

# SALES PERFORMANCE ANALYSIS OF WALMART STORES USING ADVANCED MYSQL TECHNIQUES

## **Introduction:**

- Walmart, a major retail chain, operates across several cities, offering a wide range of products. The dataset
- provided contains detailed transaction data, including customer demographics, product lines, sales figures, and
- payment methods. This project will use advanced SQL techniques to uncover actionable insights into sales
- performance, customer behavior, and operational efficiencies.

## **Business Problem:**

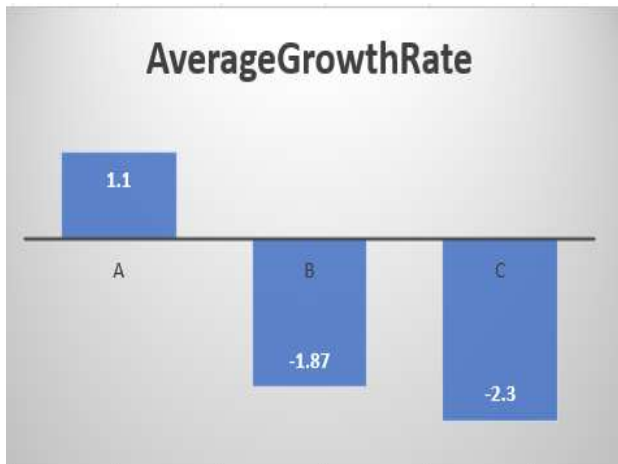
- Walmart wants to optimize its sales strategies by analyzing historical transaction data across branches,
- customer types, payment methods, and product lines. To achieve this, advanced MySQL queries will be
- employed to answer challenging business questions related to sales performance, customer segmentation, and
- product trends

## Task 1: Identifying the Top Branch by Sales Growth Rate

Walmart wants to identify which branch has exhibited the highest sales growth over time. Analyze the total sales for each branch and compare the growth rate across months to find the top performer

```
with monthliesales as(
SELECT branch,
       EXTRACT(MONTH FROM sale_date) AS SaleMonth ,
       round(sum(total),2)AS total_sales
From walmartsales
GROUP BY branch ,EXTRACT(MONTH FROM sale_date)
ORDER BY SaleMonth),
SalesGrowth AS (
  SELECT Branch, SaleMonth, total_sales,
         LAG(total_sales, 1, 0) OVER (PARTITION BY Branch ORDER BY SaleMonth) AS PreviousMonthSales,
         CASE
           WHEN LAG(total_sales, 1, 0) OVER (PARTITION BY Branch ORDER BY SaleMonth) IS NULL OR LAG(total_sales, 1, 0) OVER (PARTITION BY Branch ORDER
BY SaleMonth) = 0 THEN 0
           ELSE (total_sales - LAG(total_sales, 1, 0) OVER (PARTITION BY Branch ORDER BY SaleMonth)) / LAG(total_sales, 1, 0) OVER (PARTITION BY Branch
ORDER BY SaleMonth) * 100
         END AS GrowthRate
  FROM
    MonthlySales),
FinalResult AS (
  SELECT Branch,
         AVG(GrowthRate) AS AverageGrowthRate
  FROM SalesGrowth
  GROUP BY Branch)
SELECT
  Branch, ROUND(AverageGrowthRate, 2) AS AverageGrowthRate
FROM FinalResult
ORDER BY AverageGrowthRate DESC;
```

Result Grid			Filter Rows:	Exp
	Branch	AverageGrowthRate		
▶	A	1.1		
	C	-1.87		
	B	-2.3		



### CTE 1: monthlysales

What it does:

Groups sales data by branch and month.

Calculates the monthly total sales per branch.

Extracts the month number (1–12) from the sale date.

Rounds total sales to 2 decimal places.

Why it's done:

To prepare a clean, monthly view of total sales by each branch. This serves as the base for calculating month-over-month growth.

### CTE 2: SalesGrowth

What it does:

Retrieves previous month's sales using the LAG() function for each branch.

Computes the growth rate :  $\text{GrowthRate} = \frac{\text{Current Sales} - \text{Previous Sales}}{\text{Previous Sales}} \times 100$   
 GrowthRate =  $\frac{\text{Current Sales} - \text{Previous Sales}}{\text{Previous Sales}} \times 100$   
 Sets growth to 0 if no previous month exists or if previous sales = 0.

Why it's done:

To track how sales have grown or declined month-over-month. Helps compare performance within each branch over time.

### CTE 3: FinalResult

What it does:

Takes all monthly growth rates per branch. Calculates the average growth rate per branch.

Why it's done: To understand the overall performance trend of each branch.

Useful for identifying consistently growing or underperforming branches.

### Final SELECT

What it does:

Selects each branch with its rounded average growth rate.

Sorts the branches in descending order of growth.



Why it's done:

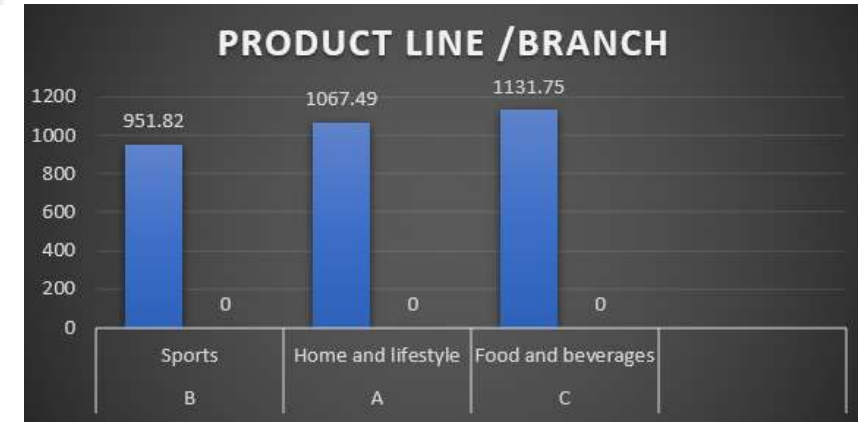
To present final results clearly. Helps in spotting the top-performing branches.

## Task 2: Finding the Most Profitable Product Line for Each Branch

Walmart needs to determine which product line contributes the highest profit to each branch . The profit margin should be calculated based on the difference between the gross income and cost of goods sold.

```
With ProfitData AS (  
    SELECT branch ,  
           `Product line`,  
           round(SUM(`gross income`),2) AS total_profit  
    FROM walmartsales  
    GROUP BY `Product line`,branch),  
RankedProfit AS (  
    SELECT *,  
           RANK() OVER (PARTITION BY Branch ORDER BY total_profit DESC) AS ranks  
    FROM ProfitData)  
SELECT branch , `Product line`, total_profit,  
       CASE  
         WHEN total_profit >= 1000 THEN 'High'  
         WHEN total_profit >= 500 THEN 'Medium'  
         ELSE 'Low'  
       END AS profit_margin_group  
FROM RankedProfit  
WHERE rank = 1;
```

Result Grid				
Filter Rows:		Export:  Wrap Cell Content: 		
	branch	Product line	total_profit	profit_margin_group
▶	A	Home and lifestyle	1067.49	High
	B	Sports and travel	951.82	Medium
	C	Food and beverages	1131.75	High



**CTE 1: ProfitData**

What it does:  
Groups data by branch and product line.  
Calculates total profit (gross income) for each product line within each branch.Rounds the profit to 2 decimal places.  
Why it's done:  
To understand how much profit each product line contributes in every branch.  
Prepares data for ranking and profit categorization.

**2. CTE 2: RankedProfit**

What it does:  
Assigns a rank to each product line within a branch based on total profit, in descending order.  
Uses RANK() function to find the most profitable product line per branch.  
Why it's done:  
to identify the top-earning product line in each branch.Enables filtering later to focus only on highest contributors.

**3. Final SELECT**

What it does:  
Filters to only show product lines that are ranked 1 (most profitable) in each branch.  
Adds a profit margin group:High if profit ≥ 1000Medium if profit ≥ 500Low if profit < 500  
Why it's done:  
To highlight the most profitable product line in each branch.  
Categorizes them for easier comparison and business decision-making.

### Task 3: Analyzing Customer Segmentation Based on Spending

Walmart wants to segment customers based on their average spending behavior. Classify customers into three tiers: High, Medium, and Low spenders based on their total purchase amounts

```
WITH CustomerSpending AS (  
    SELECT  
        `Customer ID`,  
        round(sum(Total)) AS TotalSpending  
    FROM  
        walmartsales  
    GROUP BY  
        `Customer ID`  
)  
SpendingPercentiles AS (  
    SELECT  
        TotalSpending,  
        PERCENT_RANK() OVER (ORDER BY TotalSpending) AS SpendingRank  
    FROM CustomerSpending )  
SELECT  
    cs.`Customer ID`,  
    cs.TotalSpending,  
    CASE  
        WHEN sp.SpendingRank >= 0.75 THEN 'High Spender'  
        WHEN sp.SpendingRank >= 0.25 AND sp.SpendingRank < 0.75 THEN 'Medium Spender'  
        ELSE 'Low Spender'  
    END AS SpendingSegment  
FROM CustomerSpending cs  
JOIN  
    SpendingPercentiles sp ON cs.TotalSpending = sp.TotalSpending  
ORDER BY  
    cs.TotalSpending DESC;
```

Result Grid			
Filter Rows:		Export:	
	Customer ID	TotalSpending	SpendingSegment
▶	8	26634	High Spender
	3	23402	High Spender
	2	23392	High Spender
	15	22674	High Spender
	1	22635	Medium Spender
	12	21721	Medium Spender
	11	21399	Medium Spender
	13	21064	Medium Spender
	14	21049	Medium Spender
	10	20724	Medium Spender
	6	20694	Medium Spender
	7	20628	Low Spender
	9	19662	Low Spender
	5	19632	Low Spender
	4	17657	Low Spender

### CTE 1: Customer Spending

What it does:

Groups data by Customer ID . Calculates and rounds the total amount spent by each customer

. Why it's done:

To get a summary of how much each customer has spent overall.

This becomes the basis for customer segmentation

### .2. CTE 2: Spending Percentiles

What it does:

Uses PERCENT\_RANK() to assign a percentile rank based on total spending.

Ranks customers from lowest to highest spender.

Why it's done:

To evaluate each customer’s spending in relation to others.

Prepares for assigning them into different spending tiers.

### 3. Final SELECT

What it does:

Joins total spending with percentile ranks.

Segments customers as:High Spender: top 25%Medium Spender: middle 50%Low Spender: bottom 25%Sorts customers from highest to lowest spending.

Why it's done:

To classify customers into value segments for marketing, loyalty programs, or targeted offers.

## Task 4: Detecting Anomalies in Sales Transactions

Walmart suspects that some transactions have unusually high or low sales compared to the average for the product line. Identify these anomalies.

```
WITH ProductLineStats AS (  
    SELECT `Product line`,AVG(Total) AS AvgSales, STDDEV_SAMP(Total) AS  
StdDevSales  
    FROM walmartsales  
    GROUP BY `Product line`),  
AnomalyTransactions AS (  
    SELECT ws.`Invoice ID`, ws.`Product line`,ws.Total,pls.AvgSales,pls.StdDevSales,  
    CASE  
        WHEN ws.Total > (pls.AvgSales + 3 * pls.StdDevSales) THEN 'High  
Anomaly'  
        anomalies  
        WHEN ws.Total < (pls.AvgSales - 3 * pls.StdDevSales) THEN 'Low Anomaly'  
        ELSE 'Normal'  
    END AS AnomalyStatus  
    FROM walmartsales ws  
    JOIN ProductLineStats pls ON ws.`Product line` = pls.`Product line`)  
SELECT `Invoice ID`, `Product line`, Total,AvgSales, StdDevSales, AnomalyStatus  
FROM AnomalyTransactions  
WHERE AnomalyStatus <> 'Normal';
```



Result Grid						
Filter Rows:		Export:		Wrap Cell Content:		
	Invoice ID	Product line	Total	AvgSales	StdDevSales	AnomalyStatus
▶	860-79-0874	Fashion accessories	1042.65	305.089297752809	243.56412968898715	High Anomaly
	687-47-8271	Fashion accessories	1039.29	305.089297752809	243.56412968898715	High Anomaly

### 1CTE 1: Product Line Stats

What it does:

Calculates the average sale amount (AvgSales) for each product line.

Calculates the standard deviation (StdDevSales) of total sales within each product line.

Why it's done:

To establish a statistical baseline for detecting unusually high or low sales.

Provides metrics needed to define outliers (anomalies).

### 2. CTE 2: Anomaly Transactions

What it does:

Joins individual transactions with their product line stats. Flags each transaction as: High Anomaly:

if it exceeds  $\text{Avg} + 3 \times \text{StdDev}$  Low Anomaly: if it falls below  $\text{Avg} - 3 \times \text{StdDev}$  Normal: if within the expected range

Why it's done:

To detect outlier transactions that significantly deviate from typical behavior.

Helps identify potential data issues, fraud, or rare events.

### 3. Final SELECT

Retrieves only the anomalous transactions.

Displays relevant details: invoice ID, product line, actual total, average, standard deviation, and status.

Why it's done:

To focus analysis on unusual sales activity.

Supports investigation or corrective action where needed.

## Task 5: Most Popular Payment Method by City

Walmart needs to determine the most popular payment method in each city to tailor marketing strategies

```
WITH PaymentCounts AS (  
    SELECT  
        City,  
        Payment,  
        COUNT(*) AS PaymentCount,  
        ROW_NUMBER() OVER (PARTITION BY City ORDER BY COUNT(*) DESC) AS rn  
    FROM  
        walmartsales  
    GROUP BY  
        City,  
        Payment  
)  
SELECT  
    City,  
    Payment AS MostPopularPaymentMethod  
FROM  
    PaymentCounts  
WHERE  
    rn = 1;
```

Result Grid			Filter Rows:	Export
	City	MostPopularPaymentMethod		
▶	Mandalay	Ewallet		
	Naypyitaw	Cash		
	Yangon	Ewallet		

## CTE: PaymentCounts

What it does:

Groups data by City and Payment method.

Counts how many times each payment method was used in each city.

Uses ROW\_NUMBER() to rank payment methods by frequency within each city.

Why it's done:

To determine the most commonly used payment method per city.

Prepares data for filtering only the top-ranked method in each city.

## 2. Final SELECT

What it does:

Filters the ranked data to get only row number = 1, i.e., the most used payment method per city.

Why it's done:

To present the most popular payment method in each city in a clean and focused way.

## Task 6: Monthly Sales Distribution by Gender

Walmart wants to understand the sales distribution between male and female customers on a monthly basis.

```
select
extract(month from sale_date) as salemonthly ,
gender,
ROUND(SUM(Total), 2) AS totalSales
from walmartsales
group by extract(month from sale_date) ,gender
ORDER BY
SaleMonthly,
Gender;
```

Result Grid			
	salemonthly	gender	totalSales
▶	1	Female	59138.98
	1	Male	57152.89
	2	Female	56335.56
	2	Male	40883.82
	3	Female	52408.39
	3	Male	57047.12

### .1 SELECT Clause

What it does:

Extracts the month from the sale\_date column and names it

SaleMonthly

Groups the data by gender.

Calculates and rounds the total sales for each gender per month.

Why it's done:

To analyze monthly sales trends broken down by gender.

Helps compare how male vs. female customers contributed to sales over time

### .2. GROUP BY Clause

What it does:

Groups the data by month and gender to aggregate totals correctly.

Why it's done:

Enables separate sales totals for each gender in each month.

### 3. ORDER BY Clause

What it does:

Sorts the result first by month, then by gender.

Why it's done:

Ensures the output is organized chronologically and cleanly grouped by gender.

## Task 7: Best Product Line by Customer Type

Walmart wants to know which product lines are preferred by different customer types

```
WITH ProductLineFrequencyByType AS (  
    SELECT  
        `Customer type`,  
        `Product line`,  
        COUNT(*) AS PurchaseFrequency,  
        ROW_NUMBER() OVER (PARTITION BY `Customer type` ORDER BY  
COUNT(*) DESC) AS rn  
    FROM  
        walmartsales  
    GROUP BY  
        `Customer type`,  
        `Product line`  
)  
SELECT  
    `Customer type`,  
    `Product line` AS MostFrequentProductLine,  
    PurchaseFrequency  
FROM  
    ProductLineFrequencyByType  
WHERE  
    rn = 1;
```

Result Grid			
Filter Rows: <input type="text"/>			
Export: 			
Wrap Cell Content: 			
	Customer type	MostFrequentProductLine	PurchaseFrequency
▶	Member	Food and beverages	94
	Normal	Electronic accessories	92

## CTE: Product Line Frequency By Type

What it does:

Groups data by Customer type (e.g., Member or Normal) and Product line.

Counts how many times each product line was purchased per customer type.

Ranks product lines within each customer type using ROW\_NUMBER() based on purchase frequency (most to least).

Why it's done:

To find out which product line is most frequently purchased by each type of customer.

Prepares for filtering just the top product line per customer category.

## 2. Final SELECT

What it does:

Filters to include only rank 1 entries (most purchased product line per customer type).

Displays customer type, the most frequent product line, and its purchase count.

Why it's done:

To highlight the top product preference of each customer type for targeted marketing or stocking strategies.

## Task 8: Identifying Repeat Customers

Walmart needs to identify customers who made repeat purchases within a specific time frame (e.g., within 30 days).

```
WITH CustomerPurchases AS (  
    SELECT `Customer ID`, sale_date,  
           ROW_NUMBER() OVER (PARTITION BY `Customer ID` ORDER BY  
sale_date) as PurchaseNumber,  
           LAG(sale_date, 1, NULL) OVER (PARTITION BY `Customer ID` ORDER  
BY sale_date) as PreviousPurchaseDate  
    FROM walmartsales),  
RepeatPurchaseDates AS (  
    SELECT `Customer ID`, sale_date,  
           DATEDIFF(sale_date, PreviousPurchaseDate) AS  
DaysSinceLastPurchase  
    FROM CustomerPurchases  
    WHERE PurchaseNumber > 1)  
SELECT DISTINCT `Customer ID`  
FROM RepeatPurchaseDates  
WHERE DaysSinceLastPurchase <= 30;
```



Result Grid		Filter
	Customer ID	
▶	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	
	12	
	13	
	14	
	15	

### CTE 1: CustomerPurchases

What it does:

For each Customer ID, assigns a purchase number using ROW\_NUMBER() (ordered by sale\_date). Uses LAG() to get the date of the previous purchase for each customer.

Why it's done:

To track the sequence of purchases and calculate the time between repeat purchases.

### 2. CTE 2: RepeatPurchaseDates

What it does:

Calculates the number of days between current and previous purchase using DATEDIFF().

Why it's done:

To identify how soon a customer returns after a purchase — useful for tracking loyalty or buying habits.

### 3. Final SELECT

What it does:

Filters and returns distinct customers who made repeat purchases within 30 days.

Why it's done:

To identify engaged or loyal customers based on repeat buying within a short period.

## Task 9: Finding Top 5 Customers by Sales Volume (

Walmart wants to reward its top 5 customers who have generated the most sales Revenue

```
SELECT
    `Customer ID`,
    round(SUM(Total),2)AS TotalRevenue
FROM
    walmartsales
GROUP BY
    `Customer ID`
ORDER BY
    TotalRevenue DESC
LIMIT 5;
```

	Customer ID	TotalRevenue
▶	8	26634.34
	3	23402.26
	2	23392.28
	15	22674.46
	1	22634.55

## **SELECT Clause**

What it does:

Retrieves each Customer ID.

Calculates and rounds the total revenue they generated using SUM(Total).

Why it's done:

To find out how much each customer has contributed to overall sales.

## **2. GROUP BY Clause**

What it does:

Groups data by Customer ID to aggregate revenue per customer.

Why it's done:

Ensures accurate total revenue per individual customer.

## **3. ORDER BY & LIMIT**

What it does:

Sorts customers by highest revenue first.

Limits the result to top 5 highest-spending customers.

Why it's done:

To highlight the most valuable customers based on revenue contribution.

## Task 10: Analyzing Sales Trends by Day of the Week

Walmart wants to analyze the sales patterns to determine which day of the week brings the highest sales.

```
SELECT
    DAYNAME(sale_Date) AS Weekday,
    ROUND(SUM(Total), 2) AS TotalSales
FROM
    walmartsales
GROUP BY
    Weekday
ORDER BY
    TotalSales DESC;
limit 1;
```

Result Grid			Filter Rows:
	Weekday	TotalSales	
▶	Saturday	56120.81	
	Tuesday	51482.25	
	Thursday	45349.25	
	Sunday	44457.89	
	Friday	43926.34	
	Wednesday	43731.14	
	Monday	37899.08	

Result Grid			Filter Rows:
	Weekday	TotalSales	
▶	Saturday	56120.81	

## SELECT Clause

What it does:

DAYNAME(sale\_Date) extracts the name of the day (e.g., "Monday", "Tuesday") from each sale date.

ROUND(SUM(Total), 2) calculates the total sales for each weekday and rounds it to 2 decimal places.

Why it's done:

To convert dates into readable weekday names and get accurate sales figures per day.

## 2. FROM Clause

What it does:

Pulls data from the table walmartsales.

Why it's done:

This is the source where each sale and its corresponding date are stored

## 3. GROUP BY Clause

What it does:

Groups all transactions by Weekday (e.g., all Mondays together, all Tuesdays together, etc.).

Why it's done:

To calculate total sales per weekday so we can compare daily performance.

## 4. ORDER BY Clause

What it does:

Sorts the weekdays based on TotalSales in descending order.

Why it's done:

To highlight which days had the highest sales, making it easier to spot peak business days.

[https://drive.google.com/file/d/1LR2AgvDsOUJwjYoKCHx-zRzXJUI3fEvL/view?usp=drive\\_link](https://drive.google.com/file/d/1LR2AgvDsOUJwjYoKCHx-zRzXJUI3fEvL/view?usp=drive_link)