Athena Challenge

For this challenge, I've utilized a multi-container setup defined in the `docker-compose.yaml` file to perform the ELT task. The first container runs Mage, the orchestration tool, while the second container hosts a MySQL database. Mage, which functions similarly to Airflow, uses concepts like tasks and DAGs, but refers to them as blocks and trees, respectively. Mage's block architecture makes it easier to understand and manage pipelines.

Below, I will describe how to execute the `athena_data_load__transform` pipeline, which performs the ELT job. In case you encounter any errors or are short on time, you can refer to the `answers.csv` file for the final output of the model.

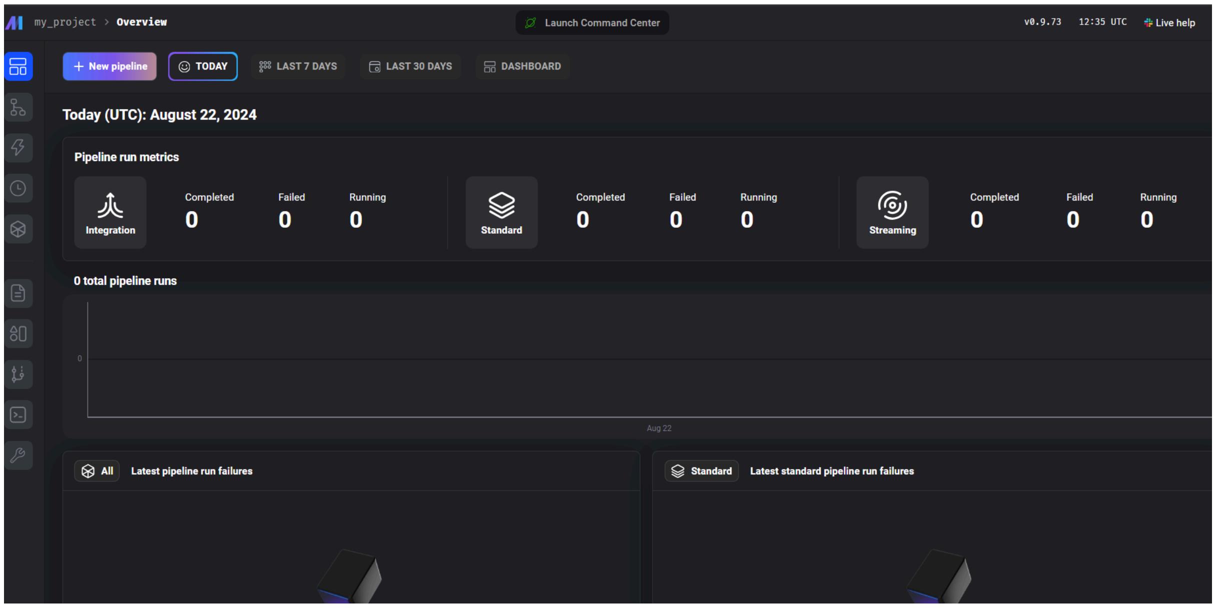
Steps to execute the whole thing:

Step 1: Clone the repo: https://github.com/rahulmopkar/Athena-challenge.git
Using git clone https://github.com/rahulmopkar/Athena-challenge.git

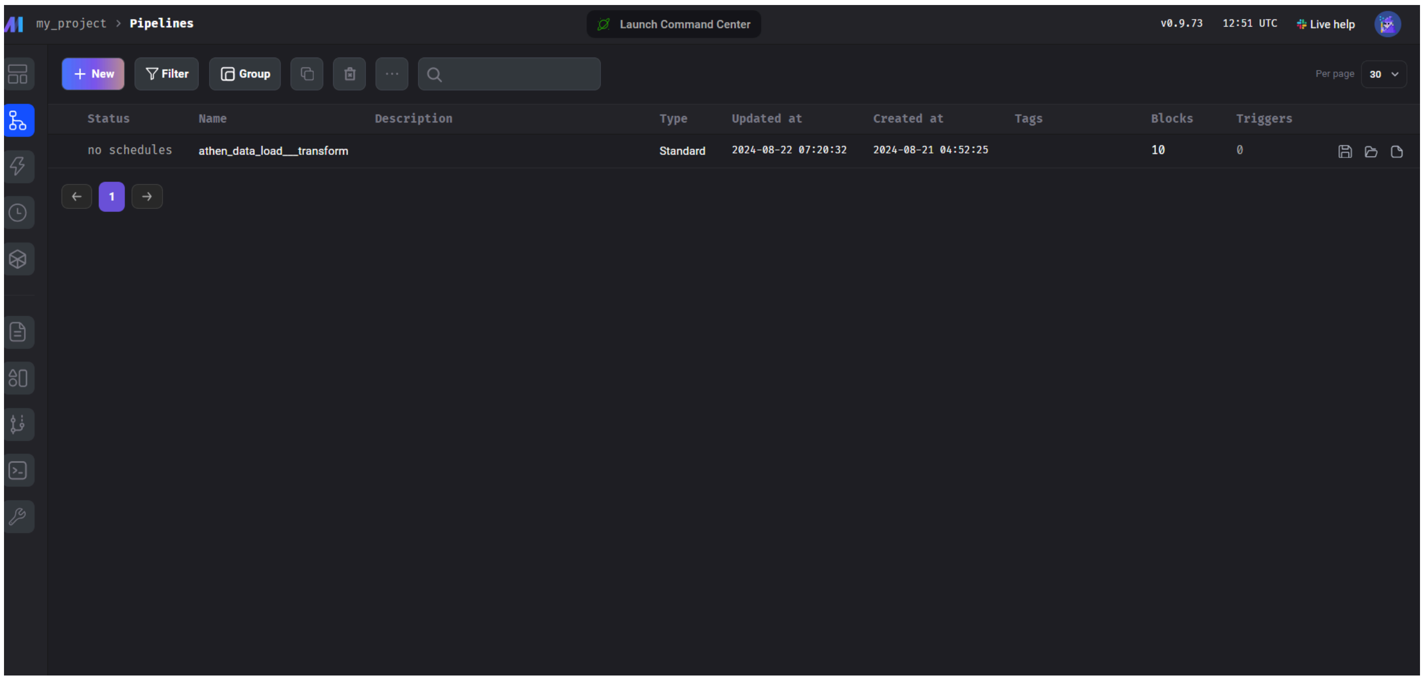
Step 2: Navigate to the folder and run the container.
Once cloned navigate to the folder Athena-challenge and execute the command docker compose up.
This command will get a mage container and a MySQL container set up with all the requirements like the database Athena-data, and run these containers.

Step 3: Open Mage in the browser.
Once this is done, and the container is running, you can go ahead and go to the link http://localhost:6789/ This will open up the mage interface where we can run the pipeline that is performing the ELT process, i.e. extracting the data from the CSV files in the data folder load them into tables and then generate the required final table from these.
(if the link does not open up in the first go give it a minute or two and press refresh)

Upon clicking on the above link you should see an interface similar to below:



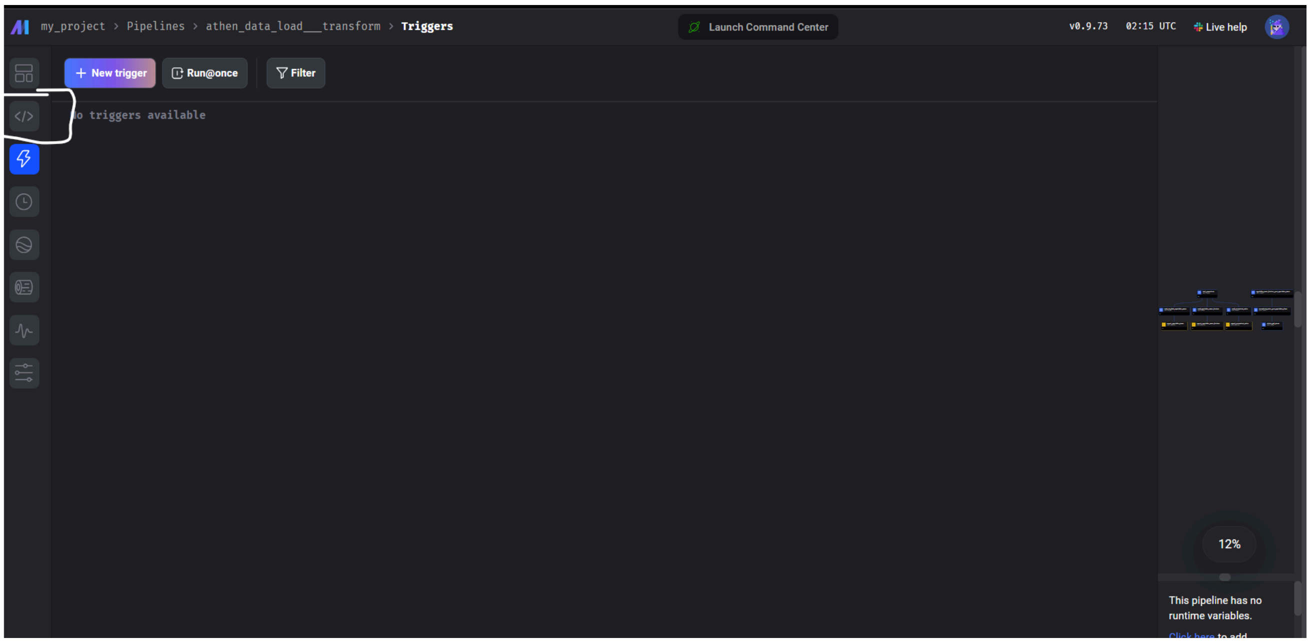
Click on the second icon that looks like a branch on the sidebar on the left. This will Open up a screen similar to below:



Status	Name	Description	Type	Updated at	Created at	Tags	Blocks	Triggers
no schedules	athen_data_load__transform		Standard	2024-08-22 07:20:32	2024-08-21 04:52:25		10	0

Step 4: Run the pipeline:

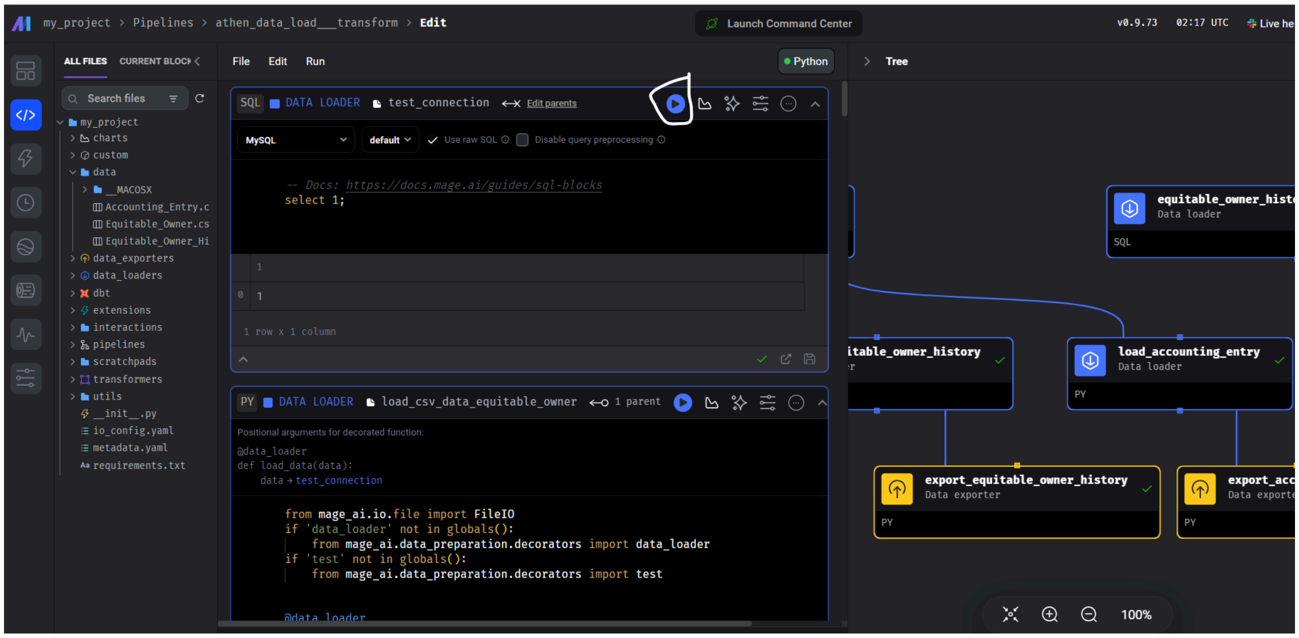
Once on the screen above click on the pipeline athena_data_load__transform, after this, you will be greeted by the screen below, since we haven't set up a trigger we will need to run this pipeline manually, to do this click on the icon circled by the white border.



No triggers available

This pipeline has no runtime variables.
Click here to add

When clicked on the icon above we are taken to the screen below:



```
-- Docs: https://docs.mage.ai/guides/sql-blocks
select 1;
```

```
1
1
1 row x 1 column
```

Positional arguments for decorated function:

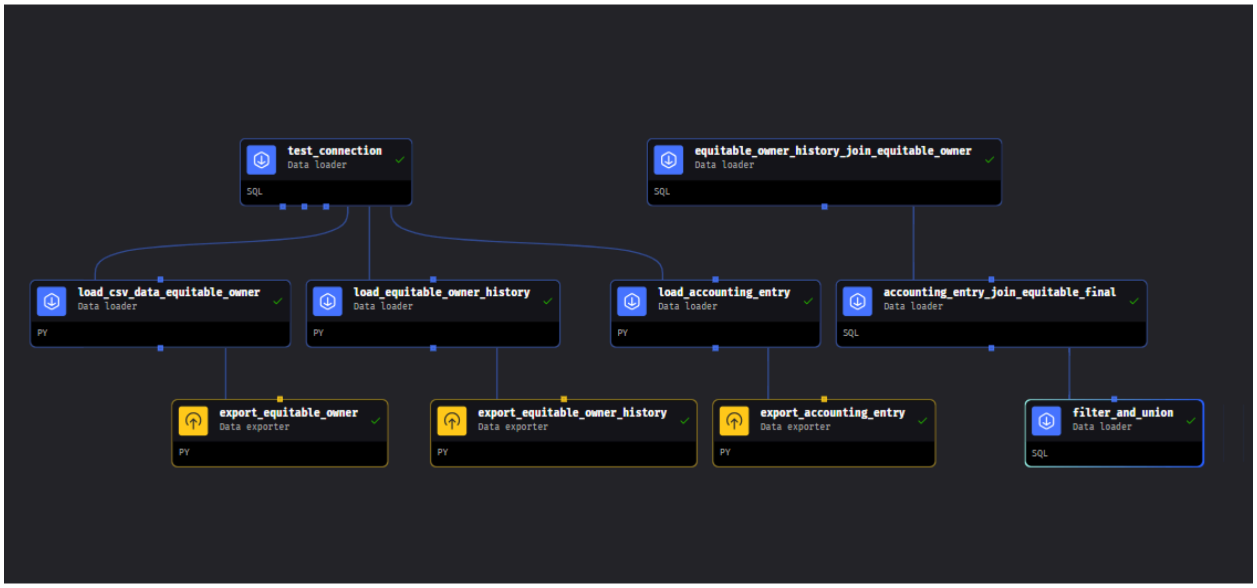
```
@data_loader
def load_data(data):
    data → test_connection
```

```
from mage_ai.io.file import FileIO
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@data_loader
```

Mage pipeline works in blocks, similar to tasks in airflow we execute each block by pressing the play button as described in the picture above, to get the final answer we will need to execute each block, let's go through what each block does before we go ahead and execute them.

To understand that let's take help from the tree structure of the blocks which resembles a DAG in airflow.



Step 5: Understanding what each block does:

1. test_connection: Since we have a multi-container setup,
   Mage - our orchestration tool for the ELT data pipeline and
   Mysql - Our local database,
   first things first we need to test if the mage container responsible for performing the ELT job can communicate with the mysql container a successful execution of this block tells us that.

2. load_csv_data_equitable_owner: This extracts the data from the Equitable_Owner.csv file.

3. export_equitable_owner: This exports the data extracted in step 2 into a table named Equitable_owner, inside a database named Athena_data.

4. Same as above applies for the blocks, load_equitable_owner_history, export_equitable_owner_history, load_accounting_entry, export_accounting_entry with tables Equitable_Owner_history and Accounting_Entry respectively in the database Athena_data.

5. The equitable_owner_history_join_equitable_owner creates a CTE that joins the tables equitable_owner_history and equitable_owner on the keys:

   Equitable_owner.id = equitable_owner_history.equitable_owner_id,
   The reason for selecting this as the join key was, that it is not null and provides a reliable source of confirming the relationship between the 2 tables.
   The result of this CTE is available in the downstream block accounting_entry_join_equitable_final, where we join the result with the table accounting_entry, to get a final join.

6. The last block is a block that performs the filter operation per the following conditions:
   - equitable owner record reallocation date is the same as the accounting entry transaction date or,
     - The CTE filter_for_rellocation_date_equal_transaction_date. We use the keys reallocation_date and the transaction_date as the join keys, the {{ df_1 }} signifies that we are using the output from the upstream block.
   - equitable owner record reallocation date is the most recent to the accounting entry transaction date.
     - The CTE is not_equal, in this CTE first things first we get rid of the IDs that have reallocation_date = transaction_date as we get these from the first CTE this is done using the subquery, and then we rank each of the IDs using the row_number window function by doing partition by on id and order by on reallocation_date since we need the most recent one, since for every id the transaction_date stays the same we can use the row_number window function and select the one with rank 1, which is done in the final select statement with row_num = 1.

   This is where the main logic to create the new model comes into play specifically the query:

```
with filter_for_rellocation_date_equal_transaction_date as (
   select equi_id, id, transaction_date, reallocation_date,
   equitable_owner_name from {{ df_1 }}
    where reallocation_date = transaction_date order by id
),

not_equal as (
 select equi_id, id,
 row_number() over (partition by id order by reallocation_date desc) as
 row_num, transaction_date, reallocation_date, equitable_owner_name
 from {{ df_1 }}
 where id not in (
 select id from filter_for_rellocation_date_equal_transaction_date
 )
)

select equi_id, id, transaction_date, reallocation_date, equitable_owner_name
from not_equal where row_num = 1
union
select equi_id, id, transaction_date, reallocation_date, equitable_owner_name
from filter_for_rellocation_date_equal_transaction_date;
```

And creates a final table Equitable_Owner_Details by unioning the result of the two CTEs that tackle the two given conditions.

Step 6: Run the blocks or check the CSV file for the final result.
Go ahead and execute the blocks one after the other, in the sequence they appear.
We can view the table results directly inside mage after executing the last block, but this will display only a limited set of rows to get the full view, let's go to the MySQL docker container by executing the following:

docker exec -it my_project-mysql mysql -u my_user -p

When prompted for password add in my_password.

Now that we are inside the MySQL container let's have a look at the final table and the answer For this challenge by doing:
select * from Athena_data.Equitable_Owner_Details;

Alternatively, you can have a look at the CSV file (answer.csv) attached to the git repo.

Stretching:

1. How would you test that this model is behaving predictably in a production environment?

I would put a test on the final table to see if it has the same number of rows as the Accounting_Entry table, this could be easily done using dbt by putting a SQL test inside the test folder.

2. How would you deploy this model to create a dataset that would be accessible for analytics?
   I would use the table created Equitable_Owner_Details, and I would use Snowflake for the whole procedure instead of MySQL, and get the data in a power bi report use DAX queries to read the table.

Addressing Security Concerns:

To simplify the setup and execution of the pipeline for the purpose of this test, I have included a `.env` file with the necessary passwords and credentials directly in the repository. While I fully understand that this is not a standard or secure practice, and that credentials should never be committed to source control, I've taken this approach solely for ease of testing within this private repository.

Given that this repository is private and only accessible to the Athena Engineering team and myself, the risk is minimized. However, in a production environment, sensitive information should always be managed securely using environment variables or secret management tools like AWS Secrets Manager or github secrets.