**Business Problem**

**/\* 1. Total Revenue generated \*/**

**SQL Query:**

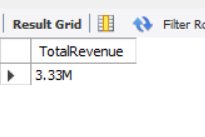
SELECT concat(round(SUM(total\_amount)/1000000,2),'M') TotalRevenue

FROM Orders

INNER JOIN Payments on Orders.payment\_id=Payments.payment\_id

where Payment\_status = 'Completed';

**Output:**

****

**/\* 2.Analyzing the sales trend over the past few years, and identifying patterns of growth or decline. \*/**

**SQL Query:**

SELECT YEAR(Orders.order\_date) YEARNUM, concat(round(SUM(total\_amount)/1000,2),'k') sales

FROM Orders

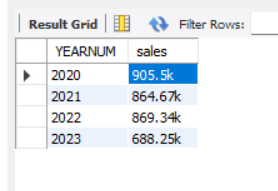
INNER JOIN Payments on Orders.payment\_id=Payments.payment\_id

where Payment\_status = 'Completed'

GROUP BY YEARNUM

ORDER BY YEARNUM;

**Output:**

****

**/\* 3. Total no of orders placed successfully with no payment failed and refunds. \*/**

**SQL Query:**

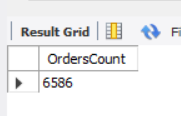
SELECT count(\*) OrdersCount

FROM Orders

INNER JOIN Payments on Orders.payment\_id=Payments.payment\_id

where Payment\_status = 'Completed';

**Output:**

****

**/\* 4. Get count of products for each category top 5\*/**

**SQL Query:**

SELECT categories.categorie\_name, COUNT(product\_id) Productcount

FROM Products

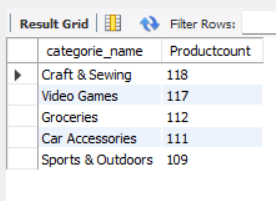
INNER JOIN Categories on Products.Categorie\_id=Categories.categorie\_id

GROUP BY categories.categorie\_name

ORDER BY Productcount DESC

LIMIT 5;

**Output:**

****

**/\* 5. Calculate the most profitable product category \*/**

**SQL Query:**

SELECT categories.categorie\_name, SUM(Order\_items.quantity\*Order\_items.price) AS Total\_Amount

FROM Categories

INNER JOIN Products on Categories.Categorie\_id=Products.categorie\_id

INNER JOIN Order\_items on Products.product\_id=Order\_items.product\_id

GROUP BY categories.categorie\_name

ORDER BY Total\_Amount DESC;

**Output:**



**/\* 6. Top 5 states with high revenue generation.\*/**

**SQL Query:**

SELECT Customers.state, round(SUM(orders.total\_amount),2) Total\_Revenue

FROM Customers

INNER JOIN Orders ON Orders.customer\_id=Customers.customer\_id

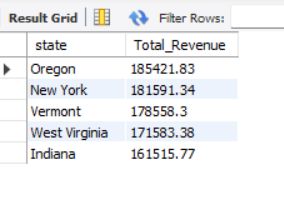
INNER JOIN Payments ON Orders.payment\_id=Payments.payment\_id

WHERE Payments.payment\_status != "Refunded"

GROUP BY Customers.State

LIMIT 5;

**Output:**



**/\* 7. From which States the products are highly in returned. \*/**

**SQL Query:**

SELECT Customers.state, COUNT(shipping.shipping\_status) Returned\_count

FROM Customers

INNER JOIN Orders ON Customers.customer\_id=Orders.customer\_id

INNER JOIN Shipping ON Orders.shipping\_id=Shipping.shipping\_id

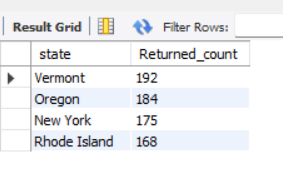
WHERE Shipping.shipping\_status= "Returned"

GROUP BY Customers.state

ORDER BY Returned\_count DESC

LIMIT 4;

**Output:**



/\* **8.** **List all products with low inventory (stock less than 20)**\*/

**SQL Query:**

SELECT products.name, inventory.stock

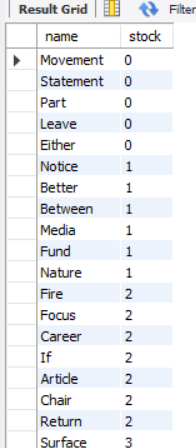
FROM products

JOIN inventory ON products.product\_id = inventory.product\_id

WHERE inventory.stock < 20

ORDER BY inventory.stock;

**Output:**



**/\* 9. Compute the average order value for each customer. Include customers with more than 5 orders.\*/**

**SQL Query:**

SELECT Customers.customer\_id, Customers.First\_name, AVG(Orders.total\_amount)/COUNT(Orders.order\_id) AOV

FROM Customers

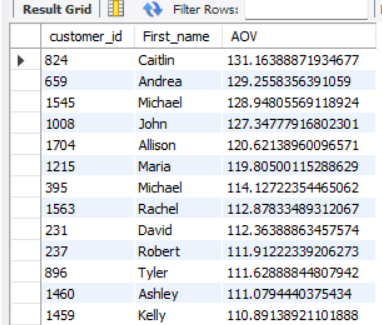
INNER JOIN Orders ON Customers.customer\_id=Orders.customer\_id

GROUP BY Customers.customer\_id, Customers.First\_name

HAVING COUNT(Orders.order\_id) > 5

ORDER BY AOV DESC;

**Output:**

****

**\* /10. Query monthly total sales for last year from the current month\*/**

**SQL Query:**

SELECT MONTH(Orders.order\_date) MonthNO, YEAR(Orders.order\_date) YearNO, ROUND(SUM(total\_amount),2) Total\_Sales

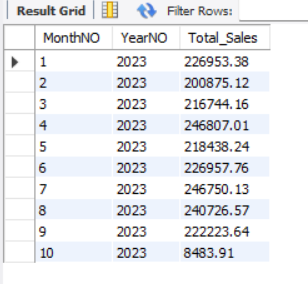
FROM Orders

GROUP BY MonthNO, YearNO

HAVING YearNO=2023

ORDER BY MonthNO, YearNO;

**Output:**

****

**/\* 11. Find the customers who have registered but never placed orders. Include customer details and the time since their registration.\*/**

**SQL Query:**

SELECT customer\_id, First\_name, timestampdiff(MONTH,created\_at,now()) monthdiff

FROM customers

WHERE customer\_id NOT IN (SELECT customer\_id FROM Orders)

ORDER BY monthdiff DESC;

**/\* 12. Find the orders with delays in delivery where the date is 7 days after the shipping date. Include customer, order details.\*/**

**SQL Query:**

WITH delay\_days\_calcu AS(

SELECT customers.first\_name, orders.order\_id, orders.order\_date, shipping.shipping\_date, datediff(shipping.shipping\_date,orders.order\_date) delay\_days

FROM orders

INNER JOIN customers ON customers.customer\_id=orders.customer\_id

INNER JOIN shipping ON orders.shipping\_id=shipping.shipping\_id

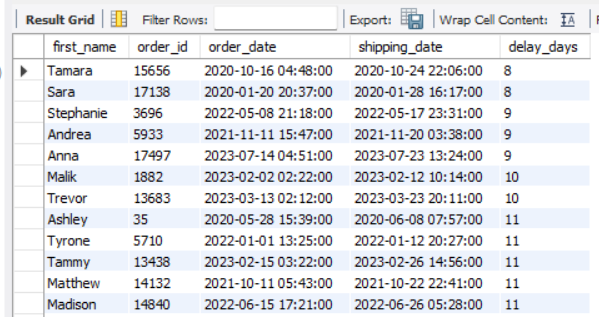
WHERE shipping.shipping\_status='Shipped')

SELECT \* FROM delay\_days\_calcu

WHERE delay\_days>7

ORDER BY delay\_days;

**Output:**

****

**/\* 13. Calculate the percentage of successful payments across all orders. Include other \*/**

**SQL Query:**

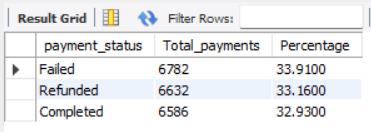
SELECT Payments.payment\_status, COUNT(\*) Total\_payments, COUNT(\*)/(SELECT COUNT(\*) FROM payments) \*100 Percentage

FROM Payments

INNER JOIN Orders ON Payments.payment\_id=orders.payment\_id

GROUP BY Payments.payment\_status;

**Output:**



**/\* 14. Find the top 5 sellers based on total value.\*/**

**SQL Query:**

SELECT Sellers.seller\_id, sellers.seller\_name, SUM(order\_items.quantity\*order\_items.price) total\_value

FROM sellers

INNER JOIN Products ON sellers.seller\_id=Products.seller\_id

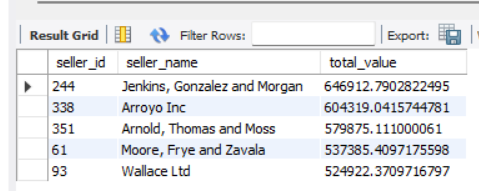
INNER JOIN order\_items ON order\_items.product\_id=Products.product\_id

GROUP BY Sellers.seller\_id, sellers.seller\_name

ORDER BY total\_value DESC

LIMIT 5;

**Output:**



**/\* 15. Calculate the profit margin for each product (difference between products and cost of goods sold). Rank produced by their profit margin, showing highest to lowest.\*/**

**SQL Query:**

SELECT products.product\_id, products.name, (SUM(order\_items.price-products.price)/(SELECT SUM(order\_items.price) FROM order\_items)) \* 100 PROFIT\_MARGIN,

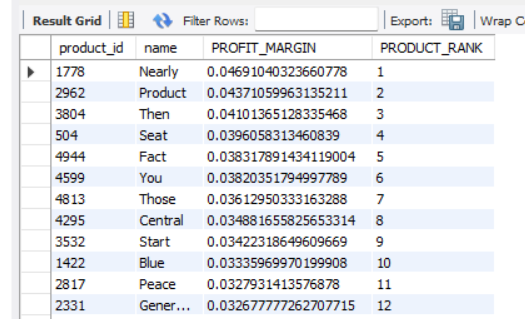
DENSE\_RANK() OVER(ORDER BY (SUM(order\_items.price-products.price)/(SELECT SUM(order\_items.price) FROM order\_items)) \* 100 DESC) PRODUCT\_RANK

FROM Products

INNER JOIN order\_items ON Products.product\_id=order\_items.product\_id

GROUP BY products.product\_id, products.name;

**Output:**

****

**/\* 16. Query top 10 product by their return number.\*/**

**SQL Query:**

SELECT products.product\_id, products.name, count(\*) return\_count FROM products

INNER JOIN order\_items ON Products.product\_id=order\_items.product\_id

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

INNER JOIN shipping ON orders.shipping\_id=shipping.shipping\_id

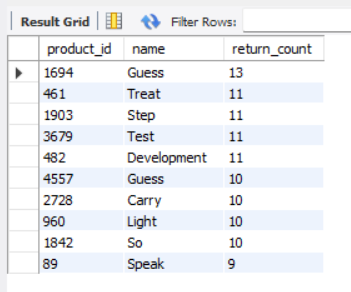
where shipping\_status='Returned'

GROUP BY products.product\_id, products.name

ORDER BY return\_count DESC

LIMIT 10:

**Output:**



**/\* 17. Query the distribution of orders placed by hour of a day. Create a time-based analysis to understand the peak time. \*/**

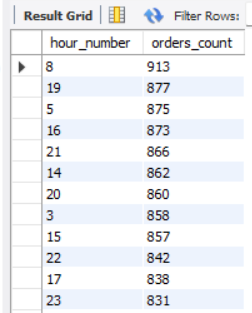
**SQL Query:**

SELECT HOUR(order\_date) hour\_number, COUNT(\*) orders\_count FROM orders

GROUP BY hour\_number

ORDER BY orders\_count DESC;

**Output:**

****

**/\* 18. Query top 5 customers with the highest no of orders for each state. \*/**

**SQL Query:**

WITH customer\_rank AS(

SELECT customers.customer\_id, first\_name, state , count(orders.order\_id) orders\_count, DENSE\_RANK() OVER(PARTITION BY state ORDER BY count(orders.order\_id) DESC ) RANK\_CUSTOMER

FROM customers

INNER JOIN orders ON customers.customer\_id=orders.customer\_id

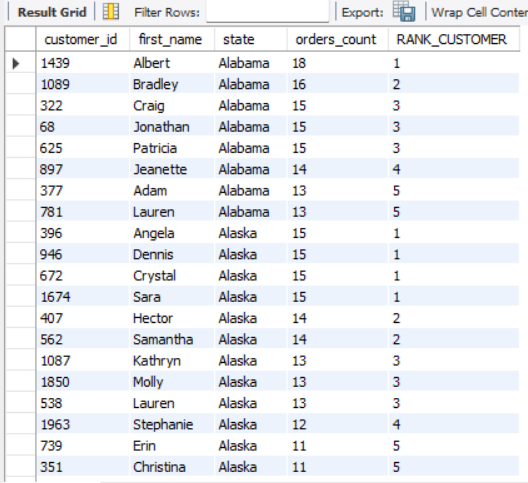
GROUP BY customers.customer\_id, first\_name, state

)

SELECT \* FROM customer\_rank

WHERE RANK\_CUSTOMER <=5;

**Output:**

****

**/\* 19. Calculate the total revenue handled by each seller. Include the total no of orders handled and the average delivery time for each provider. \*/**

**SQL QUERY:**

SELECT sellers.seller\_id, sellers.seller\_name, SUM(order\_items.price) total\_revenue, count(order\_items.order\_item\_id) no\_of\_orders,

AVG(datediff(orders.order\_date,shipping.shipping\_date)) avg\_delivery\_time

FROM sellers

INNER JOIN products ON sellers.seller\_id=products.seller\_id

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

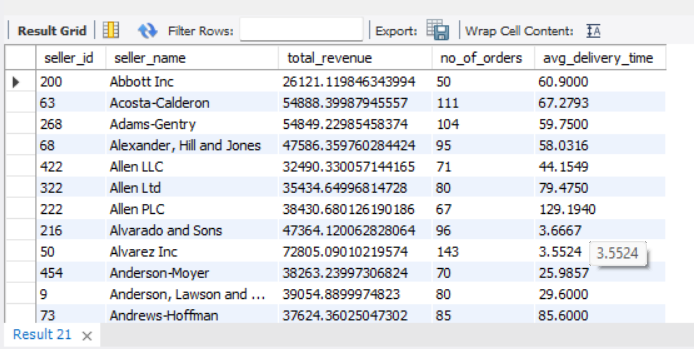
INNER JOIN orders ON order\_items.order\_id=orders.order\_id

INNER JOIN shipping ON orders.shipping\_id=shipping.shipping\_id

GROUP BY sellers.seller\_id, sellers.seller\_name

ORDER BY sellers.seller\_name;

**Output:**

****

**/\* 20. Create a function that the same quantity should be reduced from the inventory table as soon as the product is sold.**

**After adding any sales records it should update the stock in the inventory table and the product and quantity purchased. \*/**

**SQL Query:**

DELIMITER $$

CREATE PROCEDURE Update\_Inventory(IN c\_product\_id INT, IN c\_quantity INT)

BEGIN

DECLARE stockValue INT;

SELECT stock INTO stockValue

FROM inventory

WHERE product\_id=c\_product\_id;

IF stockValue > c\_quantity THEN

UPDATE inventory

SET stock=stock-c\_quantity, last\_updated= now()

WHERE product\_id=c\_product\_id;

SELECT 'Inventory updated successfully';

ELSE

SELECT 'Insufficient stock' AS message;

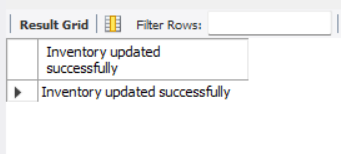
END IF;

END $$

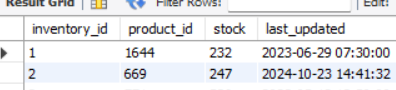
DELIMITER ;

CALL Update\_Inventory(669,100)

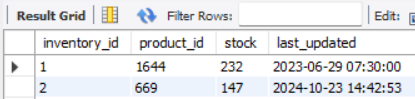
**Output:**

****

**Before Updating:**

****

**After Updating:**

****