Data Modeling with Postgres Project Starter Code

The objective of this project is to apply data modeling with Postgres and build an ETL pipeline using Python.

How to run the program with your own code

For the execution of your own code, we head to the Project Workspace.

Running the create_tables.py script in a terminal creates and initializes the tables for the sparkifydb database. Another file sql_queries.py contains all SQL queries and is imported into create_tables.py.

```
python create_tables.py
```

Running test.ipynb in a Jupyter notebook confirms that the tables were successfully created with the correct columns.

Running etl.ipynb in a Jupyter notebook develops the ETL processes for each table and is used to prepare a python script for processing all the datasets.

Running the etl.py script in a terminal to process the all the datasets. The script connects to the Sparkify database, extracts and processes the log_data and song_data, and loads data into the five tables. The file sql_queries.py contains all SQL queries and is imported into etl.py.

```
python etl.py
```

Again, running test.ipynb confirms that the records were successfully inserted into each table.

The summary of the files and folders within repo is provided in the table below:

File/Folder	Definition
data/*	Folder that contains all the json files with the data used in this project.
images/*	Folder containing the images of the project.
test.ipynb	Displays the first few rows of each table to let check the database.
create_tables.py	Drops and creates the tables. This file is to reset the tables before each time that run the ETL scripts.
etl.ipynb	Reads and processes a single file from song_data and log_data and loads the data into the tables. This notebook contains detailed instructions on the ETL process for each of the tables.
etl.py	Reads and processes files from song_data and log_data and loads them into the tables.
sql_queries.py	Containts the SQL queries to build the tables, to insert rows and a SELECT query to find song_id

File/Folder	Definition
	and artist_id based on the song name, artist name and song length.
README.md	Contains the project documentation.
README.pdf	Contains the project documentation in PDF format.

Steps to complete the project:

1. Define fact and dimension tables for a star schema for a particular analytic focus.
2. Write an ETL pipeline that transfers data from files in two local directories into these tables in Postgres using Python and SQL.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Scenario.

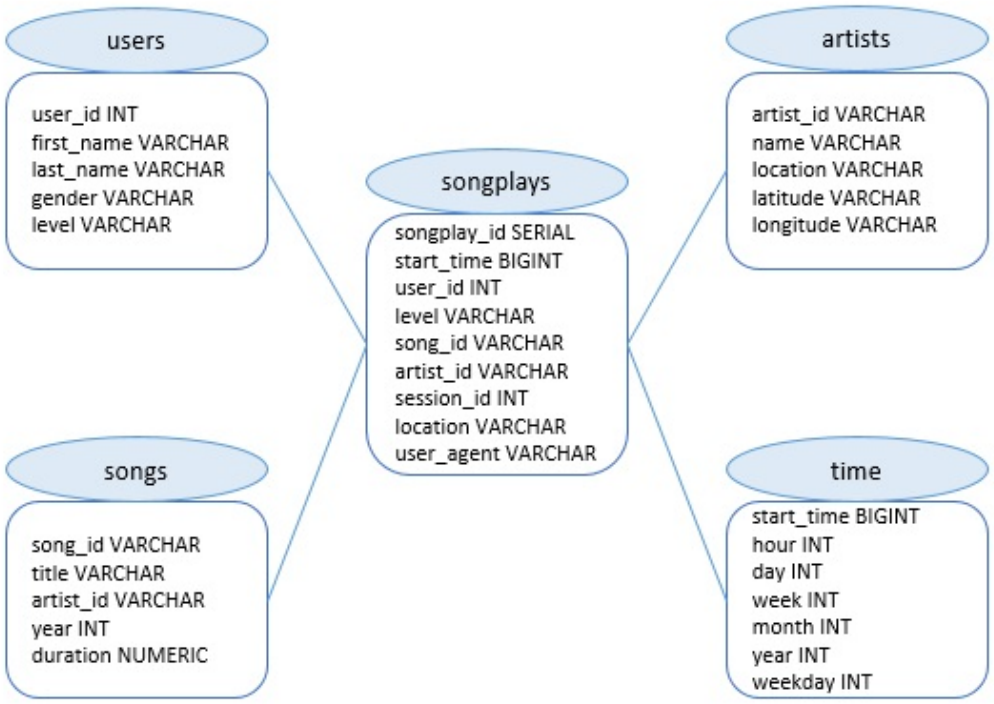
A startup called Sparkify wants to analyze the data they've been collecting on songs and user activity on their new music streaming app. The analytics team is particularly interested in understanding what songs users are listening to. Currently, they don't have an easy way to query their data, which resides in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

They'd like a data engineer to create a Postgres database with tables designed to optimize queries on song play analysis, and bring you on the project. Your role is to create a database schema and ETL pipeline for this analysis. You'll be able to test your database and ETL pipeline by running queries given to you by the analytics team from Sparkify and compare your results with their expected results.

Database Schema for Sparkify.

In our scenario, we have one dataset of songs where each file is in JSON format and contains metadata about a song and the artist of that song. The second dataset consists of log files in JSON format and simulates activity logs from a music streaming app (based on specified configurations).

Using the song and log datasets, the database schema looks like this:



Star Schema.

A star schema is the simplest style of data mart schema. The star schema consists of one or more fact tables referencing to any number of dimension tables. It has some advantages like fast aggregation for analytics, simple queries for JOINs, etc.

Facts Table.

In data warehousing, a fact table consists of measurements, metrics or facts of a business process.

ETL Pipeline

Extract, transform, load (ETL) is the general procedure of copying data from one or more sources into a destination system which represents the data differently from, or in a different context than, the sources.

Data

The data used in this project will be used for upcoming projects also. So, it is better to understand what it represents.

Song Dataset

The first dataset is a subset of real data from the Million Song Dataset. Each file is in JSON format and contains metadata about a song and the artist of that song. The files are partitioned by the first three letters of each song's track ID. For example, here are file paths to three files in this dataset.

```
data/song_data/A/A/A/TRAAAAW128F429D538.json
data/song_data/A/A/B/TRAABCL128F4286650.json
data/song_data/A/A/C/TRAACCG128F92E8A55.json
```

And below is an example of what a single song file, TRAAAAW128F429D538.json, looks like.

```
{"num_songs": 1, "artist_id": "ARD7TVE1187B99BFB1", "artist_latitude
```

Log Dataset

The second dataset consists of log files in JSON format generated by this event simulator based on the songs in the dataset above. These simulate app activity logs from Sparkify app based on specified configurations.

The log files in the dataset we'll be working with are partitioned by year and month. For example, here are file paths to three files in this dataset.

```
data/log_data/2018/11/2018-11-01-events.json
data/log_data/2018/11/2018-11-02-events.json
data/log_data/2018/11/2018-11-03-events.json
```

And below is an example of what the data in a log file, 2018-11-01-events.json, looks like.

```
{"artist":null,"auth":"Logged In","firstName":"Walter","gender":"M",
```

Process song data (song_data directory).

We will perform ETL on the files in song_data directory to create two dimensional tables: songs table and artists table.

This is what a songs file looks like:

```
{"num_songs": 1, "artist_id": "ARD7TVE1187B99BFB1", "artist_latitude
```

For songs table, we'll extract data for songs table by using only the columns corresponding to the songs table suggested in the star schema above. Similarly, we'll select the appropriate columns for artists table.

```
song_data = df.loc[:,["song_id", "title", "artist_id", "year", "du
song_data
# Looks like this
```

```
    # ['SOINLJW12A8C13314C', 'City Slickers', 'AR8IEZO1187B99055E', 2008

    artist_data = df.loc[:,["artist_id", "artist_name", "artist_location
    artist_data

    # Looks like this
    # ['AR8IEZO1187B99055E', 'Marc Shaiman', '', nan, nan]
```

Now insert the extract data into their respective tables.

```
    # insert songs data
    cur.execute(song_table_insert, song_data)
    conn.commit()

    # insert artists data
    cur.execute(artist_table_insert, artist_data)
    conn.commit()
```

Variables song_table_insert and artist_table_insert are SQL queries. These are given in sql_queries.py file.

Process log data (log_data directory).

We will perform ETL on the files in log_data directory to create the remaining two dimensional tables: time and users, as well as the songplays fact table.

This is what a single log file looks like:

```
    {"artist":null,"auth":"Logged In","firstName":"Walter","gender":"M",
    {"artist":null,"auth":"Logged In","firstName":"Kaylee","gender":"F",
    {"artist":"Des'ree","auth":"Logged In","firstName":"Kaylee","gender"
```

For time table we have ts column in log files. We will parse it as a time stamp and use python's datetime functions to create the remaining columns required for the table mentioned in the above schema.

```
        # Filter by NextSong action
        df = df[df.page == "NextSong"]
```

```
    # Convert timestamp column to datetime
    t = pd.to_datetime(df["ts"], unit = "ms")

    # Insert time data records
    time_data = (df["ts"].tolist(), t.dt.hour.values.tolist(), t.dt.
    column_labels = ("timestamp", "hour", "day", "week", "month", "y
    time_df = pd.DataFrame(list(time_data), index = list(column_labe
```

For users table, we'll extract the appropriate columns from log files as mentioned in the star schema above for users table.

```
    # Load user table
    user_df = df.sort_values(by="ts", ascending = False).loc[:,["use
```

For songplays table, we will require information from songs table, artists table and the original log files. Since the log files do not have song_id and artist_id, we need to use songs table and artists table for that. The song_select query finds the song_id and artist_id based on the title, artist_name, and duration of a song. For the remaining columns, we can select them from the log files.

```
    # Insert songplay records
    for index, row in df.iterrows():

        # Get songid and artistid from song and artist tables
        cur.execute(song_select, (row.song, row.artist, row.length))
        results = cur.fetchone()

        if results:
            songid, artistid = results
        else:
            songid, artistid = None, None

        # Insert songplay record
        songplay_data = (df.ts[index].item(), int(df.userId[index]),
        ##cur.execute(songplay_table_insert, songplay_data)
        cur.execute(songplay_table_insert, songplay_data + songplay_
```

Now insert the data into their respective tables.

```
# Insert into time table
for i, row in time_df.iterrows():
    cur.execute(time_table_insert, list(row))

# Insert user records
for i, row in user_df.iterrows():
    cur.execute(user_table_insert, row)
```

Conclusion.

We created a Postgres database with the facts and dimension table for song_play analysis. We populated it with the entries from songs and events directory. Now our data is useful for some basic aggregation and analytics.