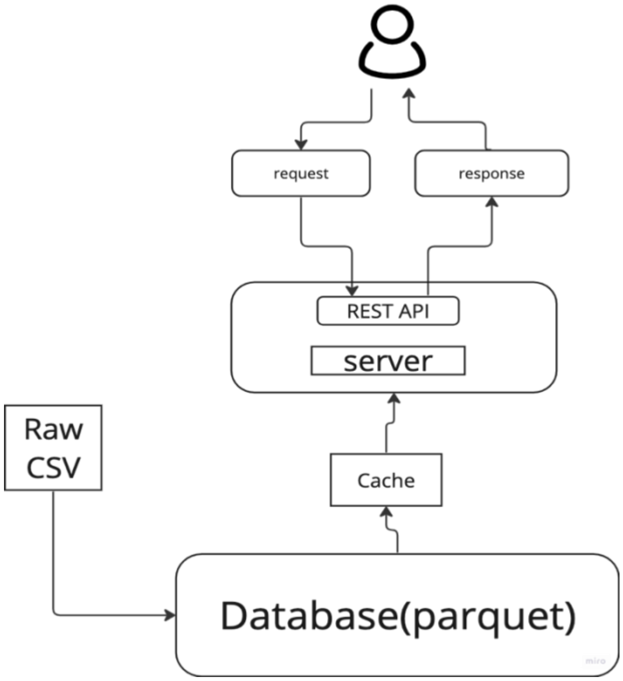
Architecture:



ETL pipeline:

Let's describe as the data flows.

The combined_transactions.csv file is read and then saved in parquet format, now the parquet files are only partitioned by transaction_type, but in the future when data becomes huge, we can add a partition by date.

Why Parquet: Parquet is a column-based storage format. If the data has too many columns, we don't have to read all of the data into memory and filter them, we just need to read in a few columns that's needed in the downstream calculations. This will significantly reduce the RAM usage and improve the performance.

Apart from that, Parquet file is more compressed than a SQLITE file, which leads to fewer IO.

A good thing in SQLITE is that humans can view the data easily, but this feature is not that important in our scenario.

So that's why I used Parquet files as my database. In my database, I saved all data with deducted data types. The types are deducted from the descriptions in the assessment.

Keeping the data types in database is important and helpful, because you just need to run it once, then every request can understand the data in the same way. Imagine you standardize the data types in your server, or determine them on the fly, it will be a performance killer since there'll be millions of requests coming in.

The ETL pipeline unit tests were passed.

Server:

After saving data with the right format, let's go to the place that will ask for data, the server.

I built this server using Flask, a very convenient Python web framework. The server is now simpe and light-weighted but can be scaled. I just exposed two endpoints where users can request (1. transaction data given user_id and (2. daily net amount data given merchant_type_code.

The first endpoint is `/api/v1/data/transactions/<userId>`. You can replace userId with an integer user_id. For example: http://127.0.0.1:3025/api/v1/data/transactions/31373

If your user_id is not an integer, error message will be returned.

The second endpoint is `/api/v1/data/netAmount/<merchantTypeCode>`. You can replace merchantTypeCode with an integer merchant_type_code. For example: http://127.0.0.1:3025/api/v1/data/netAmount/5732

If your merchant_type_code is not an integer, error message will be returned.

I used '/api/v1/data' to indicate the API, version and service I'm looking for, after that I add the data object name to it, meaning which type of data I'm looking for. Then I added the path parameters to indicate the query concepts.

The Server unit tests were passed.

The Postman & browser tests were passed.

Setup:

Clone the repo:

```
git clone https://github.com/pipipuppypaul/de-workbook  kasheesh-de-workbook
```

Install third-party packages:

```
cd ./kasheesh-de-workbook
pip3 install -r requirements.txt
```

Prepare data:

```
python3 ./src/etl_pipeline.py --input_file /Users/yuhaibo/Downloads/combined_transactions.csv --output_file /Users/yuhaibo/workspace/kasheesh-de-workbook/database
```

launch the server:

```
flask --app server.py run --port 3025
```

send data request:

1. http://127.0.0.1:3025/api/v1/data/transactions/<user_id>
2. http://127.0.0.1:3025/api/v1/data/netReturns/<merchant_type_code>

Follow Up:

1. To further improve the performance of server and APIs, we can add Cache to the system to reduce IO cost.
2. To better manage the services and APIs, we need to maintain a well-structured API documentation.
3. To handle large amount of request, we need API gateway, load balancer.
4. If we have a large amount of data, we may want to add partition to the parquet files and add index to the frequently searched keys.
5. If we want to modify the csv or parquet data, for example update or append, we hope we don't shut down the server.