D191 – Advanced Data Management

Taylor Fillers

Student ID 002308905

1. The business report that I picked from the attached data sets and associated dictionaries is how much money the company is making per customer. This is important to stakeholders and business owners/manager because it helps keep up with revenue and find out what customers are spending total at the business. It could also potentially benefit the customers too if the business decides to incorporate any type of loyalty rewards or bonuses for spending a certain amount of money.

A1. The data used for this report would be the payment information as well as the customer information. The payment information is needed to provide the stakeholders and business owners with the revenue amount. The customer information is needed to provide information about the customer.

A2. The customer table and the payment table will provide the data necessary for the detailed and the summary sections of the report. Both of those tables combined will give the total amount spent by the specific customer.

A3. In the detailed section of the report, the specific fields that will be included are as follows:

* Customer ID – (data type – integer) – customers unique ID is pulled from the customer table. It also references the payment table to uniquely identify the customer by their ID. It uses the customer\_id field.
* Customer First Name – (data type – string) – customers first name is pulled from the customer table. It uses the first\_name field.
* Customer Last Name – (data type – string) – customers last name is pulled from the customer table It uses the last\_name field.
* Amount – (data type - numeric) – amount paid is pulled from the payment table. It uses the amount field.

In the summary section of the report, the specific fields that will be included are as follows:

* Customer ID – (data type – integer) – customer ID is pulled from customer table. Each ID is unique so there will be no duplicates.
* Total Amount Spent – (data type –numeric) – the amount is pulled from the payment table to specify the amount spent per customer.

A4. One field in the detailed section that will require a custom transformation is the amount field. This will be aggregated using the SUM function to add up the total sum for each customer. This transformation is necessary so when we extract the data, we can get a sum for each individual customer, instead of having to add everything manually. Adding the totals manually would be very time consuming.

A5. The detailed section of the report will be used to give a broader overview of the data being extracted. It gives the full names of the customers along with their ID incase the stakeholders and business owners want to be able to identify exactly how much a specific individual spent with the store. This section can also be used if the stakeholders and business owners want to give out loyalty rewards and bonuses, since they can identify the customer by name, last name, and their unique ID.

The summary section of the report will be used to summarize the data into a simpler overview of the data being extracted. If the stakeholders and business owners don’t desire to see the customers name and last name, they can view this section to get a summary of what the business case is striving for. The main concern is the amount of money per customer, not personal details about the customer themselves.

A6. The report should be frequently refreshed once a month to remain relevant to stakeholders. The reason for this being, that the loyalty rewards and bonuses can be given out once a month based on the report. This will provide the stakeholders with an accurate report and be beneficial for the customer.

B.

-- Detailed report table

CREATE TABLE IF NOT EXISTS detailed (

customer\_id INT,

first\_name VARCHAR(45),

last\_name VARCHAR(45),

amount NUMERIC(9, 2)

);

-- Summary report table

CREATE TABLE IF NOT EXISTS summary (

customer\_id INT,

total\_amount\_spent NUMERIC(9, 2)

);

C.

-- Populating detailed report table

INSERT INTO detailed (customer\_id, first\_name, last\_name, amount)

SELECT c.customer\_id, c.first\_name, c.last\_name, p.amount

FROM payment AS p

INNER JOIN customer AS c ON c.customer\_id = p.customer\_id;

-- View the detailed table

SELECT \*

FROM detailed;

-- There are a grand total amount of 14,596 different amounts for each customer ID in the detailed table. This number matches up with the payment table, as there are 14,596 different amounts in that table also.

-- Inserts data into the summary table

INSERT INTO summary (customer\_id, total\_amount\_spent)

SELECT c.customer\_id, p.amount

FROM payment AS p

INNER JOIN customer AS c ON c.customer\_id = p.customer\_id;

D.

-- Transformation function

CREATE OR REPLACE FUNCTION sum\_transformation()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM summary;

INSERT INTO summary

(SELECT SUM(amount), customer\_id

FROM detailed

GROUP BY customer\_id);

RETURN NEW;

END; $$;

E.

-- Transformation trigger

DROP TRIGGER IF EXISTS sum\_transformation\_update ON detailed;

CREATE TRIGGER sum\_transformation\_update

AFTER UPDATE ON detailed

FOR EACH STATEMENT

EXECUTE PROCEDURE sum\_transformation();

F.

-- Creating a procedure

DROP PROCEDURE IF EXISTS refresh\_data();

CREATE PROCEDURE refresh\_data()

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM detailed;

INSERT INTO detailed (customer\_id, first\_name, last\_name, amount)

SELECT c.customer\_id, c.first\_name, c.last\_name, p.amount

FROM payment AS p

INNER JOIN customer AS c ON c.customer\_id = p.customer\_id;

END; $$;

-- This procedure refreshes the data in the detailed and summary tables. It empties the detailed table and with the insert into command, it inserts the new data into the detailed table. The trigger that I previously created will refresh the summary table.

-- Calls the procedure

CALL refresh\_data();

-- View results

SELECT \*

FROM detailed;

SELECT \*

FROM summary;

F1. The schedule I would use to automate the stored procedure and verify data freshness is pgAgent. The reason for this being that it allows for execution of stored procedures and periodically does a connection to the database to check if there are any jobs to execute (Dias, 2022).

H, I.

Dias, H. (2022, May 4). *An overview of job scheduling tools for postgresql*. Severalnines. Retrieved January 14, 2023, from https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/