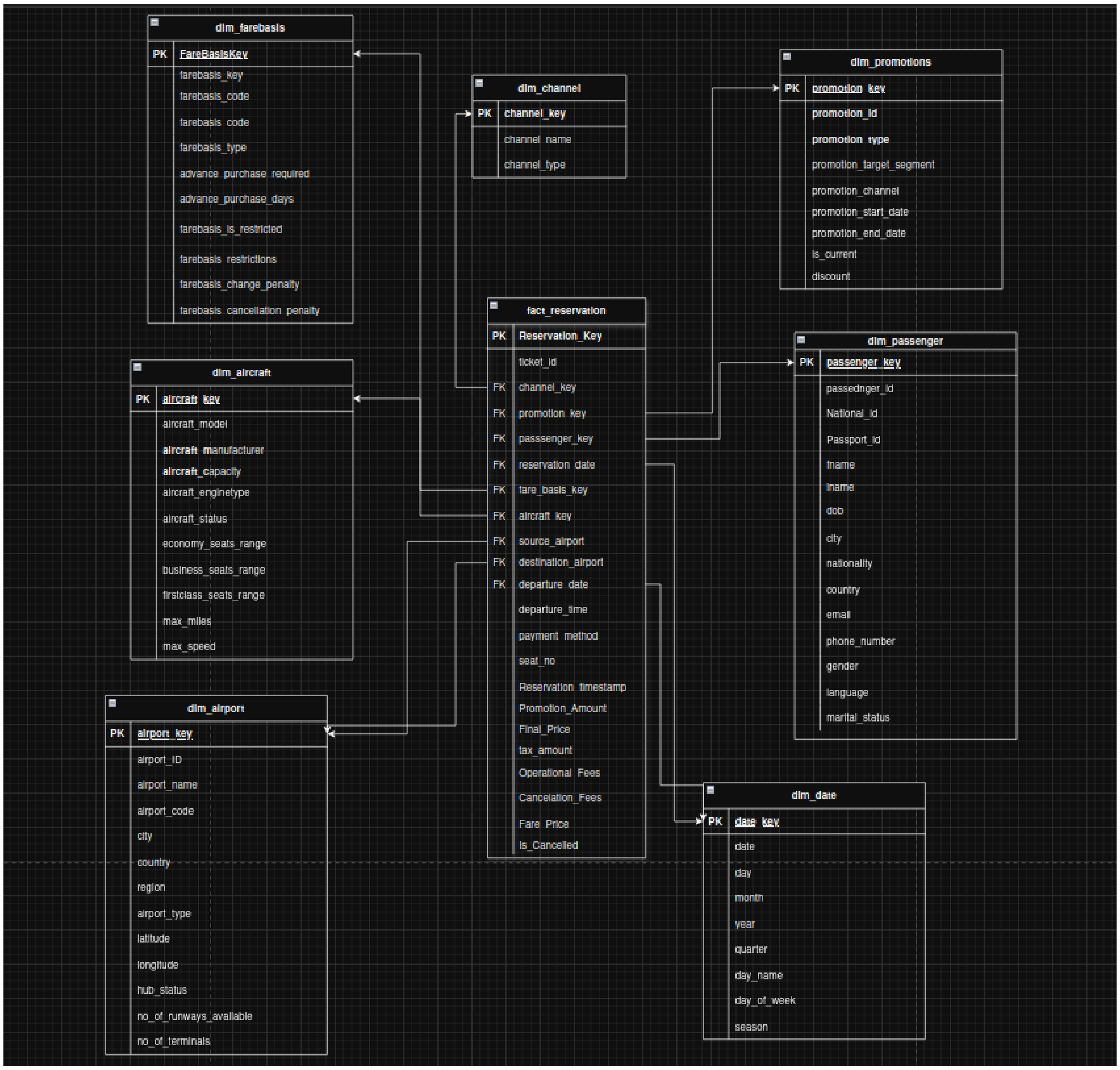
Part2 Final Notes

1- Migration:

- my DWH is on google big query
- it stored as external table on GCS storage as Parquet
- i pull data on the container local using python code then transform it into orc the put it into HDFS using script /data/pull_convert_hdfs.py
- now i have data on hdfs i need to create ddl to read this data on hive schema
- the ddl is on ddl.sql
- i assume that my fact is 10 GB , i partition it by date then bucketing it
- to select the right number of buckets i follow this formula (table size / hdfs block)
- i follow this formula to ensure that each bucket is stored on a single hdfs block to reduce shuffling and network overhead when processing this bucket
- duo to this formula the no. of bucket is (10 * 1024/128) = 80 bucket
- there is a problem that each bucket is exactly 128 MB = one hdfs block
- in the future when scaling the bucket will exceed the hdfs block size
- note that the data is partitioned i assume that each partition is 1 GB
- the new number of bucket is (1024 / 128)=8 bucket per partition
- im doubling it to 16 , this results to the size of each bucket is (1024/16) = 64 MB
- now each bucket is scalable
- i assume this numbers bcs i have no real world data to see it's real numbers
- in the future when the buckets gets bigger i can re bucket the table following the same technique
- tables are stores on hdfs , and i read it as external tables
- only transactional tables (SCD 2) are internal

- 

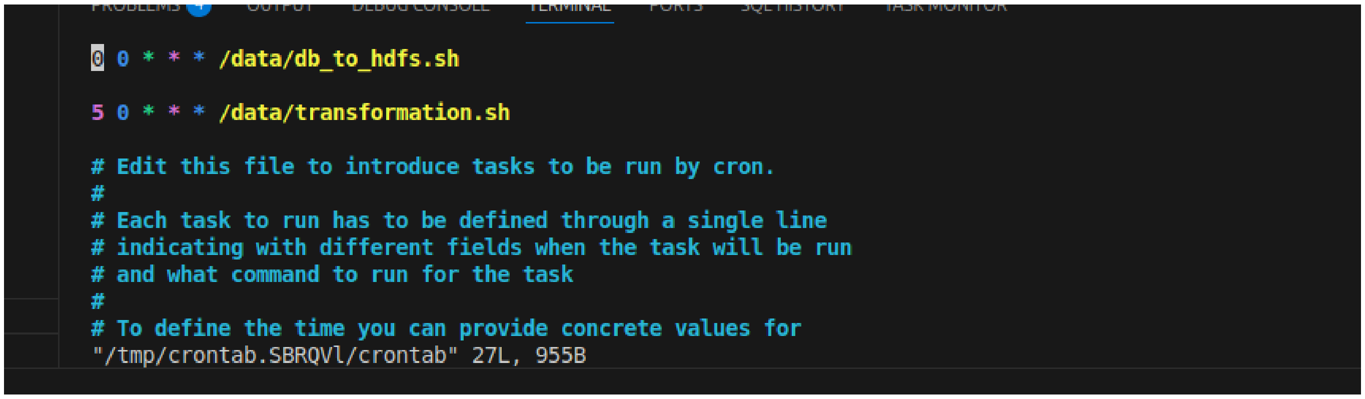
2- DB

- i've created a db schema in 3nf to be suitable for the dwh
- the creation is on db_ddl.sql
- i designed the db to be suitable for incremental load
- each db table has to additional cols (inserted at and updated at)
- the data inserted into a table automatically register it's insertion timestamp using default value of the current time stamp
- and there is a trigger that updates the (updated at) timestamp for each update on the data
- there is an extra table on the db called meta_data table
- i insert a new row each time i pull the data from the DB to keep up with the last pull timestamp
- this helps me to pull only new and updated data in the next time

- when i pull data i check the max time stamp on meta data table and go to select the data of bigger time stamp of created at col or updated at col
- i handle it using views
- and the view don't select (created at and updated at cols)
- i pull the data from the database to hive using the technique similar to the migration
- but the db is on postgres created with this command
  - `docker run --name db -e POSTGRES_PASSWORD=123 -p 5433:5432 -v postgres-data:/var/lib/postgresql/data --network hive_hadoop_cluster -d postgres:13`
- the script pull the data from the all views on the db
- this script exists in /data/db_to_hdfs.py
- the script pulls the data and convert it into orc and put it on hdfs
- now i have files of the db on hdfs, lets read it into hive and create ddl for it
- you can find the ddl on db_HQL.sql
- now we have the only new data that changed from the last incremental load
- NOW WE HAVE DONE EL lets perform T to complete our cycle (ELT)
- we should load data in landing schema
- ## Transformation and load into fact_reservation
  - ### note : i implemented this steps for the fact only
  - perform data cleaning and joins on db tables using HQL to be suitable with the dwh
  - ### cleaning :
    - handle nulls , check empty strings , trim strings (skipped bcs im sure my data is clean)
    - make a standard for the data if it comes from multiple sources but in my case it's only one source
    - then after cleaning we put the cleaned data that matches our dwh in another schema we can call it business schema
    - insert based on select from this schema to our dwh
    - you can find this steps in Transformations.sql

3- crontab:

- create two .sh scripts , one for db_to_hdfs and the other to transformation.hql and load data into the fact
- in db_to_hdfs.sh we put this command `python3 /data/db_to_hdfs.py`
- in transformation.sh we put this command `beeline -u "jdbc:hive2://localhost:10000/default" -n hadoop -p hadoop -f /data/transformation.hql`



```
0 0 * * * /data/db_to_hdfs.sh

5 0 * * * /data/transformation.sh

# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
"/tmp/crontab.SBRQVl/crontab" 27L, 955B
```

4- SCD 2: didn't implement it

- gets the new data from the db using previous technique in (*2-db*) in staging
- checks if the key of the new data exists in the dwh , if it existed :
  - update existed record and set is current to 0 and set end_date to current date
  - insert the new record with is current is 1
- if not exist just insert it withe start_date = current date and is_current = 1