**SVKM’s NMIMS**

**School of Technology Management & Engineering**

A.Y. 2023 - 24

**Course: Database Management Systems**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | Btech CE | |
| Semester | 4th | |
| Name of the Project: | Travel Booking System | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| 01 | T007 | Arin Bhasin |
| 01 | T008 | Arjav Kasliwal |
| 01 | T017 | Chhavi Ambor |
| Date of Submission: 20th March 2024 | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| T017 | Chhavi Ambor |  |
| T008 | Arjav Kasliwal |  |

**Github link of your project:**

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

**Rubrics for the Project evaluation:**

|  |  |
| --- | --- |
| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 10 marks |
| Final phase of evaluation  Implementation, presentation and viva, Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**TRAVEL BOOKING SYSTEM**

**by**

**CHHAVI AMBOR, Roll number: T017**

**ARJAV KASLIWAL, Roll number: T008**

**ARIN BHASIN, Roll number: T007**

**Course: DBMS**

**AY: 2023-24**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Storyline |  |
| **2** | Components of Database Design |  |
| **3** | Entity Relationship Diagram |  |
| **4** | Relational Model |  |
| **5** | Normalization |  |
| **6** | SQL Queries |  |
| **7** | Learning from the Project |  |
| **8** | Project Demonstration |  |
| **9** | Self-learning beyond classroom |  |
| **10** | Learning from the project |  |
| **8** | Challenges faced |  |
| **9** | Conclusion |  |

**I. Storyline**

In the dynamic landscape of travel, organizing and coordinating various aspects of a vacation can be a daunting task. The existing manual methods of planning flights, accommodations, and car rentals lack efficiency and fail to provide a seamless user experience. To address this challenge, there is a need for an integrated travel booking system that leverages a well-designed SQL database. The goal is to create a centralized platform capable of managing customer information, flights, hotels, and car rentals, ensuring a user-friendly experience for travel enthusiasts. This project aims to develop a database solution that optimizes the booking process, enhances data accuracy, and provides users with a comprehensive and efficient tool for planning and booking their dream vacations.

**II. Components of Database Design**

**Entities and Attributes:**

**1. Customers:**

- Attributes:

- CustomerID (Primary Key)

- FirstName

- LastName

- Email

- Phone

- Address

**2. Flights:**

- Attributes:

- FlightID (Primary Key)

- Airline

- DepartureCity

- ArrivalCity

- DepartureDateTime

- ArrivalDateTime

- Price

**3. Hotels:**

- Attributes:

- HotelID (Primary Key)

- HotelName

- City

- CheckInDate

- CheckOutDate

- PricePerNight

**4. CarRentals:**

- Attributes:

- RentalID (Primary Key)

- RentalAgency

- CarModel

- PickupLocation

- ReturnLocation

- PickupDateTime

- ReturnDateTime

- PricePerDay

**5. Bookings:**

- Attributes:

- BookingID (Primary Key)

- CustomerID (Foreign Key referencing Customers)

- FlightID (Foreign Key referencing Flights)

- HotelID (Foreign Key referencing Hotels)

- RentalID (Foreign Key referencing CarRentals)

- TotalPrice

- BookingDateTime

**Relationships:**

**1. Customers to Bookings:**

- Relationship: One-to-Many

- Participation: Mandatory for Customers, Optional for Bookings

**2. Flights to Bookings:**

- Relationship: One-to-Many

- Participation: Optional for Flights, Mandatory for Bookings

**3. Hotels to Bookings:**

- Relationship: One-to-Many

- Participation: Optional for Hotels, Mandatory for Bookings

**4. CarRentals to Bookings:**

- Relationship: One-to-Many

- Participation: Optional for CarRentals, Mandatory for Bookings

**5. Customers to Flights, Hotels, and CarRentals:**

- Relationship: Many-to-Many

- Participation: Optional for Customers, Optional for Flights, Hotels, and CarRentals (through intermediate tables)

**Intermediate Tables for Many-to-Many Relationships:**

**1. CustomerFlights:**

- Attributes:

- CustomerID (Foreign Key referencing Customers)

- FlightID (Foreign Key referencing Flights)

**2. CustomerHotels:**

- Attributes:

- CustomerID (Foreign Key referencing Customers)

- HotelID (Foreign Key referencing Hotels)

**3. CustomerCarRentals:**

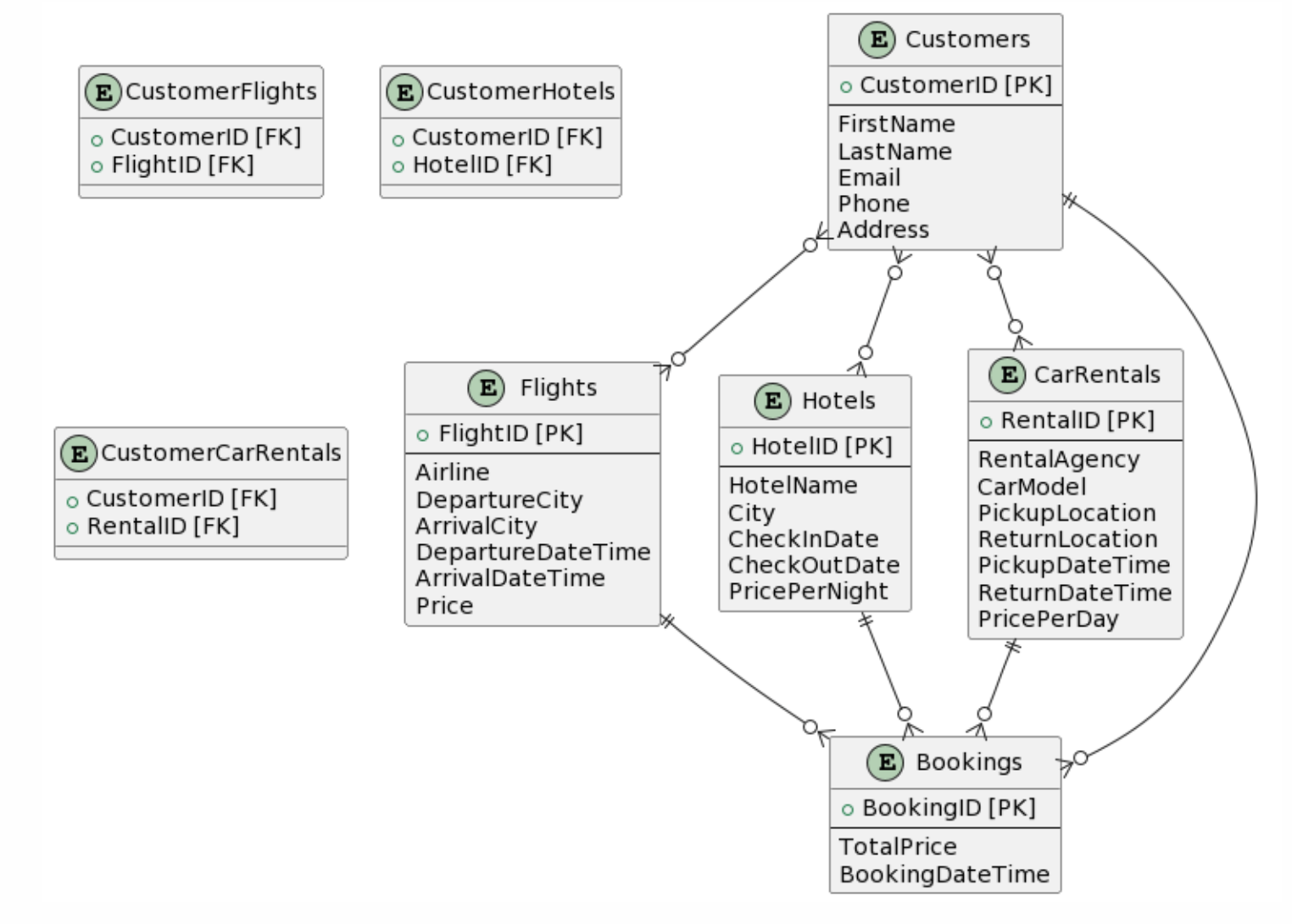
- Attributes:

- CustomerID (Foreign Key referencing Customers)

- RentalID (Foreign Key referencing CarRentals)

**III. Entity Relationship Diagram**

Draw the ER diagram here.



**IV. Relational Model**

Relational Model:

* **Customers**:
  + CustomerID (PK)
  + FirstName
  + LastName
  + Email
  + Phone
  + Address
* **Flights**:
  + FlightID (PK)
  + Airline
  + DepartureCity
  + ArrivalCity
  + DepartureDateTime
  + ArrivalDateTime
  + Price
* **Hotels**:
  + HotelID (PK)
  + HotelName
  + City
  + CheckInDate
  + CheckOutDate
  + PricePerNight
* **CarRentals**:
  + RentalID (PK)
  + RentalAgency
  + CarModel
  + PickupLocation
  + ReturnLocation
  + PickupDateTime
  + ReturnDateTime
  + PricePerDay
* **Bookings**:
  + BookingID (PK)
  + CustomerID (FK referencing Customers)
  + FlightID (FK referencing Flights)
  + HotelID (FK referencing Hotels)
  + RentalID (FK referencing CarRentals)
  + TotalPrice
  + BookingDateTime

Intermediate Tables:

* **CustomerFlights**:
  + CustomerID (FK referencing Customers)
  + FlightID (FK referencing Flights)
* **CustomerHotels**:
  + CustomerID (FK referencing Customers)
  + HotelID (FK referencing Hotels)
* **CustomerCarRentals**:
  + CustomerID (FK referencing Customers)
  + RentalID (FK referencing CarRentals)

**V. Normalization**

1. Customers Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: Since there's only one candidate key (CustomerID), and all non-key attributes depend fully on the entire key, it automatically satisfies 2NF.
   * 3NF: There are no transitive dependencies in the table, as each non-key attribute depends only on the primary key.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.
2. Flights Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: There's only one candidate key (FlightID), and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.
3. Hotels Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: There's only one candidate key (HotelID), and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.
4. CarRentals Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: There's only one candidate key (RentalID), and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.
5. Bookings Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: There's only one candidate key (BookingID), and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.

Intermediate Tables:

6. CustomerFlights Table:

* 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
* 2NF: The composite primary key (CustomerID, FlightID) is the only candidate key, and all non-key attributes depend fully on the entire key.
* 3NF: There are no transitive dependencies in the table.
* BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.

1. CustomerHotels Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: The composite primary key (CustomerID, HotelID) is the only candidate key, and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.
2. CustomerCarRentals Table:
   * 1NF: The table has no repeating groups or arrays, and each attribute value is atomic.
   * 2NF: The composite primary key (CustomerID, RentalID) is the only candidate key, and all non-key attributes depend fully on the entire key.
   * 3NF: There are no transitive dependencies in the table.
   * BCNF: Since the table has only one candidate key, it trivially satisfies BCNF.

**VI. SQL Queries**

CODE: (showing all tables)

SELECT \* FROM Customers;

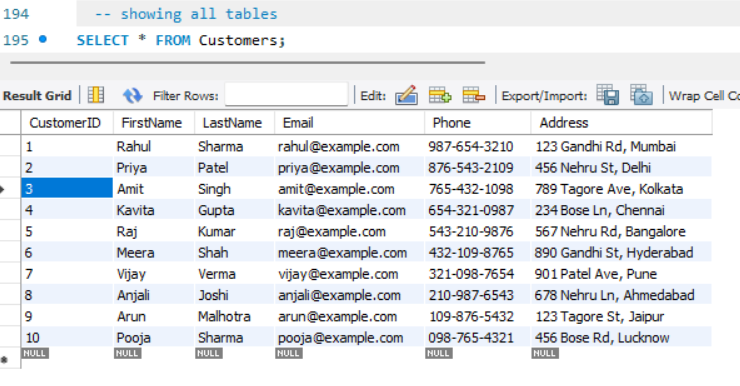
SELECT \* FROM Flights;

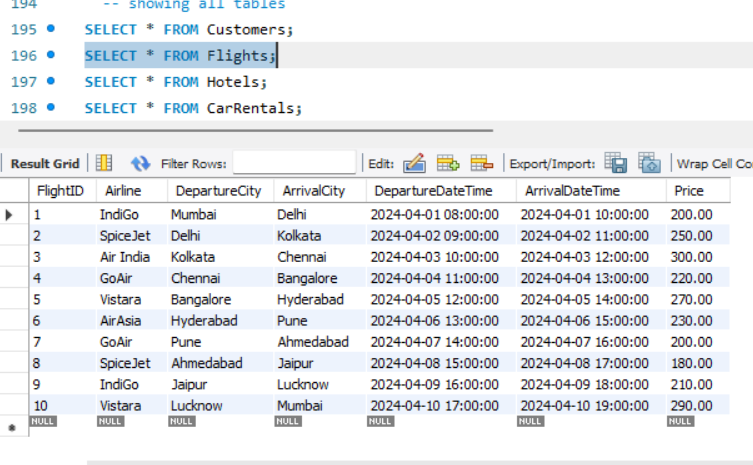
SELECT \* FROM Hotels;

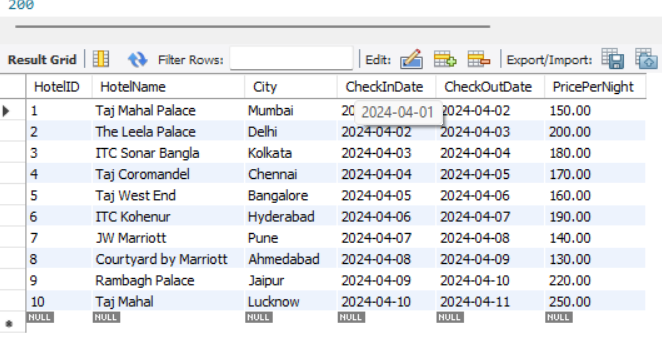
SELECT \* FROM CarRentals;

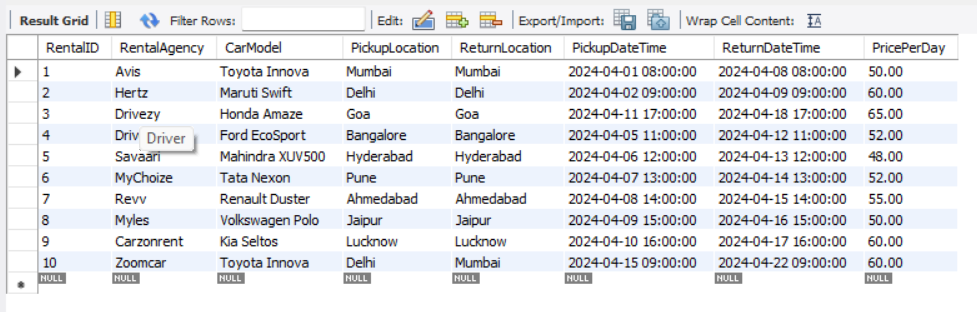
SELECT \* FROM Bookings;

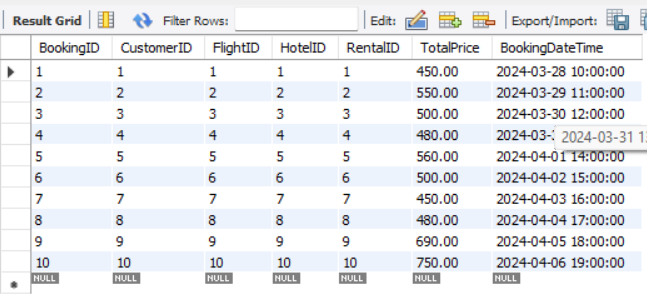
OUTPUT:











**QUERIES:**

CODE:

-- 1.Retrieve the customers who booked a flight but not a hotel:

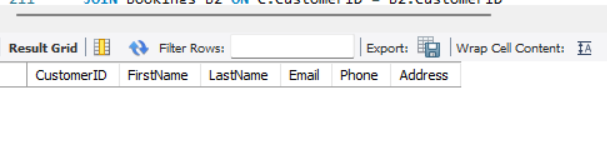
SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B ON C.CustomerID = B.CustomerID

WHERE B.FlightID IS NOT NULL AND B.HotelID IS NULL;

OUTPUT:



CODE:

-- 2.Retrieve the customers who booked both a flight and a hotel:

SELECT DISTINCT C.\*

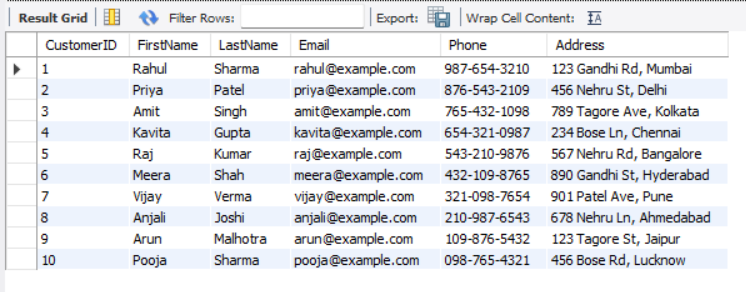
FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

WHERE B1.FlightID IS NOT NULL AND B2.HotelID IS NOT NULL;

OUTPUT:



CODE:

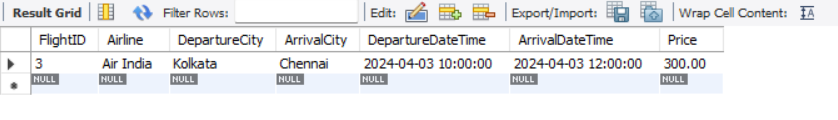
-- 3.Retrieve the flights with the highest price:

SELECT \*

FROM Flights

WHERE Price = (SELECT MAX(Price) FROM Flights);

OUTPUT:



CODE:

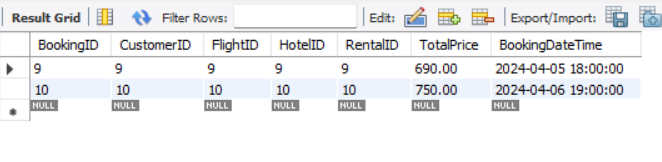
-- 4.Retrieve the bookings made after '2024-04-05':

SELECT \*

FROM Bookings

WHERE BookingDateTime > '2024-04-05';

OUTPUT:



CODE:

-- 5.Retrieve the customers who booked more than one flight:

SELECT C.CustomerID, C.FirstName, C.LastName, COUNT(\*) AS NumFlightsBooked

FROM Customers C

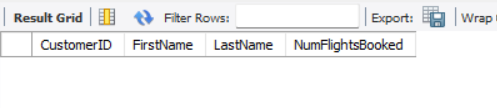
JOIN Bookings B ON C.CustomerID = B.CustomerID

WHERE B.FlightID IS NOT NULL

GROUP BY C.CustomerID

HAVING COUNT(\*) > 1;

OUTPUT:



CODE:

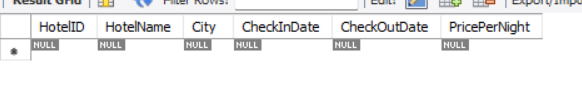
-- 6.Retrieve the hotels with no bookings:

SELECT \*

FROM Hotels

WHERE HotelID NOT IN (SELECT DISTINCT HotelID FROM Bookings WHERE HotelID IS NOT NULL);

OUTPUT:



CODE:

-- 7.Retrieve the customers who booked a flight to 'Mumbai' and a hotel in 'Mumbai':

SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

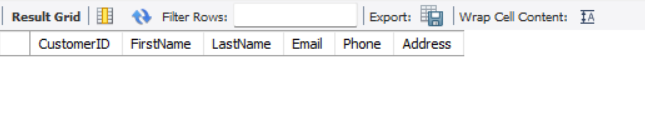
JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

JOIN Flights F ON B1.FlightID = F.FlightID

JOIN Hotels H ON B2.HotelID = H.HotelID

WHERE F.ArrivalCity = 'Mumbai' AND H.City = 'Mumbai';

OUTPUT:



CODE:

-- 8.Retrieve the customers who booked a car rental for more than 5 days:

SELECT DISTINCT C.\*

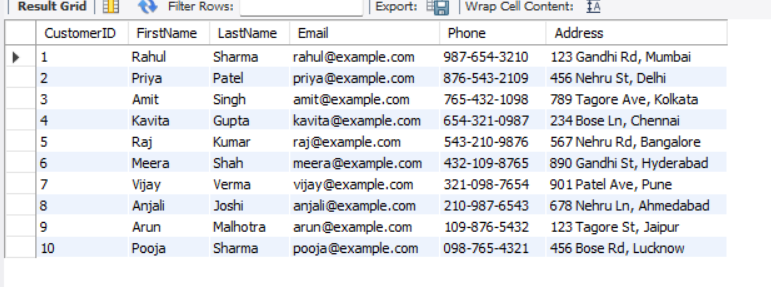
FROM Customers C

JOIN Bookings B ON C.CustomerID = B.CustomerID

JOIN CarRentals CR ON B.RentalID = CR.RentalID

WHERE DATEDIFF(CR.ReturnDateTime, CR.PickupDateTime) > 5;

OUTPUT:



CODE:

-- 9.Retrieve the customers who booked both a flight and a car rental but not a hotel:

SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

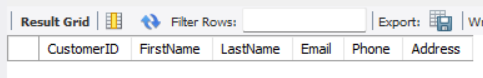
JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

JOIN Flights F ON B1.FlightID = F.FlightID

JOIN CarRentals CR ON B2.RentalID = CR.RentalID

WHERE B1.FlightID IS NOT NULL AND B2.RentalID IS NOT NULL AND B1.HotelID IS NULL;

OUTPUT:



CODE:

-- 10.Retrieve the customers who booked a hotel in 'Delhi' or a car rental from 'Mumbai':

SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

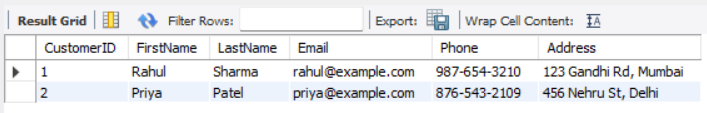
JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

JOIN Hotels H ON B1.HotelID = H.HotelID

JOIN CarRentals CR ON B2.RentalID = CR.RentalID

WHERE H.City = 'Delhi' OR CR.PickupLocation = 'Mumbai';

OUTPUT:



CODE:

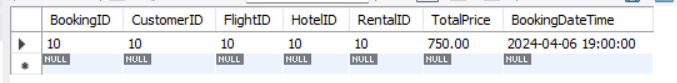
-- 11.Retrieve the bookings with a total price greater than $500 and made by customers whose last name starts with 'S':

SELECT \*

FROM Bookings

WHERE TotalPrice > 500 AND CustomerID IN (SELECT CustomerID FROM Customers WHERE LastName LIKE 'S%');

OUTPUT:



CODE:

-- 12.Retrieve the customers who booked a flight and a hotel in the same city:

SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

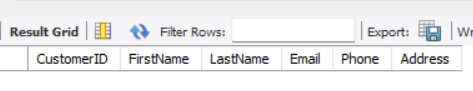
JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

JOIN Flights F ON B1.FlightID = F.FlightID

JOIN Hotels H ON B2.HotelID = H.HotelID

WHERE F.ArrivalCity = H.City;

OUTPUT:



CODE:

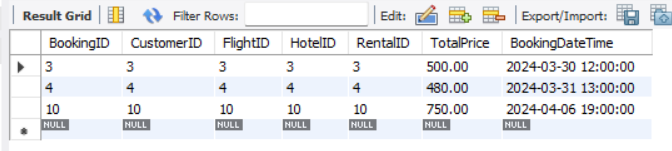
-- 13.Retrieve the bookings made on weekends (Saturday or Sunday):

SELECT \*

FROM Bookings

WHERE DAYOFWEEK(BookingDateTime) IN (1, 7);

OUTPUT:



CODE:

-- 14.Retrieve the flights with the shortest duration (in hours):

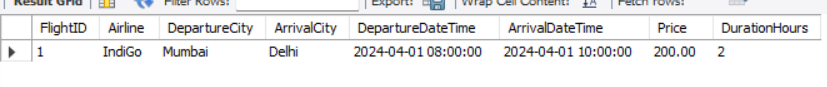
SELECT \*, TIMESTAMPDIFF(HOUR, DepartureDateTime, ArrivalDateTime) AS DurationHours

FROM Flights

ORDER BY DurationHours ASC

LIMIT 1;

OUTPUT:



CODE:

-- 15.Retrieve the customers who booked a flight to a city with a name longer than 6 characters:

SELECT DISTINCT C.\*

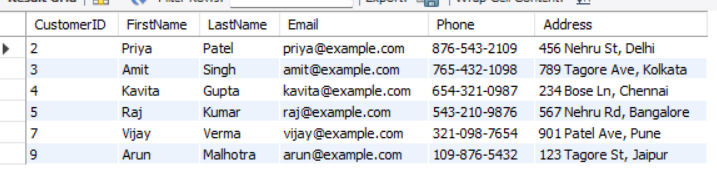
FROM Customers C

JOIN Bookings B ON C.CustomerID = B.CustomerID

JOIN Flights F ON B.FlightID = F.FlightID

WHERE LENGTH(F.ArrivalCity) > 6;

OUTPUT:



CODE:

-- 16.Retrieve the bookings with a total price greater than the average total price of bookings for each customer:

SELECT \*

FROM Bookings

WHERE TotalPrice > (

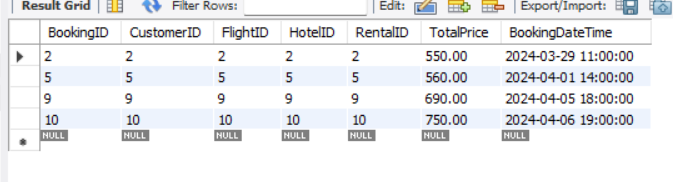
SELECT AVG(TotalPrice)

FROM Bookings

WHERE CustomerID = Bookings.CustomerID

);

OUTPUT:



CODE:

-- 17.Retrieve the customers who booked flights on weekdays (Monday to Friday):

SELECT DISTINCT C.\*

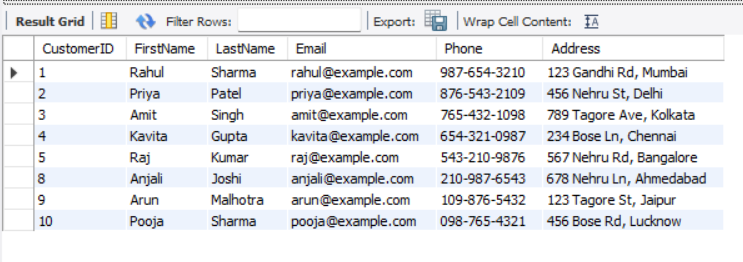
FROM Customers C

JOIN Bookings B ON C.CustomerID = B.CustomerID

JOIN Flights F ON B.FlightID = F.FlightID

WHERE DAYOFWEEK(F.DepartureDateTime) BETWEEN 2 AND 6;

OUTPUT:



CODE:

-- 18.Retrieve all customers whose CustomerID appears in the Bookings table, indicating that they have made bookings.

SELECT \*

FROM Customers

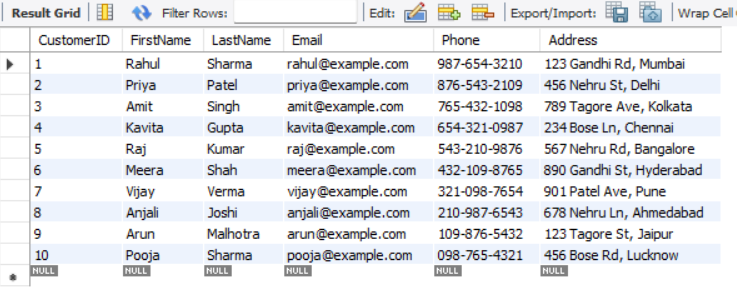
WHERE CustomerID IN (

SELECT CustomerID

FROM Bookings

);

OUTPUT:



CODE:

-- 19.Retrieve the hotels with an average price per night greater than $200:

SELECT \*

FROM Hotels

WHERE HotelID IN (

SELECT HotelID

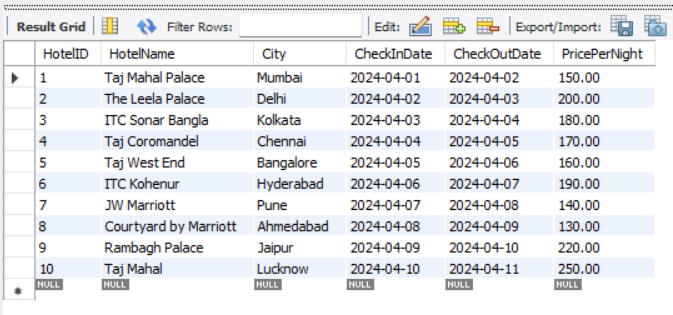
FROM Bookings

GROUP BY HotelID

HAVING AVG(TotalPrice / DATEDIFF(CheckOutDate, CheckInDate)) > 200

);

OUTPUT:



CODE:

-- 20.Retrieve the customers who booked both a flight and a hotel, with the hotel check-in date after the flight departure date:

SELECT DISTINCT C.\*

FROM Customers C

JOIN Bookings B1 ON C.CustomerID = B1.CustomerID

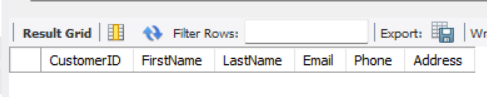
JOIN Bookings B2 ON C.CustomerID = B2.CustomerID

JOIN Flights F ON B1.FlightID = F.FlightID

JOIN Hotels H ON B2.HotelID = H.HotelID

WHERE H.CheckInDate > F.DepartureDateTime;

OUTPUT:



**VI. Project demonstration**

**MySQL Workbench:** MySQL Workbench is a visual database design tool and administration tool for MySQL databases. It provides a graphical interface for designing database schemas, creating and managing database tables, writing and executing SQL queries, and more.

**VII. Self -Learning beyond classroom**

* **Advanced SQL Queries:** Crafting complex SQL queries to meet specific requirements of the project helped me deepen my understanding of SQL, including subqueries, joins, aggregation functions, and conditional statements.
* **Database Normalization:** Applying normalization techniques (1NF, 2NF, 3NF, BCNF) to ensure the database design adheres to best practices for data organization and integrity. This practical application reinforced my understanding of database normalization concepts.
* **Data Modeling:** Designing entity-relationship diagrams (ERDs) and translating them into relational database schema helped me improve my skills in data modeling and schema design, considering entities, attributes, relationships, and constraints.
* **Project Management:** Managing the progression of the project, including database creation, schema design, data insertion, and query execution, enhanced my project management skills, particularly in organizing tasks, setting priorities, and meeting deadlines.
* **Problem-Solving:** Addressing challenges and resolving issues encountered during the project, such as optimizing queries, troubleshooting errors, and ensuring data consistency, honed my problem-solving abilities and fostered a proactive approach to tackling technical issues.
* **Documentation:** Creating clear and comprehensive documentation, including ER diagrams, schema definitions, query explanations, and project instructions, improved my ability to communicate technical concepts effectively and maintain organized project documentation.

**VIII. Learning from the Project**

* **Practical Application of Knowledge:** It allowed me to apply theoretical concepts learned in the classroom to real-world scenarios, reinforcing my understanding of database management principles and SQL query techniques.
* **Problem-Solving Skills:** Addressing challenges encountered during the project enhanced my problem-solving abilities, fostering a proactive approach to tackling technical issues.
* **Project Management Experience:** Managing the project's progression improved my project management skills, particularly in organizing tasks and meeting deadlines.
* **Technical Proficiency:** Working with various software tools and libraries commonly used for database management expanded my technical proficiency and familiarity with industry-standard tools.
* **Communication Skills:** Creating comprehensive documentation improved my ability to communicate technical concepts effectively and maintain organized project documentation.

**IX. Challenges Faced**

* **Schema Design Complexity:** Designing an efficient and normalized database schema to accommodate the various entities, attributes, and relationships while ensuring data integrity posed a significant challenge.
* **Optimizing Queries:** Crafting complex SQL queries to retrieve specific data while maintaining optimal performance required careful optimization and fine-tuning, especially when dealing with large datasets.
* **Data Consistency:** Ensuring data consistency across multiple tables and enforcing referential integrity constraints presented challenges, particularly when inserting, updating, or deleting records across related tables.
* **Handling Errors:** Troubleshooting and resolving errors encountered during database creation, data insertion, and query execution required thorough debugging and problem-solving skills.

**X. Conclusion**

* **Practical Application of Database Concepts:** Applying theoretical knowledge of database management concepts such as normalization, entity-relationship modeling, and SQL query optimization in a real-world scenario reinforced understanding and practical skills.
* **Problem-Solving and Critical Thinking:** Addressing challenges encountered during database design, data manipulation, and query execution honed problem-solving abilities and fostered critical thinking skills in analyzing requirements and devising solutions.
* **Project Management Skills:** Managing the project's progression, including task organization, timeline management, and documentation maintenance, improved project management skills and demonstrated the importance of effective project planning and execution.
* **Technical Proficiency with Database Tools:** Working with various database management tools and libraries, such as MySQL, phpMyAdmin, and SQLAlchemy, enhanced technical proficiency and familiarity with industry-standard tools used for database development and management.
* **Communication and Documentation:** Creating comprehensive documentation, including entity-relationship diagrams, schema definitions, and query explanations, improved communication skills and underscored the importance of clear and organized documentation for effective collaboration and knowledge sharing.