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

2. **users** - users in the app
    - *user_id, first_name, last_name, gender, level*
3. **songs** - songs in music database
    - *song_id, title, artist_id, year, duration*
4. **artists** - artists in music database
    - *artist_id, name, location, lattitude, longitude*
5. **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** 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**.

Project Rubric

Read the project **rubric** 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!