



# **PIZZA SALES -- SQL PROJECT**

# Project Overview: Pizza Sales Data Analysis

## Objective:

This project aims to analyze pizza sales data to extract key business insights, such as order patterns, revenue generation, popular pizza types, and customer ordering behavior. The analysis covers basic, intermediate, and advanced levels of complexity to help optimize business strategies.

## *Datasets Provided:*

- **orders.csv** – Contains details of customer orders, including order ID, order date, and time.
- **order\_details.csv** – Provides itemized order details, including pizza ID, quantity, and price.
- **pizzas.csv** – Contains information about different pizza types, including size, category, and price.
- **pizza\_types.csv** – Provides details about pizza names and categories.

# Questions:

## Basic:

Retrieve the total number of orders placed.

Calculate the total revenue generated from pizza sales.

Identify the highest-priced pizza.

Identify the most common pizza size ordered.

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

## Intermediate:

Join the necessary tables to find the total quantity of each pizza category ordered.

Determine the distribution of orders by hour of the day.

Join relevant tables to find the category-wise distribution of pizzas.

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

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

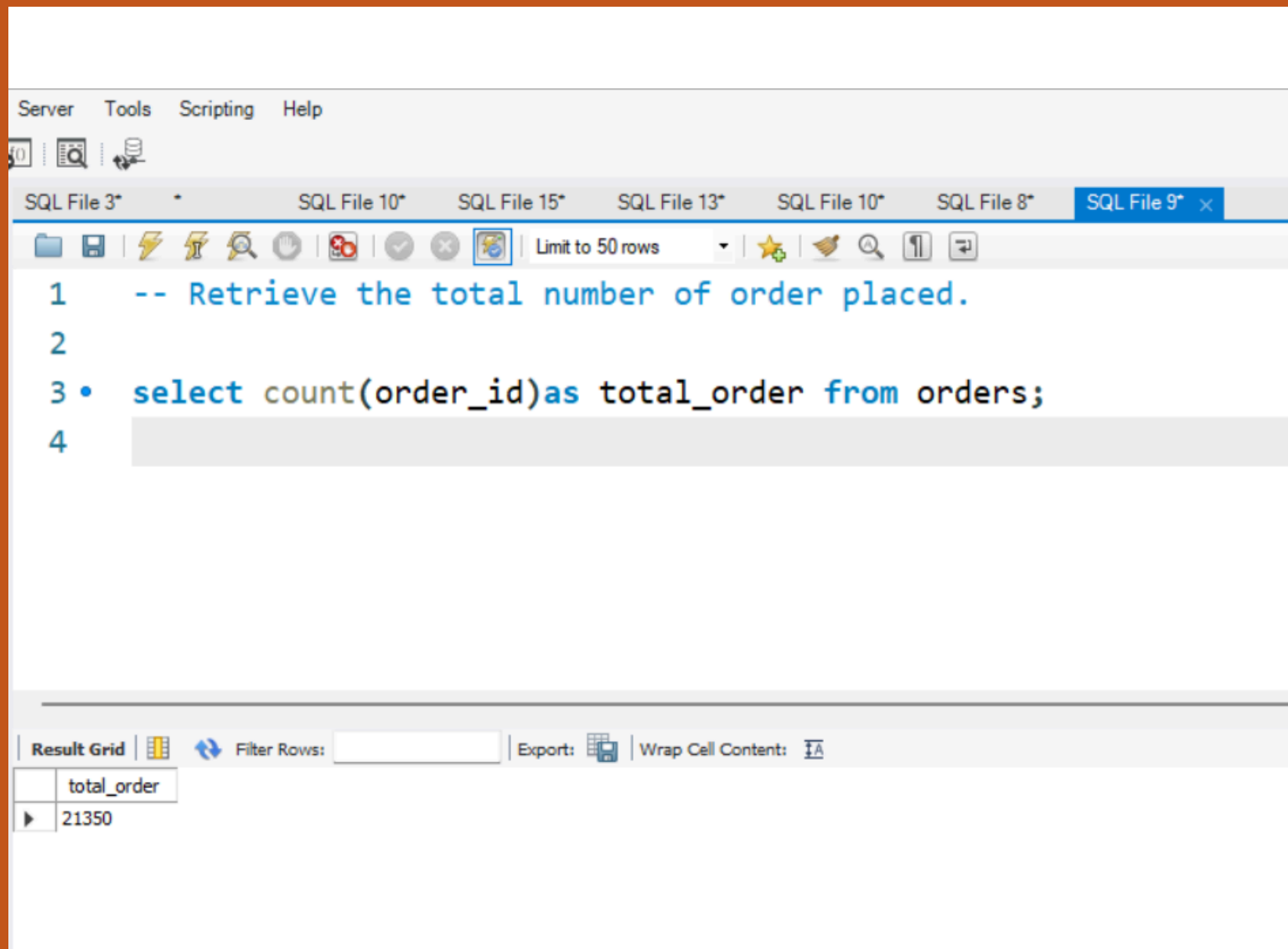
## Advanced:

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

Analyze the cumulative revenue generated over time.

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

# Retrieve the total number of orders placed



The screenshot shows a SQL IDE interface with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The main editor displays a SQL query in a file named 'SQL File 9\*'. The query is as follows:

```
1  -- Retrieve the total number of order placed.  
2  
3 • select count(order_id)as total_order from orders;  
4
```

Below the editor, the 'Result Grid' is visible, showing the output of the query:

| total_order |
|-------------|
| 21350       |

# Calculate the total revenue generated from pizza sales

The screenshot shows a SQL IDE interface with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The active window is 'SQL File 10\*'. The SQL query is as follows:

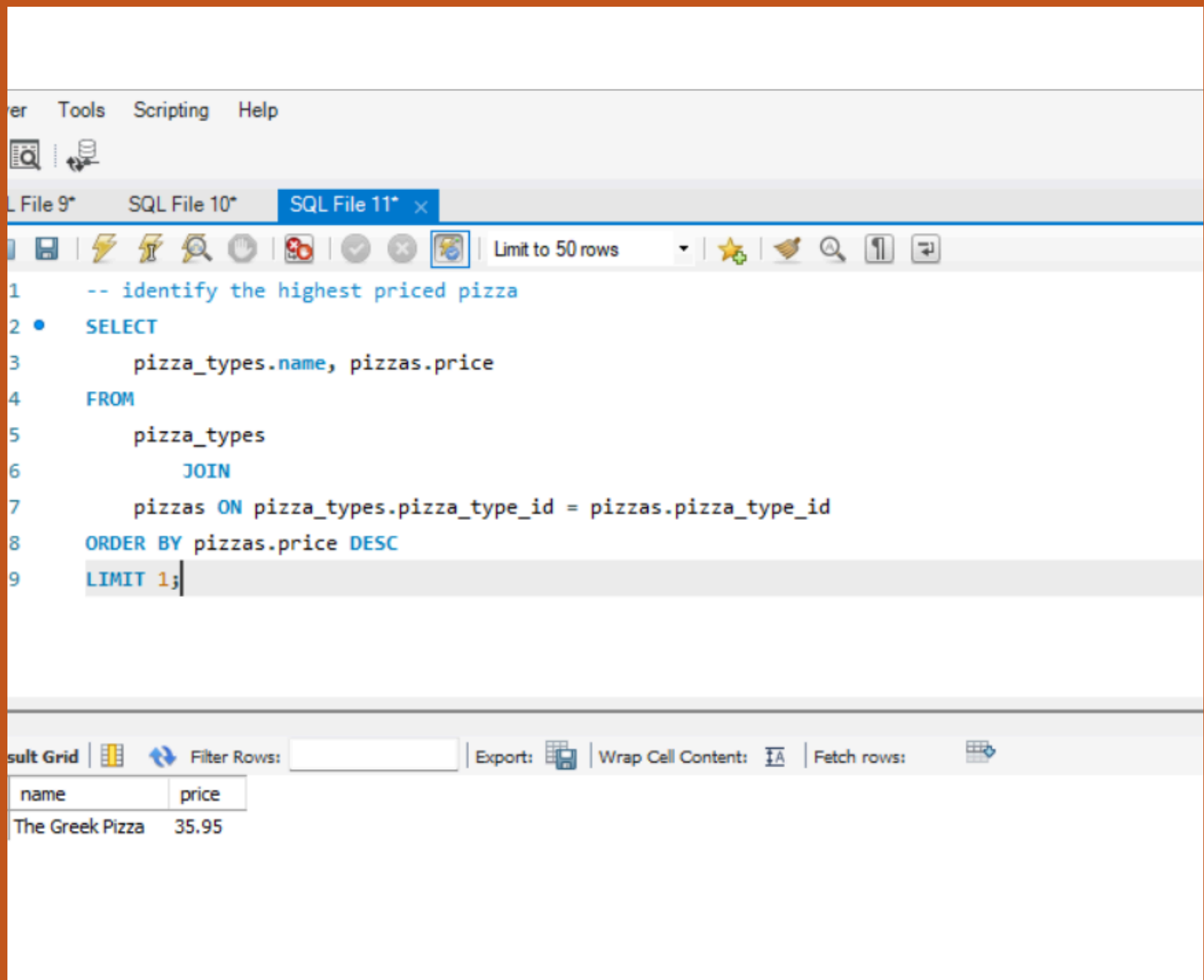
```
1  -- Calculate the total revenue generated from pizza sales;
2  SELECT
3      ROUND(SUM(orders_details.quantity * pizzas.price),
4             2) AS total_sales
5  FROM
6      orders_details
7      JOIN
8      pizzas ON pizzas.pizza_id = orders_details.pizza_id
9
```

Below the query editor, the 'Result Grid' tab is active. It displays the result of the query in a table with one column, 'total\_sales', and one row with the value '817860.05'.

| total_sales |
|-------------|
| 817860.05   |

The bottom of the window shows a tab for 'Result 1'.

# Identify the highest-priced pizza



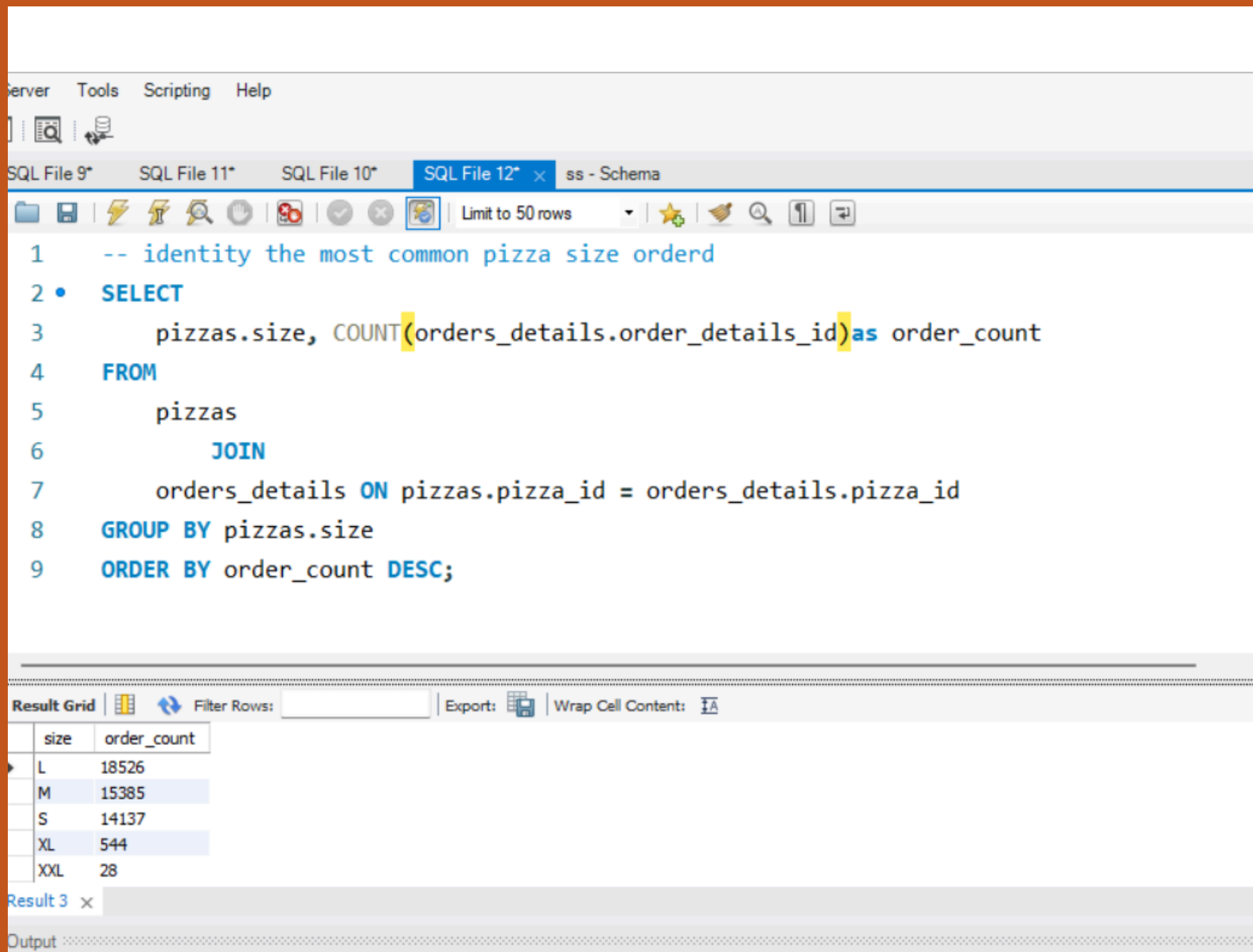
The screenshot shows a SQL IDE interface with a menu bar (File, Tools, Scripting, Help) and a toolbar. The active window is 'SQL File 11\*'. The SQL query is as follows:

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

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: 'name' and 'price'.

| name            | price |
|-----------------|-------|
| The Greek Pizza | 35.95 |

Identify the most common pizza size ordered.



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL code:

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

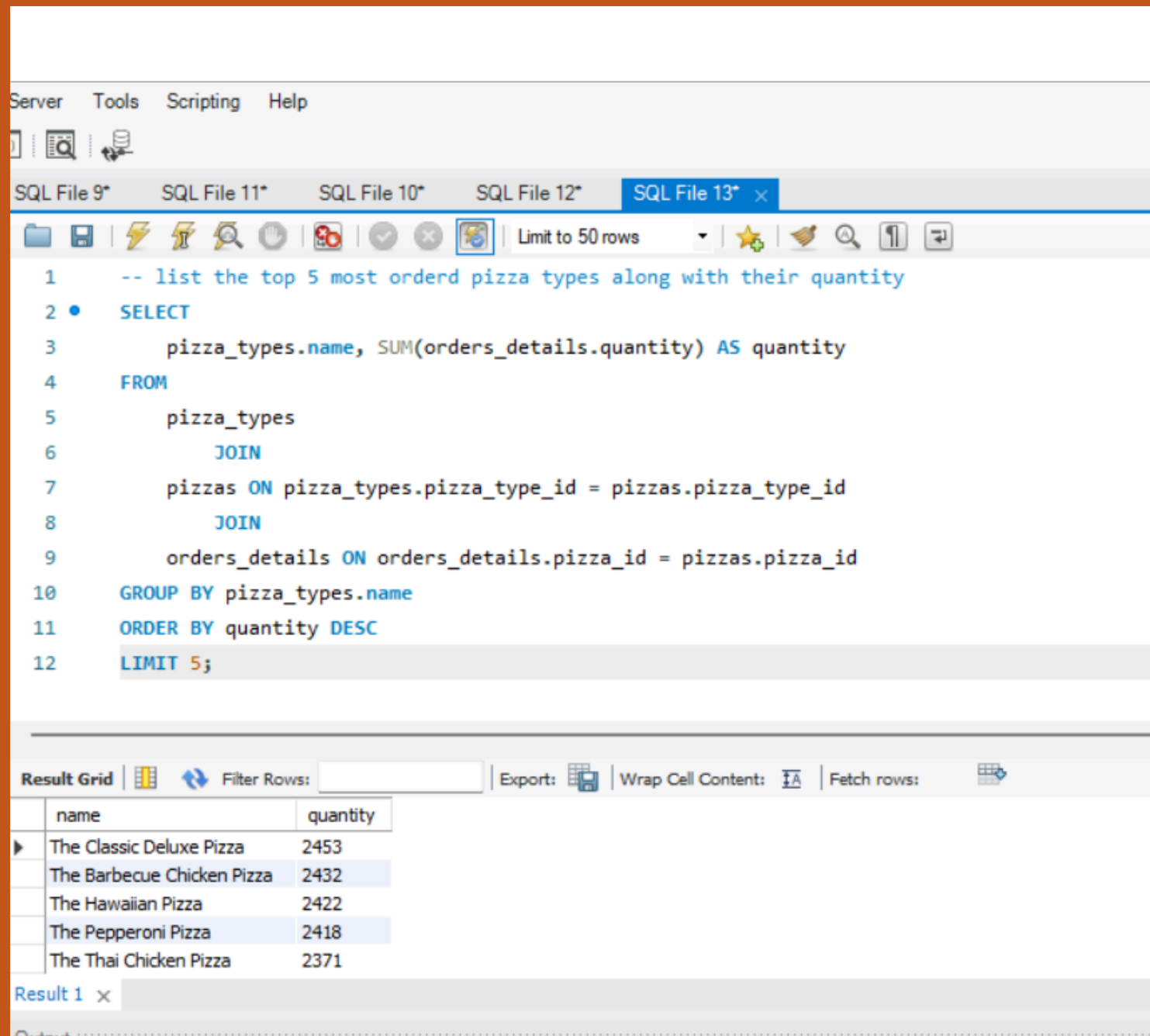
The result grid displays the following data:

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

The result grid is titled "Result 3" and includes options for "Filter Rows", "Export", and "Wrap Cell Content".



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



The screenshot shows a SQL IDE interface with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The active window is 'SQL File 13\*'. The SQL query is as follows:

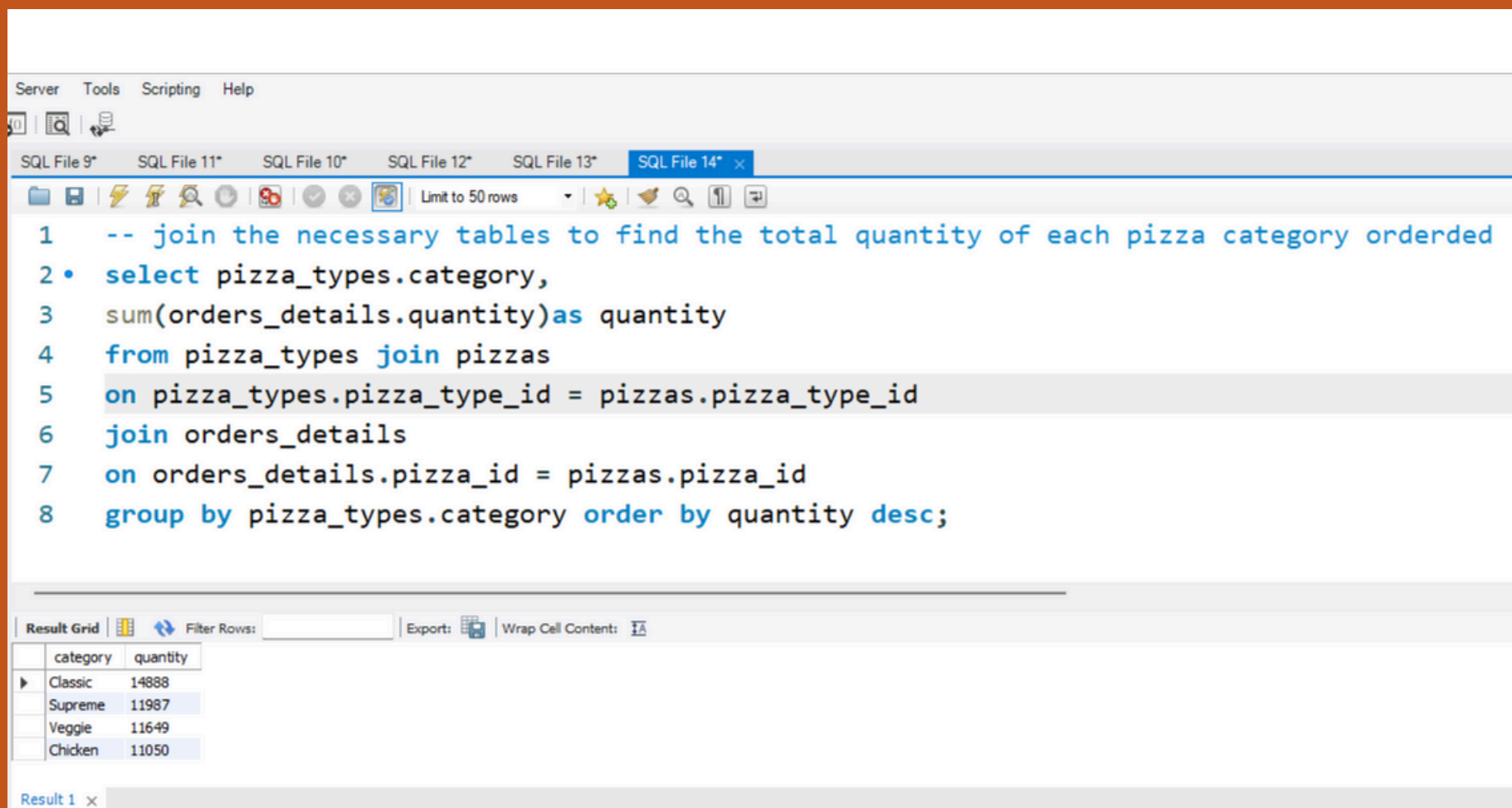
```
1  -- list the top 5 most ordered pizza types along with their quantity
2  •  SELECT
3      pizza_types.name, SUM(orders_details.quantity) AS quantity
4  FROM
5      pizza_types
6      JOIN
7      pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
8      JOIN
9      orders_details ON orders_details.pizza_id = pizzas.pizza_id
10 GROUP BY pizza_types.name
11 ORDER BY quantity DESC
12 LIMIT 5;
```

Below the query editor is the 'Result Grid' section, which includes a 'Filter Rows' input, an 'Export' button, a 'Wrap Cell Content' checkbox, and a 'Fetch rows' button. The results are displayed in a table with two columns: 'name' and 'quantity'.

| 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     |

At the bottom, there is a 'Result 1' tab and an 'Output' section.

Join the necessary tables to find the total quantity of each pizza category ordered.



The screenshot shows a SQL IDE window with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The active tab is 'SQL File 14\*'. The query editor contains the following SQL code:

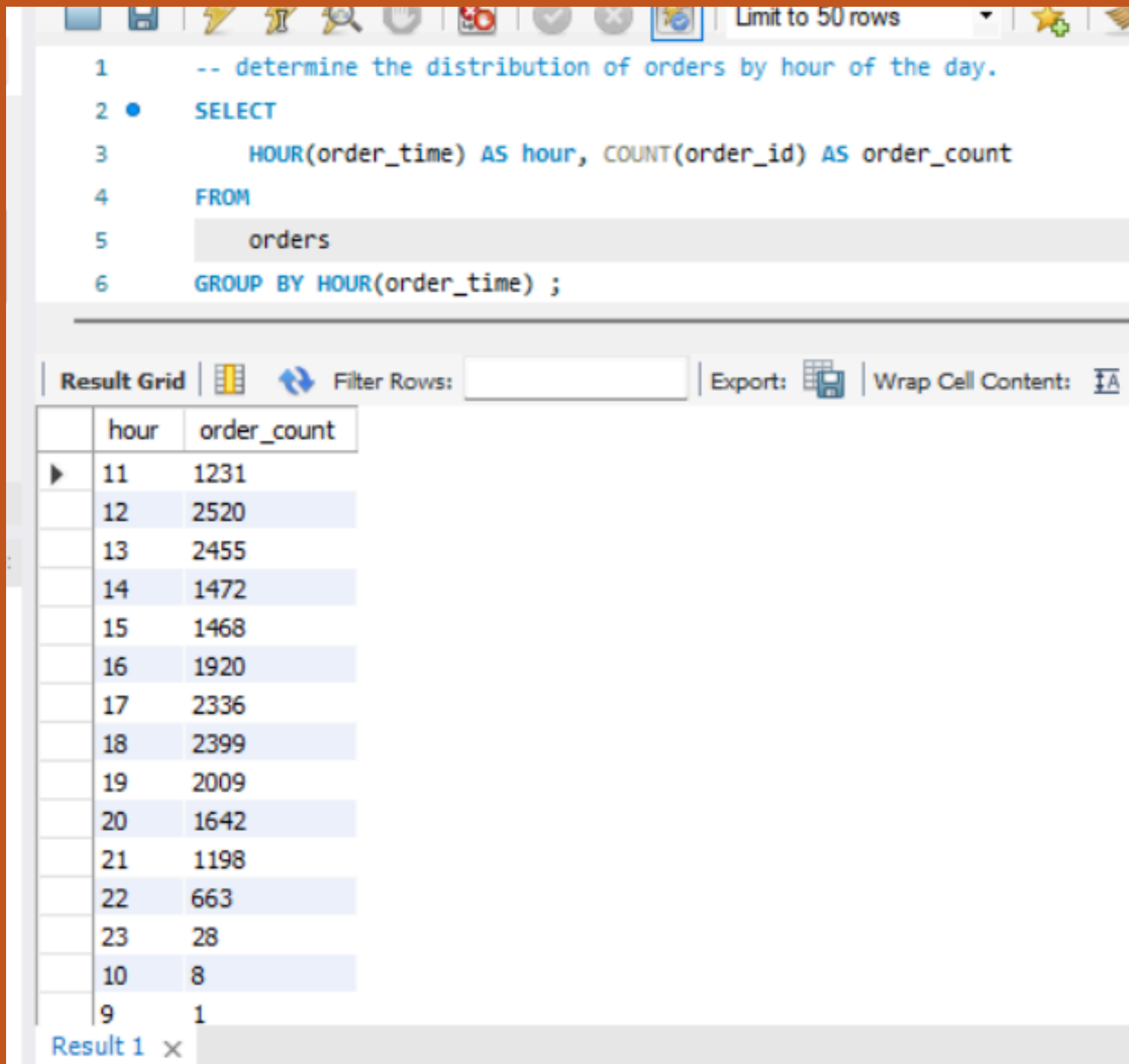
```
1  -- join the necessary tables to find the total quantity of each pizza category ordered
2 • select pizza_types.category,
3    sum(orders_details.quantity) as quantity
4  from pizza_types join pizzas
5   on pizza_types.pizza_type_id = pizzas.pizza_type_id
6  join orders_details
7   on orders_details.pizza_id = pizzas.pizza_id
8  group by pizza_types.category order by quantity desc;
```

Below the query editor is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with two columns: 'category' and 'quantity'.

| category | quantity |
|----------|----------|
| Classic  | 14888    |
| Supreme  | 11987    |
| Veggie   | 11649    |
| Chicken  | 11050    |

The bottom of the window shows 'Result 1' with a close button.

# Determine the distribution of orders by hour of the day



The screenshot shows a SQL IDE interface. At the top, there is a toolbar with various icons and a dropdown menu set to "Limit to 50 rows". Below the toolbar, a SQL query is entered in a text area. The query is as follows:

```
1  -- determine the distribution of orders by hour of the day.  
2  ●  SELECT  
3      HOUR(order_time) AS hour, COUNT(order_id) AS order_count  
4  FROM  
5      orders  
6  GROUP BY HOUR(order_time) ;
```

Below the query editor, there is a "Result Grid" section. It includes a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. 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           |

At the bottom left, there is a tab labeled "Result 1" with a close button (X).

# Join relevant tables to find the category-wise distribution of pizzas

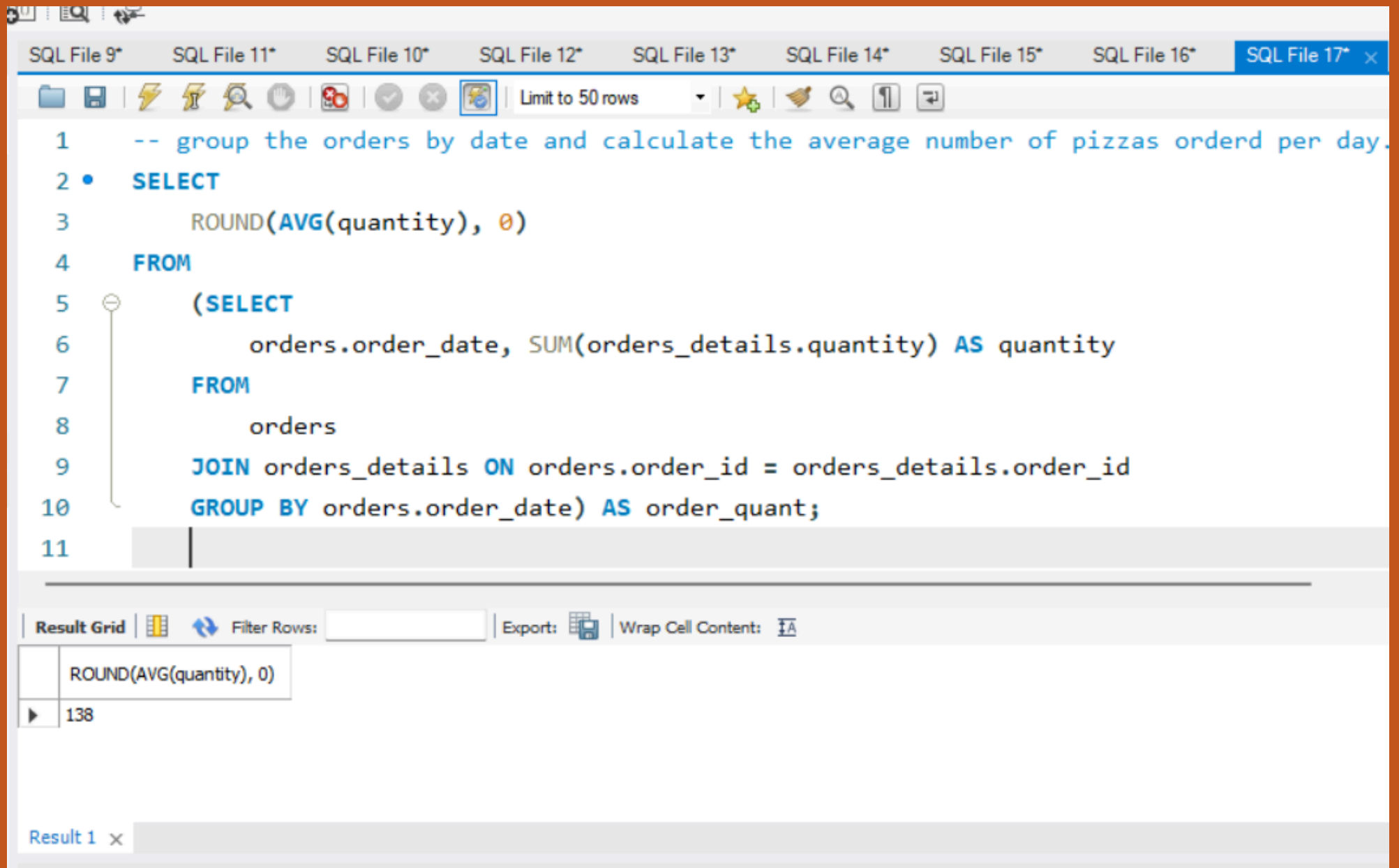
```
1  -- join the relevant tables to find the category wise distribution of pizza.
2  • select category, count(name) from pizza_types
3  group by category;_
```

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

| category | count(name) |
|----------|-------------|
| Chicken  | 6           |
| Classic  | 8           |
| Supreme  | 9           |
| Veggie   | 9           |

Result 1 x | Read O

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



The screenshot shows a SQL IDE interface with multiple tabs at the top labeled "SQL File 9\*", "SQL File 11\*", "SQL File 10\*", "SQL File 12\*", "SQL File 13\*", "SQL File 14\*", "SQL File 15\*", "SQL File 16\*", and "SQL File 17\* x". The active tab is "SQL File 17\* x". The toolbar includes icons for file operations, a "Limit to 50 rows" dropdown, and other utility icons. The SQL editor contains the following code:

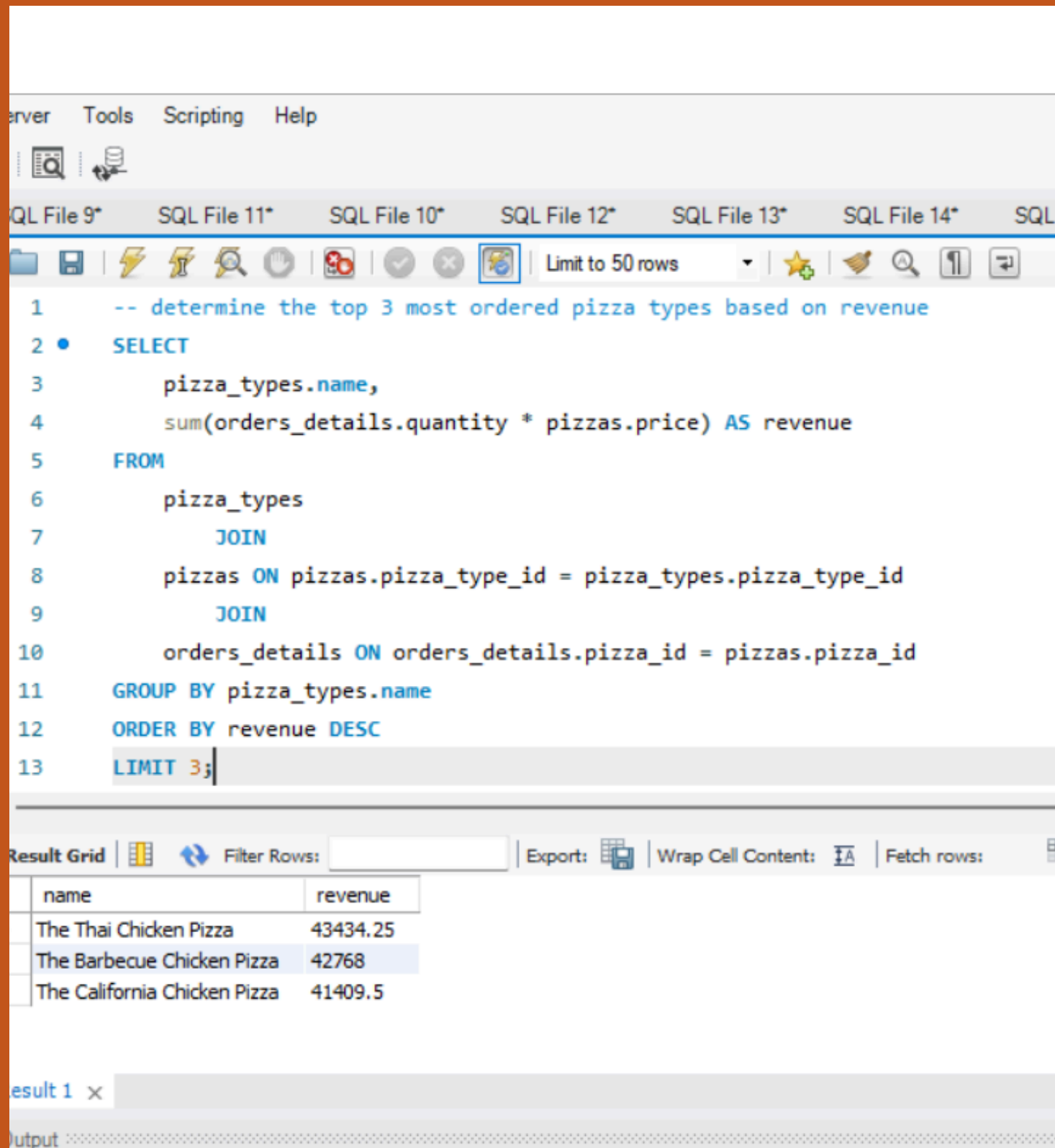
```
1  -- group the orders by date and calculate the average number of pizzas orderd per day.
2  •  SELECT
3      ROUND(AVG(quantity), 0)
4  FROM
5      (SELECT
6          orders.order_date, SUM(orders_details.quantity) AS quantity
7      FROM
8          orders
9      JOIN orders_details ON orders.order_id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

Below the editor is a "Result Grid" section with a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. The result grid displays the following data:

|   | ROUND(AVG(quantity), 0) |
|---|-------------------------|
| ▶ | 138                     |

At the bottom left, there is a tab labeled "Result 1 x".

# Determine the top 3 most ordered pizza types based on revenue



The screenshot shows a SQL IDE interface with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The main editor displays a SQL query to determine the top 3 most ordered pizza types based on revenue. The query uses a JOIN between pizza\_types, pizzas, and orders\_details tables, grouped by pizza\_type.name and ordered by revenue in descending order, limited to 3 rows.

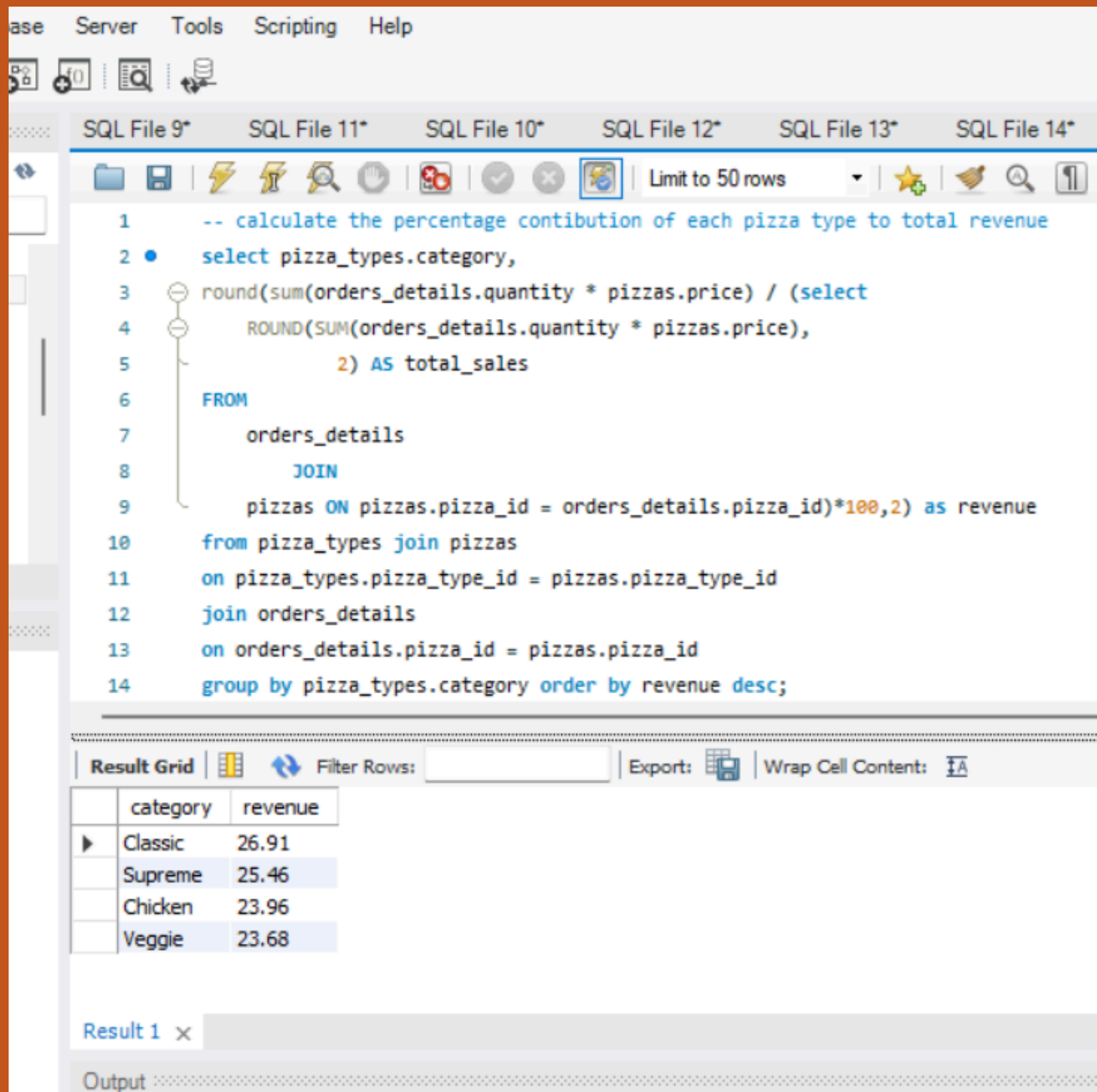
```
1  -- determine the top 3 most ordered pizza types based on revenue
2  •  SELECT
3      pizza_types.name,
4      sum(orders_details.quantity * pizzas.price) AS revenue
5  FROM
6      pizza_types
7      JOIN
8      pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
9      JOIN
10     orders_details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

Below the query editor, the 'Result Grid' tab is active, showing the results of the query. The results are displayed in a table with two columns: 'name' and 'revenue'.

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

At the bottom of the IDE, there is a 'result 1' tab and an 'Output' pane.

# Calculate the percentage contribution of each pizza type to total revenue



The screenshot shows a SQL IDE interface with a menu bar (File, Server, Tools, Scripting, Help) and a toolbar. The main window displays a SQL query in a file named 'SQL File 12\*'. The query is as follows:

```
1  -- calculate the percentage contribution of each pizza type to total revenue
2  • select pizza_types.category,
3     round(sum(orders_details.quantity * pizzas.price) / (select
4     ROUND(SUM(orders_details.quantity * pizzas.price),
5           2) AS total_sales
6  FROM
7     orders_details
8     JOIN
9     pizzas ON pizzas.pizza_id = orders_details.pizza_id)*100,2) as revenue
10 from pizza_types join pizzas
11 on pizza_types.pizza_type_id = pizzas.pizza_type_id
12 join orders_details
13 on orders_details.pizza_id = pizzas.pizza_id
14 group by pizza_types.category order by revenue desc;
```

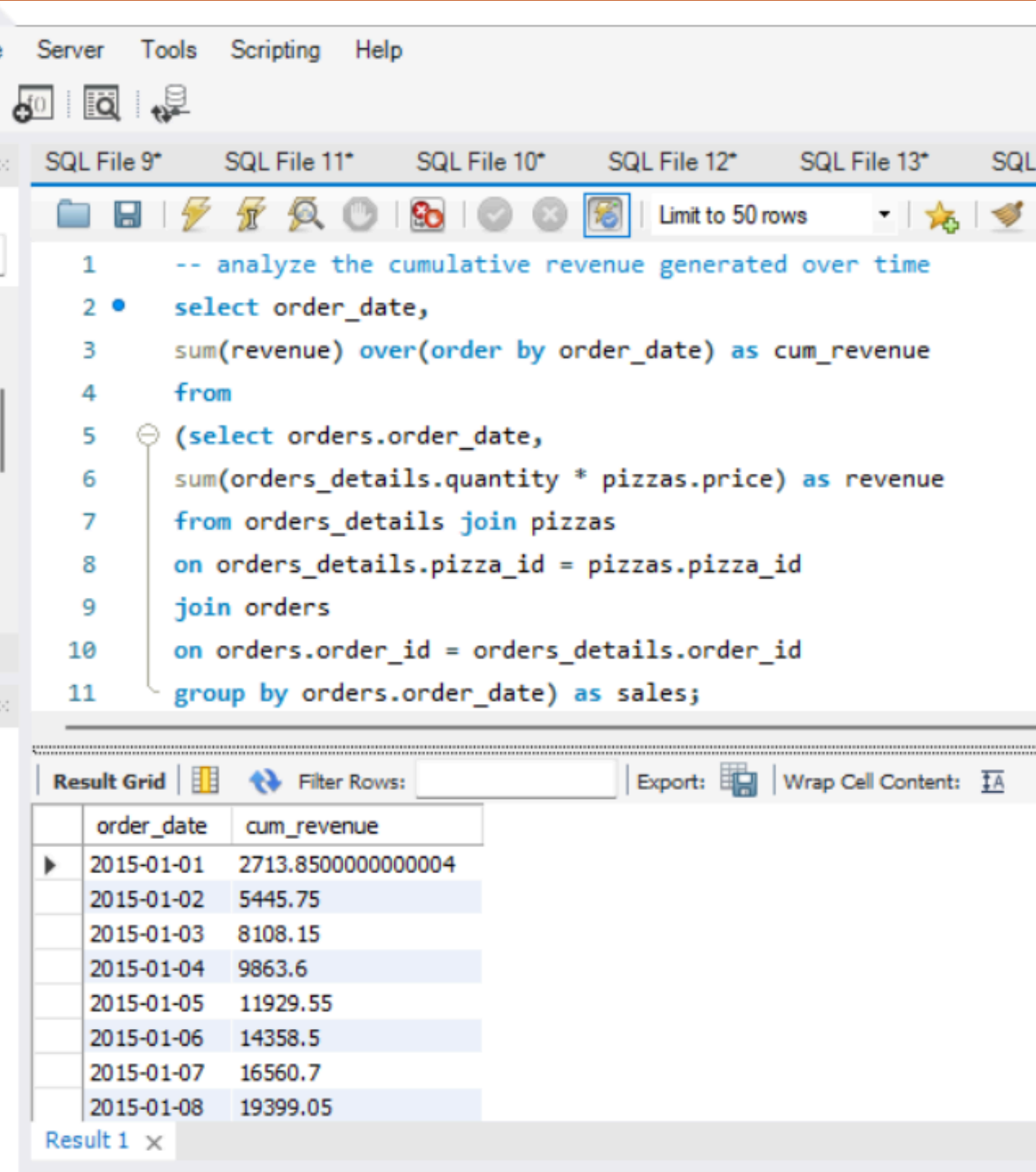
Below the query editor, the 'Result Grid' tab is active, showing the results of the query. The results are displayed in a table with two columns: 'category' and 'revenue'.

| category | revenue |
|----------|---------|
| Classic  | 26.91   |
| Supreme  | 25.46   |
| Chicken  | 23.96   |
| Veggie   | 23.68   |

The 'Result Grid' tab also includes a 'Filter Rows' input field and an 'Export' button. The 'Output' tab is visible at the bottom of the interface.



# Analyze the cumulative revenue generated over time



The screenshot displays a SQL IDE interface with a menu bar (Server, Tools, Scripting, Help) and a toolbar. The main editor shows a SQL query designed to calculate cumulative revenue over time. The query uses a subquery to join 'orders\_details' and 'pizzas' tables, calculate the revenue for each order, and then uses a window function 'sum(revenue) over(order by order\_date)' to calculate the cumulative revenue. The result is presented in a 'Result Grid' at the bottom, showing the first eight days of January 2015.

```
1  -- analyze the cumulative revenue generated over time
2  •  select order_date,
3     sum(revenue) over(order by order_date) as cum_revenue
4  from
5  (select orders.order_date,
6     sum(orders_details.quantity * pizzas.price) as revenue
7  from orders_details join pizzas
8  on orders_details.pizza_id = pizzas.pizza_id
9  join orders
10 on orders.order_id = orders_details.order_id
11  group by orders.order_date) as sales;
```

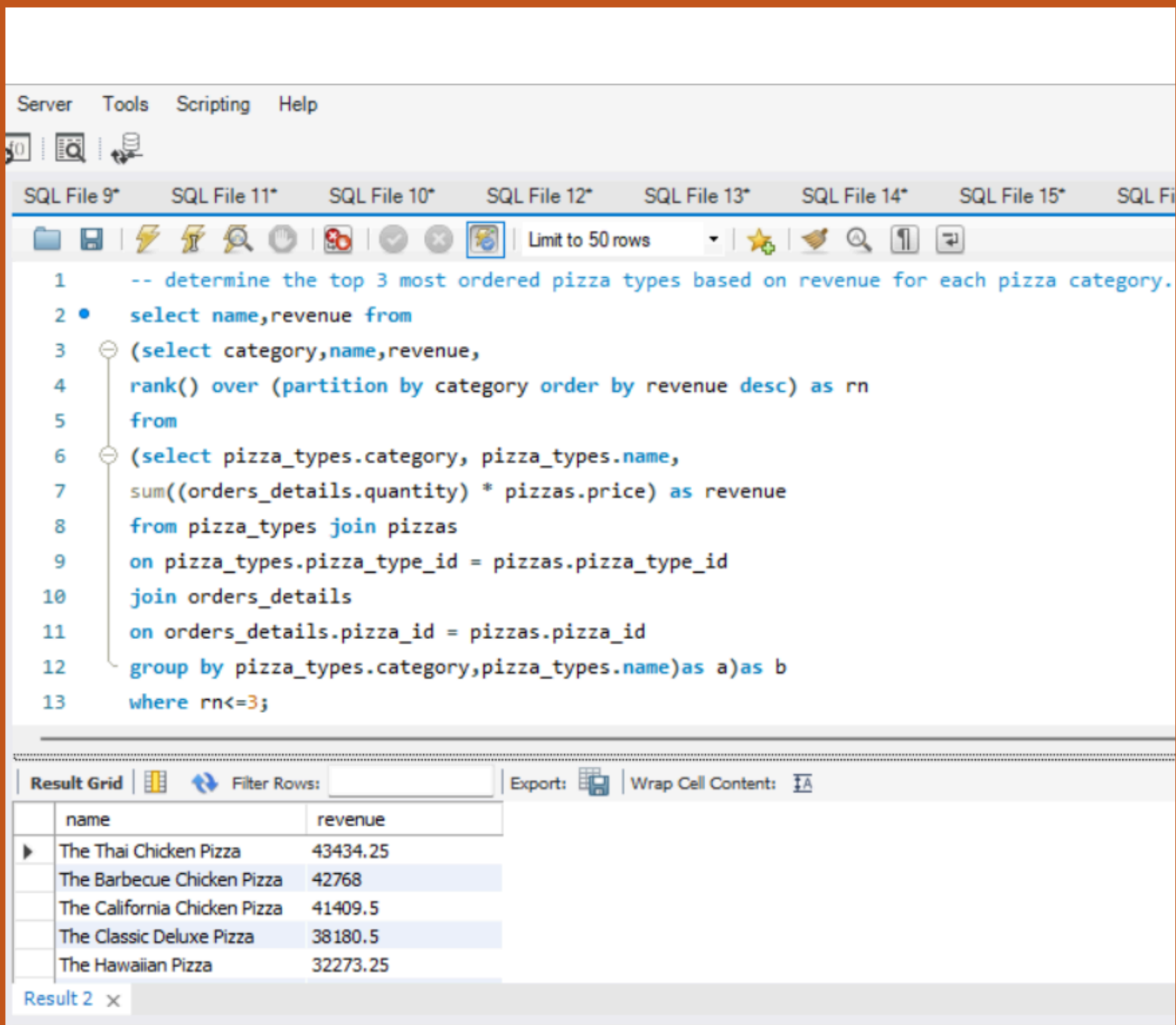
Result Grid

|   | order_date | cum_revenue         |
|---|------------|---------------------|
| ▶ | 2015-01-01 | 2713.85000000000004 |
|   | 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            |

Result 1 x



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



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query is designed to find the top 3 most ordered pizza types based on revenue for each pizza category. The query uses a subquery to calculate the revenue for each pizza type and then ranks them within each category.

```
1  -- determine the top 3 most ordered pizza types based on revenue for each pizza category.
2  • select name, revenue from
3  (select category, name, revenue,
4   rank() over (partition by category order by revenue desc) as rn
5   from
6   (select pizza_types.category, pizza_types.name,
7    sum((orders_details.quantity) * pizzas.price) as revenue
8    from pizza_types join pizzas
9    on pizza_types.pizza_type_id = pizzas.pizza_type_id
10   join orders_details
11   on orders_details.pizza_id = pizzas.pizza_id
12   group by pizza_types.category, pizza_types.name) as a) as b
13  where rn <= 3;
```

The result grid displays the following data:

|   | name                         | revenue  |
|---|------------------------------|----------|
| ▶ | The Thai Chicken Pizza       | 43434.25 |
|   | The Barbecue Chicken Pizza   | 42768    |
|   | The California Chicken Pizza | 41409.5  |
|   | The Classic Deluxe Pizza     | 38180.5  |
|   | The Hawaiian Pizza           | 32273.25 |

Result 2 x

# Project Summary.

This project focuses on analyzing pizza sales data to extract key business insights. By examining order patterns, revenue generation, and customer preferences, the analysis helps optimize business strategies. The data includes order details, pizzas, and their categories, which are used to derive valuable metrics such as total revenue, the most ordered pizza types, and sales distribution by time and category.

The analysis starts with fundamental metrics like total orders, revenue, and top-selling pizzas. It then explores order distribution across different hours of the day, pizza categories, and daily trends. Advanced analysis includes revenue contribution by pizza type, cumulative revenue tracking, and category-wise top-performing pizzas.

The insights gained support decision-making for inventory management, marketing strategies, and pricing optimization. By understanding which pizzas drive the most revenue and when customers place the most orders, businesses can improve efficiency and profitability.