CS 327E Final Project: Milestone 2

Prerequisites:

1. Completed Milestone 1.
2. Continuing to work with your partner.

Step 1. Download the **movie_tags.py** pyspark script from this link: https://github.com/cs327e-fall2017/snippets/blob/master/movie_tags.py. This is the starter code for the Spark program you will implement in this milestone. Open the program in a Python editor of your choice and read through the code.

Step 2. Open a **psql** session and connect to your Postgres RDS. Create a new table **Title_Tags** in your IMDB database. This table will store unique pairs of **title_id, tag**. The **title_id** represents the usual IMDB title identifier and the **tag** is the label given to a movie by a user in the movielens dataset. The tag column should be stored as a varchar(300) in the database. Create the primary key on this table, but do not create a foreign key constraint yet as this will slow down the load for this table. Create the table and copy the create table statement to a file named **create_title_tags.sql**. Add this file to your git repo.

Step 3. Open the **tags.csv** file from the movielens dataset. Familiarize yourself with the format of this file. Implement a function for **map()** that takes as input a line from tags.csv, parses the movie_id and tag elements, and outputs a movie_id, tag pair. Pass this function to **map** and save the output RDD as **rdd_tags**.

Step 4. The mapped RDD contains duplicate movie_id, tag pairs. Remove all duplicates using the RDD transformation **distinct()** and store the new RDD as **rdd_distinct_tags**. Refer to the Spark Programming Guide for more details on the **distinct()** transformation [3].

Step 5. The code for parsing the **links.csv** file is identical to the previous milestone and is provided to you as part of the starter code. It produces **rdd_links** which shares an element in common with **rdd_tags**, namely **movie_id**. Join **rdd_links** with **rdd_distinct_tags** on movie_id to produce a joined RDD **rdd_joined**. Refer to the Spark Programming Guide for details on how to join two RDDs [3].

Step 6. The joined RDD is now mapped with the user function **add_imdb_id_prefix()**. The mapped RDD contains elements of **imdb_id, tag** pairs, where the imdb_id value now has the appropriate tt0* prefix used by our IMDB database.

Step 7. In the **save_to_db()** function, perform a database insert for each element of the input list. The insert should write the values to the new **Title_Tags** table.

Step 8. Create a new EMR cluster by cloning a previously terminated cluster. Configure the cluster and copy your script to the master node. Run the Spark job and debug any errors encountered during execution. For this step, you should follow the procedures in our Clone EMR cluster guide, Connect and Configure EMR cluster guide, and Running Spark Scripts guide, all accessible from our Snippet Wiki [1]. Once you have verified the code, add your error-free **movie_tags.py** to your git repo.

Step 9. Connect to your IMDB database and find the number of records in the **Title_Tags** table. Copy the SQL query and output into a text file. Save the text file as **title_tags.out** and add it to your git repo.

Step 10. There are some records in the **Title_Tags** table which don't have an associated parent record in the **Title_Basics** table. These records should be removed before creating a foreign key on the **Title_Tags.title_id** column. Remove those records (188 total) with the following DELETE statement:

```
DELETE FROM Title_Tags WHERE title_id in
(
   SELECT distinct tt.title_id
   FROM title_tags tt LEFT OUTER JOIN title_basics tb
   ON tt.title_id = tb.title_id
   WHERE tb.title_id IS NULL
);
```

Now create the foreign key on **Title_Tags.title_id**. The foreign key should point to the **title_id** column of its parent table. The ALTER TABLE command page [4] in the Postgres manual has an example on how to add a foreign key constraint to an existing table. Copy the foreign key statement to a file named **alter_title_tags.sql**. Add this file to your git repo.

Step 11. Write an aggregate query that accesses the new table in some interesting way and wrap this query inside a view. This query should join **Title_Tags** with **Title_Basics**. Name this view **v_title_tags**. The view should be virtual if it executes in under 10 seconds or materialized if it runs for 10 seconds or longer. Create the view in your IMDB database and add the view definition to a file. Name the file **v_title_tags.sql** and add it to your git repo .

Step 12. Create a QuickSight analysis that visualizes the output from your view. Create a dashboard for the analysis and share it with the IAM admin user. Also, take a screenshot of the analysis and save it as a png or jpg format. Add the screenshot to your git repo.

Step 13. Locate the commit id that you will be using for your submission. This is a long 40-character that shows up on your main GitHub repo page next to the heading "Latest commit" (e.g. commit 6ca6f695bca36f7fc2c33485d1080ae30f8b9928). Locate the link to your GitHub repo (e.g. https://github.com/cs327e-fall2017/xyz.git where xyz is your repo name). Go back to your existing Stache entry and locate the read-only API endpoint and read key.

Replace the commit id, repo link, API endpoint, and read key in the json string below with your own:

```
{
  "repository-link": "https://github.com/cs327e-spring2017/xyz.git",
  "commit-id": "6ca6f695bca36f7fc2c33485d1080ae30f8b9928",
  "stache-endpoint": "/api/v1/item/read/61515",
  "stache-read-key": "b2eacb0387a919e33b27e7c03a6c5d84b71234795732be33eb28711ec16f0e21"
}
```


Create a submission.json file that contains your modified json string. Click on the Final Project Milestone 2 in Canvas and upload submission.json. **Do not add submission.json to your git repo**.

This submission is due by **Friday, 11/10 at 11:59pm**. If it's late, there will be a 10% grade reduction per late day. This late policy is also documented in the syllabus.

Additional Notes:

- The EMR service is not covered by the free tier program and it costs ~$8/hour for a single-node cluster on an m3.xlarge or m4.xlarge instance. These charges apply even when the cluster is sitting idle and there is no option to stop or suspend the EMR cluster. You should therefore always **terminate** your EMR cluster when it is not in use to avoid unnecessary charges. Follow the steps in our EMR Cluster Termination guide to destroy your EMR cluster: https://github.com/cs327e-fall2017/snippets/wiki/Terminating-EMR-cluster.

- You should perform the fixes to your code locally and not directly on the EMR master node. Each time you make a change, use scp to transfer the new script to EMR and then re-run the Spark job. In order to do this efficiently, you should have two terminal sessions open, one ssh session for running the job and one scp session for transferring the file.

References and Additional Resources:

[1] Snippets Wiki: https://github.com/cs327e-fall2017/snippets/wiki
[2] Milestone 2 Grading Rubric: http://www.cs.utexas.edu/~scohen/projects/m2-rubric.pdf
[3] Spark Programming Guide: https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html
[4] Postgres ALTER TABLE command: https://www.postgresql.org/docs/9.6/static/sql-altertable.html