# SQL BASIC KEY ANSWERS

The given case study involves analyzing point-of-sale (POS) data for a retail store. Understanding customer behavior through transaction data can help the store optimize its operations, marketing strategies, and product offerings. The data provided includes information about customer demographics, transaction details, and product categories. Analyzing this data can yield insights into sales trends, customer preferences, and overall business performance.

**Objective:** The retail store aims to understand customer behavior using point-of-sale (POS) data.

**Important Key Points and Background Information:**

Business Context

**DATA PREPARATION AND UNDERSTANDING**

1. **What is the total number of rows in each of the 3 tables in the database?**

--ans 1] --\*\*total number of\_ rows\_ from\_ 3 tables\*\*--

select count(\*) from [dbo].[Customer]

select COUNT(\*) from [dbo].[prod\_cat\_info]

select count(\*) from [dbo].[Transactions]

All This query is counting the total number of rows from the 3 tables in our database. The count(\*) function counts the number of rows.

In all these queries, [dbo] is the schema name, and Customer, prod\_cat\_info, and Transactions are the table names. The count(\*) function is an aggregate function that returns the number of items in a group. In this case, it’s returning the number of rows in each table

1. **What is the total number of transactions that have a return?**

--ans 2] --\*\*total number of\_ transaction\_ that have return\_\*\*--

select COUNT(QTY) as [total return]

from [dbo].[Transactions]

where QTY < 0

this query is calculating the total amount of all transactions that represent returns.

**where QTY < 0:** This is a condition that filters the transactions to only include those where the ‘total\_amt’ field is less than 0. In the context of transactions, a total amount less than 0 typically represents a return.

1. **As you would have noticed, the dates provided across the datasets are not in a correct format. As first steps, pls convert the date variables into valid date formats before proceeding ahead.**

--ans 3] --\*\*date\_format\_ for\_ transaction\_date\*\*-

update [dbo].[Transactions]

set [tran\_date] =convert(varchar,convert(date,[tran\_date],112),23)

 This query is updating the **‘tran\_date’** column in the **‘Transactions’ table.**

**The inner convert(date,[tran\_date],112)**is converting the ‘tran\_date’ values to a date format. **The outer convert(varchar,...,23)** is then converting that date to a varchar (string) format.

**The ‘112’ and ‘23’** are style codes that specify the format of the date.

--\*\*date\_format\_ for\_ customer\_dob\*\*--

update [dbo].[Customer]

set [DOB] = convert(varchar,convert(date,[DOB],112),23)

This query is updating the **‘DOB’** column in the **‘Customer’ table.**

**The inner convert(date,[DOB],112)**is converting the ‘DOB’ values to a date format.

**The outer convert(varchar,...,23)** is then converting that date to a varchar (string) format.

**The ‘112’ and ‘23’** are style codes that specify the format of the date.

update [dbo].[Transactions]

set [Qty] = CONVERT(float(25),[Qty])

This query is updating the **‘Qty’** column in the **‘Transactions’ table**. It’s converting the **‘Qty’** values to a float format with a precision of 25. This means the **‘Qty’** values will now be floating point numbers that can have up to **25** digits

alter table [dbo].[Transactions]

alter column [Qty] float (35)

 This query is altering the ‘Qty’ column in the ‘Transactions’ table to be of type float with a precision of 35. This means the ‘Qty’ values can now be floating point numbers that can have up to 35 digits.

**CONVERT ( data\_type [ ( length ) ] , expression [ , style ] )**

The CONVERT function in SQL Server is used to convert an expression from one data type to another.

1. **What is the time range of the transaction data available for analysis? Show the output in number of days, months and years simultaneously in different columns.**

--ans 4] --\*\*time\_ range\_ of\_ transaction\_ by\_ date\_,month\_,year\_,\*\*

select

DATEDIFF(day,min([tran\_date]),max([tran\_date])) as [days],

DATEDIFF(month,min([tran\_date]),MAX([tran\_date])) as[months],

datediff(year,MIN([tran\_date]),MAX([tran\_date])) as[years]

from [dbo].[Transactions]

---end

**DATEDIFF:** This is a function that calculates the difference between two dates and returns the result. The first parameter is the date part (day, month, year) and the second and third parameters are the start and end dates for the calculation.

**DATEDIFF(datepart, startdate, enddate)**

**min([tran\_date]) and max([tran\_date]):** These functions return the smallest and largest value of the tran\_date column respectively. In this context, they’re used to find the earliest and latest transaction dates.

**as [days], as [months], as [years]:** These are alias names for the new columns that will be created by the DATEDIFF function. The new columns will contain the number of days, months, and years respectively between the earliest and latest transaction dates.

1. **Which product category does the sub-category “DIY” belong to?**

--ans 5] --\*\* the prod\_cat having\_ sub\_category DIY\*\*-

select [prod\_cat],[prod\_subcat] from [dbo].[prod\_cat\_info]

This selects the product category and sub-category from the prod\_cat\_info table.

where [prod\_subcat] ='diy'

This filters the results to only include rows where the sub-category is “DIY”.

**DATA ANALYSIS**

1. **Which channel is most frequently used for transactions?**

--ans 1] --most frequently used channel for transaction--

SELECT TOP 1 [Store\_type],COUNT(Store\_type) as Transaction\_Count FROM [dbo].[Transactions]

GROUP BY [Store\_type]

ORDER BY Transaction\_Count Desc

**SELECT TOP 1 [Store\_type], COUNT(Store\_type) as Transaction\_Count:** This part is selecting the data to be returned. It returns the Store\_type and the count of Store\_type as Transaction\_Count. The TOP 1 clause limits the result to only the top row.

**FROM [dbo].[Transactions]:** This specifies the table from which to select the data.

**GROUP BY [Store\_type]:** This groups the selected data by the Store\_type, allowing us to calculate the sum of total\_amt for each Store\_type.

**ORDER BY [T AMOUNT] Desc :** This orders the result by the **[T AMOUNT]** in descending order.

1. **What is the count of Male and Female customers in the database?**

--ans 2] --\*\*count\_ of\_ male and\_ female\*\*-

select gender,count(gender) as[count of gender]

from [dbo].[Customer]

where gender is not null

group by gender

**SELECT gender, COUNT(gender) AS [count of gender]:** This selects each unique gender and the count of each gender.

**FROM [dbo].[Customer]:** This specifies the table from which to select the data. Replace [dbo].[Customer] with your actual table name.

**WHERE gender IS NOT NULL:** This filters the results to only include rows where the gender is not null.

**GROUP BY gender:** This groups the selected data by the gender, allowing us to count the number of each gender.

1. **From which city do we have the maximum number of customers and how many?**

--ans 3] --\*\*maximum number of customer by city \*\*--

select top 1 [city\_code],count(customer\_Id) as [number of customer] from [dbo].[Customer]

group by [city\_code]

order by [number of customer] desc

---end

**SELECT TOP 1 [city\_code], COUNT(customer\_Id) AS [number of customer]:** This selects the top city\_code and the count of customers for that city\_code. The AS [number of customer] part renames the COUNT(customer\_Id) column to number of customer in the output.

**FROM [dbo].[Customer]:** This specifies the table from which to select the data.

**GROUP BY [city\_code]:** This groups the selected data by the city\_code, allowing us to count the number of customers for each city.

**ORDER BY [number of customer] DESC:** This orders the result by the count of customers in descending order, so the city with the most customers comes first.

1. **How many sub-categories are there under the Books category?**

--ans 4] --\*\*subcategories under the books category\*\*--

select [prod\_cat],count([prod\_subcat]) as sub\_category

from [dbo].[prod\_cat\_info]

where [prod\_cat] = 'books'

group by [prod\_cat]

**SELECT [prod\_cat], COUNT([prod\_subcat]) AS sub\_category:** This selects the prod\_cat and the count of prod\_subcat for that prod\_cat. The AS sub\_category part renames the COUNT([prod\_subcat]) column to sub\_category in the output.

**FROM [dbo].[prod\_cat\_info]:** This specifies the table from which to select the data.

**WHERE [prod\_cat] = 'books':** This filters the results to only include rows where the prod\_cat is ‘books’.

**GROUP BY [prod\_cat]:** This groups the selected data by the prod\_cat, allowing us to count the number of prod\_subcat for each prod\_cat.

1. **What is the maximum quantity of products ever ordered?**

--ans 5] \*\*maximum quantity of product ever ordered\*\*

select max(abs([Qty])) as max\_Qty from [dbo].[Transactions]

**abs([Qty]):** The abs function returns the absolute value of a number

**max(abs([Qty])):** The max function returns the largest value of the selected column

1. **What is the net total revenue generated in categories Electronics and Books?**

-ans 6] -- \*\* net total revenue for\_ electronics and books\*\*--

select p.prod\_cat,SUM(t.total\_amt) as total\_revenue

from [dbo].[prod\_cat\_info] as p

join [dbo].[Transactions] as t on t.prod\_cat\_code = p.prod\_cat\_code

where total\_amt > 0

group by p.prod\_cat

having p.prod\_cat = 'electronics' or p.prod\_cat = 'books'

**SELECT p.prod\_cat, SUM(t.total\_amt) AS total\_revenue:** This selects each prod\_cat and the sum of total\_amt for that prod\_cat. The AS total\_revenue part renames the SUM(t.total\_amt) column to total\_revenue in the output.

**FROM [dbo].[prod\_cat\_info] as p**

**JOIN [dbo].[Transactions] as t ON t.prod\_cat\_code = p.prod\_cat\_code:** This joins the prod\_cat\_info table and the Transactions table on the prod\_cat\_code column.

**WHERE total\_amt > 0:** This filters the results to only include rows where the total\_amt is greater than 0.

**GROUP BY p.prod\_cat:** This groups the selected data by the prod\_cat, allowing us to calculate the sum of total\_amt for each prod\_cat.

**HAVING p.prod\_cat = 'electronics' OR p.prod\_cat = 'books':** This filters the grouped results to only include rows where the prod\_cat is either ‘Electronics’ or ‘Books’.

1. **How many customers have >10 transactions with us, excluding returns?**

--ans 7] --\*\*customer having\_ >10 transction with\_ excluding returns\_\*\*\*--

**select [cust\_id],count([transaction\_id]) as [transaction > 10]**

This selects each cust\_id and the count of transaction\_id for that cust\_id. The AS [transaction > 10] part renames the COUNT([transaction\_id]) column to transaction > 10 in the output.

**from [dbo].[Transactions]**

**where total\_amt>0**

This filters the results to only include rows where the total\_amt is greater than 0 (excluding returns).

**group by [cust\_id]**

This groups the selected data by the cust\_id, allowing us to count the number of transactions for each customer.

**having count([transaction\_id]) > 10**

This filters the grouped results to only include customers who have more than 10 transactions.

1. **What is the combined revenue earned from the “Electronics” & “Clothing” categories, from “Flagship stores”?**

--ans 8] --//combined revenue for for the 'electronics and clothing from flagship stores//--

**select t.store\_type, sum(** CASE WHEN t.total\_amt > 0 THEN t.total\_amt ELSE 0 END**) as [combined revenue]**

This part is selecting the data to be returned. It returns the store\_type and the sum of total\_amt where total\_amt is greater than 0. If total\_amt is not greater than 0, it returns 0. The sum is returned as a new column named ‘combined revenue’

**from prod\_cat\_info as p**

**join [dbo].[Transactions] as t on t.prod\_cat\_code = p.prod\_cat\_code**

This part of the query is joining three tables

**where Store\_type = 'flagship store' and ( p.prod\_cat = 'clothing' or p.prod\_cat = 'electronics')**

This is the filter applied to the data. It only selects the records where the Store\_type is ‘flagship store’ and the prod\_cat is either ‘clothing’ or ‘electronics’.

**group by Store\_type**

: This part groups the selected data by Store\_type. This means that for each unique Store\_type, the query will return a single row with that Store\_type and the sum of the total\_amt for that Store\_type

1. **9. What is the total revenue generated from “Male” customers in “Electronics” category? Output should display total revenue by prod sub-cat.**

--ans 9] --//total revenue from male for custumers from electronics as output subcat//--

**select p.prod\_subcat,c.gender,SUM(CASE WHEN t.total\_amt > 0 THEN t.total\_amt ELSE 0 END) as revenue**

This part is selecting the data to be returned. It returns the store\_type and the sum of total\_amt where total\_amt is greater than 0. If total\_amt is not greater than 0, it returns 0. The sum is returned as a new column named ‘combined revenue’

**from [dbo].[Customer] as c**

**join [dbo].[Transactions] as t on t.cust\_id =c .customer\_Id**

**join [dbo].[prod\_cat\_info] as p on p.prod\_cat\_code = t.prod\_cat\_code**

This part of the query is joining three tables

**where c.Gender = 'm' and( p.prod\_cat = 'electronics')**

This is the filter applied to the data. It only selects the records where the Gender is ‘m’ (male) and the prod\_cat is ‘electronics’.

**group by p.prod\_subcat,Gender**

This part groups the selected data by prod\_subcat and Gender. This means that for each unique combination of prod\_subcat and Gender, the query will return a single row with that prod\_subcat, Gender, and the sum of the total\_amt for that group.

1. **What is percentage of sales and returns by product sub category; display only top 5 sub categories in terms of sales?**

**WITH SalesReturns AS (**

**SELECT P.prod\_subcat,**

**SUM(CASE WHEN T.total\_amt > 0 THEN T.total\_amt ELSE 0 END) AS Total\_Sales,**

**SUM(CASE WHEN T.total\_amt < 0 THEN -T.total\_amt ELSE 0 END) AS Total\_Returns**

**FROM Transactions T**

**JOIN prod\_cat\_info P ON T.prod\_cat\_code = P.prod\_cat\_code**

**GROUP BY P.prod\_subcat**

**),**

**SalesReturnsPercentages AS (**

**SELECT prod\_subcat, Total\_Sales, Total\_Returns,**

**Total\_Sales / (Total\_Sales + Total\_Returns) \* 100 AS Sales\_Percentage,**

**Total\_Returns / (Total\_Sales + Total\_Returns) \* 100 AS Returns\_Percentage**

**FROM SalesReturns )**

**SELECT TOP 5 \***

**FROM SalesReturnsPercentages**

**ORDER BY Sales\_Percentage DESC;**

* **SalesReturns Common Table Expression (CTE): This part of the query calculates the total sales and returns for each product sub-category. It does this by joining the Transactions table (T) and the prod\_cat\_info table (P) on the product category code (prod\_cat\_code). It then groups the results by product sub-category (prod\_subcat) and calculates the sum of the total amount (total\_amt) for sales (where total\_amt > 0) and returns (where total\_amt < 0).**
* **SalesReturnsPercentages CTE: This part of the query calculates the percentage of sales and returns for each product sub-category. It does this by dividing the total sales and total returns for each sub-category by the sum of total sales and total returns, and then multiplying by 100 to convert it to a percentage.**
* **Main Query: This part of the query selects the top 5 product sub-categories in terms of sales percentage. It does this by ordering the results of the SalesReturnsPercentages CTE by Sales\_Percentage in descending order and then selecting the top 5.**

1. **For all customers aged between 25 to 35 years find what is the net total revenue generated by these consumers in last 30 days of transactions from max transaction date available in the data?**

**with tmp\_1 as (**

**select MAX([tran\_date]) as m\_date from [dbo].[Transactions] )**

**select SUM([total\_amt]) as total\_revenue**

**from [dbo].[Transactions] as t**

**join [dbo].[Customer] as c**

**cross join tmp\_1 as m on t.cust\_id=c.customer\_Id**

**where DATEDIFF(year,c.dob,m\_date) between 25 and 35 and t.tran\_date>=DATEADD(day,- 30,m\_date)**

**with tmp\_1 as ( select MAX([tran\_date]) as m\_date from [dbo].[Transactions] ) :-**

This part of the query creates a temporary table tmp\_1 that contains the maximum transaction date (m\_date) from the Transactions table. This is done because we need to compare each transaction’s date with the maximum transaction date in the subsequent steps.

**select SUM([total\_amt]) as total\_revenue**

**from [dbo].[Transactions] as t**

**join [dbo].[Customer] as c**

**cross join tmp\_1 as m on t.cust\_id=c.customer\_Id**

This part of the query specifies the tables from which the data is retrieved.

It joins the Transactions table (aliased as t), the Customer table (aliased as c), and the temporary table tmp\_1 (aliased as m).

The join is based on the condition that the cust\_id in the Transactions table is the same as the customer\_Id in the Customer table.

The CROSS JOIN with tmp\_1 is used to make the maximum transaction date available for every row in the result set.

**JOIN:** The JOIN operation is used to combine rows from two or more tables based on a related column between them.

In your query, JOIN [dbo].[Customer] AS c ON t.cust\_id = c.customer\_Id is joining the Transactions table (aliased as t) and the Customer table (aliased as c) based on the condition that the cust\_id in the Transactions table is the same as the customer\_Id in the Customer table. This means that for each transaction, it will find the corresponding customer based on the cust\_id and customer\_Id match, and combine the information from both tables into one row.

**CROSS JOIN:** The CROSS JOIN operation returns the Cartesian product of rows from the tables in the join.

In other words, it will combine each row from the first table with each row from the second table. If the first table has n rows and the second table has m rows, the result will be a table with n\*m rows.

In your query, CROSS JOIN tmp\_1 AS m is used to make the maximum transaction date available for every row in the result set. Since tmp\_1 only contains one row (the maximum transaction date), the CROSS JOIN simply adds this information to every row in the result set.

-The subquery tmp\_1 finds the maximum transaction date (m\_date) from the Transactions table.

-The main query then calculates the sum of the total\_amt field from the Transactions table (t), where the transactions are made by customers (c) aged between 25 and 35 years.

-The age of the customers is calculated using the DATEDIFF function with their date of birth (dob) and the maximum transaction date (m\_date).

-Only transactions that occurred within the last 30 days from the maximum transaction date are considered, which is ensured by the condition t.tran\_date >= DATEADD(day, -30, m\_date).

1. **Which product category has seen the max value of returns in the last 3 months of transactions?**

DECLARE @MaxDate DATE;

SELECT @MaxDate = MAX(tran\_date) FROM Transactions;

-- Calculate the total returns for each product category in the last 3 months from the max transaction date

WITH CategoryReturns AS (

SELECT P.prod\_cat, SUM(T.total\_amt) AS Total\_Returns FROM Transactions T

JOIN prod\_cat\_info P ON T.prod\_cat\_code = P.prod\_cat\_code

WHERE T.tran\_date BETWEEN DATEADD(MONTH, -3, @MaxDate) AND @MaxDate AND T.total\_amt < 0

GROUP BY P.prod\_cat

)

-- Select the category with the maximum returns

SELECT TOP 1 \*

FROM CategoryReturns

ORDER BY Total\_Returns DESC;

* **Find the Maximum Transaction Date:** The first part of the query finds the maximum transaction date in the Transactions table. This is done using the MAX function on the tran\_date column. The result is stored in a variable @MaxDate.
* **Calculate Total Returns for Each Product Category**: The second part of the query calculates the total returns for each product category in the last 3 months from the maximum transaction date. It does this by joining the Transactions table (T) and the prod\_cat\_info table (P) on the product category code (prod\_cat\_code). It then filters for transactions where the transaction date (tran\_date) is in the last 3 months from the maximum transaction date and the total amount (total\_amt) is less than 0 (indicating returns). The results are grouped by product category (prod\_cat) and the sum of the total amount of these transactions is calculated to find the total returns for each product category. The results are stored in a Common Table Expression (CTE) called CategoryReturns.
* **Select the Category with the Maximum Returns**: The third part of the query selects the product category with the maximum returns. It does this by ordering the results of the CategoryReturns CTE by Total\_Returns in descending order and then selecting the top 1.

1. **Which store-type sells the maximum products; by value of sales amount and by quantity sold?**

alter table [dbo].[Transactions]

alter column [Qty] numeric

select top 1 Store\_type,SUM([total\_amt]) as [maximum sale],sum (qty) as [maximum qty] from [dbo].[Transactions]

where total\_amt>0 and Qty> 0

group by Store\_type

order by [maximum sale] desc

* The ALTER TABLE statement modifies the Qty column in the Transactions table to be of numeric data type. This is to ensure that the quantity can be correctly summed up in the subsequent query.
* The main query then calculates the sum of the total\_amt and Qty fields from the Transactions table, where the transactions are sales (i.e., total\_amt and Qty are greater than 0).
* The results are grouped by the store type (Store\_type), and the total sales amount (maximum sale) and total quantity sold (maximum qty) for each store type are calculated.
* The TOP 1 clause combined with the ORDER BY clause ensures that only the store type with the maximum sales amount is returned.
* So, the store type that sells the maximum products by value of sales amount and by quantity sold is returned by this query.

1. **What are the categories for which average revenue is above the overall average**

select p.prod\_cat,avg(total\_amt) as averge\_amt from [dbo].[Transactions] ast

join [dbo].[prod\_cat\_info] as p on p.prod\_cat\_code = t.prod\_cat\_code

group by p.prod\_cat

having avg(total\_amt) > (select AVG([total\_amt]) as [total average] from [dbo].[Transactions] )

* The main query calculates the average of the total\_amt field from the Transactions table (t), where the transactions are sales (i.e., total\_amt is greater than 0).
* The results are grouped by the product category (p.prod\_cat), and the average sales amount (averge\_amt) for each product category is calculated.
* The HAVING clause then filters out the product categories whose average sales amount is less than the overall average sales amount. The overall average sales amount is calculated as the average of the total\_amt field from the Transactions table, where the transactions are sales.
* So, the product categories for which the average revenue is above the overall average are returned by this query.

**15 Find the average and total revenue by each subcategory for the categories which are among top 5 categories in terms of quantity sold**

SELECT

p.prod\_subcat,

AVG(t.total\_amt) as AverageRevenue,

SUM(t.total\_amt) as TotalRevenue

FROM

Transactions t

JOIN

prod\_cat\_info p ON t.prod\_cat\_code = p.prod\_cat\_code AND t.prod\_subcat\_code = p.prod\_sub\_cat\_code

WHERE

t.prod\_cat\_code IN (

SELECT TOP 5 prod\_cat\_code

FROM Transactions

GROUP BY prod\_cat\_code

ORDER BY SUM(Qty) DESC

)

GROUP BY

p.prod\_subcat;

Absolutely, let’s break down the SQL queries step by step:

1. **Identify the top 5 categories in terms of quantity sold:**

SELECT TOP 5 prod\_cat\_code, SUM(Qty) as TotalQuantityFROM TransactionsGROUP BY prod\_cat\_codeORDER BY TotalQuantity DESC;

SELECT TOP 5 prod\_cat\_code, SUM(Qty) as TotalQuantity: This line is selecting the top 5 product categories (prod\_cat\_code) and the total quantity sold for each (SUM(Qty)). The SUM(Qty) function adds up all the quantities sold for each product category.

FROM Transactions: This line is specifying the table from which to select the data, in this case, the Transactions table.

GROUP BY prod\_cat\_code: This line is grouping the selected data by product category. This means that the quantities sold will be summed up for each unique product category.

ORDER BY TotalQuantity DESC: This line is ordering the grouped data by the total quantity sold in descending order. This means that the product categories with the highest quantities sold will appear first.

1. **Find the average and total revenue by each subcategory for these top 5 categories:**

Absolutely, let’s break down the SQL queries step by step:

1. **Identify the top 5 categories in terms of quantity sold:**

SELECT TOP 5 prod\_cat\_code, SUM(Qty) as TotalQuantityFROM TransactionsGROUP BY prod\_cat\_codeORDER BY TotalQuantity DESC;

SELECT TOP 5 prod\_cat\_code, SUM(Qty) as TotalQuantity: This line is selecting the top 5 product categories (prod\_cat\_code) and the total quantity sold for each (SUM(Qty)). The SUM(Qty) function adds up all the quantities sold for each product category.

FROM Transactions: This line is specifying the table from which to select the data, in this case, the Transactions table.

GROUP BY prod\_cat\_code: This line is grouping the selected data by product category. This means that the quantities sold will be summed up for each unique product category.

ORDER BY TotalQuantity DESC: This line is ordering the grouped data by the total quantity sold in descending order. This means that the product categories with the highest quantities sold will appear first.

1. **Find the average and total revenue by each subcategory for these top 5 categories:**

SELECT

p.prod\_subcat,

AVG(t.total\_amt) as AverageRevenue,

SUM(t.total\_amt) as TotalRevenue

FROM

Transactions t

JOIN

prod\_cat\_info p ON t.prod\_cat\_code = p.prod\_cat\_code AND t.prod\_subcat\_code = p.prod\_sub\_cat\_code

WHERE

t.prod\_cat\_code IN (

SELECT TOP 5 prod\_cat\_code

FROM Transactions

GROUP BY prod\_cat\_code

ORDER BY SUM(Qty) DESC

)

GROUP BY

p.prod\_subcat;

**Explain**

**Identify the top 5 categories in terms of quantity sold:**

SELECT TOP 5 prod\_cat\_code

FROM Transactions

GROUP BY prod\_cat\_code

ORDER BY SUM(Qty) DESC

* SELECT TOP 5 prod\_cat\_code, SUM(Qty) as TotalQuantity: This line is selecting the top 5 product categories (prod\_cat\_code) and the total quantity sold for each (SUM(Qty)). The SUM(Qty) function adds up all the quantities sold for each product category.
* FROM Transactions: This line is specifying the table from which to select the data, in this case, the Transactions table.
* GROUP BY prod\_cat\_code: This line is grouping the selected data by product category. This means that the quantities sold will be summed up for each unique product category.
* ORDER BY TotalQuantity DESC: This line is ordering the grouped data by the total quantity sold in descending order. This means that the product categories with the highest quantities sold will appear first.
* **Find the average and total revenue by each subcategory for these top 5 categories:**

SELECT

p.prod\_subcat,

AVG(t.total\_amt) as AverageRevenue,

SUM(t.total\_amt) as TotalRevenue

FROM

Transactions t

JOIN

prod\_cat\_info p ON t.prod\_cat\_code = p.prod\_cat\_code AND t.prod\_subcat\_code = p.prod\_sub\_cat\_code

WHERE

t.prod\_cat\_code IN (

SELECT TOP 5 prod\_cat\_code

FROM Transactions

GROUP BY prod\_cat\_code

ORDER BY SUM(Qty) DESC

)

GROUP BY

p.prod\_subcat;

* SELECT p.prod\_subcat, AVG(t.total\_amt) as AverageRevenue, SUM(t.total\_amt) as TotalRevenue: This line is selecting the subcategory (p.prod\_subcat), the average revenue (AVG(t.total\_amt)), and the total revenue (SUM(t.total\_amt)) for each subcategory.
* FROM Transactions t JOIN prod\_cat\_info p ON t.prod\_cat\_code = p.prod\_cat\_code AND t.prod\_subcat\_code = p.prod\_sub\_cat\_code: This line is joining the Transactions table and the prod\_cat\_info table based on matching product category and subcategory codes.
* WHERE t.prod\_cat\_code IN (SELECT TOP 5 prod\_cat\_code FROM Transactions GROUP BY prod\_cat\_code ORDER BY SUM(Qty) DESC): This line is filtering the data to include only the transactions for the top 5 categories. The subquery inside the IN clause is the same query we used in step 1 to identify the top 5 categories.
* GROUP BY p.prod\_subcat: This line is grouping the selected data by subcategory. This means that the average and total revenues will be calculated for each unique subcategory.