Moises Mercado

D326 Advanced Data Management PA

6/2/2024

1. Summarize onereal-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment:

A common issue I have seen in a business can sometimes be not enough inventory or an oversupply of inventory. I will Analyze what are the top 5 most rented films in each category. This way we can identify customer preferences and guide inventory decisions. This information is crucial to identify to ensure we are not oversupplying or undersupplying on some specific films.

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

Following fields will be included in the *summary table*:

* category\_name (VARCHAR (25))
* top\_1\_film (VARCHAR (255))
* top\_2\_film (VARCHAR (255))
* top\_3\_film (VARCHAR (255))
* top\_4\_film (VARCHAR (255))
* top\_5\_film (VARCHAR (255))

Following fields will be included in the *detailed* table:

* film\_id (INT/PRIMARY KEY)
* title (VARCHAR (255))
* category\_id (INT/)
* category\_name (VARCHAR (25))
* rental\_count (INT)

1. Describe the types of data fields used for the report:

Data types include the following: VARCHAR, and INTEGER.

1. 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.

The tables from the dataset that will provide necessary data are film, category, film\_category, and rental.

1. 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).

A custom transformation might be needed to convert category\_id to category\_name using a user-defined function for easier data interpretation. This function will also ensure that it retrieves the category name from the Category table using the provided category ID, enhancing data retrieval accuracy and reliability.

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

The *detailed table* can be used for granular analysis, such as seeing rental count for each film/category. The *summary table* provides a direct answer to the business question, showing the top 5 films in each category as well as total rental count for each category.

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

So as new data is inserted in the detailed table, summary table will also be updated. The procedure refresh\_film\_data() is designed to revert tables back to their original state. This can be useful if data is lost, corrupted, or has significant errors, and this procedure should only be called if we run into the issue as previously stated.

Code Starts here:

**--Section B: Perform a transformation(s) with a user defined function**

CREATE OR REPLACE FUNCTION get\_category\_name (category\_id\_arg INT) RETURNS VARCHAR AS $$

DECLARE

Category\_name\_var VARCHAR;

BEGIN

SELECT name INTO category\_name\_var FROM category WHERE category\_id = category\_id\_arg;

RETURN category\_name\_var;

END;

$$ LANGUAGE plpgsql;

**--Test function:**

SELECT get\_category\_name(6) AS category\_name;

**--drop tables if exist**

DROP TABLE IF EXISTS film\_rental\_details;

DROP TABLE IF EXISTS category\_top\_films\_summary;

**--Section C: Create a Detailed and Summary Table**

CREATE TABLE category\_top\_films\_summary (

category\_name VARCHAR(25) UNIQUE,

top\_1\_film VARCHAR(255),

top\_2\_film VARCHAR(255),

top\_3\_film VARCHAR(255),

top\_4\_film VARCHAR(255),

top\_5\_film VARCHAR(255),

total\_category\_rental\_count INT

);

CREATE TABLE film\_rental\_details (

film\_id INT PRIMARY KEY,

title VARCHAR(255),

category\_id INT,

category\_name VARCHAR(25),

rental\_count INT

);

**--Test to ensure tables are created successfully**

SELECT \*FROM category\_top\_films\_summary;

SELECT \*FROM film\_rental\_details;

**--Section D: Provide query in a text format that will extract the raw data needed for the detailed section.**

INSERT INTO film\_rental\_details (film\_id, title, category\_id, category\_name, rental\_count)

SELECT f.film\_id, f.title, fc.category\_id, c.name AS category\_name, COUNT(r.rental\_id) AS rental\_count

FROM film f

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

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, fc.category\_id, c.name

ORDER BY fc.category\_id, rental\_count DESC;

**--quick note: Section E is executed before section D, the reason being is because in the trigger function whenever data is inserted/updated/deleted from detailed table. Summary table is then updated with the latest changes. This way summary table is always accurate and up-to-date.**

**--Section E: Create a Trigger on 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()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

WITH ranked\_films AS (

SELECT

category\_name,

title,

ROW\_NUMBER() OVER (PARTITION BY category\_name ORDER BY rental\_count DESC) AS rank

FROM film\_rental\_details

) , top\_films AS (

SELECT

category\_name,

MAX(CASE WHEN rank = 1 THEN title END) AS top\_1\_film,

MAX(CASE WHEN rank = 2 THEN title END) AS top\_2\_film,

MAX(CASE WHEN rank = 3 THEN title END) AS top\_3\_film,

MAX(CASE WHEN rank = 4 THEN title END) AS top\_4\_film,

MAX(CASE WHEN rank = 5 THEN title END) AS top\_5\_film

FROM ranked\_films

GROUP BY category\_name

), total\_rentals AS (

SELECT

category\_name,

SUM(rental\_count) AS total\_category\_rental\_count

FROM film\_rental\_details

GROUP BY category\_name

)

INSERT INTO category\_top\_films\_summary ( category\_name, top\_1\_film, top\_2\_film, top\_3\_film, top\_4\_film, top\_5\_film, total\_category\_rental\_count)

SELECT tf.category\_name, tf.top\_1\_film, tf.top\_2\_film, tf.top\_3\_film, tf.top\_4\_film, tf.top\_5\_film, tr.total\_category\_rental\_count

FROM top\_films tf

JOIN total\_rentals tr ON tf.category\_name = tr.category\_name

ON CONFLICT (category\_name) DO UPDATE

SET top\_1\_film = EXCLUDED.top\_1\_film,

top\_2\_film = EXCLUDED.top\_2\_film,

top\_3\_film = EXCLUDED.top\_3\_film,

top\_4\_film = EXCLUDED.top\_4\_film,

top\_5\_film = EXCLUDED.top\_5\_film,

total\_category\_rental\_count = EXCLUDED.total\_category\_rental\_count;

RETURN NULL;

END;

$$;

**--CREATE TRIGGER STATEMENT**

CREATE TRIGGER update\_summary

AFTER INSERT OR UPDATE OR DELETE ON film\_rental\_details

FOR EACH ROW

EXECUTE FUNCTION update\_summary();

**--Test to ensure tables are populated with correct information**

SELECT \*FROM film\_rental\_details;--1000rows

SELECT \* FROM category\_top\_films\_summary;--top\_1\_film = Zorro ARK in Comedy category

**--ADD extra row to detailed table**

INSERT INTO film\_rental\_details (film\_id, title, category\_id, category\_name, rental\_count)

VALUES (1001, 'The Fall Guy', 5, 'Comedy', 40);

**--After inserting into details table, it should have 1001 rows and on summary table the Fall Guy will be the top\_1\_film in comedy**

**--SECTION F: Stored procedure to refresh data in both the detailed table and the summary table**

CREATE OR REPLACE PROCEDURE refresh\_film\_data()

LANGUAGE plpgsql

AS $$

BEGIN

--Clear existing data and reset auto-increment counters

DELETE FROM film\_rental\_details;

DELETE FROM category\_top\_films\_summary;

--Inserting raw data extraction logic from part D

INSERT INTO film\_rental\_details (film\_id, title, category\_id, category\_name, rental\_count)

SELECT f.film\_id, f.title, fc.category\_id, c.name AS category\_name, COUNT(r.rental\_id) AS rental\_count

FROM film f

JOIN film\_category fc ON f.film\_id =fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

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, fc.category\_id, c.name

ORDER BY fc.category\_id, rental\_count DESC;

END;

$$;

CALL refresh\_film\_data();

**--F1:   Identify a relevant job scheduling tool that can be used to automate the stored procedure:**

**--Extension pg\_cron is an extension for PostgreSQL that allows you to schedule SQL commands to run at specified times or intervals directly within the database. This is great because it integrates seamlessly with PostgreSQL and can be used to automate the execution of the ‘refresh\_film\_data()’ stored procedure.**

**References:**

Dias, H. (2020) *An Overview of Job Scheduling Tools for PostgreSQL*, *severalnines*. Available at: https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/ (Accessed: 03 July 2024).

Malik, U., Goldwasser, M. and Johnston, B. (2019) *SQL for data analytics: Perform fast and efficient data analysis with the power of SQL*. Birmingham: Packt Publishing, Limited. (Accessed: 03 July 2024).