main

Data_Warehouse_Project / README.md

Abhaycl Add files via upload

History

1 contributor

Raw | Blame

```
210 lines (137 sloc)   11.2 KB
```

Data Warehouse Project Starter Code

The objective of this project is to apply what we've learned on data warehouses and AWS to build an ETL pipeline for a database hosted on Redshift.

How to run the program with our own code

Project Requirements

The requirements for the project are a valid aws account, with accompanying security credentials, as well as a python environment, which satisfies the module requirements given.

You will need to add aws access key and secret information to the dwf.cfg file, under [AWS_ACCESS]. This is not to be comitted to git.

The parameterization of the dwh.cfg file is shown below.

```
[CLUSTER]
HOST=This parameter will be defined according to the configuration set up
```

```
DB_NAME=This parameter will be defined according to the configuration set up
DB_USER=This parameter will be defined according to the configuration set up
DB_PASSWORD=This parameter will be defined according to the configuration set up
DB_PORT=5439
CLUSTER_IDENTIFIER=This parameter will be defined according to the configuration
set up
NODE_TYPE=ds2.xlarge
NODE_COUNT=2

[AWS_ACCESS]
AWS_ACCESS_KEY_ID=This parameter will be defined according to the configuration
set up
AWS_SECRET_ACCESS_KEY=This parameter will be defined according to the
configuration set up
AWS_REGION=us-west-2

[IAM_ROLE]
NAME=dwhRole
POLICY_NAME=AmazonS3ReadOnlyAccess
ARN=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
REDSHIFT_ARN=This parameter will be defined according to the configuration set up

[S3]
LOG_DATA=s3://udacity-dend/log_data
LOG_JSONPATH=s3://udacity-dend/log_json_path.json
SONG_DATA=s3://udacity-dend/song_data
```

We also have to create our security group which has to be assigned to the default VPC. Because the creation of our cluster has to belong to a VPC.

NOTE: *To follow IAC (Infrastructure as Code) practices, and to allow us to easily start and stop the redshift cluster to save costs, we can use the following scripts;*

```
redshift_start.py
redshift_stop.py
```

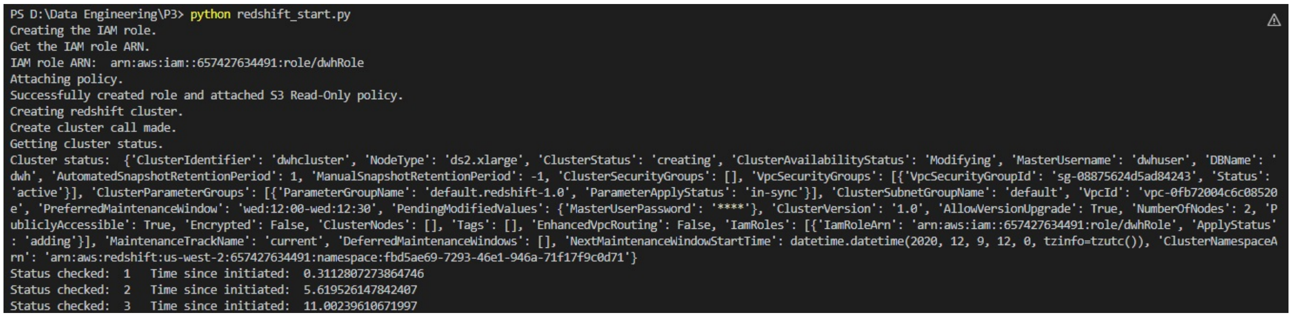
The scripts will create/remove the neccessary resources for redshift to run.

For the execution of our own code, we go to the project workspace or to our own terminal in Visual Code.

In the project workspace we can open a terminal and run the following files:

Start the redshift cluster.

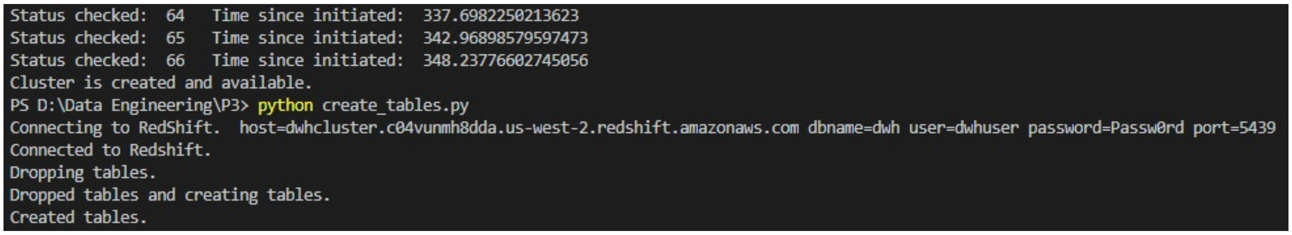
```
  python redshift_start.py
```



```
PS D:\Data Engineering\P3> python redshift_start.py
Creating the IAM role.
Get the IAM role ARN.
IAM role ARN:  arn:aws:iam::657427634491:role/dwhRole
Attaching policy.
Successfully created role and attached S3 Read-Only policy.
Creating redshift cluster.
Create cluster call made.
Getting cluster status.
Cluster status:  {'ClusterIdentifier': 'dwhcluster', 'NodeType': 'ds2.xlarge', 'ClusterStatus': 'creating', 'ClusterAvailabilityStatus': 'Modifying', 'MasterUsername': 'dwhuser', 'DBName': '
dwh', 'AutomatedSnapshotRetentionPeriod': 1, 'ManualSnapshotRetentionPeriod': -1, 'ClusterSecurityGroups': [], 'VpcSecurityGroups': [{'VpcSecurityGroupId': 'sg-08875624d5ad84243', 'Status':
'active'}], 'ClusterParameterGroups': [{'ParameterGroupName': 'default.redshift-1.0', 'ParameterApplyStatus': 'in-sync'}], 'ClusterSubnetGroupName': 'default', 'VpcId': 'vpc-0fb72004c6c08520
e', 'PreferredMaintenanceWindow': 'wed:12:00-wed:12:30', 'PendingModifiedValues': {'MasterUserPassword': '****'}, 'ClusterVersion': '1.0', 'AllowVersionUpgrade': True, 'NumberOfNodes': 2, 'P
ubliclyAccessible': True, 'Encrypted': False, 'ClusterNodes': [], 'Tags': [], 'EnhancedVpcRouting': False, 'IamRoles': [{'IamRoleArn': 'arn:aws:iam::657427634491:role/dwhRole', 'ApplyStatus'
: 'adding'}], 'MaintenanceTrackName': 'current', 'DeferredMaintenanceWindows': [], 'NextMaintenanceWindowStartTime': datetime.datetime(2020, 12, 9, 12, 0, tzinfo=tzutc()), 'ClusterNamespaceA
rn': 'arn:aws:redshift:us-west-2:657427634491:namespace:fbd5ae69-7293-46e1-946a-71f17f9c0d71'}
Status checked:  1   Time since initiated:  0.3112807273864746
Status checked:  2   Time since initiated:  5.619526147842407
Status checked:  3   Time since initiated:  11.00239610671997
```

Create the tables.

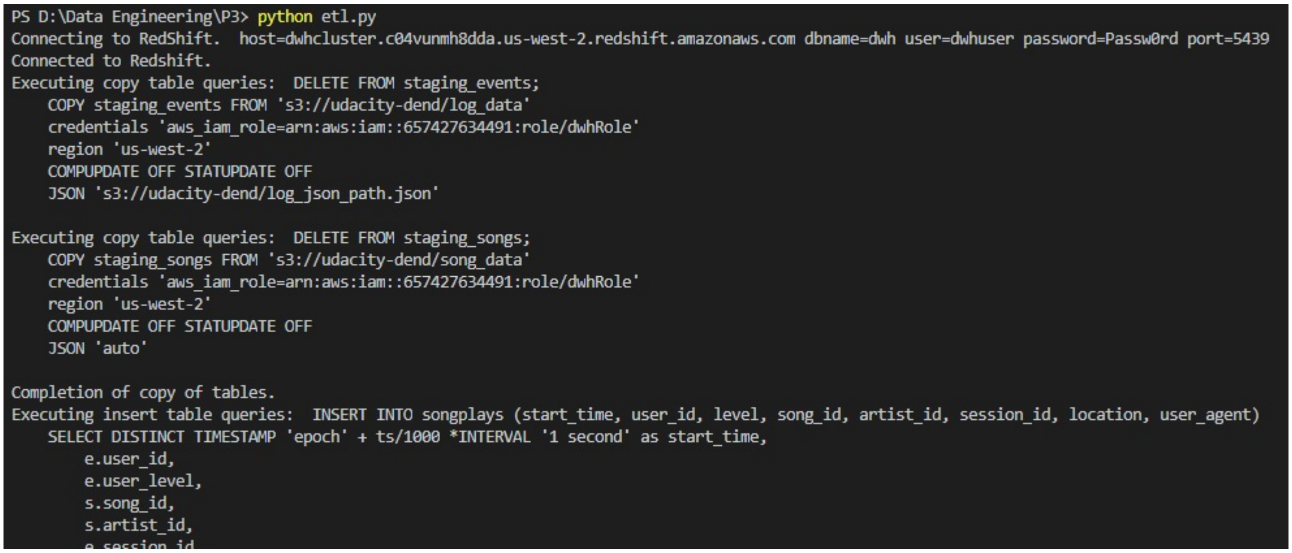
```
python create_tables.py
```



```
Status checked:  64   Time since initiated:  337.6982250213623
Status checked:  65   Time since initiated:  342.96898579597473
Status checked:  66   Time since initiated:  348.23776602745056
Cluster is created and available.
PS D:\Data Engineering\P3> python create_tables.py
Connecting to RedShift.  host=dwhcluster.c04vunmh8dda.us-west-2.redshift.amazonaws.com dbname=dwh user=dwhuser password=Passw0rd port=5439
Connected to Redshift.
Dropping tables.
Dropped tables and creating tables.
Created tables.
```

Data transformations.

```
python etl.py
```



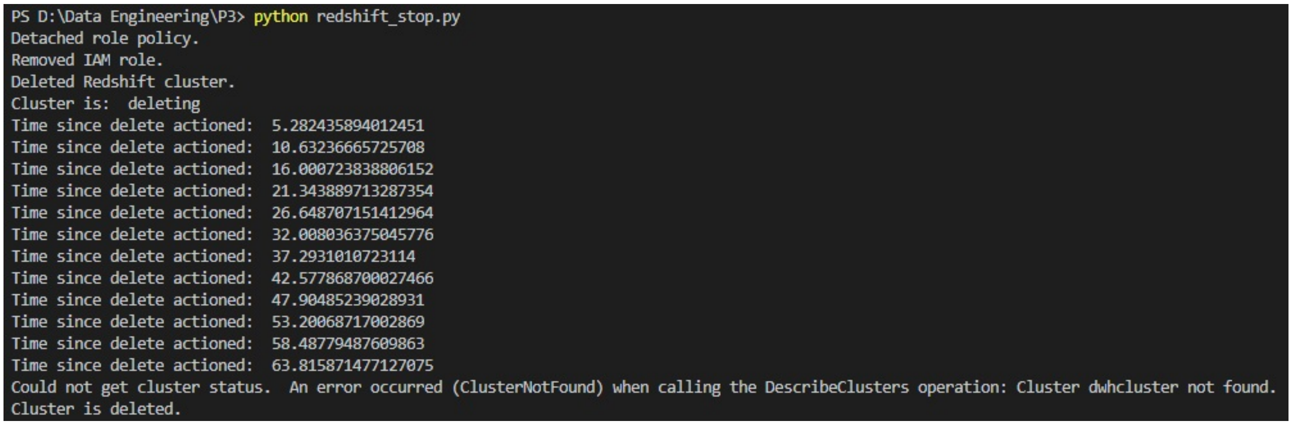
```
PS D:\Data Engineering\P3> python etl.py
Connecting to RedShift.  host=dwhcluster.c04vunmh8dda.us-west-2.redshift.amazonaws.com dbname=dwh user=dwhuser password=Passw0rd port=5439
Connected to Redshift.
Executing copy table queries:  DELETE FROM staging_events;
    COPY staging_events FROM 's3://udacity-dend/log_data'
    credentials 'aws_iam_role=arn:aws:iam::657427634491:role/dwhRole'
    region 'us-west-2'
    COMPUPDATE OFF STATUPDATE OFF
    JSON 's3://udacity-dend/log_json_path.json'

Executing copy table queries:  DELETE FROM staging_songs;
    COPY staging_songs FROM 's3://udacity-dend/song_data'
    credentials 'aws_iam_role=arn:aws:iam::657427634491:role/dwhRole'
    region 'us-west-2'
    COMPUPDATE OFF STATUPDATE OFF
    JSON 'auto'

Completion of copy of tables.
Executing insert table queries:  INSERT INTO songplays (start_time, user_id, level, song_id, artist_id, session_id, location, user_agent)
    SELECT DISTINCT TIMESTAMP 'epoch' + ts/1000 *INTERVAL '1 second' as start_time,
        e.user_id,
        e.user_level,
        s.song_id,
        s.artist_id,
        e.session_id
```

Stop the redshift cluster.

```
python redshift_stop.py
```



```
PS D:\Data Engineering\P3> python redshift_stop.py
Detached role policy.
Removed IAM role.
Deleted Redshift cluster.
Cluster is:  deleting
Time since delete actioned:  5.282435894012451
Time since delete actioned:  10.63236665725708
Time since delete actioned:  16.000723838806152
Time since delete actioned:  21.343889713287354
Time since delete actioned:  26.648707151412964
Time since delete actioned:  32.008036375045776
Time since delete actioned:  37.2931010723114
Time since delete actioned:  42.577868700027466
Time since delete actioned:  47.90485239028931
Time since delete actioned:  53.20068717002869
Time since delete actioned:  58.48779487609863
Time since delete actioned:  63.815871477127075
Could not get cluster status.  An error occurred (ClusterNotFound) when calling the DescribeClusters operation: Cluster dwhcluster not found.
Cluster is deleted.
```

The summary of the files and folders within repo is provided in the table below:

File/Folder	Definition
images/*	Folder containing the images of the project.
auxiliary.py	Auxiliary functions for creating the connection to the RedShift cluster.
create_tables.py	Functions to create the fact and dimension tables for the star schema in Redshift.
dwh.cfg	Contains the parameterization offdhcvv z Redshift, IAM role, AWS access and S3.
etl.py	Contains the queries for loading the S3 data into staging tables on Redshift and then process that data into our analytics tables on Redshift.
redshift_start.py	Contains the necessary processes to create and start up our cluster.
redshift_stop.py	Contains the necessary processes to stop and delete our cluster.
sql_queries.py	Contains the SQL statements that will be implemented for the creation of the tables and for the etl process.
README.md	Contains the project documentation.
README.pdf	Contains the project documentation in PDF format.

Steps to complete the 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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Scenario

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.

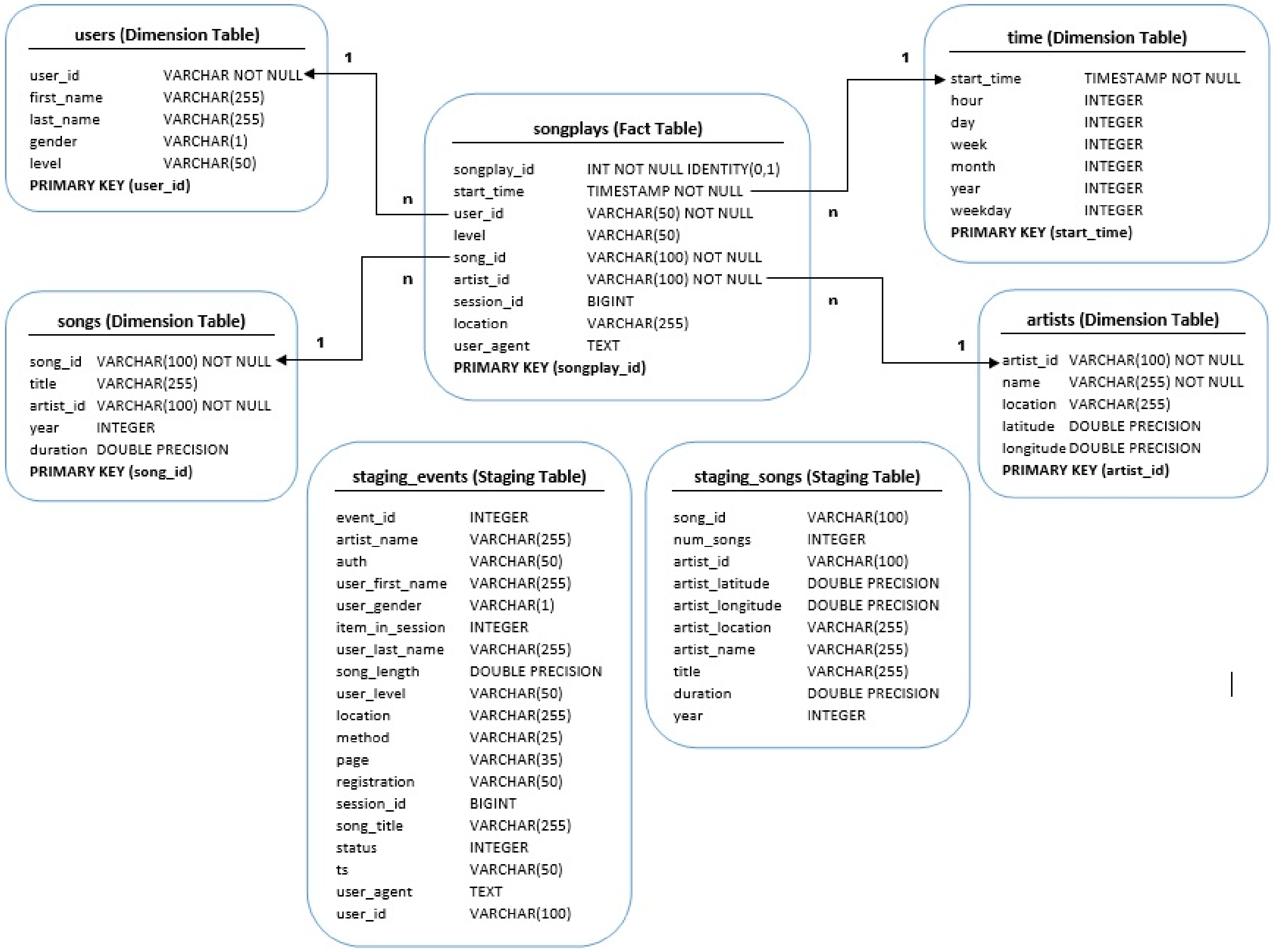
Schema definition

To represent this context a star schema has been used.

The songplays table is the core of this schema, is it our fact table and it contains foreign keys to four tables:

```
* start_time REFERENCES time(start_time)
* user_id REFERENCES users(user_id)
* song_id REFERENCES songs(song_id)
* artist_id REFERENCES artists(artist_id)
```

There are also two staging tables; One for event dataset and one for song dataset.



users (Dimension Table)

user_id	VARCHAR NOT NULL
first_name	VARCHAR(255)
last_name	VARCHAR(255)
gender	VARCHAR(1)
level	VARCHAR(50)

PRIMARY KEY (user_id)

time (Dimension Table)

start_time	TIMESTAMP NOT NULL
hour	INTEGER
day	INTEGER
week	INTEGER
month	INTEGER
year	INTEGER
weekday	INTEGER

PRIMARY KEY (start_time)

songplays (Fact Table)

songplay_id	INT NOT NULL IDENTITY(0,1)
start_time	TIMESTAMP NOT NULL
user_id	VARCHAR(50) NOT NULL
level	VARCHAR(50)
song_id	VARCHAR(100) NOT NULL
artist_id	VARCHAR(100) NOT NULL
session_id	BIGINT
location	VARCHAR(255)
user_agent	TEXT

PRIMARY KEY (songplay_id)

songs (Dimension Table)

song_id	VARCHAR(100) NOT NULL
title	VARCHAR(255)
artist_id	VARCHAR(100) NOT NULL
year	INTEGER
duration	DOUBLE PRECISION

PRIMARY KEY (song_id)

artists (Dimension Table)

artist_id	VARCHAR(100) NOT NULL
name	VARCHAR(255) NOT NULL
location	VARCHAR(255)
latitude	DOUBLE PRECISION
longitude	DOUBLE PRECISION

PRIMARY KEY (artist_id)

staging_events (Staging Table)

event_id	INTEGER
artist_name	VARCHAR(255)
auth	VARCHAR(50)
user_first_name	VARCHAR(255)
user_gender	VARCHAR(1)
item_in_session	INTEGER
user_last_name	VARCHAR(255)
song_length	DOUBLE PRECISION
user_level	VARCHAR(50)
location	VARCHAR(255)
method	VARCHAR(25)
page	VARCHAR(35)
registration	VARCHAR(50)
session_id	BIGINT
song_title	VARCHAR(255)
status	INTEGER
ts	VARCHAR(50)
user_agent	TEXT
user_id	VARCHAR(100)

staging_songs (Staging Table)

song_id	VARCHAR(100)
num_songs	INTEGER
artist_id	VARCHAR(100)
artist_latitude	DOUBLE PRECISION
artist_longitude	DOUBLE PRECISION
artist_location	VARCHAR(255)
artist_name	VARCHAR(255)
title	VARCHAR(255)
duration	DOUBLE PRECISION
year	INTEGER

Preamble

In this project we are going to use two Amazon Web Services, S3 (Data storage) and Redshift (Data warehouse with columnar storage).

Data sources are provided by two public S3 buckets. One bucket contains info about songs and artists, the second has info concerning actions done by users (which song are listening, etc..). The objects contained in both buckets are JSON files. The song bucket has all the files under the same directory but the event ones don't, so we need a descriptor file (also a JSON) in order to extract data from the folders by path. We used a descriptor file because we don't have a common prefix on folders.

The Redshift service is where data will be ingested and transformed, in fact though COPY command we will access to the JSON files inside the buckets and copy their content on our staging tables.

Redshift Considerations

The schema design in redshift can heavily influence the query performance associated. Some relevant areas for query performance are:

```
* Defining how redshift distributes data across nodes.
* Defining the sort keys, which can determine ordering and speed up joins.
* Definining foreign key and primarty key constraints.
```

Data Distribution

How data is distributed is orchestrated by the selected distribution style. When using a 'KEY' distribution style, we inform redshift on how the data should be distributed across nodes, as data will be distributed such that data with that particular key are allocated to the same node.

A good selection for this distribution keys is such that data is distributed evenly, such as to prevent performance hotspots, with collocating related data such that we can easily perform joins. We essentially want to perform joins on columns which are a distribution key for both the tables. Then, redshift can run joins locally instead of having to perform network I/O. We want to choose one dimension table to use as the distribution key for a fact table when using a star schema. We want to use the dimension table which is most commonly joined.

For a slowly changing dimension table, of relatively small size (<1M entries in the case of Redshift) using an 'ALL' distribution style is a good choice. This distributes the table across all nodes for each of retrieval and performance.

Primary & Foreign Key Constraints

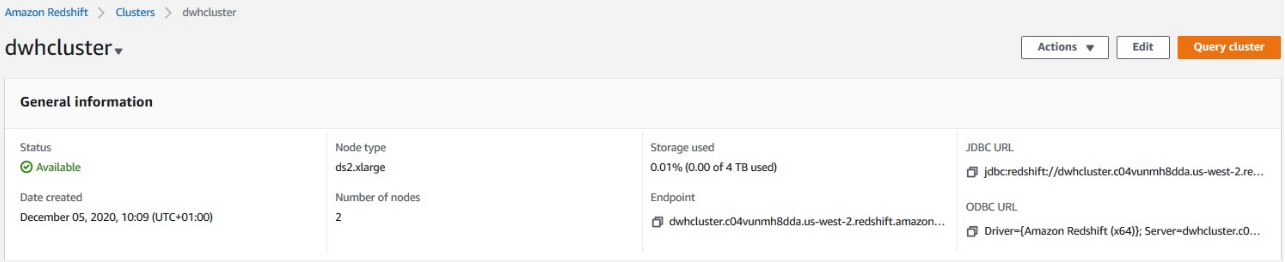
We can declare primary key and foreign key relationships between dimensions and fact tables for star schemas. Redshift uses this information to optimize queries, by eliminating redundant joins. We must ensure that primary key constraints are enforced, with no duplicate inserts.

ETL process

In this project most of ETL is done with SQL (Python used just as bridge), transformation and data normalization is done by Query, check out the sql_queries python module.

Comments.

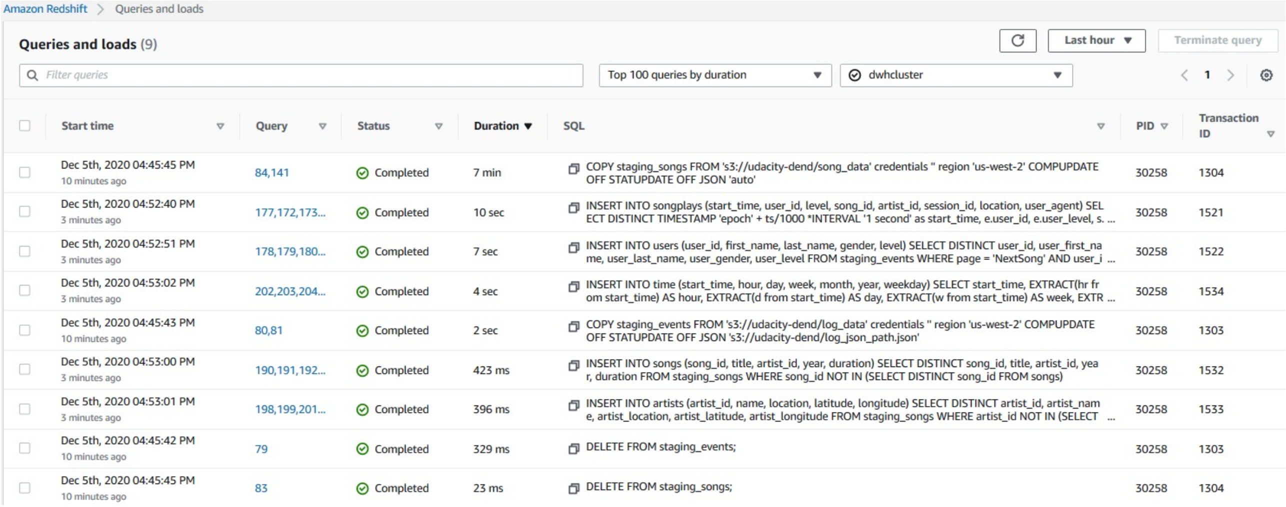
Information on the redshift cluster.



General information

Status ✓ Available	Node type ds2.xlarge	Storage used 0.01% (0.00 of 4 TB used)	JDBC URL jdbc:redshift://dwhcluster.c04vunmh8dda.us-west-2.re...
Date created December 05, 2020, 10:09 (UTC+01:00)	Number of nodes 2	Endpoint dwhcluster.c04vunmh8dda.us-west-2.redshift.amazon...	ODBC URL Driver={Amazon Redshift (x64)}; Server=dwhcluster.c0...

All the queries that are executed in the cluster.


Amazon Redshift > Queries and loads
Last hour ▾
Terminate query
Filter queries
Top 100 queries by duration ▾
dwhcluster ▾
< 1 >

Queries and loads (9)

□	Start time	Query	Status	Duration ▼	SQL	PID	Transaction ID
□	Dec 5th, 2020 04:45:45 PM 10 minutes ago	84,141	✓ Completed	7 min	COPY staging_songs FROM 's3://udacity-dend/song_data' credentials '' region 'us-west-2' COMPUPDATE OFF STATUPDATE OFF JSON 'auto'	30258	1304
□	Dec 5th, 2020 04:52:40 PM 3 minutes ago	177,172,173...	✓ Completed	10 sec	INSERT INTO songplays (start_time, user_id, level, song_id, artist_id, session_id, location, user_agent) SELECT DISTINCT TIMESTAMP 'epoch' + ts/1000 *INTERVAL '1 second' as start_time, e.user_id, e.user_level, s. ...	30258	1521
□	Dec 5th, 2020 04:52:51 PM 3 minutes ago	178,179,180...	✓ Completed	7 sec	INSERT INTO users (user_id, first_name, last_name, gender, level) SELECT DISTINCT user_id, user_first_name, user_last_name, user_gender, user_level FROM staging_events WHERE page = 'NextSong' AND user_i ...	30258	1522
□	Dec 5th, 2020 04:53:02 PM 3 minutes ago	202,203,204...	✓ Completed	4 sec	INSERT INTO time (start_time, hour, day, week, month, year, weekday) SELECT start_time, EXTRACT(hr from start_time) AS hour, EXTRACT(d from start_time) AS day, EXTRACT(w from start_time) AS week, EXTR ...	30258	1534
□	Dec 5th, 2020 04:45:43 PM 10 minutes ago	80,81	✓ Completed	2 sec	COPY staging_events FROM 's3://udacity-dend/log_data' credentials '' region 'us-west-2' COMPUPDATE OFF STATUPDATE OFF JSON 's3://udacity-dend/log_json_path.json'	30258	1303
□	Dec 5th, 2020 04:53:00 PM 3 minutes ago	190,191,192...	✓ Completed	423 ms	INSERT INTO songs (song_id, title, artist_id, year, duration) SELECT DISTINCT song_id, title, artist_id, year, duration FROM staging_songs WHERE song_id NOT IN (SELECT DISTINCT song_id FROM songs)	30258	1532
□	Dec 5th, 2020 04:53:01 PM 3 minutes ago	198,199,201...	✓ Completed	396 ms	INSERT INTO artists (artist_id, name, location, latitude, longitude) SELECT DISTINCT artist_id, artist_name, artist_location, artist_latitude, artist_longitude FROM staging_songs WHERE artist_id NOT IN (SELECT ...	30258	1533
□	Dec 5th, 2020 04:45:42 PM 10 minutes ago	79	✓ Completed	329 ms	DELETE FROM staging_events;	30258	1303
□	Dec 5th, 2020 04:45:45 PM 10 minutes ago	83	✓ Completed	23 ms	DELETE FROM staging_songs;	30258	1304

A possible improvement for data management would have been to use a 'sort key' determines the order with which data is stored on disk for a particular table. Query Performance is increased when the sort key is used in the where clause. Only one sort key can be specified, with multiple columns. Using a 'Compound Key', specifies precedence in columns, and sorts by the first key, then the second key. 'Interleaved Keys', treat each column with equal importance. Compound keys can improve the performance of joins, group by, and order by statements.