Report - Data Engineer

Wellington Rodrigo Monteiro

Curitiba, 2020

1 Folder structure

The current folder is composed of the following structure:

- **\files:** responsible of storing all the scripts/code used in this challenge;

  converter.py: responsible of converting the sales order reviews CSV to a better format (see the Section 2.3);

  raw.sql: responsible of creating all the external tables in Redshift;

  real.sql: responsible of creating all the real tables in Redshift (i.e. *materialization*);

  olist_order_reviews_dataset_engineered.csv: example of an engineered CSV file;

- **\screenshots:** contains all the screenshots to be used as proof.

2 Thoughts

2.1 General Infrastructure

As suggested, the Amazon Web Services (AWS) structure was used to receive, process and manage incoming data from sales orders. By using a cloud-managed solution my proposal is the following considering my current technical background (i.e. using the Microsoft (Azure) stack):

- Should Olist use the Microsoft stack all the incoming CSVs would be kept in the *Azure Data Lake* solution and its mirrored DBMS structure would be managed in *Azure Data Warehouse* (now known as *Synapse*). Data Lake ensures the proper access management in folder/subfolder/file levels as well as allowing the usage of SDKs in order to develop new applications or integrating current ones to this end including, but not limited to the use of other cloud solutions such as *Data Factory* or *Databricks*.

- Considering the actual scenario where AWS is adopted instead I understand these solutions have their counterparts in this cloud solution. Therefore, a *S3* bucket would be used to store the incoming CSV files while *Redshift* would manage the DBMS structure. *Glue* could be used to run Python scripts to clear the CSVs before being ingested by Redshift.

- Independently of the solution chosen, the general proposal of maintaining a external table structure inside a DBMS (where such external tables read from the CSVs) remains since it does work well both to provide a SQL-like interface to access information out of these files as well as replicating the structure itself. However, if the data is accessed frequently and in greater volume an alternative is to *materialize* these external tables in DBMS standard tables through a scheduled stored procedure that could be executed in out of office hours depending on the application. Therefore, any views used by other teams and/or accessed externally would read either directly from these real tables or their views instead of reading from the external tables and suffering a performance hit due to it. Out of experience some BI users (either *Tableau* and *PowerBI*) might clog all the available resources depending on their filters chosen and the tables being read. Therefore, the faster these database requests are, the better it is system-wise.

2.2 Dataset Analysis

Considering the question risen on the actual DBMS model which was deemed as adequate for the data scientists and BI analysts I beg to differ. Many of the datasets provided are not normalized which may cause severe inconsistencies as well as spending more money than required with data storage. Some of the issues found were:

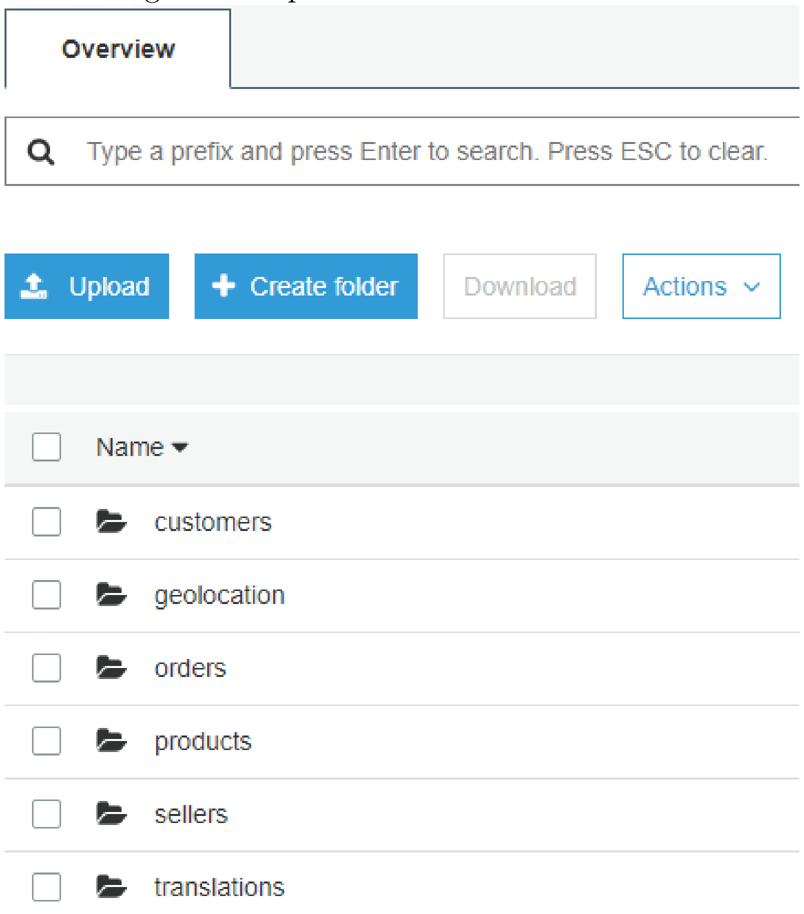
- *olist_customers_dataset.csv*: there are two IDs for the same customer; the ZIP code is not zerofilled and the city/state could be managed in a separate, own table/dataset (i.e. *olist_geolocation_dataset.csv*);

- *olist_geolocation_dataset.csv*: I understand the ZIP code was managed in quotes in order to ensure any converter would read this column as text instead of a number. Having that said, there are multiple entries where the ZIP code have several latitude/longitude combinations. Considering the analysis scope I understand these multiple points are not required. Therefore, I strongly recommend dropping any duplicates found by ZIP code;

- *olist_order_items_dataset.csv*: No changes. Under several cases I noticed the price and/or the freight value were the same for orders containing multiple items, but this behavior does not repeat itself for all the cases;

- *olist_order_payments_dataset.csv*: I strongly recommend creating a new table to store the payment types composed of just their labels and their identifiers (primary key/identity type);

Figure 1: Topmost level of the *raw* folder.


Overview
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
Download
Actions
Name

customers
geolocation
orders
products
sellers
translations

- *olist_order_reviews_dataset.csv*: The review ID seems to be unnecessary. In this dataset, the biggest problem in this dataset lies in the text. There are several cases where either the comment or the title are not enclosed by double quotes. Furthermore, in several cases the comment brokes down into several lines. Yet, I understand the six commas at the end of each data instance helps in the post-processing. More on this on the Section 2.3;

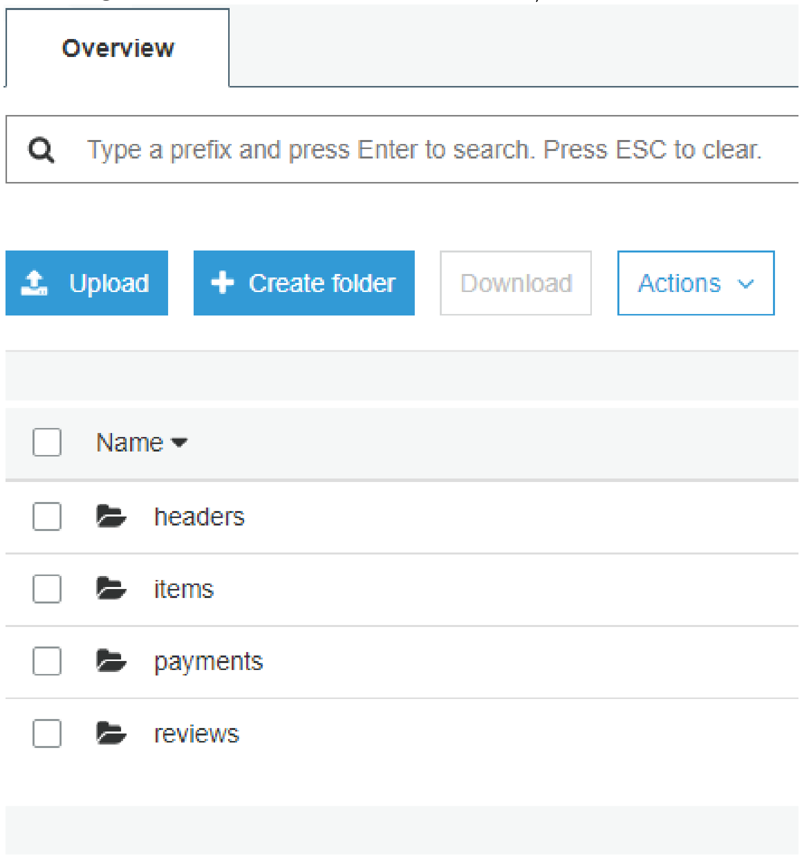
- *olist_orders___dataset.csv*: There are no standards adopted in the usage of the order and customer IDs – in some cases they are enclosed by double quotes, but there is not a pattern or a rationale as far as I could tell. I would also strongly recommend the creation of a new table to store the order statuses with their label and a primary key set as an identity;

- *olist_order_reviews_dataset.csv*: There biggest problem in this dataset lies in the text. There are several cases where either the comment or the title are not enclosed by double quotes. Furthermore, in several cases the comment brokes down into several lines. Yet, I understand the six commas at the end of each data instance helps in the post-processing. More on this on the Section 2.3;

- *olist_products___dataset.csv*: There are no standards adopted in the usage of the product IDs as much as it happened in the *olist_order_reviews_dataset.csv*. While some of the product characteristics does not seem to happen useful I suggest keeping them all since their data usage is relatively low. I would also suggest having a separate table to store the product categories and fixing some typos on the column names (e.g. *lenght* instead of *length*).

- *olist_sellers_dataset.csv*: the ZIP code is not zerofilled and the city/state could be managed in a separate, own table/dataset (i.e. *olist_geolocation_dataset.csv*);
- *product_category_name_translation.csv*: I would reuse this table to store the product category IDs as primary keys to be used in the *olist_products_dataset.csv*.

Figure 2: Structure inside the *raw/orders* folder.


Overview
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
Download
Actions
Name

headers
items
payments
reviews

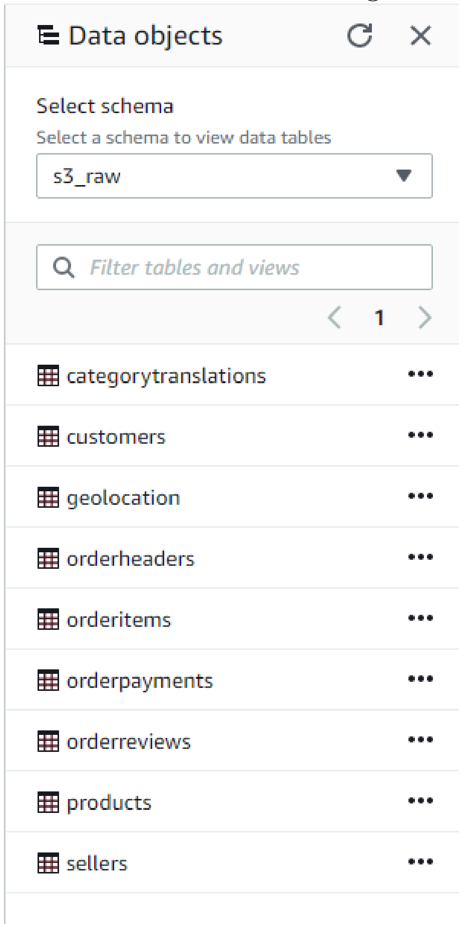
2.3 Data Management

I would recommend storing the aforementioned files in a *S3* bucket inside a *raw* folder. This folder, which its structure is shown in the Fig. 2.2 and Fig. 2.2 would be in charge of storing all the files received without cleaning whatsoever. These files are then processed by a Glue script (activated through S3 triggers) written in Python. One example I wrote is available inside the *\files\converter.py* file intended to use regex to properly identify the rows in the review, join all the text in a single review within a single line and enclose all the not null texts in double quotes. This script was written to process the file in smaller batches considering the file size in Production. Furthermore, the idea of these scripts is to properly modify the files allowing them to be easily read by Redshift.

All the processed files or files ready to be processed by Redshift through *CREATE EXTERNAL TABLE* (Fig. 2.3) would remain in a separate, mirrored S3 folder called *\engineered*. All the external table creation queries are available in the *\files\raw.sql*.

As soon as the external tables were created I did create another structure for the real tables (*materialization*) *\files\real.sql*. This structure, as shown in the Fig. 2.4, includes the proper table normalization as suggested in the Section 2.2 and the creation of stored procedures used to be executed through a job schedule.

Figure 3: Redshift schema containing the raw tables.



Data objects

Select schema

Select a schema to view data tables

s3_raw

Filter tables and views

1

categorytranslations

customers

geolocation

orderheaders

orderitems

orderpayments

orderreviews

products

sellers

2.4 General Comments

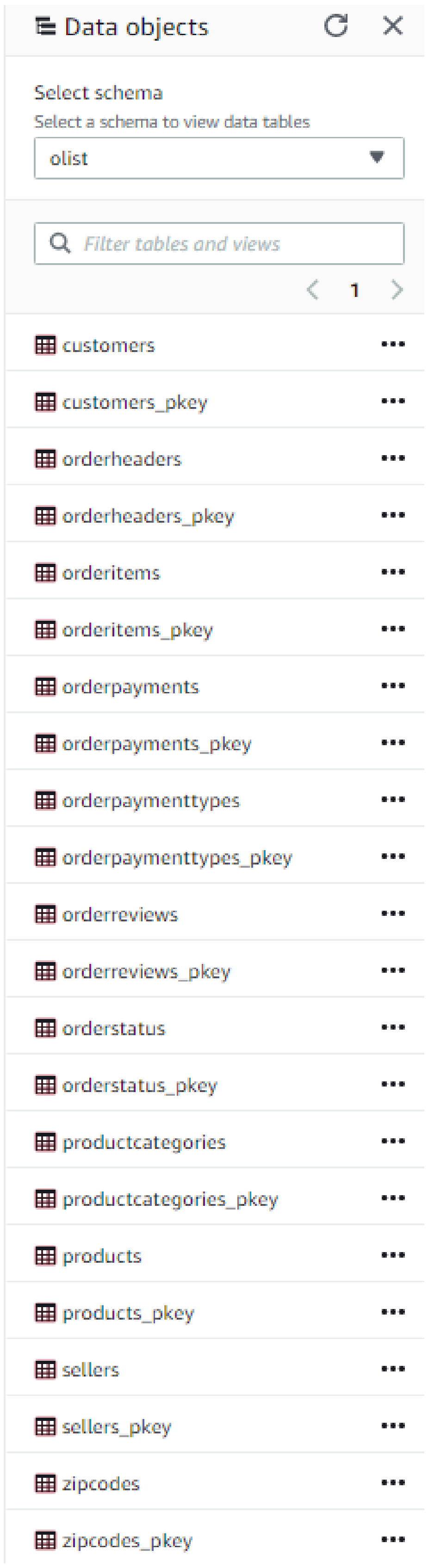
Considering the data volume being managed as well as what is expected to happen in the foreseeable future (i.e. an increasing amount in the volume) I would suggest first understanding the current IT architecture in place and their operation limits. Having that said, I would search for other users and companies that kept a structure similar to ours and look for any issues they had in doing so. By doing that I believe I would have a better picture on the future issues we might find in the future. After considering these points I would suggest and/or consider a possible migration – yet, I would focus on the solutions on premises and attempt to proactively determine if they are or could be a bottleneck in the future, changing them for cloud solutions should the scale and/or price are prone to be issues.

Considering the BI solutions (dashboards) and a possible issue found within the data infrastructure I would first attempt to identify patterns within the dashboards (e.g. what are the dashboards that are usually experiencing performance issues and/or what are the times where it does happen). After that I would identify if the issue is located in the queries or in the connection itself. If the problem is located in heavy queries I would find ways to create new materialized views or tables containing only the required data. It could increase the data volume but we would have better performance overall. Another option is to find the users that are generating the heaviest queries and attempt to fix the queries and/or educate the users. Therefore, I look forward to provide solutions with the current infrastructure instead of just allocating more resources in an indiscriminate fashion.

Furthermore, considering the data analysis tools are on premises and the database is located on AWS as well as a possible migration for an all-cloud infrastructure I would first evaluate the current scenario as well as the scenario expected in the foreseeable future. I would not change the architecture if the tools are able to sustain the expected workloads and if the cost is justifiable. Note there is a huge difference between *an issue within the IT infrastructure* and *insufficient IT infrastructure*, being the latter not mentioned in the original text at all. On the other hand, if the scalability is paramount or if a bottleneck architecture-wise is foreseen I would focus in the migration instead. If the upper management believes the cost is not needed I would attempt to analyze what are the expected OPEX and CAPEX costs for the next cycle (3 up to 5 years depending on the company practices) should we remain with the current structure or modify it versus any expected performance and bottlenecks. I would also suggest reviewing the current contracts with our vendors in order to find out any possible discounts or possibilities available with better costs. Finally, I would also find other alternatives with small operational costs that would result in huge performance gains (e.g. adopting message queues in some operations). If these alternatives are possible to be implemented I would first suggest creating a MVP and using it as a use case for the upper management.

Last, should the Data Governance team recommends the prioritization of process documentation instead of refactoring 500 low-performing scripts I would suggest a parallel development composed of smaller steps considering both are relevant. Therefore, I would identify out of the 500 scripts the top 5 or 10 worst performing scripts and refactor them. Then, I would document this process immediately. It would reduce the pressure on both sides. Afterwards, I would focus on the next top 5 or 10 and resuming the documentation investing the same time, repeating this procedure until both cases are resolved.

Figure 4: Redshift schema containing the real tables. The tables containing the _*pkey* suffix were automatically created and are in charge of managing the primary keys.



Data objects

Select schema
Select a schema to view data tables
olist

Filter tables and views

1

customers
customers_pkey
orderheaders
orderheaders_pkey
orderitems
orderitems_pkey
orderpayments
orderpayments_pkey
orderpaymenttypes
orderpaymenttypes_pkey
orderreviews
orderreviews_pkey
orderstatus
orderstatus_pkey
productcategories
productcategories_pkey
products
products_pkey
sellers
sellers_pkey
zipcodes
zipcodes_pkey