UDACITY

< Return to Classroom

Data Modeling with Postgres

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 specifications require changes

Dear Student,

This is great progress. Most of the core tasks are done, and only a bit of fine tuning remains.

Kindly go through the suggestions thoroughly as they are very critical to the project. I hope you take them constructively; as an opportunity to learn and grow and also improve the overall quality of your submission.

Good luck. Stay Udacious.

Table Creation

The script, `create_tables.py`, runs in the terminal without errors. The script successfully connects to the Sparkify database, drops any tables if they exist, and creates the tables.
The script `DROP`s and `CREATE`s the required tables without any errors.

CREATE statements in `sql_queries.py` specify all columns for each of the five tables with the right data types and conditions.

You can implement `NOT NULL` constraint on the foreign keys in the CREATE statements. However, please be careful with this implementation when you implement the `INSERT` functions (see rubric "ETL script properly processes transformations in Python") since the dataset contains some null values.

You should run the tests under the `Sanity Tests` section at the end of the `test.ipynb` notebook to check your work for obvious errors.

Good job adding the right PRIMARY KEY, NOT NULLs and Data types in the columns.

To improve further, you can look into how you can add FOREIGN KEYs in your fact table.

1. Foreign Keys in PostgreSQL
2. Why should I use FOREIGN KEYs in database?

ETL

The script, `etl.py`, runs in the terminal without errors. The script connects to the Sparkify database, extracts and processes the `log_data` and `song_data`, and loads data into the five tables.

Since this is a subset of the much larger dataset, the solution dataset will only have 1 row with values for value containing ID for both `songid` and `artistid` in the fact table. Those are the only 2 values that the query in the `sql_queries.py` will return that are not-NONE. The rest of the rows will have NONE values for those two variables.

It's okay if there are some null values for song titles and artist names in the `songplays` table. There is only 1 actual row that will have a songid and an artistid.

You should run the tests under the `Sanity Tests` section at the end of the `test.ipynb` notebook to check your work for obvious errors.

Nicely done. The script runs without any errors, and populates all the tables appropriately.

INSERT statements are correctly written for each table, and handle existing records where appropriate. `songs` and `artists` tables are used to retrieve the correct information for the `songplays` INSERT.

You should run the tests under the `Sanity Tests` section at the end of the `test.ipynb` notebook to check your work for obvious errors.

Great job so far. There's just one small change required here.

For the `user_table_insert` you have put `DO NOTHING` when there is a conflict. But would we like to ignore it? The `level` field is used for `free` and `paid` services for a `user`. Consider someone subscribes by paying a fee who happened to be a `free` user before. In that scenario, we would like to update the `level` of the user to

`paid`. The current implementation will ignore the update, which is not desired. You can do that by `ON CONFLICT (user_id) DO UPDATE SET level=EXCLUDED.level`

Code Quality

- Create a `README` file with the following information:
    - a summary of the project
    - how to run the Python scripts
    - an explanation of the files in the repository
- `DOCSTRING` statements have been added in each function in etl.py file to describe what each function does.

Good work with the README file. You have added most of the information required by the rubric. There's just one point left out, "How to run the Python scripts". Please add this as well.

As an improvement, you can try adding in images, like screenshots of your final tables, ER diagram of the schema, and adding more details about your project, like a description of the dataset.

In the `etl.py` file, comments should be used effectively and each function should have a docstring. You should remove in-line comments that were clearly part of the project instructions, and include parameter descriptions and data types in the docstrings to improve the project.

An example of a Python function with proper docstrings

```
def add_binary(a, b):
    '''
    Returns the sum of two decimal numbers in binary digits.

            Parameters:
                    a (int): A decimal integer
                    b (int): Another decimal integer

            Returns:
                    binary_sum (str): Binary string of the sum of a and b
    '''
    binary_sum = bin(a+b)[2:]
```

```
    return binary_sum
```

While functionality is important, it's also equally important that you present your project in a professional and polished manner. That's the objective of this rubric.

Please check the following links for more info.

- Mastering Markdown
- Python Docstrings (With Examples)

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

Great job here. You have separated all the tasks into their respective functions whose names reflect what they do.

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

START