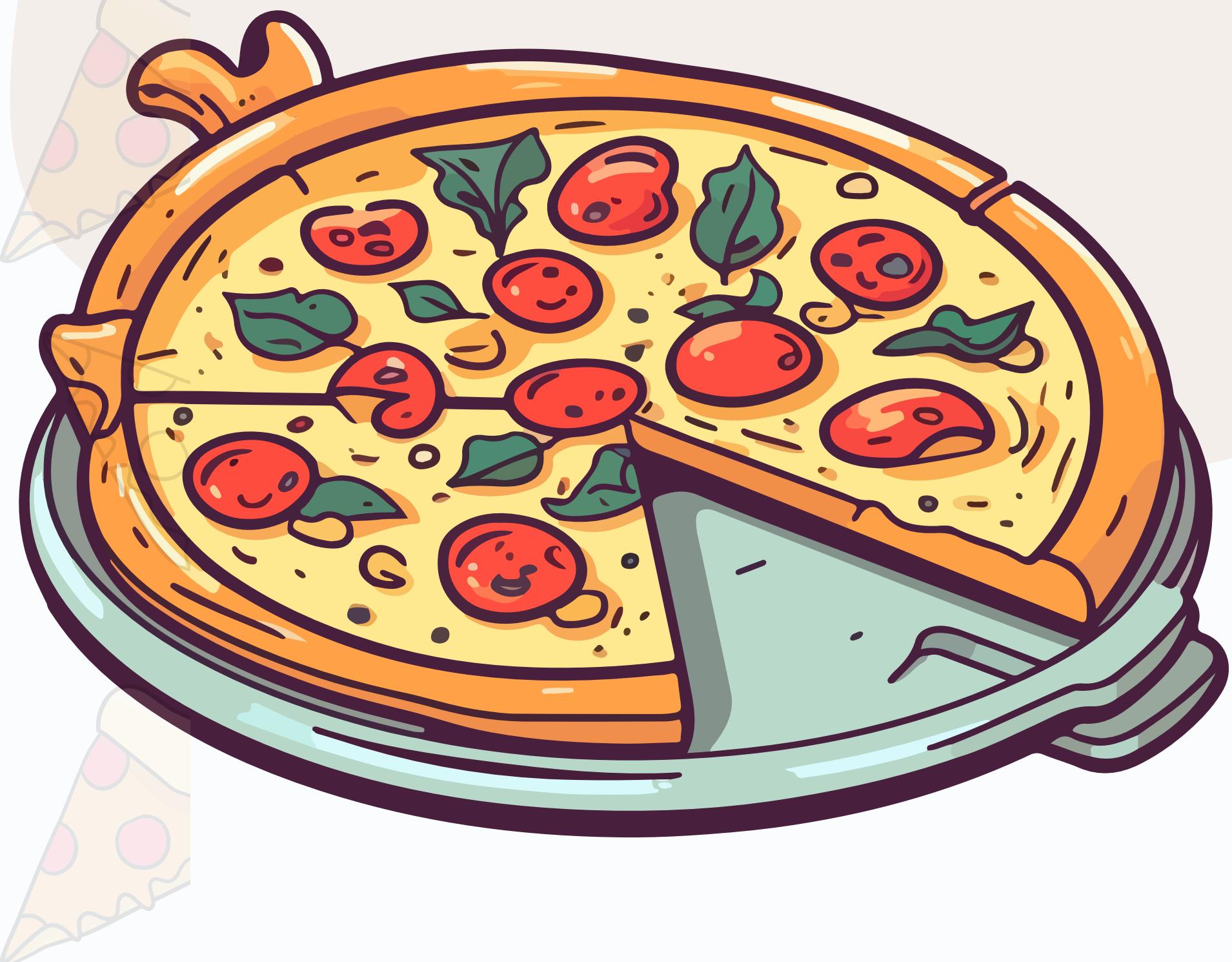


Analyzing Pizza Sales Data with Real-World Queries

A hands-on [SQL Project](#)
exploring insights from a pizza
sales dataset



Why Pizza Sales Data?

Analyzing pizza sales data offers a great opportunity to apply SQL in solving real-world business problems. Understanding sales performance, customer preferences, and revenue patterns is critical for any business aiming to improve profitability and operational efficiency.

By answering questions such as the total number of orders, the most common pizza sizes, and category-wise revenue contributions, this project mimics the type of analysis businesses need to make data-driven decisions.

The goal is to demonstrate how SQL can be used not just for retrieving data but for generating actionable insights that help businesses optimize their offerings, identify trends, and ultimately enhance decision-making.

Table of Contents

Key Queries Covered in This Project



- 01 Sales Performance Analysis**
Identifying sales trends, evaluating team performance, and comparing historical sales data to improve forecasting.
- 02 Customer Behavior Insights**
Analyzing customer purchase patterns, preferences, and behaviors to enhance segmentation and improve marketing strategies.
- 03 Revenue Trends**
Monitoring revenue growth, key drivers, profit margins, and providing insights for strategic financial planning.
- 04 Operational Efficiency Metrics**
Assessing process efficiency, resource utilization, and cost-effectiveness to optimize overall business performance.

01. Retrieve the total number of orders placed

This query calculates the total number of orders placed in the dataset to understand the overall sales volume.

The screenshot shows a MySQL Workbench interface. The query editor window contains the following code:

```
1 -- Que 1: the total number of orders placed
2
3 • SELECT count(order_id) as total_order FROM orders;
4
```

The result grid shows a single row of data:

total_order
21350

The status bar at the bottom indicates "Result 5" and "Read Only".

02. Calculate the total revenue generated from pizza sales

This query sums up the total revenue by multiplying the price of each pizza by the quantity sold, providing insights into the business's total earnings.

The screenshot shows a MySQL Workbench interface with two tabs at the top: 'Que 1' and 'Que 2'. The 'Que 2' tab is active, displaying the following SQL query:

```
1 -- Que 2: Calculate the total revenue generated from pizza sales.
2
3 • SELECT
4     ROUND(SUM(order_details.quantity * pizzas.price),
5           2) AS total_revenue
6
7 FROM
8     order_details
9     JOIN
10    pizzas ON pizzas.pizza_id = order_details.pizza_id
```

Below the query editor is a 'Result Grid' pane. It contains a single row with one column labeled 'total_revenue' and a value of '817860.05'.

total_revenue
817860.05

03. Identify the highest-priced pizza

This query retrieves the highest-priced pizza from the menu, helping the business identify premium offerings.

The screenshot shows a MySQL Workbench interface. The top bar has tabs for 'Que 1', 'Que 2', and 'Que 3' (which is selected). Below the tabs is a toolbar with various icons. The main area contains a numbered SQL query:

```
1 -- Que 3: Identify the highest-priced pizza.
2
3 • SELECT
4     pizza_types.name, pizzas.price
5 FROM
6     pizza_types
7         JOIN
8     pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
9 ORDER BY pizzas.price DESC
10 LIMIT 1;
```

Below the query, there's a 'Result Grid' section with the following data:

name	price
The Greek Pizza	35.95

04. Identify the most common pizza size ordered

This query counts the number of orders by pizza size to find out which size is most frequently ordered by customers.

The screenshot shows a MySQL Workbench interface with the following details:

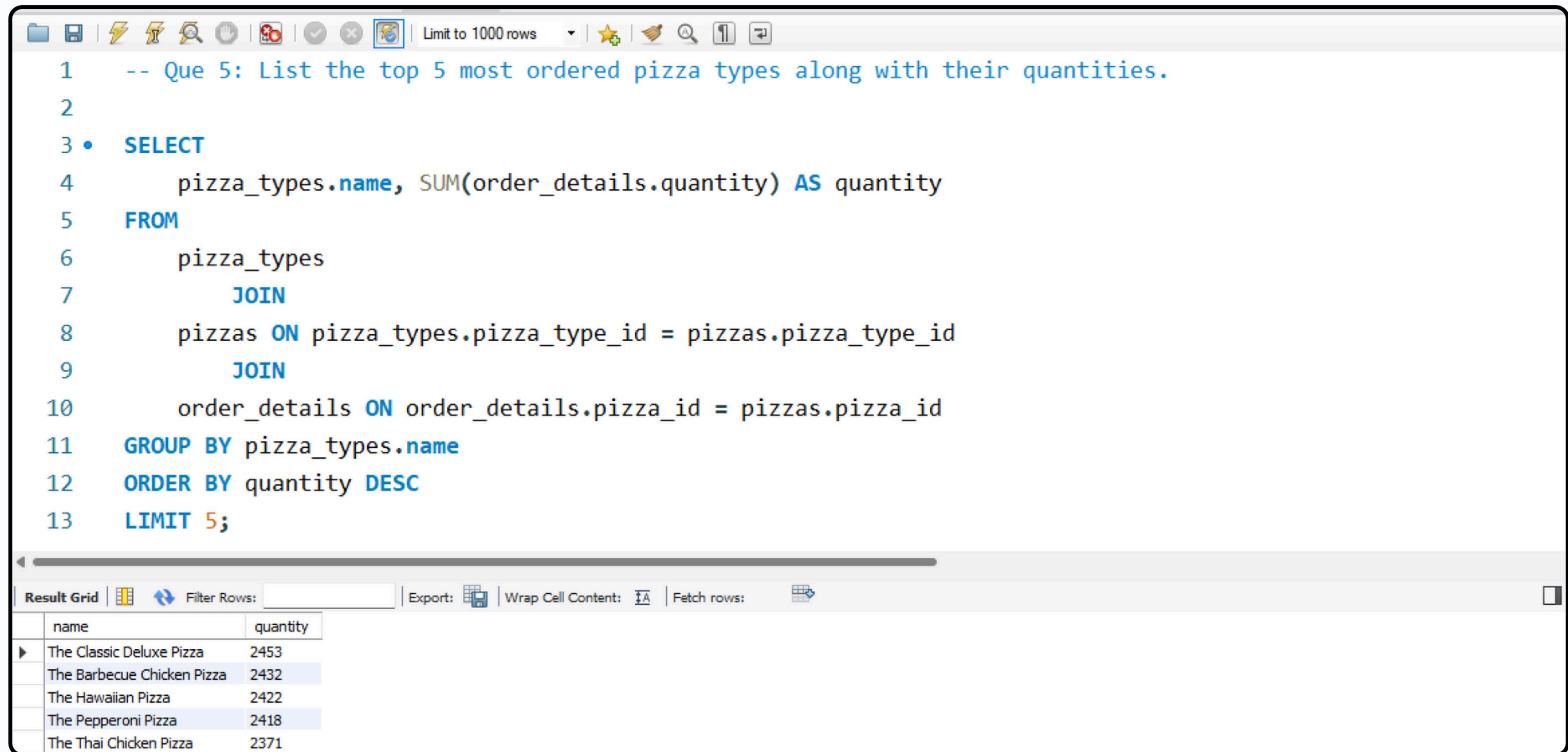
- Query Editor:** Tab titled "Que 4" is active. The query code is:

```
1 -- Que 4: Identify the most common pizza size ordered.
2 • SELECT
3     pizzas.size,
4     COUNT(order_details.order_details_id) AS order_count
5 FROM
6     pizzas
7     JOIN
8         order_details ON pizzas.pizza_id = order_details.pizza_id
9 GROUP BY pizzas.size
10 ORDER BY order_count DESC;
```
- Result Grid:** Shows the results of the query in a tabular format. The columns are "size" and "order_count". The data is:

size	order_count
L	18526
M	15385
S	14137
XL	544
XXL	28
- Bottom Navigation:** Shows "Result 2" and a refresh icon.

05. List the top 5 most ordered pizza types along with their quantities

This query ranks the top 5 pizzas based on the total quantity ordered, helping the business focus on its best-selling items



The screenshot shows a MySQL Workbench interface with the following details:

- Toolbar:** Includes icons for file operations, search, and connection management.
- Query Editor:** Displays the SQL query:

```
1 -- Que 5: List the top 5 most ordered pizza types along with their quantities.
2
3 • SELECT
4     pizza_types.name, SUM(order_details.quantity) AS quantity
5 FROM
6     pizza_types
7     JOIN
8     pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
9     JOIN
10    order_details ON order_details.pizza_id = pizzas.pizza_id
11   GROUP BY pizza_types.name
12   ORDER BY quantity DESC
13   LIMIT 5;
```
- Result Grid:** Shows the resulting data:

name	quantity
The Classic Deluxe Pizza	2453
The Barbecue Chicken Pizza	2432
The Hawaiian Pizza	2422
The Pepperoni Pizza	2418
The Thai Chicken Pizza	2371

06. Find the total quantity of each pizza category ordered

This query joins the necessary tables to aggregate the total quantity of each pizza category, such as vegetarian or non-vegetarian pizzas

```
1  -- Que 6: Join the necessary tables to find the total quantity of each pizza category ordered.  
2  
3 • SELECT  
4      pizza_types.category AS category,  
5      SUM(order_details.quantity) AS quantity  
6  FROM  
7      pizza_types  
8          JOIN  
9      pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id  
10         JOIN  
11      order_details ON order_details.pizza_id = pizzas.pizza_id  
12  GROUP BY category  
13  ORDER BY quantity DESC;
```

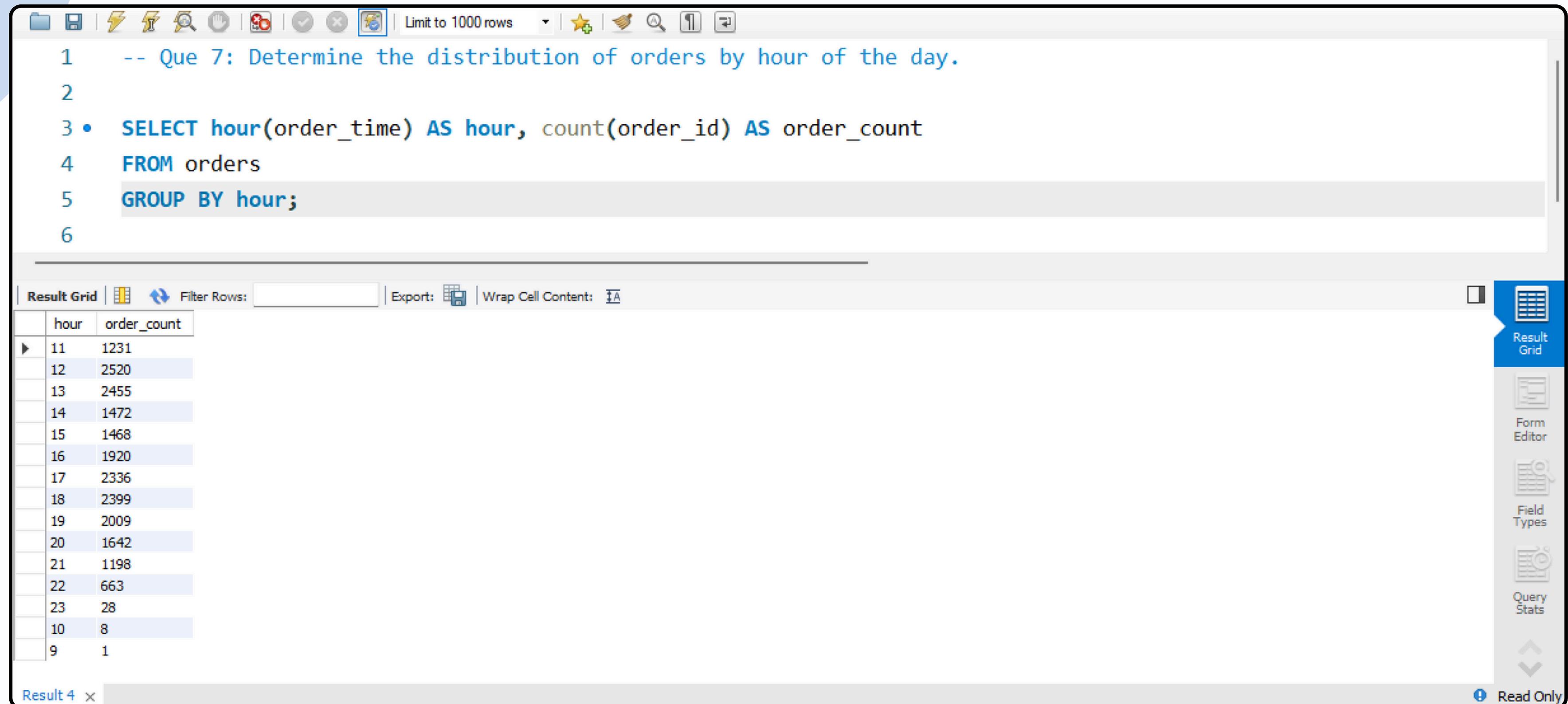
Result Grid | Filter Rows: _____ | Export: _____ | Wrap Cell Content: _____

category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

Result Grid

07. Determine the distribution of orders by hour of the day

This query analyzes the time of day when orders were placed, helping the business identify peak hours for sales



The screenshot shows a MySQL Workbench interface. The query editor window contains the following SQL code:

```
1 -- Que 7: Determine the distribution of orders by hour of the day.  
2  
3 • SELECT hour(order_time) AS hour, count(order_id) AS order_count  
4 FROM orders  
5 GROUP BY hour;  
6
```

The result grid displays the following data:

hour	order_count
11	1231
12	2520
13	2455
14	1472
15	1468
16	1920
17	2336
18	2399
19	2009
20	1642
21	1198
22	663
23	28
10	8
9	1

The interface includes various toolbars and a sidebar with icons for Result Grid, Form Editor, Field Types, and Query Stats.

08. Find the category-wise distribution of pizzas

This query joins relevant tables and calculates the distribution of orders for each pizza category, providing deeper insights into customer preferences.

The screenshot shows a database query editor interface with the following details:

- Toolbar:** Includes icons for file operations, search, and various database functions.
- Query Editor:** Displays the following SQL code:

```
1 -- Que 8: Join relevant tables to find the category-wise distribution of pizzas.
2
3 • SELECT category, count(name) AS pizza_name
4   FROM pizza_types
5   GROUP BY category
6   ORDER BY pizza_name DESC;
7
```
- Result Grid:** Shows the output of the query in a tabular format.

category	pizza_name
Supreme	9
Veggie	9
Classic	8
Chicken	6
- Bottom Right Panel:** A sidebar with buttons for "Result Grid" (which is selected) and "Form Editor".

09. Group orders by date and calculate the average number of pizzas ordered per day

This query groups orders by date and computes the average number of pizzas ordered per day, helping the business understand daily sales trends

```
1 -- Que 9: Group the orders by date and calculate the average number of pizzas ordered per day.  
2  
3 • SELECT  
4     ROUND(AVG(quantity), 0) AS avg_pizza_ordered_per_day  
5 FROM  
6     (SELECT  
7         orders.order_date, SUM(order_details.quantity) AS quantity  
8     FROM  
9         orders  
10    JOIN order_details ON orders.order_id = order_details.order_id  
11    GROUP BY orders.order_date) AS order_quantity;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:	Result Grid
	avg_pizza_ordered_per_day				
▶	138				

10. Determine the top 3 most ordered pizza types based on revenue

This query identifies the top 3 pizza types that generated the highest revenue, offering insights into the most profitable items.

```
1 -- Que 10: Determine the top 3 most ordered pizza types based on revenue.  
2  
3 • SELECT  
4     pizza_types.name,  
5     ROUND(SUM(order_details.quantity * pizzas.price),  
6             2) AS revenue  
7 FROM  
8     pizza_types  
9     JOIN  
10    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id  
11     JOIN  
12    order_details ON order_details.pizza_id = pizzas.pizza_id  
13 GROUP BY pizza_types.name  
14 ORDER BY revenue DESC  
15 LIMIT 3;
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Fetch rows:

	name	revenue
▶	The Thai Chicken Pizza	43434.25
	The Barbecue Chicken Pizza	42768
	The California Chicken Pizza	41409.5

Result Grid

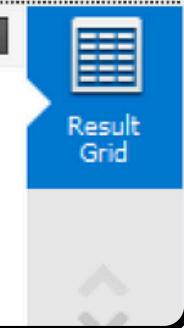
11. Calculate the percentage contribution of each pizza type to total revenue

This query calculates the percentage contribution of each pizza type to the overall revenue, helping the business assess the importance of each product in driving sales.

```
1  -- Que 11: Calculate the percentage contribution of each pizza type to total revenue.
2
3 • SELECT
4      pizza_types.category,
5      ROUND((SUM(order_details.quantity * pizzas.price) / (SELECT
6          ROUND(SUM(order_details.quantity * pizzas.price),
7              2) AS total_revenue
8
9      FROM
10         order_details
11
12         JOIN
13             pizzas ON pizzas.pizza_id = order_details.pizza_id)) * 100,
14
15     2) AS revenue
16
17     FROM
18         pizza_types
19
20         JOIN
21             pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
22
23         JOIN
24             order_details ON order_details.pizza_id = pizzas.pizza_id
25
26     GROUP BY pizza_types.category
27
28     ORDER BY revenue DESC;
```

Result Grid | Filter Rows: Export: Wrap Cell Content:

	category	revenue
▶	Classic	26.91
	Supreme	25.46
	Chicken	23.96
	Veggie	23.68



12. Analyze the cumulative revenue generated over time

This query tracks cumulative revenue over time to show how sales have grown, providing valuable insights for trend analysis.

The screenshot shows a database query editor window with the following details:

- Query Text:**

```
1 -- Que 12: Analyze the cumulative revenue generated over time.
2
3 • SELECT
4     order_date,
5     sum(revenue) over (ORDER BY order_date) as cumu_revenue
6 FROM
7
8 (SELECT
9     orders.order_date,
10    SUM(order_details.quantity * pizzas.price) AS revenue
11 FROM
12    order_details
13      JOIN
14    pizzas ON order_details.pizza_id = pizzas.pizza_id
15      JOIN
16    orders ON orders.order_id = order_details.order_id
17 GROUP BY orders.order_date) As sales;
```
- Result Grid:**

order_date	cumu_revenue
2015-01-01	2713.8500000000004
2015-01-02	5445.75
2015-01-03	8108.15
2015-01-04	9863.6
2015-01-05	11929.55
2015-01-06	14358.5
2015-01-07	16560.7
2015-01-08	19399.05
2015-01-09	21526.4
- Toolbar:** Includes standard icons for file operations, search, and refresh.
- Right Panel:** Shows a sidebar with navigation links: Result Grid (selected), Form Editor, and Field Types.

13. Determine the top 3 most ordered pizza types based on revenue for each category

This query identifies the top 3 pizza types with the highest revenue in each category, offering a more granular view of profitability across different product lines.

```
1  -- Que 13: Determine the top 3 most ordered pizza types based on revenue for each pizza category.  
2  
3 • SELECT  
4      category, name, revenue  
5  FROM  
6  (SELECT  
7      category, name, revenue,  
8      rank() over (PARTITION BY category ORDER BY revenue DESC) AS rn  
9  FROM  
10 (SELECT  
11     pizza_types.category,  
12     pizza_types.name,  
13     SUM(order_details.quantity * pizzas.price) AS revenue  
14   FROM  
15     pizza_types  
16       JOIN  
17     pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id  
18       JOIN  
19     order_details ON order_details.pizza_id = pizzas.pizza_id  
20   GROUP BY pizza_types.category , pizza_types.name) AS a) AS b  
21 WHERE rn <=3;  
22
```

Result Grid | Filter Rows: Export: Wrap Cell Content:

The screenshot shows a database interface with a results grid. The grid has columns for 'category', 'name', and 'revenue'. The data is as follows:

category	name	revenue
Chicken	The Thai Chicken Pizza	43434.25
Chicken	The Barbecue Chicken Pizza	42768
Chicken	The California Chicken Pizza	41409.5
Classic	The Classic Deluxe Pizza	38180.5
Classic	The Hawaiian Pizza	32273.25
Classic	The Pepperoni Pizza	30161.75

Result Grid | Form Editor

Challenges Faced & Lessons Learned

Overcoming complex queries and unlocking key business insights through hands-on SQL practice



01

Handling Data Complexity

Joining multiple tables and managing relationships accurately was crucial for category-wise and revenue-based queries. This improved my understanding of complex joins and table structures.

02

Managing Aggregations

Writing queries with GROUP BY and HAVING clauses for meaningful insights taught me how to handle advanced aggregations effectively.

Challenges Faced & Lessons Learned

Overcoming complex queries and unlocking key business insights through hands-on SQL practice



03

Optimizing Query Performance

Ensuring that queries ran efficiently on large datasets helped me learn optimization techniques and best practices for performance tuning.

04

Date and Time Analysis

Working with time-based queries, such as analyzing order distribution by hour and cumulative revenue over time, enhanced my skills in using SQL date and time functions.

05

Deriving Actionable Insights:

Interpreting the query results and translating them into meaningful business insights, such as identifying peak sales periods and high-revenue products, strengthened my analytical thinking.

Conclusion

“ This project provided a hands-on experience in applying SQL to a real-world dataset, enhancing both my technical and analytical skills. By working through various levels of complexity—from basic metrics to advanced revenue analysis—I gained a deeper understanding of SQL’s powerful role in data-driven decision-making. It also reinforced the importance of clean, efficient query writing and interpreting results to derive actionable insights.

Overall, this project not only improved my proficiency in SQL techniques such as joins, aggregations, and time-based analysis but also demonstrated how data analysis can help businesses optimize their operations, identify trends, and maximize profitability.



Thank you



[advittiple](https://www.linkedin.com/in/advittiple)



advittiple.connect@gmail.com



[advittiple](https://github.com/advittiple)