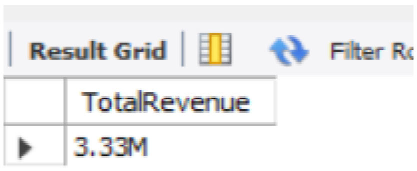
Business Problem

/* 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:



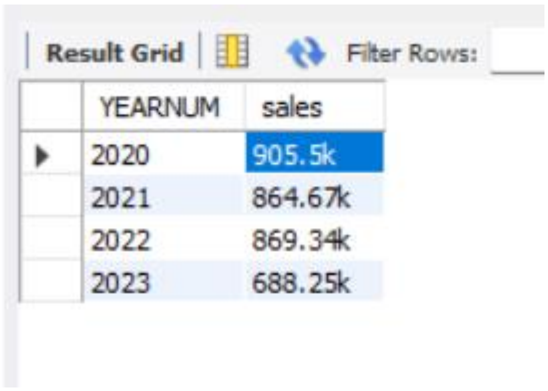
	TotalRevenue
▶	3.33M

/* 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:



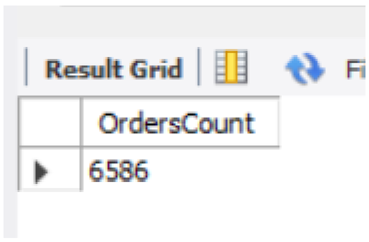
	YEARNUM	sales
▶	2020	905.5k
	2021	864.67k
	2022	869.34k
	2023	688.25k

/* 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:



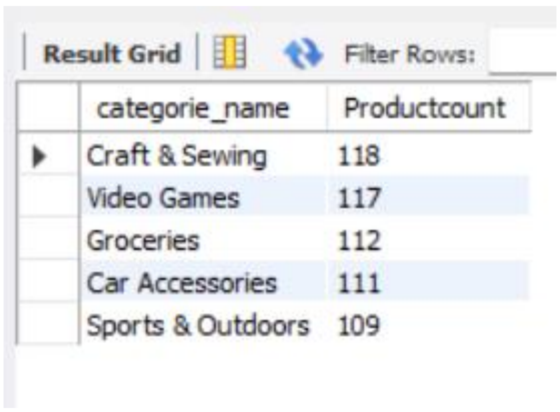
	OrdersCount
▶	6586

/* 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:



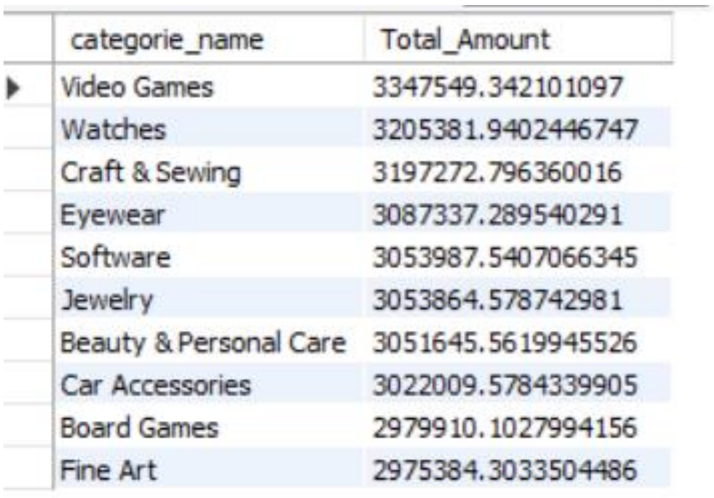
	categorie_name	Productcount
▶	Craft & Sewing	118
	Video Games	117
	Groceries	112
	Car Accessories	111
	Sports & Outdoors	109

/* 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:



categorie_name	Total_Amount
Video Games	3347549.342101097
Watches	3205381.9402446747
Craft & Sewing	3197272.796360016
Eyewear	3087337.289540291
Software	3053987.5407066345
Jewelry	3053864.578742981
Beauty & Personal Care	3051645.5619945526
Car Accessories	3022009.5784339905
Board Games	2979910.1027994156
Fine Art	2975384.3033504486

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

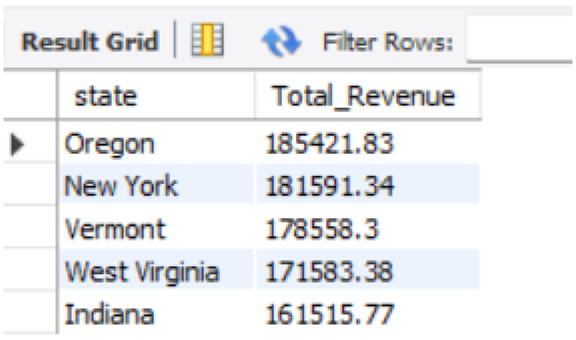
SQL Query:

```
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:



state	Total_Revenue
Oregon	185421.83
New York	181591.34
Vermont	178558.3
West Virginia	171583.38
Indiana	161515.77

/* 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

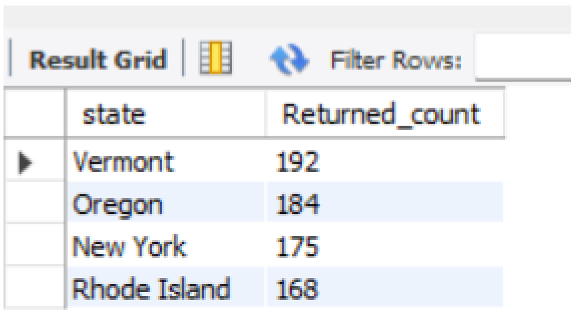
WHERE Shipping.shipping_status= "Returned"

GROUP BY Customers.state

ORDER BY Returned_count DESC

LIMIT 4;

Output:



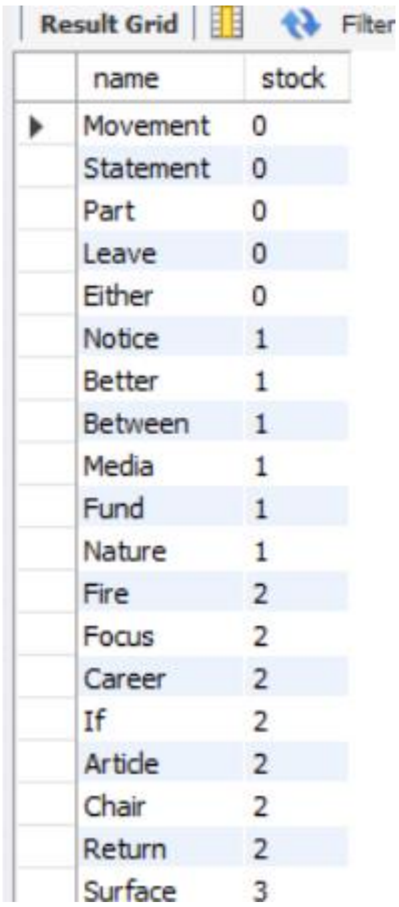
state	Returned_count
Vermont	192
Oregon	184
New York	175
Rhode Island	168

/* 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:



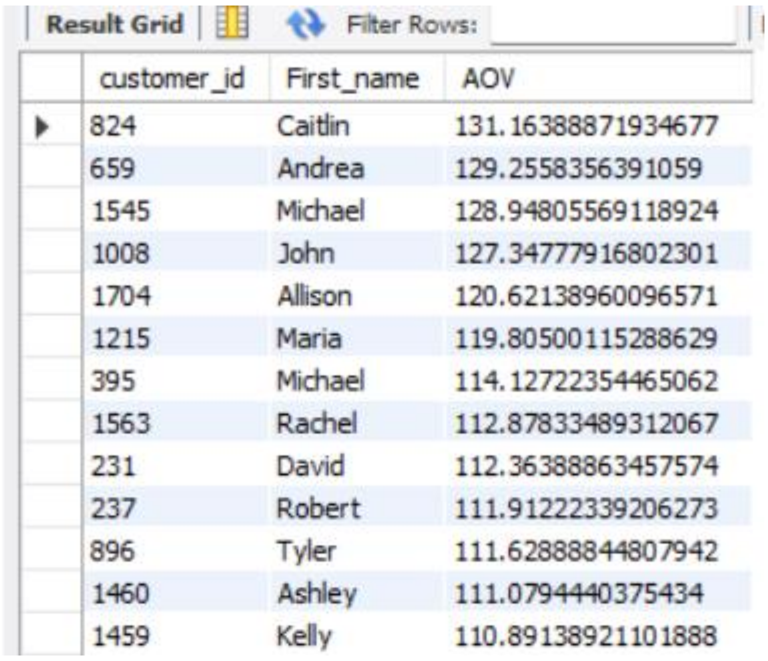
name	stock
Movement	0
Statement	0
Part	0
Leave	0
Either	0
Notice	1
Better	1
Between	1
Media	1
Fund	1
Nature	1
Fire	2
Focus	2
Career	2
If	2
Article	2
Chair	2
Return	2
Surface	3

/* 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:



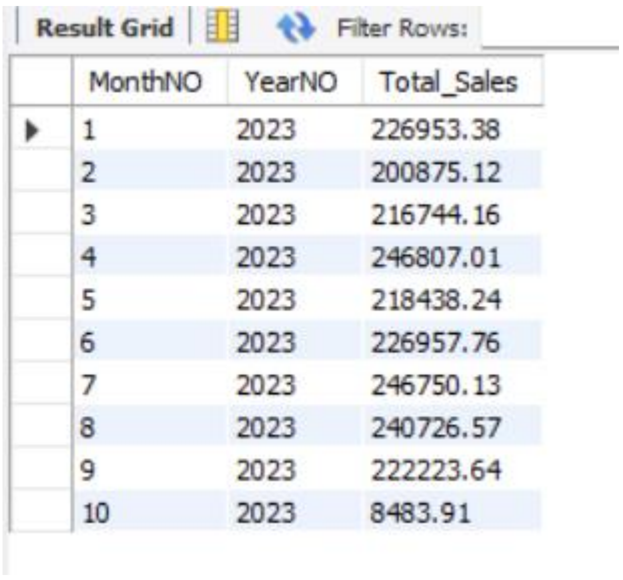
customer_id	First_name	AOV
824	Caitlin	131.16388871934677
659	Andrea	129.2558356391059
1545	Michael	128.94805569118924
1008	John	127.34777916802301
1704	Allison	120.62138960096571
1215	Maria	119.80500115288629
395	Michael	114.12722354465062
1563	Rachel	112.87833489312067
231	David	112.36388863457574
237	Robert	111.91222339206273
896	Tyler	111.62888844807942
1460	Ashley	111.0794440375434
1459	Kelly	110.89138921101888

*** /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:


Result Grid | Filter Rows:

MonthNO	YearNO	Total_Sales
1	2023	226953.38
2	2023	200875.12
3	2023	216744.16
4	2023	246807.01
5	2023	218438.24
6	2023	226957.76
7	2023	246750.13
8	2023	240726.57
9	2023	222223.64
10	2023	8483.91

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

SQL Query:

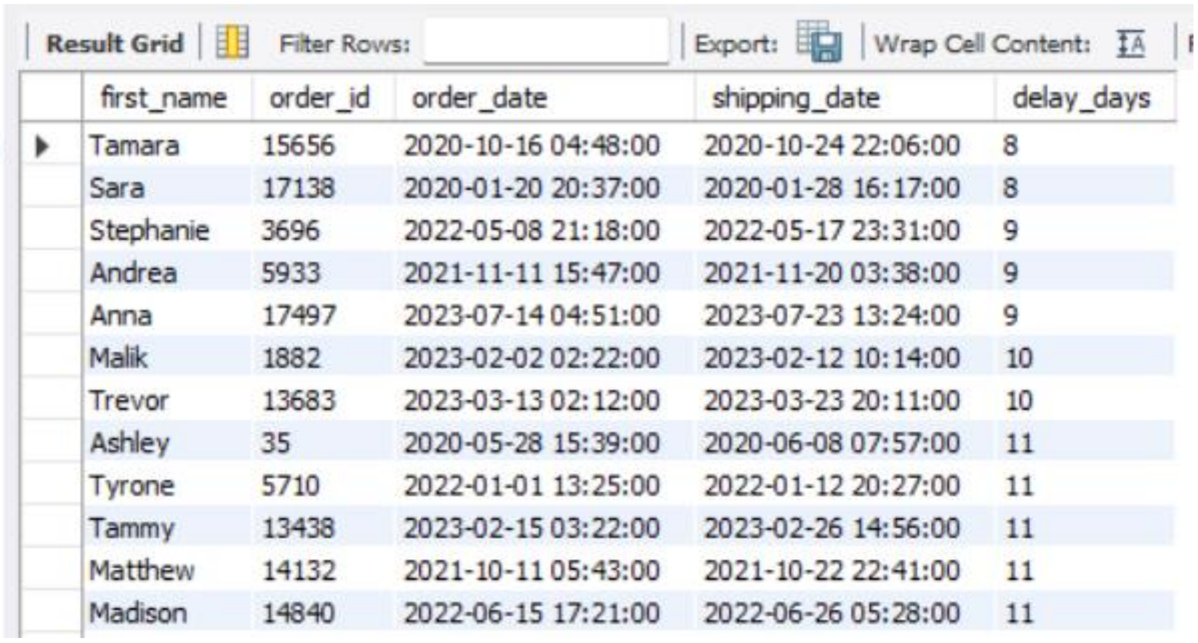
```
SELECT customer_id, First_name, timestampdiff(MONTH,created_at,now()) monthdiff

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:



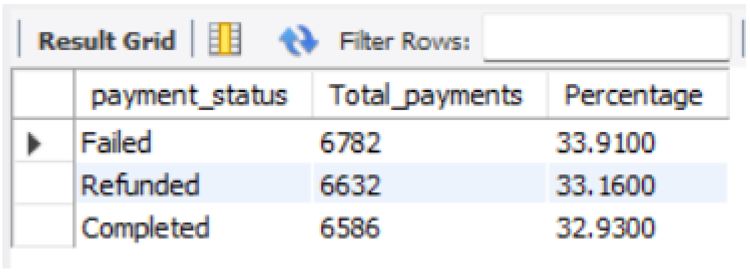
first_name	order_id	order_date	shipping_date	delay_days
Tamara	15656	2020-10-16 04:48:00	2020-10-24 22:06:00	8
Sara	17138	2020-01-20 20:37:00	2020-01-28 16:17:00	8
Stephanie	3696	2022-05-08 21:18:00	2022-05-17 23:31:00	9
Andrea	5933	2021-11-11 15:47:00	2021-11-20 03:38:00	9
Anna	17497	2023-07-14 04:51:00	2023-07-23 13:24:00	9
Malik	1882	2023-02-02 02:22:00	2023-02-12 10:14:00	10
Trevor	13683	2023-03-13 02:12:00	2023-03-23 20:11:00	10
Ashley	35	2020-05-28 15:39:00	2020-06-08 07:57:00	11
Tyrone	5710	2022-01-01 13:25:00	2022-01-12 20:27:00	11
Tammy	13438	2023-02-15 03:22:00	2023-02-26 14:56:00	11
Matthew	14132	2021-10-11 05:43:00	2021-10-22 22:41:00	11
Madison	14840	2022-06-15 17:21:00	2022-06-26 05:28:00	11

/* 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:



payment_status	Total_payments	Percentage
Failed	6782	33.9100
Refunded	6632	33.1600
Completed	6586	32.9300

/* 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

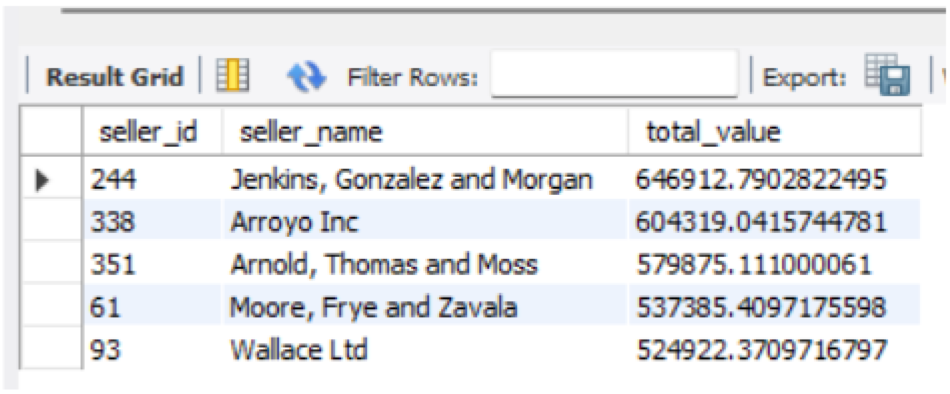
INNER JOIN order_items ON order_items.product_id=Products.product_id

GROUP BY Sellers.seller_id, sellers.seller_name

ORDER BY total_value DESC

LIMIT 5;

Output:



seller_id	seller_name	total_value
244	Jenkins, Gonzalez and Morgan	646912.7902822495
338	Arroyo Inc	604319.0415744781
351	Arnold, Thomas and Moss	579875.111000061
61	Moore, Frye and Zavala	537385.4097175598
93	Wallace Ltd	524922.3709716797

/* 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,

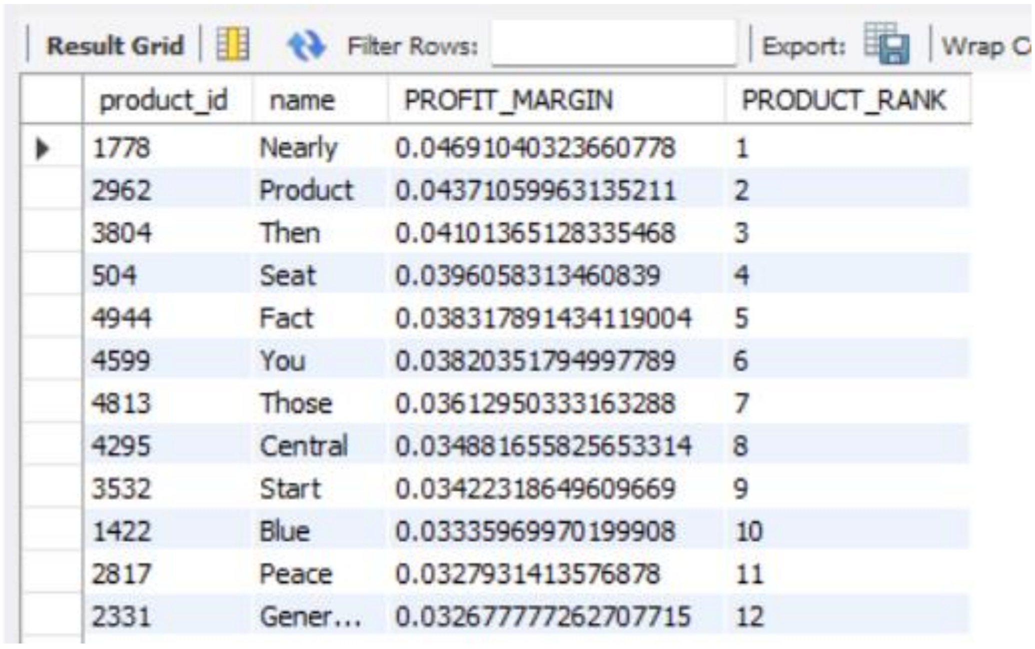
DENSE_RANK() OVER(ORDER BY (SUM(order_items.price-products.price)/(SELECT SUM(order_items.price) FROM order_items)) * 100 DESC) PRODUCT_RANK

FROM Products

INNER JOIN order_items ON Products.product_id=order_items.product_id

GROUP BY products.product_id, products.name;

Output:



product_id	name	PROFIT_MARGIN	PRODUCT_RANK
1778	Nearly	0.04691040323660778	1
2962	Product	0.04371059963135211	2
3804	Then	0.04101365128335468	3
504	Seat	0.0396058313460839	4
4944	Fact	0.038317891434119004	5
4599	You	0.03820351794997789	6
4813	Those	0.03612950333163288	7
4295	Central	0.034881655825653314	8
3532	Start	0.03422318649609669	9
1422	Blue	0.03335969970199908	10
2817	Peace	0.0327931413576878	11
2331	Gener...	0.032677777262707715	12

/* 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

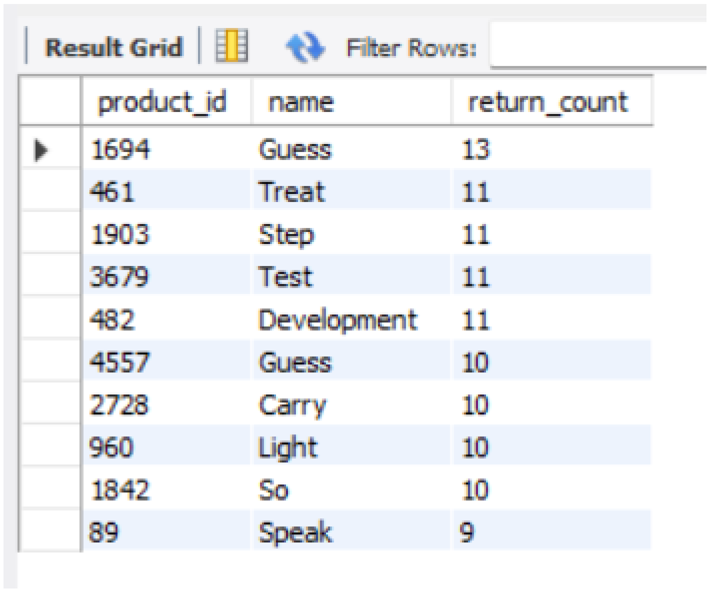
where shipping_status='Returned'

GROUP BY products.product_id, products.name

ORDER BY return_count DESC

LIMIT 10:

Output:


Result Grid | Filter Rows:

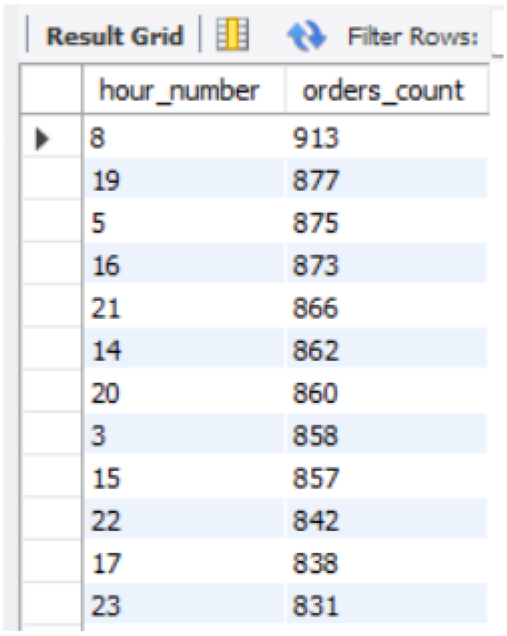
product_id	name	return_count
1694	Guess	13
461	Treat	11
1903	Step	11
3679	Test	11
482	Development	11
4557	Guess	10
2728	Carry	10
960	Light	10
1842	So	10
89	Speak	9

/* 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:


Result Grid | Filter Rows:

hour_number	orders_count
8	913
19	877
5	875
16	873
21	866
14	862
20	860
3	858
15	857
22	842
17	838
23	831

/* 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

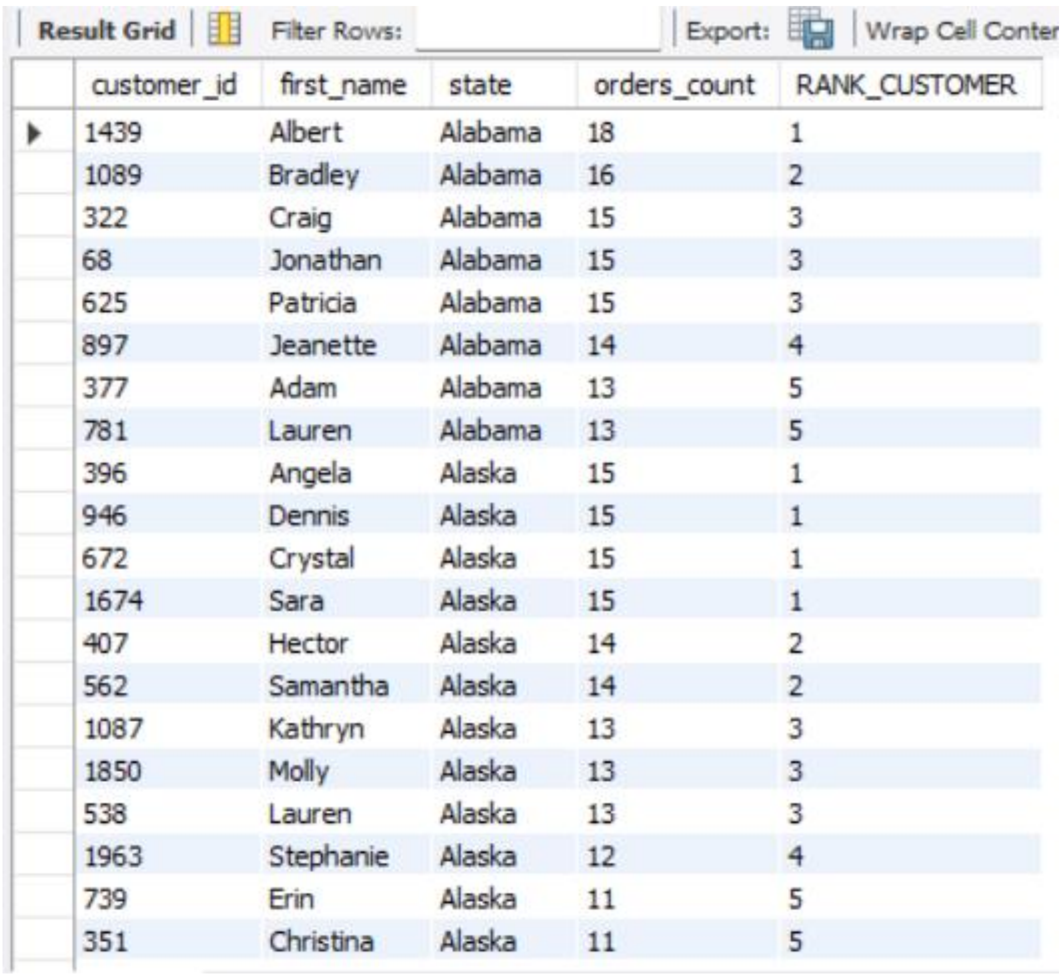
GROUP BY customers.customer_id, first_name, state

)

SELECT * FROM customer_rank

WHERE RANK_CUSTOMER <=5;

Output:



customer_id	first_name	state	orders_count	RANK_CUSTOMER
1439	Albert	Alabama	18	1
1089	Bradley	Alabama	16	2
322	Craig	Alabama	15	3
68	Jonathan	Alabama	15	3
625	Patricia	Alabama	15	3
897	Jeanette	Alabama	14	4
377	Adam	Alabama	13	5
781	Lauren	Alabama	13	5
396	Angela	Alaska	15	1
946	Dennis	Alaska	15	1
672	Crystal	Alaska	15	1
1674	Sara	Alaska	15	1
407	Hector	Alaska	14	2
562	Samantha	Alaska	14	2
1087	Kathryn	Alaska	13	3
1850	Molly	Alaska	13	3
538	Lauren	Alaska	13	3
1963	Stephanie	Alaska	12	4
739	Erin	Alaska	11	5
351	Christina	Alaska	11	5

/* 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

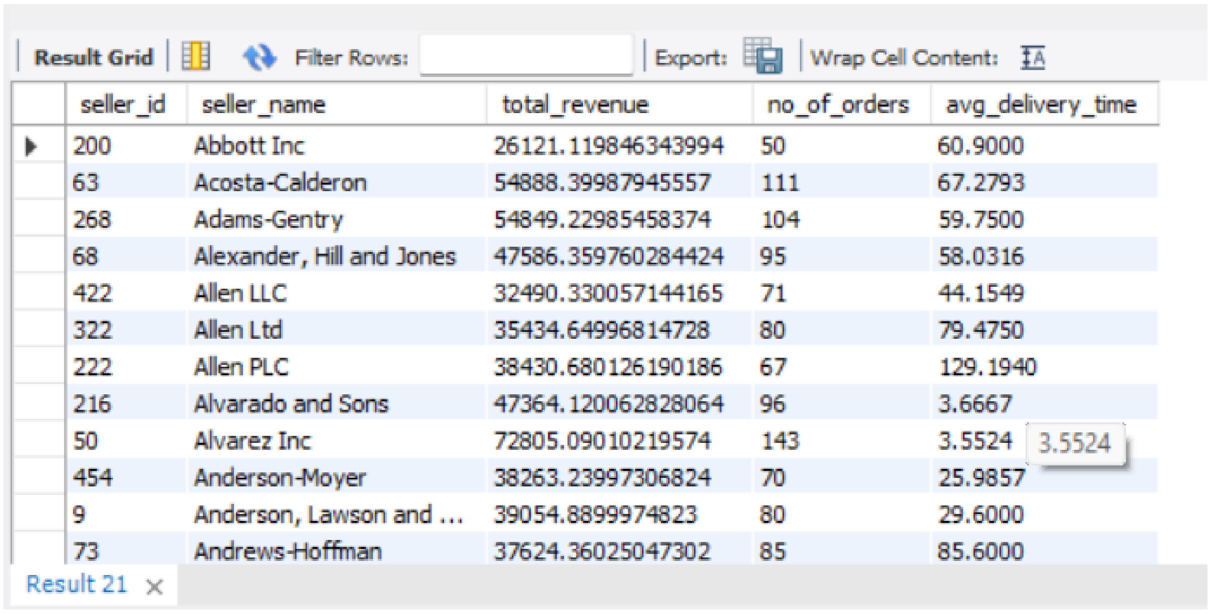
INNER JOIN orders ON order_items.order_id=orders.order_id

INNER JOIN shipping ON orders.shipping_id=shipping.shipping_id

GROUP BY sellers.seller_id, sellers.seller_name

ORDER BY sellers.seller_name;

Output:



seller_id	seller_name	total_revenue	no_of_orders	avg_delivery_time
200	Abbott Inc	26121.119846343994	50	60.9000
63	Acosta-Calderon	54888.39987945557	111	67.2793
268	Adams-Gentry	54849.22985458374	104	59.7500
68	Alexander, Hill and Jones	47586.359760284424	95	58.0316
422	Allen LLC	32490.330057144165	71	44.1549
322	Allen Ltd	35434.64996814728	80	79.4750
222	Allen PLC	38430.680126190186	67	129.1940
216	Alvarado and Sons	47364.120062828064	96	3.6667
50	Alvarez Inc	72805.09010219574	143	3.5524
454	Anderson-Moyer	38263.23997306824	70	25.9857
9	Anderson, Lawson and ...	39054.8899974823	80	29.6000
73	Andrews-Hoffman	37624.36025047302	85	85.6000

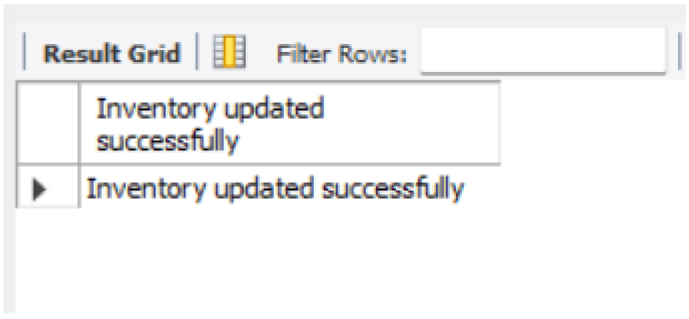
/* 20. Create a function that the same quantity should be reduced from the inventory table as soon as the product is sold.

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

SQL Query:

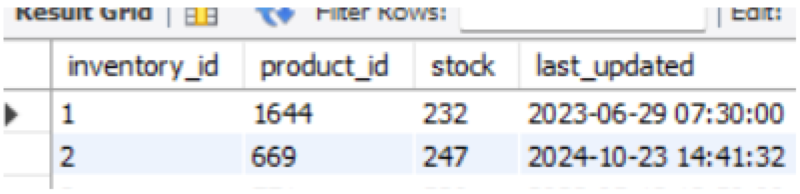
```
DELIMITER $$

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:



```
Inventory updated
successfully
Inventory updated successfully
```

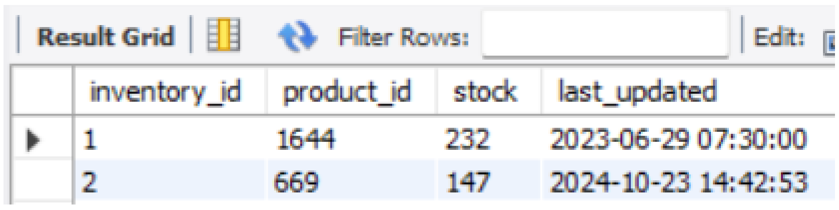
Before Updating:



Result Grid | Filter Rows: | Edit:

	inventory_id	product_id	stock	last_updated
▶	1	1644	232	2023-06-29 07:30:00
	2	669	247	2024-10-23 14:41:32

After Updating:



Result Grid | Filter Rows: | Edit:

	inventory_id	product_id	stock	last_updated
▶	1	1644	232	2023-06-29 07:30:00
	2	669	147	2024-10-23 14:42:53