equal-experts-logical-attentive-profuse-permission-097291b93635/

├── Dockerfile

├── README.md

├── poetry.lock

├── pyproject.toml

├── equalexperts\_dataeng\_exercise/

│ ├── \_\_init\_\_.py

│ ├── ingest.py

│ ├── outliers.py

│ ├── db.py

├── tests/

│ ├── \_\_init\_\_.py

│ ├── db\_test.py

│ ├── exercise\_tests/

│ ├── \_\_init\_\_.py

│ ├── test\_ingestion.py

│ ├── test\_outliers.py

├── uncommitted/

│ ├── votes.jsonl

It looks like you have successfully run the tests using `pytest` and they passed. Now, let's summarize the steps you need to take to finalize your solution and complete the assignment.

### Step-by-Step Approach to Finalize Your Solution:

1. \*\*Write Tests for Ingestion\*\*:

- Ensure you have tests for the ingestion process in `tests/exercise\_tests/test\_ingestion.py`.

- Here is an example of what the test might look like:

```python

import duckdb

import pytest

import json

@pytest.fixture(scope='module')

def db\_connection():

conn = duckdb.connect(':memory:')

conn.execute("CREATE SCHEMA IF NOT EXISTS blog\_analysis;")

conn.execute("""

CREATE TABLE IF NOT EXISTS blog\_analysis.votes (

Id VARCHAR PRIMARY KEY,

PostId VARCHAR,

VoteTypeId VARCHAR,

CreationDate TIMESTAMP

);

""")

yield conn

conn.close()

def test\_ingestion(db\_connection):

conn = db\_connection

# Insert test data

test\_data = [

{"Id": "1", "PostId": "1", "VoteTypeId": "2", "CreationDate": "2022-01-02T00:00:00.000"},

{"Id": "2", "PostId": "1", "VoteTypeId": "2", "CreationDate": "2022-01-09T00:00:00.000"},

{"Id": "4", "PostId": "1", "VoteTypeId": "2", "CreationDate": "2022-01-09T00:00:00.000"},

{"Id": "5", "PostId": "1", "VoteTypeId": "2", "CreationDate": "2022-01-09T00:00:00.000"},

{"Id": "6", "PostId": "5", "VoteTypeId": "3", "CreationDate": "2022-01-16T00:00:00.000"},

{"Id": "7", "PostId": "3", "VoteTypeId": "2", "CreationDate": "2022-01-16T00:00:00.000"},

{"Id": "8", "PostId": "4", "VoteTypeId": "2", "CreationDate": "2022-01-16T00:00:00.000"},

{"Id": "9", "PostId": "2", "VoteTypeId": "2", "CreationDate": "2022-01-23T00:00:00.000"},

{"Id": "10", "PostId": "2", "VoteTypeId": "2", "CreationDate": "2022-01-23T00:00:00.000"},

{"Id": "11", "PostId": "1", "VoteTypeId": "2", "CreationDate": "2022-01-30T00:00:00.000"},

{"Id": "12", "PostId": "5", "VoteTypeId": "2", "CreationDate": "2022-01-30T00:00:00.000"},

{"Id": "13", "PostId": "8", "VoteTypeId": "2", "CreationDate": "2022-02-06T00:00:00.000"},

{"Id": "14", "PostId": "13", "VoteTypeId": "3", "CreationDate": "2022-02-13T00:00:00.000"},

{"Id": "15", "PostId": "13", "VoteTypeId": "3", "CreationDate": "2022-02-20T00:00:00.000"},

{"Id": "16", "PostId": "11", "VoteTypeId": "2", "CreationDate": "2022-02-20T00:00:00.000"},

{"Id": "17", "PostId": "3", "VoteTypeId": "3", "CreationDate": "2022-02-27T00:00:00.000"}

]

for record in test\_data:

conn.execute("""

INSERT INTO blog\_analysis.votes (Id, PostId, VoteTypeId, CreationDate)

VALUES (?, ?, ?, ?);

""", (record['Id'], record['PostId'], record['VoteTypeId'], record['CreationDate']))

# Verify data ingestion

result = conn.execute("SELECT COUNT(\*) FROM blog\_analysis.votes").fetchone()

assert result[0] == len(test\_data), f"Expected {len(test\_data)} rows, but got {result[0]}"

```

2. \*\*Answer Follow-up Questions\*\*:

- Open the README file and provide answers to the follow-up questions about data quality, scaling, and assumptions.

\*\*Example Answers\*\*:

\*\*Data Quality Measures\*\*:

- Implement validation checks to ensure data completeness and consistency.

- Use primary keys to avoid duplicate records.

- Perform data type validation and format checks.

\*\*Scaling to 10TB Dataset\*\*:

- Use distributed processing frameworks like Apache Spark.

- Partition data and use efficient storage formats like Parquet.

- Optimize queries and indexing strategies.

\*\*Assumptions\*\*:

- Assumed that the input data is in JSON Lines format.

- Assumed that the `CreationDate` field is always in a valid timestamp format.

- Assumed that the `votes.jsonl` file is available in the `uncommitted` directory.

3. \*\*Review and Refine Your Code\*\*:

- Ensure your code is clean, well-commented, and follows Python best practices.

- Check that you've handled potential errors and edge cases.

4. \*\*Commit and Push Your Changes\*\*:

```powershell

git add .

git commit -m "Completed data engineering challenge"

git push origin main

```

5. \*\*Mark the Assignment as Completed\*\*:

- Use the provided link to mark the assignment as completed.

### Final Steps:

1. \*\*Run All Tests\*\*:

Ensure all tests pass by running:

```powershell

poetry run exercise check-ingestion

poetry run exercise check-outliers

```

2. \*\*Verify Your Solution\*\*:

Double-check that your solution meets all the requirements outlined in the challenge instructions.

By following these steps, you should be able to complete the data engineering challenge successfully. If you encounter any more issues or need further clarification, please let me know.

Citations:

[1] https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/12981205/084b5fb3-7a6e-421a-9534-c3b3f49803a9/votes.jsonl

[2] https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/12981205/1b8f3fb1-1600-45f5-9cda-b0bf272e2c80/paste.txt