Meets Specifications

Congratulations !

The project meets expectation and passes all the rubrics. your hard work and patience has paid off.
I hope you had fun learning experience doing this project and is now is able to relate to relational databases like postgres.
In the next project, you will learn about Cassandra and how it is different from postgres.

I wish you all the best for next project.
Keep up the good work
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.

Good start.
I was able to run the script create_tables.py in the terminal without any error.
At the end of the execution, database and 5 tables were created flawlessly

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

Overall great job.

Table wise feedback

- Songplays
  - Primary key is correctly defined,
  - start time, user_id, level correct made as not null,
- Users
  - Primary key is correct.
- Artists-
  - Primary key is correct.
  - Latitude and Longitude is rightly set to numeric
- Songs Looks good
- Time
  - Primary key is correct.

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.

The script etl.py ran fine in the terminal.
At the end of execution, there was

- Database that contained five 5 tables.
- Each of the five tables were updated with records from log_ data and song_data file

Awesome work

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.

Nicely done.

All your insert statements are correct and uses on conflict during insert., For users we need to ensure level field is updated during each conflict, this is because user may become free user from paid or paid from free.

Code Quality

The README file includes a summary of the project, how to run the Python scripts, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.

Readme file is detailed enough to meet the expectation . This can be further enhanced

https://gist.github.com/PurpleBooth/109311bb0361f32d87a2

Detailed Readme file is important as it helps the user to present his/her case to viewer, Consider, your potential employer visits your github repo while evaluating you for the potential job, the readme file present your thought process behind the project and how you went about doing the project.

Impressive work

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

- Code is indented nicely and organised into different functions,
- There is an optimum level of comments in the code that helps describe the flow of the code.

Suggestion:

- Multi line docstring for function description
  Great work