Advanced Data Management - D326 Performance Assessment

Lauren Cobb
Western Governors University

8/18/2025

A.) The purpose of this business report is to find which genre of film generates the most revenue from rental payments in a month. This would provide insight into what films the stores should supply in order to better serve their customer base and generate income for the business.

A1.) The detailed table will use the following fields:
payment_id INT,
payment_date TEXT
rental_amount NUMERIC(10,2)
category_name VARCHAR(25)

The summary table will use the following fields:
payment_month TEXT
rental_amount NUMERIC(10,2)
category_name VARCHAR(25)

A2.) The data types used will be:
INTEGER – used for the payment_id field to list the identifying number of the transactions
TEXT – used in the payment_date and payment_month fields to list out the month the payment was made
NUMERIC – used for the rental_amount fields to display the price paid for the rental of a DVD
VARCHAR – used in the category_name to display the genre of a film

A3.) For the detailed table, I will be using the category, film_category, inventory, rental, and payment tables.
For the summary table, I will be using the detailed table, detailed_rentals.

A5.) The detailed table can be utilized to see the total number of transactions in a month and the individual cost of the rentals.

The summary table provides the stakeholders with a total sum of the revenue earned from each film category for the month.

A6.) The report is meant to show the monthly revenue generated by each film category. With this in mind, the report should be run monthly to provide the relevant information to the stakeholders. I would suggest running the report on the last day of the month, after the stores have closed and the final sales numbers are input into the database.

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

-- This function transforms the sales_month data type from timestamp to text and displays the name of the month.

```
CREATE OR REPLACE FUNCTION sales_month (payment_date timestamp)
        RETURNS text
        LANGUAGE plpgsql
AS
$$
DECLARE month_of_sale text;
BEGIN
        SELECT to_char( payment_date, 'Month') into month_of_sale;
        RETURN month_of_sale;
END;
$$
```

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

-- Code to create detailed table

```
CREATE TABLE detailed_rentals(
payment_id INT,
payment_date TEXT,
rental_amount NUMERIC(10,2),
category_name VARCHAR(25),
PRIMARY KEY (payment_id)
);
```

-- Code to create summary table

```
CREATE TABLE summary_rentals(
payment_month TEXT,
rental_amount NUMERIC(10,2),
category_name VARCHAR(25)
```

);

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.

-- Inserts data from dvdrental database into the detailed table

```
INSERT INTO detailed_rentals
SELECT
p.payment_id,
sales_month(p.payment_date),
p.amount,
c.name
FROM category c
INNER JOIN film_category fc ON c.category_id = fc.category_id
INNER JOIN inventory i ON fc.film_id = i.film_id
INNER JOIN rental r ON i.inventory_id = r.inventory_id
INNER JOIN payment p ON r.rental_id = p.rental_id
WHERE EXTRACT(YEAR FROM payment_date) = 2007
    AND EXTRACT(MONTH FROM payment_date) = 3
ORDER BY payment_id;
```

-- Inserts data from detailed_rentals into the summary table

```
INSERT INTO summary_rentals
SELECT payment_date, SUM(rental_amount), category_name
FROM detailed_rentals
GROUP BY 1,3
ORDER BY 2 DESC;
```

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.

-- Code to create function that will update the summary table when the detailed table has new inserts

```
CREATE OR REPLACE FUNCTION update_SUMMARY_table()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
BEGIN
DELETE FROM summary_rentals;
INSERT INTO SUMMARY_RENTALS
SELECT payment_date, SUM(rental_amount), category_name
FROM detailed_rentals
GROUP BY 1, 3
ORDER BY 2 DESC;
RETURN NEW;
END;
$$;
```

-- Code to create trigger that executes function to refresh summary table with data from detailed table.

```
CREATE TRIGGER new_summary_rentals
        AFTER INSERT
        ON detailed_rentals
        FOR EACH STATEMENT
        EXECUTE PROCEDURE update_SUMMARY_table();
```

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.

-- Code to create stored procedure that will update both the detailed and summary tables.

```
CREATE OR REPLACE PROCEDURE refresh_rental_tables()
LANGUAGE PLPGSQL
AS $$
BEGIN
DELETE FROM detailed_rentals;
DELETE FROM summary_rentals;
INSERT INTO detailed_rentals
SELECT
p.payment_id,
sales_month(p.payment_date),
p.amount,
c.name
FROM category c
INNER JOIN film_category fc ON c.category_id = fc.category_id
INNER JOIN inventory i ON fc.film_id = i.film_id
```

```
INNER JOIN rental r ON i.inventory_id = r.inventory_id
INNER JOIN payment p ON r.rental_id = p.rental_id
WHERE EXTRACT(YEAR FROM payment_date) = 2007
    AND EXTRACT(MONTH FROM payment_date) = 3
ORDER BY payment_id;

RETURN;
END;
$$;
```

F1.) I would suggest utilizing the pgAgent job scheduling tool in PostgreSQL. With this tool, one could run the stored procedures once a month to update the tables.

G.
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=79f2bea0-32c7-4db9-89b8-b33d012f8174

H. The only source used was the PostgreSQL Tutorial (https://neon.com/postgresql/tutorial) that was linked in the course resources. This was used when creating the function to transform data and for triggers.