UDACITY

‹ Return to "Data Engineering Nanodegree" in the classroom

DISCUSS ON STUDENT HUB

Data Warehouse

审阅

代码审阅 6

HISTORY

Requires Changes

还需满足 3 个要求 变化

Brilliant Udacity Learner,

Just some minor changes 💪
The structure of this project is already impressive and you've done a great job with most of the specification. I could appreciate the time and effort put into this submission especially on loading the data from S3 to staging tables successfully on Redshift and using appropriate data types and conditions for each tables. 👏

Feedback has been provided below to guide you on how to pass all the required specifications. Keep up the good work and I look forward to your next submission.

Extra Materials

Below are some additional links to help you deepen your understanding on the related concepts:

- Top 8 Best Practices for High-Performance ETL Processing Using Amazon Redshift
- How a data warehouse was builit using Amazon Redshift
- 3 Ways to do Redshift ETL
- Data Warehousing on AWS
- 2X Your Redshift Speed With Sortkeys and Distkeys

Table Creation

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.

Well done! The script runs and connects to the Sparkify database successfully. Drop and Create tables has been implemented correctly. 👏

CREATE statements in sql_queries.py specify all columns for both the songs and logs staging tables with the right data types and conditions.

Nice work! Appropriate data types and conditions has been used on your staging tables.

Suggestions

- `artist_latitude`, `artist_longitude` and `duration` in the **staging_songs** could be set to `float` datatype to be appropriate.
- Please check this resource to know more about SQL Data Warehouse supported data types.
- SQL data types divided into categories.
- Here a nice discussion about varchar vs int for storing a variable

CREATE statements in sql_queries.py specify all columns for each of the five tables with the right data types and conditions.

Nice work, you have identified the correct PRIMARY KEYS and data types for most of each table, however, you need to specify NOT NULL constraints where appropriate:

For example:
• **Song-plays**: `songplay_id` should be set as the PRIMARY KEY and there should be NOT NULL constraint for `start_time`, `user_id`, because they are foreign keys in this table.
• **Song table**: there should be NOT NULL constraint for `artist_id`.

Suggestions

- By default, the PRIMARY KEY constraint has the unique and not null constraint built into it, no need to specify it.
- Data Warehouse Design Techniques – Constraints and Indexes
- Here is a very nice explanation about SQL Constraints
- SQL Keys and Constraints
- Different types of constraints in SQL

ETL

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.

This will be evaluated after the changes in CREATE and INSERT statements.

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.

JOIN clause has been used to insert data into the songplays table from both staging tables. However, you also need to consider how to get rid of duplicate entries here.

Requires Changes

- Please use DISTINCT Clause in the SELECT statement to handle duplicate records.
- For the `users` table, please INSERT only rows where the page is `NextSong` as well.
- In your time table, you should select from the `songplays` table and not from `staging_events`.

Extra resources

- Basic SQL Join Types
- Different types of JOINs
- When to use SQL JOINS
- SQL SELECT DISTINCT Statement
- SQL DISTINCT examples

Code Quality

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.

Impressive README file! It contains all the necessary details for a writeup and docstrings were effectively used to explain each functions. 👏

Though there are no standard and rules for doing so, please note that it is an essential part that documenting your code is going to serve well enough for writing clean code and well-written programs.

Suggestions

To make your writeup even better, you may include screenshots of your final tables, and add more details about your project, such as the dataset and how did you clean the data. Any in-line comments that were clearly part of the project instructions should be removed, and parameter descriptions and data types can be also included in your docstrings.

You may check some links below to know more about python code documentation:

- Documenting Python Code: A Complete Guide
- PEP 257 -- Docstring Conventions

- Python Docstrings
- Different types of writing Docstrings

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

Most of the code is well optimized with intuitive and easy-to-follow structure which follows PEP8 style guidelines but please limit all lines to a maximum of 79 characters. The Python standard library is conservative and requires limiting lines to 79 characters (and docstrings/comments to 72). You may use backslash "\" for line continuation.

Suggestions

- You may find this link helpful to Check your code for PEP8 requirements.
- Here are some additional resource to know more about PEP8 style guidelines:
    - PEP 8: Style Guide for Python Code
    - How to Write Beautiful Python Code With PEP 8
    - PEP-8 Tutorial: Code Standards in Python

重新提交

下载项目

6 代码审阅评注 ›