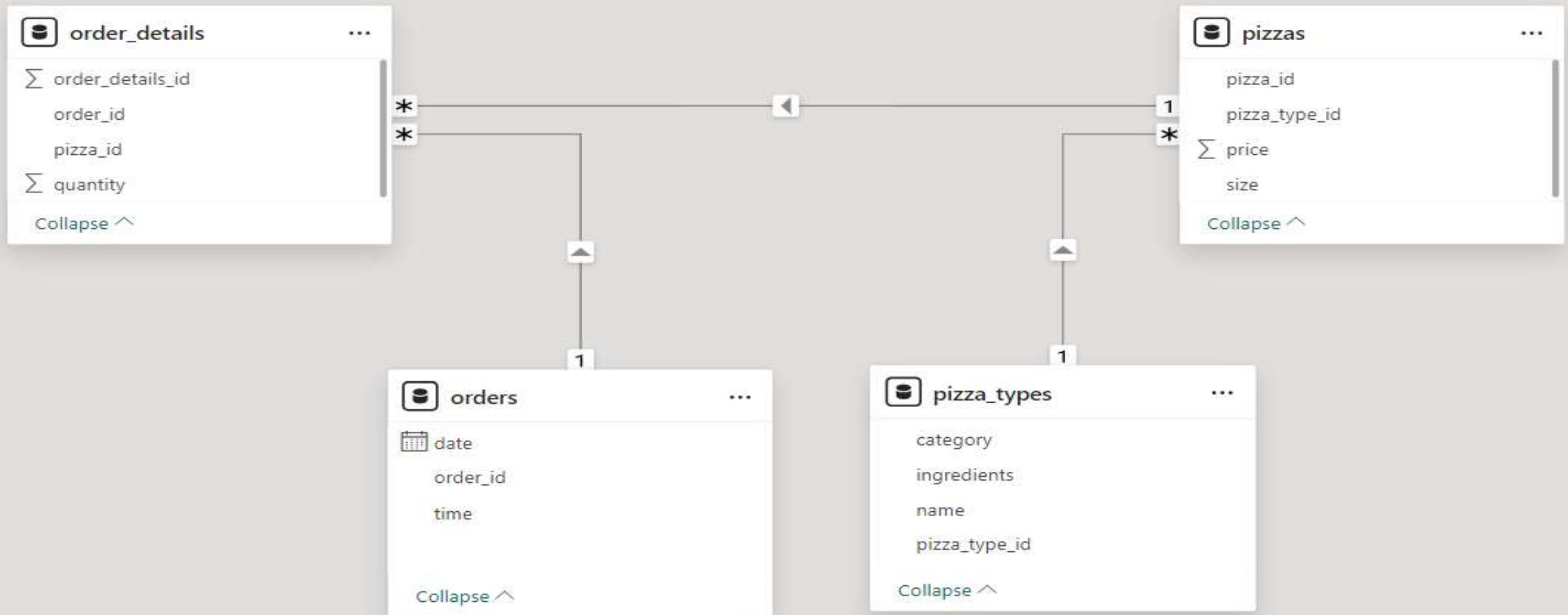


Project:

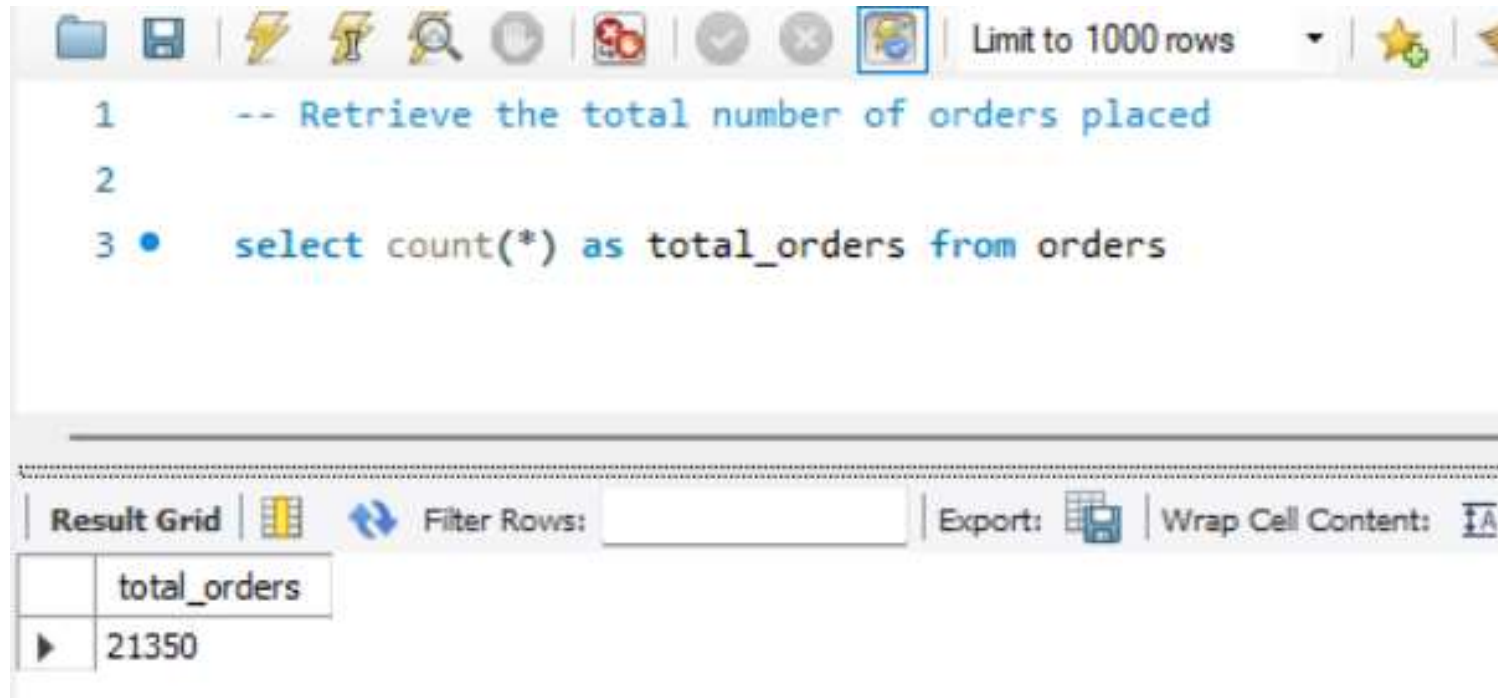
Slicing Through the Data: Insights from Pizza Sales Analysis with SQL

- ❑ In this project, we used SQL to analyze data from a Pizza Hut dataset and extracted insights by solving problem statements.
- ❑ **The Pizzahut dataset contained four tables: pizzas, pizza_types, orders, and order_details.**



Problem Statement: 1

RETRIEVE THE TOTAL NUMBER OF ORDERS PLACED.



The screenshot shows a SQL query editor interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query text is as follows:

```
1  -- Retrieve the total number of orders placed
2
3  •  select count(*) as total_orders from orders
```

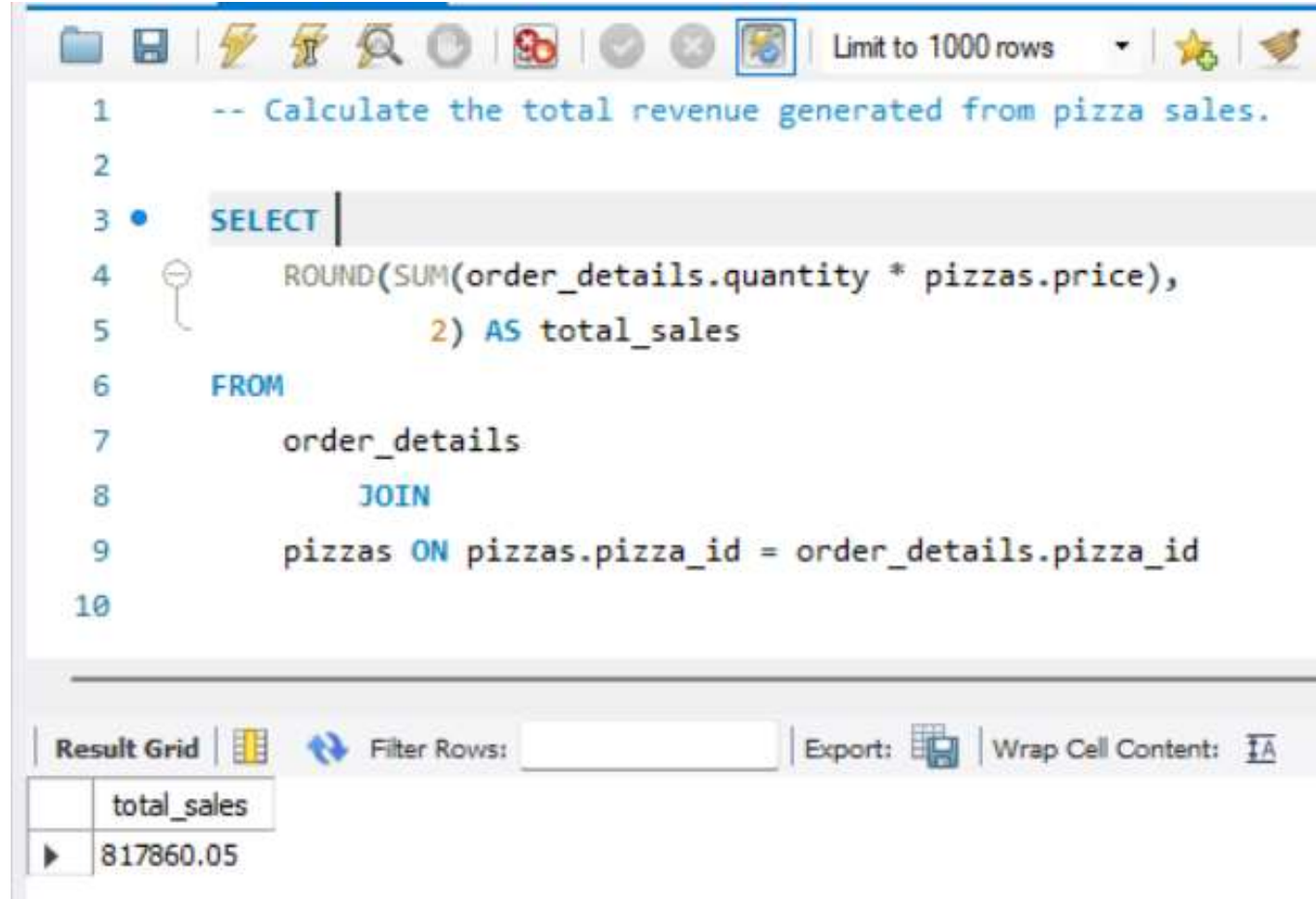
Below the query editor is a horizontal separator line. At the bottom, there is a 'Result Grid' section with a table containing one row of data:

	total_orders
▶	21350

Additional controls in the bottom right include a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' toggle.

Problem Statement: 2

CALCULATE THE TOTAL REVENUE GENERATED FROM PIZZA SALES.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

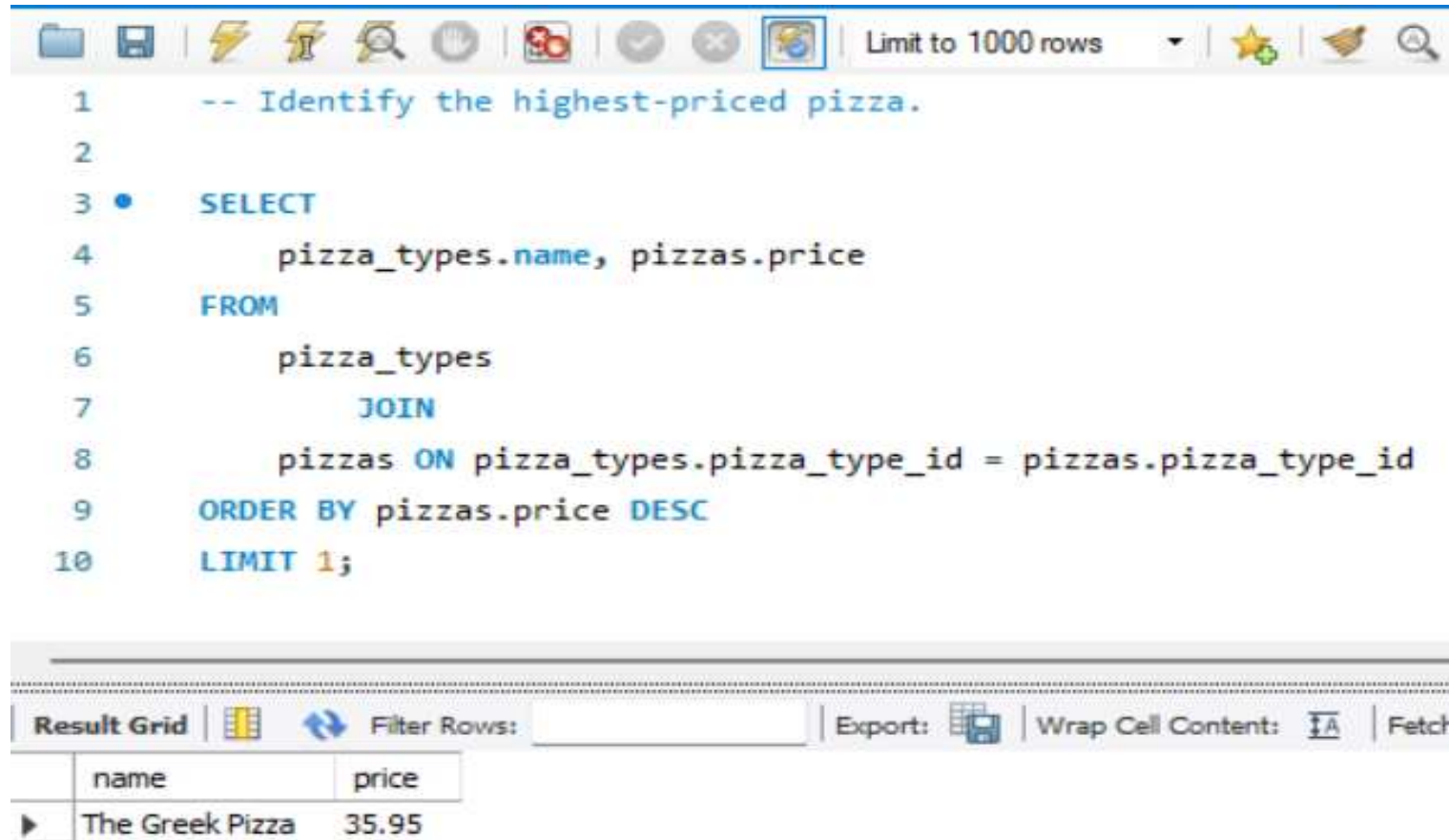
```
1  -- Calculate the total revenue generated from pizza sales.
2
3  SELECT
4      ROUND(SUM(order_details.quantity * pizzas.price),
5             2) AS total_sales
6  FROM
7      order_details
8      JOIN
9      pizzas ON pizzas.pizza_id = order_details.pizza_id
10
```

Below the editor, the 'Result Grid' tab is active, displaying the query results in a table:

total_sales
817860.05

Problem Statement: 3

IDENTIFY THE HIGHEST-PRICED PIZZA.



The image shows a SQL query editor interface. The top toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. The query is as follows:

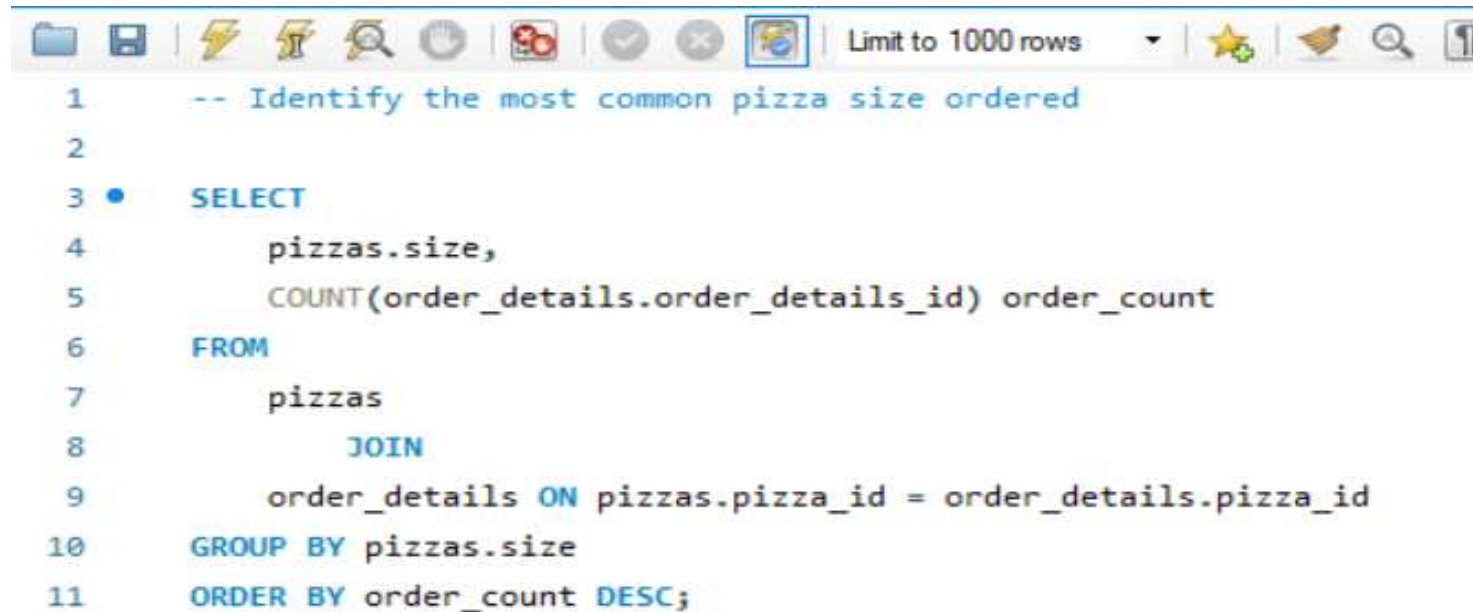
```
1  -- Identify the highest-priced pizza.
2
3  •  SELECT
4      pizza_types.name, pizzas.price
5  FROM
6      pizza_types
7      JOIN
8      pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
9  ORDER BY pizzas.price DESC
10  LIMIT 1;
```

Below the query editor is the 'Result Grid' section. It features a toolbar with 'Filter Rows', 'Export', 'Wrap Cell Content', and 'Fetch' options. The results are displayed in a table with two columns: 'name' and 'price'.

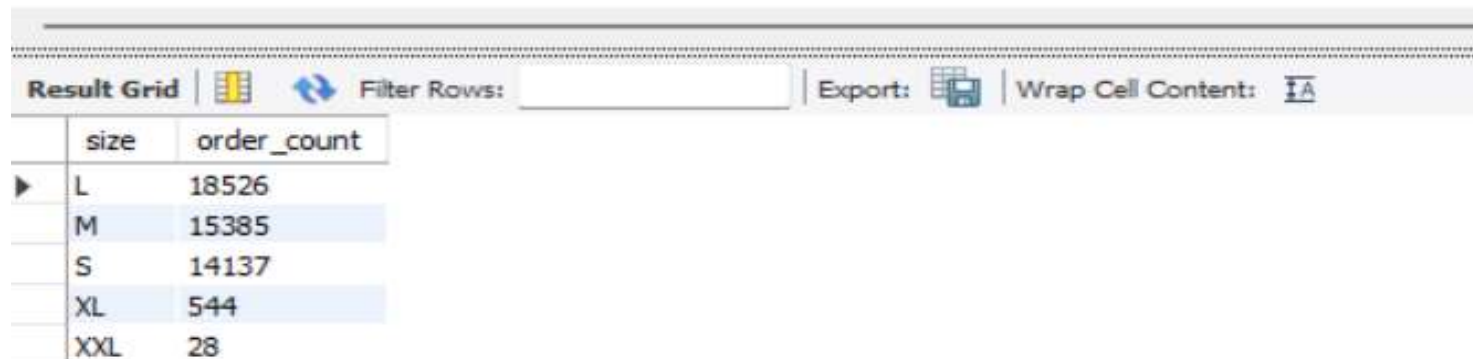
	name	price
▶	The Greek Pizza	35.95

Problem Statement: 4

IDENTIFY THE MOST COMMON PIZZA SIZE ORDERED.



```
1  -- Identify the most common pizza size ordered
2
3  •  SELECT
4      pizzas.size,
5      COUNT(order_details.order_details_id) order_count
6  FROM
7      pizzas
8      JOIN
9      order_details ON pizzas.pizza_id = order_details.pizza_id
10 GROUP BY pizzas.size
11 ORDER BY order_count DESC;
```



	size	order_count
▶	L	18526
	M	15385
	S	14137
	XL	544
	XXL	28

Problem Statement: 5

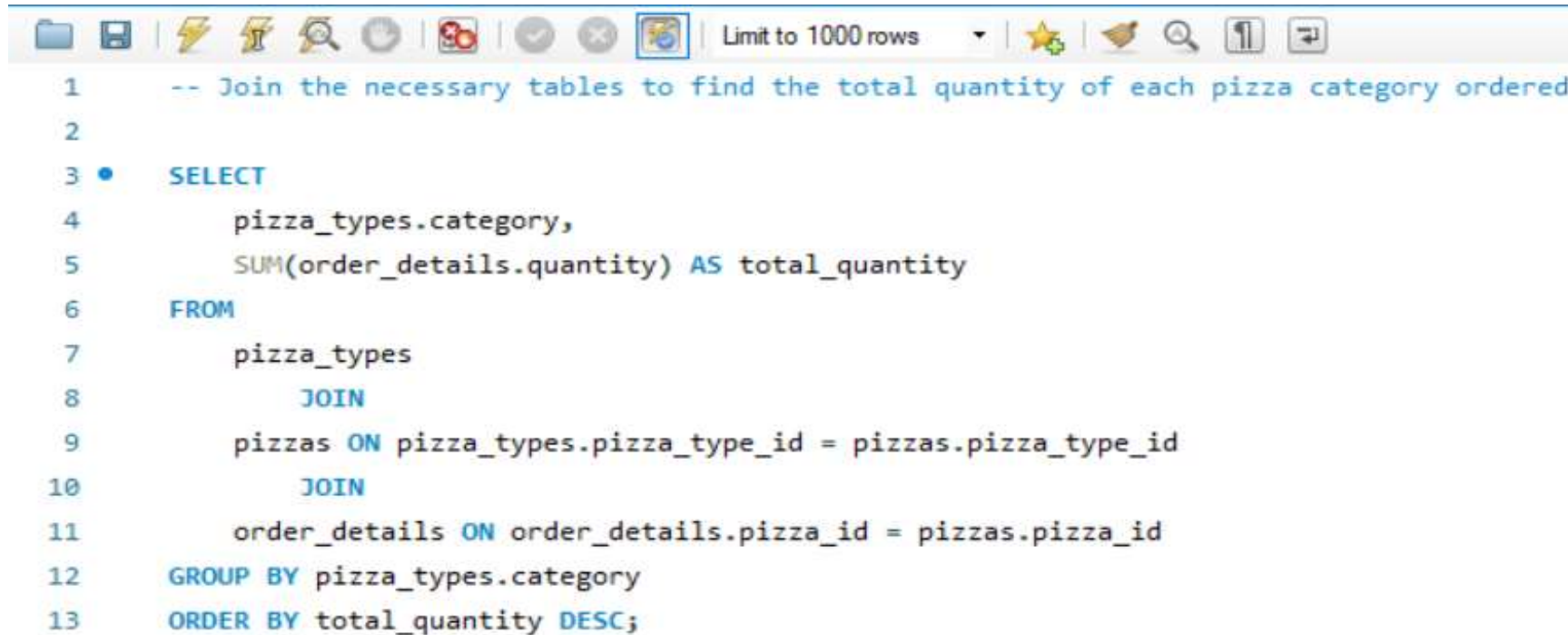
LIST THE TOP 5 MOST ORDERED PIZZA TYPES ALONG WITH THEIR QUANTITIES.

```
1  -- List the top 5 most ordered pizza types along with their quantit
2
3  •  SELECT
4      pizza_types.name,
5      SUM(order_details.quantity) AS total_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 pizza_types.name
13 ORDER BY total_quantity DESC
14 LIMIT 5;
```

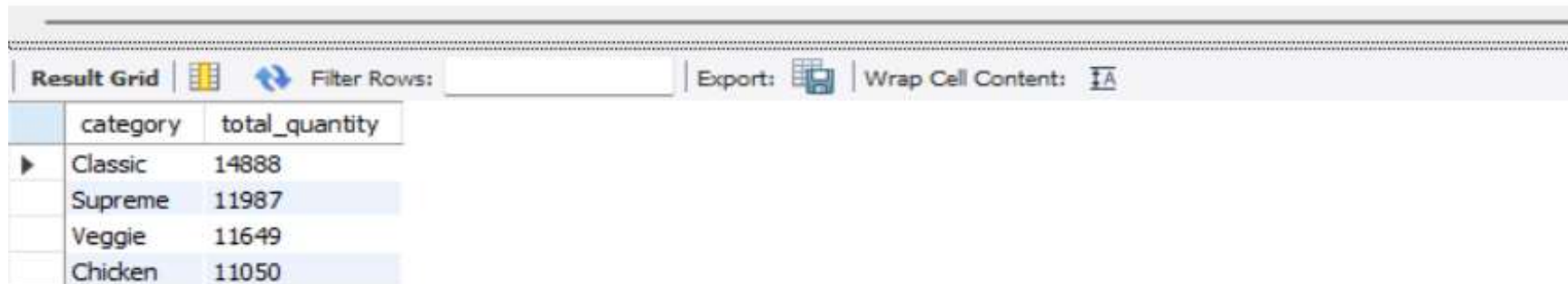
Result Grid			Filter Rows:	Export:	Wrap Cell Content:	Fetch rows
	name	total_quantity				
▶	The Classic Deluxe Pizza	2453				
	The Barbecue Chicken Pizza	2432				
	The Hawaiian Pizza	2422				
	The Pepperoni Pizza	2418				
	The Thai Chicken Pizza	2371				

Problem Statement: 6

JOIN THE NECESSARY TABLES TO FIND THE TOTAL QUANTITY OF EACH PIZZA CATEGORY ORDERED.



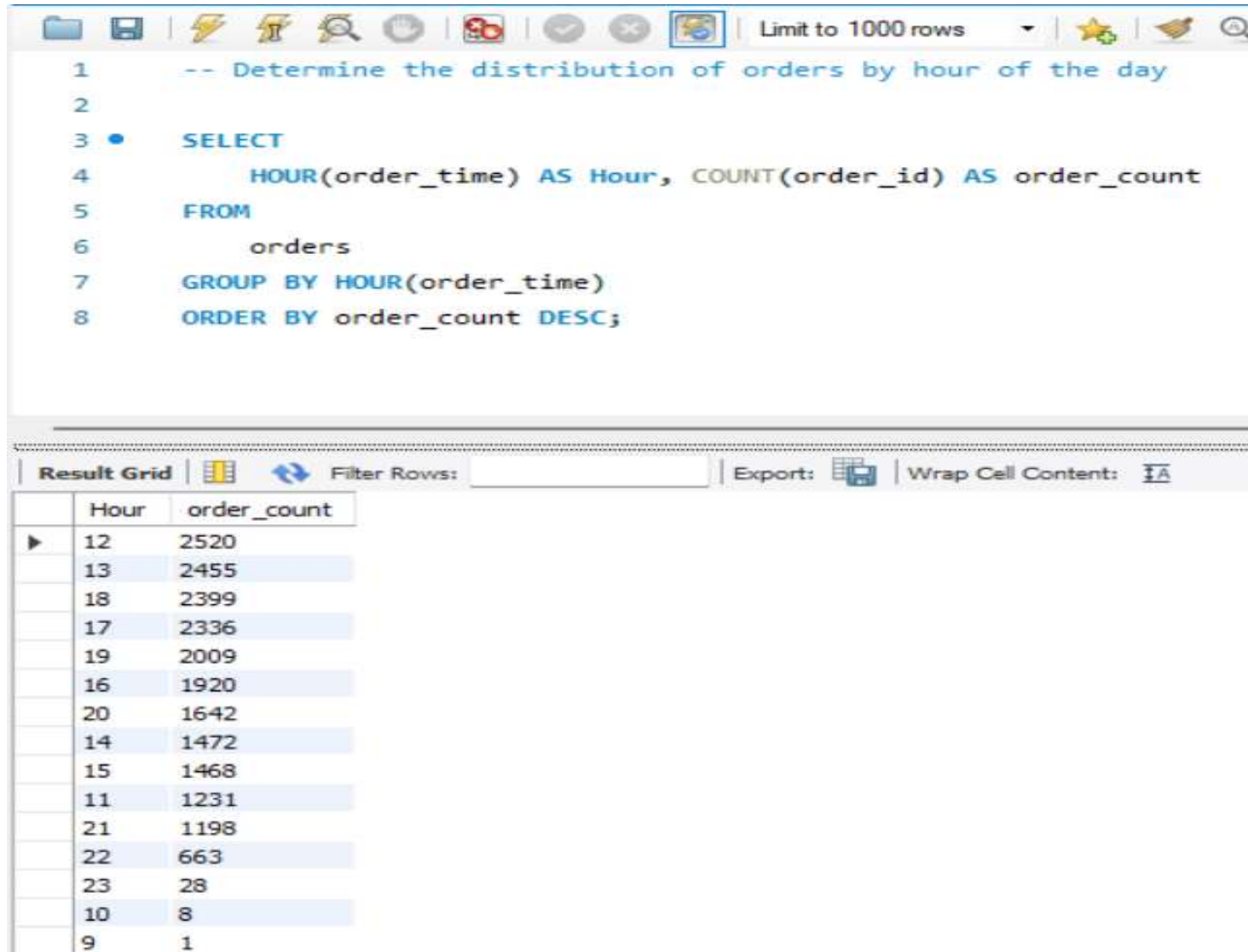
```
1  -- Join the necessary tables to find the total quantity of each pizza category ordered
2
3  •  SELECT
4      pizza_types.category,
5      SUM(order_details.quantity) AS total_quantity
6  FROM
7      pizza_types
8      JOIN
9      pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
10  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



	category	total_quantity
▶	Classic	14888
	Supreme	11987
	Veggie	11649
	Chicken	11050

Problem Statement: 7

DETERMINE THE DISTRIBUTION OF ORDERS BY HOUR OF THE DAY.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The query editor contains the following SQL code:

```
1  -- Determine the distribution of orders by hour of the day
2
3  •  SELECT
4      HOUR(order_time) AS Hour, COUNT(order_id) AS order_count
5  FROM
6      orders
7  GROUP BY HOUR(order_time)
8  ORDER BY order_count DESC;
```



Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The results are shown in a table with two columns: 'Hour' and 'order_count'. The rows are ordered by 'order_count' in descending order.

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

Problem Statement: 8

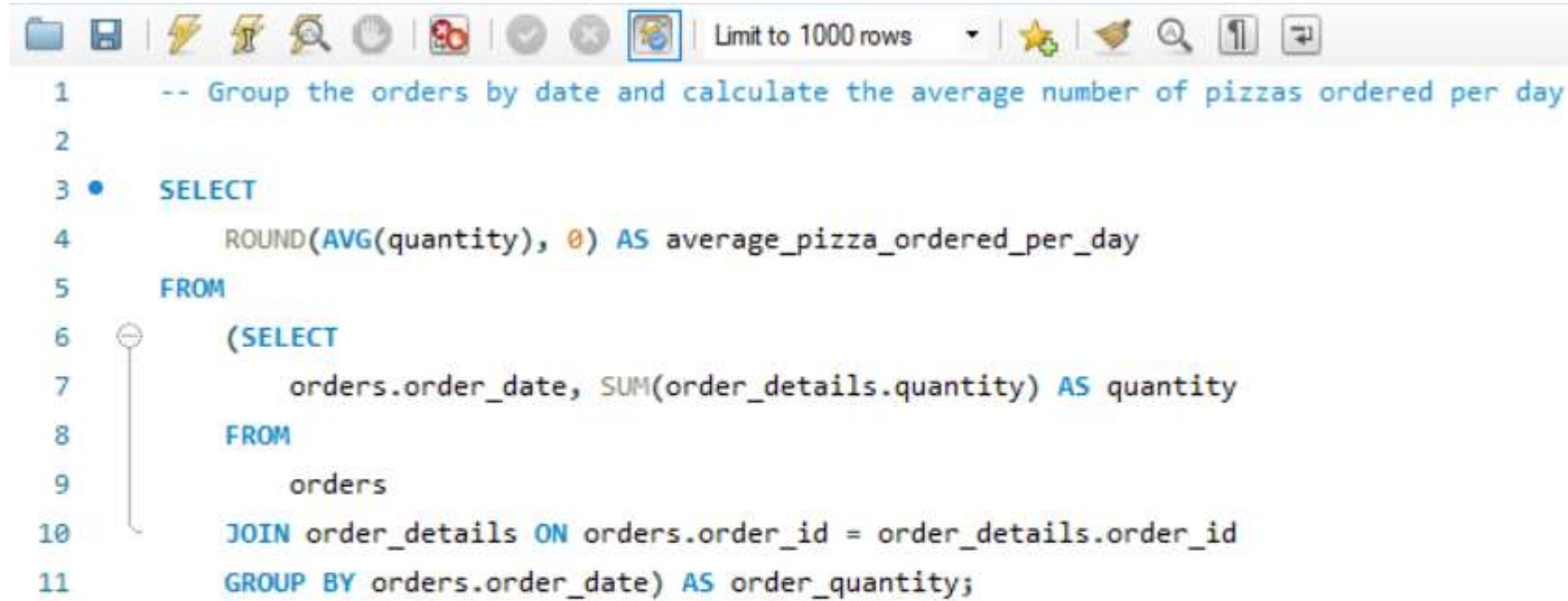
JOIN RELEVANT TABLES TO FIND THE CATEGORY-WISE DISTRIBUTION OF PIZZAS.

```
1  -- Join relevant tables to find the category-wise distribution of pizzas
2
3  •  select category , count(name) as name from pizza_types group by category;
```

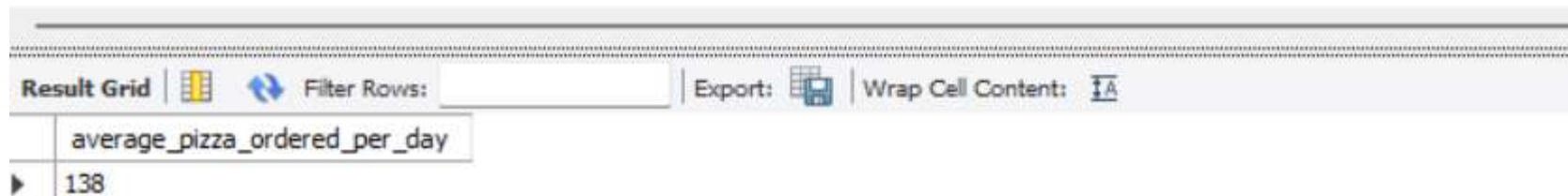
Result Grid		
Filter Rows: <input type="text"/>		
Export:  Wrap Cell Content: 		
	category	name
▶	Chicken	6
	Classic	8
	Supreme	9
	Veggie	9

Problem Statement: 9

GROUP THE ORDERS BY DATE AND CALCULATE THE AVERAGE NUMBER OF PIZZAS ORDERED PER DAY.

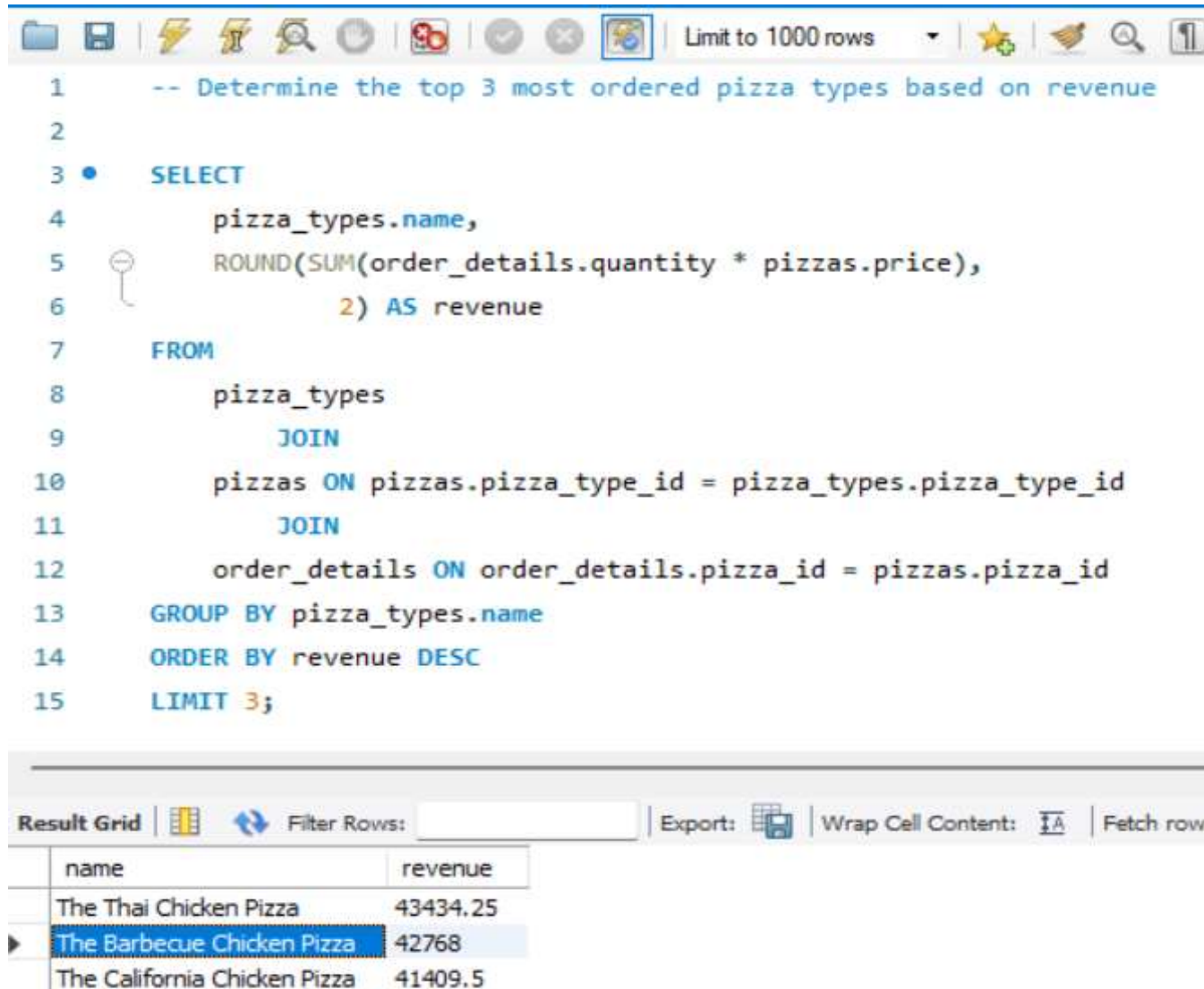


```
1  -- Group the orders by date and calculate the average number of pizzas ordered per day
2
3  •  SELECT
4      ROUND(AVG(quantity), 0) AS average_pizza_ordered_per_day
5  FROM
6      (SELECT
7          orders.order_date, SUM(order_details.quantity) AS quantity
8      FROM
9          orders
10  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



average_pizza_ordered_per_day
138

DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE.



The screenshot displays a SQL query in a code editor and its corresponding result set in a table below. The query is designed to find the top 3 most ordered pizza types based on revenue. The result set shows three rows: 'The Thai Chicken Pizza' with a revenue of 43434.25, 'The Barbecue Chicken Pizza' with a revenue of 42768, and 'The California Chicken Pizza' with a revenue of 41409.5. The second row is highlighted.

```
1  -- 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 pizzas.pizza_type_id = pizza_types.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;
```

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

Problem Statement: 11

CALCULATE THE PERCENTAGE CONTRIBUTION OF EACH PIZZA TYPE TO TOTAL REVENUE.

```
1  -- 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_sales
8      FROM
9          order_details
10         JOIN
11             pizzas ON pizzas.pizza_id = order_details.pizza_id)) * 100,
12          2) AS revenue_percent
13  FROM
14      pizza_types
15      JOIN
16          pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
17      JOIN
18          order_details ON order_details.pizza_id = pizzas.pizza_id
19  GROUP BY pizza_types.category
20  ORDER BY revenue_percent DESC;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	category	revenue_percent			
▶	Classic	26.91			
	Supreme	25.46			
	Chicken	23.96			
	Veggie	23.68			