# Project: Data Warehouse

### Introduction

A music streaming startup, Sparkify, has grown their user base and song database and want to move their processes and data onto the cloud. 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, stages them in Redshift, and transforms data into a set of dimensional tables for 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 data warehouses and AWS to build an ETL pipeline for a database hosted on Redshift. To complete the project, you will need to load data from S3 to staging tables on Redshift and execute SQL statements that create the analytics tables from these staging tables.

**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

Log data json path: s3://udacity-dend/log\_json\_path.json

**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**

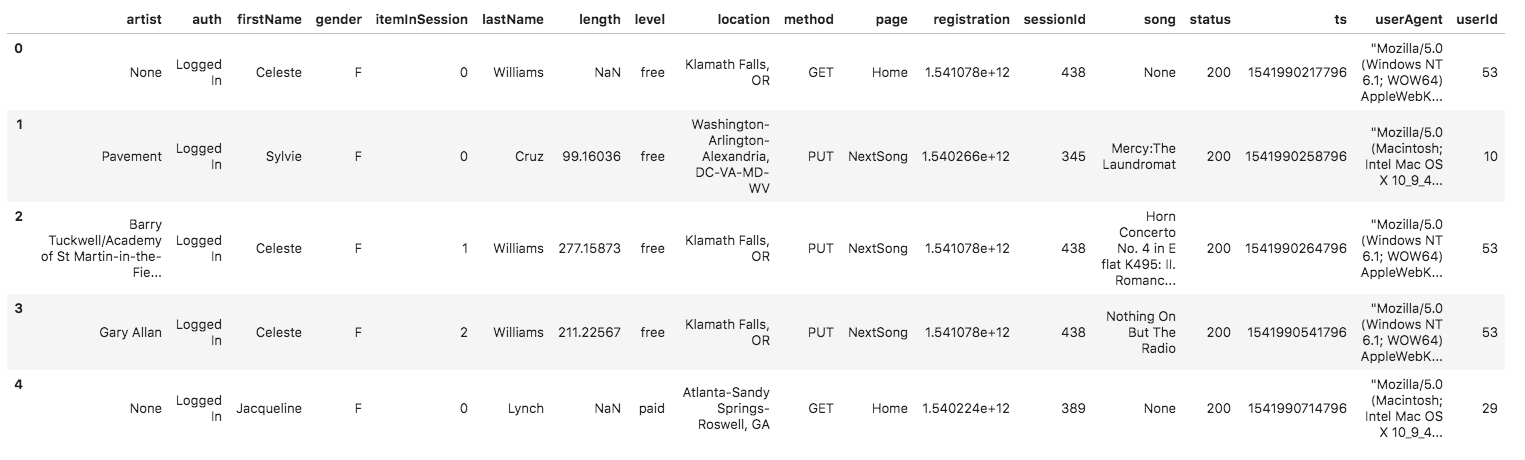
The second dataset consists of log files in JSON format generated by this [**event simulator**](https://github.com/Interana/eventsim) based on the songs in the dataset above. These simulate app activity logs from an imaginary music streaming app based on configuration settings.

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/69a25b76-3ebd-4b72-b7cb-03d82da12844/modules/58ff61b9-a54f-496d-b4c7-fa22750f6c76/lessons/b3ce1791-9545-4187-b1fc-1e29cc81f2b0/concepts/fa049d13-5e15-4333-b909-f1f6f0ce36a5)](https://classroom.udacity.com/nanodegrees/nd027/parts/69a25b76-3ebd-4b72-b7cb-03d82da12844/modules/58ff61b9-a54f-496d-b4c7-fa22750f6c76/lessons/b3ce1791-9545-4187-b1fc-1e29cc81f2b0/concepts/fa049d13-5e15-4333-b909-f1f6f0ce36a5)**

NEXT

# Schema for Song Play Analysis

Using the song and event 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 event 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 and submit your work through this workspace.

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 four files:

* create\_table.py is where you'll create your fact and dimension tables for the star schema in Redshift.
* etl.py is where you'll load data from S3 into staging tables on Redshift and then process that data into your analytics tables on Redshift.
* sql\_queries.py is where you'll define you SQL statements, which will be imported into the two other files above.
* README.md is where you'll provide discussion on your process and decisions for this ETL pipeline.

# Project Steps

Below are steps you can follow to complete each component of this project.

### Create Table Schemas

1. Design schemas for your fact and dimension tables
2. Write a SQL CREATE statement for each of these tables in sql\_queries.py
3. Complete the logic in create\_tables.py to connect to the database and create these tables
4. Write SQL DROP statements to drop tables in the beginning of create\_tables.py if the tables already exist. This way, you can run create\_tables.py whenever you want to reset your database and test your ETL pipeline.
5. Launch a redshift cluster and create an IAM role that has read access to S3.
6. Add redshift database and IAM role info to dwh.cfg.
7. Test by running create\_tables.py and checking the table schemas in your redshift database. You can use Query Editor in the AWS Redshift console for this.

### Build ETL Pipeline

1. Implement the logic in etl.py to load data from S3 to staging tables on Redshift.
2. Implement the logic in etl.py to load data from staging tables to analytics tables on Redshift.
3. Test by running etl.py after running create\_tables.py and running the analytic queries on your Redshift database to compare your results with the expected results.
4. Delete your redshift cluster when finished.

### 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.

### Note

The SERIAL command in Postgres is not supported in Redshift. The equivalent in redshift is IDENTITY(0,1), which you can read more on in the [**Redshift Create Table Docs**](https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_TABLE_NEW.html).

### Project Rubric

Read the project [**rubric**](https://review.udacity.com/#!/rubrics/2501/view) before and during development of your project to ensure you meet all specifications.

# REMINDER: Do not include your AWS access keys in your code when sharing this project!

## PROJECT SPECIFICATION

**Data Warehouse**

Table Creation

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| Table creation script runs without errors. | 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. |
| Staging tables are properly defined. | CREATE statements in sql\_queries.py specify all columns for both the songs and logs staging tables with the right data types and conditions. |
| Fact and dimensional tables for a star schema are properly defined. | CREATE statements in sql\_queries.py specify all columns for each of the five tables with the right data types and conditions. |

ETL

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| ETL script runs without errors. | The script, etl.py, runs in the terminal without errors. The script connects to the Sparkify redshift database, loads log\_data and song\_data into staging tables, and transforms them into the five tables. |
| ETL script properly processes transformations in Python. | INSERT statements are correctly written for each table and handles duplicate records where appropriate. Both staging tables are used to insert data into the songplays table. |

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. |

### Suggestions to Make Your Project Stand Out!

* Add data quality checks
* Create a dashboard for analytic queries on your new database