

# Meets Specifications

Brilliant Udacity Learner,

## Table

Congratulations! 🎉

You've successfully passed all the specifications in ONE SHOT! I must admit that the structure of this project implementation is impressive! You should be proud of the work done as you seem to have a good hold on **Data Warehouses** and **AWS** to build an **ETL pipeline** for a database hosted on **Redshift**.

Please run all code from jupyter notebook Project.ipynb this will ensure that all AWS services used is closed properly.

Thank you for the notes and I appreciate associating create Redshift Cluster using the AWS python SDK. 👍 It has been a pleasure to review this wonderful project. Please continue with this same spirit of hard work in the projects ahead. 🙌

## 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 built using Amazon Redshift](#)
- [3 Ways to do Redshift ETL](#)
- [Data Warehousing on AWS](#)
- [2X Your Redshift Speed With Sortkeys and Distkeys](#)

## 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. 🙌

### Suggestions

- Here is a nice [discussion](#) about Amazon Redshift : drop table if exists.
- A good [documentation](#) for Amazon Redshift DROP TABLE.

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 have been used on your staging tables.

### Suggestions

- 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 use of SORTKEY and DISTKEY. Create statements in `sql_queries.py` correctly specified all columns with appropriate data types and conditions. PRIMARY keys and NOT NULL constraints are also well specified. Keep it up!

ETL



## Suggestions

- By default, the PRIMARY KEY constraint has the unique and not null constraint built into it, no need to specify it.
- You can also define a foreign key constraint to specify that the values in a column (or a group of columns) must match the values appearing in some row of another table. We say this maintains the referential integrity between two related tables using `REFERENCES`.
- DISTKEY controls how data is distributed across the nodes in the Redshift cluster, and SORTKEY is a form of indexing specific columns.
  - [Choose the best sort key](#)
  - [Choose the best distribution style](#)
- Data Warehouse [Design Techniques](#)
- SQL [Keys and Constraints](#)
- Different [types of constraints](#) in SQL

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.

Splendid! The script successfully runs, copying data into your staging tables and inserting data into your final tables in the redshift database.

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.

Excellent work with your INSERT statements and utilizing `DISTINCT` appropriately to remove duplicate entries. JOIN clause has been used correctly to insert data into the songplays table from both staging tables. 🙌

Code  
Quality

## Extra resources

- [Different types of JOINS](#)
- [When to use SQL JOINS](#)
- [SQL SELECT DISTINCT Statement](#)
- [SQL DISTINCT examples](#)

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.

Nice README file! It contains necessary details for a writeup, however, I suggest using `docstrings` to effectively explain each function. Though there are no standards 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.

Here is an example of a function with [Multi-line Docstrings](#):

```
def my_function(arg1):  
    """  
    Summary line.  
  
    Extended description of function.
```

Parameters:

arg1 (int): Description of arg1

Returns:

int: Description of return value

"""

return arg1

## 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](#)

You might be interested to know more about READMEs:

- [About READMEs](#)
- [How do you put Images on the README.md file](#)
- [Make a README](#)

**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 an 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 "\n" 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](#)