Part A:

With data constantly being added, it’s a constant struggle to maintain the data without someone to constantly maintain it. However, not all problems or questions can be solved instantly. After looking through the data, I wanted to answer the question of “How much revenue does the business bring in per a DVD category?”. To do so, I created two tables to manipulate data in. The first one called ‘details’ and the second called ‘summary’. ‘Details’ contains all the necessary columns that connect the data together. With this table, I can pull all the data from one place and layout every row that is necessary to answer the initial question. The ‘details’ table contains all detailed pieces of information, while the ‘summary’ table displays a concise format of manipulated data. The ‘details’ table will take columns from ‘rental, inventory, film, film\_category and payment’ tables. And will specifically use ‘rental\_id, inventory\_id, film\_id, category\_id and amount’ from the required tables. For the ‘summary’ table, I have used information from categories and details table. Specifically using ‘name, category\_id’ from the ‘category’ table and amount from the ‘details’ table to find the sum of each category. The main aggregate function I used was ‘sum()’. To find the total revenue for each category, I took the sum of all sales made using the ‘details.amount’ column, and grouped it by ‘category\_id’ to find the revenue for each category. This was a necessary transformation, since in all other tables it simply lists the amount, a customer paid for each DVD and doesn’t make it easy to see how much revenue each category might be bringing in. Being able to know how much revenue each category brings in, will enable the store owner to buy more trending DVDs from that category to make even more money.

Overall, the ‘details’ table should be used to add new data entries of each DVD purchased from both shops. And the ‘summary’ table will be updated when the ‘details’ table is updated so that the top earning category can always be made known to the owner.

I also recommend the owner refresh the report at the end of each quarter to show stakeholders the earnings of each category. This will keep the owner and stakeholders in check with DVD trends.

A1.

Data used in this report is from the DVD Database on ‘Labs on Demand’. It will be used to show each purchase made at both DVD stores. I will be accessing this data through a virtual machine provided by WGU and will use pgAdmin to run queries. This data will be grouped by category to distinguish each DVD categories total revenue.

A2. The necessary tables needed to create the ‘details’ table are: rental, inventory, film, film\_category and payment. The necessary tables needed to create the ‘summary’ table include category and details.

A3. The necessary fields needed for the ‘details’ table include rental\_id, inventory\_id, film\_id, category\_id and amount. The necessary fields needed for the ‘summary’ table include name, category\_id, and amount.

A4. The one field that will be transformed is ‘amount’. I will take the sum of all the amounts listed in the ‘details’ table and categorize them by category\_id. This will show us the total amount of revenue brought in by each category.

A5. The ‘details’ table will be used for pulling all important data into one table to be modified. This way the existing data will not be affected. The ‘summary’ table on the other hand will be used to show the output of all revenue brought in by each category in a pleasant format. This will make it easier for the owner to show stakeholders the data without needing to explain what each row of revenue represents.

A6. The data should be refreshed at the end of each quarter. This way this can total up all the sales for the quarter and show the owner and stakeholders what categories are trending. With this trending data they can clearly know which categories to invest more money into to get more revenue.

Part B:

Graphical user interface

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Part C: Graphical user interface, text, application

Description automatically generatedGraphical user interface, application, table

Description automatically generated

\*the data is accurete as it is providing the correct sized data (smallint, int, varchar, numeric) and matches existing data from existing tables.\*

Part D:Graphical user interface, text, application

Description automatically generated

Part E:Graphical user interface, application

Description automatically generated

Part F:

Graphical user interface, application, Word

Description automatically generated

-- STORED PROCEDURE (call at the end of every quarter) [since I was unable to capture the whole snippet of code]

CREATE PROCEDURE complete\_refresh()

LANGUAGE PLPGSQL

AS $$

BEGIN

DELETE FROM details;

DELETE FROM summary;

INSERT INTO details(

rental\_id,

inventory\_id,

film\_id,

category\_id,

amount)

SELECT rental.rental\_id, inventory.inventory\_id, film.film\_id,

payment.amount, film\_category.category\_id

FROM rental

INNER JOIN inventory ON inventory.inventory\_id = rental.inventory\_id

INNER JOIN film ON film.film\_id = inventory.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 payment ON payment.rental\_id = rental.rental\_id

GROUP BY inventory.inventory\_id, rental.rental\_id, film.film\_id, film\_category.category\_id, payment.amount

ORDER BY rental.rental\_id;

END; $$;

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

F1. The stored procedure should be called at the end of every quarter. This way the data inputted from the recent quarter can be used to show the recent quarters total revenue per a category and trends. The owner/database manager should manually call ‘complete\_refresh()’ to run the stored procedure. If the company would like to go with an automated route, they could create a script to calculate the end of each quarter followed by calling the stored procedure. They could do this by first calculating the number of months/days in each set quarter, then adding that set of numbered months/days to the previous end of quarter date. Once they have the new end of quarter date, they could compare the current day with the end of quarter date to check if ‘today’ is the end of quarter or not. If it is the end of quarter, there could be a subquery that calls (call complete\_refresh()) the stored procedure. And if the date doesn’t match the final end of quarter date, then the stored procedure will not be called.

Part G (URL):

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c090a5aa-0398-42b0-8fc6-adea014e72f4>

H/I. Sources:

PostgreSQL Documentation. 2021. SQL Commands. [online] Available at: https://www.postgresql.org/docs/11/sql-commands.html [Accessed 24 November 2021].

Bradley, Barrie. “D191 Performance Assessment Information video.” D191. Western Governors University. 25 October 2021 Presentation.