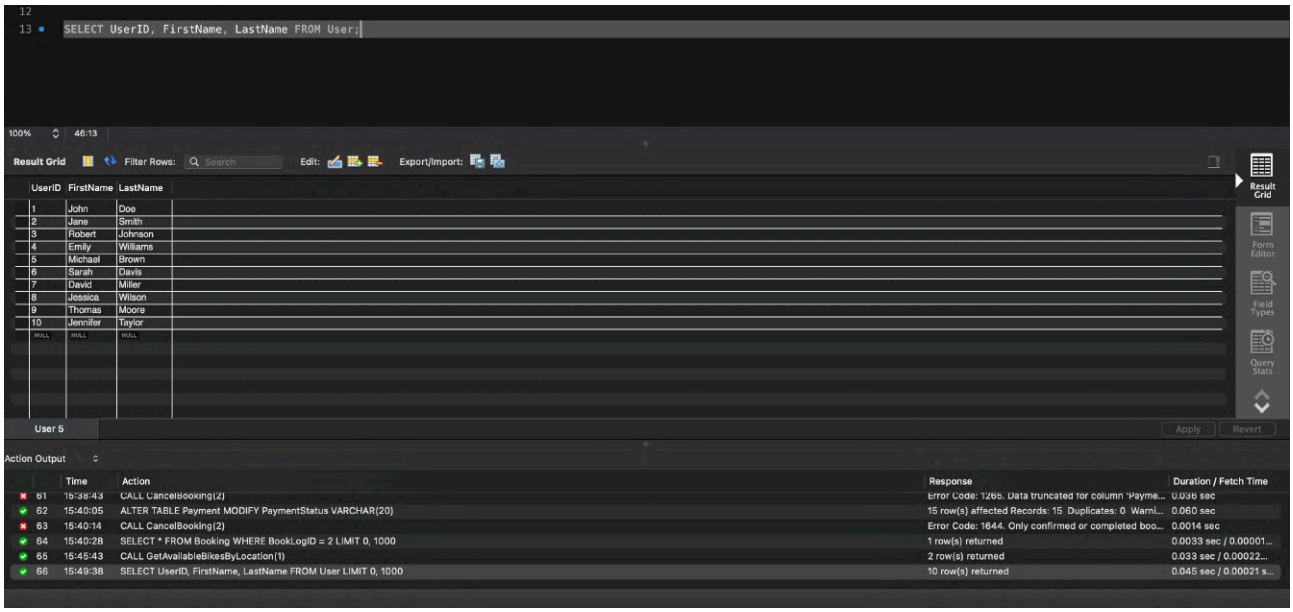
Check the available users :

SELECT UserID, FirstName, LastName FROM User;

Calling the Procedure:

CALL GetUserBookingHistory(1);

Available users:



User Booking History of UserID = 1 :

CONCLUSION

The Bike Rental System project has been designed to offer a seamless and efficient experience for both users and administrators in managing the entire lifecycle of bike rentals. Through an organised and normalised database structure, along with the use of stored procedures for tasks like booking cancellations, checking bike availability, and viewing user booking history, the system ensures accuracy, speed, and reliability. Users can conveniently browse available bikes at specific locations, make or cancel bookings, and complete their payments through multiple methods. At the same time, administrators can manage bookings and payments effectively without the need for manual tracking, reducing errors and workload.

One of the most significant benefits of this system is its contribution to environmental sustainability. By making bike rentals more accessible and efficient, it encourages more individuals to opt for bicycles over traditional motor vehicles. This shift not only helps in reducing fuel consumption and harmful greenhouse gas emissions but also promotes a healthier and cleaner urban lifestyle. As cities worldwide look for smarter, greener mobility solutions, systems like this play a crucial role in building eco-friendly transportation networks.

In summary, this Bike Rental System not only simplifies and modernises the process of renting and managing bikes but also supports a broader vision of sustainable development by promoting environmentally responsible travel choices.