Syed Khurshid
010081191
D191 – Advance Data Management

Panopto Link: Here
For this Project I am assuming the role as an owner of a DVD rental Shop. Since this is a business, the profitability and sales of the business provides an upper-level view of the business. However, due to pending financial data, I am limiting to reviewing business performance based on ***Sales made by Location***

Sales made by location
As a business owner, one of the most crucial decisions that an owner can make is to decide if the location provides a necessary stream of revenue and one of the factors that can be determined to cause that is location. Locations determine whether or not a business can be profitable or not simply by the people living at that location. So, if a location has people who are mostly in the age category who are 50 years old or older, then most likely the outcome is that they would prefer films who are classics and would prefer less Science Fiction films. This can result in decisions made to adjust the inventory of films that are less science fiction genre.

Another use of this table is to see the Sales by Film Genre and the dates. We can see which location has which Genre of Films that are being rented out. Certain dates can tell when renters are willing to rent out and with that a pattern can be determined for having appropriate inventory of films and if demand for certain Genre on certain days.

The Two tables being created are below:

Sales_location_Detailed: which provides the sales data in detailed by Genre and location where the individual sales by location which uses two fields “city.city” and “country.country” columns concatenated with spaces. The columns are taken from the City table and the Country table. The table is made from 11 tables using a JOIN statement with the main table being the “Payment” Table, the other tables that are joined are from the “Rental”, “Inventory”, ”Film”, ”Film_Category”, “Category”, “Staff”, “Store”, “Address”, “City” and “Country” Tables which have been joined using the respective Primary and Foreign keys (Please see code).
The main fields that are being utilized are the following:

- *Detail_id* which is the primary key and is a SERIAL type
- City (City Table) Concatenated with “, “ and Country (Country table) columns named as *locations.* Both City and Country are VARCHAR with 255-character limit
- *Payment_ID* (Payment Table) which is an Integer type
- *Rental_ID* (Payment Table) also an Integer type
- *Payment_Date* (Payment Table) that using CAST() has been converted to a DATE Format making it easier to read what date the payment was made instead of a TIMESTAMP format
- *Inventory_ID* (Rental Table) which is a Integer type
- Name (Category Table) which was renamed as “*Genre*” for it to be easier to identify respective film category as *Genre*. The “*Genre*” field will be VARCHAR type with 255-character limit

Syed Khurshid
010081191
D191 – Advance Data Management

- *Title* (Film Table) which is the name of the Films. It's a TEXT Type as titles can go longer than 255 characters.
- *Sale* (Payment Table) which is NUMERIC type is using the CAST() function converted to MONEY as currency format is a financial view that is easy to understand revenue to stakeholders and when presenting to view at a high level view having a money format helps in reviewing numbers clearly

Sales_Location_Summary*:* The Summary view provides the business owners a quick glance of how much revenue each location made. As a business owner I would use such a report first and compare location revenue to see under or over performing and based on that next decision to review further can be taken. All the information is pulled from the "***Sales_location_Detailed***" table summarizes the data to be presentable to the owner
This view is made up of 3 fields:

- *Summary_id* which is a serial type and is a primary key of the table
- *Locations* is a VARCHAR with 255-character limit takes in the location field from Sales_location_Detailed table
- *Sales* (Payment Table Column) is a NUMERIC type using the SUM() function location was summed up and then using the CAST() was converted to MONEY Format as the "$" makes it easier financially to identify money related transactions

In both tables the main field that is modified is the Sale(s) Field since numbers represented in Money format provide the Owner and stakeholders an easy view to differentiate between numbers that are related to Money and that are not.
The Second field that is also found in both tables is the City and Country fields that are concatenated together with a comma as a delimiter. Having the city and Country together provides the stakeholders with an idea of which city and country in case some are not aware where it is located and is easier to understand.

The Detailed table provides the data of all rental payments and transactions made by film Genre and date as well as the location. This provides the stakeholders especially the owner the ability to see which location rental is being carried out the most and the Genre. Certain locations would provide the owner an idea what the customers by location prefer to watch and accordingly inventory of certain films can then be adjusted to meet the needs of the location as well as the date provides when films are rented out the most days of the week with the date. The Summary Table provides a quick view which usually what as a business owner would want since it helps make a decision if more stores need to be added or closed to a certain location. Based on that further review of the data can then be made to take appropriate actions.

This report should be refreshed at least minimum monthly especially at the end of the month to see the performance of the previous month. This can be done using a pgAgent which is a job scheduling tool for PostgreSQL that allows the execution the "*refresh_data*" Procedure (*Found in the code chain below*) and other SQL statements or scripts. Schedules can be made on certain

Syed Khurshid
010081191
D191 – Advance Data Management
days of the month of the year and for how long which can be updated as we proceed throughout the year.
If the updated data is required on sudden request by the business owner, then the pgAgent can be made to run the Procedure with a click of a button and it will provide the updated data to the owner as per request

NOTE: SCREENSHOTS FOUND IN THE screenshot folder

Code Reference below:

```
-- Create tables
-- Here we create the detailed table
CREATE TABLE IF NOT EXISTS Sales_location_Detailed (
    detail_id SERIAL PRIMARY KEY,
    locations VARCHAR(255),
    payment_id integer,
    rental_id integer,
    payment_date timestamp,
    inventory_id integer,
    genre VARCHAR(255),
    title TEXT,
    Sale numeric
);

-- We now have created the table with its columns necessary to view the information in detail
-- With this the detailed section of the Business report is created

-- To view the table
-- SELECT * FROM Sales_location_Detailed;

-- NOW TO CREATE THE SUMMARY TABLE OF THE SALES BY LOCATION
CREATE TABLE IF NOT EXISTS Sales_Location_Summary (
    summary_ID SERIAL PRIMARY KEY,
    locations VARCHAR(255),
    Sales numeric
);

-- Now the empty table has been created
-- WE can now view the empty table
-- SELECT * FROM Sales_Location_Summary;

-- NOW WE INSERT DATA INTO THE SALES BY LOCATION DETAILED TABLE

INSERT INTO Sales_location_Detailed (
    locations, -- THIS IS A COMBINATION OF CITY AND COUNTRY TABLE
    payment_id, -- This is the Payment Table
    rental_id, -- This is from the rental table
    payment_date, -- This is from the Payment Table
    inventory_id, -- This is from the Rental Table
    genre, -- This is from the Category Table
    title, -- This is from the Film Table
    Sale -- This is from the Rental Table
)
SELECT
city.city||', '||country.country AS city_address,
payment.payment_id,
payment.rental_id,
CAST(payment.payment_date as DATE),
rental.inventory_id,
category.name AS Genre,
film.title,
CAST(payment.amount AS MONEY)
FROM payment
INNER JOIN rental ON rental.rental_id = payment.rental_id
INNER JOIN inventory ON rental.inventory_id= inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
INNER JOIN film_category ON film_category.film_id = film.film_id
INNER JOIN category ON category.category_id = film_category.category_id
INNER JOIN staff ON payment.staff_id = staff.staff_id
```

```
INNER JOIN store ON payment.staff_id = store.manager_staff_id
INNER JOIN address ON store.address_id=address.address_id
INNER JOIN city ON city.city_id =address.city_id
INNER JOIN country ON city.country_id = country.country_id
GROUP BY category.name, payment.payment_id, rental.inventory_id, city.city, country.country, film.title
ORDER BY city_address DESC;

-- This information is a combination from Rental, Inventory, Film, Film_Category, Category, Staff, Store, City and
Country Table
-- With this a combined view information is retrieved and added into the table

-- NOW WE CAN VIEW THE Sales_location_Detailed table WITH THE DATA ADDED

-- SELECT * FROM Sales_location_Detailed;

-- NOW CREATING A FUNCTION THAT WOULD UPDATE THE DATA IN THE Sales_Summary table
CREATE OR REPLACE FUNCTION summary_data_refresh()
    RETURNS TRIGGER
    AS $$
BEGIN
    -- OLD DATA NEEDS TO BE CLEARED OUT SO DATA IS FIRST CLEARED OUT
    DELETE FROM Sales_Location_Summary;
    -- NEW DATA IS THEN INSERTED
    INSERT INTO Sales_Location_Summary (
        locations,
        Sales
    )
    SELECT
    Sales_location_Detailed.locations,
    SUM(Sales_location_Detailed.sale)
    FROM Sales_location_Detailed
    GROUP BY Sales_location_Detailed.locations;
    RETURN new;
END;
$$
LANGUAGE plpgsql
-----------------------------------------------------------------
-- WE NOW CREATE THE TRIGGER FUNCTION
CREATE TRIGGER refreshing_data
AFTER INSERT ON Sales_location_Detailed
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_data_refresh();

-- PROCEDURE NEEDS TO BE CREATED TO REFRESH THE DETAILED TABLE AND THUS ALSO REFRESHING THE SUMMARY TABLE

CREATE OR REPLACE PROCEDURE refresh_data()
AS $$
BEGIN
        -- We need to renew the data so first we empty the existing data in the table
        DELETE FROM Sales_location_Detailed;
         -- WE HAVE TO RE-DO THE DATA INSERTS INTO THE DETAILED TABLE TO HAVE THE DETA PRESENTED IN THE SUMMARY TABLE
        INSERT INTO Sales_location_Detailed (
            locations, -- THIS IS A COMBINATION OF CITY AND COUNTRY TABLE
            payment_id, -- This is the Payment Table
            rental_id, -- This is from the rental table
            payment_date, -- This is from the Payment Table
            inventory_id, -- This is from the Rental Table
            genre, -- This is from the Category Table
            title, -- This is from the Film Table
            Sale -- This is from the Rental Table
        )
        SELECT
        city.city||', '||country.country AS city_address,
        payment.payment_id,
        payment.rental_id,
        CAST(payment.payment_date as DATE),
        rental.inventory_id,
        category.name AS Genre,
        film.title,
        CAST(payment.amount AS MONEY)
        FROM payment
        INNER JOIN rental ON rental.rental_id = payment.rental_id
        INNER JOIN inventory ON rental.inventory_id= inventory.inventory_id
        INNER JOIN film ON inventory.film_id = film.film_id
        INNER JOIN film_category ON film_category.film_id = film.film_id
        INNER JOIN category ON category.category_id = film_category.category_id
        INNER JOIN staff ON payment.staff_id = staff.staff_id
        INNER JOIN store ON payment.staff_id = store.manager_staff_id
```

```
        INNER JOIN address ON store.address_id=address.address_id
        INNER JOIN city ON city.city_id =address.city_id
        INNER JOIN country ON city.country_id = country.country_id
        GROUP BY category.name, payment.payment_id, rental.inventory_id, city.city, country.country, film.title
        ORDER BY city_address DESC;

END;
$$
LANGUAGE plpgsql

-- THIS PROCEDURE WILL THEN LEAD TO THE DATA BEING REFRESHED AND WILL TRIGGER THE OTHER FUNCTION

-- THIS WILL BE CALLING THE MAIN TABLE
CALL refresh_data();

-- NOW YOU CAN VIEW THE NEW DATA HERE
SELECT * FROM Sales_location_Detailed;

-- THE SUMMARY DATA CAN BE VIEWED HERE
SELECT * FROM Sales_Location_Summary;
```

Link to Panopto video:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=71a9dc50-4a4e-4299-b4b2-b0420041cd71

Note: No External References were used