

[◀ Return to Classroom](#)

Data Modeling with Postgres

REVIEW

CODE REVIEW 4

HISTORY

Meets Specifications



Dear Student,

A brilliant job following up on previous review comments. The submission demonstrates that you have a good understanding of the underlying dimensional modeling concepts in Postgres. Your code is very well written. The SQL queries are formatted properly, making them easy to read and follow.

Your project passes all the rubrics as per the project specification. I hope you learned a lot, but since learning never stops, I have added reading material on Bulk Load queries in Postgres. Also, a suggestion to generate ER diagram using `sqlalchemy` has been included. I hope you go through it.

- [fast-load-data-python-postgresql](#).
- [Pandas to PostgreSQL using Psycopg2: Bulk Insert Performance Benchmark](#)

These are an excellent read.

Finally, congratulations on completing the project! You should be very proud of your accomplishments in building an ETL pipeline for a star schema in Postgres!

Best Wishes!!

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.




- There are no errors in the 'create tables.py' file. It connects to the Sparkify database and, if any tables exists, the drop query effectively drops them.



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.

- **DATATYPE:** Excellent work with correctly selecting the data types for all the columns in tables. 
- **PRIMARY KEYS:** Correct fields are made PRIMARY KEY in all tables. 
- **NOT NULLS:** Nice job with the choice of adding NOT NULL constraint to the required columns. These constraints are implemented on database side, preventing NULL values for columns that must contain values. 

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.



EXCELLENT WORK

- The 'etl.py' file runs without errors. The connection is successfully established, and the script extracts data from the log files and loads it into the tables.



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.

- The insert queries are correctly handling conflict when data is insert into the tables. 
- For the user table insert query you are rightly updating the value for user `level` from `free` tier to `paid` tier. This helps keep track when user becomes a premium member. 



OPTIONAL

- If you'd like to do even more, consider inserting data using the COPY command to bulk insert log files instead of using INSERT on one row at a time. Here is a good read on bulk insert queries: [Fastest Way to Load Data Into PostgreSQL Using Python](#)

Code Quality

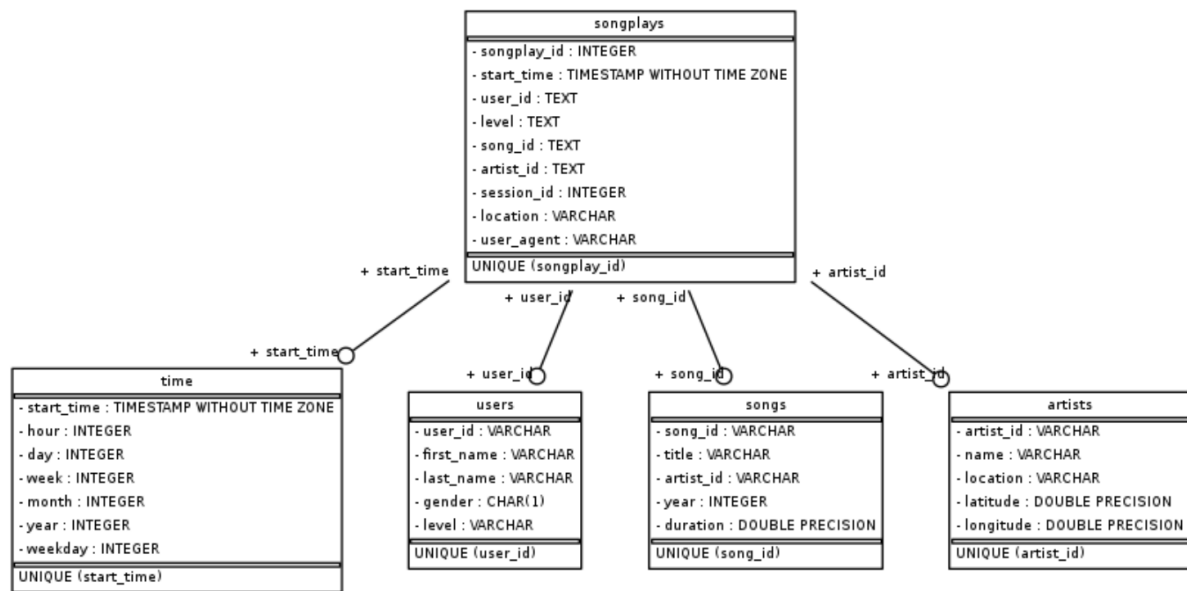


- Create a `README` file with the following information:
 - a summary of the project
 - how to run the Python scripts
 - an explanation of the files in the repository
- `DOCSTRING` statements have been added in each function in etl.py file to describe what each function does.

-  **DOCSTRING:** Docstring comments are detailed and well written.
-  **README :** Good job with the README, it is well organized. Markdown formatting is very well implemented. You've discussed all the relevant points required as per the rubric.

- **OPTIONAL**

To further enhance your project and make it look more professional, you can add an ER diagram which could help someone new quickly understand the implemented database.



You can generate the ER diagram using code as well.

This will require you to install `sqlalchemy` package from `pip` in the workspace

```


from sqlalchemy_schemadisplay import create_schema_graph
from sqlalchemy import MetaData

def main():
    graph = create_schema_graph(metadata=MetaData('postgresql://student:student@127.0.0.1/sparkifydb'))
    graph.write_png('sparkifydb_erd.png')

if __name__ == "__main__":
    main()
  
```



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

- The scripts follow PEP8 guidelines. Each task is further subdivided into simple functions, the variable and function names are intuitive. 
- For more information about PEP8 standards please check [PEP8 documentation](#)

 [DOWNLOAD PROJECT](#)



RETURN TO PATH