A.

This report provides an in-depth analysis of monthly sales revenue by store based on the DVD Dataset. By examining the sales revenue data, this report aims to identify patterns, trends, and insights that can assist in strategic decision-making and optimizing business operations. The analysis is conducted using the DVD Dataset, which includes detailed information on sales revenue from various stores. Monthly sales revenue data for each store is aggregated and analyzed to derive meaningful insights. This versatile report can be used repeatedly throughout the year to gain insight on a multitude of various facets such as what month or season is the highest grossing, which store is most profitable, or how effective any marketing campaigns were. The analysis of monthly sales revenue by store provides valuable insights that can inform strategic decision-making and drive business growth. By leveraging these insights, the business can optimize operations, enhance customer satisfaction, and ultimately increase profit.

A1.

The detailed table includes:

* store\_id
* customer\_id
* amount
* payment\_date
* payment\_id

The summary table includes:

* store\_id
* total\_sales
* sale\_month

A2.

The detailed table will comprise of the following fields as defined.

* store\_id – a unique integer identifying individual stores
* customer\_id - a unique integer identifying individual customers
* amount – numeric that represents the value of the payment
* payment\_date – varchar of when the payment transaction took place
* payment\_id – unique integer identifier for each transaction completed

The summary table will comprise of the following fields as defined.

* store\_id – a unique integer identifying individual stores
* total\_sales – numeric that represents the sum of all transactions for a given month
* month\_of\_sales – varchar representing the month of which the total\_sales value is referring to

A3.

The customer and payment tables will be the primary source of information for both the detailed and summary tables.

A4

The date of each transaction requires custom transformation to allow for quicker and easier viewing of the data presented. The time of the transaction can be removed from the timestamp since we are looking at the data from a monthly basis. The date could also be reformatted to a more widely used format like mm/dd/yy. The date will also be transformed one more time with another function for the summary table. This transformation will take the date from the last function and return the month name which will make the summary table easier to look at.

A5.

Detailed Table Section:

Business Uses:

Transaction Analysis: The detailed table provides granular transaction-level data, including store ID, customer ID, transaction amount, transaction date, and payment ID. Businesses can use this detailed information for transactional analysis, such as understanding purchasing patterns, identifying high-value customers, and detecting any anomalies or discrepancies in transactions.

Operational Insights: Detailed transaction data can offer insights into operational efficiency, such as peak transaction days or average payment amount. This information can help optimize and improve overall operational performance.

Fraud Detection: By examining transaction details, businesses can implement fraud detection algorithms or manual reviews to identify potentially fraudulent transactions based on unusual transaction amounts or irregular transaction patterns.

Summary Table Section:

Business Uses:

Monthly Sales Performance: The summary table aggregates sales data at a higher level, providing a snapshot of monthly sales performance for each store. Key fields include store ID, total sales, and the month of sale. The business can use this summary information to track sales trends over time, compare sales performance between stores, and assess overall revenue growth.

Forecasting and Planning: Summarized monthly sales data enables businesses to forecast future sales trends and plan inventory, staffing, and marketing strategies accordingly. By identifying seasonal sales patterns and historical sales trends, the business can make informed decisions to optimize resources and maximize revenue.

Performance Evaluation: Monthly sales summaries facilitate performance evaluation at both store and company levels. Managers can assess store performance against targets, identify underperforming stores, and take corrective actions as necessary. Additionally, executives can use aggregated sales data to evaluate overall business performance, set strategic goals, and allocate resources effectively.

A6.

The report should get refreshed at minimum once a month to maintain relevant data and remain useful to the business. After a month has past the report should be ran as soon as possible to allow management the most opportunity to evaluate past performance and plan for the upcoming month ahead. Using this report once a month will give the business a valuable tool to evaluate the business and make adjustments to maximize revenue.

B.

--Format Timestamp--

CREATE OR REPLACE FUNCTION format\_timestamp(timestamp\_to\_format TIMESTAMP)

RETURNS VARCHAR AS $$

BEGIN

RETURN TO\_CHAR(timestamp\_to\_format, 'MM/DD/YY');

END;

$$

LANGUAGE plpgsql;

--Return Month String--

CREATE OR REPLACE FUNCTION format\_date(payment\_date TIMESTAMP)

RETURNS VARCHAR AS $$

BEGIN

RETURN TO\_CHAR (payment\_date, 'Month');

END;

$$

LANGUAGE plpgsql;

C.

--Detailed Table--

CREATE TABLE detailed\_table(

store\_id INT,

customer\_id INT,

amount NUMERIC,

payment\_date VARCHAR,

payment\_id INT);

--Summary Table--

CREATE TABLE summary\_table(

store\_id INT,

total\_sales NUMERIC,

sale\_month VARCHAR(20));

D.

--Detailed Data--

INSERT INTO detailed\_table (

store\_id,

customer\_id,

amount,

payment\_date,

payment\_id)

SELECT

c.store\_id,

p.customer\_id,

p.amount,

format\_timestamp(CAST(p.payment\_date AS TIMESTAMP)),

p.payment\_id

FROM

customer c

JOIN

payment p ON c.customer\_id = p.customer\_id;

E.

--Trigger--

CREATE OR REPLACE FUNCTION update\_summary\_table()

RETURNS TRIGGER

LANGUAGE plpgsql as $$

BEGIN

DELETE FROM summary\_table;

INSERT INTO summary\_table

SELECT

store\_id,

SUM(amount) AS total\_sales,

format\_date(CAST(payment\_date AS TIMESTAMP)) AS sale\_month

FROM detailed\_table

GROUP BY store\_id, sale\_month

ORDER BY sale\_month, store\_id;

RETURN NEW;

END;

$$;

--Create Trigger--

CREATE OR REPLACE TRIGGER update\_summary

AFTER INSERT OR UPDATE OR DELETE ON detailed\_table

FOR EACH STATEMENT

EXECUTE PROCEDURE update\_summary\_table();

F.

--Refresh Reports--

CREATE OR REPLACE PROCEDURE refresh\_tables()

LANGUAGE plpgsql as $$

BEGIN

DELETE FROM detailed\_table;

DELETE FROM summary\_table;

INSERT INTO detailed\_table (

store\_id,

customer\_id,

amount,

payment\_date,

payment\_id)

SELECT

c.store\_id,

p.customer\_id,

p.amount,

format\_timestamp(CAST(payment\_date AS TIMESTAMP)),

p.payment\_id

FROM

customer c

JOIN

payment p ON c.customer\_id = p.customer\_id;

RETURN;

END; $$;

F1.

For scheduling tasks such as automating stored procedures in PostgreSQL (via pgAdmin), you can consider using pgAgent. pgAgent is a job scheduler specifically designed for PostgreSQL. It is a part of the pgAdmin suite and allows you to schedule tasks, including running SQL scripts, stored procedures, or any other database maintenance tasks. By using pgAgent, you can automate the execution of your stored procedures directly within pgAdmin, providing a convenient and integrated solution for scheduling tasks in PostgreSQL.

G.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0af7beb6-a460-4137-bd4e-b16301800fef

H.

No sources were utilized to support this submission.