Auriana Anderson

Jeremiah Lowhorn

SQL/NOSQL

February 16, 2025

Project 1

Part 2: Implementation

```
create schema if not exists Airlines;


create table if not exists Airlines.Booking_Agents(
Agent_id INT primary key,
Agent_name VARCHAR(255),
Agent_details TEXT
);



create table Airlines.Passengers(
Passenger_id INT primary key,
First_name VARCHAR(150) not null,
Second_name VARCHAR(150),
Last_name VARCHAR(150) not null,
Phone_number VARCHAR(20),
Email_address VARCHAR(255) unique not null,
Address_lines VARCHAR(255),
City VARCHAR(150),
State_province_county VARCHAR(150),
Country VARCHAR(150),
Other_passenger_details TEXT
```

```
);

create table Airlines.Reservation_Status(
Reservation_status_code VARCHAR(15) primary key,
Description TEXT not null
);

create table Airlines.Ticket_type(
Ticket_type_code VARCHAR(15) primary key,
Description TEXT not null
);

create table Airlines.Ticket_Class(
Travel_class_code VARCHAR(15) primary key,
Description TEXT not null
);

create table Airlines.Itinerary_reservations(
Reservation_id INT primary key,
Agent_id INT null,
Passenger_id INT not null,
Reservation_status_code VARCHAR(15) not null,
Ticket_type_code VARCHAR(15) not null,
Travel_class_code VARCHAR(15) not null,
Date_reservation_made DATE not null,
Number_in_party int not null,
foreign key (Agent_id) references Airlines.Booking_Agents(Agent_id),
```

```
foreign key (Passenger_id) references Airlines.Passengers(Passenger_id),
foreign key (Reservation_status_code) references
Airlines.Reservation_Status(Reservation_status_code),
foreign key (Ticket_type_code) references Airlines.Ticket_type(Ticket_type_code),
foreign key (Travel_class_code) references Airlines.Ticket_Class(Travel_class_code)
);

create table Airlines.Payment_status(
Payment_status_code VARCHAR(15) primary key,
Description TEXT not null
);

create table Airlines.Payments(
Payment_id INT primary key,
Payment_status_code VARCHAR(15) not null,
Payment_date DATE not null,
Payment_amount DECIMAL(10,2) not null,
foreign key (Payment_status_code) references
Airlines.Payment_status(Payment_status_code)
);

create table Airlines.Reservation_payments(
Reservation_id INT,
Payment_id INT,
primary key (Reservation_id, Payment_id),
foreign key (Reservation_id) references Airlines.Itinerary_reservations(Reservation_id),
foreign key (Payment_id) references Airlines.Payments(Payment_id)
```

```
);

create table Airlines.Airlines(
Airline_code VARCHAR(15) primary key,
Airline_name VARCHAR(255),
Description TEXT not null
);

create table Airlines.Aircraft_type(
Aircraft_type_code VARCHAR(15) primary key,
Description TEXT not null
);

create table Airlines.Ref_calendar(
Day_Date DATE primary key,
Day_number INT not null,
Business_day_yn VARCHAR(5) not null
);

create table Airlines.Airports(
airport_code VARCHAR(15) primary key,
airport_name VARCHAR(255),
airport_location TEXT,
other_details TEXT
);

create table Airlines.Flight_schedules(
```

```
Flight_number VARCHAR(20) primary key,
Airline_code VARCHAR(15) not null,
Usual_aircraft_type_code VARCHAR(15) not null,
Origin_airport_code VARCHAR(15) not null,
Destination_airport_code VARCHAR(15) not null,
Departure_date_time TIMESTAMP not null,
Arrival_date_time TIMESTAMP not null,
Day_Date DATE not null,
foreign key (Airline_code) references Airlines.Airlines(Airline_code),
foreign key (Usual_aircraft_type_code) references
Airlines.Aircraft_type(Aircraft_type_code),
foreign key (Origin_airport_code) references Airlines.Airports(airport_code),
foreign key (Destination_airport_code) references Airlines.Airports(airport_code),
foreign key (Day_Date) references Airlines.Ref_calendar(Day_Date)
);

create table Airlines.Legs(
Leg_id INT primary key,
Flight_number VARCHAR(20) not null,
Origin_airport VARCHAR(15) not null,
Destination_airport VARCHAR(15) not null,
Actual_departure_time timestamp not null,
Actual_arrival_time timestamp not null,
foreign key (Flight_number) references Airlines.Flight_schedules(Flight_number),
foreign key (Origin_airport) references Airlines.Airports(airport_code),
foreign key (Destination_airport) references Airlines.Airports(airport_code)
);
```

```
create table Airlines.Itinerary_legs(
Reservation_id INT,
Leg_id INT,
primary key (Reservation_id, Leg_id),
foreign key (Reservation_id) references Airlines.Itinerary_reservations(Reservation_id),
foreign key (Leg_id) references Airlines.Legs(Leg_id)
);


create table Airlines.Flight_costs(
Flight_number VARCHAR(20),
Aircraft_type_code VARCHAR(15) not null,
valid_from_date DATE not null,
valid_to_date DATE not null,
flight_cost DECIMAL(10,2) not null,
primary key (Flight_number, aircraft_type_code, valid_to_date),
foreign key (Flight_number) references Airlines.Flight_schedules(Flight_number),
foreign key (Aircraft_type_code) references Airlines.Aircraft_type(Aircraft_type_code)
);
```

---Write the SQL queries/views for the highlighted requirements in section 2.3

```
--User Function-
---       View his itinerary:
create view Customer_Itinerary as
select
p. Passenger_id,
p. First_name,
p. Last_name,
ir.Reservation_id,
fs.Flight_number,
fs.Origin_airport_code,
fs.Destination_airport_code,
fs.Departure_date_time,
fs.Arrival_date_time,
ir.Number_in_party
from Airlines.Passengers p
join Airlines.Itinerary_reservations ir
on  p.Passenger_id = ir.Passenger_id
join Airlines.Itinerary_legs il
on ir.Reservation_id = il.Reservation_id
join Airlines.Legs l
on il.Leg_id = l.Leg_id
join Airlines.Flight_schedules fs
on l.Flight_number = fs.Flight_number;

--Employee Function-
```

---Get all customers who have seats reserved on a given flight:

```
create view Customer_attendance as
select
p. Passenger_id,
p. First_name,
p. Last_name,
ir.Reservation_id,
fs.Flight_number
from Airlines.Passengers p
join Airlines.Itinerary_reservations ir
on  p.Passenger_id = ir.Passenger_id
join Airlines.Itinerary_legs il
on ir.Reservation_id = il.Reservation_id
join Airlines.Legs l
on il.Leg_id = l.Leg_id
join Airlines.Flight_schedules fs
on l.Flight_number = fs.Flight_number
where fs.Flight_number = 'flight_B613';
```

---Get all flights for a given airport:

```
create view Airport_flights as
select
fs.Flight_number,
fs.Origin_airport_code,
fs.Destination_airport_code,
```

```
fs.Departure_date_time
from Airlines.Flight_schedules fs
where fs.Origin_airport_code = 'LAX'
or fs.Destination_airport_code = 'LAX';

---View flight schedule:

create view All_flight_schedules as
select
fs.Flight_number,
ap1.airport_name as Origin_airport,
ap2.airport_name as Destination_airport,
fs.Departure_date_time,
fs.Arrival_date_time
from Airlines.Flight_schedules fs
join Airlines.Airports ap1
on fs.Origin_airport_code = ap1.airport_code
join Airlines.Airports ap2
on fs.Destination_airport_code = ap2.airport_code
order by fs.Departure_date_time;

---Get all flights whose arrival and departure times are on time/delayed:

create view flight_delay_status as
select
fs.Flight_number,
fs.Departure_date_time,
```

```
fs.Arrival_date_time,
case
                when fs.Departure_date_time > now() then 'scheduled'
                when fs.Arrival_date_time < now() then 'completed'
                else 'delayed'
        end as flight_delay_status
from Airlines.Flight_schedules fs

---Calculate total sales for a given flight:

create view flight_sales as
select
fs.Flight_number,
sum(p.Payment_amount) as total_sales
from Airlines.Payments p
join Airlines.Reservation_payments rp
on p.Payment_id = rp.Payment_id
join Airlines.Itinerary_reservations ir
on rp.Reservation_id = ir.Reservation_id
join Airlines.Itinerary_legs il
on ir.Reservation_id = il.Reservation_id
join Airlines.Legs l
on il.Leg_id = l.Leg_id
join Airlines.Flight_schedules fs
on l.Flight_number = fs.Flight_number
where fs.Flight_number = 'flight_B613'
group by fs.Flight_number;
```

Part 3:Discussion

One of the biggest performance constraints that I see the customer's potentially facing, when necessary views aren't creating are slow processing times. Customers who are part of an airline reservation system need to have real time data for their flights and so do the employees working for the airlines and the airport. Without having pre-defined views, customers would essentially be retrieving the data they need from huge data sets from several different tables, which may have more information than they need. The customer would also need to be able to do complex querying to be able to use this type of system which I don't see customers at an airport or airline customers (Airline employees' and airport employee's) being happy about. Imagine how fast and efficient an airport needs to be. At the kiosks, customers need to be able to quickly enter their information so that they can print their tickets. Security needs to confirm a customer's identity and if flights are valid. Flight attendants need to be able to confirm customer identity, seat, flight status and more. These are just a few processes that happen within the airport, and they need to happen quickly to process thousands of people a day who are in and out of the airport.

Now that we have thought of the actual processes that happen in the airport, that you may not think about as a customer, think about how fast these things must happen to keep the system going. There are lots of people going in and out of the airport which means the data is constantly getting larger and larger and that also means there are more and more users that will need to use the system. Now, think of the system without any views and with high traffic, this would put a huge strain on the performance of the system if there were lots of users trying to make complex queries at the same time, when this could have been minimized by creating views. This could even break the system in a sense during high volume times.

This creates huge overhead for the database because again, there are so many processes that happen at the airport and these are repetitive processes. Every time there are any requests, there is essentially complex querying involving joins, filters, and sorting across multiple tables. This means that there is constantly going to be high CPU usage which means the airport will need more expensive systems to keep up with this especially since many employees may be repeating the same process at the same time. The inefficiency here is going to become more apparent as the database gets larger because gathering necessary info for each role will be slow, querying will become slower for repetitive tasks, and the servers over time will take a huge hit. Once the servers need scaling, the prices will start rising for the airline/airport.

Some ways to alleviate any potential scaling problems with the database may be caching. I think this would be a huge help for the data that needs constant retrieval like

gaining passenger information for tickets. Caching works by creating a high-speed temporary store for data that is constantly accessed to help reduce load on the primary database. This would help with quick data retrieval. It can also reduce the load on the database which in turn would increase database performance and response times. Another option could be sharding, which is basically splitting the data in a database up across different servers. This could be done by flight number or terminal so that the data is basically in subsets. This would greatly increase read and write performance and make the database more scalable because you can keep adding more shards. One last way could be indexing which basically means the user can find info without looking through the entire dataset and 'jumping' directly to the relevant section they need. These are typically created using hash tables or B-Trees. It reduces disk reads, makes looking through sorted data more efficient, and can increase overall querying performance. This would be a great method to use on an airport database as well because you can use it on columns that are frequently queried and reduce costs by not having to scan through the entire dataset every time.

The first normal form aims to reduce data redundancy, increase organization and overall integrity. The rules are: Each column must contain single values, all values in a column are of the same type, there isn't an order of data, and column names must be unique. These guidelines would create a need for more complex querying which I don't think would necessarily be ideal for an airline reservation system. I think taking on a hybrid system would be much better suited for an airline reservation system and any other highly transactional web application because you can be selective about what is normalized and what isn't. I think using SQL is better for some of the data that needs more structure and is more transactional. This could be used for passenger info, reservations, payments, flights, and flight schedules. I think using NoSQL would be better for flight status, flight searches, and to create customer profiles (like how social media apps do). I think this is best because it keeps data integrity and performance without putting a huge strain on the database.

Views not being implemented will significantly decrease the overall performance of the database. This would in turn create lots of overhead that could have been reduced with the implementation of views. It would also increase the need for excess optimization methods and tools such as caching, sharding, or indexing. While the first normal form helps with data structure and integrity, highly transactional web applications often require a hybrid system.

Works Cited

Ashour, Mohamad. “How SQL Views Are Important in the Role of Data Analysis?” *Medium*, 25 June 2023, medium.com/@mohamad.ashour203/how-sql-views-are-important-in-the-role-of-data-analysis-a5e0b74b25ec.

“Best Practices for Solving Database Scaling Problems.” *TiDB*, 2024, www.pingcap.com/article/best-practices-for-solving-database-scaling-problems/.

Choudhury, Sid. “How Data Sharding Works in a Distributed SQL Database.” *Yugabyte*, YugabyteDB, 6 June 2019, www.yugabyte.com/blog/how-data-sharding-works-in-a-distributed-sql-database/. Accessed 16 Feb. 2025.

DbVis Software AB. “DbVisualizer.” *DbVisualizer*, DbVis Software AB, 7 Aug. 2023, www.dbvis.com/thetable/efficiently-creating-and-managing-views-in-sql/.

Goyal, Anil. “Strategies for Scaling Databases: A Comprehensive Guide.” *Medium*, 28 Mar. 2024, medium.com/@anil.goyal0057/strategies-for-scaling-databases-a-comprehensive-guide-b69cda7df1d3.

Upadhyay, Mithlesh . “First Normal Form (1NF).” *GeeksforGeeks*, 30 July 2019, www.geeksforgeeks.org/first-normal-form-1nf/.

---. “Third Normal Form (3NF) - GeeksforGeeks.” *GeeksforGeeks*, 31 July 2019, www.geeksforgeeks.org/third-normal-form-3nf/.