Assessment: SQL query and DW design suggestions

NN

February 15, 2022

1 Part 1: SQL Query

For this part, the SQL code I figured out and its results are in the next **Link**

Disclaimer: This code was developed using PLPGSQL, Postgres SQL language.

I decide to split the problem into small parts, beginning by grouping the incoming transfers by customer and month, to do that I had to join the *transfers_in* table to the *d_time* table, that way I could group by month. Same process for outgoing transactions, and pix transactions. All that using Common table expressions, what make it easy to understand. After that I started doing some summaries, for getting monthly ins and outs for normal transactions and pix transactions. I had to deal with nulls and at the end I had to make a final summary; that summary was done by using a window function, that way a can do kind of accumulator for every account base on the monthly final revenue (normal transactions and pix transactions).

2 Part 2: Suggestions for modifying data warehouse architecture.

1. Due to I don't know what database engine is being used, I would suggest to use AWS Aurora in case we have a MySQL or PostgreSQL engine because Aurora is faster and designed for cloud environments. but this is not compatible with my next point, which have more relevance because we're working on a Datawarehouse, not a transactional database.
2. Knowing that the problem is focused on Data warehouse, we're not going to have a transactional behavior, The data warehouses are designed to provide analytical in-

formation. Then, we don't need really fast queries, we need the capacity to process big data. Looking for it, I'd like to have it running on a services as AWS Redshift or Google BigQuery, that way we could process big data faster and with compressed data. I would select the best compression strategy based on AWS Redshift suggestions, e.g. for the id columns, if those columns are of the type 1, 3, 4... etc. I would apply a Delta algorithm for compression, that way the queries would be a lot faster. Same for the other columns depending on the kind of information stored in them.

3. Looking to make queries easier, I would implement a star schema (or snowflake due to the number of tables). I think for this case, the transactional tables would be the facts tables, I would come up with a transactions tables where I would store ids for the dimensions tables, as customer (customer_id), date (date_id) etc, and also the amounts, that way I would follow the star scheme design method. In the actual architecture we have smalls star schemes, e.g, the d_time table, it has only a numerical value, and ids to reference its own dimension tables (d_month, d_year, d_weekday)

4. Also, I would implement data modelation strategies, I mean, there could be others tables of information that we know are relevant for Bussiness analytics colleagues. For example, we could define a model (actually a SQL code that returns a table) containing information related with the first part of this test, some aggregations, as monthly balances, balances by city, country, maybe by product. That way, we could make the job of looking for information something self-service for Bussiness analytics colleagues. And those models (tables) should be materialized time by time, each day, or hour, depending on the information them contain.

5. Something I can infer from the diagram on the first page of the case, is that we have a ELT process instead a ETL process. That's something helpful knowing the computing capacity of modern datawarehouses. We can do most of the transformation process inside the datawarehouse.

3 Part 3: Strategy to implement the datawarehouse changes

In case the data warehouse is not running on AWS Redshift or BigQuery, I would store all the data into files, as parquets files partitionated by numeric value, for example month_id that way we could have a folder for every month of transactions. That would speeds up the import process. Having those files we would define the tables schema based on my previous suggestions (Part 2). After that it's enough just running a copy command to import the data. Otherwise, if we are going to stay with a transactional database, like MySQL o PostgreSQL, we could use AWS DMS to migrating to Aurora. Then, when we have the data at the final destination, the stored procedures and queries can easily be imported and we could start to redirect all the components to the new endpoints, just re-deploying them one by one, avoiding a massive failure.

4 Part 4: PIX Metrics

1. ROI, the return on investment. If we know how much we've invested on developing the technological implementation for PIX transactions and we know how much we earn

by certain window of time, we can calculate the ROI.

2. Transactional growth. We can see the trend of the average money transacted every month or year, if it's increasing, it means we're doing well.

3. Relation to normal transactions. We could measure how much money is transacted every month using normal transactions and PIX transactions, that way, if we see an increase on that trend, it means PIX is growing and deserve to keep developing new features to support it.

4. New users using PIX. We can compare the percentage of growth (hopefully) of people starting to use PIX transactions over others methods.