Project Report Submitted for

DATABASE MANAGEMENT SYSTEM (UCS310)

Submitted by:

(102117079) Ayush Chaturvedi

(102117072) Abhishek Meena

(102117073) Shivam Sehra

BE Second Year

Submitted to :-

MR. ANIL SIR

Computer Science and Engineering Department
TIET, Patiala

Jan-May 2023

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

INDEX

INTRODUCTION

Due to the rapid expansion of the internet and the tourist sector, more and more people are choosing to book their travel through online channels. A trustworthy, secure, and user-friendly online trip booking database system is therefore greatly needed. A system like this may give clients the tools they need to conveniently search, compare, and book flights, hotels, and other travel-related services in addition to giving travel organisations, hotel owners, tour operators, and administrators the resources they require to run their businesses successfully.

This report's overall goal is to present a thorough grasp of the prerequisites for an online travel booking database system and to emphasise the advantages such system may give for both travellers and travel agencies.

REQUIREMENT ANALYSIS

Requirement analysis for an online traveling booking database system involves identifying the needs and expectations of users, the technical requirements for the system, and the business requirements of the project.

Technical Requirements:

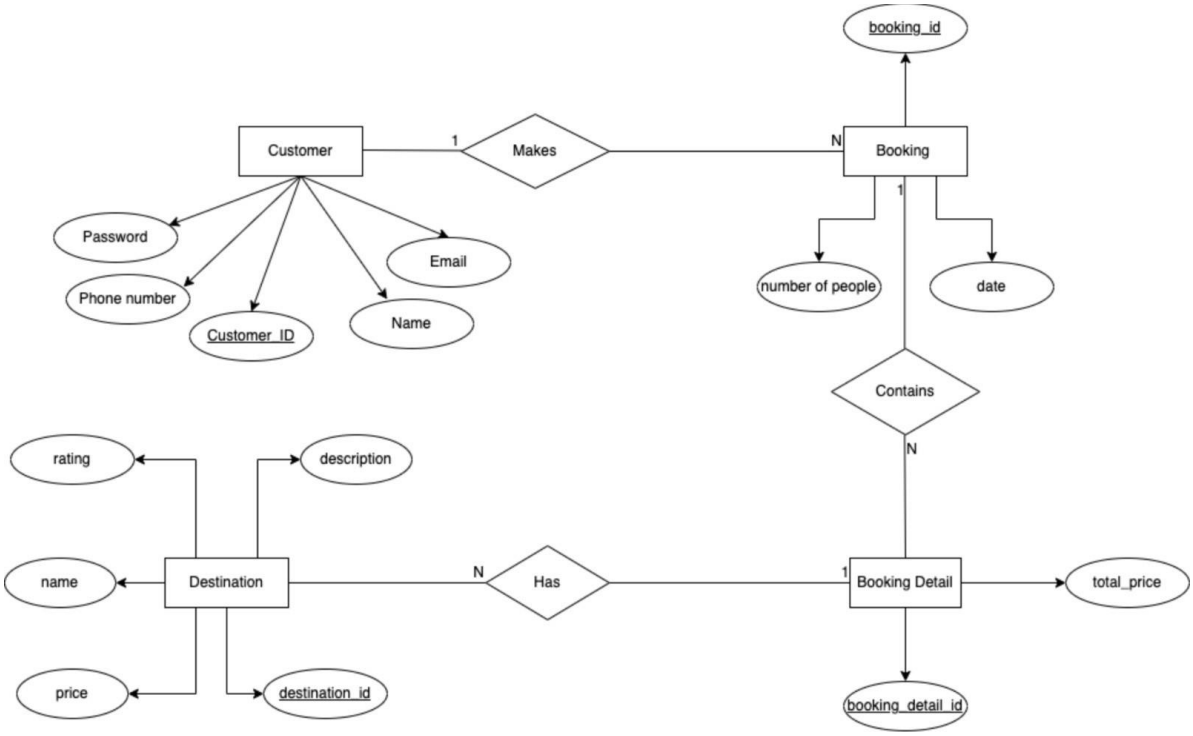
1. Security: The system should have robust security features to protect user data, payment information, and other sensitive data from unauthorized access and attacks.
2. Scalability: The system should be scalable to handle a large number of users, transactions, and data volumes.
3. Reliability: The system should be highly available and reliable, with minimal downtime or disruptions.
4. Performance: The system should be optimized for fast performance and response times, especially during peak usage periods.
5. Integration: The system should be able to integrate with various travel-related services such as payment gateways, airline booking systems, and hotel booking systems.
6. Database Management: The system should have a reliable and efficient database management system to store and manage large volumes of data.
7. User Interface: The system should have a user-friendly interface that is easy to use and navigate, with clear instructions and feedback.

Business Requirements:

1. Booking and Reservation: The system should allow users to search and book flights, hotels, car rentals, and other travel-related services.
2. Payment Processing: The system should support multiple payment options and provide a secure payment processing system.
3. Reporting and Analytics: The system should provide real-time reporting and analytics capabilities to help travel businesses monitor their operations and make data-driven decisions.
4. User Management: The system should allow users to register, log in, and manage their accounts.
5. Customer Support: The system should provide reliable customer support through multiple channels such as phone, email, and chat.
6. Marketing: The system should have marketing features to promote travel-related services to users, including targeted promotions and special offers.

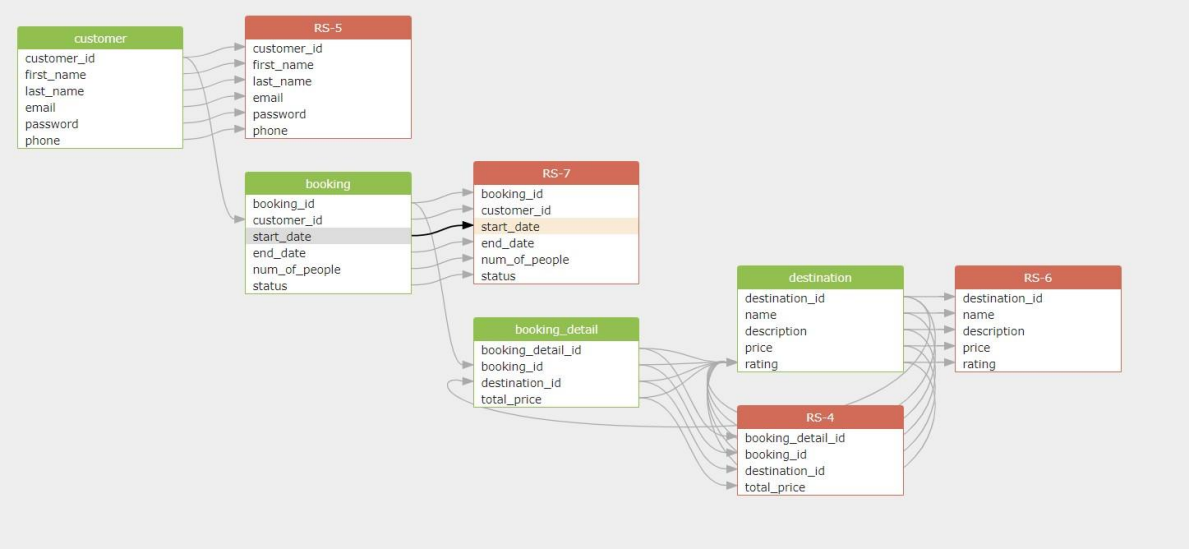
Overall, an effective online traveling booking agency database management system should meet these technical and business requirements to ensure a seamless and satisfying user experience, improve travel-related businesses' operations, and increase revenue.

E-R DIAGRAM


booking_id
Customer
1
Makes
N
Booking
1
Password
Email
Phone number
Customer_ID
Name
number of people
date
Contains
N
rating
description
name
Destination
N
Has
1
Booking Detail
total_price
price
destination_id
booking_detail_id

E-R TO TABLE

Attached below is the effective conversion of Entity-Relationship Diagram to respective Tables.



NORMALIZATION

The normalization of the database has been done up to the third normal form (3NF). Below is a breakdown of how each table is normalized:

Customer table: The Customer table appears to be in 1NF, with each row containing unique data and each column containing atomic values.

Destination table: The Destination table also appears to be in 1NF, with each row containing unique data and each column containing atomic values.

Booking table: The Booking table is in 2NF, with the primary key (booking_id) uniquely identifying each row, and each non-key column (customer_id, booking_date) being fully dependent on the primary key.

Booking_Detail table: The Booking_Detail table is in 3NF, with the primary key (booking_detail_id) uniquely identifying each row, and each non-key column being independent of other non-key columns.

Overall, the normalization of the database up to 3NF should help ensure that data is stored efficiently and consistently, and that updates and modifications to the data can be made without causing data inconsistencies or redundancies.

SQL & PL/SQL SNAPSHOTS

Creating the tables :

- Customer

```
CREATE TABLE customer (
  customer_id NUMBER(10) PRIMARY KEY,
  first_name VARCHAR2(50),
  last_name VARCHAR2(50),
  email VARCHAR2(50) UNIQUE,
  password VARCHAR2(50),
  phone VARCHAR2(20)
)

Table created.
```

- Destination

```
CREATE TABLE destination (
  destination_id NUMBER(10) PRIMARY KEY,
  name VARCHAR2(50),
  description VARCHAR2(200),
  price NUMBER(10, 2),
  rating NUMBER(1, 1)
)

Table created.
```

- Booking

```
CREATE TABLE booking (
  booking_id NUMBER(10) PRIMARY KEY,
  customer_id NUMBER(10),
  start_date DATE,
  end_date DATE,
  num_of_people NUMBER(3),
  status VARCHAR2(20),
  CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
)
```

```
Table created.
```

- Booking_detail

```
CREATE TABLE booking_detail (
  booking_detail_id NUMBER(10) PRIMARY KEY,
  booking_id NUMBER(10),
  destination_id NUMBER(10),
  total_price NUMBER(10, 2),
  CONSTRAINT fk_booking FOREIGN KEY (booking_id) REFERENCES booking(booking_id),
  CONSTRAINT fk_destination FOREIGN KEY (destination_id) REFERENCES destination(destination_id)
)
```

```
Table created.
```

Inserting into tables:

- Customer

```
SELECT * FROM CUSTOMER
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	PASSWORD	PHONE
1	John	Doe	johndoe@gmail.com	12345	123-456-7890
2	Jane	Doe	janedoe@gmail.com	12345	987-654-3210
3	Bob	Smith	bobsmith@gmail.com	12345	555-555-5555

Download CSV

3 rows selected.

- Destination

```
SELECT * FROM DESTINATION
```

DESTINATION_ID	NAME	DESCRIPTION	PRICE	RATING
2	London	Big Ben	40	9
3	Paris	Eiffel Tower	30	8
1	New York	Statue of Liberty	50	9

Download CSV

3 rows selected.

- Booking

```
SELECT * FROM BOOKING
```

BOOKING_ID	CUSTOMER_ID	START_DATE	END_DATE	NUM_OF_PEOPLE	STATUS
1	1	03-MAR-23	03-APR-23	1	Confirmed
2	2	04-APR-23	04-MAY-23	2	Pending
3	3	05-MAY-23	05-JUN-23	3	Confirmed

Download CSV

3 rows selected.

- Booking_detail

```
SELECT * FROM BOOKING_DETAIL
```

BOOKING_DETAIL_ID	BOOKING_ID	DESTINATION_ID	TOTAL_PRICE
1	1	1	50
2	2	2	40
3	3	3	30

Download CSV

3 rows selected.

- Procedures

```
CREATE OR REPLACE PROCEDURE get_bookings_by_customer(
  customer_id IN NUMBER,
  bookings OUT SYS_REFCURSOR
)
IS
BEGIN
  OPEN bookings FOR
    SELECT * FROM booking WHERE customer_id = customer_id;
END;
```

```
Procedure created.
```

- Triggers

```
CREATE OR REPLACE TRIGGER update_booking_status
AFTER UPDATE ON Booking_Detail
FOR EACH ROW
DECLARE
    v_booking_id NUMBER;
BEGIN
    SELECT booking_id
    INTO v_booking_id
    FROM Booking_Detail
    WHERE booking_detail_id = :new.booking_detail_id;
    UPDATE Booking
    SET status =
        (CASE
            WHEN EXISTS (SELECT 1 FROM Booking_Detail WHERE booking_id = v_booking_id AND status = 'Pending') THEN 'Pending'
            WHEN EXISTS (SELECT 1 FROM Booking_Detail WHERE booking_id = v_booking_id AND status = 'Cancelled') THEN 'Cancelled'
            ELSE 'Confirmed'
        END)
    WHERE booking_id = v_booking_id;
END;
```

```
Trigger created.
```

Queries :

- Retrieve all the bookings made for a particular destination.(New York).

```
SELECT Booking_Detail.booking_id, Booking.customer_id, Booking.end_date
FROM Booking_Detail
INNER JOIN Destination ON Booking_Detail.destination_id = Destination.destination_id
INNER JOIN Booking ON Booking_Detail.booking_id = Booking.booking_id
WHERE Destination.name = 'New York'
```

BOOKING_ID	CUSTOMER_ID	END_DATE
1	1	03-APR-23

Download CSV

- Retrieve the most popular destinations along with the number of times they have been booked.

```
SELECT Destination.name, COUNT(Booking_Detail.booking_id) AS num_bookings
FROM Destination
INNER JOIN Booking_Detail ON Destination.destination_id = Booking_Detail.destination_id
GROUP BY Destination.name
ORDER BY num_bookings DESC
```

NAME	NUM_BOOKINGS
London	1
New York	1
Paris	1

Download CSV

3 rows selected.

- Retrieve the total number of bookings made by a particular customer

```
SELECT Customer.customer_id, COUNT(Booking.booking_id) AS num_bookings
FROM Customer
LEFT JOIN Booking ON Customer.customer_id = Booking.customer_id
GROUP BY Customer.customer_id
```

CUSTOMER_ID	NUM_BOOKINGS
1	1
2	1
3	1

Download CSV

3 rows selected.

CONCLUSION

To conclude our report, we would like to summarize all that we have accomplished during the working and completion of our project. Firstly, we understood the need for an established and functioning Database System for an online booking & travelling agency, one such that would require an easy and effective access to its clients. Then we grasped the technical and business requirements to establish and maintain such a management system such as Security, Scalability, Reliability, Performance, Integration etc. Constructing an E-R model for our Database system was our primary goal for the initial stage of our project. After converting the E-R model into respective tables, we had to normalize the tables, so as to combat any redundancies that would arise, as those would be quite a problem for the projected scope of a database such as ours. Following the creation of the tables and inserting meaningful values into it, we executed some queries that took the advantage of concepts pertaining to SQL like Triggers, cursors and procedures, providing snapshots of the same.