



FINAL PROJECT REPORT

MET CS 779 Spring 2023 A1

Yuandi Tang

tyleyd@bu.edu U65674688

MET CS 779 Final Project

Table of Contents

1. Introduction	2
2. Data warehousing Implementations	3
1) Data Requirements	3
2) ETL	3
3) ERD	3
4) Security and Compliance	7
5) Backup & Recovery	7
6) Tuning & Optimization	13
3. Conclusion	26
1) Visual & Analysis	26
2) Communication Theories	28
Bibliography	30

1. Introduction

The research on the weather and travel is very common in our everyday life. Research on travel and weather data can provide valuable insights into the relationship between these two factors and their impact on various aspects of travel. The analysis of such data can help identify patterns and trends in travel patterns, such as the most popular destinations during certain weather conditions or the effect of weather on travel patterns.

Moreover, research on travel and weather data can also help in predicting and mitigating travel disruptions caused by adverse weather conditions. Airlines, for instance, can use weather data to anticipate delays and reroute flights, reducing the impact of weather on flight schedules. Similarly, travel agencies and transportation companies can use weather data to plan travel routes and adjust schedules to avoid weather-related disruptions.

Additionally, research on travel and weather data can also help in understanding the impact of climate change on travel patterns and the tourism industry. As weather patterns change due to climate change, this can have significant impacts on travel behavior and destinations, and research in this area can help inform policymakers and businesses in adapting to these changes:

Safety: Weather conditions can have a significant impact on aviation safety. Studying weather data can help airlines and aviation authorities predict and prepare for potential weather-related hazards such as turbulence, thunderstorms, or icing conditions.

Efficiency: Analyzing airplane data can help airlines and aviation authorities identify patterns and trends that can lead to improvements in fuel efficiency, flight times, and overall operational efficiency.

Environmental impact: Researching weather and airplane data can also help to mitigate the environmental impact of aviation by reducing fuel consumption and emissions.

Improving technology: Studying weather and airplane data can help to identify areas for technological advancements and improvements in aircraft design and performance.

The 2019 Airline Delays w/Weather and Airport Detail and Travel Dataset is a comprehensive and rich dataset that provides valuable insights into airline delays, weather patterns, and airport details. This dataset was part of the Datathon 2019 on Kaggle, which aimed to encourage data enthusiasts to explore and analyze complex datasets. The dataset includes detailed information about airline delays, such as the delay time, the cause of delay, and the airline carrier. Additionally, it includes weather-related data, such as the temperature, precipitation, and wind speed, which can have a significant impact on airline delays. The airport details section provides information about the airport location, the number of runways, and the type of airport. This dataset can be used to identify patterns in airline delays, investigate the relationship between weather conditions and delays, and explore the impact of airport infrastructure on delays. In this analysis, we will examine the key insights from the dataset and explore how airlines, airports, and weather conditions influence flight delays.

2. Data warehousing Implementations

1) Data Requirements

According to Kaggle, 2019 Airline Delays w/Weather and Airport Detail dataset is a classification dataset with detailed airline, weather, airport and employment information. Optional cancellation and delay reasons for multiclass applications. The Travel Dataset - Datathon 2019 is a comprehensive dataset including the flights, hotels and users.

These two datasets are pretty structured and relational. So no data cleaning needed in the process. The **user table** is a mocked client data, it is not relevant to the research scenario and question. So it will not be upload and process in the following.

2) ETL

ETL stands for Extract, Transform, and Load. It is a process used in data warehousing and business intelligence to extract data from multiple sources, transform the data to fit a common data model, and load the data into a target database or data warehouse.

The goal of the ETL process is to ensure that data is consistent, accurate, and complete across all sources and that it is available for analysis and reporting. ETL tools and platforms automate many of the tasks involved in the process, such as data extraction, transformation, and loading, and provide a framework for managing data quality, error handling, and data lineage.

Concerning the dataset structure and real business scenario, the research question is a Build a data warehousing for the flight, hotel and the weather ,and What can we learn from flight, weather and hotel data?

The datasets are all in .csv(comma-separated values) files, so we are using copy function to load raw data in the Postgresql.

The attributes in the raw dataset are all in TEXT format, in order to further use, the data is audited as follows:

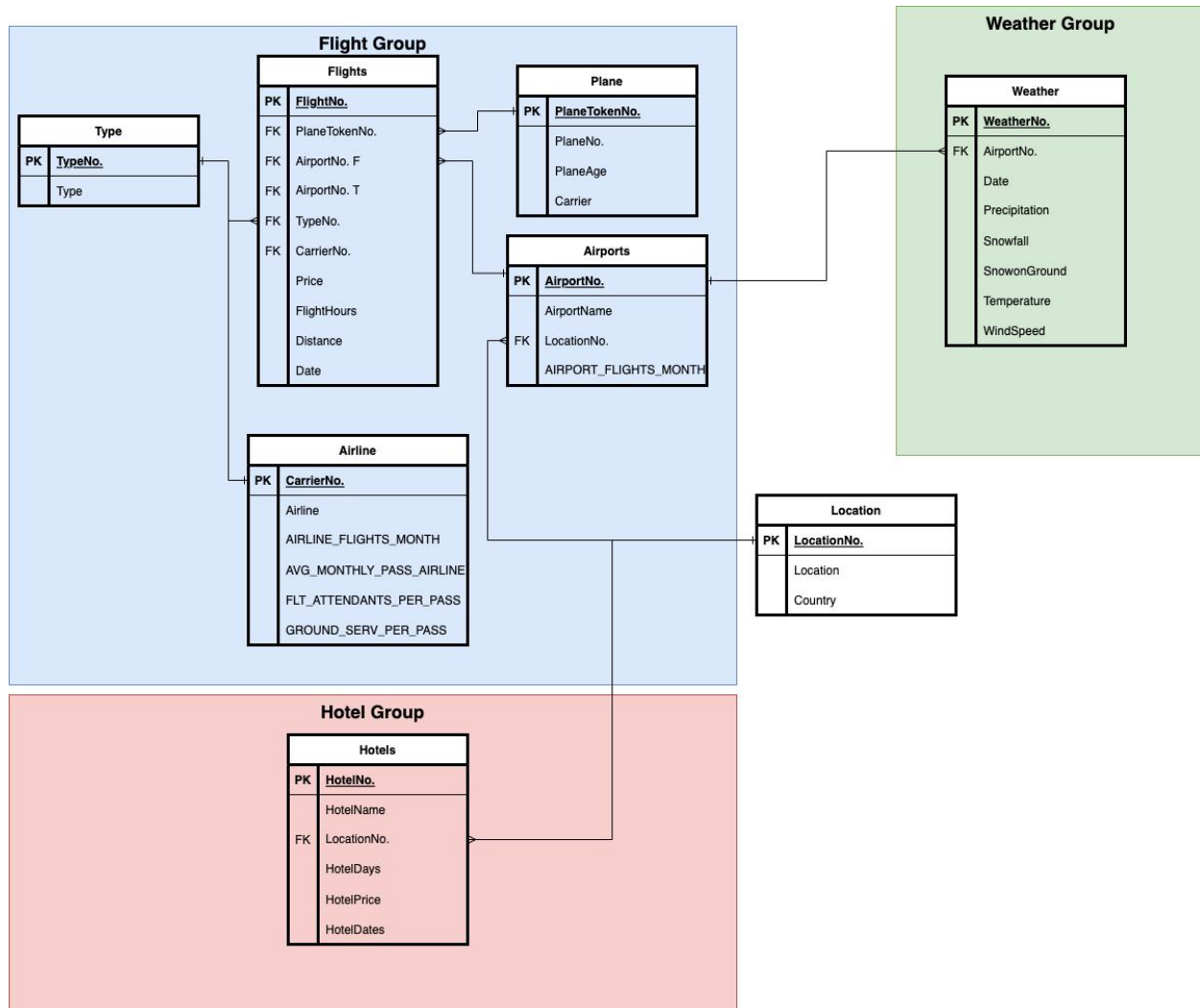
- all date DATE
- all time TIME
- flight number/weather report NUMERIC
- flight name/hotel name/name VARCHAR(25)
- PK SERIAL

3) ERD

ERD stands for Entity-Relationship Diagram. It is a graphical representation of the entities, relationships, and attributes involved in a database. ERDs are used to model the data requirements of an organization or application and to design the structure of a database. ERDs are an important tool for database design and management, enabling organizations to model

complex data structures and relationships and ensure that data is organized in a way that supports their business needs.

According the structure of raw data, the design of raw and normalized data schema.



BCNF (Boyce-Codd Normal Form) is a normal form in database normalization that addresses the issue of functional dependencies between attributes in a dataset.

A dataset is said to be in BCNF if every determinant (or candidate key) of a relation is a candidate key. In other words, a relation is in BCNF if and only if every determinant in the relation is a candidate key.

In simpler terms, BCNF requires that a dataset should not contain any overlapping functional dependencies between attributes. In a BCNF relation, every non-trivial functional dependency (where a non-prime attribute depends on a proper subset of a candidate key) must be a dependency on a candidate key.

The data has been summerized and categorized in three different data groups, one weather, one flight and one hotel. In each data group, the data has been normalized in the **BCNF form**.

After fully normalize the data, the following steps are showing the BCNF test of the most complicated transactional table: **Flights**.

Flights	
PK	<u>FlightNo.</u>
FK	PlaneTokenNo.
FK	AirportNo. F
FK	AirportNo. T
FK	TypeNo.
FK	CarrierNo.
	Price
	FlightHours
	Distance
	Date

Flights	
PK	<u>FlightNo.</u>
FK	PlaneTokenNo. AA11-1808--AA11-1809
FK	AirportNo. F 001-002
FK	AirportNo. T 003-004
FK	TypeNo. Boeing-Airbus
FK	CarrierNo. American Airlines-Delta Airlines
	Price 150-180
	FlightHours 1.5-1.67
	Distance
	Date

Looking at the table **flights**, we can see that the primary key is **planetokenno**. The other attributes in the table are **travelCode**, **userCode**, **locfrom**, **locto**, **flightType**, **price**, **time**, **distance**, **agency**, and **date**.

To determine if the table is in BCNF, we need to check if there are any functional dependencies between the non-key attributes. A functional dependency exists when the value of one attribute determines the value of another attribute. For example, if **locfrom** determines **locto**, then **locto** is functionally dependent on **locfrom**.

In this case, there don't seem to be any non-trivial functional dependencies between the non-key attributes. Therefore, the table **flights** is already in BCNF.

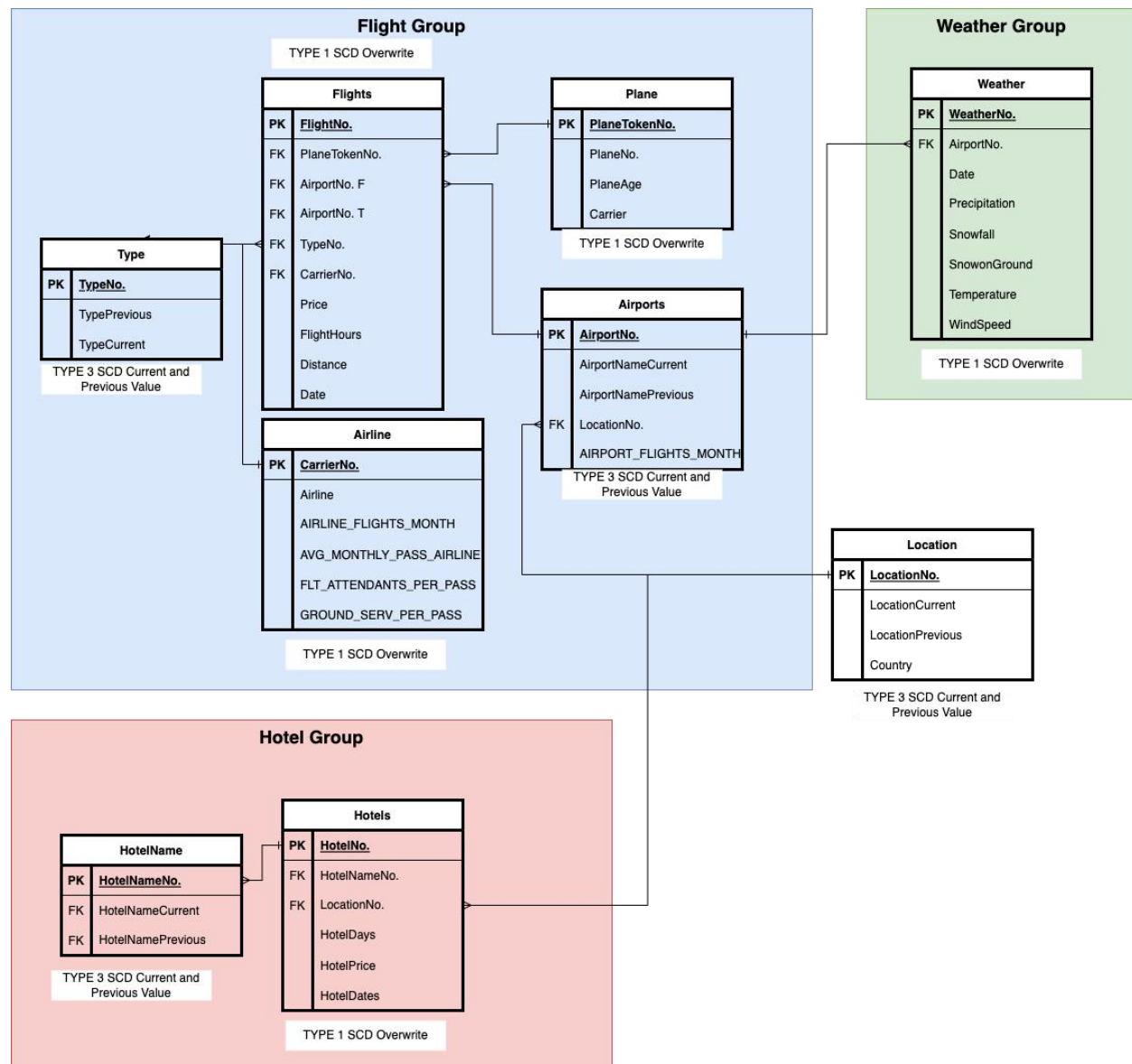
If we want to change the value of airport or type, flight or hours, we won't create duplicates and anomalies because the data is constraint on the foreign keys.

Self-Contained Data Science (SCDS) workloads, and it is a platform that allows users to run data science and machine learning workloads on distributed infrastructure, such as Hadoop or Spark clusters, without having to set up a separate environment. By using SCDS, data scientists and analysts can access and analyze large datasets more efficiently, as they can leverage the power of distributed computing to run their workloads.

SCDS can handle large-scale datasets and workloads, making it suitable for organizations dealing with big data. SCDS provides a simplified interface that abstracts the complexities of distributed computing, allowing users to focus on their data science and machine learning tasks without worrying about infrastructure setup or maintenance. SCDS allows multiple users to work on the same dataset or project simultaneously, making it easier for teams to collaborate and share their work.

SCDS provides a convenient and efficient platform for running data science and machine learning workloads on distributed infrastructure, allowing users to focus on their core tasks without the need to manage complex infrastructure.

The SCD design is shown in the following diagram:



We are using SCD1 overwrite for the hotels, airlines, plane and weather, these data is not sensitive to errors and do not need to keep the historical data. For location changes and hotel

name changes. It can easily create chaos when there is an error caused by the overwrites. It is also necessary to keep the historical data for the inspection and other uses, thus make it slow changing dimensions Type 3 to log the current and previous values.

4) Security and Compliance

GDPR stands for General Data Protection Regulation, which is a set of privacy regulations implemented by the European Union to protect the personal data of its citizens. It came into effect in May 2018 and applies to all organizations that collect or process personal data of EU citizens, regardless of where the organization is located.

To be GDPR-compliant, organizations must ensure that they collect and process personal data lawfully, transparently, and for a specific purpose. They must also obtain explicit consent from individuals for collecting their data, provide individuals with the right to access, correct, and delete their data, and ensure that the data is secure and protected from unauthorized access or breaches.

HIPAA stands for Health Insurance Portability and Accountability Act, which is a US federal law that regulates the privacy and security of protected health information (PHI). It applies to healthcare providers, health plans, and healthcare clearinghouses that transmit electronic PHI.

To be HIPAA-compliant, organizations must ensure that they protect the privacy and security of PHI, including electronic PHI (ePHI), and implement administrative, physical, and technical safeguards to protect the data from unauthorized access, use, or disclosure. They must also ensure that they obtain the necessary consent from patients for the use and disclosure of their PHI, provide patients with access to their PHI, and report any data breaches or security incidents involving PHI.

The travel data does not have confidential data on medical records or personal privacy information. The dataset is GDPR & HIPAA-compliant.

5) Backup & Recovery

Horizontal partitioning, also known as sharding, involves dividing the dataset by rows, where each partition contains a subset of rows. This technique is often used to distribute data across multiple servers or clusters to improve query performance and reduce the impact of data failures.

Vertical partitioning, on the other hand, involves dividing the dataset by columns, where each partition contains a subset of columns. This technique is often used to optimize query performance by reducing the amount of data that needs to be scanned or loaded into memory. For example, if a dataset contains both frequently accessed and rarely accessed columns, vertical partitioning can split the dataset into two partitions, with frequently accessed columns in one partition and rarely accessed columns in another.

Synchronization refers to the process of coordinating and managing access to shared resources, such as data, to avoid conflicts or inconsistencies when multiple users or processes are accessing the same resource simultaneously.

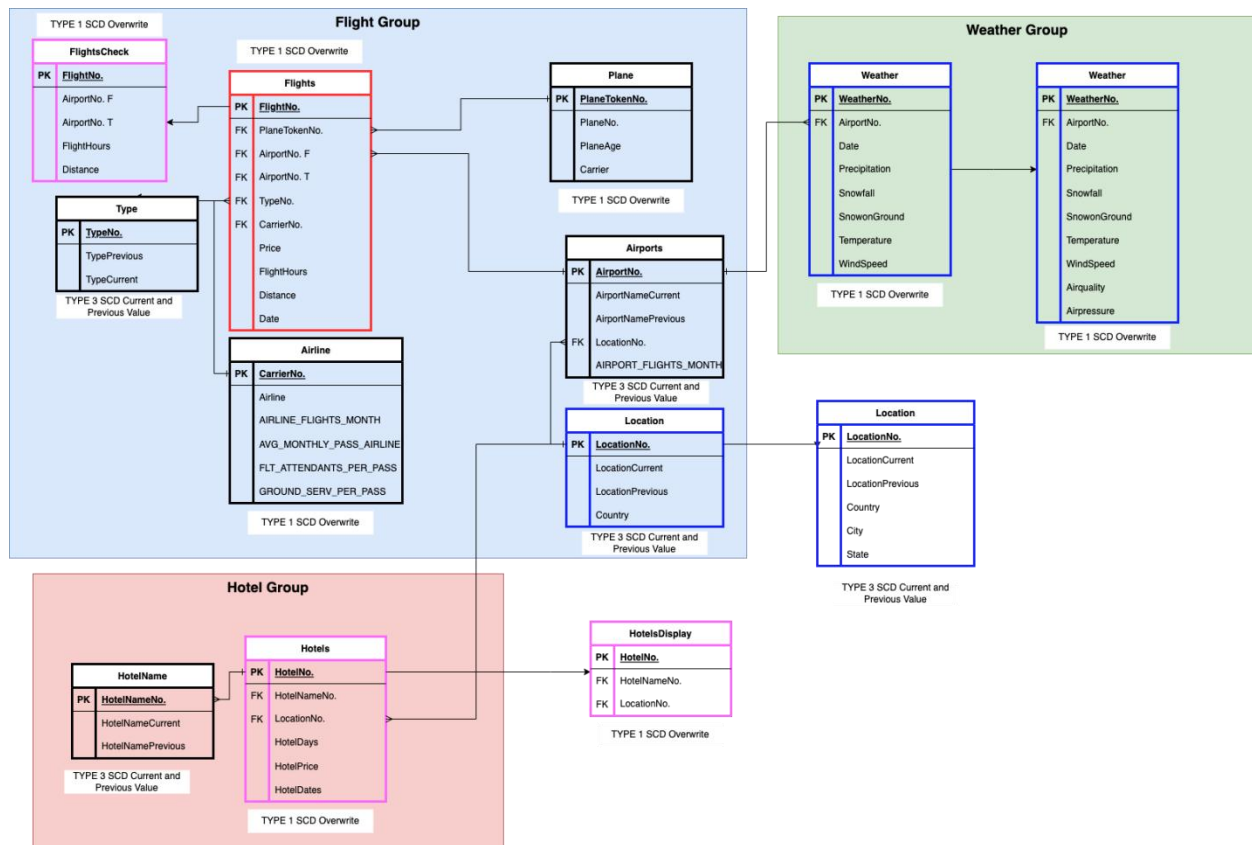
In the context of datasets, synchronization can refer to two different concepts: data synchronization and data access synchronization.

Data synchronization refers to the process of ensuring that multiple copies of the same dataset are consistent and up-to-date. This can be achieved through various techniques, such as using master-slave replication, where changes made to the master copy of the dataset are automatically replicated to slave copies, or using version control systems, where changes are tracked and merged between different versions of the dataset.

Data access synchronization, on the other hand, refers to the process of managing access to shared datasets to avoid conflicts or inconsistencies when multiple users or processes are accessing the same dataset simultaneously. This can be achieved through various techniques, such as using locking mechanisms, where a user or process acquires a lock on the dataset before accessing it, preventing other users or processes from modifying it until the lock is released.

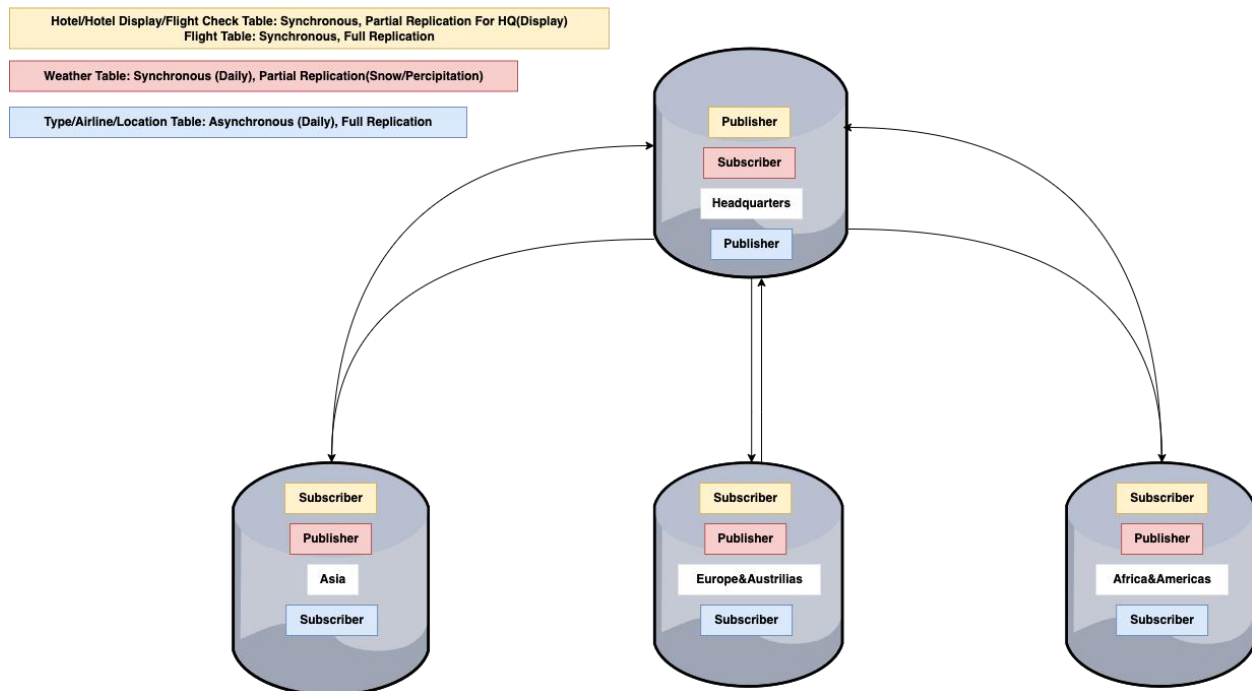
Asynchronization, or asynchronous processing, refers to a mode of operation where a program or process can continue to execute other tasks while waiting for a response or completion of a long-running task, such as data processing or retrieval. In the context of datasets, asynchronous processing can be used to improve performance and scalability by allowing multiple tasks to execute concurrently, without waiting for each other to complete.

The synchronization and partition design are shown as the following diagram:



In this diagram, Red table means Horizontal Partitioning (Shows local data for specific regions); Blue table means Vertical Partitioning (Modify attributes for regional local uses); while pink table means Vertical Partitioning (Show different tables for different department uses).

As above the hotel and flight display tables are vertical partitioning that show only some attributes in the tables for a special usage (screen display), while some locations contain State and City information, some weather data contains air quality and pressure data, that is a vertical partitioning too to use for different data servers in a global setting.

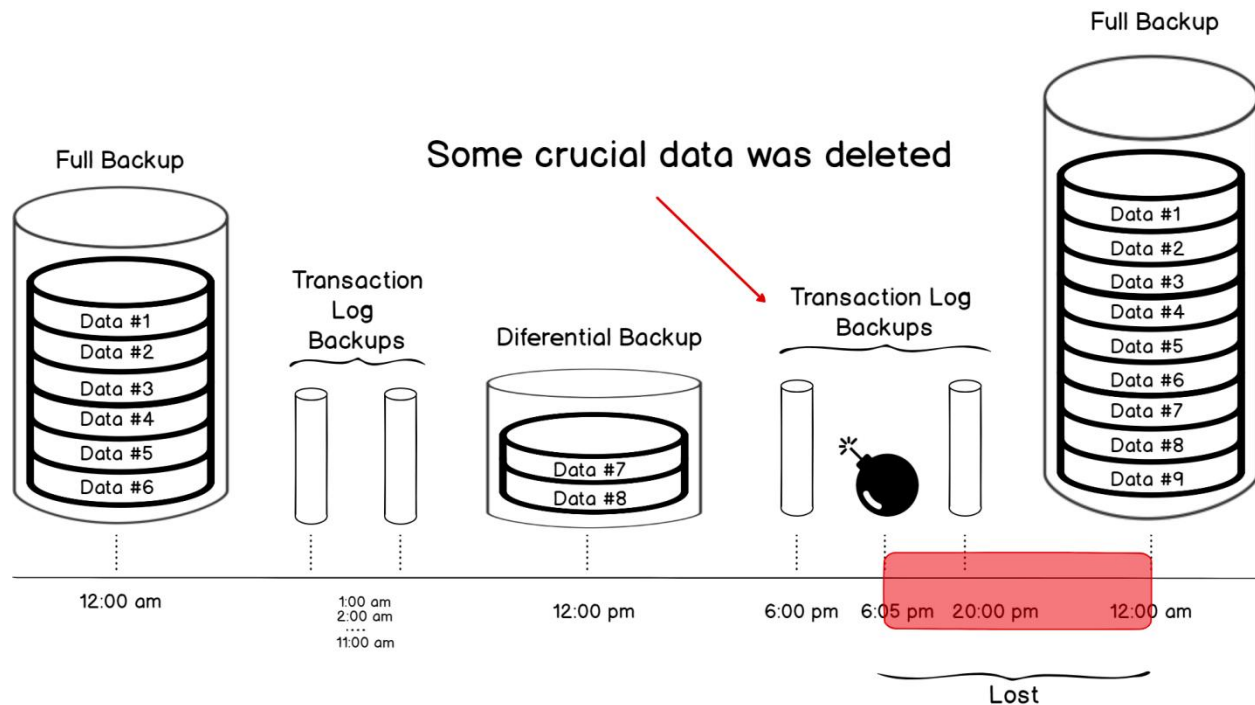


For synchronization, For the screen display data, the headquarter is the publisher of the data, the regional data servers are subscribing those data and display on their screens, full prompt display, the data will be full replication and synchronized. For the Flight Type, Airline Name and Location data, they won't change a lot throughout the time, the HQ will be the maintainer and publisher of these data, but it is not necessary to synchronize all the time and be up-to-date all the time, so it will be asynchronized. For the weather table, in order to keep it accurate enough, the data is collected by the local/regional servers(publisher) and the headquarters will be the subscriber, it will be synchronize daily with only some of the data replicated(like snow or alert weather data).

Log and recovery are important mechanisms used in database management systems to ensure data integrity, availability, and durability in the face of various failures or errors.

A database log is a record of all changes made to the database, including inserts, updates, and deletes, as well as other important events, such as database startup and shutdown. The log is typically stored on a separate disk or device to protect against disk failures or crashes. By keeping a log of all changes, the database management system can use the log to recover the database in the event of a failure or error.

Recovery refers to the process of restoring the database to a consistent state after a failure or error. Recovery can be performed in various ways, depending on the nature of the failure and the specific recovery technique used.



This backup plan contains a full backup at 12am, a transaction logs backups every hour between 1-11 am, a differential backup at 12pm, the black design will be implemented to the weather and flight data in a global database scenario, when there a millions of flight and weather data from all airports and locations in the world coming in every hour. So when there is an error happens during, the database can be easily restored to the latest version with the least cost.

There are several cloud data warehouse products available on the market, each with their own strengths and weaknesses. Here are some key factors to consider when comparing cloud data warehouses:

Scalability: One of the main advantages of cloud data warehouses is their ability to scale up or down quickly based on changing data requirements. Look for a cloud data warehouse that can handle large volumes of data and scale up or down easily without downtime or data migration.

Performance: The speed and performance of a cloud data warehouse is critical to ensure fast data processing and analytics. Look for a data warehouse that can process queries quickly and efficiently, with low latency and high throughput.

Cost: The cost of cloud data warehouses can vary widely depending on the vendor and the specific features and services included. Look for a data warehouse that fits your budget and offers transparent pricing with no hidden fees or charges.

Security: Cloud data warehouses store sensitive data, so it's important to choose a vendor that takes security seriously. Look for a data warehouse that offers robust security features, such as encryption, access controls, and audit trails.

Integration: Cloud data warehouses often need to integrate with other tools and services, such as data pipelines, ETL tools, and BI platforms. Look for a data warehouse that offers easy integration with these tools, with built-in connectors and APIs.

The major platforms are Google Bigquery, Microsoft Azure Synapse, Snowflake and Amazon Redshift, here is the differences chart for these major services:

Top Cloud Data Warehouses






	Amazon Redshift	Microsoft Azure Synapse	Google BigQuery	Snowflake Cloud Data Platform
Initial Release	2012	2016	2010	2014
Separates Storage and Compute	No	Yes	Yes	Yes
Multi-Cloud	No	No	No	Yes
Query Language	Amazon Redshift SQL	TSQL	Standard SQL 2011 & BigQuery SQL	Snowflake SQL
Elasticity	Yes - Manual	Yes - Manual and Automatic	Yes - Automatic	Yes - Automatic
MPP	Yes	Yes	Yes	Yes
Columnar	Yes	Yes	Yes	Yes
Foreign Keys	Yes	Yes	No	Yes
Transaction	ACID	ACID	ACID	ACID
Concurrency	Yes	Yes	Yes	Yes
Durability	Yes	Yes	Yes	Yes
Automation	No	No	No	No
Website	Link	Link	Link	Link
Free Trial	Yes	Yes	Yes	Yes

As we can see, most of the functions are the same, but Snowflake has a very good automatic elasticity and multicloud usage.

Concerning the implementation price, Snowflake is also the most preferable one.

Amazon Redshift: Amazon Redshift is priced based on the number of nodes and the amount of storage used, with pricing starting at \$0.25 per hour for a single-node cluster with 160GB of storage. The cost can quickly add up for larger clusters with more storage, but Redshift offers significant discounts for reserved instances and upfront payments.

Google BigQuery: Google BigQuery offers a flexible pricing model based on usage, with charges for storage, query processing, and streaming data ingestion. Storage is priced at \$0.020 per GB per month, while query processing is priced at \$5 per TB of data processed. Streaming data ingestion is priced at \$0.010 per GB.

Snowflake: Snowflake offers a pay-as-you-go pricing model based on usage, with separate charges for storage, compute, and data transfer. Storage is priced at \$23 per TB per month, while compute is priced at \$0.00056 per second per credit. Data transfer is priced at \$0.05 per GB for inbound data and \$0.09 per GB for outbound data.

Microsoft Azure Synapse Analytics: Microsoft Azure Synapse Analytics offers a flexible pricing model based on usage, with charges for storage, processing, and data movement. Storage is priced at \$20 per TB per month, while processing is priced at \$1.50 per DWU-hour (Data Warehouse Unit-hour). Data movement charges vary depending on the amount of data and the destination.

So if the database is finding a cloud service, the best migration option is to Snowflake.

6) Tuning & Optimization

Indexing is a technique used in databases to improve the speed of data retrieval operations by creating a separate data structure that maps the values of one or more columns in a table to their physical location on disk.

When a database query includes a condition on one or more columns, the query optimizer can use the index to quickly locate the rows that match the condition, rather than scanning the entire table. This can significantly improve the performance of queries that filter or sort large amounts of data.

There are three major indexing types in database management and their differences are listed as follows:

Data Types: Bitmap and B-tree indexes can handle both numeric and character data types, while hash indexes are typically limited to numeric data types

Cardinality: Bitmap indexes are effective for low cardinality columns, while B-tree and hash indexes are suitable for high cardinality columns.

Size and Storage: Bitmap indexes require less storage space compared to B-tree and hash indexes. However, B-tree indexes are more flexible and can handle large data sets. Hash indexes are typically faster than B-tree and bitmap indexes for exact match queries, but they are less flexible and less efficient for range queries.

Query Performance: Bitmap indexes are useful for boolean operations such as AND, OR, and NOT, while B-tree and hash indexes are better suited for range queries and exact match lookups, respectively.

Maintenance: B-tree and hash indexes are easier to maintain and update compared to bitmap indexes, which can become complex to maintain as data sets grow in size.

The weather data like precipitation contains over 10,000 different records with literally no repetition numeric numbers, so it is high cardinality, we using B-tree to create a faster indexing to improve the performance.

According to the data structure and the indexing design are as follows:

```
CREATE INDEX idx_weather1_hash ON project.weather USING btree(Precipitation);
```

```
CREATE INDEX idx_weather2_hash ON project.weather USING btree(Snowfall);
```

```
CREATE INDEX idx_weather3_hash ON project.weather USING btree(SnowonGround);
```



```
CREATE INDEX idx_weather4_hash ON project.weather USING btree(Temperature);
```

```
CREATE INDEX idx_weather5_hash ON project.weather USING btree(WindSpeed);
```

We can test and see if the performance changes before and after the tuning. Before Tuning, there is a long time full table scan that stuck the buffer cache and I/O.

查询 查询历史

```

300 --Tuning
301 --explain a concat query before indexing
302 EXPLAIN SELECT * FROM project.weather
303 WHERE Precipitation >2 AND SNOWFALL<5
304 ORDER BY precipitation DESC;
305 --create a indexing
306 CREATE INDEX idx_weather1_hash ON project.weather USING btree
307 CREATE INDEX idx_weather2_hash ON project.weather USING btree
308 CREATE INDEX idx_weather3_hash ON project.weather USING btree
309 CREATE INDEX idx_weather4_hash ON project.weather USING btree
310 CREATE INDEX idx_weather5_hash ON project.weather USING btree
311 --explain on it after indexing
312 EXPLAIN SELECT * FROM project.weather
313 WHERE Precipitation >2 AND SNOWFALL<5
314 ORDER BY precipitation DESC;
315

```

Data Output 消息 通知

+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	QUERY PLAN	🔒
	text	
1	Gather Merge (cost=349555.50..409261.03 rows=511726 width=292)	
2	Workers Planned: 2	
3	-> Sort (cost=348555.47..349195.13 rows=255863 width=292)	
4	Sort Key: precipitation DESC	
5	-> Parallel Seq Scan on weather (cost=0.00..255607.56 rows=255863 width=292)	
6	Filter: ((precipitation > '2'::numeric) AND (snowfall < '5'::numeric))	

After tuning, by using the indexing. It greatly improved the performance.

查询 查询历史

```

309 CREATE INDEX idx_weather4_hash ON project.weather USING btree(Temperature);
310 CREATE INDEX idx_weather5_hash ON project.weather USING btree(WindSpeed);
311 --explain on it after indexing
312 EXPLAIN SELECT * FROM project.weather
313 WHERE Precipitation >2 AND SNOWFALL<5
314 ORDER BY precipitation DESC;
315
316 --trigger prevent deletion
317 CREATE OR REPLACE FUNCTION prevent_airline_deletion()
318 RETURNS trigger AS
319 $$
320 BEGIN
321     RAISE EXCEPTION 'Deletion of records in the airline table is not allowed'
322     RETURN NULL;
323 END;

```

Data Output 消息 通知

QUERY PLAN
text

1	Index Scan Backward using idx_weather1_hash on weather (cost=0.56..240323.67 rows=92054 width=1...
2	Index Cond: (precipitation > '2'::numeric)
3	Filter: (snowfall < '5'::numeric)

Here are some discussion on how to improve the indexing.

Index Combination: In some cases, creating a combined index on multiple columns can be more effective than creating separate indexes on each individual column. This can be particularly useful if queries often filter on multiple columns. For example, you might create an index on `(Temperature, Precipitation)` instead of separate indexes on `Temperature` and `Precipitation`.

Index Type Selection: As mentioned before, the hash index may be more efficient for columns with a small number of distinct values. For example, if `Airquality` has only a few distinct values, it may be more efficient to create a hash index on this column.

Index Maintenance: Regularly maintaining the indexes can help to ensure their efficiency. This can include reindexing the table to eliminate fragmentation, or updating the statistics to help the optimizer choose the best execution plan.

Query Optimization: Optimizing the queries themselves can often have a greater impact on performance than simply creating indexes. Reviewing query execution plans and identifying areas for improvement, such as reducing unnecessary joins or subqueries, can help to improve overall performance.

Triggers and stored procedures are two common tools used in data tuning to improve the performance and reliability of database operations.

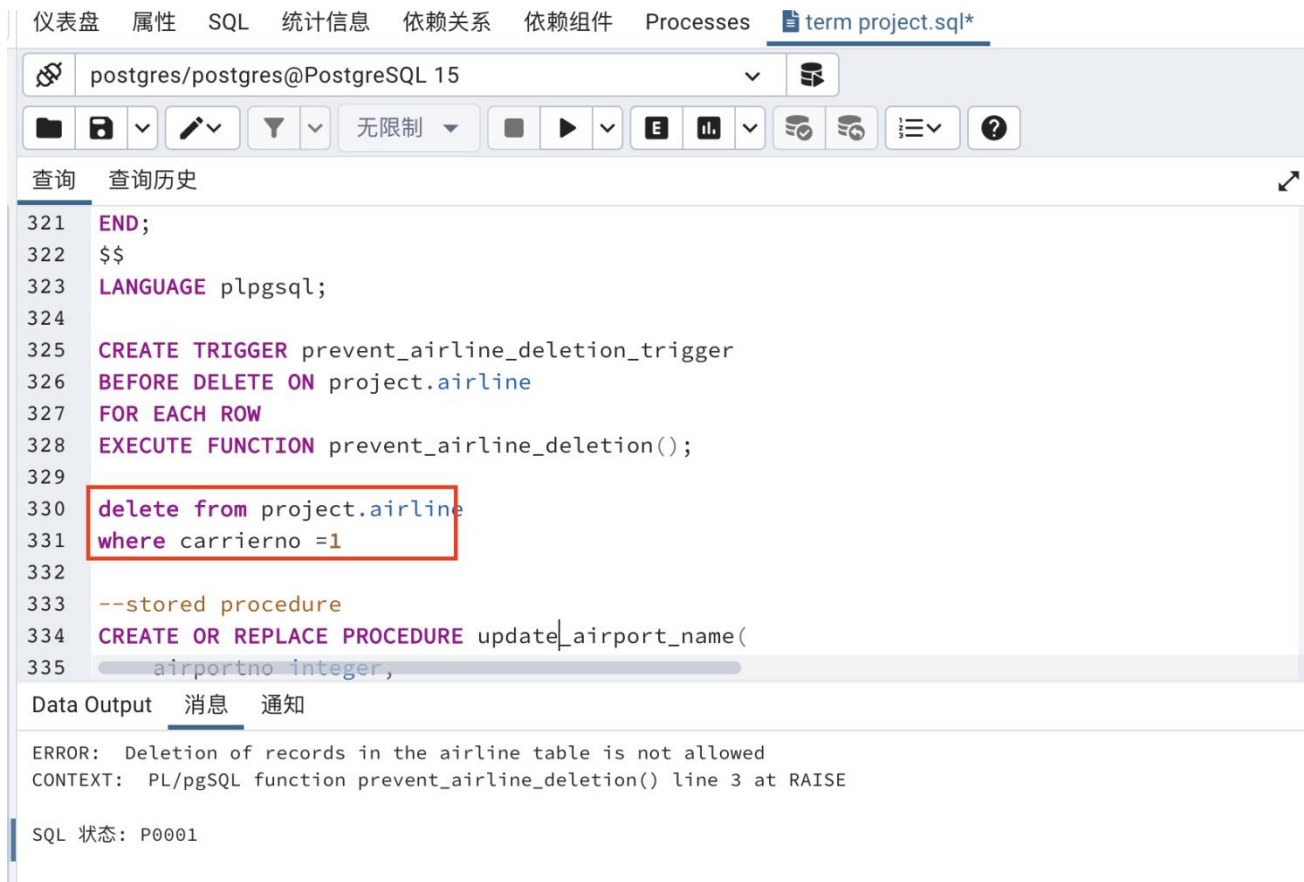
A trigger is a piece of code that is automatically executed by the database in response to a specific event, such as an insert, update, or delete operation on a table. Triggers can be used to enforce business rules, perform data validation or transformation, or update related tables. They can also be used to audit changes to the database or replicate data across multiple servers.

For the weather data, airlines is a basic element that won't change a lot, but if misdeletion happens, it would cause a messy chaos. We can set up a trigger for preventing deletion on airline table.

```
CREATE OR REPLACE FUNCTION prevent_airline_deletion()  
RETURNS trigger AS  
$$  
BEGIN  
RAISE EXCEPTION 'Deletion of records in the airline table is not allowed';  
RETURN NULL;  
END;  
$$  
LANGUAGE plpgsql;
```

```
CREATE TRIGGER prevent_airline_deletion_trigger  
BEFORE DELETE ON project.airline  
FOR EACH ROW  
EXECUTE FUNCTION prevent_airline_deletion();
```

Now we test the trigger and see if the deletion can be prevented by the trigger. As below, the trigger can successfully prevent user from misdeleting the airline table data.



```
321 END;
322 $$
323 LANGUAGE plpgsql;
324
325 CREATE TRIGGER prevent_airline_deletion_trigger
326 BEFORE DELETE ON project.airline
327 FOR EACH ROW
328 EXECUTE FUNCTION prevent_airline_deletion();
329
330 delete from project.airline
331 where carrierno =1
332
333 --stored procedure
334 CREATE OR REPLACE PROCEDURE update_airport_name(
335     airportno integer,
```

Data Output 消息 通知

ERROR: Deletion of records in the airline table is not allowed
CONTEXT: PL/pgSQL function prevent_airline_deletion() line 3 at RAISE

SQL 状态: P0001

Stored procedures, on the other hand, are precompiled blocks of code that can be executed on demand by the database. Stored procedures can be used to encapsulate complex business logic, perform batch processing, or automate repetitive tasks. They can also improve security by limiting direct access to the database and providing a layer of abstraction between the application and the database.

When we want to alter the value of the attribute "airportnamecurrent", stop doing that but save the old value into "airportnameprevious" as a record, then save another record to the attribute "airportnamecurrent". The function was realized by the following stored procedures:

```
CREATE OR REPLACE PROCEDURE update_airport_name(  
    __airport_no integer,  
    __new_name text  
)  
LANGUAGE plpgsql  
AS $$  
DECLARE
```

```
old_name text;  
  
BEGIN  
  
-- Get the current value of the "AirportNameCurrent" attribute  
  
SELECT AirportNameCurrent INTO old_name FROM project.airports WHERE AirportNo = airport_no;  
  
  
-- If the new value is the same as the old value, do nothing  
  
IF old_name = new_name THEN  
  
RETURN;  
  
END IF;  
  
  
-- Otherwise, update the "AirportNamePrevious" attribute with the old value  
  
UPDATE project.airports SET AirportNamePrevious = old_name WHERE AirportNo = airport_no;  
  
  
-- Update the "AirportNameCurrent" attribute with the new value  
  
UPDATE project.airports SET AirportNameCurrent = new_name WHERE AirportNo = airport_no;  
  
END;  
  
$$;
```

The testing of the stored procedures are as follows: We create a new record of the airport and see what happens:

```
358 END;  
359 $$;  
360 --test the stored procedures  
361 --show the original value  
362 SELECT * FROM PROJECT.airports  
363 WHERE AIRPORTNO =3;  
364 --execute the new value  
365 DO $$  
366 BEGIN  
367     CALL update_airport_name(2, '009');  
368 END;
```

Data Output 消息 通知

	airportno [PK] integer	airportnamecurrent character varying	airportnameprevious character varying
1	3	Theodore Francis Green State	[null]

```
363 WHERE AIRPORT_ID = 3;  
364 --execute the new value  
365 DO $$  
366 BEGIN  
367     CALL update_airport_name(3, 'Boston Logan');  
368 END;  
369 $$;  
370  
371 SELECT * FROM PROJECT.airports
```

Data Output 消息 通知

DO

耗时122 毫秒 成功返回查询。

```
368 END;  
369 $$;  
370 --see changes  
371 SELECT * FROM PROJECT.airports  
372 WHERE airportno = 3  
373  
374 --Dimensional&creation  
375 CREATE TABLE project.time (  
376 TIME TIMESTAMP,
```

Data Output 消息 通知

	airportno [PK] integer	airportnamecurrent character varying	airportnameprevious character varying
1	3	Boston Logan	Theodore Francis Green State

As we can see, if we use this function to change the name of the airport from Theodore Francis Green State to Boston Celtics, it changes successfully. But if we are using the same value, it keeps the record. Procedure test passed.

```
373  --execute the same value
374  DO $$
375  BEGIN
376      CALL update_airport_name(3, Boston Logan');
377  END;
378  $$;
379  --see changes
380  SELECT * FROM PROJECT.airports
381  WHERE airportno = 3
382
383
384  --Dimensional&creation
```

Data Output 消息 通知

DO

耗时91 毫秒 成功返回查询。

```

373  --execute the same value
374  DO $$
375  BEGIN
376      CALL update_airport_name(3, 'Boston Logan');
377  END;
378  $$;
379  --see changes
380  SELECT * FROM PROJECT.airports
381  WHERE airportno = 3
382
383
384  --Dimensional&creation

```

Data Output 消息 通知



	airportno [PK] integer	airportnamecurrent character varying	airportnameprevious character varying	airportflightmonth text
1	3	Boston Logan	Theodore Francis Green State	1495

We can also make a stored procedure to renew the hotel data and implemented to see the test the function, as below, the function has been fully tested.

```

CREATE OR REPLACE PROCEDURE update_hotel_name(
    hotelno integer,
    new_name varchar(25)
)
LANGUAGE plpgsql
AS $$
DECLARE
    old_name varchar(25);
BEGIN
    -- Get the current value of the "HotelNameCurrent" attribute
    SELECT HotelNameCurrent INTO old_name FROM project.HotelName WHERE HotelNameNo = hotelno;

    -- If the new value is the same as the old value, do nothing
    IF old_name = new_name THEN

```


____ RETURN;

____ END IF;

____ -- Otherwise, update the "HotelNamePrevious" attribute with the old value

____ UPDATE project.HotelName SET HotelNamePrevious = old_name WHERE HotelNameNo = hotelno;

____ -- Update the "HotelNameCurrent" attribute with the new value

____ UPDATE project.HotelName SET HotelNameCurrent = new_name WHERE HotelNameNo = hotelno;

END;

\$\$;








```
412 --show the original value
413 SELECT * FROM PROJECT.hotelname
414 WHERE hotelnameno =2;
415 --execute the new value
416 DO $$
417 BEGIN
418     CALL update_hotel_name(2,'Marriot');
419 END;
420 $$;
421 --see changes
422 SELECT * FROM PROJECT.hotelname
423 WHERE hotelnameno = 3
424 --execute the same value
425 DO $$
426 BEGIN
427     CALL update_hotel_name(2,'Marriot');
428 END;
429 $$;
430 --see changes
```

Data Output 消息 通知

	hotelnameno [PK] integer	hotelnameprevious character varying	hotelnamecurrent character varying
1	2	[null]	Hotel A

```
415 --execute the new value
416 DO $$
417 BEGIN
418     CALL update_hotel_name(2,'Marriot');
419 END;
420 $$;
421 --see changes
422 SELECT * FROM PROJECT.hotelname
423 WHERE hotelnameno = 2
424 --execute the same value
425 DO $$
426 BEGIN
427     CALL update_hotel_name(2,'Marriot');
428 END;
429 $$;
430 --see changes
```

Data Output 消息 通知

      			
	hotelnameno [PK] integer	hotelnameprevious character varying	hotelnamecurrent character varying
1	2	Hotel A	Marriot

```

421  --see changes
422  SELECT * FROM PROJECT.hotelname
423  WHERE hotelnameno = 2
424  --execute the same value
425  DO $$
426  BEGIN
427      CALL update_hotel_name(2, 'Marriot');
428  END;
429  $$;
430  --see changes
431  SELECT * FROM PROJECT.hotelname
432  WHERE hotelnameno = 2
433
434
435
436  --Dimensional & creation

```

Data Output 消息 通知

	hotelnameno [PK] integer	hotelnameprevious character varying	hotelnamecurrent character varying
1	2	Hotel A	Marriot

3. Conclusion

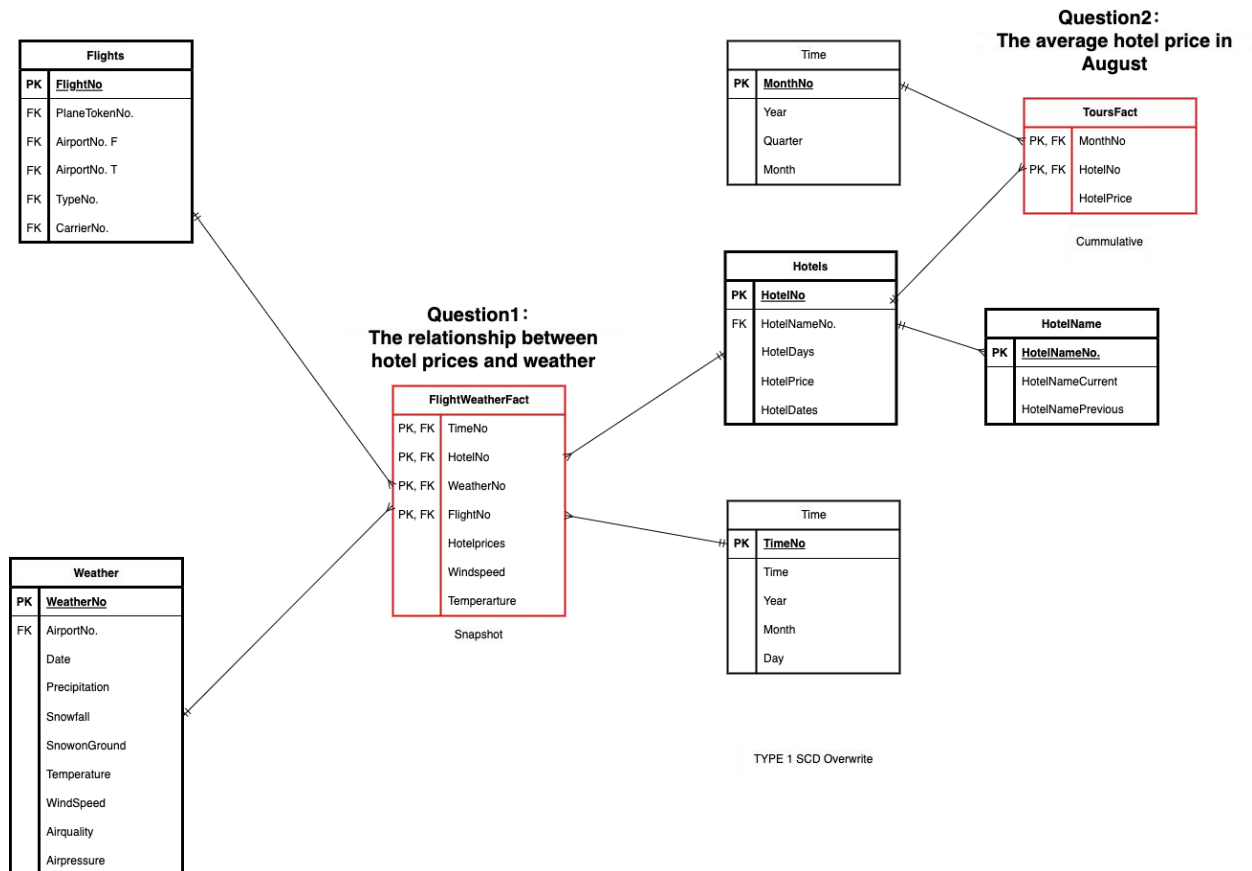
1) Visual & Analysis

Dimensional and fact tables are two key components of a star schema, a popular data modeling technique used in data warehousing.

A fact table contains the quantitative measurements, or facts, of a business process, such as sales revenue, units sold, or clicks on a website. The fact table is typically large and contains a foreign key that links to one or more dimension tables.

A dimension table provides context for the data in the fact table by describing the attributes of the business process, such as time, product, location, or customer. Dimension tables are typically smaller and contain descriptive attributes that can be used to filter or group the data in the fact table.

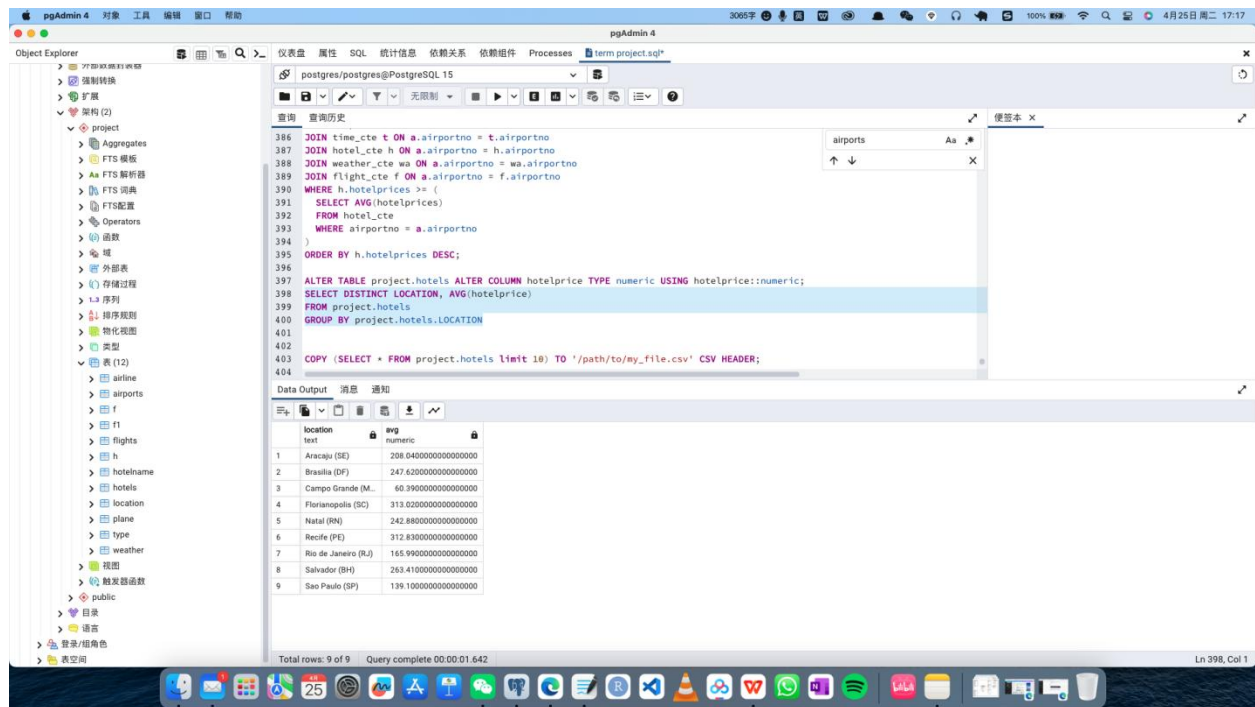
Here is a constellation design of the schema to answer some of query questions.



As we can see, in the constellation, the red *“FlightWeatherFact”* is a fact table with all dimensions provided by the black dimensional tables around it.

The dimensional design contains a snapshot and a cummulative design, these designs along with the measurements to answer the business question “The relationship between hotel prices and weather” and “The average hotel price in August”.

This is not the focus point of the warehousing project, so the dimensional design shown are conceptual here, instead, below shows a sample query build on the dimensional database and the [visualization example](#) on the dataset.



2) Communication Theories

Some communication theories can also be applied to the analysis of weather and travel data.

Elaboration Likelihood Model (ELM): This theory proposes that there are two routes to persuasion: a central route and a peripheral route. The central route involves careful consideration and evaluation of the message, while the peripheral route relies on cues such as source credibility or emotional appeals. In airline marketing, the ELM could be used to design messages that appeal to both routes, such as highlighting the safety record of the airline (central route) while also using attractive visuals and emotional music in the marketing campaign (peripheral route).

Social Exchange Theory: This theory suggests that human relationships are based on an exchange of rewards and costs. In airline marketing, the social exchange theory could be applied by emphasizing the rewards of flying with the airline (such as comfort, convenience, and status) and minimizing the costs (such as price, inconvenience, and risk).

Uses and Gratifications Theory: This theory proposes that people use media to satisfy specific needs, such as information, entertainment, or social interaction. In airline marketing, the uses and gratifications theory could be applied by designing messages that appeal to the different needs of the target audience, such as providing information about flight schedules and amenities, showcasing the destination and cultural experiences, or promoting social connections and community through travel.

Social Learning Theory: This theory suggests that people learn new behaviors by observing and imitating others. In airline marketing, the social learning theory could be applied by featuring endorsements and testimonials from satisfied customers, celebrities, or influencers who embody the values and lifestyle associated with the airline brand. This could also be extended to social media campaigns that encourage user-generated content and sharing of travel experiences.

Bibliography

- Oliver, M. B., Raney, A. A., & Bryant, J. (2019). *Media Effects: Advances in Theory and Research*.
- Coronel, C., & Morris, S. (2018). *Database Systems: Design, Implementation, & Management*. Cengage Learning.
- Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. "O'Reilly Media, Inc."
- *Database Concepts*. (n.d.).
<https://docs.oracle.com/database/121/CNCPT/indexiot.htm#CNCPT721>
- Yaseen, A. (2023). SQL Server Transaction Log and Recovery Models. SQL Shack - Articles About Database Auditing, Server Performance, Data Recovery, and More.
<https://www.sqlshack.com/sql-server-transaction-log-and-recovery-models/>