# Project: Data Lake

### Introduction

A music streaming startup, Sparkify, has grown their user base and song database even more and want to move their data warehouse to a data lake. Their data resides in S3, 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.

As their data engineer, you are tasked with building an ETL pipeline that extracts their data from S3, processes them using Spark, and loads the data back into S3 as a set of dimensional tables. This will allow their analytics team to continue finding insights in what songs their users are listening to.

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 Spark and data lakes to build an ETL pipeline for a data lake hosted on S3. To complete the project, you will need to load data from S3, process the data into analytics tables using Spark, and load them back into S3. You'll deploy this Spark process on a cluster using AWS.

**Project Datasets**

You'll be working with two datasets that reside in S3. Here are the S3 links for each:

* Song data: s3://udacity-dend/song\_data
* Log data: s3://udacity-dend/log\_data

**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**

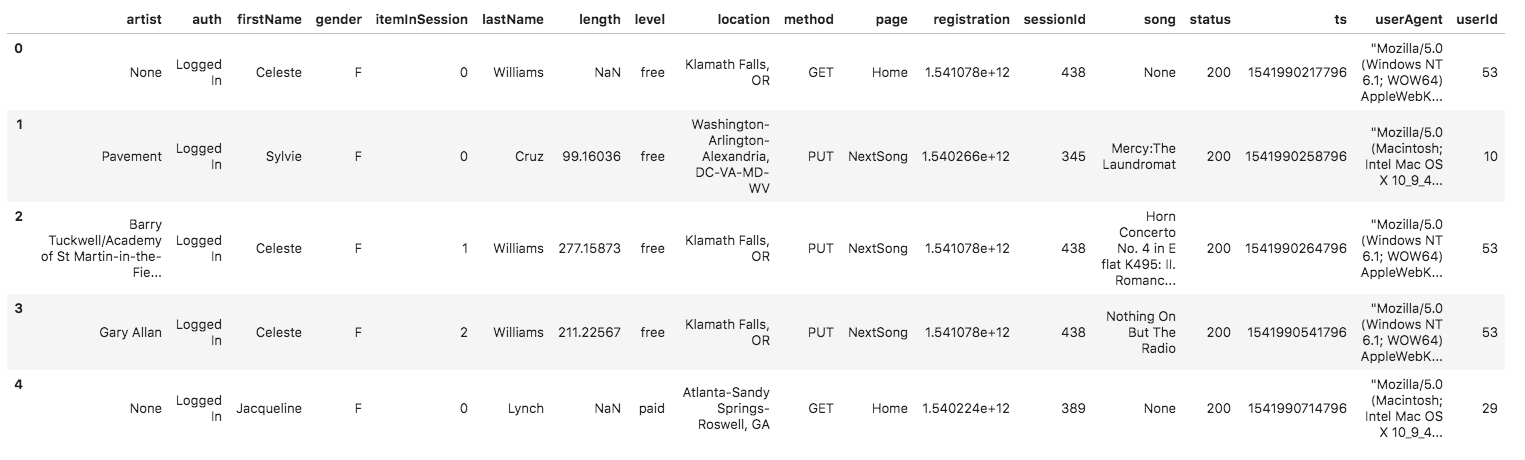
The second dataset consists of log files in JSON format generated by this [**event simulator**](https://github.com/Interana/eventsim) based on the songs in the dataset above. These simulate app activity logs from an imaginary music streaming app based on configuration settings.

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.

**[[](https://classroom.udacity.com/nanodegrees/nd027/parts/19ef4e55-151f-4510-8b5c-cb590ac52df2/modules/743a0ff8-d0ad-4a63-a459-46601f8a4446/lessons/cfdfde06-9c87-4c72-aa09-2aee8eb35675/concepts/2a3cf217-276f-4826-b6c1-6747ca73533b)](https://classroom.udacity.com/nanodegrees/nd027/parts/19ef4e55-151f-4510-8b5c-cb590ac52df2/modules/743a0ff8-d0ad-4a63-a459-46601f8a4446/lessons/cfdfde06-9c87-4c72-aa09-2aee8eb35675/concepts/2a3cf217-276f-4826-b6c1-6747ca73533b)**

NEXT

# 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

1. **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

1. **users** - users in the app
   * user\_id, first\_name, last\_name, gender, level
2. **songs** - songs in music database
   * song\_id, title, artist\_id, year, duration
3. **artists** - artists in music database
   * artist\_id, name, location, lattitude, longitude
4. **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. You can work on your project with a smaller dataset found in the workspace, and then move on to the bigger dataset on AWS.

Alternatively, you can download the template files in the Resources tab in the classroom and work on this project on your local computer.

The project template includes three files:

* etl.py reads data from S3, processes that data using Spark, and writes them back to S3
* dl.cfgcontains your AWS credentials
* README.md provides discussion on your process and decisions

### Document Process

Do the following steps in your README.md file.

1. Discuss the purpose of this database in context of the startup, Sparkify, and their analytical goals.
2. State and justify your database schema design and ETL pipeline.
3. [Optional] Provide example queries and results for song play analysis.

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

### Project Rubric

Read the project [**rubric**](https://review.udacity.com/#!/rubrics/2502/view) before and during development of your project to ensure you meet all specifications.

## PROJECT SPECIFICATION

**Data Lake**

ETL

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The etl.py script runs without errors. | The script, etl.py, runs in the terminal without errors. The script reads song\_data and load\_data from S3, transforms them to create five different tables, and writes them to partitioned parquet files in table directories on S3. |
| Analytics tables are correctly organized on S3. | Each of the five tables are written to parquet files in a separate analytics directory on S3. Each table has its own folder within the directory. Songs table files are partitioned by year and then artist. Time table files are partitioned by year and month. Songplays table files are partitioned by year and month. |
| The correct data is included in all tables. | Each table includes the right columns and data types. Duplicates are addressed where appropriate. |

Code Quality

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The project shows proper use of documentation. | 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. |
| The project code is clean and modular. | Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines. |