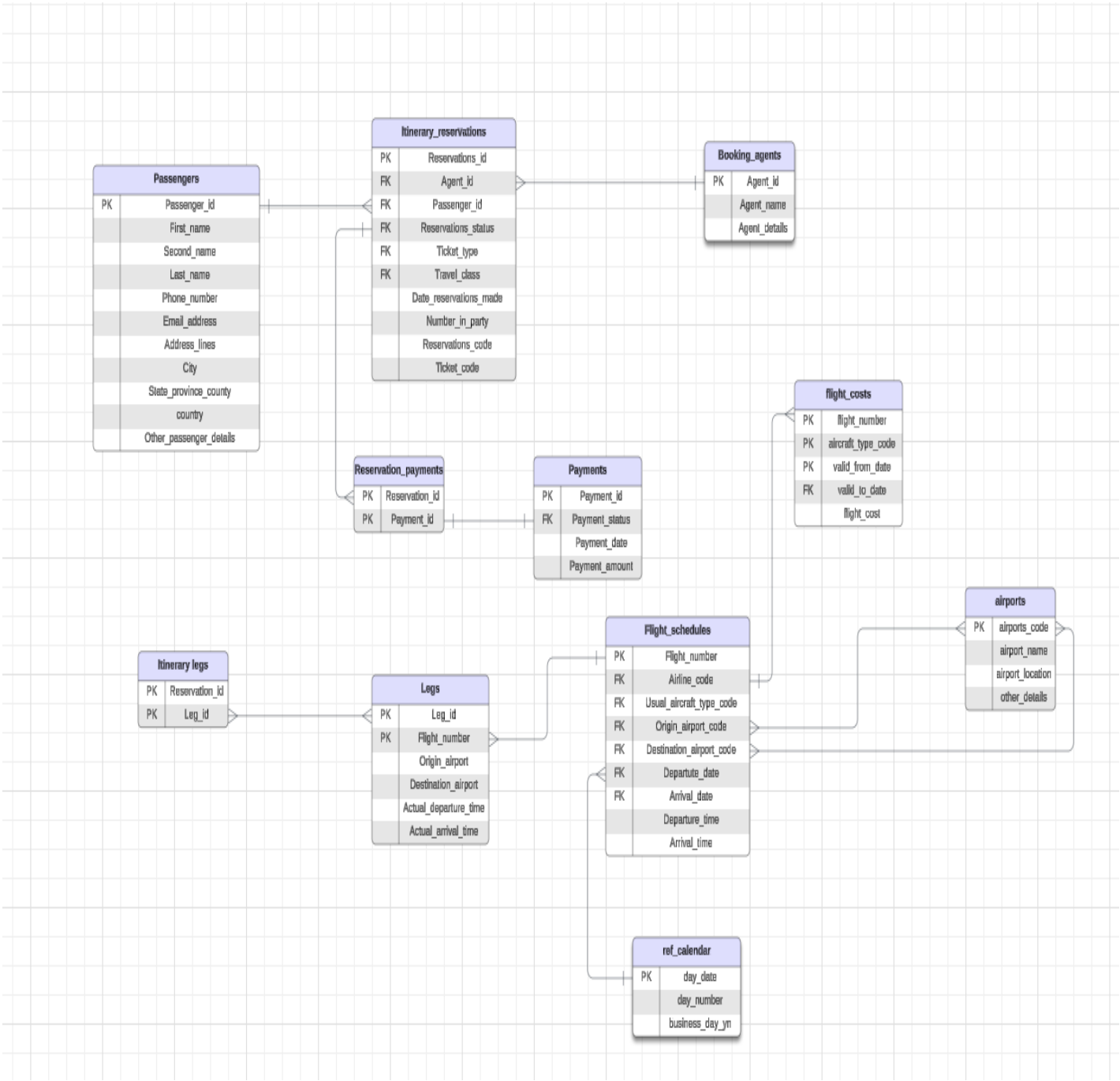
Beutah Monyoncho

PROJECT 1: SQL PROJECT

PART 1: DATA MODEL



PART 2: IMPLEMENTATION

DDL CREATE STATEMENTS

```
--Create Airports Table
CREATE TABLE Airports (
        Airport_code VARCHAR(10) PRIMARY KEY,
        Airport_name VARCHAR(100) NOT NULL,
        Airport_location VARCHAR(100) NOT NULL,
        Other_details TEXT
);
---Table : Ref_calendar
CREATE TABLE Ref_calendar (
        Day_date DATE PRIMARY KEY,
        Day_number INT NOT NULL,
        Business_day_yn BOOLEAN NOT NULL DEFAULT FALSE
);
---Table : Booking_agents
CREATE TABLE Booking_agents (
        Agent_id INT PRIMARY KEY,
        Agent_name VARCHAR(255) NOT NULL,
        Agent_details TEXT
);
--Table : Passengers
CREATE TABLE Passengers (
        Passenger_id INT PRIMARY KEY,
        First_name VARCHAR(100) NOT NULL,
        Second_name VARCHAR(100),
        Last_name VARCHAR(100) NOT NULL,
        Phone_number VARCHAR(20),
        Email_address VARCHAR(255) UNIQUE,
        Address_lines TEXT,
        City VARCHAR(100),
        State_province_county VARCHAR(100),
        Country VARCHAR(100),
        Other_passenger_details TEXT
);
---Table : Itenerary_reservations
CREATE TABLE Itenerary_reservations (
        Reservations_id INT PRIMARY KEY,
        Agent_id INT,
        Passenger_id INT NOT NULL,
        Reservations_status_code VARCHAR(50),
        Ticket_type_code VARCHAR(50),
        Travel_class_code VARCHAR(50),
        date_reservations_made DATE NOT NULL,
        number_in_party INT,
        FOREIGN KEY (Agent_id) REFERENCES Booking_agents (Agent_ID) ON DELETE
```

```
SET NULL,
        FOREIGN KEY (Passenger_id) REFERENCES Passengers(passenger_id) ON
DELETE CASCADE
);
-----Table : Flight_schedules
CREATE TABLE Flight_schedules (
        Flight_number INT PRIMARY KEY,
        Airline_code VARCHAR (20) NOT NULL,
        Usual_aircraft_type_code VARCHAR(10),
        Origin_airport_code VARCHAR(10),
        Destination_airport_code VARCHAR(10) NOT NULL,
        Departure_date_time TIMESTAMP NOT NULL,
        Arrival_date_time TIMESTAMP NOT NULL,
        Seats_available INT NOT NULL DEFAULT 0 CHECK (Seats_available >=0),
        FOREIGN KEY (Origin_airport_code) REFERENCES Airports(airport_code),
        FOREIGN KEY (Destination_airport_code) REFERENCES
Airports(Airport_code)
);
---Table : Legs
CREATE TABLE Legs (
        Leg_id INT PRIMARY KEY,
        Flight_number INT NOT NULL,
        Origin_airport VARCHAR (10) NOT NULL,
        Destination_airport VARCHAR(10) NOT NULL,
        Actual_departure_time TIMESTAMP,
        Actual_arrival_time TIMESTAMP,
        FOREIGN KEY (Flight_number) REFERENCES
Flight_schedules(Flight_number),
        FOREIGN KEY (Destination_airport) REFERENCES Airports(Airport_code)
);

---Table : Flight_costs
CREATE TABLE Flight_costs (
        Flight_number VARCHAR(20) NOT NULL,
        Aircraft_type_code VARCHAR(10) NOT NULL,
        Valid_from_date DATE NOT NULL,
        Valid_to_date DATE,
        Flight_cost NUMERIC(10,2) NOT NULL,
        PRIMARY KEY (Flight_number, Aircraft_type_code, Valid_from_date),
        FOREIGN KEY (Flight_number) REFERENCES Flight_schedules
(Flight_number)
);
---Table : Itinerary_legs (Associative Table)
CREATE TABLE Itenerary_legs (
        reservation_id INT NOT NULL,
        leg_id INT NOT NULL,
        PRIMARY KEY (reservation_id, leg_id),
        FOREIGN KEY (leg_id) REFERENCES legs(leg_id)
```

```
);
---Table : Payments
CREATE TABLE Payments (
        Payment_id INT PRIMARY KEY,
        Payment_status_code VARCHAR(10) NOT NULL,
        Payment_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        Payment_amount NUMERIC(10,2) NOT NULL CHECK (Payment_amount >0)
);
---Table : Reservation_payments
CREATE TABLE Reservations_payments (
        Reservation_id INT NOT NULL,
        Payment_id INT NOT NULL,
        PRIMARY KEY (Reservation_id, Payment_id),
        FOREIGN KEY (Reservation_id) REFERENCES
Itenerary_reservations(Reservations_id) ON DELETE CASCADE,
        FOREIGN KEY (Payment_id) REFERENCES Payments(Payment_id) ON DELETE
CASCADE
);
---Amending tables with flight_number as INT instead of VARCHAR
DROP TABLE IF EXISTS Flight_costs;
ALTER TABLE Legs DROP CONSTRAINT legs_flight_number_fkey;

---Recreate Dropped Tables
CREATE TABLE Flight_schedules (
    Flight_number VARCHAR(20) PRIMARY KEY,
    Airline_code VARCHAR(10) NOT NULL,
    Usual_aircraft_type_code VARCHAR(10),
    Origin_airport_code VARCHAR(10) NOT NULL,
    Destination_airport_code VARCHAR(10) NOT NULL,
    Departure_date_time TIMESTAMP NOT NULL,
    Arrival_date_time TIMESTAMP NOT NULL
);

CREATE TABLE Legs (
    Leg_id INT PRIMARY KEY,
    Flight_number VARCHAR(20) NOT NULL,
    Origin_airport VARCHAR(10) NOT NULL,
    Destination_airport VARCHAR(10) NOT NULL,
    Actual_departure_time TIMESTAMP,
    Actual_arrival_time TIMESTAMP,
    FOREIGN KEY (Flight_number) REFERENCES Flight_schedules(Flight_number) ON
DELETE CASCADE
);
CREATE TABLE Flight_costs (
    Flight_number VARCHAR(20) NOT NULL,
    Aircraft_type_code VARCHAR(10) NOT NULL,
    Valid_from_date DATE NOT NULL,
    Valid_to_date DATE,
```

```
    Flight_cost DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (Flight_number, Aircraft_type_code, Valid_from_date),
    FOREIGN KEY (Flight_number) REFERENCES Flight_schedules(Flight_number)
);
----Drop and recreate Flight_costs Table
DROP TABLE IF EXISTS Flight_costs CASCADE;

CREATE TABLE Flight_costs (
    Flight_number INT NOT NULL,
    Aircraft_type_code VARCHAR(10) NOT NULL,
    Valid_from_date DATE NOT NULL,
    Valid_to_date DATE,
    Flight_cost DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (Flight_number, Aircraft_type_code, Valid_from_date),
    FOREIGN KEY (Flight_number) REFERENCES Flight_schedules(Flight_number) ON
DELETE CASCADE
);
```

SQL QUERIES/VIEWS

```
---query to get a passenger's itenerary
SELECT IR.Reservations_id, P.First_name, P.Last_name, L.Flight_number,
       A1.airport_name AS Origin, A2.airport_name AS Destination,
       F.Departure_date_time, F.Arrival_date_time
FROM Itenerary_reservations IR
JOIN Passengers P ON IR.Passenger_id = P.Passenger_id
JOIN Itenerary_legs IL ON IR.Reservations_id = IL.Reservation_id
JOIN Legs L ON IL.Leg_id = L.Leg_id
JOIN Flight_schedules F ON L.Flight_number = F.Flight_number
JOIN Airports A1 ON F.Origin_airport_code = A1.airport_code
JOIN Airports A2 ON F.Destination_airport_code = A2.airport_code
WHERE P.Passenger_id = 1232234;

--Get All Customers who have seats reserved on a given flight
--View Customers_on_Flight
CREATE VIEW Customers_On_Flight AS
SELECT P.Passenger_id, P.First_name, P.Last_name, P.Email_address,
L.Flight_number
FROM Itenerary_reservations IR
JOIN Passengers P ON IR.Passenger_id = P.Passenger_id
JOIN Itenerary_legs IL ON IR.Reservations_id = IL.Reservation_id
JOIN Legs L ON IL.Leg_id = L.Leg_id;

---View: Flights_By_Airport
CREATE VIEW Flights_By_Airport AS
SELECT F.Flight_number, F.Airline_code, A1.airport_name AS Origin,
       A2.airport_name AS Destination, F.Departure_date_time,
```

```
F.Arrival_date_time
FROM Flight_schedules F
JOIN Airports A1 ON F.Origin_airport_code = A1.airport_code
JOIN Airports A2 ON F.Destination_airport_code = A2.airport_code;

---View : Flight_Schedule_View
CREATE VIEW Flight_Schedule_View AS
SELECT Flight_number, Airline_code, Origin_airport_code,
Destination_airport_code,
       Departure_date_time, Arrival_date_time
FROM Flight_schedules;

--View: Flight_Status : Get all flights whose arrival and departure times are
on time/delayed
CREATE VIEW Flight_Status AS
SELECT L.Flight_number, A1.airport_name AS Origin, A2.airport_name AS
Destination,
       L.Actual_departure_time, L.Actual_arrival_time,
       F.Departure_date_time, F.Arrival_date_time,
       CASE
           WHEN L.Actual_departure_time > F.Departure_date_time THEN
'Delayed'
           WHEN L.Actual_arrival_time > F.Arrival_date_time THEN 'Delayed'
           ELSE 'On Time'
       END AS Status
FROM Legs L
JOIN Flight_schedules F ON L.Flight_number = F.Flight_number
JOIN Airports A1 ON F.Origin_airport_code = A1.airport_code
JOIN Airports A2 ON F.Destination_airport_code = A2.airport_code;

--To check flight delays:
SELECT * FROM Flight_Status WHERE Status = 'Delayed';

--View : Flight_Sales
CREATE VIEW Flight_Sales AS
SELECT L.Flight_number, SUM(P.Payment_amount) AS Total_Sales
FROM Reservations_payments RP
JOIN Payments P ON RP.Payment_id = P.Payment_id
JOIN Itenerary_reservations IR ON RP.Reservation_id = IR.Reservations_id
JOIN Itenerary_legs IL ON IR.Reservations_id = IL.Reservation_id
JOIN Legs L ON IL.Leg_id = L.Leg_id
GROUP BY L.Flight_number;
--To get sales for a flight:
SELECT * FROM Flight_Sales WHERE Flight_number = '12346';
```

PART 3

Performance constraints that Customers will potentially face if you do not create Views for the queries described in the CUSTOMER FUNCTIONS include encountering complex queries while they might be non-technical. VIEWS simplify complex queries by breaking them down into smaller, more manageable parts (aspiringyouths.com). Second, by not using VIEWS, the customers will take longer time to retrieve data. VIEWS improves performance by precomputing the results of commonly used queries, reducing the amount of time it takes to retrieve data. Third, by not creating VIEWS, customers will be dealing with detailed data which may take their time to access. VIEWS help to present only the aggregated data and hide the detailed data from users.

Querying a VIEW can introduce some performance overhead, especially when a view is built on other views or involves extensive computations like joins or aggregates (coursehorse.com). In this case, standard queries on base tables will often yield faster results because they require less processing overhead.

Do you think query performance will scale as the database scales up? As data volumes grow and databases become more complex, poorly optimized queries can significantly slow down applications, reduce productivity, and even cause downtime (sql-creator.com). However, with some strategies, you can dramatically improve query efficiency, speed up data retrieval, and enhance overall database performance.

You can alleviate any potential scaling problems with DB by employing the right strategies that will make a significant difference in performance, reliability, and cost-efficiency. Sharding is one of the techniques for enhancing database scalability by distributing data across multiple servers (pingcap.com). This method involves partitioning a large database into smaller, more manageable pieces known as shards. Each shard operates as an independent database, allowing for parallel processing and improved performance. Second technique is replication. Replication involves creating copies of the database and distributing them across multiple servers. This method enhances data availability and fault tolerance by ensuring that data is replicated and accessible even if one server fails. Third technique is Caching. Caching is a technique used to temporarily store frequently accessed data in a high-speed storage layer, reducing the load on the primary database and speeding up data retrieval times.

First-normal form RDBMS is not best suited for a highly transactional web application because of scalability challenges as managing high transaction volumes can be a challenge as per geeksforgeeks.org (2025). Also, it is complex in design and maintenance as ensuring proper normalization while avoiding excessive complexity can be time-consuming. Third, its fixed schema can be a limitation in a fast-evolving environments.

REFERENCES

1. AspiringYouths (2024) *Advantages and Disadvantages of views in SQL*. Accessed at [https://aspiringyouths.com/advantages-disadvantages/views-in-sql/.]
2. GeeksforGeeks (2025) *RDBMS benefits and limitations*. Accessed at [https://www.geeksforgeeks.org/rdbms-benefits-and-limitations/.]
3. *Optimizing SQL Performance: Tips for faster queries and better databases*. Accessed at [https://sql-creator.com/blog/optimizing-sql-performance-tips-for-faster-queries-and-better-databases.]
4. PingCAP (2024) *Best practices for solving database scaling Problems*. Accessed at [https://www.pingcap.com/article/best-practices-for-solving-database-scaling-problems/.]