Cab Booking System for Data Analysis

This article provides a step-by-step guide to understanding how a cab booking system efficiently stores, manages, and analyzes operational data. The project serves as a practical approach to learning fundamental database and backend concepts, including:

- Creating structured relational databases for cab operations.
- Inserting and managing customer, driver, cab, booking, and trip data.
- Running data analysis queries to reveal business insights.
- Measuring customer behavior, driver performance, and revenue trends.

By the end of this project, readers will understand how SQL is used to manage and analyze cab booking systems effectively.

Project Overview

This case study simulates a cab booking company's database system. The goal is to understand how structured data can be used to store, manage, and analyze real-world cab booking operations. You'll create database tables, insert meaningful records, and write SQL queries to generate insights related to customer behavior, driver performance, and trip trends.

Table of Contents:

Let's start building !

Step 1: Setting Up the Database Schema

Create Database

```
CREATE DATABASE cb;
USE cb;
```

Customers Table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    RegistrationDate DATE
);
```

Drivers Table

```
CREATE TABLE Drivers (
    DriverID INT PRIMARY KEY,
    Name VARCHAR(100),
    JoinDate DATE
);
```

Cabs Table

```
CREATE TABLE Cabs (
    CabID INT PRIMARY KEY,
    DriverID INT,
    VehicleType VARCHAR(20),
    PlateNumber VARCHAR(20),
    FOREIGN KEY (DriverID) REFERENCES Drivers(DriverID)
);
```

Bookings Table

```sql
CREATE TABLE Bookings (
    BookingID INT PRIMARY KEY,
    CustomerID INT,
    CabID INT,
    BookingDate DATETIME,
    Status VARCHAR(20),
    PickupLocation VARCHAR(100),
    DropoffLocation VARCHAR(100),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (CabID) REFERENCES Cabs(CabID)
);
```

TripDetails Table

```sql
CREATE TABLE TripDetails (
    TripID INT PRIMARY KEY,
    BookingID INT,
    StartTime DATETIME,
    EndTime DATETIME,
    DistanceKM FLOAT,
    Fare FLOAT,
    FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)
);
```

Feedback Table

```sql
CREATE TABLE Feedback (
    FeedbackID INT PRIMARY KEY,
    BookingID INT,
    Rating FLOAT,
    Comments TEXT,
    FeedbackDate DATE,
    FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)
);
```

Step 2: Data Insertion

After setting up the schema, we now populate each table with initial sample data. This step ensures that we have enough variety to run real-time analytics and answer business queries later in the project.

Insert Customers:

```
INSERT INTO Customers (CustomerID, Name, Email, RegistrationDate) VALUES
(1, 'Alice Johnson', 'alice@example.com', '2023-01-15'),
(2, 'Bob Smith', 'bob@example.com', '2023-02-20'),
(3, 'Charlie Brown', 'charlie@example.com', '2023-03-05'),
(4, 'Diana Prince', 'diana@example.com', '2023-04-10');
```

Insert Drivers:

```
INSERT INTO Drivers (DriverID, Name, JoinDate) VALUES
(101, 'John Driver', '2022-05-10'),
(102, 'Linda Miles', '2022-07-25'),
(103, 'Kevin Road', '2023-01-01'),
(104, 'Sandra Swift', '2022-11-11');
```

Insert Cabs:

```
INSERT INTO Cabs (CabID, DriverID, VehicleType, PlateNumber) VALUES
(1001, 101, 'Sedan', 'ABC1234'),
(1002, 102, 'SUV', 'XYZ5678'),
(1003, 103, 'Sedan', 'LMN8901'),
(1004, 104, 'SUV', 'PQR3456');
```

Insert Bookings:

```
INSERT INTO Bookings (BookingID, CustomerID, CabID, BookingDate,
Status, PickupLocation, DropoffLocation) VALUES
```

```
(201, 1, 1001, '2024-10-01 08:30:00', 'Completed', 'Downtown',
'Airport'),
(202, 2, 1002, '2024-10-02 09:00:00', 'Completed', 'Mall',
'University'),
(203, 3, 1003, '2024-10-03 10:15:00', 'Canceled', 'Station',
'Downtown'),
(204, 4, 1004, '2024-10-04 14:00:00', 'Completed', 'Suburbs',
'Downtown'),
(205, 1, 1002, '2024-10-05 18:45:00', 'Completed', 'Downtown',
'Airport'),
(206, 2, 1001, '2024-10-06 07:20:00', 'Canceled', 'University',
'Mall');
```

Insert Trip Details:

```
INSERT INTO TripDetails (TripID, BookingID, StartTime, EndTime,
DistanceKM, Fare) VALUES
(301, 201, '2024-10-01 08:45:00', '2024-10-01 09:20:00', 18.5,
250.00),
(302, 202, '2024-10-02 09:10:00', '2024-10-02 09:40:00', 12.0,
180.00),
(303, 204, '2024-10-04 14:10:00', '2024-10-04 14:40:00', 10.0,
150.00),
(304, 205, '2024-10-05 18:50:00', '2024-10-05 19:30:00', 20.0,
270.00);
```

Insert Feedback:

```
INSERT INTO Feedback (FeedbackID, BookingID, Rating, Comments,
FeedbackDate) VALUES
(401, 201, 4.5, 'Smooth ride', '2024-10-01'),
(402, 202, 3.0, 'Driver was late', '2024-10-02'),
(403, 204, 5.0, 'Excellent service', '2024-10-04'),
(404, 205, 2.5, 'Cab was not clean', '2024-10-05');
```

Step 3 : SQL Queries to Explore Insights

Now that the database is set up with records, we'll begin answering real-world questions through SQL queries. These queries will help:

- Monitor operational efficiency
- Understand customer behavior
- Track driver performance
- Measure revenue and route performance

The next section includes SQL queries grouped by business goal along with explanations:

- Customer & Booking Analysis
- Driver Performance
- Revenue Trends
- Operational Efficiency
- Predictive & Comparative Metrics

Step 4 : Business Analysis & Use Cases

1. Customer and Booking Analysis

```
SELECT c.CustomerID, c.Name, COUNT(*) AS CompletedBookings
FROM Customers c
JOIN Bookings b ON c.CustomerID = b.CustomerID
WHERE b.Status = 'Completed'
GROUP BY c.CustomerID, c.Name
ORDER BY CompletedBookings DESC
```

Insight: Helps identify loyal, engaged customers who complete bookings regularly.Use the SQL questions listed above to analyze:

2. Customers with More Than 30% Cancellations

```
SELECT CustomerID,
       SUM(CASE WHEN Status = 'Canceled' THEN 1 ELSE 0 END) AS
Cancelled,
       COUNT(*) AS Total,
       ROUND(100.0 * SUM(CASE WHEN Status = 'Canceled' THEN 1 ELSE 0
END) / COUNT(*), 2) AS CancellationRate
FROM Bookings
GROUP BY CustomerID
HAVING CancellationRate > 30;
```

Insight: Identifies customers with a high cancellation rate. These might be users with erratic plans or bad app experience.

3. Busiest Day of the Week

```
SELECT DATE_FORMAT(BookingDate, "%W") AS DayOfWeek, COUNT(*) AS
TotalBookings
FROM Bookings
GROUP BY DATE_FORMAT(BookingDate, "%W")
ORDER BY TotalBookings DESC;
```

Insight: Reveals demand trends for resource and marketing planning.

4. Drivers with Average Rating < 3 in Last 3 Months

```
SELECT d.DriverID, d.Name, AVG(f.Rating) AS AvgRating
FROM Drivers d
JOIN Cabs c ON d.DriverID = c.DriverID
JOIN Bookings b ON c.CabID = b.CabID
JOIN Feedback f ON b.BookingID = f.BookingID
WHERE f.Rating IS NOT NULL AND f.FeedbackDate >= DATEADD(MONTH,
-3, GETDATE())
GROUP BY d.DriverID, d.Name
HAVING AVG(f.Rating) < 3.0;
```

Insight: Spot underperforming drivers who might need training or action.

5. Top 5 Drivers by Total Distance Covered

```
    SELECT d.DriverID, d.Name, AVG(f.Rating) AS AvgRating
    FROM Drivers d
    JOIN Cabs c ON d.DriverID = c.DriverID
    JOIN Bookings b ON c.CabID = b.CabID
    JOIN Feedback f ON b.BookingID = f.BookingID
    WHERE f.Rating IS NOT NULL AND f.FeedbackDate >= DATEADD(MONTH,
    -3, GETDATE())
    GROUP BY d.DriverID, d.Name
    HAVING AVG(f.Rating) < 3.0;
```

Insight: Identify highly active drivers and reward them.

6. Drivers with High Cancellation Rate (>25%)

```
SELECT d.DriverID, d.Name, SUM(t.DistanceKM) AS TotalDistance
FROM Drivers d
JOIN Cabs c ON d.DriverID = c.DriverID
JOIN Bookings b ON c.CabID = b.CabID
JOIN TripDetails t ON b.BookingID = t.BookingID
WHERE b.Status = 'Completed'
GROUP BY d.DriverID, d.Name
ORDER BY TotalDistance DESC
LIMIT 5;
```

Insight: Recognize driver behavior issues early and improve service.

7. Monthly Revenue in the Last 6 Months

```
    SELECT d.DriverID, d.Name,
           SUM(CASE WHEN b.Status = 'Canceled' THEN 1 ELSE 0 END) *
    100.0 / COUNT(*) AS CancellationRate
    FROM Drivers d
    JOIN Cabs c ON d.DriverID = c.DriverID
```

```
    JOIN Bookings b ON c.CabID = b.CabID
    GROUP BY d.DriverID, d.Name
    HAVING CancellationRate > 25;
```

Insight: Observe monthly income trends for financial forecasting.

8. Top 3 Routes by Booking Volume

```
    SELECT MONTH(t.EndTime) AS Month, SUM(t.Fare) AS Revenue
    FROM TripDetails t
    JOIN Bookings b ON t.BookingID = b.BookingID
    WHERE b.Status = 'Completed' AND t.EndTime >= DATEADD(MONTH,
    -6, GETDATE())
    GROUP BY MONTH(t.EndTime)
    ORDER BY Month;
```

Insight: Popular routes help optimize pricing and fleet assignment.

9. Driver Ratings vs Earnings

```
SELECT d.DriverID, AVG(f.Rating) AS AvgRating, COUNT(*) AS
TotalTrips, SUM(t.Fare) AS TotalEarnings
FROM Drivers d
JOIN Cabs c ON d.DriverID = c.DriverID
JOIN Bookings b ON c.CabID = b.CabID
JOIN Feedback f ON b.BookingID = f.BookingID
JOIN TripDetails t ON b.BookingID = t.BookingID
GROUP BY d.DriverID
ORDER BY AvgRating DESC;
```

Insight: Analyze the relationship between earnings and customer satisfaction.

10. Average Wait Time Per Pickup Location

```
    SELECT PickupLocation,
           AVG(DATEDIFF(MINUTE, b.BookingDate, t.StartTime)) AS
    AvgWaitTimeMins
    FROM Bookings b
    JOIN TripDetails t ON b.BookingID = t.BookingID
    WHERE b.Status = 'Completed'
    GROUP BY PickupLocation
    ORDER BY AvgWaitTimeMins DESC;
```

Insight: Identify bottlenecks and long wait times by location.

11. Common Cancellation Reasons

```
SELECT PickupLocation,
       AVG(DATEDIFF(MINUTE, b.BookingDate, t.StartTime)) AS
AvgWaitTimeMins
FROM Bookings b
JOIN TripDetails t ON b.BookingID = t.BookingID
WHERE b.Status = 'Completed'
GROUP BY PickupLocation
ORDER BY AvgWaitTimeMins DESC;
```

Insight: Directly learn why users cancel most often.

12. Revenue by Trip Type (Short vs Long)

```
    SELECT
        CASE
            WHEN DistanceKM < 5 THEN 'Short'
            ELSE 'Long'
        END AS TripType,
```

```
    COUNT(*) AS NumTrips,
    SUM(Fare) AS TotalRevenue
FROM TripDetails
GROUP BY
    CASE
        WHEN DistanceKM < 5 THEN 'Short'
        ELSE 'Long'
    END;
```

Insight: Compare how much short and long trips contribute to business.

13. Sedan vs SUV Revenue Comparison

```
SELECT
    CASE
        WHEN DistanceKM < 5 THEN 'Short'
        ELSE 'Long'
    END AS TripType,
    COUNT(*) AS NumTrips,
    SUM(Fare) AS TotalRevenue
FROM TripDetails
GROUP BY
    CASE
        WHEN DistanceKM < 5 THEN 'Short'
        ELSE 'Long'
    END;
```

Insight: Optimize fleet planning based on vehicle profitability.

14. Predict Customer Churn

```
SELECT c.VehicleType, SUM(t.Fare) AS Revenue
FROM Cabs c
JOIN Bookings b ON c.CabID = b.CabID
```

```
    JOIN TripDetails t ON b.BookingID = t.BookingID
    WHERE b.Status = 'Completed'
    GROUP BY c.VehicleType;
```

Insight: Identify at-risk customers for re-engagement campaigns.

15. Weekend vs Weekday Performance

```
    SELECT
        CASE
            WHEN DATENAME(WEEKDAY, BookingDate) IN ('Saturday',
    'Sunday') THEN 'Weekend'
            ELSE 'Weekday'
        END AS DayType,
        COUNT(*) AS TotalBookings,
        SUM(t.Fare) AS TotalRevenue
    FROM Bookings b
    JOIN TripDetails t ON b.BookingID = t.BookingID
    GROUP BY
        CASE
            WHEN DATENAME(WEEKDAY, BookingDate) IN ('Saturday',
    'Sunday') THEN 'Weekend'
            ELSE 'Weekday'
        END;
```

Insight: Understand day-of-week trends for promotional planning.

Step 6 : Conclusion

This project demonstrated the power of SQL in transforming cab booking operations data into actionable business insights. From customer behavior and driver performance to revenue trends and operational efficiency, each query revealed patterns that help optimize services, improve user satisfaction, and guide data-driven decision-making. Well-structured SQL logic can empower strategic planning and improve real-world business outcomes.