D326 Advanced Data Management
Tristan Tate

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.

- ★ **The DVD Rental store wants to accurately find poor performing films based on low rental activity. The goal would be to optimize unused or low usage inventory items OR propose a chance to boost rental activity for these films through marketing or other various efforts.**

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

DETAILED TABLE (Fields)

film_id	Unique id for each movie, used to track performance
title	Name of movie for ease of identifying
rental_rate	Cost to rent movie
total_rentals	Total number of rentals for movie
rental_date	The most recent date the movie was rented
flagged_performance	Indicates whether a movie is performing poorly based on defined rental thresholds

SUMMARY TABLE (Fields)

film_id	Unique id for each poor-performing movie
title	Title for poor-performing movie
Price_category	Text categories for film rental prices (low, medium, high)
total_rentals	Total number of rentals for poor performing movie
days_not_rented	How many days since last rental

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

INTEGER	film_id, total_rentals, days_not_rented
VARCHAR	Title, price_category
NUMERIC	rental_rate
TIMESTAMP	rental_date
BOOLEAN	flagged_performance lagged_performance

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.

film	Film IDs and Titles of Movies
rental	For rental transactions and dates, rental activity
inventory	Acts as a bridge between film and rental, for tracking exact inventory

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).

- ★ There are two fields that can be transformed with a user-defined function.
- ★ **rental_rate** : to be categorized into three categories of (Low, Medium, High) for a column called price_category. This will make it easier to analyze and showcase how pricing could possibly affect performance for certain movies.
- ★ **rental_Date :** to be transformed into a days_not_rented for ease of readability.

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

- ★ **The detailed table** offers a "birds eye view" of all data related to the business question. From here, further insights and patterns can be collected. Understanding the context of all movies and connected attributes will give stakeholders a better visual on how the business question could be answered or found.
- ★ **The summary table** offers a more focused and actionable view. This visual/insight should directly answer the business question on low performing movies. The use cases for this insight can directly assist in decisions for optimizing inventory, marketing, and pricing efforts.

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

- ★ A monthly report should be made to remain relevant and consistent. With movies consistently performing poorly after repeated months, there can be actions taken to cycle out that inventory and bring in new titles. This would keep inventory active and not stale. A monthly report gives adequate time for poor performing movies to improve based on business actions, such as marketing efforts or price adjustments

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

```
CREATE OR REPLACE FUNCTION categorize_rental_rate(rate numeric)
RETURNS TEXT
LANGUAGE plpgsql
AS
$$
DECLARE category TEXT;
BEGIN
category := CASE
WHEN rate <= 1.99 THEN 'Low'
WHEN rate BETWEEN 2.00 AND 3.99 THEN 'Medium'
ELSE 'High'
END;
RETURN category;
END;
$$

------
CREATE OR REPLACE FUNCTION days_since_rental(rental_date TIMESTAMP)
RETURNS INTEGER
LANGUAGE plpgsql
AS
$$
BEGIN
--using date_part for difference in days between fixed static timestamp and rental_date
RETURN DATE_PART('day', '2006-02-14 00:00:00'::TIMESTAMP - rental_date);
END;
$$
```

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

```
CREATE TABLE detailed_film_data (
film_id int PRIMARY KEY,
title VARCHAR(100),
rental_rate NUMERIC(4,2),
total_rentals int,
rental_date TIMESTAMP DEFAULT '2004-01-01 00:00:00',
flagged_performance BOOLEAN );

CREATE TABLE summary_film_data (
film_id int PRIMARY KEY,
```

```
title VARCHAR(100) NOT NULL,
price_category VARCHAR(7) CHECK (price_category IN ('Low', 'Medium', 'High')),
total_rentals int NOT NULL,
days_not_rented INT NOT NULL );
```

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.

```
INSERT INTO detailed_film_data
SELECT
f.film_id,
f.title,
f.rental_rate,
COUNT(r.rental_id) AS total_rentals,
MAX(r.rental_date) as rental_date,
CASE
WHEN COUNT(r.rental_id) < 5
AND      ('2006-02-04 15:16:02'::TIMESTAMP - MAX(rental_date) > INTERVAL '180')
THEN TRUE
ELSE FALSE
END AS flagged_performance
FROM  film f
LEFT JOIN
inventory i ON f.film_id = i.film_id
LEFT JOIN
rental r ON  i.inventory_id = r.inventory_id
GROUP BY
f.film_id, f.title, f.rental_rate;
```

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.

```
CREATE OR REPLACE FUNCTION update_summary_film_data()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM summary_film_data;
INSERT INTO summary_film_data
SELECT
film_id,
title,
```

```
categorize_rental_rate(rental_rate) AS price_category,
total_rentals,
COALESCE(days_since_rental(rental_date), 180) AS days_not_rented
FROM detailed_film_data
WHERE flagged_performance = TRUE;
RETURN NEW;
END;
$$
```

```
--create trigger

CREATE TRIGGER trigger_update_summary

AFTER INSERT OR UPDATE OR DELETE ON detailed_film_data

FOR EACH STATEMENT

EXECUTE PROCEDURE update_summary_film_data();
```

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.

```
CREATE OR REPLACE PROCEDURE refresh_film_tables()
LANGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM detailed_film_data;
DELETE FROM detailed_film_data;
INSERT INTO detailed_film_data
SELECT
f.film_id,
f.title,
f.rental_rate,
COUNT(r.rental_id) AS total_rentals,
MAX(r.rental_date) as rental_date,
CASE
WHEN (COUNT(r.rental_id)) <= 5
AND('2006-02-04 15:16:02'::TIMESTAMP - MAX(rental_date) > INTERVAL '180')
OR MAX(r.rental_date) IS NULL
```

```
THEN TRUE
ELSE FALSE
END AS flagged_performance
FROM  film f
LEFT JOIN
inventory i ON f.film_id = i.film_id
LEFT JOIN
rental r ON  i.inventory_id = r.inventory_id
GROUP BY  f.film_id, f.title, f.rental_rate;
END;
$$
CALL refresh_film_tables();
```

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

After reviewing features available in pgAdmin, and searching the internet… I quickly found that pgAgent is a job schedulding tool that is made specifically for PostgreSQL. This tool can automate stored procedures! This automation can be done at different times of the day and simply runs automatically. This was the first thing that came up in the help section of pgAdmin, and the first tool that popped up for a google search. This is the relevant job scheduling tool for automating our store procedures.

Reference: pgAdmin Development Team. (n.d.). *pgAgent Jobs*. Retrieved from https://www.pgadmin.org/docs/pgadmin4/development/pgagent_jobs.html