D191 PERFORMANCE ASSESSMENT

Ignacio-Manuel Atilano

1. Business Application: Figure out which genres bring in the most revenue.

Detailed table – genre\_sales

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Table Derivation | Datatype | Description |
| title | film | Varying character | This field holds the title of the movie. |
| genre | category | Varying character | This field holds the genre value. |
| number\_of\_rentals | rental | Big Integer | It shows how often a movie within a particular genre has been rented. |
| payment | payment | Money | It shows the money a title has made within a particular genre. |

Summary Table

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Table Derivation | Data Type | Description |
| genre | genre\_sales | Varying character | This field holds the genre. |
| number\_of\_rentals | genre\_sales | Big Integer | It shows how many times a particular genre has been rented. |
| total\_revenue | genre\_sales | Money | It shows how much revenue has been generated from a particular genre. |

**Q**: Identify the specific fields that will be included in the detailed table and the summary table of the report.

**A**:

The data type is shown in the tables above, each with a column specifying the data type. I will also describe it here.

In the detailed table, as shown above, the variables title and genre use the varying character data type, which ensures a precise representation of movie titles and genres. The number of rentals variable uses a big integer data type, providing an accurate rental count. Finally, the payment uses the money data type, ensuring exact revenue representation.

Next, we have the summary table with a total of three fields. Firstly, the genre variable also employs a varying character datatype. Then, we have the number of rentals using a big integer data type. Finally, the total\_revenue variable uses the money data type.

1. **Q**: Identify at least two specific tables from the given data set that will provide the data necessary for the report's detailed and summary table sections.

**A**:

The tables used to make the detailed table include the rental table, payment table, film table, film\_category table, and the category table. As a quick side note, there are no fields used from the film\_category table; however, it does have an associative relationship between the film table and the category table, which means if I wanted a field from the category table, I would need to go through film then the film\_category table to access the category table.

The summary table is called genre\_sales\_summary, which derives from the genre\_sales or the detailed table. However, assuming my code did not employ my detailed table, and I had to use a select query to populate my table manually, I would use the same tables as the detailed table, which includes the rental table, payment table, film table, film\_category table, and category table.

Note: You can see what fields are derived from the tables in A.1 above.

1. **Q**: Identify at least **one** field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

**A**:

The fields requiring a custom transformation are the number\_of\_rentals and payment fields within the trigger function. The number\_of\_rentals variable is used in the detailed table with an aggregate function to calculate the total rentals for a particular movie in a genre. The payment field sums the revenue a particular genre made. The previously mentioned query will show users what genre is most profitable in the detailed table.

1. **Q**: Explain the different business uses of the detailed table section and the summary table section of the report.

**A**: The detailed table's business case gives readers an idea of movies from which genres make the most money. The business case the summary table caters to is showing C-Suite executives’ vital information in a concise format of what genres make the most revenue.

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

**CREATE** **OR** **REPLACE** **FUNCTION** insert\_trigger\_genre\_sale()

**RETURNS** **TRIGGER**

**LANGUAGE** PLPGSQL

**AS** $$

**BEGIN**

-- We want to update our large sales by channel table, and the easiest way is to delete the table and reinsert all + new data

-- Clear data from this table

**DELETE** **FROM** genre\_sales\_summary;

**INSERT** **INTO** genre\_sales\_summary

**SELECT** genre,

**COUNT**(number\_of\_rentals) **AS** number\_of\_rentals,

**SUM**(payment) **AS** total\_revenue

**FROM** genre\_sales

**GROUP** **BY** **1**

**ORDER** **BY** **3** **DESC**

**LIMIT** **5**;

**RETURN** **NEW**;

**COMMIT**;

**END**;

$$;

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

Detailed table – genre\_sales

**CREATE** **TABLE** IF **NOT** **EXISTS** genre\_sales(

title VARCHAR(**50**),

genre VARCHAR(**50**),

number\_of\_rentals BIGINT,

payment MONEY );

Summary Table – genre\_sales\_summary

**CREATE** **TABLE** IF **NOT** **EXISTS**

genre\_sales\_summary(

genre VARCHAR(**50**),

number\_of\_rentals BIGINT,

total\_revenue MONEY);

1. 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.

Detailed Section SQL Code

**SELECT**

film.title,

category.name **AS** genre,

**COUNT**(rental.rental\_id) **AS**

number\_of\_rentals,

(**SUM**(payment.amount))::NUMERIC(**5**,**2**)::MONEY

**AS** total\_revenue

**FROM** inventory

-- link (inventory) to (rental) table

**LEFT** **JOIN** rental **ON** rental.inventory\_id = inventory.inventory\_id

-- link (rental) to (payment) table

**LEFT** **JOIN** payment **ON** payment.rental\_id = rental.rental\_id

-- get genres through the following tables

-- inventory -> film -> film\_category -> category

-- link (inventory) to (film)

**LEFT** **JOIN** film **ON** film.film\_id = inventory.film\_id

-- link (film) to (film\_category)

**LEFT** **JOIN** film\_category **ON** film\_category.film\_id = film.film\_id

-- link (film\_category) to (category)

**LEFT** **JOIN** category

**ON** category.category\_id = film\_category.category\_id

**GROUP** **BY**

film.title,

category.name

**ORDER** **BY** genre, total\_revenue **DESC**;

1. 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.

**CREATE** **TRIGGER** genre\_sales\_summary\_insert\_trigger

**AFTER** **INSERT** **OR** **DELETE**

**ON** genre\_sales

**FOR** **EACH** **STATEMENT**

**EXECUTE** **PROCEDURE** insert\_trigger\_genre\_sale();

1. Provide an original stored procedure in a text format that can be used to refresh the data in both the detailed and summary tables. The procedure should clear the contents of the detailed and summary tables and perform the raw data extraction from part D.

**A**: Just a quick note: I only insert data into the genre\_sales table since the summary table has a trigger that is automatically updated according to the genre\_sales or detailed table.

**CREATE** **OR** **REPLACE** **PROCEDURE** refresh\_genre\_tables()

**LANGUAGE** PLPGSQL

**AS** $$

**BEGIN**

**DELETE** **FROM** genre\_sales;

**DELETE** **FROM** genre\_sales\_summary;

-- refresh genre\_sales table FIRST

**INSERT** **INTO** genre\_sales

**SELECT**

film.title,

category.name **AS** genre,

**COUNT**(rental.rental\_id) **AS** number\_of\_rentals,

(**SUM**(payment.amount))::NUMERIC(**5**,**2**)::MONEY **AS** total\_revenue

**FROM** inventory

-- link (inventory) to (rental) table

**LEFT** **JOIN** rental **ON** rental.inventory\_id = inventory.inventory\_id

-- link (rental) to (payment) table

**LEFT** **JOIN** payment **ON** payment.rental\_id = rental.rental\_id

-- get genres through the following tables

-- inventory -> film -> film\_category -> category

-- link (inventory) to (film)

**LEFT** **JOIN** film **ON** film.film\_id = inventory.film\_id

-- link (film) to (film\_category)

**LEFT** **JOIN** film\_category **ON** film\_category.film\_id = film.film\_id

-- link (film\_category) to (category)

**LEFT** **JOIN** category

**ON** category.category\_id = film\_category.category\_id

**GROUP** **BY**

film.title,

category.name

**ORDER** **BY** genre, total\_revenue **DESC**;

**RETURN**;

**END**;

$$;

1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

The best scheduling tool to use would be pg\_cron. The previous software offers several advantages. Firstly, it is platform-independent because it only interacts with a PostgreSQL database. Other scheduling tools, such as Workflows and Redwood, work strictly with GCP or Google Cloud Platform; however, Redwood can work with other cloud solutions, but we only need to work with the database for what we need. Secondly, it is lightweight. We only need to refresh the detailed and summary tables based on a frequency we will discuss shortly. Scheduling tools like pgAgent are excellent, offering functionality to interact with events and triggers. However, pg\_cron is more lightweight, and while pgAgent offers more functionality, it would require more resource usage. Thus, the cost would outweigh the benefit. Finally, pg\_cron is free. Overall, pg\_cron would be the best choice for automating the stored procedure.

Since there are locations throughout the United States, the earliest time zone to consider is HST, and the latest time zone to consider would be the East Coast or EST. Keeping in mind that store hours are from 2:00 PM to 7:00 PM, the total time the servers will be inserting data will be a range of 10 hours, 2 PM in HST to 7 PM EST. Since the data set does not deal with more than 10,000 records at a time, the procedure should run every hour to ensure data freshness.