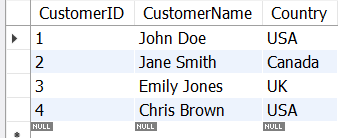
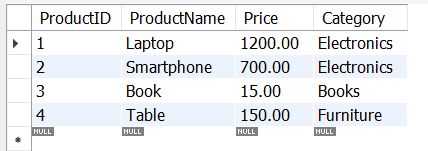
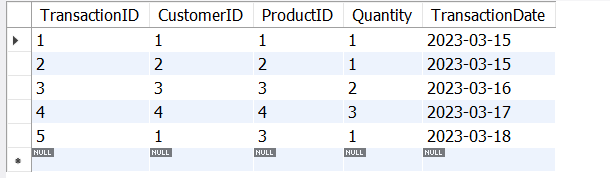
# Problem statement 1

**Table present :**

1. Customers
2. Products
3. Transactions

## Questions / Answers:

1. What is the total revenue generated from each product category?

**Code Used :**

SELECT

Products.Category,

SUM(Transactions.Quantity \* Products.Price) AS TotalRevenue FROM

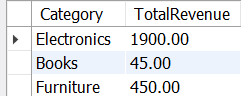
Transactions JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Products.Category;

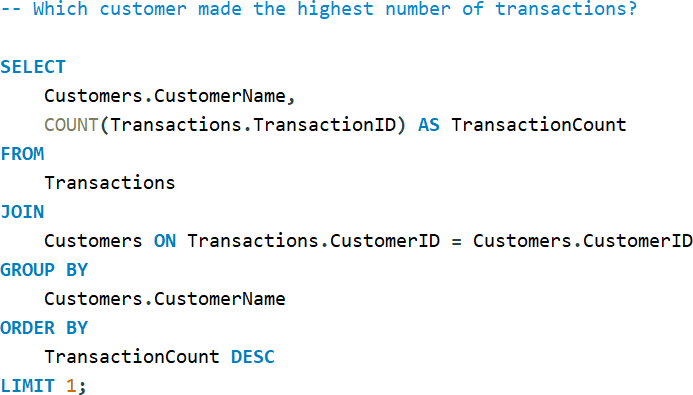
**Explanation:**

This query calculates the total revenue generated from each product category by joining the Transactions and Products tables. It multiplies the quantity of each product sold by its price and then sums the results for each category. The GROUP BY clause ensures that the revenue is grouped by product category, giving the total revenue for each one.

**Output:**

1. Which customer made the highest number of transactions?

### Code used:



**Explanation:**

The query joins the Transactions and Customers tables to get the customer names.It counts the number of transactions each customer made. The results are ordered in descending order, and LIMIT 1 ensures that only the customer with the highest number of transactions is returned.

### Output:

1. List products that have never been sold.

### Code used:

SELECT

ProductName FROM

Products WHERE

ProductID NOT IN ( SELECT ProductID FROM Transactions

);

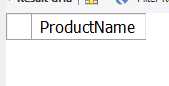
### Explanation:

Since every product in your Products table has been involved in a transaction based on the data you provided:

* **Laptop (ProductID: 1)** has been sold.
* **Smartphone (ProductID: 2)** has been sold.
* **Book (ProductID: 3)** has been sold.
* **Table (ProductID: 4)** has been sold.

This means there are no unsold products, so the query correctly returns an empty result.

### Output:



1. What is the average transaction value for each country?

### Code used:

SELECT

Customers.Country,

AVG(Transactions.Quantity \* Products.Price) AS AvgTransactionValue FROM

Transactions JOIN

Customers ON Transactions.CustomerID = Customers.CustomerID JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Customers.Country;

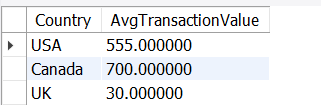
**Explanation:**

**Customers.Country**: We want to calculate the average transaction value for each country, so we're grouping by Country.

**AVG(Transactions.Quantity \* Products.Price)**: The average transaction value is calculated by multiplying the Quantity of products sold by their Price and then averaging the results for each country.

**JOIN**: We join the Transactions table with the Customers and Products tables to link transactions to the customers and products involved.

**Output:**



1. Which product category is most popular in terms of quantity sold?

**Code used** :

SELECT

Products.Category,

SUM(Transactions.Quantity) AS TotalQuantitySold FROM

Transactions JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Products.Category ORDER BY

TotalQuantitySold DESC LIMIT 1;

**Explanation:**

**SUM(Transactions.Quantity)**: This sums up the total quantity sold for each product category.

**GROUP BY Products.Category**: This groups the data by product category, so we can calculate the total quantity sold per category.

**ORDER BY TotalQuantitySold DESC**: The results are ordered by the total quantity sold, from highest to lowest.

**LIMIT 1**: This ensures that only the most popular product category (the one with the highest total quantity sold) is returned.

**Output:**



1. Identify customers who have spent more than $1000 in total.

### Code :

SELECT

Customers.CustomerName,

SUM(Transactions.Quantity \* Products.Price) AS TotalSpent FROM

Transactions JOIN

Customers ON Transactions.CustomerID = Customers.CustomerID JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Customers.CustomerName HAVING

SUM(Transactions.Quantity \* Products.Price) > 1000;

### Explanation :

**SUM(Transactions.Quantity \* Products.Price)**: This calculates the total amount spent by each customer by multiplying the quantity of products purchased by their price.

**GROUP BY Customers.CustomerName**: Groups the transactions by customer so we can calculate the total for each customer.

**HAVING SUM(...) > 1000**: Filters the results to only include customers who have spent more than

$1000 in total.

### Output:

1. How many transactions involved purchasing more than one item?

### Code:

SELECT

COUNT(\*) AS TransactionsWithMultipleItems FROM

Transactions WHERE

Quantity > 1;

### Explanation :

**COUNT(\*)**: This counts the number of rows that meet the specified condition.

**FROM Transactions**: We're selecting from the Transactions table.

**WHERE Quantity > 1**: This condition filters the transactions to only include those where more than one item was purchased.

### Output:

1. What is the difference in total sales between 'Electronics' and 'Furniture' categories?

### Code:

SELECT

SUM(CASE WHEN Products.Category = 'Electronics' THEN Transactions.Quantity \* Products.Price ELSE 0 END) AS TotalSalesElectronics,

SUM(CASE WHEN Products.Category = 'Furniture' THEN Transactions.Quantity \* Products.Price ELSE 0 END) AS TotalSalesFurniture,

SUM(CASE WHEN Products.Category = 'Electronics' THEN Transactions.Quantity \* Products.Price ELSE 0 END) -

SUM(CASE WHEN Products.Category = 'Furniture' THEN Transactions.Quantity \* Products.Price ELSE 0 END) AS SalesDifference

FROM

Transactions JOIN

Products ON Transactions.ProductID = Products.ProductID;

### Explanation:

**SUM(CASE WHEN ...)**: This construct allows you to calculate the total sales for each category separately.

* For 'Electronics', it sums up the product of Quantity and Price where the category is 'Electronics'.
* For 'Furniture', it does the same for the 'Furniture' category.

**SalesDifference**: This subtracts the total sales of 'Furniture' from 'Electronics' to give you the difference in total sales between the two categories.

**JOIN**: The JOIN links the Transactions table with the Products table to access the product categories.

### Output:

1. Which country has the highest average spending per transaction?

### Code:

SELECT

Customers.Country,

AVG(Transactions.Quantity \* Products.Price) AS AverageSpendingPerTransaction FROM

Transactions JOIN

Customers ON Transactions.CustomerID = Customers.CustomerID JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Customers.Country ORDER BY

AverageSpendingPerTransaction DESC LIMIT 1;

### Explanation :

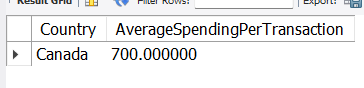
**AVG(Transactions.Quantity \* Products.Price)**: This calculates the average spending per transaction for each country by multiplying the quantity of products by their price.

**GROUP BY Customers.Country**: This groups the data by country so that you can calculate the average for each one.

**ORDER BY AverageSpendingPerTransaction DESC**: This orders the results in descending order, so the country with the highest average spending appears first.

**LIMIT 1**: This returns only the top result, which is the country with the highest average spending per transaction.

### Output:



1. For each product, calculate the total revenue and categorize its sales volume as 'High' (more than $500), 'Medium' ($100-$500), or 'Low' (less than $100)

### Code:

SELECT

Products.ProductName,

SUM(Transactions.Quantity \* Products.Price) AS TotalRevenue, CASE

WHEN SUM(Transactions.Quantity \* Products.Price) > 500 THEN 'High'

WHEN SUM(Transactions.Quantity \* Products.Price) BETWEEN 100 AND 500 THEN 'Medium'

ELSE 'Low'

END AS SalesVolume FROM

Transactions JOIN

Products ON Transactions.ProductID = Products.ProductID GROUP BY

Products.ProductName;

**Explanation :**

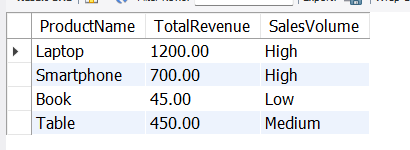
**SUM(Transactions.Quantity \* Products.Price) AS TotalRevenue**: This calculates the total revenue for each product by multiplying the quantity sold by the product price.

**CASE Statement**: This categorizes the total revenue into:

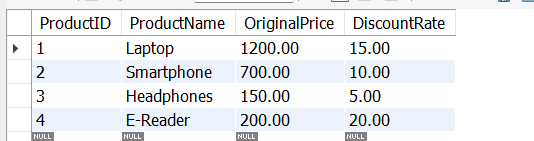
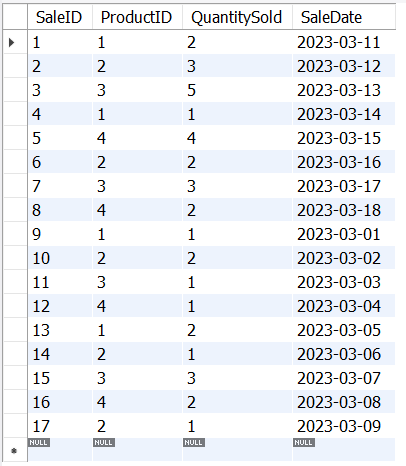
* 'High' if total revenue is greater than $500.
* 'Medium' if total revenue is between $100 and $500.
* 'Low' if total revenue is less than $100.

**GROUP BY Products.ProductName**: This groups the results by product name, allowing the total revenue and sales volume to be calculated for each product.

**Output:**



# Problem statement 2 Tables Present:

1. Products :
2. Sales:

**Questions / Answers:**

1. How much revenue was generated each day of the sale?

### Code:

SELECT

SaleDate,

SUM(QuantitySold \* (OriginalPrice \* (1 - DiscountRate / 100))) AS DailyRevenue FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

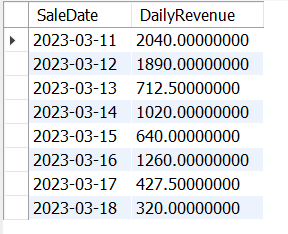
SaleDate BETWEEN '2023-03-11' AND '2023-03-18' -- The sale period GROUP BY

SaleDate ORDER BY

SaleDate;

### Explanation:

This query joins the Sales and Products tables, calculates the discounted price for each product, and then aggregates the revenue per day during the sale period.

Output: 

1. Which product had the highest sales volume during the sale?

### Code:

SELECT

Products.ProductName, SUM(Sales.QuantitySold) AS TotalQuantitySold

FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18' -- The sale period GROUP BY

Products.ProductName ORDER BY

TotalQuantitySold DESC

LIMIT 1; -- Get the product with the highest sales volume

### Explanation:

**SELECT Products.ProductName, SUM(Sales.QuantitySold)**: This selects the product name from the Products table and calculates the total quantity sold for each product by summing the QuantitySold from the Sales table.

**FROM Sales JOIN Products ON Sales.ProductID = Products.ProductID**: The Sales and Products tables are joined on the common ProductID column. This allows the query to retrieve the product names associated with each sale.

**WHERE SaleDate BETWEEN '2023-03-11' AND '2023-03-18'**: This filters the sales data to only include sales that occurred during the specified sale period.

**GROUP BY Products.ProductName**: This groups the sales data by product name, so the SUM(Sales.QuantitySold) is calculated for each product.

**ORDER BY TotalQuantitySold DESC**: The results are ordered in descending order based on the total quantity sold, meaning the product with the highest sales volume appears at the top.

**LIMIT 1**: This limits the result to only one row, giving the product with the highest total sales during the specified sale period.

### Output:

1. What was the total discount given during the sale period?

### Code:

SELECT

SUM(QuantitySold \* (OriginalPrice \* (DiscountRate / 100))) AS TotalDiscountGiven FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18'; -- The sale period

### Explanation:

This query calculates the **total discount** given during the sale period (March 11 to March 18, 2023).

* It works by joining the Sales and Products tables based on the product sold (ProductID).
* For each sale, the query calculates the discount amount per unit by multiplying the

**original price** of the product by the **discount rate** (expressed as a percentage).

* This discount per unit is then multiplied by the **quantity sold** to get the total discount for that sale.
* Finally, the query sums up the total discount across all sales that occurred during the specified sale period.

This result represents the overall discount customers received on their purchases during the sale.

### Output:

1. How does the sale performance compare in terms of units sold before and during the sale?

### Code:

-- How does the sale performance compare in terms of units sold before and during the sale?

SELECT CASE

WHEN SaleDate < '2023-03-11' THEN 'Before Sale'

WHEN SaleDate BETWEEN '2023-03-11' AND '2023-03-18' THEN 'During Sale'

END AS SalePeriod, SUM(QuantitySold) AS TotalUnitsSold

FROM

Sales GROUP BY

SalePeriod ORDER BY

SalePeriod;

### Explanation :

This query categorizes sales into two periods: **before the sale** (before March 11, 2023) and **during the sale** (March 11 to March 18, 2023).

* It uses a CASE statement to label each sale as either "Before Sale" or "During Sale" based on the sale date (SaleDate).
* The query then calculates the **total number of units sold** in each period by summing up

the QuantitySold for all products.

* The results are grouped by the sale period (SalePeriod), which ensures that sales before and during the sale are separated.
* Finally, the results are ordered by the sale period for clarity.

This query provides a comparison of how many units were sold before and during the sale period, offering insight into the impact of the sale on sales volume.

### Output:

1. What was the average discount rate applied to products sold during the sale?

### Code :

SELECT

SUM(QuantitySold \* DiscountRate) / SUM(QuantitySold) AS AvgDiscountRate FROM

Sales JOIN

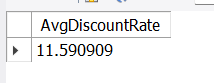
Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18'; -- The sale period

### Explanation:

This query calculates the **average discount rate** applied to products sold during the sale period (March 11 to March 18, 2023), using a **weighted average** approach based on the quantity sold.

* It joins the Sales and Products tables on ProductID to access product information related to each sale.
* For each sale, it multiplies the **discount rate** by the **quantity sold** to weight the discount rate by the number of units sold.
* The query then divides the sum of the weighted discount rates by the total number of units sold during the sale period. This gives the average discount rate, where products with higher sales volumes contribute more heavily to the average.
* The result represents the overall average discount rate across all products sold during the sale period.

**Output**: 

1. a) Which day had the highest revenue

### Code:

SELECT

SaleDate,

SUM(QuantitySold \* (OriginalPrice \* (1 - DiscountRate / 100))) AS DailyRevenue FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18' -- The sale period GROUP BY

SaleDate ORDER BY

DailyRevenue DESC

LIMIT 1; -- Get the day with the highest revenue

### Explanation :

This query finds the day with the **highest revenue** during the sale period (March 11 to March 18, 2023).

* It joins the Sales and Products tables to access product information for each sale.
* For each sale, the **revenue per product** is calculated by multiplying the **quantity sold**

by the **discounted price** of the product (original price reduced by the discount rate).

* It then sums the revenue for each product sold on a particular day to get the **total daily revenue**.
* The query groups the results by SaleDate to calculate the total revenue for each day.
* Finally, it orders the results by DailyRevenue in descending order and uses LIMIT 1

to return the day with the highest total revenue during the sale.

This gives insight into which specific day generated the most sales revenue during the sale period.

### Output:

b) what was the top-selling product on that day?

### Code:

SELECT

Products.ProductName, SUM(Sales.QuantitySold) AS TotalQuantitySold

FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate = ( -- Subquery to get the day with the highest revenue SELECT

SaleDate FROM

Sales JOIN

Products ON Sales.ProductID = Products.ProductID WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18' GROUP BY

SaleDate ORDER BY

SUM(QuantitySold \* (OriginalPrice \* (1 - DiscountRate / 100))) DESC LIMIT 1

) GROUP BY

Products.ProductName

ORDER BY

TotalQuantitySold DESC

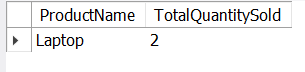
LIMIT 1; -- Get the top-selling product

### Explanation:

This query finds the **top-selling product** on the day with the **highest revenue** during the sale period (March 11 to March 18, 2023).

* The subquery first identifies the day with the highest revenue.
* The main query then calculates the total units sold for each product on that day by summing the QuantitySold.
* It groups the results by ProductName and orders by the total quantity sold, returning the

**top-selling product**.

**Output: **

1. How many units were sold per product category during the sale? (Assuming product categories can be derived from product names or an additional field)

### Code:

SELECT CASE

WHEN ProductName IN ('Laptop', 'Smartphone', 'Headphones') THEN 'Electronics' WHEN ProductName = 'E-Reader' THEN 'E-Reader'

ELSE 'Other' END AS Category,

SUM(QuantitySold) AS TotalUnitsSold

FROM

Sales s JOIN

Products p ON s.ProductID = p.ProductID WHERE

s.SaleDate BETWEEN '2023-03-11' AND '2023-03-20' -- Sale period GROUP BY

Category;

### Explanation:

This query groups products into categories and calculates the total units sold for each category during the sale period (March 11 to March 20, 2023).

* The CASE statement categorizes products as **'Electronics'** (for Laptop, Smartphone, and Headphones), **'E-Reader'**, or **'Other'**.
* It joins the Sales and Products tables on ProductID.
* The query then sums the QuantitySold for each category and groups the results by the product category.
* This provides the total units sold for each product category during the sale period.

### Output:

1. What was the total number of transactions each day?

### Code:

SELECT

SaleDate,

COUNT(\*) AS TotalTransactions FROM

Sales GROUP BY

SaleDate ORDER BY

SaleDate;

### Explanation :

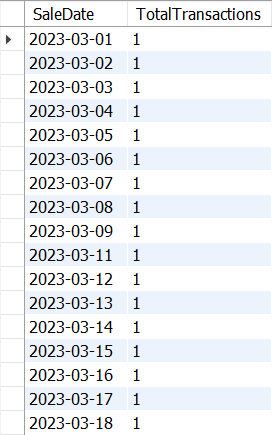
This query calculates the **total number of transactions** (or sales) for each day.

* It groups the sales data by SaleDate, counting the number of transactions (using

COUNT(\*)) that occurred on each day.

* The GROUP BY SaleDate ensures that the count is performed for each individual date.
* Finally, the results are ordered by SaleDate to display the total transactions in chronological order.

This provides insight into the daily volume of transactions across all products.



### Output:

1. Which product had the largest discount impact on revenue?

### Code:

SELECT

Products.ProductID AS ProductID, Products.ProductName AS ProductName,

SUM(Sales.QuantitySold \* Products.OriginalPrice) AS TotalRevenueBeforeDiscount,

SUM(Sales.QuantitySold \* Products.OriginalPrice \* (1 - Products.DiscountRate / 100)) AS TotalRevenueAfterDiscount,

SUM(Sales.QuantitySold \* Products.OriginalPrice) - SUM(Sales.QuantitySold \* Products.OriginalPrice \* (1 - Products.DiscountRate / 100)) AS TotalDiscountImpact

FROM

Products JOIN

Sales ON Products.ProductID = Sales.ProductID GROUP BY

Products.ProductID, Products.ProductName ORDER BY

TotalDiscountImpact DESC

LIMIT 1; -- Fetching the product with the largest discount impact

### Explanation:

This query calculates the **revenue and discount impact** for each product and identifies the product with the largest discount impact.

* **TotalRevenueBeforeDiscount**: The total revenue if products were sold at their original price.
* **TotalRevenueAfterDiscount**: The total revenue after applying the discount.
* **TotalDiscountImpact**: The difference between the revenue before and after the discount, representing the total financial impact of the discount for each product.

It joins the Products and Sales tables on ProductID, groups the results by product, and orders by the **TotalDiscountImpact** in descending order. The query uses LIMIT 1 to return the product with the highest discount impact.

### Output:

1. Calculate the percentage increase in sales volume during the sale compared to a similar period before the sale.

### Code:

WITH SaleVolume AS ( SELECT

SUM(QuantitySold) AS TotalSalesDuringSale FROM

Sales WHERE

SaleDate BETWEEN '2023-03-11' AND '2023-03-18'

),

BeforeSaleVolume AS ( SELECT

SUM(QuantitySold) AS TotalSalesBeforeSale FROM

Sales WHERE

SaleDate BETWEEN '2023-03-01' AND '2023-03-10'

) SELECT

TotalSalesDuringSale, TotalSalesBeforeSale,

((TotalSalesDuringSale - TotalSalesBeforeSale) / TotalSalesBeforeSale) \* 100 AS PercentageIncrease

FROM

SaleVolume, BeforeSaleVolume;

### Explanation:

This SQL query is designed to compare sales volumes during a specific sale period with sales volumes from the period immediately before the sale. It calculates the percentage increase in sales during the sale compared to the period before the sale.

Here’s a breakdown of the query:

1. **WITH SaleVolume AS (...)**: This Common Table Expression (CTE) calculates the total sales during the sale period from **March 11 to March 18, 2023**. It sums the QuantitySold from the Sales table for that date range.
2. **BeforeSaleVolume AS (...)**: This CTE calculates the total sales volume before the sale,

from **March 1 to March 10, 2023**. It also sums the QuantitySold for this earlier date range.

1. **SELECT TotalSalesDuringSale, TotalSalesBeforeSale, ...**: The main query retrieves the total sales during the sale period and the total sales before the sale.
2. **((TotalSalesDuringSale - TotalSalesBeforeSale) / TotalSalesBeforeSale) \* 100 AS PercentageIncrease**: This expression calculates the percentage increase in sales:
   * It subtracts the total sales before the sale from the total sales during the sale.
   * Then, it divides that difference by the total sales before the sale.
   * Finally, it multiplies by 100 to express the result as a percentage.
3. **FROM SaleVolume, BeforeSaleVolume**: The query selects from the two CTEs defined earlier. Because both CTEs return a single value, this cross join is appropriate and results in a single row with the desired calculations.

In summary, the query provides the total sales during and before the sale period, along with the percentage increase in sales during the sale period compared to the previous period.

### Output: