CS 327E Lab 3: Aggregate Queries and Visualizations

Prerequisites:
1. Completed Lab 2.
2. Continuing to work with your partner.
3. Set up an Amazon QuickSight account.
4. Connected to Postgres RDS from QuickSight.

Part 1:

Step 0. Follow the steps in our snippets wiki to satisfy prerequisites 3-4 [1].

Step 1. Create a new folder in your local git repository called **lab3**. All the work that your team will do for this lab will go into this folder.

Step 2. Write 6 aggregate queries over your IMDB database. Together, these 6 queries should satisfy the following criteria:

- Each of the 10 base tables must be accessed at least once
- Each query must have at least 1 `aggregate function`
- Each query must have exactly 1 `group by clause`
- Each query must have exactly 1 `order by clause`
- There must be at least 3 `inner joins` across all queries
- There must be at least 2 `outer joins` across all queries
- There must be at least 3 `where clauses` across all queries
- There must be at least 3 `having clauses` across all queries

Run each query in psql and fix any syntax errors. Review the output from each query. If the results look surprising or strange, break-up the final query into smaller units and verify each one. Fix any logic flaws that you encounter. Once you have verified the query, use the following format to document it:

/* Query 1: Directors who have directed at least 5 titles per year between the years 2007 - 2017 */

```
select primary_name as director_name, start_year as year, count(*) as
title_count
from title_basics tb join directors d on tb.title_id = d.title_id
join person_basics pb on pb.person_id = d.person_id
where start_year between 2007 and 2017
group by primary_name, start_year
having count(*) >= 5
order by start_year, count(*);
```

Notice that the comment above the SQL provides a brief explanation of the query in English.

Place the 6 queries in a file called imdb_aggr_queries.sql and add it to your git repo.

Step 3. Create a virtual view or materialized view for each aggregate query in Step 2. The choice of the view will depend on the execution time of each query. To time the query, run the following commands in psql:

```
\timing
\pset pager
SELECT ...
```

The output will report the query's execution time in milliseconds. If the execution time < 10 seconds, create a virtual view for the query. If the execution time >= 10 seconds, create a materialized view for the query. Give each view a short descriptive name and prefix the name with "v_" so that it can be distinguished easily from the database tables. For example, `v_productive_directors`.

To create a virtual view, use the syntax:

```
CREATE VIEW v_productive_directors AS
  SELECT …
```

To create a materialized view, use the syntax:

```
CREATE MATERIALIZED VIEW v_productive_directors AS
  SELECT …
```

To modify a view, first drop the view using the `DROP VIEW` or `DROP MATERIALIZED VIEW` command and then re-create it with the modified SQL query. Refer to the instacart code snippets [4] for working examples on how to create virtual and materialized views from an aggregate query.

If you created a materialized view, you should also query the view by running:

```
select * from v_productive_directors;
```

If the execution time is still >= 10 seconds, add a limit clause to the query to bring the time under 10 seconds. If you already have a limit clause, reduce the number of rows returned from the query.

Place the SQL for the 6 views in a file called imdb_views.sql and add it to your git repo.

Step 4. Create a visualization (or what QuickSight calls an "Analysis") for each of the 6 views that you created in Step 3. QuickSight requires that a data set be created for each accessed table or view, so you will end up with 6 data sets in this section.

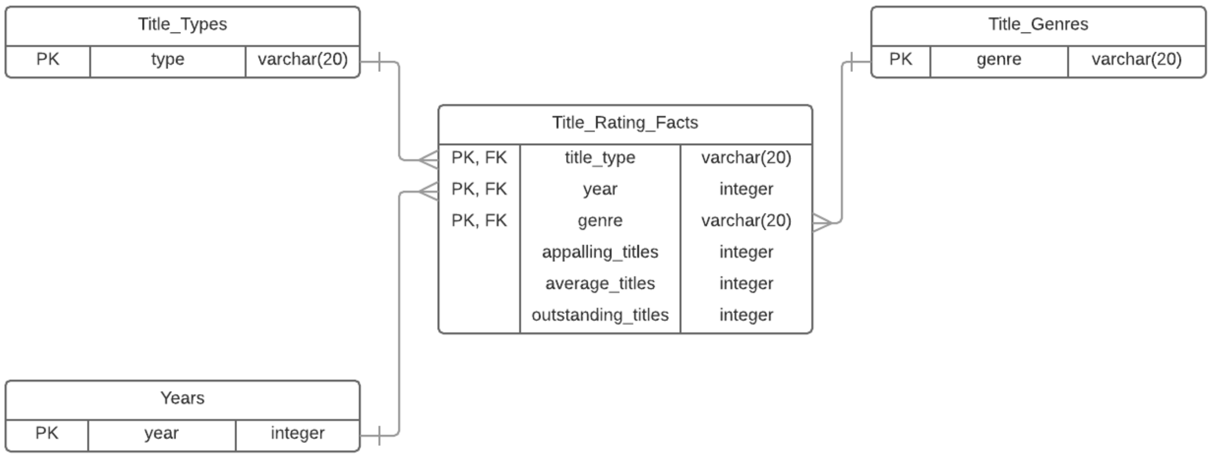
To create a data set, choose the data source for your Postgres RDS instance and select the view that you want to visualize (e.g. `v_productive_directors`). If you created a virtual view, the view shows up in the table list when creating a new data set. If you created a materialized view, you won't see the view show up there and you'll need to use the custom SQL tool to find it. To do that, choose "Edit/Preview data" and then click on the down arrow next to "tables" on the left-hand side of the screen. Select the link "Switch to custom SQL" and enter the query in the Custom SQL text box. For the materialized view in this example, the query would be: `select * from v_productive_directors`.

From the Analysis screen, select the fields to visualize and the visual type. You'll want to experiment with different visual types until you find one that visualizes the data in a compelling way. If Auto-Graph does a good job, then you don't need to look for an alternate visual type.

You should now have 6 visuals or analyses, one corresponding to each database view. Take a screenshot of each analysis/visual and add it to your lab3 folder. Also, create a dashboard for each analysis/visual and share it with the IAM admin user by following the steps documented in our QuickSight wiki [1].

Part 2:

Step 5. We're now going to switch gears and return to the dimensional schema from Lab 2. This schema has a central fact table with three fact columns: appalling_titles, average_titles, and outstanding_titles. These facts store counts of titles that are "appalling", "average" and "outstanding" in their respective columns. Recall that to be considered "appaling", a title's average rating must be <= 2.0; to be considered "average", a title's average rating is between 2.1 and 7.9; and to be considered "outstanding", a title rating's is >= 8.0. These three facts are organized by three dimension columns: title_type, year, and genre. Below is a diagram of the schema:



a) Write an aggregate query over the IMDB base tables to compute each fact and its dimension columns as follows:

- `title_type, year, genre, appalling_titles`
- `title_type, year, genre, average_titles`

- `title_type, year, genre, outstanding_titles`

b) Use the `CREATE TABLE AS SELECT` construct to create a derived table for storing the results from each query. Name the tables `Title_Rating_Facts_Appalling`, `Title_Rating_Facts_Average`, and `Title_Rating_Facts_Outstanding`. Create the three individual fact tables in your IMDB database.

c) Write a query that joins the individual fact tables, `Title_Rating_Facts_Appalling`, `Title_Rating_Facts_Average`, and `Title_Rating_Facts_Outstanding`, to output the final fact table, `Title_Rating_Facts`. Hint: think carefully about the join types for this query to avoid losing records from the individual fact tables.

d) For each fact column in the `Title_Rating_Facts` table, update all the null values to 0.

e) Create a primary key on the `Title_Rating_Facts` table. Hint: you will need to remove a few records from the table in order to successfully create the primary key. Make sure you document the delete statement you used to remove those records!

f) Write an aggregate query over `Title_Rating_Facts` that adds up the number of outstanding titles grouped by `year` and `genre`. Note: we are ommiting `title_type` from this query. Include only the years since 1930. Include only the groups that have > 0 outstanding titles. Order the results first by year and and second by genre and limit the results to 100 records. Create a virtual view for this query and name the view `v_outstanding_titles_by_year_genre`.

g) Create a visual in QuickSight to visualize the output from the outstanding titles view. As before, create a QuickSight data set for the view and then create the analysis. Take a screenshot of your visual and add it to your repo. Also, create a dashboard from the analysis and share it with the IAM admin user.

Place the SQL statements for steps a) – f) in a file called imdb_populate_fact.sql. Add this file to your git repo.

Step 6. Find the **Stache** entry that you used for Lab 2. Update the entry to include the QuickSight account details:

```
{
  "aws-username": "shouldbeAdmin",
  "aws-password": "mypassword",
  "aws-console-link": "myconsolelink",
  "rds-endpoint-link": "myrdsendpoint_without_port_number",
  "rds-username": "shouldbeMaster",
  "rds-password": "myrdspassword",
  "quicksight-account":  "what-you-named-your-QuickSight-account"
}
```

Make sure to save the updates to your Stache entry so that we can access your QuickSight account.

Step 7. 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 the 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 Lab 3 Assignment in Canvas and upload submission.json. **Do not add submission.json to your git repo**.

This submission is due by **Friday, 10/20 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.

References and Resources:

[1] Snippets Wiki: https://github.com/cs327e-fall2017/snippets/wiki
[2] Lab 3 Grading Rubric: http://www.cs.utexas.edu/~scohen/projects/lab3-rubric.pdf
[3] Instacart Aggregate Queries: https://tinyurl.com/y6uj6f3w
[4] Instacart Views: https://tinyurl.com/yc6wqpzo