

RESTAURANT

RESTAURANT ORDER ANALYSIS

ASHEL

TOTAL ORDERS



223
225
230



TOTAL ORDERS

TOTAL 3180
REDD 390
DIAE 4380



TOTAL REVENUE

329-3.90



TOP-SELLING ITEMS



TOTAL ORDERS



TOTAL ORDERS



TOP-REVENUE



TOTAL REVENUE

TOTAL ORDERS



COP-SELLING ITEMS



2.29.910

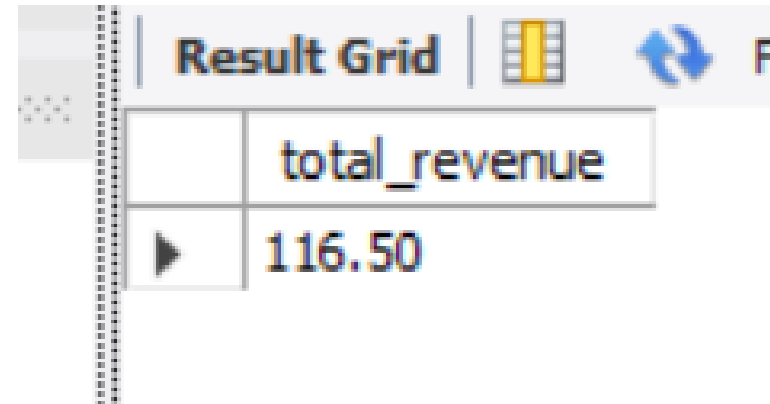
TOP-SELLING ITEMS



4:6.50

1. Calculate the total revenue generated from all orders.

```
SELECT SUM(total_price) AS  
total_revenue  
FROM orders;
```



A screenshot of a database query result grid. The grid has a header row with the column name 'total_revenue' and a data row with the value '116.50'. The grid is titled 'Result Grid' and includes icons for refreshing and exporting.

	total_revenue
▶	116.50

2. Count the total number of orders placed?

```
SELECT COUNT(order_id) AS  
total_orders FROM orders;
```

	total_orders
▶	4

3.Retrieve all orders with their respective order details using INNER JOIN?

```
SELECT o.order_id,  
o.customer_id, o.order_date,  
od.menu_item, od.quantity,  
od.price FROM orders  
oINNER JOIN order_details  
od ON o.order_id =  
od.order_id;
```

	order_id	customer_id	order_date	menu_item	quantity	price
▶	1	101	2025-03-01	Burger	2	12.75
	1	101	2025-03-01	Fries	1	5.00
	2	102	2025-03-01	Pizza	1	18.75
	3	103	2025-03-02	Pasta	2	20.00
	4	101	2025-03-03	Salad	1	10.00
	4	101	2025-03-03	Steak	1	22.25

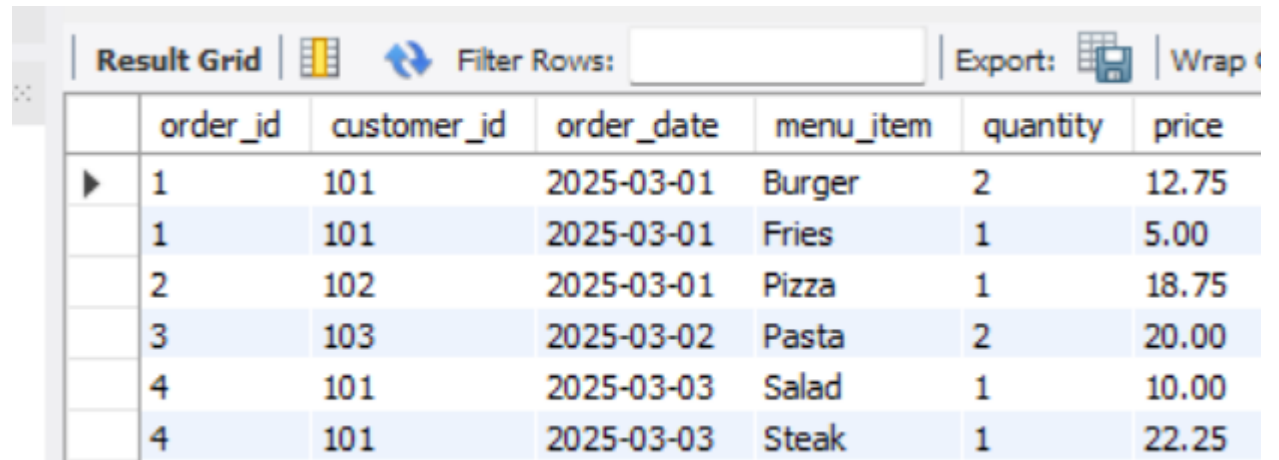
.4. Retrieve all orders, including those without order details, using LEFT JOIN?

```
SELECT o.order_id,  
o.customer_id, o.order_date,  
od.menu_item, od.quantity,  
od.price FROM orders oLEFT  
JOIN order_details od ON  
o.order_id = od.order_id;
```

Result Grid						
Filter Rows: <input type="text"/>						
Export: <input type="text"/> Wrap Cel						
	order_id	customer_id	order_date	menu_item	quantity	price
▶	1	101	2025-03-01	Burger	2	12.75
	1	101	2025-03-01	Fries	1	5.00
	2	102	2025-03-01	Pizza	1	18.75
	3	103	2025-03-02	Pasta	2	20.00
	4	101	2025-03-03	Salad	1	10.00
	4	101	2025-03-03	Steak	1	22.25

5. Retrieve all order details, including those without corresponding orders, using RIGHT JOIN.

```
SELECT o.order_id,  
o.customer_id, o.order_date,  
od.menu_item, od.quantity,  
od.price FROM orders oRIGHT  
JOIN order_details od ON  
o.order_id = od.order_id;
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid displays the results of a SQL query. The columns are: order_id, customer_id, order_date, menu_item, quantity, and price. The data is as follows:

	order_id	customer_id	order_date	menu_item	quantity	price
▶	1	101	2025-03-01	Burger	2	12.75
	1	101	2025-03-01	Fries	1	5.00
	2	102	2025-03-01	Pizza	1	18.75
	3	103	2025-03-02	Pasta	2	20.00
	4	101	2025-03-03	Salad	1	10.00
	4	101	2025-03-03	Steak	1	22.25

6. Calculate the total quantity of each menu item sold.

```
SELECT menu_item,  
SUM(quantity) AS  
total_quantity_sold FROM  
order_details GROUP BY  
menu_item;
```

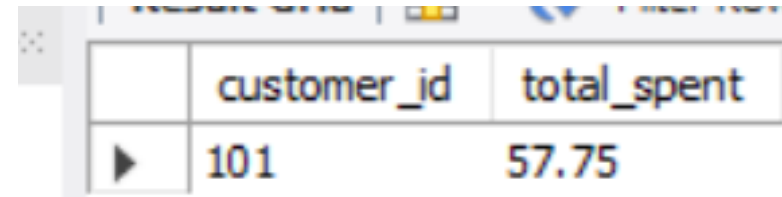


A screenshot of a database query result window. The window has a title bar with 'Results of SQL' and a 'Filter Results' button. Below the title bar is a table with two columns: 'menu_item' and 'total_quantity_sold'. The table contains six rows of data, with alternating light blue and white background colors for each row. The first row is highlighted with a mouse cursor. To the left of the table, there is a vertical toolbar with icons for expand, collapse, and refresh.

	menu_item	total_quantity_sold
▶	Burger	2
	Fries	1
	Pizza	1
	Pasta	2
	Salad	1
	Steak	1

7. Identify the top1 customers who have spent the most

```
SELECT customer_id,  
SUM(total_price) AS  
total_spent FROM orders  
GROUP BY customer_id  
ORDER BY total_spent DESC  
LIMIT 1;
```

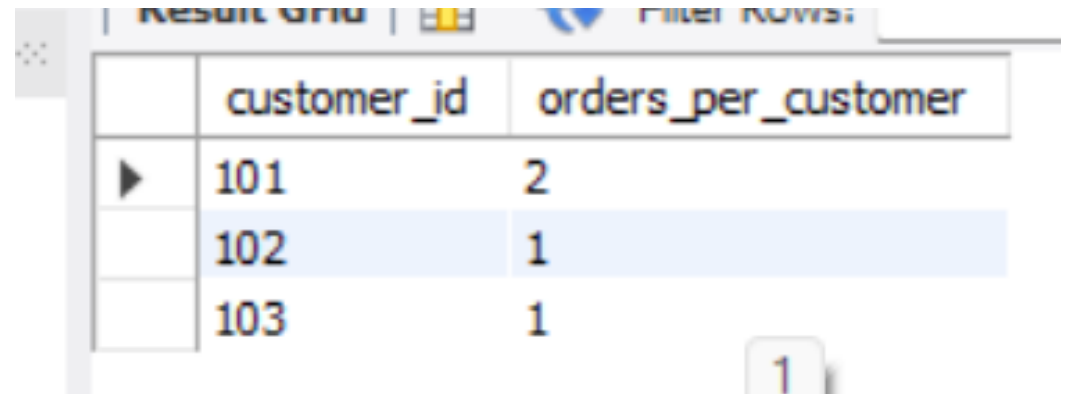


A screenshot of a database query result window. The window has a title bar with a close button and a search icon. Below the title bar is a table with two columns: 'customer_id' and 'total_spent'. The first row of the table shows the value '101' under 'customer_id' and '57.75' under 'total_spent'. A small black triangle icon is visible in the first column of the first row.

	customer_id	total_spent
▶	101	57.75

8.Count the number of orders per customer.

```
SELECT customer_id,  
COUNT(order_id) AS  
orders_per_customer FROM  
orders GROUP BY  
customer_id;
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains three columns: 'customer_id' and 'orders_per_customer'. There are three rows of data. The first row has '101' and '2'. The second row has '102' and '1'. The third row has '103' and '1'. The second row is highlighted in blue. A small '1' is visible in the bottom right corner of the grid area.

	customer_id	orders_per_customer
▶	101	2
	102	1
	103	1

9.find the total sum of price ?

```
select sum(total_price)  
from orders;
```

	sum(total_price)
▶	116.50

10. Find the date with the highest total sales.

```
SELECT order_date,  
SUM(total_price) AS  
total_sales FROM orders  
GROUP BY order_date  
ORDER BY total_sales DESC  
LIMIT 1;
```

	order_date	total_sales
▶	2025-03-01	44.25

11. Find the month with the highest revenue

```
SELECT DATE_FORMAT(order_date,  
'%Y-%m') AS month, SUM(total_price)  
AS revenue FROM orders GROUP BY  
month ORDER BY revenue DESC LIMIT 1;
```

	month	revenue
▶	2025-03	116.50

12. Find the total revenue generated each day

```
SELECT order_date,  
SUM(total_price) AS  
daily_revenue FROM orders GROUP  
BY order_date ORDER BY  
order_date;
```

	order_date	daily_revenue
▶	2025-03-01	44.25
	2025-03-02	40.00
	2025-03-03	32.25

13. Find customers who placed their first order in the last 30 days

```
SELECT customer_id,  
MIN(order_date) AS  
first_order_date FROM orders  
GROUP BY customer_id HAVING  
first_order_date >= CURDATE() -  
INTERVAL 30 DAY;
```

	customer_id	first_order_date
▶	101	2025-03-01
	102	2025-03-01
	103	2025-03-02

14.Retrieve the top 3
busiest order hours

```
SELECT HOUR(order_time) AS  
order_hour, COUNT(order_id) AS  
total_orders FROM orders GROUP  
BY order_hour ORDER BY  
total_orders DESC LIMIT 3;
```

	order_hour	total_orders
▶	12	1
	13	1
	19	1

15. Find the most expensive order food ?

```
SELECT * FROM orders  
ORDER BY total_price DESC  
LIMIT 1;
```

	order_id	customer_id	order_date	order_time	total_price
▶	3	103	2025-03-02	19:45:00	40.00
✱	NULL	NULL	NULL	NULL	NULL

16. List all foreign keys in the database

```
SELECT TABLE_NAME, COLUMN_NAME,  
CONSTRAINT_NAME,  
REFERENCED_TABLE_NAME,  
REFERENCED_COLUMN_NAME FROM  
INFORMATION_SCHEMA.KEY_COLUMN  
_USAGE WHERE  
REFERENCED_TABLE_SCHEMA =  
'RestaurantDB';
```

	TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
▶	order_details	order_id	order_details_ibfk_1	orders	order_id

17. Find all orders that have corresponding order details (Primary and Foreign Key Relation)

```
SELECT o.order_id, o.order_date,  
o.total_price, od.menu_item,  
od.quantity, od.price FROM orders  
o JOIN order_details od ON  
o.order_id = od.order_id;
```



The screenshot shows a database query result window titled 'Result 8'. It displays a table with 7 columns: order_id, order_date, total_price, menu_item, quantity, and price. The table contains 5 rows of data, representing the results of a join between the 'orders' and 'order_details' tables. The first row shows order_id 1 with a total price of 25.50, which is composed of 1 Fries (5.00). The second row shows order_id 2 with a total price of 18.75, composed of 1 Pizza (18.75). The third row shows order_id 3 with a total price of 40.00, composed of 2 Pasta (20.00). The fourth row shows order_id 4 with a total price of 32.25, composed of 1 Salad (10.00). The fifth row shows order_id 4 with a total price of 32.25, composed of 1 Steak (22.25). The table is displayed in a light blue and white striped format.

	order_id	order_date	total_price	menu_item	quantity	price
	1	2025-03-01	25.50	Fries	1	5.00
	2	2025-03-01	18.75	Pizza	1	18.75
	3	2025-03-02	40.00	Pasta	2	20.00
	4	2025-03-03	32.25	Salad	1	10.00
	4	2025-03-03	32.25	Steak	1	22.25

18.Count how many orders
have at least one order detail

```
SELECT COUNT(DISTINCT  
order_id) AS valid_orders FROM  
order_details;
```

	valid_orders
▶	4

19. Identify orders where the total price does not match the sum of item prices (Data Integrity Check)

```
SELECT o.order_id, o.total_price,  
SUM(od.price * od.quantity) AS  
calculated_price FROM orders o JOIN  
order_details od ON o.order_id =  
od.order_id GROUP BY o.order_id,  
o.total_price HAVING o.total_price <>  
calculated_price;
```

	order_id	total_price	calculated_price
▶	1	25.50	30.50

20. Find the most expensive order based on total price
(Primary Key Usage)

```
SELECT * FROM orders WHERE  
total_price = (SELECT MAX(total_price)  
FROM orders);
```

	order_id	customer_id	order_date	order_time	total_price
▶	3	103	2025-03-02	19:45:00	40.00
✱	NULL	NULL	NULL	NULL	NULL




21.Retrieve the total revenue grouped by each customer ID

```
SELECT customer_id,  
SUM(total_price) AS total_spent  
FROM orders GROUP BY customer_id  
ORDER BY total_spent DESC;
```

	customer_id	total_spent
▶	101	57.75
	103	40.00
	102	18.75

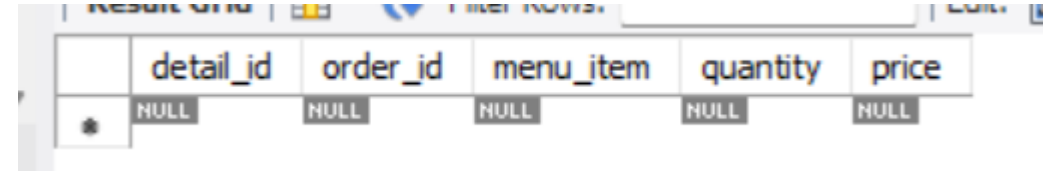
22.Retrieve all orders placed on a specific date.

```
SELECT * FROM orders WHERE  
order_date = '2025-03-01';
```

Result Grid   Filter Rows: <input type="text"/>						Edit: 	
	order_id	customer_i	order_date	order_time	total_price		
▶	1	101	2025-03-01	12:30:00	25.50		
	2	102	2025-03-01	13:15:00	18.75		
✱	NULL	NULL	NULL	NULL	NULL		

23.Retrieve the order details of a specific order ID (e.g., 101).

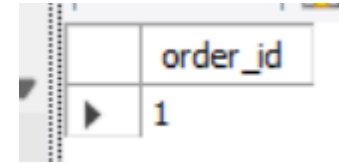
```
SELECT * FROM order_details  
WHERE order_id = 101;
```



	detail_id	order_id	menu_item	quantity	price
*	NULL	NULL	NULL	NULL	NULL

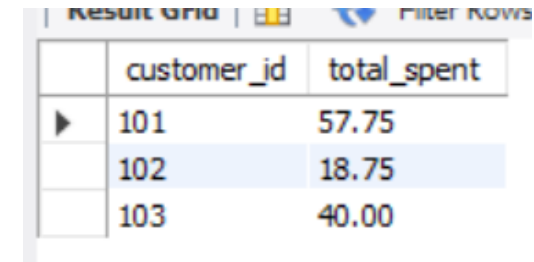
24. Identify orders that contain a specific menu item (e.g., 'Burger').

Identify orders that contain a specific menu item (e.g., 'Burger').



Find the total amount spent by each customer

```
SELECT customer_id,  
SUM(total_price) AS total_spent  
FROM orders GROUP BY  
customer_id;
```



The screenshot shows a database interface with a 'Result Grid' tab. It displays the results of the SQL query, showing three rows of data. The first row is for customer_id 101 with a total_spent of 57.75. The second row is for customer_id 102 with a total_spent of 18.75. The third row is for customer_id 103 with a total_spent of 40.00. The second row is highlighted in blue.

	customer_id	total_spent
▶	101	57.75
	102	18.75
	103	40.00

26.Retrieve order details of a specific menu item efficiently

```
SELECT * FROM order_details WHERE  
menu_item = 'Pizza';
```

Result Grid					
	detail_id	order_id	menu_item	quantity	price
▶	3	2	Pizza	1	18.75
•	NULL	NULL	NULL	NULL	NULL

27Get the latest order placed
by each customer

```
SELECT o1.*  
FROM orders o1  
WHERE order_date = (  
    SELECT MAX(order_date)  
    FROM orders o2  
    WHERE o1.customer_id =  
    o2.customer_id  
);
```

	order_id	customer_id	order_date	order_time	total_price
▶	2	102	2025-03-01	13:15:00	18.75
	3	103	2025-03-02	19:45:00	40.00
	4	101	2025-03-03	14:20:00	32.25
*	NULL	NULL	NULL	NULL	NULL

28.Retrieve all orders along
with their corresponding
order details

```
SELECT o.order_id, o.customer_id,  
o.order_date, od.menu_item, od.quantity,  
od.price  
FROM orders o  
JOIN order_details od ON o.order_id =  
od.order_id;
```

	order_id	customer_id	order_date	menu_item	quantity	price
▶	1	101	2025-03-01	Burger	2	12.75
	1	101	2025-03-01	Fries	1	5.00
	2	102	2025-03-01	Pizza	1	18.75
	3	103	2025-03-02	Pasta	2	20.00
	4	101	2025-03-03	Salad	1	10.00

29. Find the total revenue per customer

```
SELECT o.customer_id,  
SUM(o.total_price) AS total_spent  
FROM orders o  
GROUP BY o.customer_id;
```

	customer_id	total_spent
▶	101	57.75
	102	18.75
	103	40.00

30. Get a list of all menu items ordered along with their order date

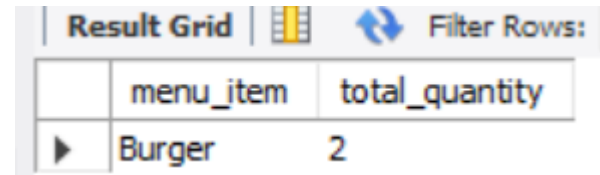
```
SELECT o.order_date, od.menu_item,  
od.quantity FROM orders o JOIN  
order_details od ON o.order_id =  
od.order_id;
```

	order_date	menu_item	quantity
▶	2025-03-01	Burger	2
	2025-03-01	Fries	1
	2025-03-01	Pizza	1
	2025-03-02	Pasta	2
	2025-03-03	Salad	1

Result 11 of 11

31. Get the most frequently ordered menu item

```
SELECT menu_item, SUM(quantity) AS  
total_quantity FROM  
order_details GROUP BY  
menu_item ORDER BY total_quantity  
DESC LIMIT 1;
```



The screenshot shows a 'Result Grid' window with a toolbar containing a 'Filter Rows' button. The grid displays the results of the SQL query, with columns 'menu_item' and 'total_quantity'. The first row shows 'Burger' with a total quantity of 2.

	menu_item	total_quantity
▶	Burger	2

32. Get the order with the highest total price

```
SELECT * FROM orders ORDER BY  
total_price DESC LIMIT 1;
```

	order_id	customer_id	order_date	order_time	total_price
▶	3	103	2025-03-02	19:45:00	40.00
✱	NULL	NULL	NULL	NULL	NULL

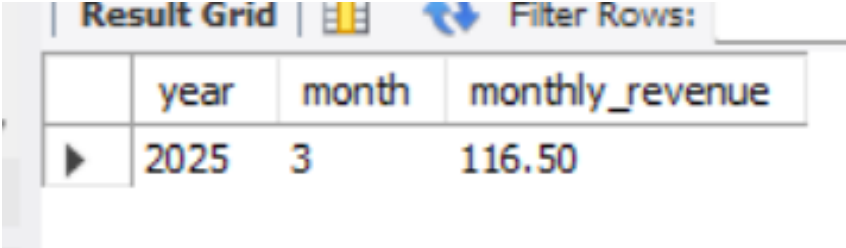
33.Total order value per
each order

```
SELECT order_id, SUM(quantity * price)  
AS total_order_valueFROM  
order_detailsGROUP BY order_id;
```

	order_id	total_order_value
▶	1	30.50
	2	18.75
	3	40.00
	4	32.25

34. Retrieve the total revenue per month

```
SELECT YEAR(order_date) AS year,  
       MONTH(order_date) AS month,  
       SUM(total_price) AS monthly_revenue  
FROM orders  
  
GROUP BY YEAR(order_date),  
         MONTH(order_date)  
  
ORDER BY year DESC, month DESC;
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid has four columns: an empty column, 'year', 'month', and 'monthly_revenue'. A single row of data is displayed with values 2025, 3, and 116.50. Above the grid, there is a 'Filter Rows:' button with a funnel icon.

	year	month	monthly_revenue
▶	2025	3	116.50

35. Find customers who placed consecutive orders on two consecutive days

```
SELECT customer_id, order_date,  
       LAG(order_date, 1) OVER  
       (PARTITION BY customer_id ORDER BY  
        order_date) AS previous_order  
FROM orders;
```

	customer_id	order_date	previous_order
▶	101	2025-03-01	NULL
	101	2025-03-03	2025-03-01
	102	2025-03-01	NULL
	103	2025-03-02	NULL

36. Get the cumulative revenue per day

```
SELECT order_date,  
       SUM(total_price) OVER (ORDER BY  
order_date) AS cumulative_revenue  
FROM orders;
```

	order_date	cumulative_revenue
▶	2025-03-01	44.25
	2025-03-01	44.25
	2025-03-02	84.25
	2025-03-03	116.50

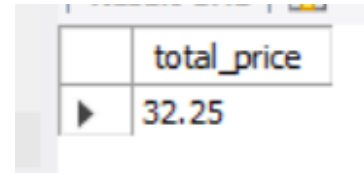
37. Find the moving average of daily sales over the last 7 days

```
SELECT order_date,  
       SUM(total_price) OVER (ORDER BY  
order_date ROWS BETWEEN 6  
PRECEDING AND CURRENT ROW) AS  
moving_avg_sales  
FROM orders;
```

	order_date	moving_avg_sales
▶	2025-03-01	25.50
	2025-03-01	44.25
	2025-03-02	84.25
	2025-03-03	116.50

38.Retrieve the second
highest order total

```
SELECT DISTINCT total_price  
FROM orders  
ORDER BY total_price DESC  
LIMIT 1 OFFSET 1;
```



	total_price
▶	32.25

39. Rank customers based on their total spending

```
SELECT customer_id, SUM(total_price)
AS total_spent, RANK() OVER
(ORDER BY SUM(total_price) DESC) AS
spending_rank FROM orders GROUP BY
customer_id;
```

	customer_id	total_spent	spending_rank
▶	101	57.75	1
	103	40.00	2
	102	18.75	3

40. Find customers who have placed at least 2 orders

```
SELECT customer_id, COUNT(order_id)  
AS order_count FROM orders GROUP BY  
customer_id HAVING COUNT(order_id)  
>= 2;
```

	customer_id	order_count
▶	101	2

41.Retrieve orders with the highest quantity of a single menu item

```
SELECT order_id, menu_item,  
MAX(quantity) AS highest_quantity  
FROM order_details  
GROUP BY order_id, menu_item  
ORDER BY highest_quantity DESC  
LIMIT 1;
```

	order_id	menu_item	highest_quantity
▶	1	Burger	2

42. Find the total revenue generated by each day of the week

```
SELECT DAYNAME(order_date) AS  
day_of_week, SUM(total_price) AS  
total_revenue  
FROM orders  
GROUP BY DAYNAME(order_date)  
ORDER BY total_revenue DESC;
```

	day_of_week	total_revenue
▶	Saturday	44.25
	Sunday	40.00
	Monday	32.25

43. Find the most popular menu item for each customer

```
SELECT customer_id, menu_item,  
SUM(quantity) AS total_ordered  
FROM orders o  
JOIN order_details od ON o.order_id =  
od.order_id  
GROUP BY customer_id, menu_item  
ORDER BY customer_id, total_ordered  
DESC;
```

	customer_id	menu_item	total_ordered
►	101	Burger	2
	101	Fries	1
	101	Salad	1
	101	Steak	1
	102	Pizza	1

Result 28 ▼

44.Retrieve customers who
have spent more than the
average order value

```
SELECT customer_id, SUM(total_price)  
AS total_spent  
FROM orders  
GROUP BY customer_id  
HAVING SUM(total_price) > (SELECT  
AVG(total_price) FROM orders);
```

	customer_id	total_spent
▶	101	57.75
	103	40.00

45. Retrieve customers who
have ordered at least 2
different menu items

```
SELECT order_id  
FROM  
order_details  
GROUP BY  
order_id  
HAVING COUNT(DISTINCT  
menu_item) >= 2;
```

	order_id
▶	1
	4

46. Find the Total spent the order id

```
SELECT customer_id, total_spent FROM  
(  SELECT customer_id,  
SUM(total_price) AS total_spent,  
RANK() OVER (ORDER BY  
SUM(total_price) DESC) AS  
spending_rank  FROM orders  
GROUP BY customer_id)  
ranked_customers WHERE  
spending_rank = 2;
```

	customer_id	total_spent
▶	103	40.00

47. Find the percentage contribution of each customer to total revenue

```
SELECT customer_id,  
       SUM(total_price) AS  
customer_revenue,  
       (SUM(total_price) * 100) / (SELECT  
SUM(total_price) FROM orders) AS  
revenue_percentage  
FROM orders  
GROUP BY customer_id;
```

	customer_id	customer_revenue	revenue_percentage
▶	101	57.75	49.570815
	102	18.75	16.094421
	103	40.00	34.334764

48. Find customers who placed an order within 30 days of their first order

```
SELECT customer_id, order_id,  
order_date  
FROM orders o1  
WHERE EXISTS (  
    SELECT 1 FROM orders o2  
    WHERE o1.customer_id =  
o2.customer_id  
    AND o2.order_date BETWEEN  
o1.order_date AND  
DATE_ADD(o1.order_date, INTERVAL  
30 DAY)  
);
```

	customer_id	order_id	order_date
▶	101	1	2025-03-01
	102	2	2025-03-01
⋮	103	3	2025-03-02
	101	4	2025-03-03
*	NULL	NULL	NULL

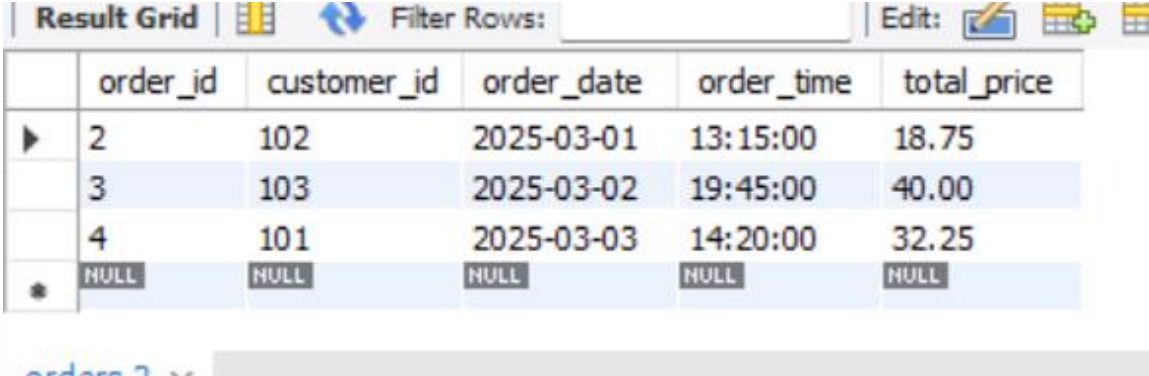
49. Retrieve customers who placed orders on both weekdays and weekends

```
SELECT customer_id
FROM orders
GROUP BY customer_id
HAVING COUNT(DISTINCT CASE WHEN
DAYOFWEEK(order_date) IN (1,7) THEN
1 END) > 0
AND COUNT(DISTINCT CASE WHEN
DAYOFWEEK(order_date) NOT IN (1,7)
THEN 1 END) > 0;
```

	customer_id
▶	101

50.Retrieve the most recent order for each customer

```
SELECT o.*  
FROM orders o  
WHERE order_date = (  
    SELECT MAX(order_date)  
    FROM orders o2  
    WHERE o.customer_id =  
o2.customer_id  
);
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains a table with 6 columns: order_id, customer_id, order_date, order_time, and total_price. There are 4 data rows and one row with NULL values. The interface includes a 'Filter Rows' search bar and an 'Edit' button.

	order_id	customer_id	order_date	order_time	total_price
▶	2	102	2025-03-01	13:15:00	18.75
	3	103	2025-03-02	19:45:00	40.00
	4	101	2025-03-03	14:20:00	32.25
*	NULL	NULL	NULL	NULL	NULL

RESTAURANT
ORDER
ANALYTICS



ORDER ANALYTICS

THANK YOU
FOR CHOOSING OUR

Restaurant



8%

