Data Information:

The data is obtained from Maven Analytics (Free Data Sets & Dataset Samples, 2020). The following data is for public use i.e., it is licenced public.

a) There are total 4 tables that the proposed database contains:

i) flights: The table flights contain 11 entities. The entities are: AIRLINE, TAIL_NUMBER, ORIGIN_AIRPORT, DESTINATION_AIRPORT, DEPARTURE_TIME, DEPARTURE_DELAY, WHEELS_OFF, SCHEDULED_ARRIVAL, CANCELLATION_REASON
(Free Data Sets & Dataset Samples, 2020). This table has about 500 rows.

ii) airports: The table airport has 7 different entities. The entities are: IATA_CODE, STATE (Free Data Sets & Dataset Samples, 2020). This table 321 rows.

iii) airlines: The table airlines have 2 entities named IATA_CODE and AIRLINE (Free Data Sets & Dataset Samples, 2020). This table has 14 rows.

iv) cancellation_codes: The table cancellation_codes have 2 entities named CANCELLATION_REASON and CANCELLATION_DESCRIPTION (Free Data Sets & Dataset Samples, 2020). This table has 4 rows.

Business Rules:

Three business rules that I had implemented in my database are:

Referential Integrity: Foreign key constraints are used to ensure referential integrity between tables. For example, in the **flight1** table, foreign key constraints are applied to the **ORIGIN_AIRPORT**, **DESTINATION_AIRPORT**, **AIRLINE**, and **CANCELLATION_REASON** columns, referencing the respective primary keys in the **airports**, **airlines**, and **cancellation_codes** tables. This ensures that data integrity is maintained, and only valid values can be inserted into these columns no NULL values are allowed as constraints for that are also added.

Unique Identifiers: Primary keys are used to ensure that each record in a table is uniquely identified. For example, in the **airlines** table, the **IATA_CODE** column is the primary key, ensuring that each airline is uniquely identified by its IATA code. Similarly, in the **airports** table, the **IATA_CODE** column is the primary key that is used to uniquely identify airports.

Data Constraint: Data validation is obtained by keeping a **NOT NULL** data constraint. In the **flight1** table, columns **TAIL_NUMBER**, **AIRLINE**, **ORIGIN_AIRPORT**, **DESTINATION_AIRPORT**, **ARRIVAL_TIME**, **ARRIVAL_DELAY**, and **CANCELLATION_REASON** are assigned a constraint of **NOT NULL**, because the reason is that they can't have null values in their respective columns as airport name, airline name and time cannot be null. Additionally, appropriate data types are specified for each column to ensure consistency and accuracy in the stored data.

Query Explanation:

Query 1: This query helps us get specific details about flights, such as their tail number, scheduled departure and arrival times, departure delay, arrival delay, and total delay. It only selects flights where the scheduled arrival time is after 100 (which could represent a specific time frame).

Query 2: Here, we aim to gather information about flights along with the states where their origin and destination airports are located. This helps us understand the geographical context of each flight. The results are organized by the airport codes of origin and destination in descending order.

Query 3: In this query, we're interested in knowing how many flights depart from each state. By counting the number of flights departing from each state, we can get an idea of the air traffic in different regions. The results are grouped by state, giving us a clear picture of flight activity across different areas.

Query 4: This query fetches details of all flights that weren't canceled. It ensures we only see information about flights that took off. Additionally, it includes the reason for cancellation from the **cancellation_codes** table, providing more context about why certain flights were canceled.

Query 5: This set of queries introduces a view called **flight_details**, which contains information about flights departing from California. After creating the view, we retrieve and display all the flight details from it. Then, we update the **flight1** table to mark flights with a departure delay exceeding 50 minutes as canceled due to 'A' reason. Finally, we check the **flight_details** view again to see the impact of this update.

Explanation of stored procedures:

1. **DeleteFromTables Procedure**:
    - **What it Does**: This stored procedure is for deleting flights from the **flight1** table based on how delayed they were for departure and arrival.
    - **How to Use It**: We can specify it how long a delay needs to be for it to count as significant for departure **d_delay** and arrival **a_delay**. Then, the stored procedure deletes flights that didn't have delays longer than passed into it as parameters.
    - **Example Call**: If we want to delete the flights that have less than **10** minutes of departure delay and less than **20** minutes of arrival delay then we can call the procedure like this: **CALL DeleteFromTables(10, 20);**.
2. **UpdateFromTable Procedure**:
    - **What it Does**: This procedure updates the cancellation reason accordingly in the table where the departure and arrival delay are less than the one that is passed into the stored procedure as parameters.
    - **How to Use It**: The procedure updates and changes the cancellation reason to A where the departure delay is less than **d_delay** and to B where arrival delay is less than **a_delay**, which is passed as parameters into the stored procedures.
    - **Example Call**: If we call the procedure as **CALL UpdateFromTable(40,50)** then it changes the cancellation reason from anything present to A where the departure delay is less than **40** and to B where arrival delay is less than **50.**

No information will be returned by the stored procedures as it uses IN and not INOUT or OUT.

References:

Free Data Sets & Dataset Samples. (2020, September 10). *Airline Flight Delays*. Maven Analytics. https://mavenanalytics.io/data-

playground?dataStructure=2lXwWbWANQgI727tVx3DRC&page=5&pageSize=5