

Database Management System

*Project Report
On*

BIKE RENTAL SYSTEM

By : Nilay Singh

INDEX

S. No.	Content	Page. No.
01.	Introduction	3 - 4
02.	ER-Diagram	5 - 10
03.	ER to Table	11-13
04.	Normalization	14-16
05.	SQL & PL/SQL	17-39
06.	Conclusion	40

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:

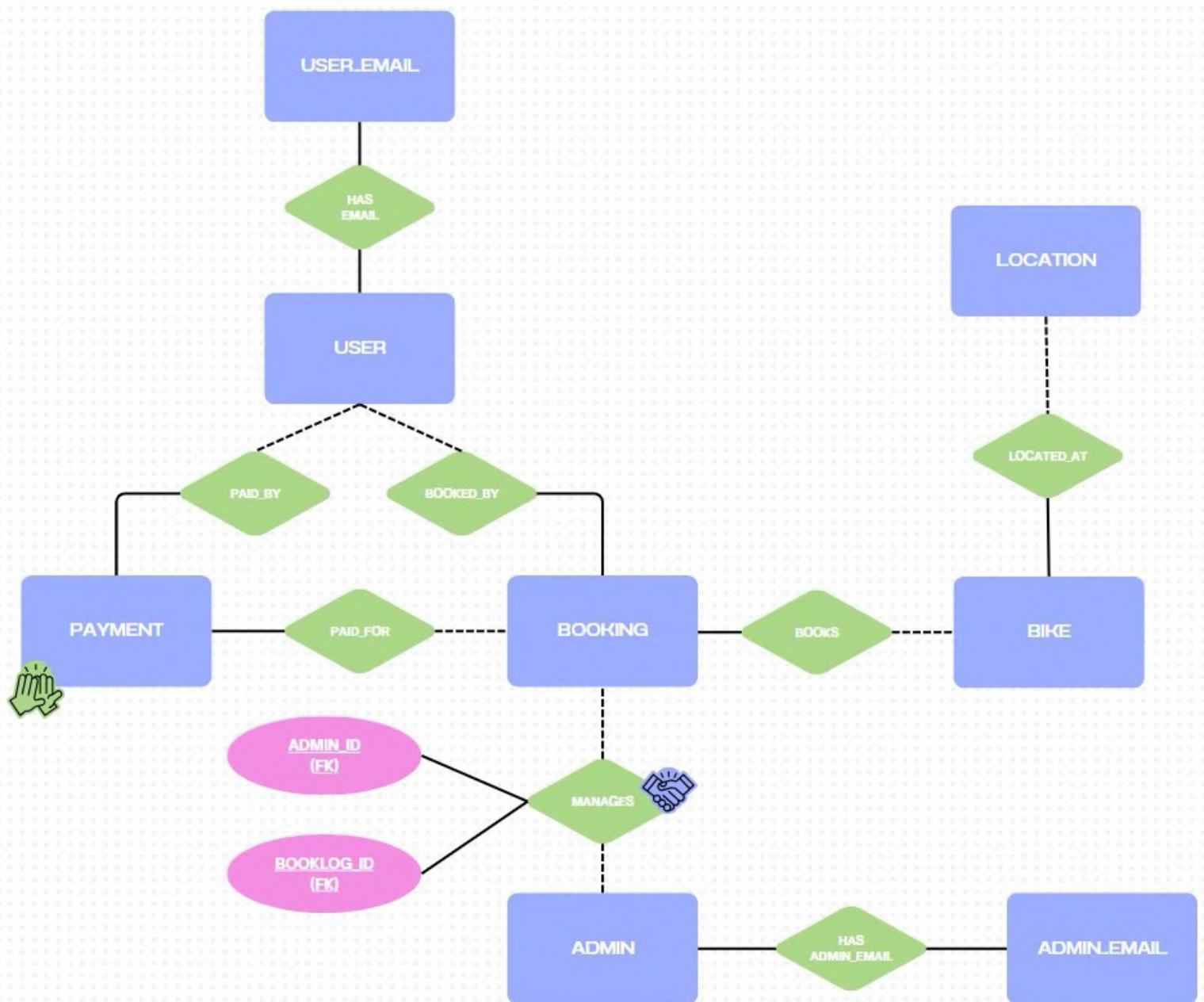
A database is an application that stores the organised collection of records. It can be accessed and managed by the user very easily. Today, many databases available like MySQL, Sybase , Oracle, MongoDB, PostgreSQL, SQL Server, etc. We used MySQL server for managing data.

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.

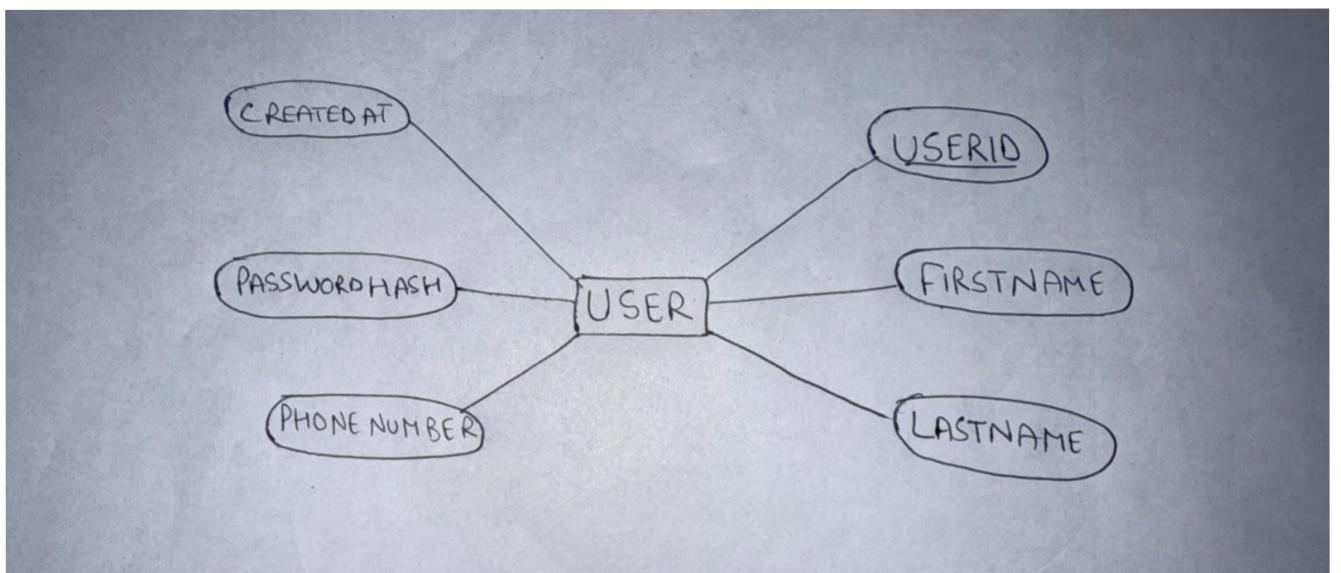
MySQL follows the working of Client-Server Architecture. This model is designed for the end-users called clients to access the resources from a central computer known as a server using network services. Here, the clients make requests through a graphical user interface (GUI), and the server will give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

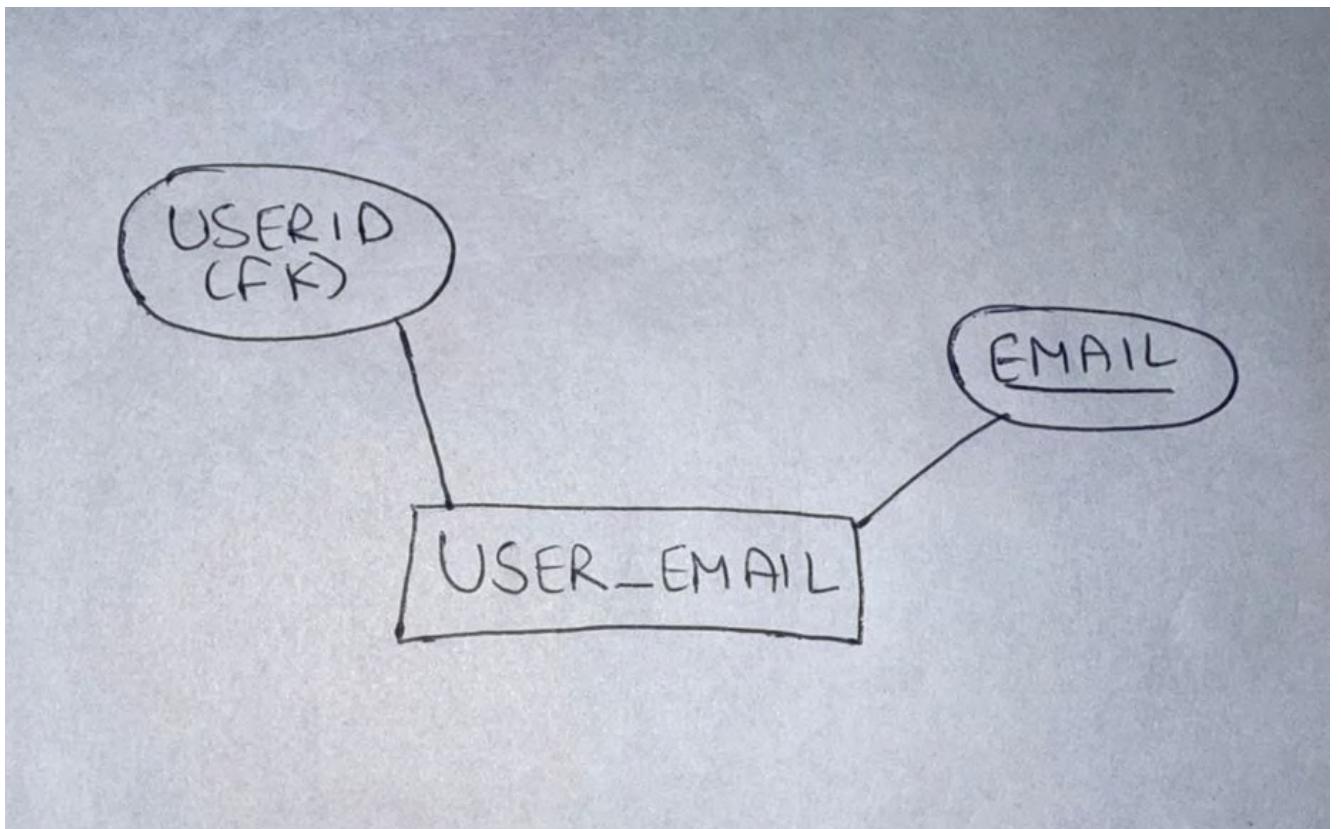
2. ER-DIAGRAM



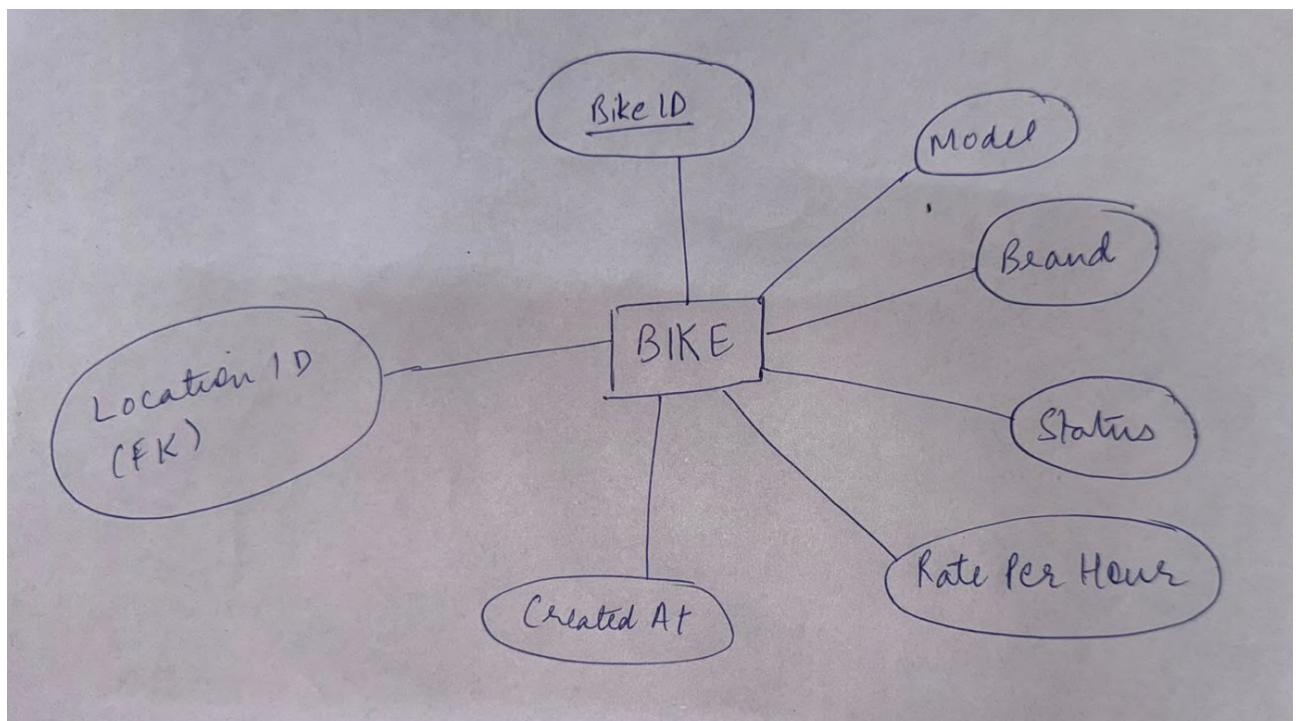
1. USER



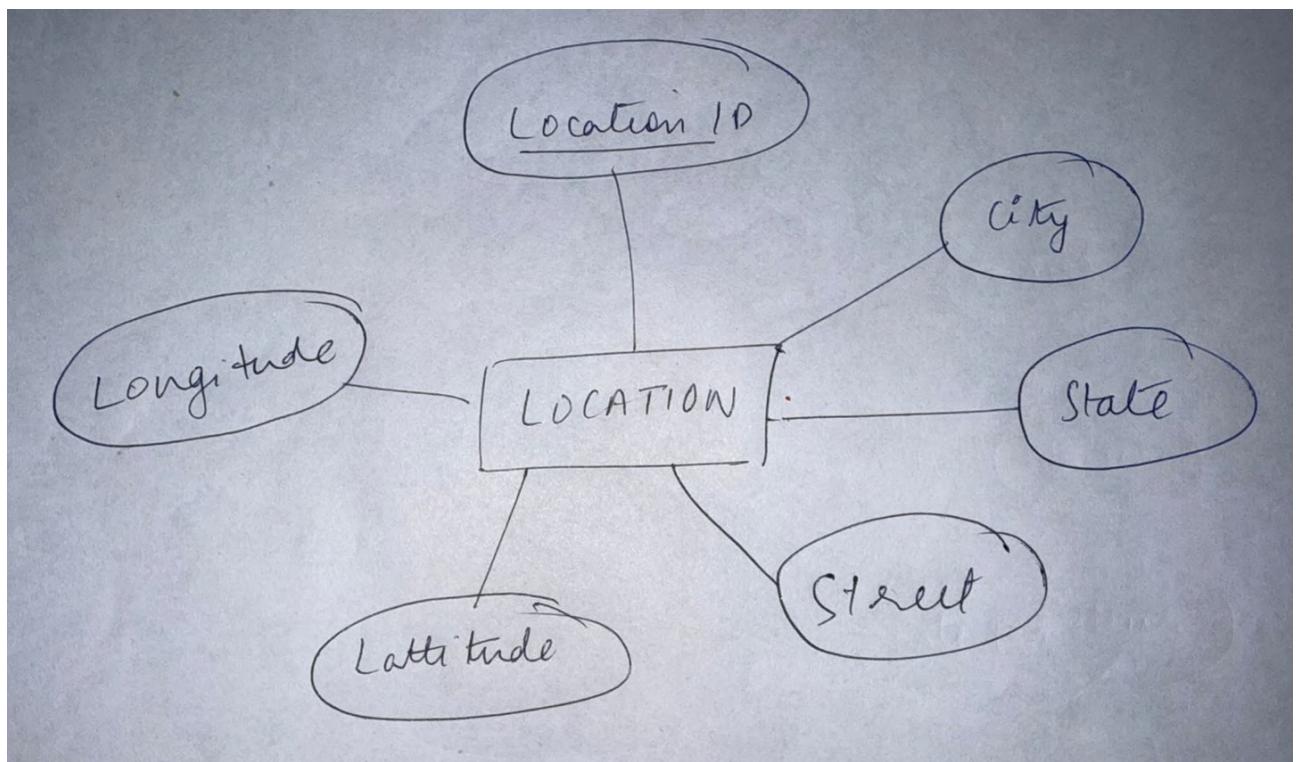
2. USER_EMAIL



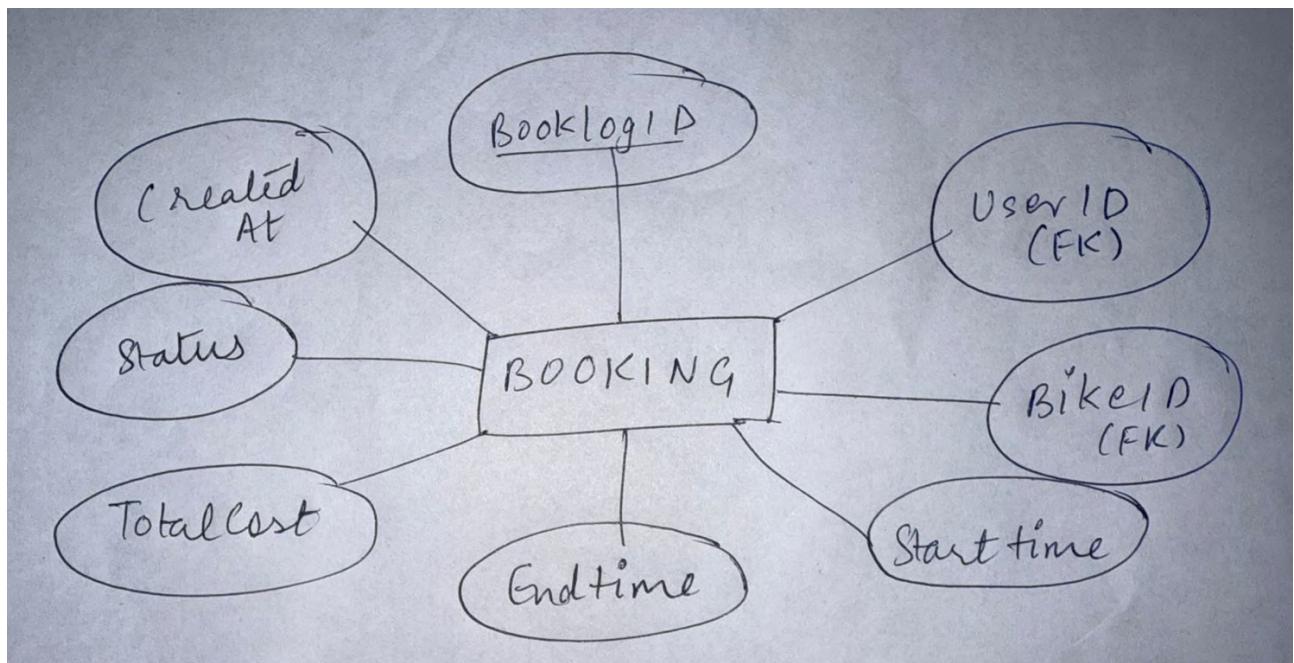
3. BIKE



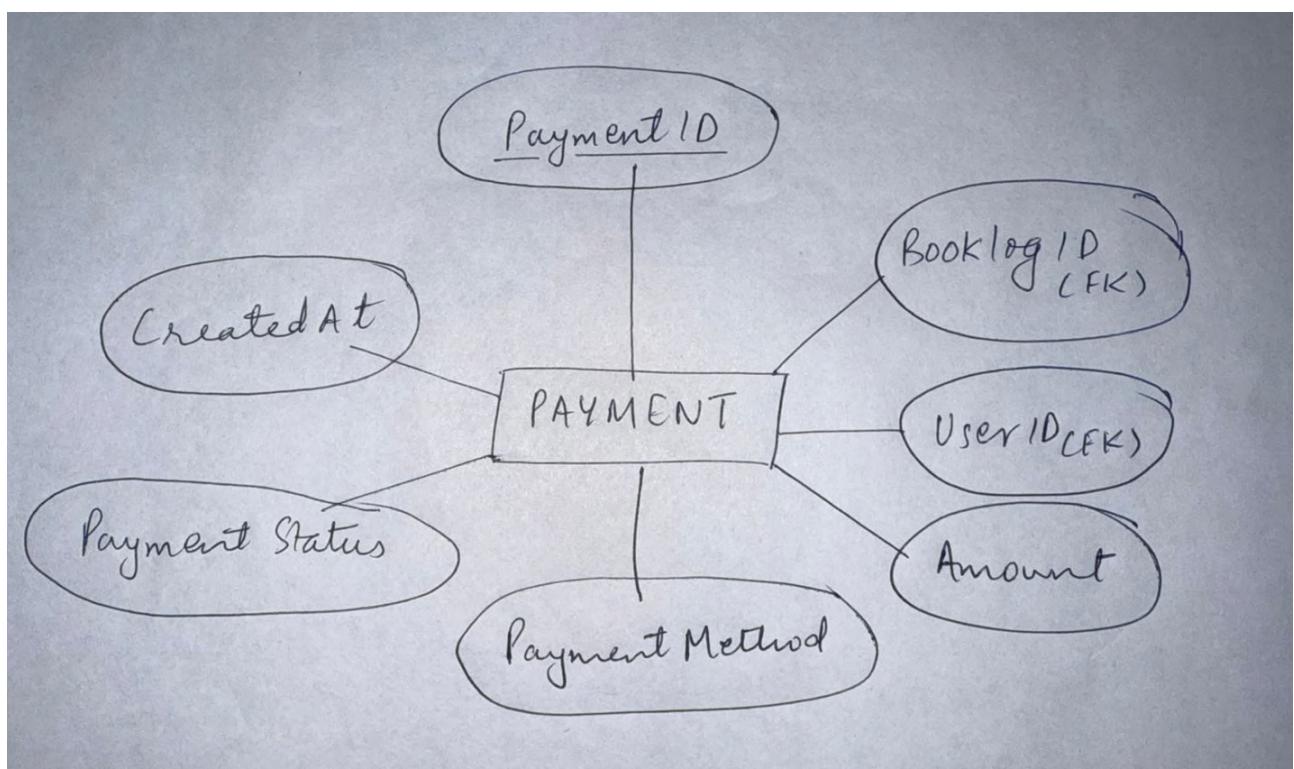
4. LOCATION



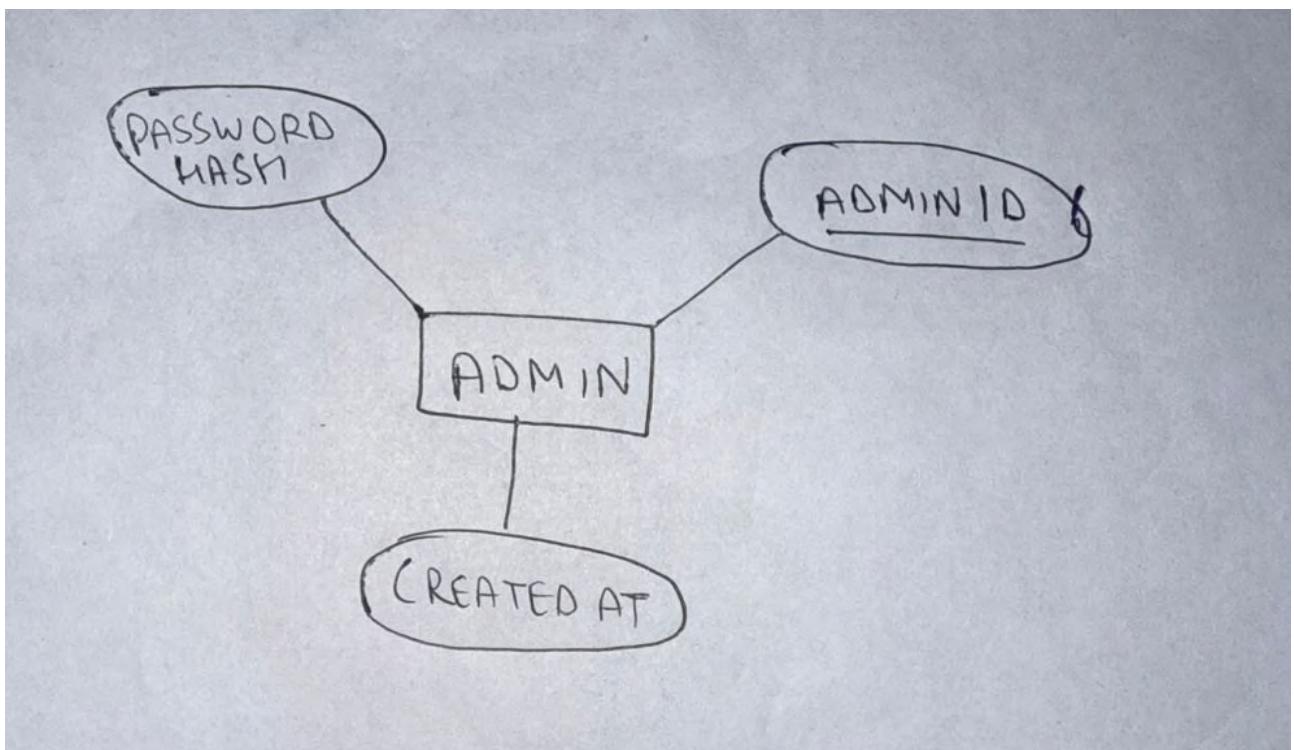
5. BOOKING



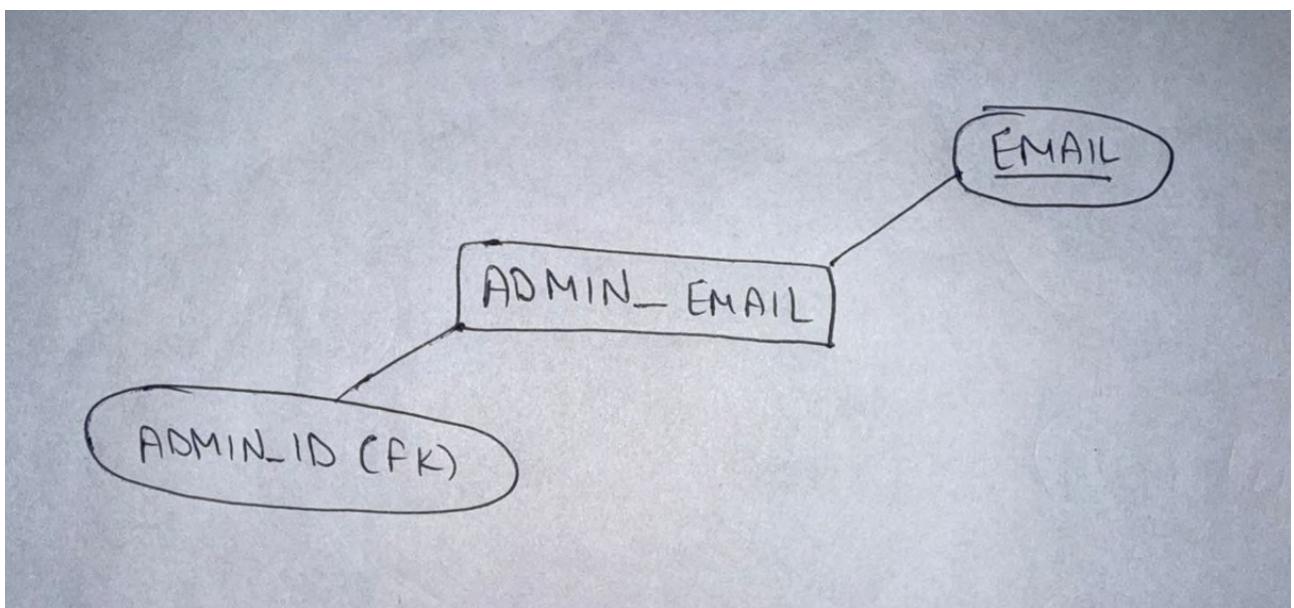
6. PAYMENT



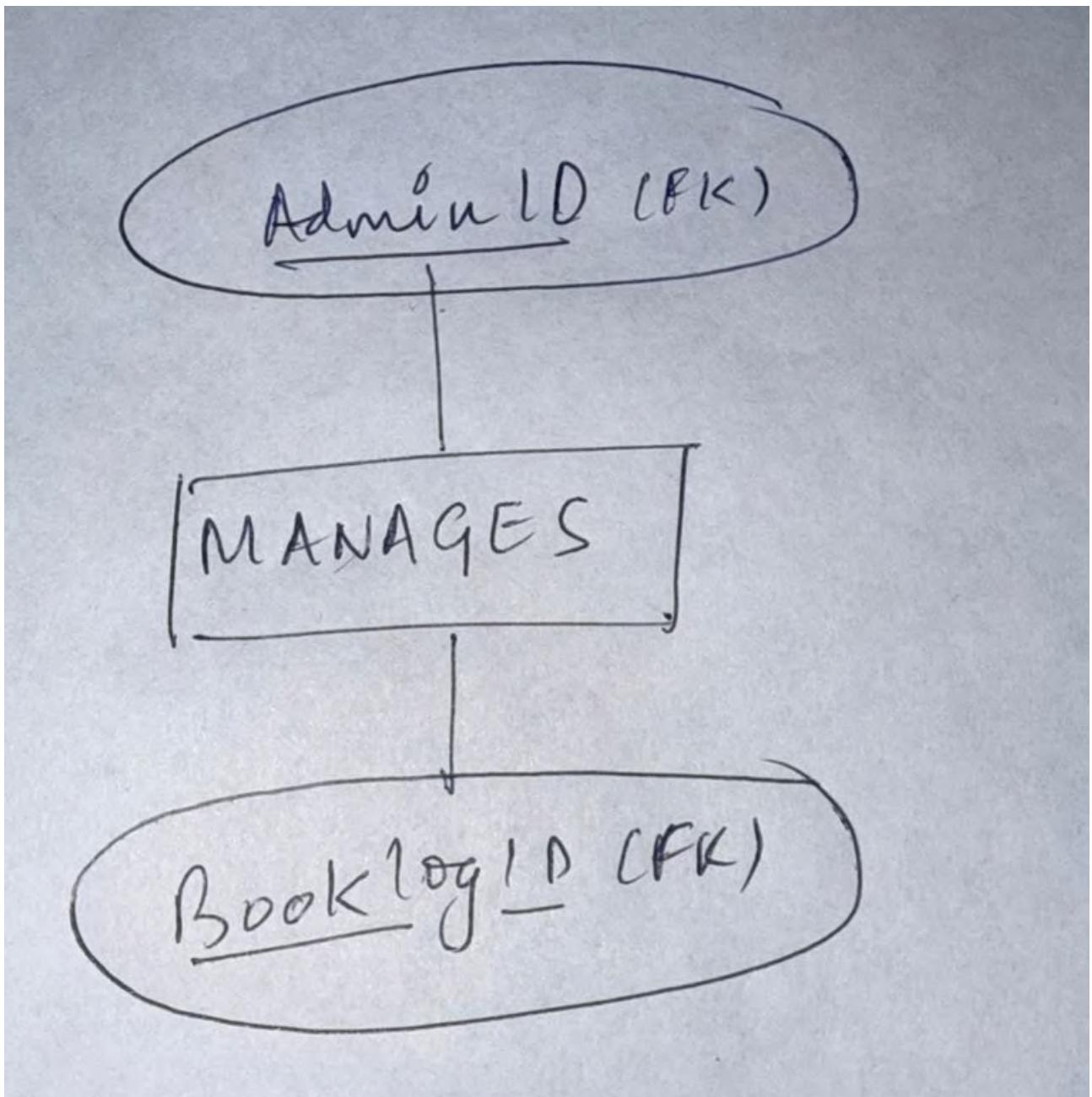
7. ADMIN



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. MANAGES

- 1. *Admin (0,1) ↔ Booking (1,N)*
 - An admin can manage multiple bookings, but a booking is managed by at most one admin (optional).

2. BOOKS

- 2. *User (1,N) ↔ Booking (1,1)*
 - A user can create multiple bookings, but each booking is made by exactly one user.

3. ASSIGNED_TO (between BIKE and BOOKING)

- 3. *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. LOCATED_AT

- 4. *Bike (0,N) ↔ Location (1,1)*
 - A bike is assigned to exactly one location, and a location can have multiple bikes (or none).

5. PAYS_FOR

- 5. *Booking (1,1) ↔ Payment (1,N)*
 - A booking can have multiple payments (e.g., installments), but each payment is tied to one booking.

6. MAKES_PAYMENT

- 6. *User (1,1) ↔ Payment (0,N)*
 - A user can make multiple payments, but each payment is made by one user.

7.

Conversion of ER to Tables

1. USER

1. $\text{UserID} \rightarrow \{\text{FirstName}, \text{LastName}, \text{Email}, \text{PhoneNumber}, \text{PasswordHash}, \text{CreatedAt}\}$

2. BIKE

1. $\text{BikeID} \rightarrow \{\text{Model}, \text{Brand}, \text{Status}, \text{LocationID}, \text{RatePerHour}, \text{CreatedAt}\}$

3. LOCATION

$\text{LocationID} \rightarrow \{\text{City}, \text{State}, \text{Street}, \text{Latitude}, \text{Longitude}\}$

4. BOOKING

$\text{BooklogID} \rightarrow \{\text{UserID}, \text{BikeID}, \text{StartTime}, \text{EndTime}, \text{TotalCost}, \text{Status}, \text{CreatedAt}\}$

5. PAYMENT

$\text{PaymentID} \rightarrow \{\text{BooklogID}, \text{UserID}, \text{Amount}, \text{PaymentMethod}, \text{PaymentStatus}, \text{CreatedAt}\}$

6. ADMIN

$\text{AdminID} \rightarrow \{\text{Email}, \text{PasswordHash}, \text{CreatedAt}\}$

7. MANAGES (Admin-Booking Relationship)

$\text{AdminID} \rightarrow \text{BooklogID}$

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*
 - $UserID \rightarrow \{FirstName, LastName, PhoneNumber, PasswordHash, CreatedAt\}$
2. *USER_EMAIL*
 - $Email \rightarrow UserID$
 - $UserID \rightarrow Email$ (*1:1 relationship, since each user has one email and vice versa*)
3. *BIKE*
 - $BikeID \rightarrow \{Model, Brand, Status, LocationID, RatePerHour, CreatedAt\}$
 - $LocationID \rightarrow \{City, State, Street, Latitude, Longitude\}$ (*via LOCATION table*)
4. *LOCATION*
 - $LocationID \rightarrow \{City, State, Street, Latitude, Longitude\}$
5. *BOOKING*
 - $BooklogID \rightarrow \{UserID, BikeID, StartTime, EndTime, TotalCost, Status, CreatedAt\}$
 - $\{UserID, BikeID, StartTime\} \rightarrow BooklogID$ (*Candidate key for uniqueness*)
6. *PAYMENT*
 - $PaymentID \rightarrow \{BooklogID, UserID, Amount, PaymentMethod, PaymentStatus, CreatedAt\}$
 - $BooklogID \rightarrow \{UserID, BikeID, ...\}$ (*Partial dependency resolved via foreign key*)
7. *ADMIN*
 - $AdminID \rightarrow \{PasswordHash, CreatedAt\}$
8. *ADMIN_EMAIL*
 - $Email \rightarrow AdminID$
 - $AdminID \rightarrow Email$ (*1:1 relationship*)
9. *MANAGES* (*Admin-Booking Relationship*)
 - $\{AdminID, BooklogID\} \rightarrow \{\}$ (*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;
```

The screenshot shows the MySQL Workbench interface. In the top-left corner, there is a tab labeled "Query 1". Below the tabs, there is a toolbar with various icons. The main area contains the following SQL code:

```
1 -- Creating the Database  
2 * CREATE DATABASE BikeRentalSystem;  
3  
4 -- 2. Use the database  
5 * USE BikeRentalSystem;
```

Below the code, there is a table titled "Actions" which shows the execution details:

Time	Action	Response	Duration / Fetch Time
16:47:50	CREATE DATABASE BikeRentalSystem	1 row(s) affected	0.029 sec
16:47:50	USE BikeRentalSystem	0 row(s) affected	0.0012 sec

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)  
);
```

The screenshot shows the MySQL Workbench interface. In the top-left corner, there is a tab labeled "Query 1". Below the tabs, there is a toolbar with various icons. The main area contains the following SQL code:

```
7 -- 3. LOCATION TABLE  
8 * CREATE TABLE Location (  
9     LocationID INT AUTO_INCREMENT PRIMARY KEY,  
10    City VARCHAR(50),  
11    State VARCHAR(50),  
12    Street VARCHAR(100),  
13    Latitude DECIMAL(10, 8),  
14    Longitude DECIMAL(10, 8)  
15 );  
16
```

Action	Output	Time	Action	Response	Duration / Fetch Time
✓ 1	CREATE TABLE Location (LocationID INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL... 0 row(s) affected	16:54:00			0.010 sec

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
);
```

```
10
11      -- 4. USER TABLE
12
13 • CREATE TABLE User (
14     UserID INT AUTO_INCREMENT PRIMARY KEY,
15     FirstName VARCHAR(50),
16     LastName VARCHAR(50),
17     PhoneNumber VARCHAR(20),
18     PasswordHash VARCHAR(255),
19     CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
20 );
21
22
23      -- 5. USER EMAIL TABLE
24
25 • CREATE TABLE User_Email (
26     Email VARCHAR(100) UNIQUE,
27     UserID INT UNIQUE,
28     FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
29 );
30
31
32 );
```

		Time	Action	Response	Duration / Fetch Time
✓ 1	CREATE TABLE Location (LocationID INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL... 0 row(s) affected	16:54:00			0.010 sec
✓ 2	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), Passwo... 0 row(s) affected	16:54:26			0.023 sec
✓ 3	CREATE TABLE User_Email (Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE) 0 row(s) affected	16:54:30			0.014 sec

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
);
```

```
-- 35
36      -- 6. ADMIN TABLE
37  * ⊖ CREATE TABLE Admin (
38      AdminID INT AUTO_INCREMENT PRIMARY KEY,
39      PasswordHash VARCHAR(255),
40      CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
41  );
42
43      -- 7. ADMIN EMAIL TABLE
44  * ⊖ CREATE TABLE Admin_Email (
45      Email VARCHAR(100) UNIQUE,
46      AdminID INT UNIQUE,
47      FOREIGN KEY (AdminID) REFERENCES Admin/AdminID) ON DELETE CASCADE
48  );
49
```

Action	Output	Time	Action	Response	Duration / Fetch Time
✓ 1	CREATE TABLE Location (LocationId INT AUTO_INCREMENT PRIMARY KEY, City VARCHAR(50), State VARCHAR(50), Street VARCHAR(100), Latitude DECIMAL...,	16:54:00		0 row(s) affected	0.010 sec
✓ 2	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), Passwo...,	16:54:26		0 row(s) affected	0.023 sec
✓ 3	CREATE TABLE User_Email (Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE)	16:54:30		0 row(s) affected	0.014 sec
✓ 4	CREATE TABLE Admin (AdminID INT AUTO_INCREMENT PRIMARY KEY, PasswordHash VARCHAR(255), CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP)	16:55:20		0 row(s) affected	0.014 sec
✓ 5	CREATE TABLE Admin_Email (Email VARCHAR(100) UNIQUE, AdminID INT UNIQUE, FOREIGN KEY (AdminID) REFERENCES Admin(AdminID) ON DELETE CASCADE)	16:55:25		0 row(s) affected	0.014 sec

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	0.01 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:

```

-- 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..., Email VARCHAR(100) UNIQUE, UserID INT UNIQUE, FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE)	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..., Email VARCHAR(100) UNIQUE, AdminID INT UNIQUE, FOREIGN KEY (AdminID) REFERENCES Admin/AdminID ON DELETE CASCADE)	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..., BooklogID INT, UserID INT, Amount DECIMAL(6,2), PaymentMethod ENUM('Cash', 'Wallet',...)	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

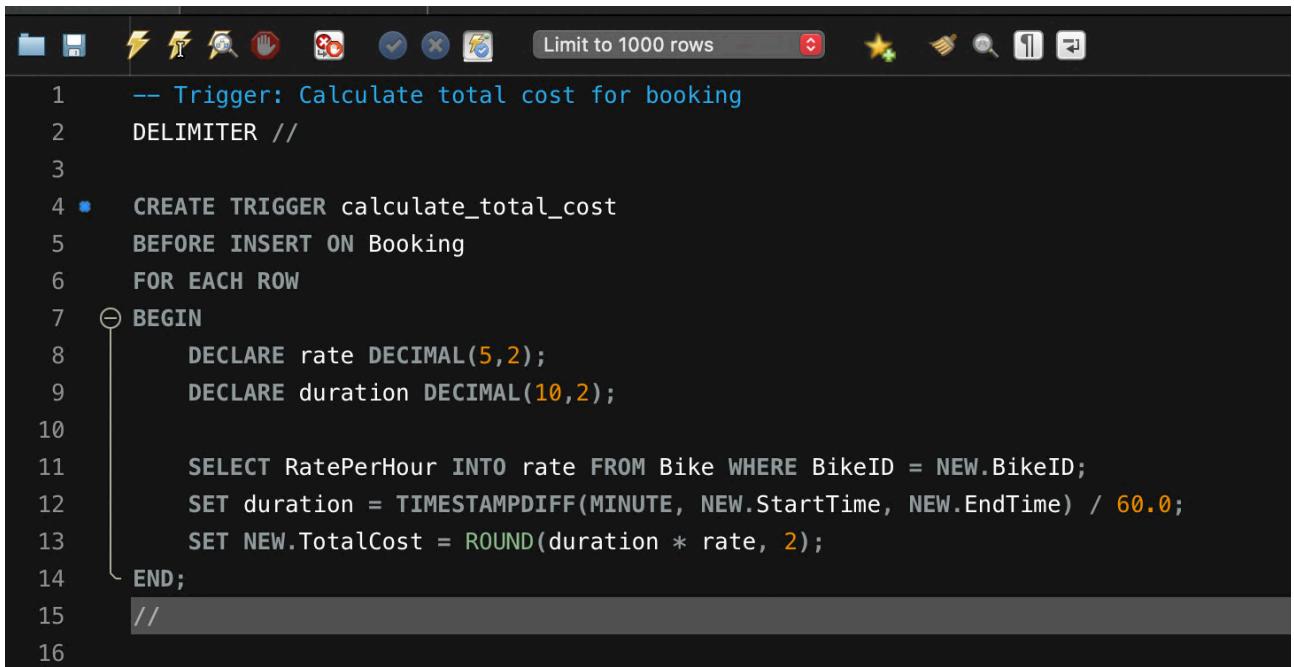
7. Creating a Trigger to Calculate the total cost of booking

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.



The screenshot shows the MySQL Workbench interface with the trigger code for 'calculate_total_cost' pasted into the SQL editor. The code is identical to the one provided above, including the comments and the use of the 'rate' and 'duration' variables to calculate the total cost based on the booking duration and the bike's rate per hour.

```
-- Trigger: Calculate total cost for 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;
//
```

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;

//

```
16
17  DELIMITER //
18
19  CREATE TRIGGER prevent_double_booking
20  BEFORE INSERT ON Booking
21  FOR EACH ROW
22  BEGIN
23      DECLARE count_bookings INT;
24
25      SELECT COUNT(*) INTO count_bookings
26      FROM Booking
27      WHERE BikeID = NEW.BikeID
28      AND NEW.StartTime < EndTime
29      AND NEW.EndTime > StartTime;
30
31      IF count_bookings > 0 THEN
32          SIGNAL SQLSTATE '45000';
33          SET MESSAGE_TEXT = 'Bike already booked during this time';
34      END IF;
35  END;
36  //
```

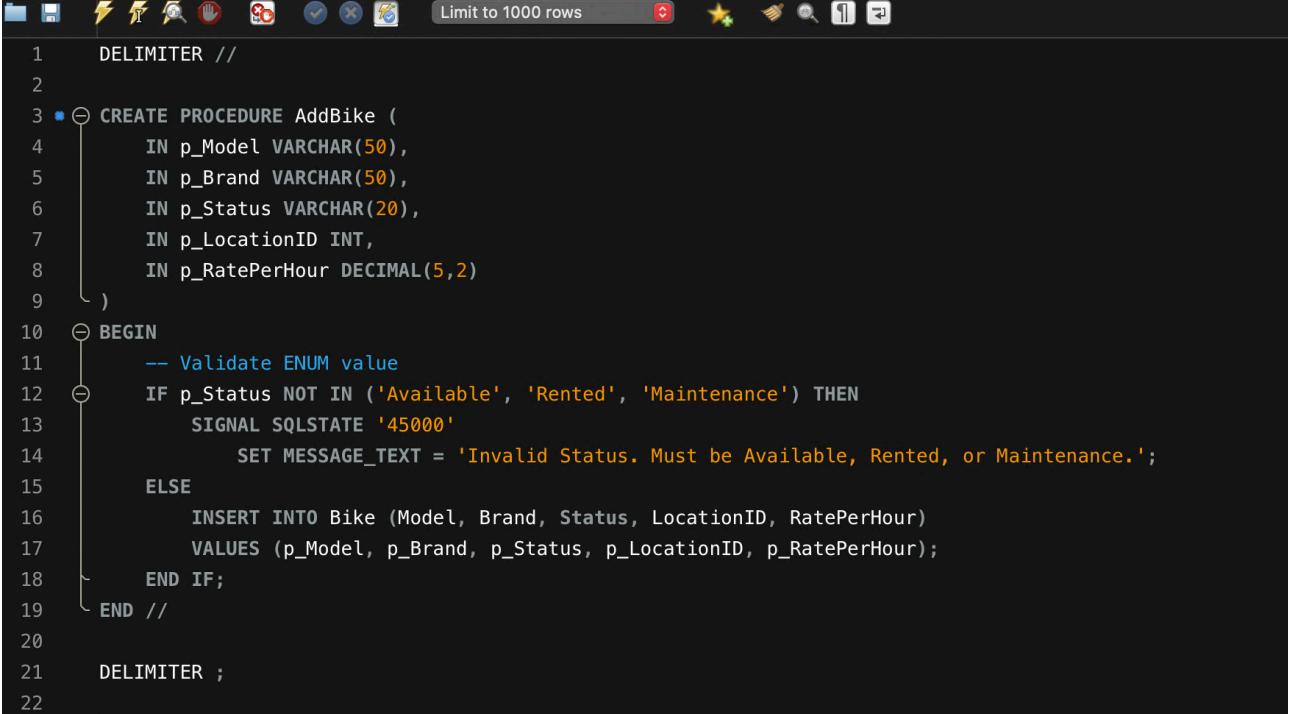
Action Output

Time	Action	Response	Duration / Fetch Time
16-09-37	CREATE TABLE BIKE (BikeID INT AUTO_INCREMENT PRIMARY KEY, Model VARCHAR(50), Brand VARCHAR(50), Status ENUM('Available', 'Rented', 'Maintenance'), Location VARCHAR(100))	0 rows affected	0.009 sec
7 16:56:51	CREATE TABLE Booking (BooklogID INT AUTO_INCREMENT PRIMARY KEY, UserID INT, BikeID INT, StartTime DATETIME, EndTime DATETIME, TotalCost DECIMAL(6,2))	0 rows 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 (BooklogID) REFERENCES Booklog(BooklogID))	0 rows 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 rows affected	0.017 sec
10 18:09:36	CREATE TRIGGER calculate_total_cost BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE total_cost DECIMAL(5,2); DECLARE duration DECIMAL(10,2); SELECT duration * (EndTime - StartTime) INTO total_cost; END;	0 rows affected	0.070 sec
11 18:13:13	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings; IF count_bookings > 0 THEN SIGNAL SQLSTATE '45000'; SET MESSAGE_TEXT = 'Bike already booked during this time'; END IF; END;	Error Code: 1064. You have an error in your SQL syntax near 'SELECT COUNT(*) INTO count_bookings' at line 1	0.013 sec
12 18:13:54	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings; IF count_bookings > 0 THEN SIGNAL SQLSTATE '45000'; SET MESSAGE_TEXT = 'Bike already booked during this time'; END IF; END;	Error Code: 1064. You have an error in your SQL syntax near 'SELECT COUNT(*) INTO count_bookings' at line 1	0.00086 sec
13 18:14:05	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings; IF count_bookings > 0 THEN SIGNAL SQLSTATE '45000'; SET MESSAGE_TEXT = 'Bike already booked during this time'; END IF; END;	Error Code: 1064. You have an error in your SQL syntax near 'SELECT COUNT(*) INTO count_bookings' at line 1	0.00044 sec
14 18:15:16	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT; SELECT COUNT(*) INTO count_bookings; IF count_bookings > 0 THEN SIGNAL SQLSTATE '45000'; SET MESSAGE_TEXT = 'Bike already booked during this time'; END IF; END;	Error Code: 1064. You have an error in your SQL syntax near 'SELECT COUNT(*) INTO count_bookings' at line 1	0.00077 sec
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; IF count_bookings > 0 THEN SIGNAL SQLSTATE '45000'; SET MESSAGE_TEXT = 'Bike already booked during this time'; END IF; END;	0 rows affected	0.064 sec

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 //
```



The screenshot shows the MySQL Workbench interface with the stored procedure code for 'AddBike'. The code is identical to the one provided above, including the DELIMITER // at the top and the END // at the bottom. The code is syntax-highlighted, with keywords like CREATE, PROCEDURE, IF, SIGNAL, and SET in blue, and identifiers like Model, Brand, Status, LocationID, RatePerHour, and the procedure name in black. Brackets and parentheses are shown in grey. The code is presented in a scrollable window with line numbers on the left.

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 //
```

```
38     DELIMITER //
39
40 •  CREATE TRIGGER update_bike_status_after_booking
41     AFTER INSERT ON Booking
42     FOR EACH ROW
43     BEGIN
44         IF NEW.Status = 'Confirmed' THEN
45             UPDATE Bike
46             SET Status = 'Rented'
47             WHERE BikeID = NEW.BikeID;
48         ELSEIF NEW.Status = 'Cancelled' THEN
49             UPDATE Bike
50             SET Status = 'Available'
51             WHERE BikeID = NEW.BikeID;
52         END IF;
53     END //
```

DELIMITER ;

Action	Time	Time	Action	Response	Duration / Fetch Time
✓ 9	16:07:08	CREATE TABLE Payment (PaymentID INT AUTO_INCREMENT PRIMARY KEY, BookingsID INT, UserID INT, Amount DECIMAL(10,2), PaymentMethod ENUM('Cash', 'Wallet', ...)		0 row(s) affected	0.01 sec
✓ 10	18:09:36	CREATE TRIGGER calculate_total_cost BEFORE INSERT ON Booking FOR EACH ROW BEGIN	DECLARE duration DECIMAL(10,2);	0 row(s) affected	0.070 sec
✗ 11	18:13:13	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN	DECLARE count_bookings INT	Error Code: 1064. You have an error in your SQL syn...	0.013 sec
✗ 12	18:13:54	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN	DECLARE count_bookings INT	Error Code: 1064. You have an error in your SQL syn...	0.00086 sec
✗ 13	18:14:05	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN	DECLARE count_bookings INT	Error Code: 1064. You have an error in your SQL syn...	0.00044 sec
✗ 14	18:15:16	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN	DECLARE count_bookings INT	Error Code: 1064. You have an error in your SQL syn...	0.00077 sec
✓ 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	0 row(s) affected	0.064 sec
✗ 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,...	IF NEW.Status = 'Confirmed' THEN	0 row(s) affected	0.088 sec
✓ 18	08:05:02	CREATE TRIGGER update_bike_status_after_booking AFTER INSERT ON Booking FOR EACH ROW BEGIN	UPDATE Bike SET Status =...	0 row(s) affected	0.090 sec

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.

```

42
43     DELIMITER //
44
45 ■ ⊖ CREATE PROCEDURE CancelBooking(
46     IN p_BooklogID INT
47 )
48 BEGIN
49
50     DECLARE v_BikeID INT;
51     DECLARE v_Status VARCHAR(20);
52
53     -- Get the Bike ID and Status from Booking
54     SELECT BikeID, Status INTO v_BikeID, v_Status
55     FROM Booking
56     WHERE BooklogID = p_BooklogID;
57
58     -- Check if the booking is Confirmed or Completed
59     IF v_Status IN ('Confirmed', 'Completed') THEN
60         -- Update Bike Status
61         UPDATE Bike
62             SET Status = 'Available'
63             WHERE BikeID = v_BikeID;
64
65         -- Update Booking Status
66         UPDATE Booking
67             SET Status = 'Cancelled'
68             WHERE BooklogID = p_BooklogID;
69
70         -- Update Payment Status
71         UPDATE Payment
72             SET PaymentStatus = 'Refunded'
73             WHERE BooklogID = p_BooklogID;
74
75         -- Signal error
76         SIGNAL SQLSTATE '45000'
77             SET MESSAGE_TEXT = 'Only confirmed or completed bookings can be cancelled.';
78     END IF;
79
80 END //
81
82     DELIMITER ;

```

Action Output

	Time	Action	Responsible
✓ 10	18:08:30	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...
✗ 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...
✗ 16	07:48:19	Apply changes to new_table	0 rows
✓ 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 rows
✓ 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 =...
✓ 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...

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
```

Action Output

	Time	Action	Response
✓ 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 rows
✓ 2	16:54:26	CREATE TABLE User (UserID INT AUTO_INCREMENT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), PhoneNumber VARCHAR(20), PasswordHash...)	0 rows
✓ 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 rows
✓ 4	16:55:20	CREATE TABLE Admin (AdminID INT AUTO_INCREMENT PRIMARY KEY, PasswordHash VARCHAR(255), CreatedAt DATETIME DEFAULT CURRENT_TIMESTAMP)	0 rows
✓ 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 rows
✓ 6	16:56:37	CREATE TABLE Bike (BikeID INT AUTO_INCREMENT PRIMARY KEY, Model VARCHAR(50), Brand VARCHAR(50), Status ENUM('Available', 'Rented', 'Maintenance'), LocationID INT, ...)	0 rows
✓ 7	16:56:51	CREATE TABLE Booking (BooklogID INT AUTO_INCREMENT PRIMARY KEY, UserID INT, BikeID INT, StartTime DATETIME, EndTime DATETIME, TotalCost DECIMAL(6,2))	0 rows
✓ 8	16:57:05	CREATE TABLE Manages (AdminID INT, BooklogID INT, PRIMARY KEY (AdminID, BooklogID), FOREIGN KEY (AdminID) REFERENCES Admin/AdminID), FOREIGN KEY (Boo...)	0 rows
✓ 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 rows
✓ 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 rows
✗ 11	18:13:13	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error O
✗ 12	18:13:54	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error O
✗ 13	18:14:05	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error O
✗ 14	18:15:16	CREATE TRIGGER prevent_double_booking BEFORE INSERT ON Booking FOR EACH ROW BEGIN DECLARE count_bookings INT	Error O
✓ 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 rows
✗ 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 rows
✓ 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 rows
✓ 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 rows
✓ 20	08:14:25	CREATE PROCEDURE GetAvailableBikesByLocation (IN p_LocationID INT) BEGIN SELECT BikeID, Model, Brand, RatePerHour FROM Bike WHERE LocationID = p_Location...	0 rows

13. Create a Procedure to Get User Booking History

DELIMITER //

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

DELIMITER ;

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code for the procedure is displayed, starting with a DELIMITER //, followed by the CREATE PROCEDURE statement, and ending with a final DELIMITER ;. Below the code, the Action Output pane shows a history of database operations. The log includes several error messages (red X) related to triggers named prevent_double_booking, and successful statements (green checkmark) for creating the procedure and other objects like AddBike and GetAvailableBikesByLocation.

Action	Time	Output
CREATE PROCEDURE GetUserBookingHistory (12:13:54	CREATE PROCEDURE GetUserBookingHistory (
IN pUserID INT	18:14:05	IN pUserID INT
)	18:15:16)
BEGIN	18:15:23	BEGIN
SELECT B.BooklogID, B.StartTime, B.EndTime, B.TotalCost, B.Status	07:48:19	SELECT B.BooklogID, B.StartTime, B.EndTime, B.TotalCost, B.Status
FROM Booking B	07:54:30	FROM Booking B
WHERE B.UserID = pUserID	08:05:02	WHERE B.UserID = pUserID
ORDER BY B.StartTime DESC;	08:10:00	ORDER BY B.StartTime DESC;
END //	08:14:25	END //
DELIMITER ;	08:16:55	DELIMITER ;
Apply changes to new_table		Apply changes to new_table

14. Entering information

1. Location Table

The screenshot shows the MySQL Workbench interface with the Location table selected. The table has columns: LocationID, City, State, Street, Latitude, and Longitude. The data includes 10 rows of city information with their coordinates.

LocationID	City	State	Street	Latitude	Longitude
1	New York	NY	123 Broadway	40.712776	-74.005974
2	Los Angeles	CA	'456 Sunset Blvd'	34.0522350	-118.243683
3	Chicago	IL	789 Michigan Ave	41.8781140	-87.29798
4	Houston	TX	101 Main Street	29.760427	-95.369804
5	Miami	FL	'202 Ocean Drive'	25.761681	-80.191788
6	Seattle	WA	'303 Pine St'	47.606299	-122.332071
7	Boston	MA	'484 Beacon St'	42.3600881	-71.058884
8	San Francisco	CA	'505 Market St'	37.774929	-122.419418
9	Denver	CO	'606 Colfax Ave'	39.39326	-104.990251
10	Austin	TX	'707 Congress Ave'	30.267153	-97.743057

Action Output:

Time	Action	Response	Duration / Fetch Time
35 08:37:48	ALTER TABLE Location MODIFY COLUMN Longitude DECIMAL(11,8), MODIFY COLUMN Latitude DECIMAL(11,8)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.079 sec
36 08:38:45	INSERT INTO Location (City, State, Street, Latitude, Longitude) VALUES ('New York', 'NY', '123 Broadway', 40.712776, -74.005974), ('Los Angeles', 'CA', '456 Sunset Blvd', 34.0522350, -118.24368300)	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.0057 sec
37 08:39:02	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0017 sec / 0.0002...
38 08:40:57	Refresh Recordset	There are pending changes. Please commit or rollback.	
39 08:41:32	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0004 sec / 0.000...
40 08:42:26	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0012 sec / 0.00013...

2. User Table

The screenshot shows the MySQL Workbench interface with the User table selected. The table has columns: UserID, FirstName, LastName, PhoneNumber, and PasswordHash. The data includes 10 rows of user information.

UserID	FirstName	LastName	PhoneNumber	PasswordHash	CreatedAt
1	John	Doe	555-0101	\$2a\$10\$XpLj5z8hQNQ9E9Z5Jrcwve	2025-05-07 08:43:45
2	Jane	Smith	555-0202	\$2a\$10\$Yq2K6e9JR.P97BA3JUDxwe	2025-05-07 08:43:45
3	Robert	Johnson	555-0303	\$2a\$10\$zr3LtaR9aT.S.09G084jVeXe	2025-05-07 08:43:45
4	Emily	Williams	555-0404	\$2a\$10\$4a4NBb1LT.RH2C5jWfZze	2025-05-07 08:43:45
5	Michael	Brown	555-0505	\$2a\$10\$9t9q0d2mL.S2130\$jYyze	2025-05-07 08:43:45
6	Sarah	Davis	555-0606	\$2a\$10\$Cu409d3nV.T34k7jhixc	2025-05-07 08:43:45
7	David	Miller	555-0707	\$2a\$10\$Dv5P9e4w.UH5k18jZ1zye	2025-05-07 08:43:45
8	Jessica	Wilson	555-0808	\$2a\$10\$EW601T5Cx.VSL6H9Ajzze	2025-05-07 08:43:45
9	Thomas	Moore	555-0909	\$2a\$10\$Fx7R2gqY.Wd7N0Bk4ze	2025-05-07 08:43:45
10	Jennifer	Taylor	555-1010	\$2a\$10\$Gy853h7rZ.X7NB011cm5ze	2025-05-07 08:43:45

Action Output:

Time	Action	Response	Duration / Fetch Time
3/ 08:39:02	SELECT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0001 sec / 0.00002...
38 08:40:57	Refresh Recordset	There are pending changes. Please commit or rollback.	
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\$XpLj5z8hQNQ9E9Z5Jrcwve'), ('Jane', 'Smith', '555-0202', '\$2a\$10\$Yq2K6e9JR.P97BA3JUDxwe')	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.055 sec
42 08:44:08	SELECT * FROM User LIMIT 0, 1000	10 row(s) returned	0.0092 sec / 0.0005...

3. User_Email Table

```

30 • INSERT INTO User_Email (Email, UserID) VALUES
31   ('john.doe@example.com', 1),
32   ('jane.smith@example.com', 2),
33   ('robert.j@example.com', 3),
34   ('emily.w@example.com', 4),
35   ('michael.l@example.com', 5),
36   ('sarah.h@example.com', 6),
37   ('david.m@example.com', 7),
38   ('jessica.v@example.com', 8),
39   ('thomas.m@example.com', 9),
40   ('jennifer.t@example.com', 10);
41
42
43 • SELECT * FROM User_Email ;
44

```

1:44

Result Grid | Filter Rows: Q Search Export: Result Grid | Form Editor | Field Types | Query Stats | Read Only

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

User_Email 6

Action Output

Time	Action	Response	Duration / Fetch Time
39 08:41:32	INSERT * FROM Location LIMIT 0, 1000	10 row(s) returned	0.0004 sec / 0.000...
40 08:42:26	SELECT * FROM Location	10 row(s) returned	0.0012 sec / 0.0001...
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 Warnin...	0.055 sec
42 08:44:08	SELECT * FROM User LIMIT 0, 1000	10 row(s) returned	0.0092 sec / 0.0005...
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 Warnin...	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

```

44
45 • INSERT INTO Admin (PasswordHash) VALUES
46   ('$2a$10$Xp1J5z8hQNQ9E9Z5jTcwve'),
47   ('$2a$10$Yq2K6z9R.P7F8A3jUdXwe'),
48   ('$2a$10$Zt3L7a0k5.Q9G0B4jYeYxe'),
49   ('$2a$10$A4#M8b1lT.R1H2C5jWZze'),
50   ('$2a$10$B5N9c2nU.S21D6jXgY0e');
51
52 • SELECT * FROM Admin ;
53
54
55

```

1:53

Result Grid | Filter Rows: Q Search Edit: Export/Import: Result Grid | Form Editor | Field Types | Query Stats | Apply

AdminID	PasswordHash	CreatedAt
1	\$2a\$10\$Xp1J5z8hQNQ9E9Z5jTcwve	2025-05-07 08:48:15
2	\$2a\$10\$Yq2K6z9R.P7F8A3jUdXwe	2025-05-07 08:48:15
3	\$2a\$10\$Zt3L7a0k5.Q9G0B4jYeYxe	2025-05-07 08:48:15
4	\$2a\$10\$A4#M8b1lT.R1H2C5jWZze	2025-05-07 08:48:15
5	\$2a\$10\$B5N9c2nU.S21D6jXgY0e	2025-05-07 08:48:15
NULL	NULL	NULL

Admin 7

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 Warnin...	0.055 sec
42 08:44:08	SELECT * FROM User LIMIT 0, 1000	10 row(s) returned	0.0092 sec / 0.0005...
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 Warnin...	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\$Yq2K6z9R.P7F8A3jUdXwe'), ('\$2a\$10\$Zt3L7a0k5.Q9G0B4jYeYxe'), ('\$2a\$10\$A4#M8b1lT.R1H2C5jWZze'), ('\$2a\$10\$B5N9c2nU.S21D6jXgY0e')	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

```

54 • INSERT INTO Admin_Email (Email, AdminID) VALUES
55 ('admin1@bikerental.com', 1),
56 ('admin2@bikerental.com', 2),
57 ('admin3@bikerental.com', 3),
58 ('admin4@bikerental.com', 4),
59 ('admin5@bikerental.com', 5);
60
61 • SELECT * FROM Admin_Email
62
63
64
65

```

Result Grid | Filter Rows: Search Export:

Email	AdminID
admin1@bikerental.com	1
admin2@bikerental.com	2
admin3@bikerental.com	3
admin4@bikerental.com	4
admin5@bikerental.com	5

Admin_Email 8 | Only

Action Output |

Time	Action	Response	Duration / Fetch Time
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.o@example.com', 5)	10 rows(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.016 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\$xp1J5zBhQNQ9E9ZjTcwve'), ('\$2a\$10\$Yq2K6z9R.P7fF8A3 Udxwe'), ('\$2a\$10\$z3.7a0k5.Q9G0B4 VeYxe'), ('\$2a\$10\$As...')	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.027 sec
46 08:48:30	SELECT * FROM Admin LIMIT 0, 1000	5 row(s) returned	0.0094 sec / 0.0005...
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 Warnings: 0	0.025 sec
48 08:50:25	SELECT * FROM Admin_Email LIMIT 0, 1000	5 row(s) returned	0.010 sec / 0.0006...

6. Bike Table

```

63 • INSERT INTO Bike (Model, Brand, Status, LocationID, RatePerHour) VALUES
64 ('Roadster 500', 'Giant', 'Available', 1, 15.75),
65 ('Mountain Pro', 'Trek', 'Available', 2, 18.50),
66 ('City Cruiser', 'Schwinn', 'Rented', 3, 12.00),
67 ('Speed Demon', 'Specialized', 'Maintenance', 4, 20.00),
68 ('Urban Commuter', 'Cannondale', 'Available', 5, 14.25),
69 ('Trail Blazer', 'Scott', 'Available', 6, 16.75),
70 ('Beach Cruiser', 'Electra', 'Rented', 7, 11.50),
71 ('Hybrid X', 'Fuji', 'Available', 8, 17.25),
72 ('Gravel Grinder', 'Canyon', 'Available', 9, 19.00),
73 ('Folding Pro', 'Brompton', 'Maintenance', 10, 22.50),
74 ('E-Bike Plus', 'Rad Power', 'Available', 1, 25.00),
75 ('Kids Bike', 'Huffy', 'Available', 2, 8.50);
76
77 • SELECT * FROM Bike;
78
79
80

```

Result Grid | Filter Rows: Search Edit:

BikeID	Model	Brand	Status	LocationID	RatePerHour	CreatedAt
1	Roadster 500	Giant	Available	1	15.75	2025-05-07 08:51:37
2	Mountain Pro	Trek	Available	2	18.50	2025-05-07 08:51:37
3	City Cruiser	Schwinn	Rented	3	12.00	2025-05-07 08:51:37
4	Speed Demon	Specialized	Maintenance	4	20.00	2025-05-07 08:51:37
5	Urban Commuter	Cannondale	Available	5	14.25	2025-05-07 08:51:37
6	Trail Blazer	Scott	Available	6	16.75	2025-05-07 08:51:37
7	Beach Cruiser	Electra	Rented	7	11.50	2025-05-07 08:51:37
8	Hybrid X	Fuji	Available	8	17.25	2025-05-07 08:51:37
9	Gravel Grinder	Canyon	Available	9	19.00	2025-05-07 08:51:37
10	Folding Pro	Brompton	Maintenance	10	22.50	2025-05-07 08:51:37
11	E-Bike Plus	Rad Power	Available	1	25.00	2025-05-07 08:51:37
12	Kids Bike	Huffy	Available	2	8.50	2025-05-07 08:51:37
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Bike 9 | Apply

Action Output |

Time	Action	Response	Duration / Fetch Time
45 08:48:15	INSERT INTO Admin (PasswordHash) VALUES ('\$2a\$10\$xp1J5zBhQNQ9E9ZjTcwve'), ('\$2a\$10\$Yq2K6z9R.P7fF8A3 Udxwe'), ('\$2a\$10\$z3.7a0k5.Q9G0B4 VeYxe'), ('\$2a\$10\$As...')	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.027 sec
46 08:48:30	SELECT * FROM Admin LIMIT 0, 1000	5 row(s) returned	0.0094 sec / 0.0005...
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 Warnings: 0	0.025 sec
48 08:50:25	SELECT * FROM Admin_Email LIMIT 0, 1000	5 row(s) returned	0.010 sec / 0.0006...
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', 'Schwinn', 'Rented', 3, 12.00), ('Speed Demon', 'Specialized', 'Maintenance', 4, 20.00), ('Urban Commuter', 'Cannondale', 'Available', 5, 14.25), ('Trail Blazer', 'Scott', 'Available', 6, 16.75), ('Beach Cruiser', 'Electra', 'Rented', 7, 11.50), ('Hybrid X', 'Fuji', 'Available', 8, 17.25), ('Gravel Grinder', 'Canyon', 'Available', 9, 19.00), ('Folding Pro', 'Brompton', 'Maintenance', 10, 22.50), ('E-Bike Plus', 'Rad Power', 'Available', 1, 25.00), ('Kids Bike', 'Huffy', 'Available', 2, 8.50)	12 row(s) affected Records: 12 Duplicates: 0 Warnings: 0	0.017 sec
50 08:51:37	SELECT * FROM Bike LIMIT 0, 1000	12 row(s) returned	0.00099 sec / 0.000...

7. Booking Table

```

79 • INSERT INTO Booking (UserID, BikeID, StartTime, EndTime, TotalCost, Status) VALUES
80 (1, 3, '2023-06-01 10:00:00', '2023-06-01 12:30:00', 30.00, 'Completed'),
81 (2, 1, '2023-06-02 09:00:00', '2023-06-02 11:00:00', 31.50, 'Completed'),
82 (3, 2, '2023-06-03 14:00:00', '2023-06-03 17:00:00', 35.50, 'In Progress'),
83 (4, 5, '2023-06-04 08:00:00', '2023-06-04 10:00:00', 28.50, 'Upcoming'),
84 (5, 4, '2023-06-05 13:00:00', '2023-06-05 15:00:00', 40.00, 'Cancelled'),
85 (6, 6, '2023-06-06 10:00:00', '2023-06-06 14:00:00', 67.00, 'Completed'),
86 (7, 7, '2023-06-07 09:00:00', '2023-06-07 12:00:00', 34.50, 'Completed'),
87 (8, 8, '2023-06-08 13:00:00', '2023-06-08 16:00:00', 51.75, 'In Progress'),
88 (9, 9, '2023-06-09 08:00:00', '2023-06-09 11:00:00', 57.00, 'Upcoming'),
89 (10, 10, '2023-06-10 14:00:00', '2023-06-10 17:00:00', 67.50, 'Cancelled'),
90 (11, 11, '2023-06-11 10:00:00', '2023-06-11 13:00:00', 75.00, 'Completed'),
91 (12, 12, '2023-06-12 09:00:00', '2023-06-12 11:00:00', 17.00, 'Completed'),
92 (13, 1, '2023-06-13 14:00:00', '2023-06-13 18:00:00', 63.00, 'In Progress'),
93 (14, 2, '2023-06-14 08:00:00', '2023-06-14 10:00:00', 37.00, 'Upcoming'),
94 (15, 3, '2023-06-15 13:00:00', '2023-06-15 16:00:00', 36.00, 'Cancelled');
95
96 • SELECT * FROM Booking
97

```

⌚ 24:46

Result Grid | Filter Rows: Search Edit: Export/Import: Result Grid Form Editor Field Types Query Stats

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	35.50	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	1	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	11	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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Booking 10 | Action Output

```

Action Output
Time Action Response Duration / Fetch Time
47 08:53:52 INSERT INTO Admin_Email (Email, AdminID) VALUES ('admin@bikerental.com', 1), ('admin2@bikerental.com', 2), ('admin3@bikerental.com', 3), ('admin4@bikerental.com', 4), ('ad... b row(s) affected Records: 5 Duplicates: 0 Warnings: 0 0.02b sec
48 08:54:25 SELECT * FROM Admin_Email LIMIT 0, 1000 5 row(s) returned 0.010 sec / 0.00060...
49 08:54:37 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 Warnings: 0 0.017 sec
50 08:54:50 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 Warnings: 0 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

```

98
99 • INSERT INTO Manages (AdminID, BooklogID) VALUES
100 (1, 1), (2, 2), (3, 3), (4, 4), (5, 5),
101 (1, 6), (2, 7), (3, 8), (4, 9), (5, 10),
102 (1, 11), (2, 12), (3, 13), (4, 14), (5, 15);
103
104 • SELECT * FROM Manages
105
106
107

```

⌚ 1:10

Result Grid | Filter Rows: Search Edit: Export/Import: Result Grid Form Editor Field Types Query Stats

AdminID	BooklogID
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
NULL	NULL

Manages 11 | Action Output

```

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 Warnings: 0 0.01 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 Warnings: 0 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, 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 Warnings: 0 0.0078 sec
54 08:55:20 SELECT * FROM Manages LIMIT 0, 1000 15 row(s) returned 0.00081 sec / 0.000...

```

9. Payment Table

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window containing SQL queries for creating a table and inserting data. Below it is a results grid showing the inserted data. On the right side, there are various toolbars and a sidebar with icons for result grid, form editor, field types, and query stats.

```

106 • INSERT INTO Payment (BooklogID, UserID, Amount, PaymentMethod, PaymentStatus) VALUES
107   (1, 1, 30.00, 'Card', 'Completed'),
108   (2, 2, 31.50, 'Wallet', 'Completed'),
109   (3, 3, 55.50, 'Card', 'Processing'),
110   (4, 4, 28.50, 'Cash', 'Pending'),
111   (5, 5, 40.00, 'Card', 'Failed'),
112   (6, 6, 67.00, 'Card', 'Completed'),
113   (7, 7, 34.50, 'Wallet', 'Completed'),
114   (8, 8, 51.75, 'Card', 'Processing'),
115   (9, 9, 57.00, 'Cash', 'Pending'),
116   (10, 10, 67.50, 'Card', 'Failed'),
117   (11, 1, 75.00, 'Card', 'Completed'),
118   (12, 2, 17.00, 'Wallet', 'Completed'),
119   (13, 3, 63.00, 'Card', 'Processing'),
120   (14, 4, 37.00, 'Cash', 'Pending'),
121   (15, 5, 36.00, 'Card', 'Failed');
122
123 • SELECT * FROM Payment ;
124
  
```

Result Grid:

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	5	5	36.00	Card	Failed	2025-05-07 08:57:04
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Action Output:

Time	Action	Response	Duration / Fetch Time
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')	1 row(s) affected Records: 1b Duplicates: 0 Warn... 15 row(s) returned	0.008 sec 0.0012 sec / 0.00029...
52	08:54:02 SELECT * FROM Booking LIMIT 0, 1000	15 row(s) returned	0.0001 sec / 0.00000...
53	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 Warn... 15 row(s) returned	0.0078 sec 0.00081 sec / 0.00000...
54	08:55:20 SELECT * FROM Manages LIMIT 0, 1000	15 row(s) returned	0.0001 sec / 0.00000...
55	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... 56	15 row(s) affected Records: 15 Duplicates: 0 Warn... 15 row(s) returned	0.015 sec 0.0024 sec / 0.00066...
56	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';`

The screenshot shows the MySQL Workbench interface. It displays the execution of a stored procedure (CALL AddBike) and the resulting data in a results grid. A log of actions is also shown at the bottom.

```

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

Result Grid:

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
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Action Output:

Time	Action	Response	Duration / Fetch Time
53	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)	1b row(s) affected Records: 1b Duplicates: 0 Warn... 15 row(s) returned	0.0078 sec 0.00081 sec / 0.00000...
54	08:55:20 SELECT * FROM Manages LIMIT 0, 1000	15 row(s) affected Records: 15 Duplicates: 0 Warn... 15 row(s) returned	0.0001 sec / 0.00000...
55	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... 56	15 row(s) affected Records: 15 Duplicates: 0 Warn... 15 row(s) returned	0.015 sec 0.0024 sec / 0.00066...
56	08:57:21 SELECT * FROM Payment LIMIT 0, 1000	15 row(s) returned	0.0024 sec / 0.00066...
57	15:21:47 CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50)	1 row(s) affected	0.065 sec
58	15:21:59 SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek' LIMIT 0, 1000	2 row(s) returned	0.0097 sec / 0.00068...

Testing Error Cases :

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

Returns:

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

The screenshot shows the MySQL Workbench interface with the SQL editor and the log viewer. The log viewer displays the following entries:

Action	Time	Response	Duration / Fetch Time
SELECT * FROM Managers LIMIT 0, 1000	54 08:56:20	10 row(s) returned	0.00001 sec / 0.0000...
INSERT INTO Payment (BooklogID, UserID, Amount, PaymentMethod, PaymentStatus) VALUES (1, 1, 30.00, 'Card', 'Completed')	55 08:57:04	15 row(s) affected Records: 15 Duplicates: 0 Warnings: 0	0.015 sec
SELECT * FROM Payment LIMIT 0, 1000	56 08:57:21	15 row(s) returned	0.0024 sec / 0.0006...
CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50)	57 15:21:47	1 row(s) affected	0.065 sec
SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek' LIMIT 0, 1000	58 15:21:59	2 row(s) returned	0.0097 sec / 0.0058...
CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00)	59 15:26:01	Error Code: 1644. Invalid Status. Must be Available, Rented, or Maintenance.	0.032 sec

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:

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

- SQL Editor:** Contains the following SQL code:


```

1 CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50);
2 SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek';
3
4 CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00);
5
6 -- CANCEL BOOKING PROCEDURE
7
8 SELECT * FROM Booking WHERE BookLogID = 2;
      
```
- Result Grid:** Displays a single row of booking information:

BookLogID	UserID	BikeID	StartTime	EndTime	TotalCost	Status	CreatedAt
2	2	1	2023-06-02 09:00:00	2023-06-02 11:00:00	31.50	Completed	2025-05-07 08:53:52
- Action Output:** Shows the execution history with the following rows:

Time	Action	Response	Duration / Fetch Time
56 08:57:21	SELECT * FROM Payment LIMIT 0, 1000	15 row(s) returned	0.0024 sec / 0.00066...
57 15:21:47	CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50)	1 row(s) affected	0.065 sec
58 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...
59 15:25:01	CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00)	Error Code: 1644. Invalid Status. Must be Available,...	0.032 sec
60 15:38:08	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000	1 row(s) returned	0.043 sec / 0.00028...

After Calling the Procedure :

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

- SQL Editor:** Contains the following SQL code:


```

1 CALL AddBike('Mountain Pro', 'Trek', 'Available', 3, 18.50);
2 SELECT * FROM Bike WHERE Model = 'Mountain Pro' AND Brand = 'Trek';
3
4 CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00);
5
6 -- CANCEL BOOKING PROCEDURE
7
8 SELECT * FROM Booking WHERE BookLogID = 2;
9 CALL CancelBooking(2);
10
11
      
```
- Result Grid:** Displays a single row of booking information, identical to the previous state:

BookLogID	UserID	BikeID	StartTime	EndTime	TotalCost	Status	CreatedAt
2	2	1	2023-06-02 09:00:00	2023-06-02 11:00:00	31.50	Cancelled	2025-05-07 08:53:52
- Action Output:** Shows the execution history with the following rows:

Time	Action	Response	Duration / Fetch Time
59 15:25:01	CALL AddBike('Test Bike', 'Brand', 'InvalidStatus', 1, 15.00)	Error Code: 1644. Invalid Status. Must be Available,...	0.032 sec
60 15:38:08	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000	1 row(s) returned	0.043 sec / 0.00028...
61 15:38:43	CALL CancelBooking(2)	Error Code: 1265. Data truncated for column 'PaymentStatus' at row 1	0.036 sec
62 15:40:05	ALTER TABLE Payment MODIFY PaymentStatus VARCHAR(20)	15 row(s) affected	0.060 sec
63 15:40:14	CALL CancelBooking(2)	Error Code: 1644. Only confirmed or completed boo...	0.0014 sec
64 15:40:28	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000	1 row(s) returned	0.0033 sec / 0.0001...

3. Get Available Bikes at a location Procedure

Calling the Procedure :

CALL GetAvailableBikesByLocation(1);

The screenshot shows the MySQL Workbench interface. At the top, there is a command line with the following text:

```
10  
11 • CALL GetAvailableBikesByLocation(1);|
```

Below the command line is a result grid titled "Result Grid". The grid has four columns: BikeID, Model, Brand, and RatePerHour. The data is as follows:

BikeID	Model	Brand	RatePerHour
1	Roadster 500	Giant	15.75
11	E-Bike Plus	Rad Power	25.00

At the bottom of the result grid, it says "Result 4".

Below the result grid is an "Action Output" section. It contains a table with three columns: Action, Time, and Action. The actions listed are:

Action	Time	Action
60	15:38:08	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000
61	15:38:43	CALL CancelBooking(2)
62	15:40:05	ALTER TABLE Payment MODIFY PaymentStatus VARCHAR(20)
63	15:40:14	CALL CancelBooking(2)
64	15:40:28	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000
65	15:45:43	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:

The screenshot shows the MySQL Workbench interface. At the top, there is a SQL editor window with the following content:

```
12
13 • SELECT UserID, FirstName, LastName FROM User;
```

Below the editor is a Result Grid showing the following data:

UserID	FirstName	LastName
1	John	Doe
2	Jane	Smith
3	Robert	Johnson
4	Emily	Williams
5	Michael	Brown
6	Sarah	Davis
7	David	Miller
8	Jessica	Wilson
9	Thomas	Moore
10	Jennifer	Taylor

At the bottom of the interface, there is an Action Output section showing the history of actions taken:

Time	Action	Response	Duration / Fetch Time
61 15:49:43	CALL CancelBooking(2)	Error Code: 1265. Data truncated for column 'PaymentStatus' at row 1	0.0056 sec
62 15:49:05	ALTER TABLE Payment MODIFY PaymentStatus VARCHAR(20)	15 row(s) affected Records: 15 Duplicates: 0 Warnings: 0	0.0060 sec
63 15:49:14	CALL CancelBooking(2)	Error Code: 1644. Only confirmed or completed bookings	0.0014 sec
64 15:49:28	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000	1 row(s) returned	0.0033 sec / 0.00001...
65 15:49:43	CALL GetAvailableBikesByLocation()	2 row(s) returned	0.033 sec / 0.00022...
66 15:49:38	SELECT UserID, FirstName, LastName FROM User LIMIT 0, 1000	10 row(s) returned	0.045 sec / 0.00021...

User Booking History of UserID = 1 :

The screenshot shows the MySQL Workbench interface. At the top, there is a SQL editor window with the following content:

```
15 • CALL GetUserBookingHistory(1);
```

Below the editor is a Result Grid showing the following data:

BooklogID	StartTime	EndTime	TotalCost	Status
11	2023-06-11 10:00:00	2023-06-11 13:00:00	75.00	Completed
1	2023-06-01 10:00:00	2023-06-01 12:30:00	30.00	Completed

At the bottom of the interface, there is an Action Output section showing the history of actions taken:

Time	Action	Response	Duration / Fetch Time
62 15:49:05	ALTER TABLE Payment MODIFY PaymentStatus VARCHAR(20)	15 row(s) affected Records: 15 Duplicates: 0 Warnings: 0	0.0060 sec
63 15:49:14	CALL CancelBooking(2)	Error Code: 1644. Only confirmed or completed bookings	0.0014 sec
64 15:49:28	SELECT * FROM Booking WHERE BookLogID = 2 LIMIT 0, 1000	1 row(s) returned	0.0033 sec / 0.00001...
65 15:49:43	CALL GetAvailableBikesByLocation()	2 row(s) returned	0.033 sec / 0.00022...
66 15:49:38	SELECT UserID, FirstName, LastName FROM User LIMIT 0, 1000	10 row(s) returned	0.045 sec / 0.00021...
67 15:50:26	CALL GetUserBookingHistory(1)	2 row(s) returned	0.057 sec / 0.00032...

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.