SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

MINI PROJECT REPORT

ON

"Flight Reservation Management System"

B. TECH

COMPUTER SCIENCE AND ENGINEERING

BATCH: 2023-27

SESSION:2024-25

Course Name – Database Management System

Course Code - 0705210505

SEMESTER - 4th

SECTION – C

Submitted by:

Name: Janhavi Warhade **PRN: 23070521175**

Name: Krutik Gajbhiye **PRN: 23070521181**

1. Introduction 3
2. Objectives 3
3. Theoretical Framework 3
    - 3.1. Relational Database Model 3
    - 3.2. Entity-Relationship (ER) Modelling 4
    - 3.3. SQL Constraints 4
    - 3.4. SQL Joins 4
    - 3.5. PL/SQL Functions and Procedures 5
    - 3.6. Triggers 5
    - 3.7. Views 5
4. System Analysis and Design 6
5. Database Schema Design 8
6. Implementation 10
    - 6.1. Table Creation 10
    - 6.2. Adding Constraints 12
    - 6.3. Advanced SQL Queries 13
    - 6.4. PL/SQL Functions and Procedures 16
    - 6.5. Triggers 21
    - 6.6. Views 23
7. Conclusion 28

1. Introduction

The **Flight reservation management system** is designed to streamline and automate the various operations involved in managing flight reservations. This system encompasses the management of flights, customers, passport, tickets, and payment status. By leveraging a relational database, the system ensures data integrity, facilitates efficient data retrieval, and automates critical business processes through advanced SQL features, PL/SQL functions, and triggers.

2. Objectives

The primary objectives of this project are:

1. **Database Design and Implementation**: To design a robust relational database that effectively manages real estate data, including houses, sales, customers, offers, and salesmen.
2. **Data Integrity Enforcement**: To maintain data accuracy and consistency using SQL constraints such as primary keys and foreign keys.
3. **Process Automation**: To automate essential business processes through PL/SQL functions and triggers, enhancing operational efficiency.
4. **Efficient Data Retrieval**: To develop advanced SQL queries that facilitate comprehensive reporting and data analysis.
5. **Performance Optimization**: To enhance database performance using indexing and optimized query structures.

3. Theoretical Framework

3.1. Relational Database Model

Theory: The relational database model organizes data into tables (relations) consisting of rows and columns. Each table represents an entity, and relationships between entities are established through keys. The model emphasizes data integrity and supports operations using Structured Query Language (SQL).

Project Example: In the Flight reservation management system, entities such as **Airline**, **Flight**, and **Passenger** are represented as separate tables. Each entity's attributes are stored as columns within these tables, ensuring a structured and organized data repository.

3.2. Entity-Relationship (ER) Modeling

Theory: ER modeling is a conceptual framework used to represent data structures and relationships in a database. It utilizes entities, attributes, and relationships to visually depict how data interacts within the system.

Project Example: The ER diagram for the project includes entities like **Airline**, **Flight**, **Passenger**, **Booking**, and **Payment**. For instance, the **Flight** entity is related to **Passenger**, **Booking**, and **Payment** through defined relationships, illustrating how these entities interact within the flight reservation management process.

3.3. SQL Constraints

Theory: SQL constraints enforce rules on data in tables to maintain accuracy and integrity. Common constraints include:

- **Primary Key**: Uniquely identifies each record in a table.
- **Foreign Key**: Ensures referential integrity between related tables.
- **Unique**: Ensures all values in a column are distinct.
- **Not Null**: Ensures a column cannot have NULL values.
- **Check**: Ensures values in a column meet specific criteria.

3.4. SQL Joins

Theory: SQL joins combine rows from two or more tables based on related columns. The primary types of joins include:

- **INNER JOIN**: Returns records with matching values in both tables.
- **LEFT (OUTER) JOIN**: Returns all records from the left table and matched records from the right table.
- **RIGHT (OUTER) JOIN**: Returns all records from the right table and matched records from the left table.
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table.

3.5. PL/SQL Functions and Procedures

Theory: PL/SQL (Procedural Language/SQL) extends SQL with procedural features like loops, conditionals, and exception handling. Functions return values, while procedures perform actions without returning values directly.

- **Procedure**: auto_approve_offers automatically approves offers that meet or exceed the asking price.

3.6. Triggers

Theory: Triggers are special procedures that automatically execute in response to specific events on a table, such as INSERT, UPDATE, or DELETE operations. They are used to enforce business rules, maintain audit trails, and automate tasks.

3.7. Views

Theory: Views are virtual tables derived from one or more tables. They simplify complex queries, enhance security by restricting data access, and provide a consistent interface to the data.

4. ER Diagram Explanation

The ER diagram represents the entities and their relationships within the real estate system:

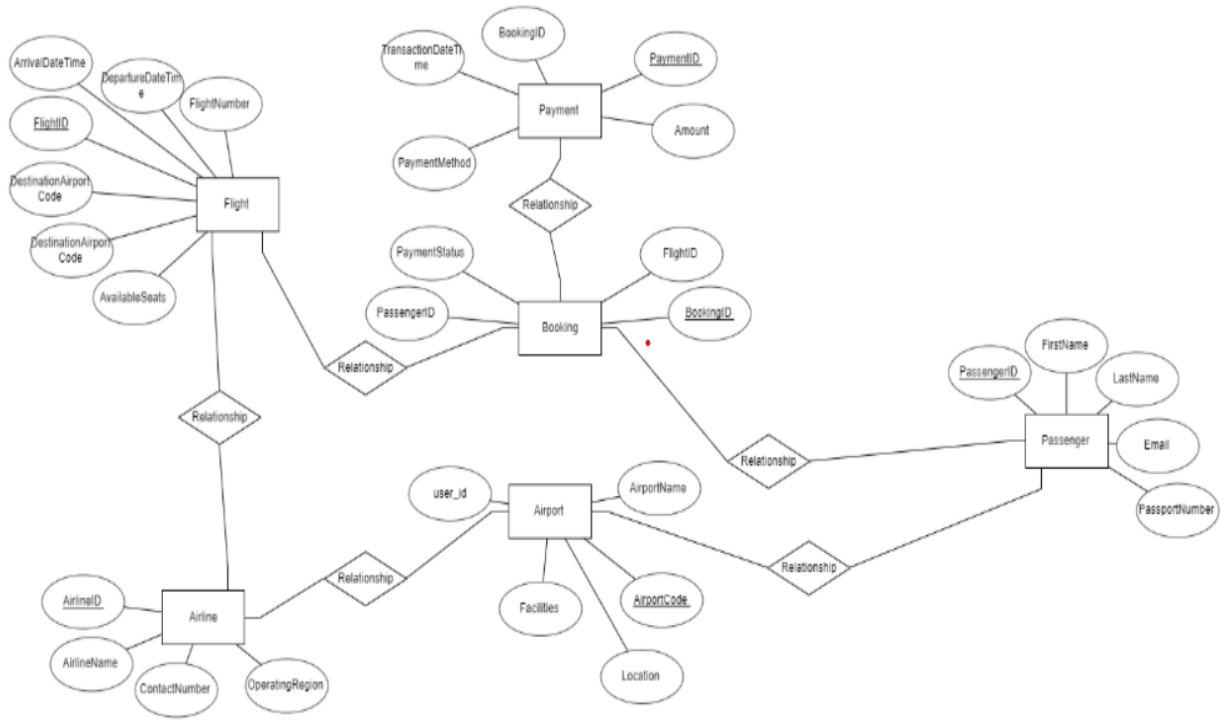

Fig 1 : Entity-Relationship Model

ENTITIES AND ATTRIBUTES

1. Flight :- Represents a scheduled air travel journey.
   Attributes:
   - FlightID (PK)
   - FlightNumber
   - DepartureDateTime
   - ArrivalDateTime
   - AvailableSeats
   - DepartureAirportCode
   - DestinationAirportCode

   Relationships:

- Belongs to one Airline

- Linked to multiple Bookings

2. Passenger :- Represents a customer/passenger using the system.

Attributes:

- PassengerID
- FirstName
- LastName
- Email
- PassportNumber

Relationships:

- Can make multiple Bookings
- Involved in Payments via Booking

3. Booking :- Represents a reservation made by a Passenger for a Flight.

Attributes:

- BookingID
- FlightID
- PassengerID
- PaymentStatus

Relationships:

- Each Booking is for one Flight and one Passenger
- Linked to one Payment

4. Payment :- Represents the transaction made for a Booking.

Attributes:

- PaymentID
- BookingID
- Amount
- PaymentMethod
- TransactionDateTime

Relationships:

- A Passenger can make many Bookings.
- Each Booking is for one Flight and one Passenger.
- Each Booking has one associated Payment.
- A Payment includes details like Amount, Method, and Transaction Date.
- A Flight departs from and arrives at Airports.
- A Flight is operated by one Airline.
- An Airline can have many Flights.
- Airports are connected to Flights via Departure and Destination codes.

5.Database Schema Design

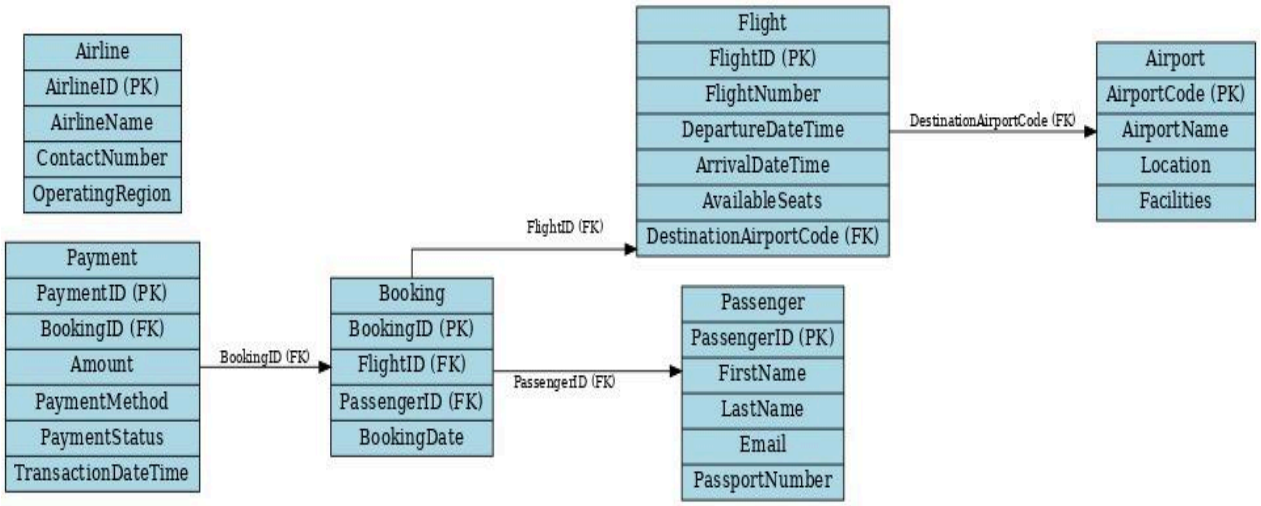

Fig 2 : Schema Diagram

5.1. Tables and Relationships

1.Airline
Description: Represents the airline companies operating flights.
Attributes:

- AirlineID : Unique identifier for each airline.
- AirlineName: Name of the airline company.
- ContactNumber: Official contact number of the airline.
- OperatingRegion: Region(s) in which the airline operates.

2.Airport
Description: Represents airports where flights depart from or arrive at.

Attributes:

- AirportCode : Unique code for the airport (e.g., BOM, JFK).
- AirportName: Full name of the airport.
- Location: City or region where the airport is located.
- Facilities: Services or amenities available at the airport (e.g., WiFi, Lounges).

3.Flight

Description: Represents scheduled flights operated by airlines.

Attributes:

- FlightID: Unique identifier for each flight.
- FlightNumber: Flight number (e.g., AI202).
- DepartureDateTime: Scheduled date and time of departure.
- ArrivalDateTime: Scheduled date and time of arrival.
- AvailableSeats: Number of seats available for booking.
- DestinationAirportCode: Refers to AirportCode in Airport table to indicate destination.

4.Passenger

Description: Represents individuals who travel using the flight service.

Attributes:

- PassengerID (PK): Unique identifier for each passenger.
- FirstName: Passenger's first name.
- LastName: Passenger's last name.
- Email: Contact email address.
- PassportNumber: Official passport number (important for international travel).

5.Booking

Description: Represents the reservation of a flight by a passenger.

Attributes:

- BookingID (PK): Unique booking reference number.
- FlightID (FK): References FlightID to indicate the booked flight.
- PassengerID (FK): References PassengerID to indicate the person who booked.
- BookingDate: Date when the booking was made.

6.Payment

Description: Represents payment details for a specific booking.

Attributes:

- PaymentID (PK): Unique payment transaction ID.
- BookingID (FK): References BookingID to associate the payment with a booking.

- Amount: Total amount paid.
- PaymentMethod: Method used (e.g., Credit Card, UPI, PayPal).
- PaymentStatus: Status of payment (e.g., Paid, Pending).
- TransactionDateTime: Date and time when the payment was made.

5.2. Relationships

- One Airport → Many Flights (via DestinationAirportCode).
- One Flight → Many Bookings.
- One Passenger → Many Bookings.
- One Booking → One Payment.
- One Payment → One Booking.

6. Implementation

6.1. Table Creation & Data Insertion

Table : Passenger at Airport

```
CREATE TABLE PassengersAtAirport (
    passenger_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    passport_number VARCHAR(20) UNIQUE,
    contact_number VARCHAR(15) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    airport_id INT NOT NULL,
    check_in_time DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (airport_id) REFERENCES Airports(airport_id)
);
INSERT INTO PassengersAtAirport (first_name, last_name, passport_number, contact_number,
    email, airport_id, check_in_time)
VALUES
('Rahul', 'Sharma', 'X1234567', '+91-9876543210', 'rahul.sharma@example.com', 1, '2025-02
    -27 08:30:00'),
('Priya', 'Mehta', 'Y9876543', '+91-9823456789', 'priya.mehta@example.com', 2, '2025-02-27
    09:00:00'),
('Amit', 'Verma', 'Z4567891', '+91-9812345678', 'amit.verma@example.com', 3, '2025-02-27
    07:45:00'),
('Sneha', 'Kapoor', 'W6543219', '+91-9832109876', 'sneha.kapoor@example.com', 1, '2025-02
    -27 06:50:00'),
('Rohan', 'Patil', 'P2345678', '+91-9875671234', 'rohan.patil@example.com', 2, '2025-02-27
    10:15:00');
SELECT * FROM PassengersAtAirport;
```

passenger_id	first_name	last_name	passport_number	contact_number	email	airport_id	check_in_time
1	Rahul	Sharma	X1234567	+91-9876543210	rahul.sharma@example.com	1	2025-02-27 08:30:00
2	Priya	Mehta	Y9876543	+91-9823456789	priya.mehta@example.com	2	2025-02-27 09:00:00
3	Amit	Verma	Z4567891	+91-9812345678	amit.verma@example.com	3	2025-02-27 07:45:00
4	Sneha	Kapoor	W6543219	+91-9832109876	sneha.kapoor@example.com	1	2025-02-27 06:50:00
5	Rohan	Patil	P2345678	+91-9875671234	rohan.patil@example.com	2	2025-02-27 10:15:00

Table 2: Booking of Passengers

```sql
CREATE TABLE Booking (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    passenger_id INT NOT NULL,
    flight_id INT NOT NULL,
    payment_status VARCHAR(20) CHECK (payment_status IN ('Pending', 'Completed', 'Failed')),
    FOREIGN KEY (passenger_id) REFERENCES PassengersAtAirport(passenger_id)
);
```

```sql
INSERT INTO Booking (passenger_id, flight_id, payment_status)
VALUES
(1, 101, 'Completed'),
(2, 102, 'Pending'),
(4, 103, 'Failed'),
(5, 104, 'Completed');
```

booking_id	passenger_id	flight_id	payment_status
1	1	101	Completed
2	2	102	Pending
3	4	103	Completed
4	5	104	Failed

6.2. Adding Constraints

After inserting data, it's essential to enforce data integrity through primary keys and foreign keys.

Table 1 : Update

```
UPDATE PassengersAtAirport
SET last_name = 'Varma'
WHERE first_name = 'Rahul' AND last_name = 'Sharma';
SELECT * FROM PassengersAtAirport;
```

passenger_id	first_name	last_name	passport_number	contact_number	email	airport_id	check_in_time
1	Rahul	Varma	X1234567	+91-9876543210	rahul.sharma@example.com	1	2025-02-27 08:30:00
2	Priya	Mehta	Y9876543	+91-9823456789	priya.mehta@example.com	2	2025-02-27 09:00:00
3	Amit	Verma	Z4567891	+91-9812345678	amit.verma@example.com	3	2025-02-27 07:45:00
4	Sneha	Kapoor	W6543219	+91-9832109876	sneha.kapoor@example.com	1	2025-02-27 06:50:00
5	Rohan	Patil	P2345678	+91-9875671234	rohan.patil@example.com	2	2025-02-27 10:15:00

Table 2 : Delete

```
DELETE FROM PassengersAtAirport
WHERE first_name = 'Amit' AND last_name = 'Verma';
SELECT * FROM PassengersAtAirport;
```

passenger_id	first_name	last_name	passport_number	contact_number	email	airport_id	check_in_time
1	Rahul	Varma	X1234567	+91-9876543210	rahul.sharma@example.com	1	2025-02-27 08:30:00
2	Priya	Mehta	Y9876543	+91-9823456789	priya.mehta@example.com	2	2025-02-27 09:00:00
4	Sneha	Kapoor	W6543219	+91-9832109876	sneha.kapoor@example.com	1	2025-02-27 06:50:00
5	Rohan	Patil	P2345678	+91-9875671234	rohan.patil@example.com	2	2025-02-27 10:15:00

Explanation:

- **Primary Keys** ensure each record in a table is unique.
- **Foreign Keys** enforce referential integrity by linking related records across tables.

6.3. Advanced SQL Queries:

Retrieve information based on Tables created.

Fig 3 : Query

```sql
SELECT COUNT(*) AS total_passengers FROM PassengersAtAirport;

-- Average ticket price
SELECT AVG(ticket_price) AS average_ticket_price FROM PassengersAtAirport;

-- Total ticket revenue
SELECT SUM(ticket_price) AS total_ticket_revenue FROM PassengersAtAirport;
```

```
4
4475.375
17901.5
```

Fig 4 : Query

```sql
SELECT first_name, last_name, ticket_price,
       ROUND(ticket_price, 0) AS rounded_price
FROM PassengersAtAirport;
SELECT first_name, last_name, ticket_price,
       ABS(ticket_price) AS absolute_price
FROM PassengersAtAirport;
SELECT first_name, last_name, ticket_price,
       MOD(ticket_price, 1000) AS remainder
FROM PassengersAtAirport;
```

first_name	last_name	ticket_price	rounded_price	absolute_price	remainder
Rahul	Varma	5000.50	5001	5000.50	0.50
Priya	Mehta	4200.75	4201	4200.75	200.75
Sneha	Kapoor	3900.00	3900	3900.00	900.00
Rohan	Patil	4800.25	4800	4800.25	800.25

Fig 5 : Query

```
SELECT first_name, last_name,
       UPPER(first_name) AS upper_first_name,
       UPPER(email) AS upper_email
FROM PassengersAtAirport;
SELECT first_name, last_name,
       LOWER(last_name) AS lower_last_name,
       LOWER(email) AS lower_email
FROM PassengersAtAirport;
SELECT first_name, last_name, email,
       SUBSTR(email, 1, 5) AS email_prefix
FROM PassengersAtAirport;
```

first_name	last_name	upper_first_name	upper_email	lower_last_name	lower_email	email_prefix
Rahul	Varma	RAHUL	RAHUL.SHARMA@EXAMPLE.COM	varma	rahul.sharma@example.com	rahul
Priya	Mehta	PRIYA	PRIYA.MEHTA@EXAMPLE.COM	mehta	priya.mehta@example.com	priya

Fig 6 : Query

```
SELECT first_name, last_name, ticket_price,
       TO_CHAR(ticket_price, '999,999.99') AS formatted_price
FROM PassengersAtAirport;
SELECT first_name, last_name, check_in_time,
       TO_CHAR(check_in_time, 'DD-Mon-YYYY HH:MI AM') AS formatted_checkin
FROM PassengersAtAirport;
SELECT TO_TIMESTAMP('2025-02-27 08:30:00', 'YYYY-MM-DD HH24:MI:SS') AS
    converted_timestamp;
```

first_name	last_name	ticket_price	formatted_price	check_in_time	formatted_checkin
Rahul	Varma	5000.50	5,000.50	2025-02-27 08:30:00	27-Feb-2025 08:30 AM
Priya	Mehta	4200.75	4,200.75	2025-02-27 09:00:00	27-Feb-2025 09:00 AM

2) **Retrieve Highest ticket price from Passenger**

AND

```
SELECT * FROM PassengersAtAirport
WHERE airport_id = 1 AND ticket_price > 4000;
```

passenger_id	first_name	last_name	airport_id	ticket_price
1	Rahul	Varma	1	5000.50

OR

```
SELECT * FROM PassengersAtAirport
WHERE airport_id = 1 OR ticket_price > 4500;
```

passenger_id	first_name	last_name	airport_id	ticket_price
1	Rahul	Varma	1	5000.50
4	Sneha	Kapoor	1	3900.00
5	Rohan	Patil	2	4800.25

NOT

```
SELECT * FROM PassengersAtAirport
WHERE NOT airport_id = 1;
```

passenger_id	first_name	last_name	airport_id	ticket_price
2	Priya	Mehta	2	4200.75
5	Rohan	Patil	2	4800.25

LIKE

```
SELECT * FROM PassengersAtAirport
WHERE email LIKE '%@example.com';
```

passenger_id	first_name	last_name	email
1	Rahul	Varma	rahul.sharma@example.com
2	Priya	Mehta	priya.mehta@example.com
4	Sneha	Kapoor	sneha.kapoor@example.com
5	Rohan	Patil	rohan.patil@example.com

BETWEEN

```
SELECT * FROM PassengersAtAirport
WHERE ticket_price BETWEEN 4000 AND 5000;
```

passenger_id	first_name	last_name	ticket_price
2	Priya	Mehta	4200.75
5	Rohan	Patil	4800.25

IN

```
SELECT * FROM PassengersAtAirport
WHERE airport_id IN (1, 2);
```

passenger_id	first_name	last_name	airport_id
1	Rahul	Varma	1
2	Priya	Mehta	2
4	Sneha	Kapoor	1
5	Rohan	Patil	2

6.4. PL/SQL Functions and Procedures

SIMPLE PROGRAM

```
SQL> --PL/SQL code to print product of two numbers taken from the user.
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2  --taking input for variable a
  3  a integer := &a ;
  4  --taking input for variable b
  5  b integer := &b ;
  6  c integer ;
  7  BEGIN
  8  c := a * b ;
  9  dbms_output.put_line('product of '||a||' and '||b||' is = '||c);
 10  END;
 11  /
Enter value for a: 8
old   3: a integer := &a ;
new   3: a integer := 8 ;
Enter value for b: 1
old   5: b integer := &b ;
new   5: b integer := 1 ;
product of 8 and 1 is = 8

PL/SQL procedure successfully completed.
```

```
SQL> --PL/SQL code to print sum of two numbers taken from the user.
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2  --taking input for variable a
  3  a integer := &a ;
  4  --taking input for variable b
  5  b integer := &b ;
  6  c integer ;
  7  BEGIN
  8  c := a + b ;
  9  dbms_output.put_line('sum of '||a||' and '||b||' is = '||c);
 10  END;
 11  /
Enter value for a: 2
old   3: a integer := &a ;
new   3: a integer := 2 ;
Enter value for b: 3
old   5: b integer := &b ;
new   5: b integer := 3 ;
sum of 2 and 3 is = 5

PL/SQL procedure successfully completed.
```

IF THEN

```
SQL> DECLARE
  2  HINDI INTEGER := &HINDI;
  3  BEGIN
  4  IF HINDI > 33 THEN
  5  DBMS_OUTPUT.PUT_LINE('PASS IN HINDI SUBJECT');
  6  END IF;
  7  END;
  8  /
Enter value for hindi: 67
old   2: HINDI INTEGER := &HINDI;
new   2: HINDI INTEGER := 67;
PASS IN HINDI SUBJECT

PL/SQL procedure successfully completed.
```

IF THEN ELSE

```
SQL> DECLARE
  2  HINDI INTEGER := &HINDI;
  3  BEGIN
  4  IF HINDI > 33 THEN
  5  DBMS_OUTPUT.PUT_LINE('PASS IN HINDI SUBJECT');
  6  ELSE
  7  DBMS_OUTPUT.PUT_LINE('FAILED IN HINDI SUBJECT');
  8  END IF;
  9  END;
 10  /
Enter value for hindi: 12
old   2: HINDI INTEGER := &HINDI;
new   2: HINDI INTEGER := 12;
FAILED IN HINDI SUBJECT

PL/SQL procedure successfully completed.
```

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  HINDI INTEGER := &HINDI;
  3  MATHS INTEGER := &MATHS;
  4  SCIENCE INTEGER := &SCIENCE;
  5  SANSKRIT INTEGER := &SANSKRIT;
  6  BEGIN
  7  IF HINDI > 33 THEN
  8     DBMS_OUTPUT.PUT_LINE('PASS IN HINDI SUBJECT');
  9  ELSE
 10     DBMS_OUTPUT.PUT_LINE('FAIL IN HINDI SUBJECT');
 11  END IF;
 12  IF MATHS > 33 THEN
 13     DBMS_OUTPUT.PUT_LINE('PASS IN MATHS SUBJECT');
 14  ELSE
 15     DBMS_OUTPUT.PUT_LINE('FAIL IN MATHS SUBJECT');
 16  END IF;
 17  IF SCIENCE > 33 THEN
 18     DBMS_OUTPUT.PUT_LINE('PASS IN SCIENCE SUBJECT');
 19  ELSE
 20     DBMS_OUTPUT.PUT_LINE('FAIL IN SCIENCE SUBJECT');
 21  END IF;
 22  IF SANSKRIT > 33 THEN
 23     DBMS_OUTPUT.PUT_LINE('PASS IN SANSKRIT SUBJECT');
 24  ELSE
 25    DBMS_OUTPUT.PUT_LINE('FAIL IN SANSKRIT SUBJECT');
 26  END IF;
 27  END;
 28  /
Enter value for hindi: 67
old   2: HINDI INTEGER := &HINDI;
new   2: HINDI INTEGER := 67;
Enter value for maths: 23
old   3: MATHS INTEGER := &MATHS;
new   3: MATHS INTEGER := 23;
Enter value for science: 19
old   4: SCIENCE INTEGER := &SCIENCE;
new   4: SCIENCE INTEGER := 19;
Enter value for sanskrit: 90
old   5: SANSKRIT INTEGER := &SANSKRIT;
new   5: SANSKRIT INTEGER := 90;
PASS IN HINDI SUBJECT
FAIL IN MATHS SUBJECT
FAIL IN SCIENCE SUBJECT
PASS IN SANSKRIT SUBJECT

PL/SQL procedure successfully completed.
```

FOR LOOP

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  TOTAL INTEGER :=0;
  3  BEGIN
  4  FOR i IN 1..5 LOOP
  5  TOTAL := TOTAL+i;
  6  END LOOP;
  7  DBMS_OUTPUT.PUT_LINE(TOTAL);
  8  END;
  9  /
15

PL/SQL procedure successfully completed.
```

WHILE LOOP

```
SQL> DECLARE
  2   TOTAL INTEGER :=0; COUNTER INTEGER :=1;
  3   BEGIN
  4   WHILE COUNTER<=10 LOOP
  5   TOTAL := TOTAL+COUNTER;
  6   COUNTER := COUNTER+1;
  7   END LOOP;
  8   DBMS_OUTPUT.PUT_LINE(TOTAL);
  9  END;
 10  /
55

PL/SQL procedure successfully completed.
```

```
SQL> CREATE OR REPLACE FUNCTION decimal_to_binary (dec_num IN NUMBER)
  2  RETURN VARCHAR2 IS
  3      binary_result VARCHAR2(100) := '';
  4      num NUMBER := dec_num;
  5      remainder NUMBER;
  6  BEGIN
  7      IF num = 0 THEN
  8          RETURN '0';
  9      END IF;
 10
 11      WHILE num > 0 LOOP
 12          remainder := MOD(num, 2);
 13          binary_result := remainder || binary_result;
 14          num := TRUNC(num / 2);
 15      END LOOP;
 16
 17      RETURN binary_result;
 18  END;
 19  /

Function created.

SQL> SELECT decimal_to_binary(10) FROM dual;

DECIMAL_TO_BINARY(10)
--------------------------------------------------------------------------------
1010

SQL>
```

```
SQL> CREATE TABLE bank_account (
  2      account_no NUMBER PRIMARY KEY,
  3      account_holder VARCHAR2(100),
  4      balance NUMBER CHECK (balance >= 0)
  5  );

Table created.

SQL> INSERT INTO bank_account VALUES (101, 'Alice', 5000);

1 row created.

SQL> INSERT INTO bank_account VALUES (102, 'Bob', 3000);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> CREATE OR REPLACE PROCEDURE transfer_funds (
  2      sender_acct IN NUMBER,
  3      receiver_acct IN NUMBER,
  4      transfer_amount IN NUMBER
  5  ) AS
  6      sender_balance NUMBER;
  7  BEGIN
  8      SELECT balance INTO sender_balance FROM bank_account WHERE account_no = sender_acct;
  9
 10      IF sender_balance < transfer_amount THEN
 11          RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in sender account');
 12      END IF;
 13
 14      UPDATE bank_account SET balance = balance - transfer_amount WHERE account_no = sender_acct;
 15      UPDATE bank_account SET balance = balance + transfer_amount WHERE account_no = receiver_acct;
 16
 17      COMMIT;
 18
 19      DBMS_OUTPUT.PUT_LINE('Transaction successful! ' || transfer_amount || ' transferred from ' || sender_acct || ' to ' || receiver_acct);
 20  EXCEPTION
 21      WHEN NO_DATA_FOUND THEN
 22          RAISE_APPLICATION_ERROR(-20002, 'One or both accounts do not exist');
 23      WHEN OTHERS THEN
 24          ROLLBACK;
```

```
 20  EXCEPTION
 21      WHEN NO_DATA_FOUND THEN
 22          RAISE_APPLICATION_ERROR(-20002, 'One or both accounts do not exist');
 23      WHEN OTHERS THEN
 24          ROLLBACK;
 25          RAISE_APPLICATION_ERROR(-20003, 'Transaction failed due to an unexpected error');
 26  END;
 27  /

Procedure created.

SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
  2      transfer_funds(101, 102, 1000);
  3  END;
  4  /
Transaction successful! 1000 transferred from 101 to 102

PL/SQL procedure successfully completed.

SQL> SELECT * FROM bank_account;

ACCOUNT_NO
----------
ACCOUNT_HOLDER
--------------------------------------------------------------------------------
   BALANCE
----------
       101
Alice
      4000

       102
Bob
      4000

ACCOUNT_NO
----------
ACCOUNT_HOLDER
--------------------------------------------------------------------------------
   BALANCE
----------
```

6.5. Triggers

```
SQL*Plus: Release 11.2.0.4.0 Production on Mon Apr 14 14:47:48 2025

Copyright (c) 1982, 2013, Oracle.  All rights reserved.

Enter user-name: scott
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> -- Step 1: Drop tables if they already exist (optional, safe for re-run)
SQL> BEGIN
  2      EXECUTE IMMEDIATE 'DROP TABLE emp_audit';
  3  EXCEPTION
  4      WHEN OTHERS THEN NULL;
  5  END;
  6  /

PL/SQL procedure successfully completed.

SQL>
SQL> BEGIN
  2      EXECUTE IMMEDIATE 'DROP TABLE employees';
  3  EXCEPTION
  4      WHEN OTHERS THEN NULL;
  5  END;
  6  /

PL/SQL procedure successfully completed.

SQL>
```

```
SQL>
SQL> -- Step 2: Create main table
SQL> CREATE TABLE employees (
  2       emp_id NUMBER PRIMARY KEY,
  3       emp_name VARCHAR2(100),
  4       emp_salary NUMBER
  5  );

Table created.

SQL>
SQL> -- Step 3: Create audit table
SQL> CREATE TABLE emp_audit (
  2       emp_id NUMBER,
  3       emp_name VARCHAR2(100),
  4       action_date DATE,
  5       action_type VARCHAR2(20)
  6  );

Table created.

SQL>
SQL> -- Step 4: Create AFTER INSERT trigger
SQL> CREATE OR REPLACE TRIGGER trg_emp_after_insert
  2  AFTER INSERT ON employees
  3  FOR EACH ROW
  4  BEGIN
  5      INSERT INTO emp_audit (emp_id, emp_name, action_date, action_type)
  6      VALUES (:NEW.emp_id, :NEW.emp_name, SYSDATE, 'INSERT');
  7  END;
  8  /

Trigger created.
```

```
SQL>
SQL> -- Step 5: Create BEFORE UPDATE trigger
SQL> CREATE OR REPLACE TRIGGER trg_emp_before_update
  2  BEFORE UPDATE ON employees
  3  FOR EACH ROW
  4  BEGIN
  5      INSERT INTO emp_audit (emp_id, emp_name, action_date, action_type)
  6      VALUES (:OLD.emp_id, :OLD.emp_name, SYSDATE, 'UPDATE');
  7  END;
  8  /

Trigger created.

SQL>
SQL> -- Step 6: Create BEFORE DELETE trigger
SQL> CREATE OR REPLACE TRIGGER trg_emp_before_delete
  2  BEFORE DELETE ON employees
  3  FOR EACH ROW
  4  BEGIN
  5      INSERT INTO emp_audit (emp_id, emp_name, action_date, action_type)
  6      VALUES (:OLD.emp_id, :OLD.emp_name, SYSDATE, 'DELETE');
  7  END;
  8  /

Trigger created.

SQL>
SQL> -- Step 7: Insert a record
SQL> INSERT INTO employees (emp_id, emp_name, emp_salary)
  2  VALUES (101, 'Aarav', 55000);

1 row created.
```

```
SQL>
SQL> -- Step 8: Update the record
SQL> UPDATE employees
  2  SET emp_salary = 60000
  3  WHERE emp_id = 101;

1 row updated.

SQL>
SQL> -- Step 9: Delete the record
SQL> DELETE FROM employees
  2  WHERE emp_id = 101;

1 row deleted.

SQL>
SQL> -- Step 10: View the audit log
SQL> SELECT * FROM emp_audit;

    EMP_ID
----------
EMP_NAME
--------------------------------------------------
ACTION_DA ACTION_TYPE
--------- --------------------
       101
Aarav
14-APR-25 INSERT

       101
Aarav
14-APR-25 UPDATE
```

```
SQL> -- Step 10: View the audit log
SQL> SELECT * FROM emp_audit;

    EMP_ID
----------
EMP_NAME
--------------------------------------------------------------------
ACTION_DA ACTION_TYPE
--------- --------------------
       101
Aarav
14-APR-25 INSERT

       101
Aarav
14-APR-25 UPDATE

    EMP_ID
----------
EMP_NAME
--------------------------------------------------------------------
ACTION_DA ACTION_TYPE
--------- --------------------

       101
Aarav
14-APR-25 DELETE


SQL>
```

6.6. Views

Examples:

Purpose: To provide a consolidated view of both tables.

a) Passengers who have completed payment

```sql
SELECT first_name, last_name, email
FROM PassengersAtAirport
WHERE passenger_id IN (
    SELECT passenger_id FROM Booking WHERE payment_status = 'Completed'
);
```

first_name	last_name	email
Rahul	Varma	rahul.sharma@example.com
Sneha	Kapoor	sneha.kapoor@example.com

b) Flights where payment is still pending

```sql
SELECT flight_id FROM Booking
WHERE payment_status = 'Pending';
```

flight_id
102

c) Passenger with the highest ticket price

```sql
SELECT * FROM PassengersAtAirport
WHERE ticket_price = (SELECT MAX(ticket_price) FROM PassengersAtAirport);
```

passenger_id	first_name	last_name	passport_number	contact_number
1	Rahul	Varma	X1234567	+91-9000000001

email	airport_id	check_in_time	ticket_price
rahul.sharma@example.com	1	2025-02-27 08:30:00	5000.50

JOINS

Inner join – passengers with their booking details

```sql
SELECT p.passenger_id, p.first_name, p.last_name, b.booking_id, b.flight_id, b.payment_status
FROM PassengersAtAirport p
INNER JOIN Booking b ON p.passenger_id = b.passenger_id;
```

passenger_id	first_name	last_name	booking_id	flight_id	payment_status
1	Rahul	Varma	1	101	Completed
2	Priya	Mehta	2	102	Pending
4	Sneha	Kapoor	3	103	Completed
5	Rohan	Patil	4	104	Failed

Left join – Show all passengers and their booking info

```sql
SELECT p.passenger_id, p.first_name, p.last_name, b.booking_id, b.flight_id, b.payment_status
FROM PassengersAtAirport p
LEFT JOIN Booking b ON p.passenger_id = b.passenger_id;
```

passenger_id	first_name	last_name	booking_id	flight_id	payment_status
1	Rahul	Varma	1	101	Completed
2	Priya	Mehta	2	102	Pending
4	Sneha	Kapoor	3	103	Completed
5	Rohan	Patil	4	104	Failed

Right join – Show all bookings and their passenger info

```sql
SELECT p.passenger_id, p.first_name, p.last_name, b.booking_id, b.flight_id, b.payment_status
FROM PassengersAtAirport p
RIGHT JOIN Booking b ON p.passenger_id = b.passenger_id;
```

passenger_id	first_name	last_name	booking_id	flight_id	payment_status
1	Rahul	Varma	1	101	Completed
2	Priya	Mehta	2	102	Pending
4	Sneha	Kapoor	3	103	Completed
5	Rohan	Patil	4	104	Failed

Full outer join – show all passengers and booking

```sql
SELECT p.passenger_id, p.first_name, p.last_name, b.booking_id, b.flight_id, b.payment_status
FROM PassengersAtAirport p
LEFT JOIN Booking b ON p.passenger_id = b.passenger_id
UNION
SELECT p.passenger_id, p.first_name, p.last_name, b.booking_id, b.flight_id, b.payment_status
FROM PassengersAtAirport p
RIGHT JOIN Booking b ON p.passenger_id = b.passenger_id;
```

passenger_id	first_name	last_name	booking_id	flight_id	payment_status
1	Rahul	Varma	1	101	Completed
2	Priya	Mehta	2	102	Pending
4	Sneha	Kapoor	3	103	Completed
5	Rohan	Patil	4	104	Failed

Natural join

```
SELECT *
FROM PassengersAtAirport
NATURAL JOIN Booking;
```

passenger_id	first_name	last_name	passport_number	contact_number
1	Rahul	Varma	X1234567	+91-9000000001
2	Priya	Mehta	Y9876543	+91-9000000002
4	Sneha	Kapoor	Z5555555	+91-9000000004
5	Rohan	Patil	A1111111	+91-9000000005

email	airport_id	check_in_time	ticket_price	booking_id	flight_id	payment_
rahul.sharma@example.com	1	2025-02-27 08:30:00	5000.50	1	101	Complete
priya.mehta@example.com	2	2025-02-27 09:00:00	4200.00	2	102	Pending
sneha.kapoor@example.com	3	2025-02-27 10:15:00	3800.75	3	103	Complete
rohan.patil@example.com	4	2025-02-27 11:30:00	3500.25	4	104	Failed

Semi join

```
SELECT *
FROM PassengersAtAirport p
WHERE EXISTS (
    SELECT 1
    FROM Booking b
    WHERE p.passenger_id = b.passenger_id
);
```

passenger_id	first_name	last_name	passport_number	contact_number
1	Rahul	Varma	X1234567	+91-9000000001
2	Priya	Mehta	Y9876543	+91-9000000002
4	Sneha	Kapoor	Z5555555	+91-9000000004
5	Rohan	Patil	A1111111	+91-9000000005

email	airport_id	check_in_time	ticket_price
rahul.sharma@example.com	1	2025-02-27 08:30:00	5000.50
priya.mehta@example.com	2	2025-02-27 09:00:00	4200.00
sneha.kapoor@example.com	3	2025-02-27 10:15:00	3800.75
rohan.patil@example.com	4	2025-02-27 11:30:00	3500.25

Anti join

```sql
SELECT *
FROM PassengersAtAirport p
WHERE NOT EXISTS (
    SELECT 1
    FROM Booking b
    WHERE p.passenger_id = b.passenger_id
);
```

passenger_id	first_name	last_name	passport_number	contact_number	email
3	Amit	Verma	B2222222	+91-9000000003	amit.verma@example.com

airport_id	check_in_time	ticket_price
5	2025-02-27 12:45:00	3100.00

Theta join

```
SELECT p.passenger_id, p.first_name, p.last_name, p.ticket_price, b.flight_id
FROM PassengersAtAirport p
JOIN Booking b
ON p.passenger_id = b.passenger_id
WHERE p.ticket_price > (b.flight_id * 30);
```

passenger_id	first_name	last_name	ticket_price	flight_id
1	Rahul	Varma	5000.50	101
2	Priya	Mehta	4200.00	102
4	Sneha	Kapoor	3800.75	103

Division join

```
SELECT p.passenger_id, p.first_name, p.last_name
FROM PassengersAtAirport p
WHERE NOT EXISTS (
    SELECT f.flight_id
    FROM Booking f
    WHERE NOT EXISTS (
        SELECT b.flight_id
        FROM Booking b
        WHERE b.passenger_id = p.passenger_id AND b.flight_id = f.flight_id
    )
);
```

passenger_id	first_name	last_name
1	Rahul	Varma

7. Conclusion

The **Flight reservation management system** effectively addresses the core requirements of managing flight operations by leveraging a well-designed relational database. The implementation of primary and foreign keys ensures data integrity, while advanced SQL queries facilitate comprehensive data retrieval and reporting. PL/SQL functions and triggers automate critical business processes, enhancing operational efficiency and reducing manual intervention.

Overall, the system offers a scalable and robust solution for flight reservation management, with the potential for future enhancements such as integrating

with web applications, implementing user authentication, and expanding reporting capabilities.