**D191 PA**

**Martin Antilley**

**ID: 010353070**

A.  Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.

The data included in the DVD rental database can be used to show how different movie genres perform compared to one another. A detailed table showing the total rentals and gross sales for each genre will be created. This detailed table will also show what percentage the rental and sales numbers represent to the total rentals and sales. Finally, by taking the percent of sales divided by the percent of rentals for each genre, we can see if certain genres create more sales with fewer rentals.

The top performing genres will be shown in a summary table. The genres will be judged on total sales, total rentals, and normalized sales. The genres with the highest value of these metrics will be featured in the summary table.

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

Detailed Table: **detailed\_table**

* genre: *varchar,* retrieved from **category**; what genre the movie is
* count\_rentals: *numeric,* retrieved from **rentals**; number of rentals per genre
* percent\_rentals: *numeric,* retrieved from **rentals**; genre specific rentals compared to total
* sum\_sales: *numeric,* retrieved from **payment**; sum of payment amounts per genre
* percent\_sales: *numeric,* retrieved from **payment**; genre specific sales compared to total
* normalized\_sales: *numeric,* retrieved from **rentals** and **payment**; ratio of percent\_sales to percent\_rentals

Summary Table: **summary\_table**

* performance\_indicator: *varchar,* determined by business; the metric that the genres will be evaluated on
* genre: *varchar,* retrieved from **detailed\_table**; what genre achieved the highest value for the performance\_indicator
* value: *numeric*, retrieved from **detailed\_table**; the highest value of the respective performance\_indicator

1. Describe the types of data fields used for the report.

The data types used are *varchar* and *numeric*. The *varchar* data type can hold a character string of varying length. The *numeric* data type can hold various types of numbers.

1. Identify at least two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The tables used to make the **detailed\_table** are:

* category: to retrieve the *name* column (renamed to *genre*)
* film\_category: used to connect **category** to **rental** and **payment**
* inventory: used to connect **category** to **rental** and **payment**
* rental: to retrieve the *rental\_id* column (to be counted)
* payment : to retrieve the *amount* column (renamed to *sales* and summed)

The **summary\_table** is created using data from the **detailed\_table**.

1. 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).

**percent\_rentals**: created with a user-defined function *percent\_of* which returns a fraction corresponding to the two values passed into it. The **count\_rentals** value is passed in with the total number of rentals across all genres.

**percent\_sales**: created with a user-defined function *percent\_of* which returns a fraction corresponding to the two values passed into it. The **sum\_sales** value is passed in with the gross amount of sales across all genres.

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

The summary table will give the business a high-level view of which genres are performing well at certain metrics. This will give a quick view to the business to help with decisions of which genres to invest in keeping on their shelves or which genres to buy more of.

The detailed table will give the business the raw sales and rental data to see how well not only the top-performing genres are doing, but every other genre as well. This will allow the business to see which genres are rented the most/least and which genres create the most/least sales. This will allow the business the ability to reassess what genre of movies they can promote to boost sales or to not buy more of.

1. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

This report should be updated every month to stay relevant as certain genres might do better at different times of the year (Romance around Valentine’s, Horror around Halloween) and the performance of genres might reflect general changing trends of consumer preferences.

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

CREATE OR REPLACE FUNCTION percent\_of(

part DECIMAL(10,4),

total DECIMAL(10,4)

)

RETURNS DECIMAL(10, 2)

LANGUAGE plpgsql

AS

$$

DECLARE percentage DECIMAL(10, 5);

BEGIN

SELECT part **/** total INTO percentage;

RETURN CAST(percentage**\***100 AS DECIMAL(10,2));

END;

$$;

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

The code below creates **detailed\_table** and **summary\_table**. The accuracy of the data was verified by manually calculating certain columns.

CREATE TABLE detailed\_table

AS

SELECT

genre,

COUNT(rental\_id) AS count\_rentals,

percent\_of(COUNT(rental\_id), (SELECT COUNT(rental\_Id) FROM raw\_data)) AS

percent\_rentals,

SUM(sales) AS sum\_sales,

percent\_of(SUM(sales), (SELECT SUM(sales) FROM raw\_data)) AS percent\_sales,

CAST(

percent\_of(SUM(sales), (SELECT SUM(sales) FROM raw\_data))

**/** percent\_of(COUNT(rental\_id), (SELECT COUNT(rental\_Id) FROM raw\_data))

AS DECIMAL(10,2)) AS normalized\_sales

FROM raw\_data

GROUP BY genre

ORDER BY normalized\_sales DESC;

CREATE TABLE summary\_table (

performance\_indicator VARCHAR,

genre VARCHAR,

val NUMERIC

);

INSERT INTO summary\_table

VALUES (

'MOST SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*sum\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'MOST RENTALS',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL),

(SELECT a.*count\_rentals*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'HIGHEST NORMALIZED SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*normalized\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL)

);

D.  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.

CREATE TABLE raw\_data

AS

SELECT

c.name as genre,

r.rental\_id,

p.amount as sales

FROM category c

LEFT JOIN film\_category f ON c.category\_id = f.category\_id

LEFT JOIN inventory i ON f.film\_id = i.film\_id

LEFT JOIN rental r ON i.inventory\_id = r.inventory\_id

LEFT JOIN payment p ON r.rental\_id = p.rental\_id

ORDER BY rental\_id;

E.  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 OR REPLACE FUNCTION trigger\_function()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM summary\_table;

INSERT INTO summary\_table

VALUES (

'MOST SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*sum\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'MOST RENTALS',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL),

(SELECT a.*count\_rentals*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'HIGHEST NORMALIZED SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*normalized\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL)

);

RETURN NEW;

END;

$$;

CREATE TRIGGER trig

AFTER INSERT

ON detailed\_table

FOR EACH STATEMENT

EXECUTE PROCEDURE trigger\_function();

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

CREATE PROCEDURE stored\_procedure()

LANGUAGE plpgsql

AS $$

BEGIN

DROP TABLE IF EXISTS raw\_data;

DROP TABLE IF EXISTS detailed\_table;

DROP TABLE IF EXISTS summary\_table;

CREATE TABLE raw\_data

AS

SELECT

c.*name* as genre,

r.*rental\_id*,

p.*amount* as sales

FROM category c

LEFT JOIN film\_category f ON c.*category\_id* **=** f.*category\_id*

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

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

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

ORDER BY rental\_id;

CREATE TABLE detailed\_table

AS

SELECT

genre,

COUNT(rental\_id) AS count\_rentals,

percent\_of(COUNT(rental\_id), (SELECT COUNT(rental\_Id) FROM raw\_data)) AS percent\_rentals,

SUM(sales) AS sum\_sales,

percent\_of(SUM(sales), (SELECT SUM(sales) FROM raw\_data)) AS percent\_sales,

CAST(

percent\_of(SUM(sales), (SELECT SUM(sales) FROM raw\_data))

**/** percent\_of(COUNT(rental\_id), (SELECT COUNT(rental\_Id) FROM raw\_data))

AS DECIMAL(10,2)) AS normalized\_sales

FROM raw\_data

GROUP BY genre

ORDER BY normalized\_sales DESC;

CREATE TABLE summary\_table (

performance\_indicator VARCHAR,

genre VARCHAR,

val NUMERIC

);

INSERT INTO summary\_table

VALUES (

'MOST SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*sum\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*sum\_sales* **<** b.*sum\_sales*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'MOST RENTALS',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL),

(SELECT a.*count\_rentals*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*count\_rentals* **<** b.*count\_rentals*

WHERE b.*genre* IS NULL)

);

INSERT INTO summary\_table

VALUES (

'HIGHEST NORMALIZED SALES',

(SELECT a.*genre*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL),

(SELECT a.*normalized\_sales*

FROM detailed\_table a

LEFT OUTER JOIN detailed\_table b

ON a.*normalized\_sales* **<** b.*normalized\_sales*

WHERE b.*genre* IS NULL)

);

RETURN;

END;

$$;

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

This stored procedure can be run monthly by using pgAdmin which allows for regular execution of stored procedures. Running the stored procedure monthly will ensure the data is up to date as trends change over time.

G.  Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the functionality of the code used for the analysis.

Link: <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=aa327db8-9267-4fb7-b59a-afbd015c9ac7>

H.  Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.

No sources were used in this submission.