D326

Advanced Data Management
Performance Assessment

Hillary Wolfenbarger
Western Governors University

A. Business report summary

This report is intended to determine the monthly revenue for each store. It includes a comprehensive table that lists all the rental sales made between February and May in both stores, as well as a summary table that shows the total sales for each month. By analyzing these tables, stakeholders can identify the month with the highest revenue for both stores and note any trends.

1. Specific fields that will be included in the detailed table:

rental_id (INT), store_id (INT), payment_date (VARCHAR 25), payment_amount (numeric 7,2).

Specific fields that will be included in the summary table:

store_id (INT), payment_month (VARCHAR 25), total_revenue (numeric10,2).

2. Types of data fields used for the report:

The data types present in both tables are integer, varchar, and numeric.

3. Tables where data was pulled from:

The data used to populate the detailed report was pulled from the Payment and Staff tables. The summarized report was populated from the detailed table.

4. One field in the detailed table that was transformed by a user-defined function:

The payment_date column was in the timestamp format "YYYY-MM-DD." The user-defined function extracted the month and converted it to text.

5. The different business uses of both reports:

The comprehensive report provides a breakdown of all the sales made in both stores over a period of four months. This information can be utilized to identify repeat customers, pinpoint high sales periods throughout the months, and identify pricing trends. It can also help determine whether the pricing of the most rented products affects the overall rental price.

The summarized report can help identify high-volume sale months in both stores, aiding in new advertising, sales, and price reductions.

6. **How frequently the report should be refreshed:**

   The reports should be refreshed monthly, so they reflect the monthly sales accurately.

B. User-defined function code:

```
CREATE OR REPLACE FUNCTION rental_by_month (rental_date TIMESTAMP)
RETURNS TEXT
LANGUAGE plpgsql
AS
$$
DECLARE
  rental_month TEXT;
BEGIN
  rental_month := TO_CHAR(rental_date, 'Month');
  RETURN rental_month;
END;
$$;
```

C. SQL code that creates the detailed and summary tables:

Detailed table:

```
CREATE TABLE detailed_rental (
  rental_id INT,
  store_id INT,
  payment_date VARCHAR(25),
  payment_amount numeric (7,2)
   );
```

Summary table:

```
CREATE TABLE summary_rental (
  store_id INT,
  payment_month VARCHAR(25),
  total_revenue numeric(10,2)
);
```

D. SQL query that will extract raw data needed for the detailed table:

```
INSERT INTO detailed_rental
SELECT p.rental_id, s.store_id, rental_by_month(p.payment_date),
p.amount
FROM payment p
JOIN rental r ON p.rental_id = r.rental_id
JOIN staff s ON r.staff_id = s.staff_id
ORDER BY 2,3,4;
```

E. SQL code that creates a trigger on the detailed table that will continually update the summary table as data is added to the detailed table:

```
CREATE OR REPLACE FUNCTION sum_trigger_function()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM summary_rental;
INSERT INTO summary_rental
```

```
SELECT store_id, payment_date, SUM(payment_amount)
FROM detailed_rental
GROUP BY store_id, payment_date
ORDER BY 1,2;
RETURN NEW;
END;
$$;
```

```
CREATE OR REPLACE TRIGGER updated_summary
AFTER INSERT
ON detailed_rental
FOR EACH STATEMENT
EXECUTE PROCEDURE sum_trigger_function();
```

F. **Original stored procedure that can refresh the data in *both* the detailed table and summary table:**

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

INSERT INTO detailed_rental
SELECT p.rental_id, s.store_id, rental_by_month(p.payment_date),
p.amount
FROM payment p
JOIN rental r ON p.rental_id = r.rental_id
JOIN staff s ON r.staff_id = s.staff_id
ORDER BY 2,3,4;

RETURN;
END;
$$;
```

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

For automating the execution of the store procedure used in this report, pgAgent would be the best choice, as PostgreSQL was used. PgAgent can be easily installed into pgAdmin, where a schedule can be set to run the procedure periodically. Since the reports are generated monthly, it is recommended to set the procedure to run once a month, preferably on the first day of the month.

G. Panopto video recording:

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c3adf88d-f8e3-4fb2-a644-b12f0132a2cd

H. Acknowledge all utilized sources:

No sources were used to support my submission.