D191 PERFORMANCE ASSESSMENT

A. **Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.**

A comprehensive business report that summarizes the DVD business's active customers based on the number of rentals they have made. This report will provide valuable insights for targeted marketing strategies, customer retention initiatives, and promotional campaigns. The report includes essential details such as the customer's first name, last name, and email address for contact information. It further includes metrics such as total\_rentals, indicating each customer's overall engagement with the rental services. Additionally, the report categorizes customers into frequent\_rentals and non\_frequent\_rentals based on a custom transformation. "Frequent" customers, those with five or more rentals, are identified to highlight their loyalty and higher rental activity, while "non-frequent" customers have less than five rentals, signifying occasional renters. This detailed report will empower the DVD business to develop targeted marketing strategies, foster customer retention, and execute effective loyalty rewards programs based on customers' rental behavior.

1. **Identify the specific fields that will be included in the detailed table and the summary table of the report.**

DETAILED TABLE

The table contains the following relevant fields:

FROM THE CUSTOMER TABLE

1. customer\_id: A unique identifier for each customer, enabling proper identification.

2. first\_name: The customer's first name, for personalized communication.

3. last\_name: The customer's last name, ensuring accurate identification.

4. email: The customer's email address, providing a means of communication.

FROM THE RENTAL TABLE

5. rental\_id: A unique identifier for each rental transaction, ensuring traceability.

6. rental\_date: The date of each rental, aiding in tracking rental trends over time.

7. total\_rentals: A count of total rentals made by each customer, indicating their rental history.

8. frequent\_customer: A custom-transformed field indicating whether the customer is "frequent" or "non-frequent" based on the number of rentals they made.

SUMMARY TABLE

The table includes the following essential fields:

1. first\_name: The customer's first name.

2. last\_name: The customer's last name.

3. email: The customer's email address, providing contact information.

4. total\_rentals: The total number of rentals made by each customer, indicating their overall engagement with our rental services.

5. frequent\_rentals: The count of rentals categorized as "frequent" based on custom transformation. A customer is marked as "frequent" if they have made five or more rentals, signifying their loyalty and higher rental activity.

6. non\_frequent\_rentals: The count of rentals categorized as "non-frequent" based on custom transformation. Customers with less than five rentals fall into this category, indicating occasional renters.

2. **Describe the types of data fields used for the report**.

The table contains the data fields:

1. customer\_id: INT (Numeric Data)

2. first\_name: VARCHAR (Varying Character Text Data)

3. last\_name: VARCHAR (Varying Character Text Data)

4. email: VARCHAR (Varying Character Text Data)

5. rental\_id: SMALLINT (Numeric Data)

6. rental\_date: DATETIME (Date/Time Data)

7. total\_rentals: INT (Numeric Data)

8. frequent\_customer: CHAR (Fixed Character Text Data)

3. **Identify *at least* two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report**.

* Detailed Table: customer, rental.
* Summary Table: customer, rental.

4.  **Identify *at least* one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of *N* to *No* and *Y* to *Yes*)**.

The field **frequent\_customer** requires a custom transformation because it takes the count of rentals (total\_rentals) and categorizes customers as "Yes" if they have made five or more rentals (considered frequent customers), and "No" if they have made fewer than five rentals. This transformation provides additional insight into customer behavior and allows stakeholders to identify customers with higher rental activity, potentially helping to target loyalty programs or personalized marketing offers to frequent customers.

5. **Explain the different business uses of the detailed table section and the summary table section of the report.**

DETAILED TABLE

The detailed table, named "customer\_rental\_history," provides a comprehensive breakdown of each customer's rental transactions. It offers an in-depth view of individual rental activities, allowing stakeholders to delve into specific rental details. The "customer\_rental\_history" detailed table is valuable for thorough analysis, assisting in identifying individual rental patterns, evaluating customer preferences, and facilitating targeted customer service efforts. This level of detail equips stakeholders with a comprehensive understanding of customer behavior and supports data-driven decision-making to enhance overall business performance.

SUMMARY TABLE

The summary table, named "customer\_summary," provides a concise and insightful overview of each customer's rental history. It presents key aggregated data, enabling a quick assessment of customer rental patterns and preferences. In this table, we group the data by customer's first name, last name, and email, ensuring a unique identifier for each customer. The "customer\_summary" table will serve as a powerful tool for decision-makers, offering a comprehensive view of customer behavior and rental habits, facilitating targeted marketing strategies, and fostering customer retention initiatives.

6. **Explain how frequently your report should be refreshed to remain relevant to stakeholders**.

The report should be scheduled to run automatically on a MONTHLY basis, ensuring that the data remains up-to-date for shareholders and managers to monitor each customer’s rental transaction, history rental patterns, preferences, and facilitating targeted customer service efforts. This scheduled execution guarantees that the stored procedure runs consistently and regularly, maintaining data freshness and providing the necessary information for decision-making and analysis.

**B. Provide original code for function(s) in text format that perform the transformation(s) you identified in part A4.**

-- Create the user-defined function

CREATE OR REPLACE FUNCTION transform\_frequent\_customer(count\_value INTEGER)

RETURNS TEXT AS

$$

BEGIN

IF count\_value >= 5 THEN

RETURN 'Yes';

ELSE

RETURN 'No';

END IF;

END;

$$

LANGUAGE plpgsql;

**C. Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.**

**Detailed Table**

-- Create the detailed table

DROP TABLE IF EXISTS customer\_rental\_history

CREATE TABLE customer\_rental\_history AS

SELECT c.customer\_id, c.first\_name, c.last\_name, c.email, r.rental\_id, r.rental\_date,

COUNT(r.rental\_id) AS total\_rentals,

transform\_frequent\_customer(COUNT(r.rental\_id)) AS frequent\_customer

FROM

customer c

LEFT JOIN

rental r ON c.customer\_id = r.customer\_id

GROUP BY

c.customer\_id, c.first\_name, c.last\_name, c.email, r.rental\_id, r.rental\_date;

**Summary Table**

-- Create the summary table

DROP TABLE IF EXISTS customer\_summary

CREATE TABLE customer\_summary AS

SELECT first\_name, last\_name, email, COUNT(rental\_id) AS total\_rentals,

SUM(CASE WHEN transform\_frequent\_customer(total\_rentals) = 'Yes' THEN 1 ELSE 0 END) AS frequent\_rentals,

SUM(CASE WHEN transform\_frequent\_customer(total\_rentals) = 'No' THEN 1 ELSE 0 END) AS non\_frequent\_rentals

FROM customer\_rental\_history

GROUP BY first\_name, last\_name, email;

**Explanation**

**Summary Table:**

1. The SELECT statement retrieves the necessary fields for the summary table FROM the detailed table ‘customer\_rental\_history’.

2. The ‘GROUP BY’ clause groups the data by ‘first\_name’, ‘last\_name’, and ‘email’, which represents a unique customer identifier. This grouping will ensure that we get summary information for each customer.

3. The ‘COUNT(rental\_id)’ function calculates the ‘total\_rentals’ for each customer, which represents the total number of rentals made by each customer.

4. The ‘SUM(CASE WHEN ...)’ statements are used to calculate the number of frequent and non-frequent rentals for each customer based on the `frequent\_customer` field from the `customer\_rental\_history` table. If the `frequent\_customer` is 'Yes', it is considered a frequent rental; if 'No', it is considered a non-frequent rental.

5. The results are stored in the `customer\_summary` table, which will serve as the suitable summary table for the business.

**Detailed Table:**

The SELECT statement retrieves the necessary fields for the detailed report: customer\_id, first\_name, last\_name, email, rental\_id, rental\_date, total\_rentals and frequent\_customer.

The query uses a LEFT JOIN clause to combine data from the customer and rental tables. The customer table contains customer information, and the rental table holds rental data.

1. The COUNT() function is used to calculate the total\_rentals for each customer. It counts the number of rentals made by each customer based on their rental IDs.
2. The CASE() statement is used to create the frequent\_customer field. It checks if the total\_rentals for a customer are greater than or equal to 5. If yes, it sets the value to 'Yes'; otherwise, it sets it to 'No'. This transformation categorizes customers as frequent or non-frequent based on the number of rentals they've made.
3. The GROUP BY clause groups the results by customer\_id, first\_name, last\_name, email, rental\_id and rental\_date to ensure each rental is listed separately for each customer.
4. The INTO TABLE clause is used to insert the results into a new table named customer\_rental\_history, which will store the detailed data for the report.

**D. Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.**

-- Extract raw data for detailed section with data transformation function

INSERT INTO customer\_rental\_history (customer\_id, first\_name, last\_name, email, rental\_id, rental\_date, total\_rentals, frequent\_customer )

SELECT c.customer\_id, c.first\_name, c.last\_name, c.email, r.rental\_id, r.rental\_date,

COUNT(r.rental\_id) AS total\_rentals,

transform\_frequent\_customer(COUNT(r.rental\_id)) AS frequent\_customer

FROM

customer c

LEFT JOIN

rental r ON c.customer\_id = r.customer\_id

GROUP BY

c.customer\_id, c.first\_name, c.last\_name, c.email, r.rental\_id, r.rental\_date;

**E. Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.**

– creates trigger function

CREATE OR REPLACE FUNCTION fn\_update\_customer\_summary()

RETURNS TRIGGER AS

$$

BEGIN

-- Update total\_rentals in customer\_summary

UPDATE customer\_summary AS cs

SET total\_rentals = total\_rentals + 1

WHERE cs.first\_name = NEW.first\_name

AND cs.last\_name = NEW.last\_name

AND cs.email = NEW.email;

-- Update frequent\_rentals and non\_frequent\_rentals in customer\_summary

UPDATE customer\_summary AS cs

SET frequent\_rentals = frequent\_rentals + 1

WHERE cs.first\_name = NEW.first\_name

AND cs.last\_name = NEW.last\_name

AND cs.email = NEW.email

AND NEW.frequent\_customer = 'Yes';

UPDATE customer\_summary AS cs

SET non\_frequent\_rentals = non\_frequent\_rentals + 1

WHERE cs.first\_name = NEW.first\_name

AND cs.last\_name = NEW.last\_name

AND cs.email = NEW.email

AND NEW.frequent\_customer = 'No';

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

– create the trigger

CREATE TRIGGER trg\_update\_summary

AFTER INSERT ON customer\_rental\_history

FOR EACH ROW

EXECUTE FUNCTION fn\_update\_customer\_summary();

**F. Provide an original stored procedure in a text format that can be used to refresh the data in *both* the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.**

-- This stored procedure refreshes the data in both the detailed and summary tables.

CREATE OR REPLACE PROCEDURE sp\_refresh\_rental\_data()

LANGUAGE plpgsql

AS

$$

BEGIN

-- Step 1: Clear the contents of the detailed table.

DELETE FROM customer\_rental\_history;

-- Step 2: Perform raw data extraction and insert into the detailed table.

INSERT INTO customer\_rental\_history (

customer\_id,

first\_name,

last\_name,

email,

rental\_id,

rental\_date,

total\_rentals,

frequent\_customer

)

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

c.email,

r.rental\_id,

r.rental\_date,

COUNT(r.rental\_id) AS total\_rentals,

transform\_frequent\_customer(COUNT(r.rental\_id)) AS frequent\_customer

FROM

customer c

LEFT JOIN

rental r ON c.customer\_id = r.customer\_id

GROUP BY

c.customer\_id, c.first\_name, c.last\_name, c.email, r.rental\_id, r.rental\_date;

-- Step 3: Clear the contents of the summary table.

DELETE FROM customer\_summary;

-- Step 4: Perform raw data extraction and insert into the summary table.

INSERT INTO customer\_summary (

first\_name,

last\_name,

email,

total\_rentals,

frequent\_rentals,

non\_frequent\_rentals

)

SELECT

first\_name,

last\_name,

email,

COUNT(rental\_id) AS total\_rentals,

SUM(CASE WHEN transform\_frequent\_customer(total\_rentals) = 'Yes' THEN 1 ELSE 0 END) AS frequent\_rentals,

SUM(CASE WHEN transform\_frequent\_customer(total\_rentals) = 'No' THEN 1 ELSE 0 END) AS non\_frequent\_rentals

FROM

customer\_rental\_history

GROUP BY

first\_name, last\_name, email;

END;

$$;

**1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.**

A relevant job scheduling tool that can be used to automate the stored procedure in PostgreSQL is "**pg\_cron** (Dias, 2022).” pg\_cron is an extension for PostgreSQL that allows you to schedule jobs to run periodically.

To use pg\_cron, you need to install the extension in your PostgreSQL database and configure it accordingly. Once pg\_cron is set up, you can create a cron job that calls the stored procedure sp\_refresh\_rental\_data() at the desired frequency (in this case, MONTHLY). The pg\_cron will handle the execution of the stored procedure as per the defined schedule (monthly ), ensuring that the detailed and summary tables are regularly refreshed with up-to-date data.

**G. Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the functionality of the code used for the analysis.**

LINK: [https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=9a7bdcd2-4f7f-46ad-ade7-b052009b51ee]

**H. Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.**

*PostgreSQL tutorial*. PostgreSQL Tutorial – Comprehensive Postgresql Tutorial. (n.d.-a). http://www.postgresqltutorial.com/   
[This site was used as a reference resource ONLY for concepts such as triggers and stored procedures. I did not copy any code from the site ]

Dias, H. (2022, May 4). *An overview of job scheduling tools for postgresql*. Severalnines. https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/

[This site was used to look up information concerning relevant job scheduling tools used to automate stored procedures in PostgreSQL. I did not copy any code from the site]

No sources were copied for this homework.