SQL Tiny Shop Sales Case Study

Kyra Brown

Case Study Questions

1) Which product has the highest price? Only return a single row. 2) Which customer has made the most orders?

3) What’s the total revenue per product?

4) Find the day with the highest revenue. -

5) Find the first order (by date) for each customer

Data Used from URL: https://d-i

motion.com/lessons/customer orders-analysis/

6) Find the top 3 customers who have ordered the most distinct products

7) Which product has been bought the least in terms of quantity?

8) What is the median order total?

9) For each order, determine if it was ‘Expensive’ (total over 300), ‘Affordable’ (total over 100), or ‘Cheap’. 10) Find customers who have ordered the product with the highest price.

\*\*Answered using MS SQL Server\*\*

Case Study Answers:

1.

SELECT TOP 1

products.product\_id,

products.product\_name,

MAX(price) AS highest\_price

FROM

products

GROUP BY

product\_name,

product\_id

ORDER BY

highest\_price

DESC

Result:



Explanation: In this question, we're looking for the product that costs the highest.

Step 1: We first take the specified item we're looking for, in this case the product name, and select it from our products table. Step 2: Next we set the resulting product name to be ordered by the price in a descending order.

Step 3: Lastly, we limit the results to show only the highest costing product name being Product M.

2.

SELECT TOP 1 orders.customer\_id, order\_items.quantity AS num\_of\_orders

FROM orders

INNER JOIN order\_items

ON orders.order\_id = order\_items.order\_id

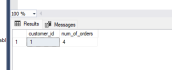
ORDER BY order\_items.quantity DESC

;

SQL Tiny Shop Sales Case Study Page 1

;

Result:



Explanation: In this question, looking for the customer id that made the most orders.

Step 1: We first need to understand what information we need to select, which is the customer id. The customer id appears in 2 tables which are customers and orders. In this case, we're going to pull the customer id from the orders table along with the quantity from the order\_items table nicknamed to num\_of\_orders just to confirm our answer.

Step 2: Now that we've selected the customer id and specified the table it will come from, we'll then find the table containing a shared field, which in this case is the order\_id field from both the orders and order\_items table and join them where they are equal to each other on that field.

Step 3: We have now established our joined tables. Since we're trying to figure out which customer made the most orders, we'll set the order to descending and limit the results to just 1 to get our answer of customer\_id: 1 having the most with 4 orders.

3.

SELECT DISTINCT

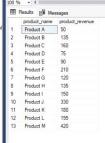
products.product\_name,

SUM(products.price \* order\_items.quantity) OVER(PARTITION BY products.product\_name )AS revenue

FROM order\_items

JOIN products ON products.product\_id=order\_items.product\_id

Result:



Explanation: In this question, we're asked to calculate the total revenue per product.

Step 1: Since what we're looking for is the total revenue per product, we first select distinct product names along with the sum of quantity multiplied by price to get the standard revenue.

Step 2: Now that we have an expression to calculate the revenue, we'll then partition the expression by the product name so that it gives us a total based on the specified product.

Step 3: Since the quantity and price reside in different tables, which are order\_items and products, we must first locate the field shared by the two tables. In this case, the shared field is the product id.

Step 4: Now that we've determined the shared field, its time to join the tables where the product\_id equals one another. Now that the tables are joined, the SUM function can properly run and return our revenue per product

4.

SELECT DISTINCT orders.order\_date, SUM(order\_items.quantity \* products.price) OVER(PARTITION BY orders.order\_date) AS revenue FROM orders

JOIN order\_items

ON orders.order\_id=order\_items.order\_id

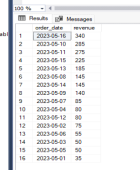
JOIN products

ON products.product\_id=order\_items.product\_id

ORDER BY revenue DESC

Result:

SQL Tiny Shop Sales Case Study Page 2



Explanation: In this problem we're looking for the day with the highest revenue.

Step 1: Since we're looking for the highest selling day, we'll start with selecting distinct order dates from our orders table. Since we're going based on revenue, we can create an expression to calculate the revenue.

Step 2: After creating the expression, we set the expression to calculate revenue by date by partitioning the expression and givig it the nickname revenue.

Step 3: Now that we've set the partition parameters, we must now find the common field between the orders and order\_items table, which in this case is the order\_id and join them together.

Step 4: Now that we've joined the orders and orde\_items table, in order for the Sum expression to run properly, we must also find the shared field between the products and order\_items table, which in this case is the product\_id.

Step 5: Now that we've successfully joined the tables on the necessary fields, the expression that calculates the revenue can run properly. And since we're looking for the highest selling date, we order the results by revenue descending to get our answer of 2023-05-16

5.

SELECT DISTINCT

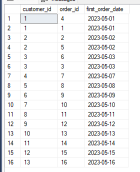
orders.customer\_id,

orders.order\_id,

MIN(orders.order\_date) OVER(PARTITION BY orders.customer\_id) AS first\_order\_date

FROM orders;

Result:



Explanation: In this problem, we're looking for the first order date for each customer.

Step 1: We start with selecting distinct customer\_IDs and the order\_IDs as well from our orders table.

Step 2: Since we're looking for earlier dates, we can use the min to get the earliest date.

Step 3: After creating our expression, we then partition the aggregated order\_dates by the customer\_ID and nicknamed the field first\_order\_date. SQL Tiny Shop Sales Case Study Page 3

6.

SELECT TOP 3

c.customer\_id,

c.first\_name,

c.last\_name,

COUNT(DISTINCT order\_items.product\_id) AS product\_count

FROM customers c

INNER JOIN orders

ON c.customer\_id=orders.customer\_id

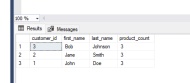
INNER JOIN order\_items

ON order\_items.order\_id=orders.order\_id

GROUP BY c.customer\_id, c.first\_name, c.last\_name

ORDER BY product\_count DESC

Result:



Explanation: In this problem, we're looking for the top 3 customers that have ordered the most distinct products.

Step 1: We begin by selecting our needed information, which in this case is the first and last name and customer\_id, from our customers table. Step 2: We then create an expression to count the distinct product\_IDs from our order\_items table and nicknamed the field product\_count.

Step 3: Since a lot of the data we need is separated between our tables, we first find our shared field between our customers table and orders table, which in this case is the customer\_id, and join them together on that field.

Step 4: Now that we’ve joined our customers and orders table, we'll next find our shared field but this time between the order\_items table and orders table, which in this case is the order\_id.

Step 5: Now that we've joined all of our needed tables, we then specify the structure of our results using group by to group the results how we want them. We then finally order the results in descending order based on the product\_count to get our answer.

7.

WITH Ranked\_Products AS (

SELECT DISTINCT

products.product\_name,

products.product\_id,

SUM(order\_items.quantity) AS total\_quantity\_sold,

RANK() OVER(ORDER BY SUM(order\_items.quantity)) AS prod\_rank

FROM

products

LEFT JOIN

order\_items

ON

order\_items.product\_id=products.product\_id

GROUP BY

products.product\_name,

products.product\_id

)

SELECT TOP 1

prod\_rank,

product\_id,

product\_name,

total\_quantity\_sold

FROM

Ranked\_Products

ORDER BY

total\_quantity\_sold

SQL Tiny Shop Sales Case Study Page 4

total\_quantity\_sold

ASC

Result:



Explanation: In this problem, we're tasked with finding the product that sold lowest in terms of quantity.

Step 1: I first create a window function nicknamed to Ranked\_Products to make selecting everything we need a lot easier later. We're looking for product names, so we first select the product name and id from our products table along with the sum of our quantities nicknamed to total quantity sold from our order\_items table. I create an expression to rank based on quantity and nicknamed it to prod\_rank.

Step 2: Next, we need to locate the common field from both the order\_items and products table, which in this case is the product\_id field, and join them where they're equal to one another on that field. We then group the results of our window function to so that product name is first and product\_id is after it.

Step 3: Now that we've closed up our window function, we can then go straight to extracting from the Ranked\_Products temporary table. We select our prod\_rank, product\_id, product\_name, and the total\_quantity\_sold.

Step 4: We lastly order the results by the quantity sold in ascending order and limited our results to just one row.

SQL Tiny Shop Sales Case Study Page 5