Caspar Health Technical Challenge

Patient Data ELT

Jon Fernández Bedia

Jon Fernández Bedia

Table of contents

1. Goal of the project

The goal of this project is to showcase an end to end ELT pipeline from a data source to any data warehouse/data source using Python, SQL and Git to answer the following question:

- Find the patient(s) with the most generated minutes.

In order to answer this question, you've been provided with the following three datasets: - steps.csv - ID: table ID - External_id: patient_id - Steps: number of generated steps per submission time - Submission_time: the time when the steps are submitted - Updated_at: last time when the row was updated - exercises.csv - ID: table ID - External_id: patient_id - minutes: total duration of an exercise in minutes - completed_at: the time when the exercise was completed - Updated_at: last time when the row was updated - patients.csv - Patient_id - First_name - Last_name - Country

To solve the problem above, these are the business rules, assumptions and some handy tips: - As you have probably already noticed, in the steps.csv dataset, there is no column with the completed minutes, but rather a column with the submitted steps. That's fine, as there is a business requirement saying how to convert steps into minutes using the following formula ->

- A single patient can submit steps multiple times, and can complete multiple exercises.
- Please use Python only for data extraction and injection, and use SQL for data manipulation.
- Have in mind that multiple patients can generate the same amount of minutes, meaning that the output table can in theory have more than 1 row
- The output should have the following columns

patient_id:int	first_name:str	last_name:str	country:str	total_minutes:int

Feel free to be creative and if you have knowledge of any of the following technologies (Docker, cloud services, AirFlow, DBT), feel free to use them as well in your solution. If not, no worries, you will learn them at Caspar :)

1.1 Deploy Airflow environemnt

2. Modus Operandi

The following document aims to gather all the steps I thought about during the development of the technical challenge.

3. Landing the challenge

First, I read the challenge several times to makes sure I understood the task. At first glance, it does not seem a very complex task, also the datasets are not very big so I suppose I will not have any issues processing the data.

The final output is based on the join of patients with steps and exercises accordingly and calculating the maximum sum of minutes (coming from steps and exercises) grouped by patient.

4. Infrastructure discussion

Since I am familiar with Airflow and dbt (I did no use it in 2 years so lets see what changed) and the challenge itself does not seem extremely complex I would love to invest some time in creating a proper infrastructure: - Dockerized Airflow and DBT with connection to Snowflake

However, I have never deployed Airflow locally from scratch (everywhere I went, it was already there) so I will leave deploying dbt in Airflow for the end.

I decided to use Snowflake as database based on: - I did not use it before and I want to learn a new DB engine. - It was listed in the nice-to-have list of the job offer.

Also, I would like to implement data QA tests (table constraints, business criteria and schema checks), a CI/CD pipeline to automate python tests, add renovate bot to the repository and documentation deployment to the git project ,but we will see how it goes.

Not sure if I will create a dashboard for the results, but it would be nice as well. I have some ideas for other KPIs: - Apart from , we can show as well the minutes coming from steps and the ones coming from exercises, it could be a useful KPI to get more insights about the rehabilitation. - We can also split the maximum total minutes per country and, as in the previous point, split it in exercises and steps. - Steps and exercises submission_time(s) graphs (respectively) , it would be great to know when (along the year) are usually the patients doing more steps and exercises, maybe it has some effect on their rehabilitation time.

5. Setting up Github

I am not sure if it is relevant but as I am using a laptop with an already paired Gitlab account, I had to create another SSH key and assign this new SSH key to the Github account I am going to use for the challenge:

And here add:

In Github SSH configuration section add the content of as always. Finally, and I think this changed recently, I had to log in with github-cli .

6. Setting up Snowflake

I am starting by setting up the Snowflake user for dbt as recommended in one of the Snowflake's quickstart guides.

6.1 Early data model approach

After that, we create the data model for our patients, exercises and steps. The assumptions and standards I chose to follow:

- We prefer plural from singular table namings so as SQL code is more intuitive:
- We prefer explicit over implicit type definition (i.e. we use instead of so the amount of decimals is properly defined in the code even though is the standard in Snowflake for numeric data types)
- Same goes for string data types, we will be using which would be the same as using but we rather define the maximum length of the field in our code.
- It would be great to align with the team in charge of building the source of the data so as we can define properly the limits of the values in the columns and use it as a second type validation.
- In exercises and steps tables, column names will be modified to .
- For the timestamp columns, I used TIMESTAMP_TZ Snowflake type since, in the data, it looks like the UTC offset is defined after the timestamp (i.e.). However, we will need to take into account that section defined in the documentation when using this field for creating KPIs, since the offset of some countries change during the year but not the value of the field. If possible I would ask the team in charge of creating the source data to send us the values of the TIMEZONE together with the timestamp (without the UTC offset in this case) so as we can calculate the UTC time in place when needed.
- In steps tables, column name will be modified to , like that all the columns with type will have the same suffix and we will be able to identify the type of the column by its name.
- When trying to test how Snowflake is reading the timestamp values from the spreadsheet I got so I tried:

It looks like it can read it know, we will take care of this when importing the data to Snowflake.

The table creation script I used for the staging tables is the following:

7. Setting up DBT

7.1 Setting up the Python environment

I am going to be using uv as a python package manager to start with the dbt dependencies. It is being a while since I wanted to try uv out, is supposed to be very fast.

**Yes, it was fast indeed.*

I am not using since looks expensive for what it offers and it is not really complicated to set up and deploy but maybe I regret it. I run to create the dbt project. Then, to install the dependencies ( ). The dbt profile would look as follows:

It took me some time to figure the Snowflake connection parameters out to create the dbt profile, more specifically; - **account**: I had to run a query on Snowflake to get it.

- **password and user**: I did not know if it was the Snowflake account or the database specific account.

I added a and macros as recommended in the Snowflake quickstart guide I am following.

7.2 Loading raw data

After setting up dbt, I decided to try to import the raw data into Snowflake. I added the CSV files into the folder and tried to run but I got the following error:

And since I could not see the query, I went to the logs, and apparently I was trying to run but, of course, schema does not exist, it should be instead. For some reason, I thought that DBT was creating a new schema when adding data from seeds.

I took the decision of adding the suffix to the tables names containing raw data as specified in dbt documentation. I learned that, according to this post, apparently is not the greatest option to bulk raw data into Snowflake and, honestly, I would rather define a dbt model with COPY INTO table clause, but I do not have any personal cloud storage account so I am going with dbt seeds for this very specific challenge.

The data bulk worked with no issue for and , however, for , I had to use option (only for the very first time we bulk the data) since the 1st column name is empty. Due to that, the 1st column name for this table will have as a column name instead of as planned. We could have avoided this by loading the data directly from a cloud storage and just not selecting that column. I am not planning on using the column anyway in further tables, so it should not be an issue.

7.3 Dimensions and facts

Once our raw data has been loaded, is time to discuss which transformations we will be doing to our data.

For this specific case and with no further knowledge about future requirements, I would go with a simple dimensions and facts data model design where Patients will be the main dimension and Steps and Exercises the facts that do not make sense without our dimension. The data model design has not a big impact in this specific case apart from helping understand our data and defining the primary keys (in) and foreign keys (in and).

This tables will be kept in schema (as defined in the Snowflake guide we are following) since they are part of the base layer of our model, and they all should have a or raw data as source. In this state we will be taking care of the transformations mentioned in section Early data model approach and automatic dbt data QA tests defined on the table constraints such as: - primary keys (unique, not null) - foreign keys (referenced to the origin) - type validation

I added constraint to almost every field in the data model because the data shows that it is possible, however, it would be great to confirm and truly redefine which columns expect values and which do not.

Since we do not have any constraint in one model that applies to multiple columns, we will be only defining constraints and not constraints. Also, we are going to use one file per model to define the schema instead of defining all the schemas in the same file so the code structure scales up in a clean and organized way.

7.4 Analysis tables

Finally, we are building two analysis tables and we will be using schema for that.

First, we will join patients data with steps and exercises data so as we can group it and sum it. We will also be creating the KPI , a sum of the minutes coming from steps and exercises, so as we can get the maximum value(s) in the last results table. And last but not list, we will create a column with a window function ordered by descendant, which will return the same value in case the maximum value is repeated for different patients.

Second, we will obtain our aimed KPI filtering the rank field we created in the previous table with .

patient_id:int	first_name:str	last_name:str	country:str	total_minutes:number(38,3)
356134	Austin	Ellis	Germany	54954235.000

*_ column is decimal type (and it was proposed to be type) since one of the columns involved in the KPI had 3 decimals as well.

I would love to create more KPIs as defined in Infrastructure discussion section, but I am going to focus on deploying dbt in Airflow, adding python pytest tests, sqlfluff, ruff and proper documentation first.

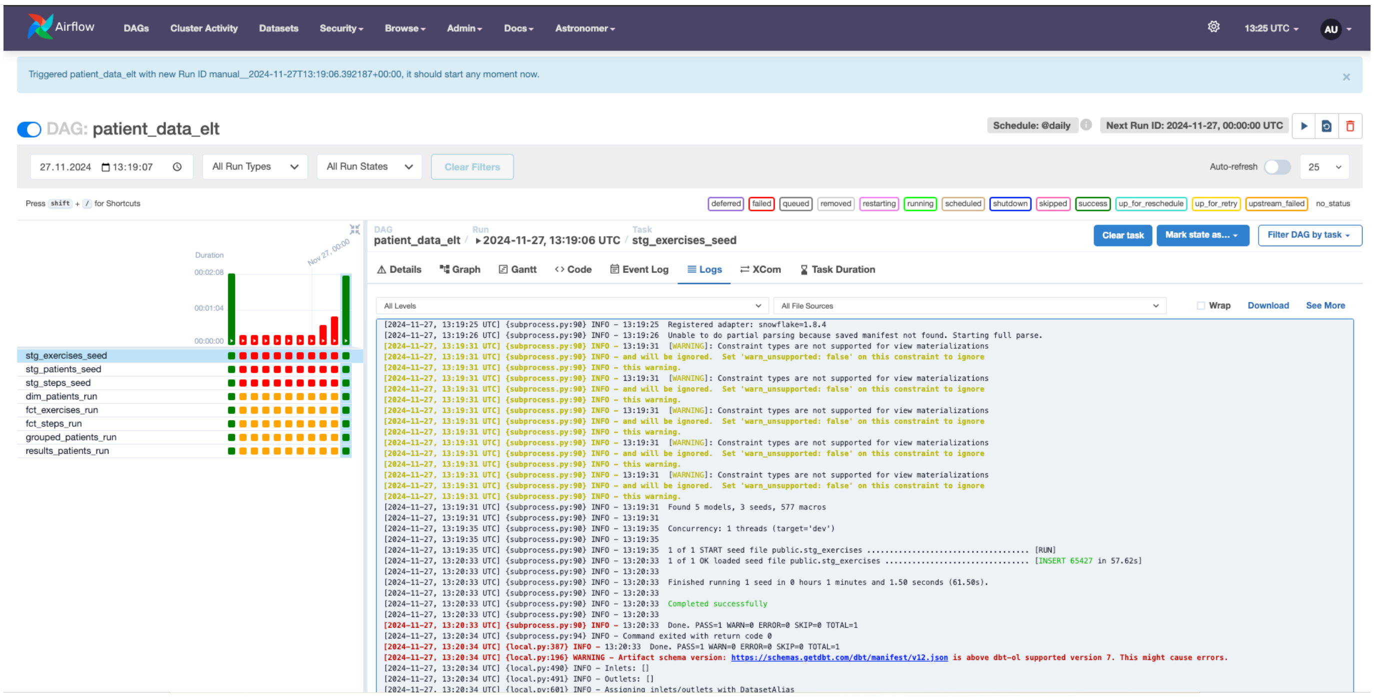
8. Setting up Airflow

Following the Snowflake guide we might need to get rid of uv for setting up python requirements and go with plain pip requirement management. We will now prepare the repository for encapsulating our model creation with dbt inside an airflow repository ready to be deployed.

We can create a dockerized airflow repository by creating a new folder and running (from astro), after that I will just add all my dbt packages inside folder.

8.1 Setting up the DAG

The DAG is only composed by: 1. Dbt profile definition: Which basically, thanks to a very cool cosmos library, reads all the credentials from the predefined Airflow connections in . Given the use case, I am not going to spend more time trying to hide the secrets since in a real-life scenario I would just use Airflow AWS Secrets Manager Backend for getting the credentials of the connections. 2. Dbt DAG: We create tasks that run every existing model in DBT, even the seeds creating a module.



Triggered patient_data_elt with new Run ID manual__2024-11-27T13:19:06.392187+00:00, it should start any moment now.

```
[2024-11-27, 13:19:25 UTC] {subprocess.py:90} INFO - 13:19:25  Registered adapter: snowflake=1.8.4
[2024-11-27, 13:19:26 UTC] {subprocess.py:90} INFO - 13:19:26  Unable to do partial parsing because saved manifest not found. Starting full parse.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  [WARNING]: Constraint types are not supported for view materializations
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - and will be ignored.  Set 'warn_unsupported: false' on this constraint to ignore
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - this warning.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  [WARNING]: Constraint types are not supported for view materializations
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - and will be ignored.  Set 'warn_unsupported: false' on this constraint to ignore
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - this warning.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  [WARNING]: Constraint types are not supported for view materializations
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - and will be ignored.  Set 'warn_unsupported: false' on this constraint to ignore
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - this warning.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  [WARNING]: Constraint types are not supported for view materializations
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - and will be ignored.  Set 'warn_unsupported: false' on this constraint to ignore
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - this warning.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  [WARNING]: Constraint types are not supported for view materializations
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - and will be ignored.  Set 'warn_unsupported: false' on this constraint to ignore
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - this warning.
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31  Found 5 models, 3 seeds, 577 macros
[2024-11-27, 13:19:31 UTC] {subprocess.py:90} INFO - 13:19:31
[2024-11-27, 13:19:35 UTC] {subprocess.py:90} INFO - 13:19:35  Concurrency: 1 threads (target='dev')
[2024-11-27, 13:19:35 UTC] {subprocess.py:90} INFO - 13:19:35
[2024-11-27, 13:19:35 UTC] {subprocess.py:90} INFO - 13:19:35  1 of 1 START seed file public.stg_exercises .................................... [RUN]
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33  1 of 1 OK loaded seed file public.stg_exercises ................................ [INSERT 65427 in 57.62s]
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33  Finished running 1 seed in 0 hours 1 minutes and 1.50 seconds (61.50s).
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33  Completed successfully
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33
[2024-11-27, 13:20:33 UTC] {subprocess.py:90} INFO - 13:20:33  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
[2024-11-27, 13:20:34 UTC] {subprocess.py:94} INFO - Command exited with return code 0
[2024-11-27, 13:20:34 UTC] {local.py:387} INFO - 13:20:33  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
[2024-11-27, 13:20:34 UTC] {local.py:196} WARNING - Artifact schema version: https://schemas.getdbt.com/dbt/manifest/v12.json is above dbt-ol supported version 7. This might cause errors.
[2024-11-27, 13:20:34 UTC] {local.py:490} INFO - Inlets: []
[2024-11-27, 13:20:34 UTC] {local.py:491} INFO - Outlets: []
[2024-11-27, 13:20:34 UTC] {local.py:601} INFO - Assigning inlets/outlets with DatasetAlias
```

8.2 Astro and cosmos

I did not use and until today and is crazy how easy they make: - **Astro**: It deploys the webserver, the scheduler and the trigger (I guess this is the Celery queue). I just did not need to think about it, which is nice for this use case. However, I can imagine that those layers of abstraction that adds on top of Airflow might be a bit cumbersome when we want to adjust Airflow to some specific needs. Also, it took me quite some time to discover how to deploy the connections defined on :

- **Cosmos**: I really do not have any complains about it for now but again, it could be not the best option if we want to really get the most out of dbt.

9. Setting up tests (TNDD)

Well, I would have loved to apply TDD during the development of these tasks, however, I got excited by the use of uv at the beginning and I forgot about it. That is why we will can call it Test Non Driven Development in this case.

I did not add any python tests myself since I did not see the case for it, maybe it would have been great to: - Test somehow the connection with Snowflake (with a given dbt profile) - Test the output of the dbt models somehow.

What I did add is sqlfluff and ruff, which are very nice SQL and python linters that help so much following predefined code standards.

10. Faced problems

This section aims to gather the most relevant problems I faced during the development of the technical challenge. However, I believe most of them were already described in the *Modus Operandi* documentation.

10.1 Deploying Airflow

Deplying Airflow is a bit of a hustle having in mind that you need at least three containers running as services (scheduler, webserver and executor) and another two containers for databases (airflow and celery).

10.2 Raw data bulking

When developing locally, it would be great to have access to some kind of cloud storage to be able to replicate an end-to-end system behaviour without the need of using big platforms that yes, have free tiers but also ask for you card number.

11. Requirement manager

I really like uv python manager is very fast indeed, however, I felt a bit lost when trying to dump or compile the packages in uv to a requirements.txt file and making a 3rd party docker image not explode while trying to read this requirements.txt. Maybe it gets solved by building myself the docker image but for sure I was not able to fully fix this issue during the challenge.

12. Future implementation(s)

The following section aims to gather all those point I was not able to finish and that I think it would be great to implement some day.

1. I would definitely not use and go for writing my **own docker image for Airflow** so as I can handle dependencies better and in order to know what is going on behind the scenes.
2. I would also like to have **separate dependencies** for developing the code and code that is being pushed to production. There must be a way of defining those with uv as we can do with pipenv.
3. I would be nice to have **CI/CD pipelines** that everytime we push code to the repository, the run pytest and our linters. Also it would be great to have a job that creates a Github Page out of our documentation as in Github Pages. I tried but I think it does not work with private repositories
4. I would like to add **more tests** as I already mentioned in the in the *Modus Operandi* documentation and maybe follow TDD somehow next time.