

SQL Reinforcement Project

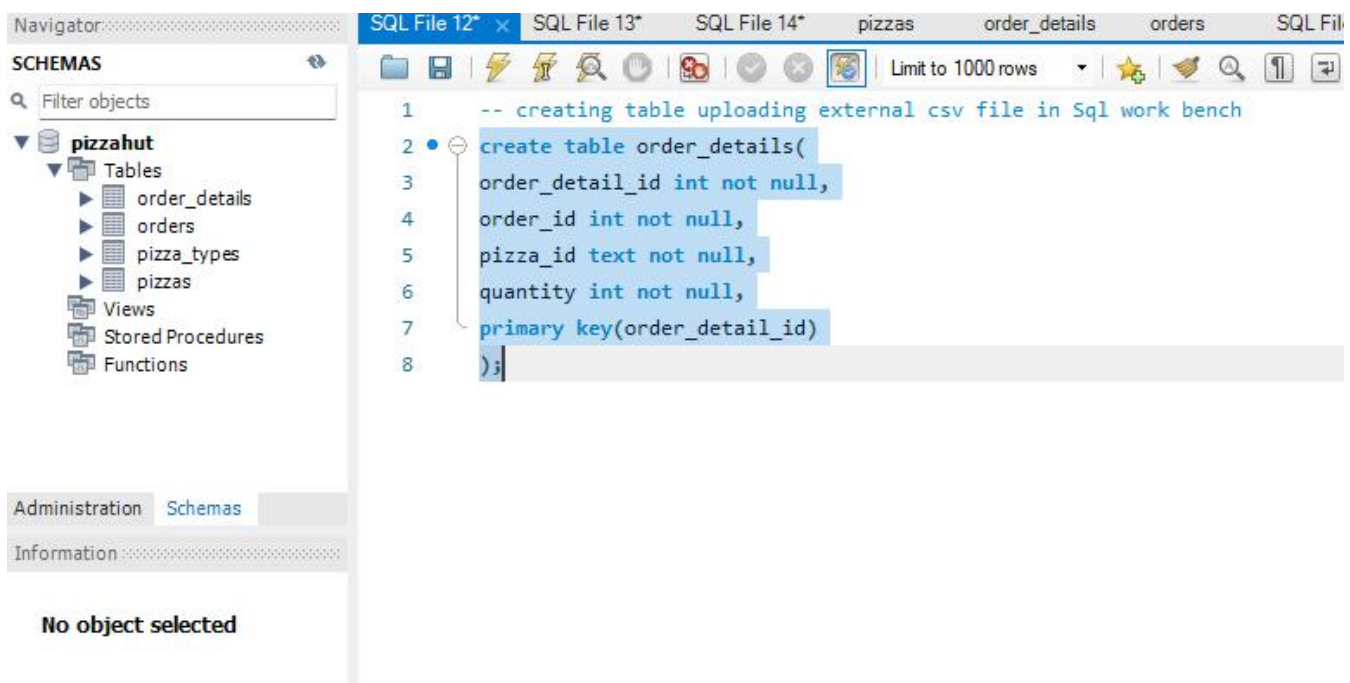
Question 1:

creating table uploading external csv file in Sql work bench

Query:

```
create table order_details(  
order_detail_id int not null,  
order_id int not null,  
pizza_id text not null,  
quantity int not null,  
primary key(order_detail_id)  
);
```

Output:



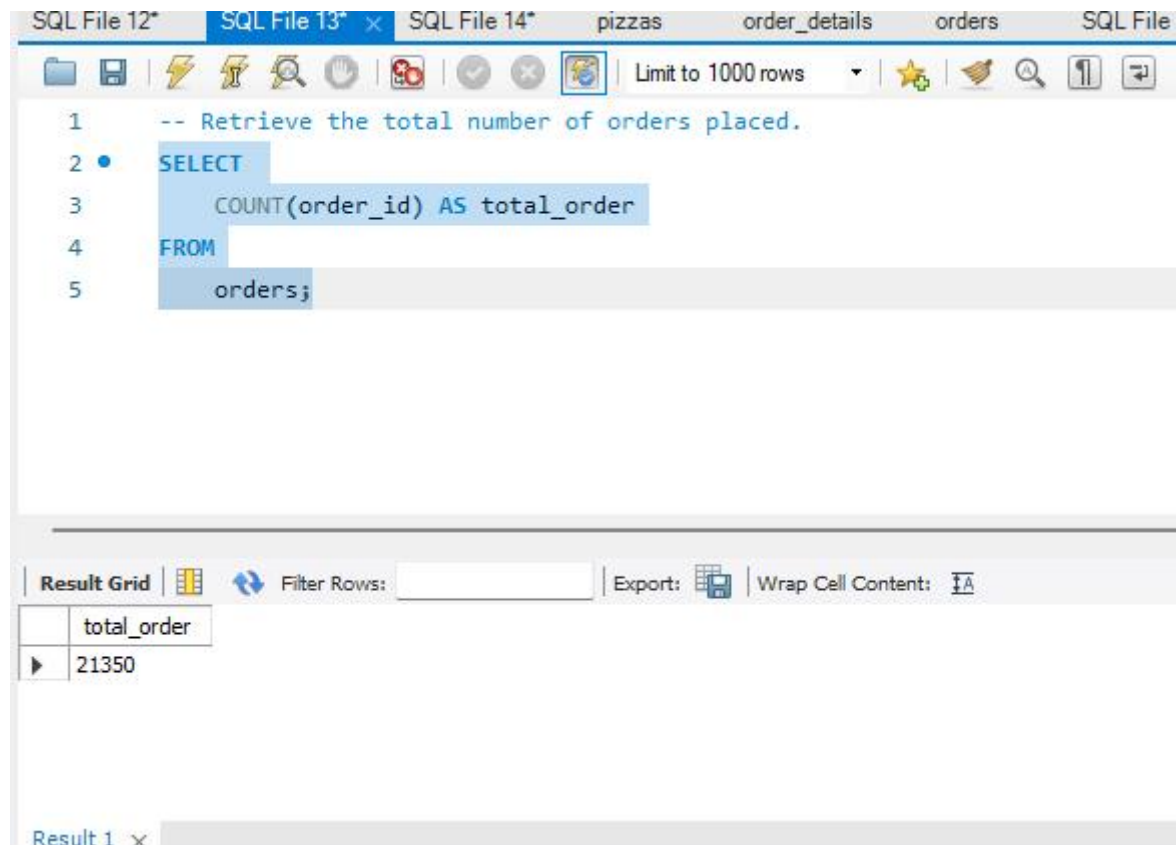
Question 2:

Retrieve the total number of orders placed.

Query:

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

Output:



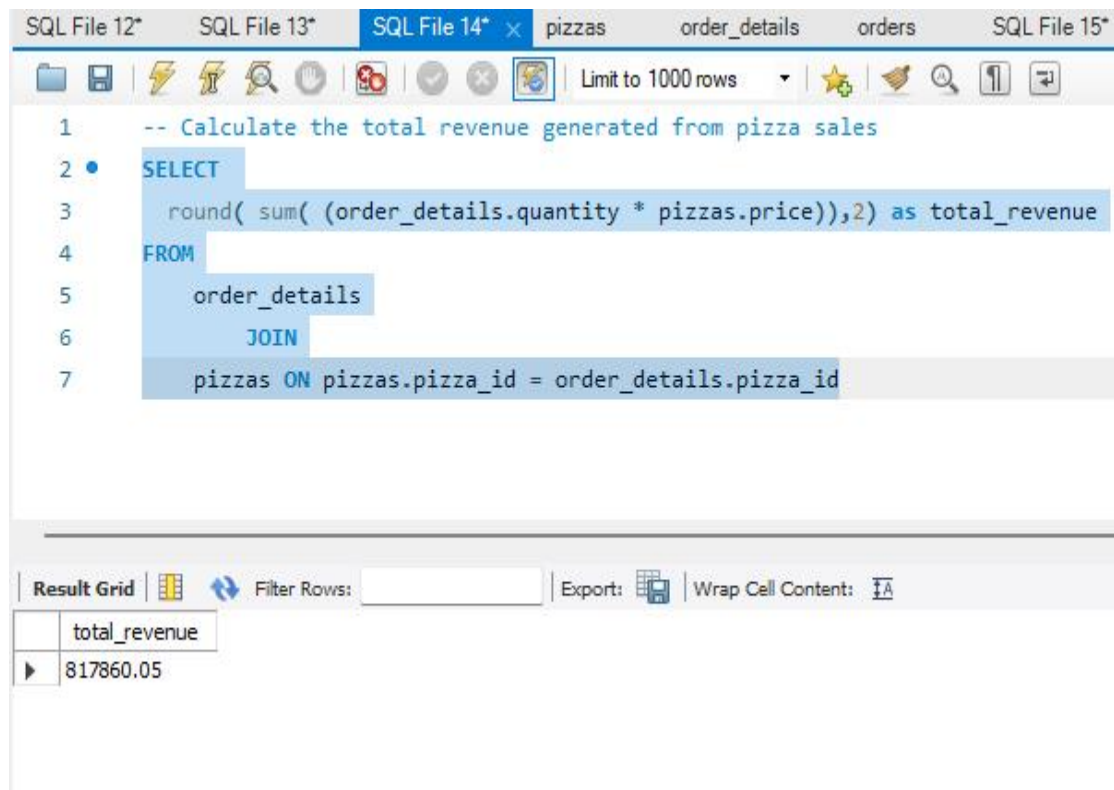
Question 3:

Calculate the total revenue generated from pizza sales

Query:

```
SELECT  
    round( sum( (order_details.quantity * pizzas.price)),2) as total_revenue  
FROM  
    order_details  
    JOIN  
    pizzas ON pizzas.pizza_id = order_details.pizza_id
```

Output:



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

```
1  -- Calculate the total revenue generated from pizza sales
2  • SELECT
3      round( sum( (order_details.quantity * pizzas.price)),2) as total_revenue
4  FROM
5      order_details
6      JOIN
7      pizzas ON pizzas.pizza_id = order_details.pizza_id
```

The result grid shows the following output:

| total_revenue |
|---------------|
| 817860.05 |

Question 4:

Identify the highest-priced pizza.

Query:

```
SELECT
    pizza_types.name, pizzas.price
FROM
    pizza_types
    JOIN
    pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
ORDER BY pizzas.price desc limit 1;
```

Output:

The screenshot shows a SQL IDE with multiple tabs: 'SQL File 12*', 'SQL File 13*', 'SQL File 14*', 'pizzas', 'order_details', 'orders', and 'SC'. The 'pizzas' tab is active, displaying a SQL query. The 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 pizzas.pizza_type_id = pizza_types.pizza_type_id
8  ORDER BY pizzas.price desc limit 1;
```

The screenshot shows the 'Result Grid' of the SQL IDE. It contains a single row of data:

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

Question 5:

Identify the most common pizza size ordered.

Query:

```
SELECT
    pizzas.size,
    COUNT(order_details.order_detail_id) AS order_count
FROM
    order_details
    JOIN
    pizzas ON pizzas.pizza_id = order_details.pizza_id
GROUP BY size
ORDER BY order_count DESC;
```

Output:

```

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

```

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

Question 6:

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

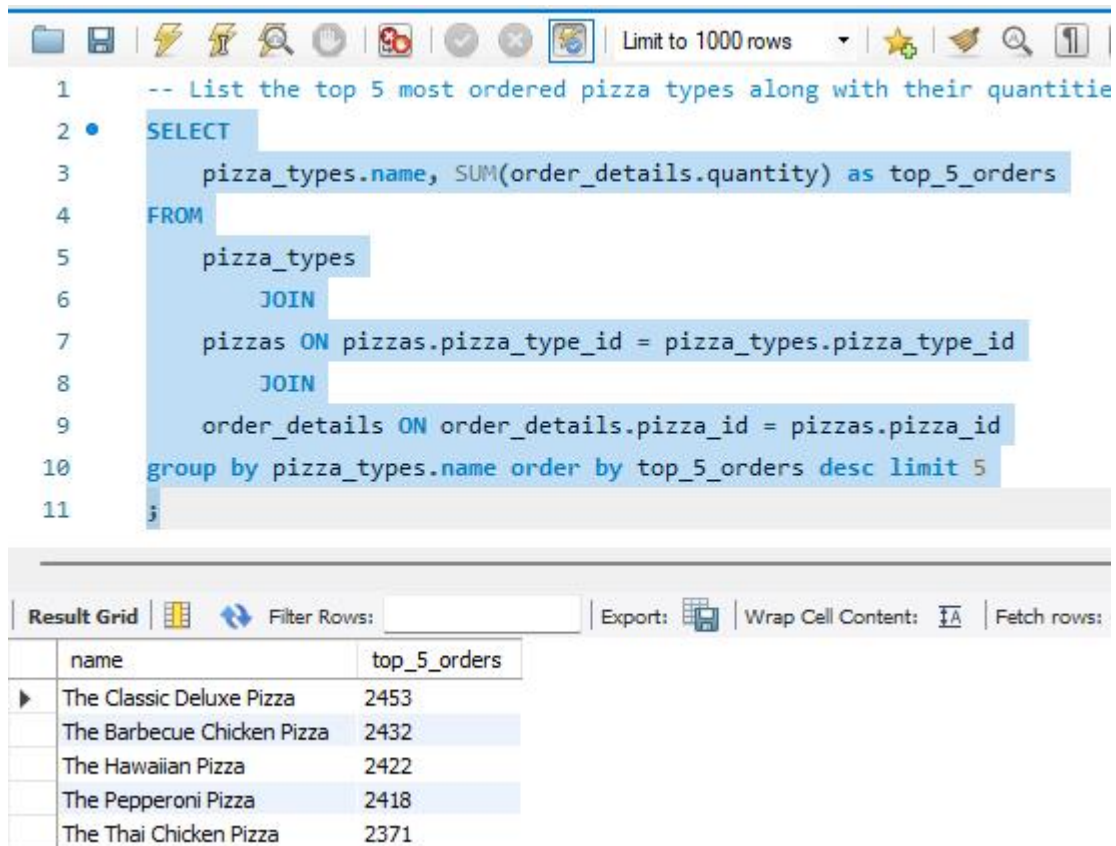
Query:

```

SELECT
    pizza_types.name, SUM(order_details.quantity) as top_5_orders
FROM
    pizza_types
    JOIN
    pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
    JOIN
    order_details ON order_details.pizza_id = pizzas.pizza_id
group by pizza_types.name order by top_5_orders desc limit 5
;

```

Output:



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  -- List the top 5 most ordered pizza types along with their quantitie
2  •  SELECT
3      pizza_types.name, SUM(order_details.quantity) as top_5_orders
4  FROM
5      pizza_types
6      JOIN
7      pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
8      JOIN
9      order_details ON order_details.pizza_id = pizzas.pizza_id
10 group by pizza_types.name order by top_5_orders desc limit 5
11 ;
```

Below the editor is the 'Result Grid' tab, which displays the query results in a table:

| | name | top_5_orders |
|---|----------------------------|--------------|
| ▶ | The Classic Deluxe Pizza | 2453 |
| | The Barbecue Chicken Pizza | 2432 |
| | The Hawaiian Pizza | 2422 |
| | The Pepperoni Pizza | 2418 |
| | The Thai Chicken Pizza | 2371 |

Question 7:

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

Query:

```
SELECT
    pizza_types.category,
    SUM(order_details.quantity) AS total_quantity_of_each_pizza_category_ordered
FROM
    pizza_types
    JOIN
    pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
    JOIN
    order_details ON pizzas.pizza_id = order_details.pizza_id
GROUP BY pizza_types.category;
```

Output:

```
2
3 • SELECT
4     pizza_types.category,
5     SUM(order_details.quantity) AS total_quantity_of_each_pizza_category_ordered
6 FROM
7     pizza_types
8     JOIN
9     pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
10    JOIN
11    order_details ON pizzas.pizza_id = order_details.pizza_id
12 GROUP BY pizza_types.category;
```

| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|--------------|---|--------------------|
| | | | |
| | category | total_quantity_of_each_pizza_category_ordered | |
| ▶ | Classic | 14888 | |
| | Veggie | 11649 | |
| | Supreme | 11987 | |
| | Chicken | 11050 | |

Question 8:

Determine the distribution of orders by hour of the day.

Query:

```
SELECT
    HOUR(order_time) as Every_Hour, COUNT(order_id) as Order_by_hour
FROM
    orders
GROUP BY HOUR(order_time) order by Order_by_hour desc ;
```

Output:

| | |
|---|--|
| 1 | -- Determine the distribution of orders by hour of the day. |
| 2 | • SELECT |
| 3 | HOUR(order_time) as Every_Hour, COUNT(order_id) as Order_by_hour |
| 4 | FROM |
| 5 | orders |
| 6 | GROUP BY HOUR(order_time) order by Order_by_hour desc ; |

| | | | |
|-------------|--------------|---------------|--------------------|
| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
| | | | |
| | Every_Hour | Order_by_hour | |
| ▶ | 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 | |

Question 9:

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

Query:

```

SELECT
    category, COUNT(name) AS number_of_pizzas
FROM
    pizza_types
GROUP BY category
ORDER BY number_of_pizzas DESC;

```

Output:


```

1      -- Join relevant tables to find the category
2
3      SELECT
4          category,
5          COUNT(name) AS number_of_pizzas
6      FROM
7          pizza_types
8      GROUP BY
9          category
10     ORDER BY
11         number_of_pizzas DESC;

```

| Result Grid | | Filter Rows: | Export: |
|-------------|----------|------------------|---------|
| | category | number_of_pizzas | |
| ▶ | Supreme | 9 | |
| | Veggie | 9 | |
| | Classic | 8 | |
| | Chicken | 6 | |

Question 10:

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

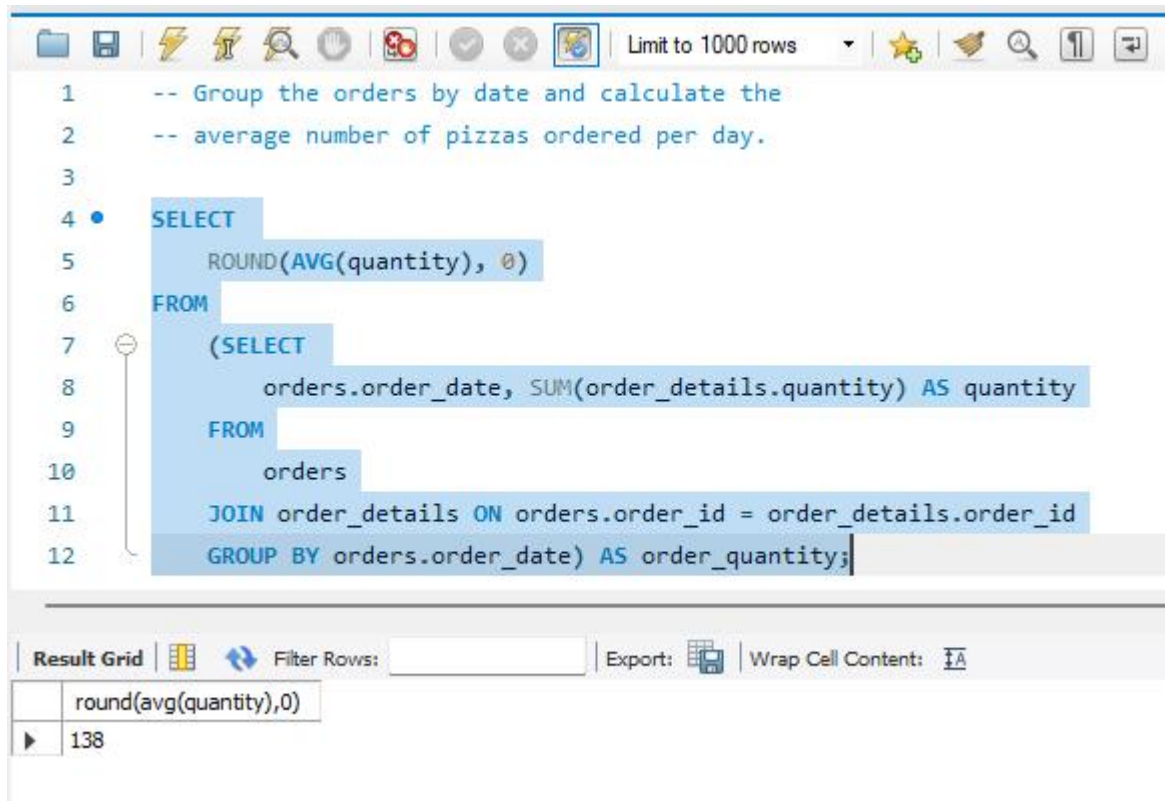
Query:

```

SELECT
    ROUND(AVG(quantity), 0)
FROM
    (SELECT
        orders.order_date, SUM(order_details.quantity) AS quantity
    FROM
        orders
    JOIN order_details ON orders.order_id = order_details.order_id
    GROUP BY orders.order_date) AS order_quantity;

```

Output:



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

Result Grid

| round(avg(quantity),0) |
|------------------------|
| 138 |

Question 11:

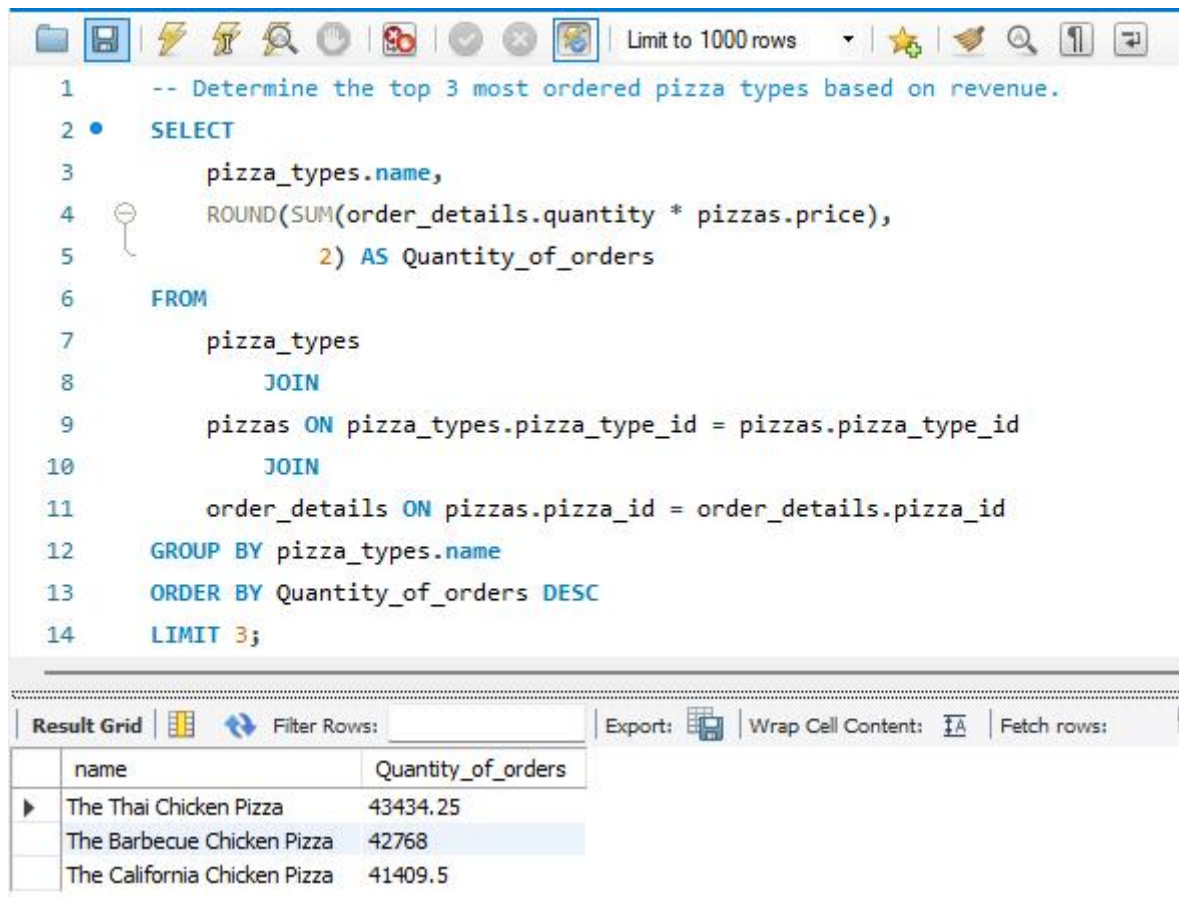
Determine the top 3 most ordered pizza types based on revenue.

Query:

```
SELECT
    pizza_types.name,
    ROUND(SUM(order_details.quantity * pizzas.price),
        2) AS Quantity_of_orders
FROM
    pizza_types
    JOIN
        pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
    JOIN
        order_details ON pizzas.pizza_id = order_details.pizza_id
GROUP BY pizza_types.name
ORDER BY Quantity_of_orders DESC
```

LIMIT 3;

Output:



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

```
1  -- Determine the top 3 most ordered pizza types based on revenue.
2  SELECT
3      pizza_types.name,
4      ROUND(SUM(order_details.quantity * pizzas.price),
5             2) AS Quantity_of_orders
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 pizzas.pizza_id = order_details.pizza_id
12 GROUP BY pizza_types.name
13 ORDER BY Quantity_of_orders DESC
14 LIMIT 3;
```

Below the editor is the 'Result Grid' tab, which displays the following data:

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

Question 12:

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

Query:

```
SELECT
    pizza_types.category,
    ROUND(SUM(pizzas.price * order_details.quantity) / (SELECT
        ROUND(SUM((order_details.quantity * pizzas.price)),
            2) AS total_revenue
    FROM
        order_details
    JOIN
```

pizzas ON pizzas.pizza_id = order_details.pizza_id) * 100,

1) AS revenue

FROM

pizza_types

JOIN

pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id

JOIN

order_details ON pizzas.pizza_id = order_details.pizza_id

GROUP BY pizza_types.category

ORDER BY revenue DESC

LIMIT 5;

Output:

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

| Result Grid | | | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|----------|---------|--------------|---------|--------------------|
| | category | revenue | | | |
| ► | Classic | 26.9 | | | |
| | Supreme | 25.5 | | | |
| | Chicken | 24 | | | |
| | Veggie | 23.7 | | | |

Question 13:

Analyze the cumulative revenue generated over time.

Query:

```
select order_date,  
sum(revenue) over (order by order_date) as cum_revenue  
from  
(SELECT  
    orders.order_date,  
    sum(order_details.quantity * pizzas.price) AS revenue  
FROM  
    order_details  
    JOIN  
    pizzas ON order_details.pizza_id = pizzas.pizza_id  
join orders on  
orders.order_id=order_details.order_id group by orders.order_date)as sales;
```

Output:

| | |
|----|---|
| 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 |
| 6 | orders.order_date, |
| 7 | sum(order_details.quantity * pizzas.price) AS revenue |
| 8 | FROM |
| 9 | order_details |
| 10 | JOIN |
| 11 | pizzas ON order_details.pizza_id = pizzas.pizza_id |
| 12 | join orders on |
| 13 | orders.order_id=order_details.order_id group by orders.order_date)as sales; |

| | | | |
|-------------|--------------------|---------|--------------------|
| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
| order_date | cum_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 | | |
| 2015-01-10 | 23990.350000000002 | | |

Question 14:

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

Query:

```

select name, revenue from
(select category,name,revenue,
rank()over (partition by category order by revenue desc) as rn
from
(select
pizza_types.category,pizza_types.name,sum((order_details.quantity)*pizzas.price)
as revenue
from pizza_types

```

join pizzas

on pizza_types.pizza_type_id = pizzas.pizza_type_id

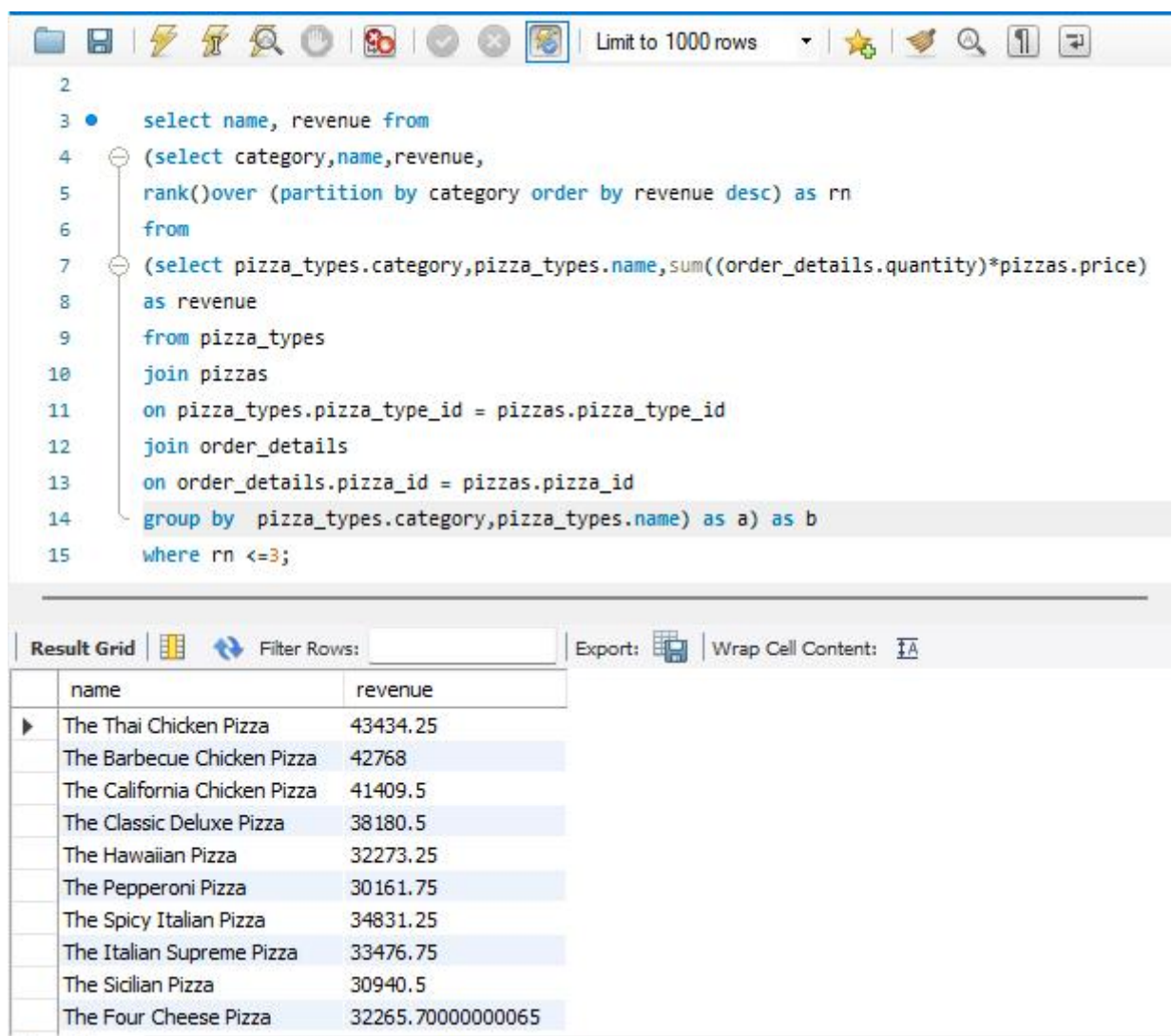
join order_details

on order_details.pizza_id = pizzas.pizza_id

group by pizza_types.category,pizza_types.name) as a) as b

where rn <=3;

Output:



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:

```
2
3 • select name, revenue from
4 (select category,name,revenue,
5 rank()over (partition by category order by revenue desc) as rn
6 from
7 (select pizza_types.category,pizza_types.name,sum((order_details.quantity)*pizzas.price)
8 as revenue
9 from pizza_types
10 join pizzas
11 on pizza_types.pizza_type_id = pizzas.pizza_type_id
12 join order_details
13 on order_details.pizza_id = pizzas.pizza_id
14 group by pizza_types.category,pizza_types.name) as a) as b
15 where rn <=3;
```

Below the query editor is the '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: 'name' and 'revenue'.

| | 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 |
| | The Pepperoni Pizza | 30161.75 |
| | The Spicy Italian Pizza | 34831.25 |
| | The Italian Supreme Pizza | 33476.75 |
| | The Sicilian Pizza | 30940.5 |
| | The Four Cheese Pizza | 32265.70000000065 |