D191 Performance Assessment

PostgreSQL

Ruben Huerta
WESTERN GOVERNORS UNIVERSITY

Business Question:

What are our best-selling movies?

This project is about trying to figure out what movies are the money makers for my DVD movie rental business, Lockbuster. The business will benefit from my reports because they identify the best-selling movies in our inventory and other beneficial information. These reports will help in gaining more customers, increase revenue and reduce costs.

PART A

1. Describe the data used for the report.

```
CREATE TABLE movie_rental_sales AS

SELECT i.film_id, f.title, CASE l.language_id

WHEN 1 THEN 'Yes' ELSE 'No' END AS is_english, r.rental_id, r.customer_id,
p.payment_id, p.amount, p.payment_date

FROM rental r

LEFT JOIN payment p

ON r.rental_id=p.rental_id

LEFT JOIN inventory i

ON i.inventory_id=r.inventory_id
```

```
LEFT JOIN film f

ON f.film_id=i.film_id

LEFT JOIN language l

ON l.language_id=f.language_id

WHERE p.payment_id IS NOT NULL;



CREATE TABLE movie_sales_summary AS

SELECT title, SUM(amount) as sales, is_english

FROM movie_rental_sales

GROUP BY title, is_english
ORDER BY sales DESC;
```

The value data types for my used columns are inherited from their original tables:

i.film_id: smallint **f.title:** charactervarying **l.language_id**: int

is_english: string (transformation) **r.rental_id:** int **r.customer_id:** smallint

p.payment_id: int **p.amount:** numeric(5,2)

p.payment_date: timestamp without time zone

2. Identify **two** or more specific tables from the given dataset that will provide the data necessary for the detailed and the summary sections of the report.

   Table List:

   Inventory, Film, Language, Rental, and Payment

3. Identify the specific fields that will be included in the detailed and the summary sections of the report.

   Detailed Table Fields:

   film_id, title, language_id, is_english, rental_id, customer_id, payment_id, payment_amount, payment_date

   Summary Table Fields:

   title, sales, is_english

4. Identify **one** field in the detailed section that will require a custom transformation and explain why it should be transformed. For example, you might translate a field with a value of ‘N’ to ‘No’ and ‘Y’ to ‘Yes’.

   I chose to transform language {1 -> ‘Yes’ English | NOT 1 -> ‘No’ not English} because most of the movies, especially the most relevant to my reports are in English. Having this column could help in visualizing how much inventory we actually need to maintain in foreign movies.

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

Detailed Report:

The detailed report can be used in some of the following business use cases: figuring out the best-selling movie, worst selling movie, and help to identify any movies we should have more or less of. It can also help to figure out our most frequent customers, customers who have spent the most money, which would both based be on customer IDs.

Summary Report:

The summary report's business use case is to show which movies bring the business the most money. It could also show which ones make the least money if ordered differently.

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

   Stakeholders typically prefer to see quarterly reports that show them how their investment is doing as it may fluctuate over weeks or a month. Most companies report to their stakeholders quarterly and then provide a reflection and a way forward at the end of the fiscal year to reassure stakeholders.

   In order to maintain relevant and fresh data, the tables should be refreshed ever quarter of the financial fiscal year. This way there is less reporting to look back on when end-of-year comes around and so data isn't skewed. For example, if we submit a report to stakeholders at the end of January (1 month), but a snow storm hit us for a couple of weeks in the month then we will likely show poor margins because of circumstances out of our control. The poor weather, however, could bounce sales the following week because everyone wants to get out or grab a good movie to get comfortable at home with as the snow melts. The timing of these reports is important to maintain happy investors.

Part B – Creating Tables

```
SELECT i.film_id, f.title, CASE l.language_id

WHEN 1 THEN 'Yes' ELSE 'No' END AS is_english, r.rental_id,
r.customer_id, p.payment_id, p.amount, p.payment_date

INTO movie_rental_sales

FROM rental r

LEFT JOIN payment p

ON r.rental_id=p.rental_id

LEFT JOIN inventory i

ON i.inventory_id=r.inventory_id

LEFT JOIN film f

ON f.film_id=i.film_id

LEFT JOIN language l

ON l.language_id=f.language_id
WHERE p.payment_id IS NOT NULL;

SELECT title, SUM(amount) as sales, is_english

INTO movie_rentals_summary

FROM movie_rental_sales

GROUP BY title, is_english
ORDER BY sales DESC;
```

Part C – Extracting Raw Data for Detailed Section and Verifying Data Accuracy

```
SELECT i.film_id, f.title, CASE l.language_id
WHEN 1 THEN 'Yes' ELSE 'No' END AS is_english, r.rental_id, r.customer_id, p.payment_id, p.amount, p.payment_date
INTO movie_rental_sales
FROM rental r
LEFT JOIN payment p
ON r.rental_id=p.rental_id
LEFT JOIN inventory i
ON i.inventory_id=r.inventory_id
LEFT JOIN film f
ON f.film_id=i.film_id
LEFT JOIN language l
ON l.language_id=f.language_id
WHERE p.payment_id IS NOT NULL;
```

Part C – Continued

```
SELECT payment_id FROM movie_rental_sales
WHERE payment_id IS NOT NULL
EXCEPT
SELECT payment_id FROM payment
WHERE payment_id IS NOT NULL;
```

If nothing populates, that means our data from movie_rental_sales is the same as the data in payment.

```
SELECT payment_id FROM movie_rental_sales
EXCEPT
SELECT payment_id FROM payment
WHERE payment_id IS NULL;
```

In order to test this, we can use the above code and receive table values from movie_rental_sales because the tables have mismatching data now that only one table includes NULL values and the other does not. The code in movie_rental_sales excludes NULL from its creation.

Part D – Transformation Function

```
CREATE OR REPLACE FUNCTION is_movie_english(input_var INT)
RETURNS varchar(3)
LANGUAGE plpgsql
AS $$
DECLARE output_var varchar(3);
BEGIN
        SELECT CASE WHEN input_var = 1 THEN 'Yes'
        ELSE 'No'
        END
INTO output_var;
RETURN output_var;
END;
$$;
```

Test with: SELECT is_movie_english();

Part E – Trigger That "Cascades" Updates from the Detailed Table to the Summary Table

```
CREATE OR REPLACE FUNCTION alpha_trigger_function()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
DELETE FROM movie_rentals_summary;
INSERT INTO movie_rentals_summary
SELECT title, SUM(amount) as sales, is_english
FROM movie_rental_sales
GROUP BY title, is_english
ORDER BY sales DESC;
RETURN NULL;
END;
$$;
CREATE TRIGGER new_rental_sale
AFTER INSERT
ON movie_rental_sales
FOR EACH STATEMENT
EXECUTE PROCEDURE alpha_trigger_function();
```

Test with: INSERT INTO movie_rental_sales (film_id, title, language_id, rental_id, customer_id, payment_id, amount)

VALUES (9999, 'Passing D191', 99, 9999, 9999, 9999, 9.99)

DROP TRIGGER new_rental_sale ON movie_rental_sales CASCADE;

Part F- Stored Procedure to Refresh Data in Both the Detailed and Summary Tables

```
CREATE OR REPLACE PROCEDURE refresh_tables()
LANGUAGE plpgsql
AS $$
BEGIN
DELETE FROM movie_rental_sales;
DELETE FROM movie_rentals_summary;

INSERT INTO movie_rental_sales
SELECT i.film_id, f.title, CASE l.language_id
WHEN 1 THEN 'Yes' ELSE 'No' END AS is_english, r.rental_id, r.customer_id,
p.payment_id, p.amount, p.payment_date
FROM rental r
LEFT JOIN payment p
ON r.rental_id=p.rental_id
LEFT JOIN inventory i
ON i.inventory_id=r.inventory_id
LEFT JOIN film f
ON f.film_id=i.film_id
LEFT JOIN language l
ON l.language_id=f.language_id
WHERE p.payment_id IS NOT NULL;

INSERT INTO movie_rentals_summary
SELECT title, SUM(amount) as sales, is_english
FROM movie_rental_sales
GROUP BY title, is_english
ORDER BY sales DESC;

RETURN;
END;
$$;
```

Test with:

```
SELECT COUNT(*) FROM movie_rental_sales;
DELETE FROM movie_rental_sales WHERE film_id BETWEEN 100 AND 400;
CALL refresh_tables();
```

F1 – Explain the How of the Stored Procedure Schedule

In order to call my stored procedure on a scheduled basis I would use a Linux crontab. With this method, I would schedule my cronjob to call my stored procedure every 3 months or rather every quarter. The reasoning is answered more in-depth in section **A6**. However, to summarize my reasoning, I chose this frequency in order to not spam stakeholders with reports, it is typical of financial reports to be released quarterly, and there are less reports to consolidate at an end-of-year meeting.

References

SQL Tutorial. (n.d.). W3Schools. https://www.w3schools.com/sql/

PostgreSQL Sample Database. (2022, August 28). PostgreSQL Tutorial. https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/

PostgreSQL: Documentation. (n.d.). The PostgreSQL Global Development Group. https://www.postgresql.org/docs/