Airline Reservation and Management System (Version - 2)

Problem Statement:

SkyHigh Airlines, in response to increasing demand, is implementing a comprehensive airline reservation and management system to handle passenger bookings, optimize seating, and track flight operations. The system should manage flights, handle passenger details, ensure seat allocations based on preferences, and allow passengers to earn and redeem rewards through a distinct loyalty program. Additionally, the system will provide updates on flight status to enhance user experience and operational efficiency.

Each flight has unique details, including the destinations it serves, the times it departs and arrives, the classes of travel available (e.g., Economy, Business, First). Every booking is uniquely identified and connects passengers to their reserved seats while linking the reservation to the corresponding flight. Passengers are central to this system, and the system should maintain their personal information and preferences for seating (e.g., window or aisle).

Each flight operates with a detailed record of available and occupied seats. The system must ensure that every seat on a flight is accounted for, whether it is available for booking, or booked. The seat assignment status should dynamically update as passengers confirm or cancel their bookings.

SkyHigh Airlines also runs a dedicated loyalty program to reward frequent flyers. This program uniquely tracks the miles accumulated by passengers and allows them to redeem their points for exclusive services, such as free upgrades or discounts. Notably, this loyalty program operates independently of the core passenger database, ensuring focused and streamlined rewards management.

Efficient seat allocation is essential, as it ensures that passengers' seating preferences are respected, and any cancellations or changes in bookings are reflected accurately to accommodate new travelers.

Note: Define data types as per the given case study (enum,date,etc).

Deliverables:

Q1) Draw an E-R Diagram for the Airline Reservation and Management System, illustrating components given in the above case study and infer the relationships and constraints within the system.

Q2) Convert the E-R Diagram into a relational schema, detailing constraints to provide a clear view of the database design.

Q3) Develop the database with all relevant tables, establishing proper relationships and constraints. Populate each table with at least 5 records. Attach screenshots of SQL queries for table creation and the populated data.

Q4) SkyHigh Airlines operates with multiple branches in different locations. Each branch manages its seat allocation, and the central server maintains an authoritative record of all bookings. When a local branch cancels a booking (deletes a seat allocation), this deletion must reflect on the central server to ensure accurate and synchronized data. To ensure that seat allocation data is synchronized across the server and local branches, the Federated Engine is used. Demonstrate deletion of a record from the local branch and the reflection of the same on the central server using Federated Engine. Attach relevant screenshots from both machines.

Q5) SkyHigh Airlines has received feedback that passengers often need to update their seat preferences after booking a flight. To address this, the airline wants to implement a feature where passengers can change their seat preferences (e.g., from window seat to an aisle seat) at any time. Create a **stored procedure** to handle this update efficiently. The procedure should allow airline staff to specify a passenger's ID and their new seat preference, updating the passenger's record accordingly.

Attach screenshots of an example execution of this stored procedure in the submission file.

Q6) SkyHigh Airlines wants to automatically update the seat status to "Booked" when a new booking is made. This trigger should execute after a new row is inserted into the `Booking` table and update the corresponding seat status in the `Seat_Allocation` table.

Create a front-end to demonstrate the trigger's functionality and attach screenshots showing its execution.

Q7) SkyHigh Airlines wants to offer a discount for loyal passengers based on their accumulated loyalty points. Implement a function that calculates a discount percentage on ticket prices based on the total loyalty points accumulated by a passenger.

- Create a user-defined SQL function `calculate_discount` that takes:
    1. `loyalty_points` (INT) – total points accumulated from the loyalty table.
    2. `base_ticket_price` (DECIMAL) – ticket price for the flight.
    3. Returns a discounted ticket price based on loyalty points.

Use the function to display for each passenger:

- Passenger's full name
- Base ticket price (input)
- Discounted ticket price (using the function)

Q8) Demonstrate transaction isolation (Level-serializable) by implementing the following:

1. SkyHigh Airlines has an internal system where customer service representatives can update passengers' seat classes based on seat availability or loyalty program upgrades. Due to a system lag or simultaneous updates, two representatives attempt to change the class of a seat for the same passenger at the same time. This could lead to inconsistent data if not handled properly.

   You are asked to demonstrate the impact of concurrent updates on the `Passengers` table and how transaction isolation can prevent issues like race conditions.

Q9) Below are the queries to create and populate a simple employee database. Please copy and paste these queries to set up the database, and then use them to answer the following question based on the provided code.

```
-- Employee table
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100)
);

-- Employee_hierarchy table
CREATE TABLE employee_hierarchy (
    employee_id INT,
    manager_id INT,
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);

-- Insert employees
INSERT INTO employees (employee_id, employee_name) VALUES
(1, 'Alice'),
(2, 'Bob'),
(3, 'Charlie');


-- Insert employee hierarchy (relationships between employees)
INSERT INTO employee_hierarchy (employee_id, manager_id) VALUES
(2, 1),  -- Bob reports to Alice
(3, 1);  -- Charlie reports to Alice
```

Write a **recursive query** that can be used to find all employees who report to Alice directly or indirectly. Paste screenshots of the output in the submission report.