ACA Project headway

1. Construct database structure and think about entities that should be created.
2. Prepare to work with AWS Cloud
3. Create a database and entities inside it about which the structure should have been ready
4. Fill with data database all entities with data except “Transaction tables”
5. Create scripts that will generate data for “Transaction tables” using data stored in other tables
6. Data generation using AWS Lambda and scheduling
7. Create data pipeline using our database as source and AWS S3 as staging area or Data Lake

Paragraph 1.
Create database structure, entities and dependencies.

There is a typical model of a database of films and actors, my database is about Anime. The main entities in this database are Anime, their genres, studios that animated, creators of the original, voice actors, languages, licenses and users. As “Transaction tables” I have Ratings table and Wishlist table.
Each user has the opportunity to make this or that anime for himself of interest, and for this there is a Wishlist table where the names of the anime that the user either wanted to watch, or in the process, or has already done, are recorded. And using this information, the business logic decides whether the user has the right to rate this anime for which the Ratings table is used.

Paragraph 2.
Start working with AWS Cloud.

First, I created AWS account and after that I read some documentation and watch some videos from YouTube about AWS services (RDS, cloud9, lambda, event bridge and others).

Paragraph 3.
Creation of Database and its entities.

Created Postgres database “AnimeDataBase” in my AWS instance, after started to create my entities with its dependencies. After that, it was necessary to somehow deal with the fields create and modified dates, so that, for example, if a new user appeared, we could find out when he was created and if after that he changes something, we could track it. And so, I decided to write a trigger that would change the modified date field every time the user changes information about himself. This kind of trigger I wrote for “Animes”, “Creator_Origins”, “Users”, “Wishlist”, “Raitings” tables.

Paragraph 4.
Fill data in non-transactional tables.

In order to fill my tables with data, I just wrote a script that downloads all html files from certain sites in which the data that I need is located. After that, I wrote scripts for each table in C # which filtered the data for these tables in its own way from the html files and generated the Insert command for each of the tables.

Paragraph 5.
Fill data in non-transactional tables.

In order to fill these tables with data, I wrote python scripts that generate data using the above tables in accordance with certain business logic. I used “PSYCOPG2 ” libraire for creating connection to my database

Paragraph 6.
Data generation using AWS lambda and scheduling.

The task was to put the above scripts in AWS Lamda. And in order for them to work, it was necessary to create AWS Lambda Layers for all the libraries that I used for each Lamda. To do this, it was necessary to create an environment using AWS Cloud9 and already using this environment to create Layers. After that, it was necessary to put a scheduler over these Lambdas that would simulate the appearance of new transactions, and for this I used the AWS Event Bridge and connected it with my Lambdas.

Paragraph 7.
Create data pipeline using our database as source and AWS S3 as staging area or Data Lake.

I need to do data incremental load from my source database to AWS S3 storing data in parquet format. And for this, I created a separate table as metadata about all my other tables, which the script uses in order to understand what type this or that table belongs to in order to understand from what point the new data should go. In this table, I have two metrics that my script uses to understand how to get new data: maxmodified and maxid. These columns record the data of the last row of each table according to its own metric, for some it is the Id of the table whose rows do not change but increase and the ModifiedDate of the table whose rows can change. This script takes from this table all the necessary metadata to properly implement the Incremental load of data. After the data is taken, the script converts it into parquet format and put these files in AWS S3 storage. After that, only the script refreshes the data in the table with meta information, thereby announcing that the process has been successfully completed and the metadata can be changed. All this happens within the framework of one operation, for example, as a transaction, which means that if something goes wrong, then everything is canceled.