UDACITY

< Return to Classroom

Investigate a Relational Database

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Student,
You have put dedicated effort into this project and it paid off. Congratulations on meeting all the specifications of the project! You have demonstrated a very good **SQL** skills. You have done an excellent job querying the given database and answering some interesting questions like **how the two stores in the dataset compare with respect to their rental orders and finding out that there is not much of a difference**.

You also did a fantastic job of incorporating the **previous reviewer** suggestions. **As a different reviewer,** I have left some additional comments. I made these comments marked as Suggestions to help you improve the project. It does not require you to resubmit the project. You have already passed the project. **Congratulations!** If you are uploading this project to your portfolio or sharing it with your potential employer, it is a good idea to address these comments. It also gives you an opportunity to appreciate the complete essence of this project. Keep up all the great work you are doing. Good luck with your future projects!

Here are a few resources that may help your continued learning:

- To further develop your skills in **SQL**, you can practice with HackerRank, which challenges you with increasingly difficult problems.
- To know more about different kind of charts and their design, you can take a look at Data Visualization with Tableau given in EXTRACURRICULAR section of this nanodegree.

Queries

All SQL queries run without errors and produce the intended results.

Excellent job writing error-free SQL queries, which effectively used `GROUP BY` and `AGGREGATION` clauses. You have efficiently combined many SQL clauses and functions in some of the complex queries like the one below.

```
WITH top3 AS (
    SELECT
        c.customer_id,
        SUM(p.amount) AS total_amount
    FROM
        customer AS c
        JOIN payment AS p
                ON c.customer_id = p.customer_id
    GROUP BY
        c.customer_id
    ORDER BY
        total_amount DESC
    LIMIT 3),
tab2 AS (
    SELECT
        (first_name || ' ' || last_name) AS full_name,
        DATE_TRUNC('month', payment_date) AS pay_date_mon,
        COUNT(p.amount) AS pay_count_mon,
        SUM(p.amount) AS pay_amount_mon
    FROM
        top3
        JOIN customer AS c
                ON top3.customer_id = c.customer_id
        JOIN payment AS p
                ON p.customer_id = c.customer_id
    WHERE
        payment_date >= '2007-01-01'
        AND payment_date < '2008-01-01'
    GROUP BY
        full_name,
        pay_date_mon
    ORDER BY
        full_name,
        pay_date_mon
)
SELECT
    full_name,
    pay_date_mon,
    pay_amount_mon,
    LAG(pay_amount_mon) OVER (
                                                        PARTITION BY full_name
                                                        ORDER BY pay_date_mon) AS
```

```
                                                          ORDER BY pay_date_mon) AS
lag,
        pay_amount_mon - LAG(pay_amount_mon) OVER (

PARTITION BY full_name

ORDER BY pay_date_mon) AS mon_diff
FROM
    tab2;
```

Excellent job making sure that there is **no additional data prep** (sorting, filtering, renaming, etc.) between the query output and the visualization by doing all required aggregation in the SQL query itself.

Each SQL query needs to include one or more explicit JOINs. The JOIN or JOINs should be necessary to the query.

If a question does not require a JOIN please change the question to be one that does.

Each SQL query needs to include one or more aggregations. This could be a COUNT, AVG, SUM, or other aggregation.

At least 2 of the 4 SQL queries need to include either a subquery OR a CTE.

Fabulous job with the **CTEs**! This is one of the challenging parts of SQL. You have demonstrated a solid grasp of this concept. You effectively used CTEs in your queries, where it is relevant.

Note: It is also important to understand the difference between a CTE and subquery. A subquery has to be defined repeatedly, but a CTE can be defined once and then can be referenced multiple times. See this StackOverflow thread to further understand the difference between subquery and CTE.

At least 1 of the 4 queries should use a Window Function.

Splendid job using `NTILE()` WINDOW function effectively to get standard quartile and `LAG()` to get difference between adjoining rows. This is another challenging part of the SQL and it is nice to see your mastery in this concept. Here is a useful resource to know more about PostgreSQL window function.

The SQL queries are well formatted and use aliases.

It is good that you terminated each SQL query with a semicolon. See this stackoverflow thread to understand why it is a good idea to terminate SQL query with a semicolon.

Good job **commenting out** all the text that is not part of the SQL query.

Presentation

Each slide should have a question and an appropriate visualization descriptions to address the question. The slides should be free of significant factual, spelling and grammatical mistakes.

Suggestions

You used **only bar charts in this project, which is fine as far as completing this project.** But it is a good idea to know about different kind of charts one can use to represent different kinds of insights. For instance, when data involves time series (month, year etc.) **line chart** works best (e.g., store performance over time). To depict relationship between two quantitative variables **scatterplot** works best (e.g., relationship between rental duration of movies and number of rentals). To depict distribution of variables **histogram** or **box plot** is suitable (e.g., distribution of number of rentals per movie). Here are some resources that help you choosing right kind of visualizations to represent various types of data/insights.

Choosing the right chart type for your insight.
How to pick the right chart type.
Data Visualization with Tableau.

All visualizations should make logical sense and provide accurate analysis based on their query results.

1. All visualizations include a title and axis labels, have a legend where applicable, and are easily understood.
2. Every visualization should have:
    - chart title
    - x axis title
    - x axis label
    - y axis title
    - y axis labels

Visualizations are formatted well paying attention to each of the chart elements like **chart title, axis titles** and **axistick labels**.

DOWNLOAD PROJECT

RETURN TO PATH