# Project: Data Modeling with Postgres

### Introduction

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.

### Project Description

In this project, you'll apply what you've learned on data modeling with Postgres and build an ETL pipeline using Python. To complete the project, you will need to define fact and dimension tables for a star schema for a particular analytic focus, and write an ETL pipeline that transfers data from files in two local directories into these tables in Postgres using Python and SQL.

**Schema for Song Play Analysis**

Using the song and log datasets, you'll need to create a star schema optimized for queries on song play analysis. This includes the following tables.

**Fact Table**

* **songplays** - records in log data associated with song plays i.e. records with page NextSong
  + *songplay\_id, start\_time, user\_id, level, song\_id, artist\_id, session\_id, location, user\_agent*

**Dimension Tables**

* **users** - users in the app
  + *user\_id, first\_name, last\_name, gender, level*
* **songs** - songs in music database
  + *song\_id, title, artist\_id, year, duration*
* **artists** - artists in music database
  + *artist\_id, name, location, latitude, longitude*
* **time** - timestamps of records in **songplays** broken down into specific units
  + *start\_time, hour, day, week, month, year, weekday*

**Project Template**

To get started with the project, go to the workspace on the next page, where you'll find the project template files. You can work on your project and submit your work through this workspace. Alternatively, you can download the project template files from the Resources folder if you'd like to develop your project locally.

In addition to the data files, the project workspace includes six files:

1. test.ipynb displays the first few rows of each table to let you check your database.
2. create\_tables.py drops and creates your tables. You run this file to reset your tables before each time you run your ETL scripts.
3. etl.ipynb reads and processes a single file from song\_data and log\_data and loads the data into your tables. This notebook contains detailed instructions on the ETL process for each of the tables.
4. etl.py reads and processes files from song\_data and log\_data and loads them into your tables. You can fill this out based on your work in the ETL notebook.
5. sql\_queries.py contains all your sql queries, and is imported into the last three files above.
6. README.md provides discussion on your project.

**Project Steps**

Below are steps you can follow to complete the project:

**Create Tables**

1. Write CREATE statements in sql\_queries.py to create each table.
2. Write DROP statements in sql\_queries.py to drop each table if it exists.
3. Run create\_tables.py to create your database and tables.
4. Run test.ipynb to confirm the creation of your tables with the correct columns. Make sure to click "Restart kernel" to close the connection to the database after running this notebook.

**Build ETL Processes**

Follow instructions in the etl.ipynb notebook to develop ETL processes for each table. At the end of each table section, or at the end of the notebook, run test.ipynb to confirm that records were successfully inserted into each table. Remember to rerun create\_tables.py to reset your tables before each time you run this notebook.

**Build ETL Pipeline**

Use what you've completed in etl.ipynb to complete etl.py, where you'll process the entire datasets. Remember to run create\_tables.py before running etl.py to reset your tables. Run test.ipynb to confirm your records were successfully inserted into each table.

**Document Process**

Do the following steps in your README.md file.

* Discuss the purpose of this database in the context of the startup, Sparkify, and their analytical goals.

This project implements data analytics solution for Sparkify using the data generated by the user’s interaction with the application.

This solution will provide the capability to analyze this data from several perspectives like usage and content trends. Therefore will help to improve the application and to provides better content to end users.

* State and justify your database schema design and ETL pipeline.

The solution is based on a star schema, which is database model specifical for analytical purposes. It is based on the next Fact and dimension tables implemented in PostgreSQL:

* + Fact Table
    - songplays - records in log data associated with song plays i.e. records with page NextSong
  + Dimension Tables
    - users - users in the app
    - songs - songs in music database
    - artists - artists in music database
    - time - timestamps of records in songplays broken down into specific units

Also has been implemented a ETL pipeline that transfers data from files in two local directories into the database.

* Provide example queries and results for song play analysis.

With this solution has been provided several analytical reports like

* + top 10 artists and songs

%sql SELECT T3.NAME AS ARTIST, T2.TITLE AS SONG, COUNT(T1.SONG\_ID) AS TOTAL \

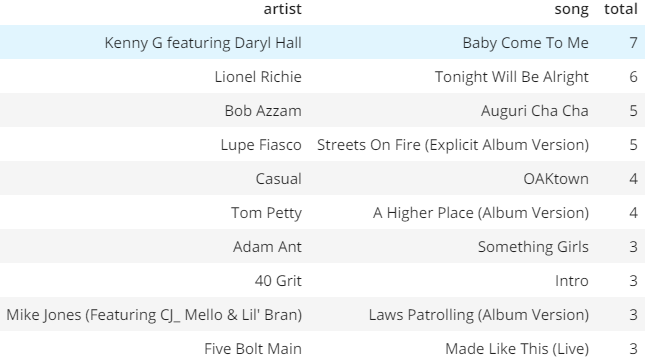
FROM SONGPLAYS T1 JOIN SONGS T2 ON T1.SONG\_ID = T2.SONG\_ID \

JOIN ARTISTS AS T3 ON T2.ARTIST\_ID = T3.ARTIST\_ID \

GROUP BY T3.NAME, T2.TITLE \

ORDER BY TOTAL DESC \

LIMIT 10



* + the total plays by week

# GET TOTAL PLAYS BY WEEK

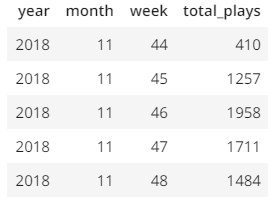
%sql SELECT T2.YEAR, T2.MONTH, T2.WEEK, COUNT(T1.SONGPLAY\_ID) AS TOTAL\_PLAYS \

FROM SONGPLAYS T1 \

JOIN TIME T2 ON T1.START\_TIME = T2.START\_TIME \

GROUP BY T2.YEAR, T2.MONTH, T2.WEEK \

ORDER BY T2.YEAR, T2.MONTH, T2.WEEK

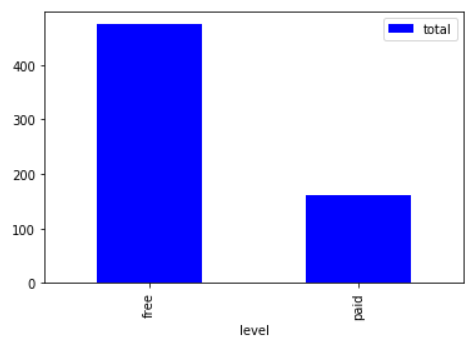


* + total users by level.

%sql SELECT T1.LEVEL, COUNT(T1.USER\_ID) AS TOTAL \

FROM USERS T1 \

GROUP BY T1.LEVEL





Here's a [guide](https://www.markdownguide.org/basic-syntax/) on Markdown Syntax.

**NOTE: You will not be able to run test.ipynb, etl.ipynb, or etl.py until you have run create\_tables.py at least once to create the sparkifydb database, which these other files connect to.**

# Song Dataset

The first dataset is a subset of real data from the [Million Song Dataset](https://labrosa.ee.columbia.edu/millionsong/). 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 filepaths to two files in this dataset.

song\_data/A/B/C/TRABCEI128F424C983.json

song\_data/A/A/B/TRAABJL12903CDCF1A.json

And below is an example of what a single song file, TRAABJL12903CDCF1A.json, looks like.

{"num\_songs": 1, "artist\_id": "ARJIE2Y1187B994AB7", "artist\_latitude": null, "artist\_longitude": null, "artist\_location": "", "artist\_name": "Line Renaud", "song\_id": "SOUPIRU12A6D4FA1E1", "title": "Der Kleine Dompfaff", "duration": 152.92036, "year": 0}

# Log Dataset

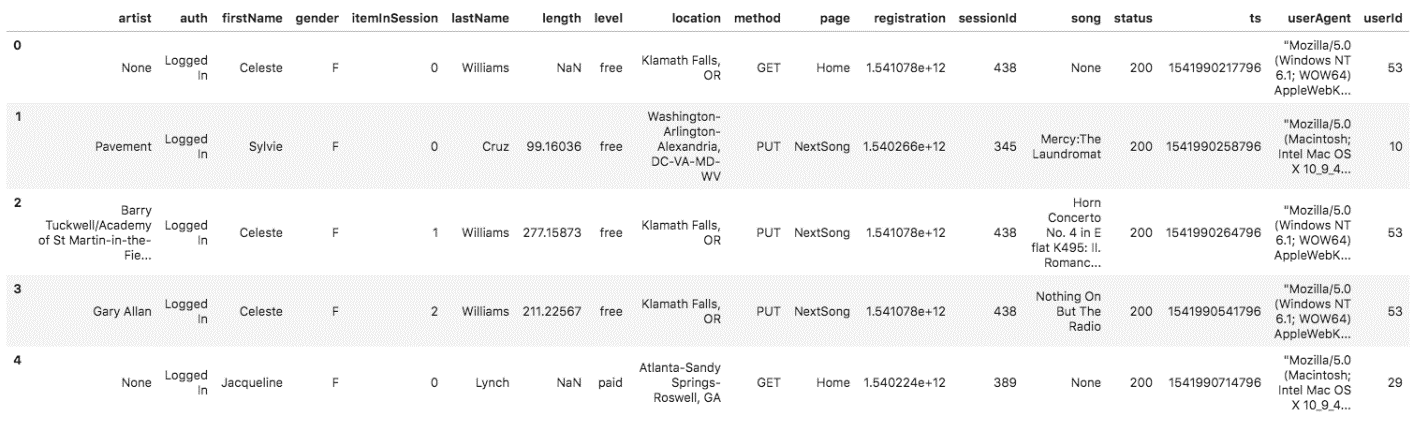
The second dataset consists of log files in JSON format generated by this [event simulator](https://github.com/Interana/eventsim) based on the songs in the dataset above. These simulate activity logs from a music streaming app based on specified configurations.

The log files in the dataset you'll be working with are partitioned by year and month. For example, here are filepaths to two files in this dataset.

log\_data/2018/11/2018-11-12-events.json

log\_data/2018/11/2018-11-13-events.json

And below is an example of what the data in a log file, 2018-11-12-events.json, looks like.



If you would like to look at the JSON data within log\_data files, you will need to create a pandas dataframe to read the data. Remember to first import JSON and pandas libraries.

df = pd.read\_json(filepath, lines=True)

For example, df = pd.read\_json('data/log\_data/2018/11/2018-11-01-events.json', lines=True) would read the data file 2018-11-01-events.json.

In case you need a refresher on JSON file formats, [here is a helpful video](https://www.youtube.com/watch?time_continue=1&v=hO2CayzZBoA).