

SQL project

Name:

Ranjeet Chaudhary

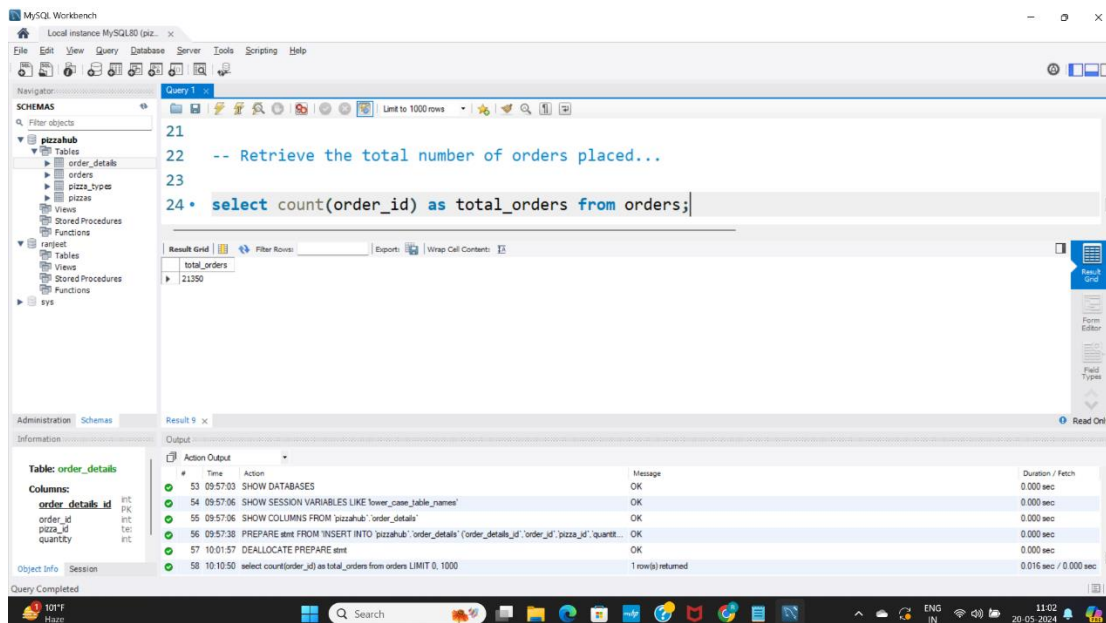
DataSets:

Pizza sales

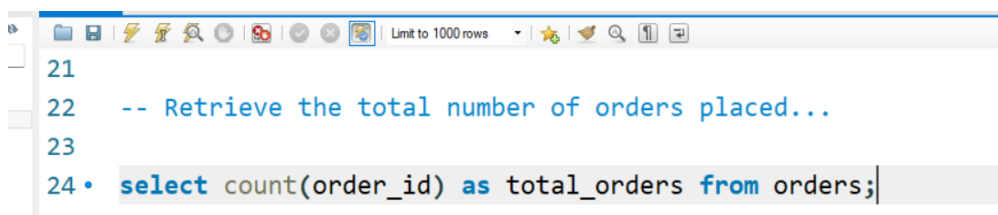
Github: <https://github.com/Ranjeet-pro>

1. Retrieve the total number of orders placed

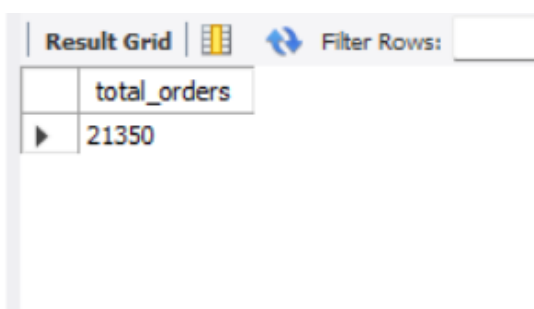
SQL workbench window



SQL Query

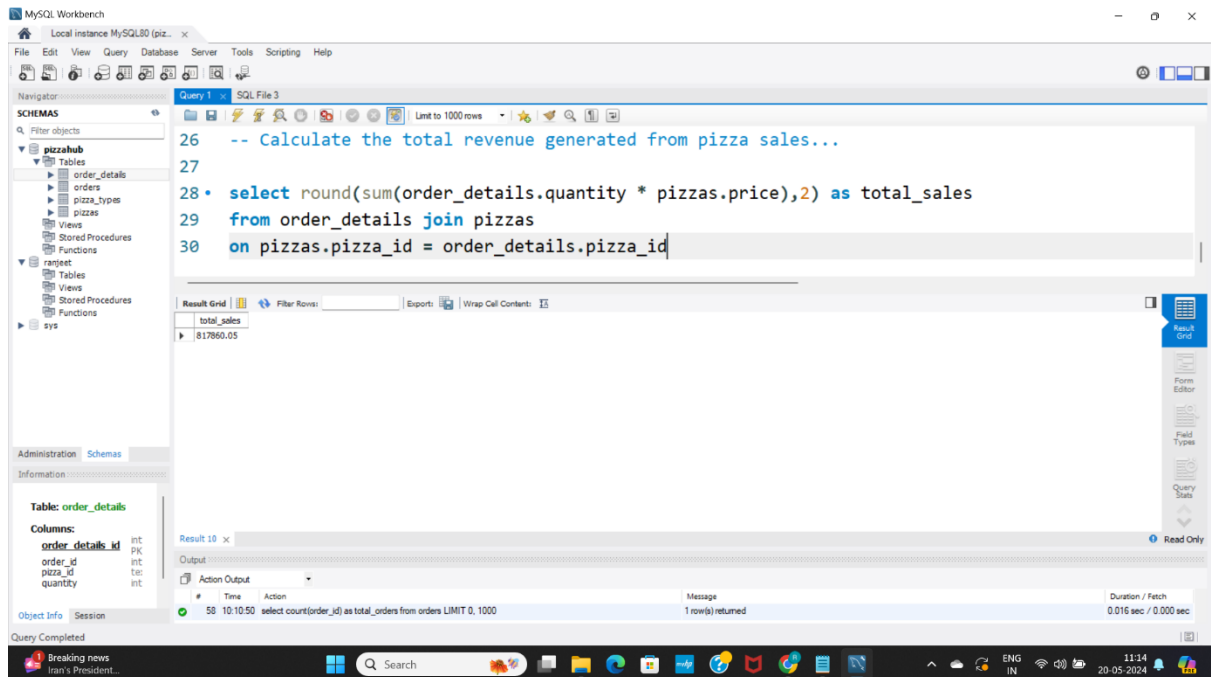


Result output



2. Calculate the total revenue generated from pizza sales.

SQL workbench window



SQL Query

```
27
28 • select round(sum(order_details.quantity * pizzas.price),2) as total_sales
29 from order_details join pizzas
30 on pizzas.pizza_id = order_details.pizza_id
```

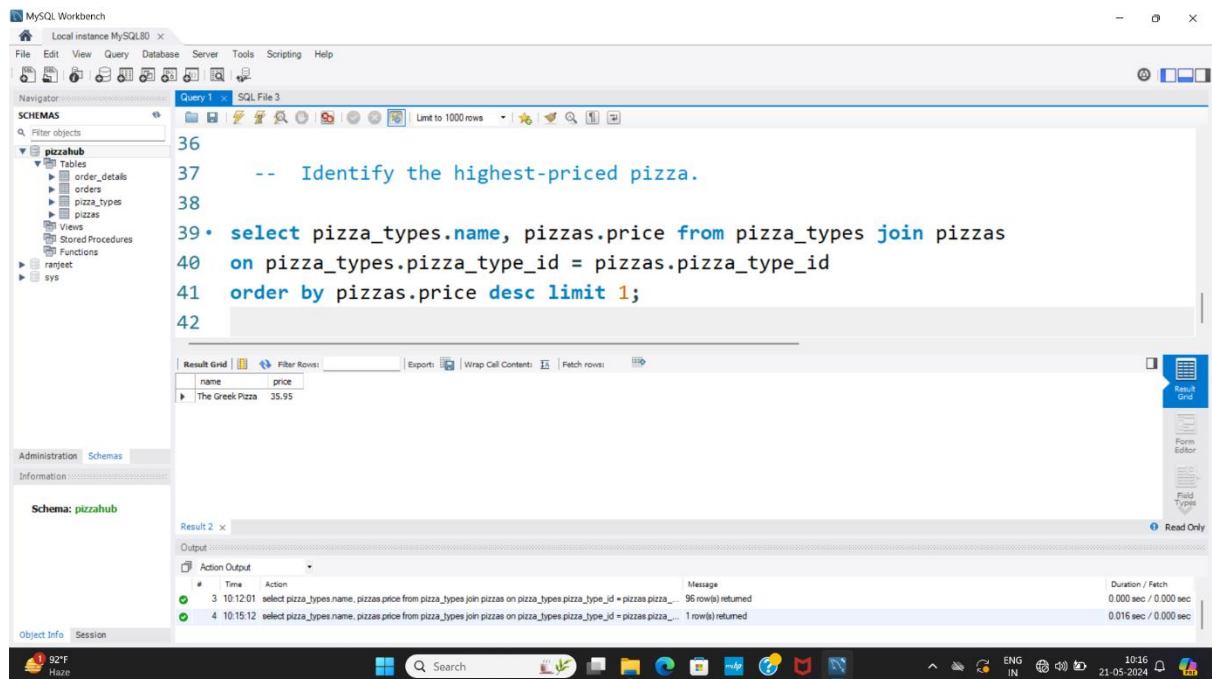
Result output

The close-up of the Result Grid shows the following data:

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

3. Identify the highest-priced pizza

SQL workbench window



SQL Query

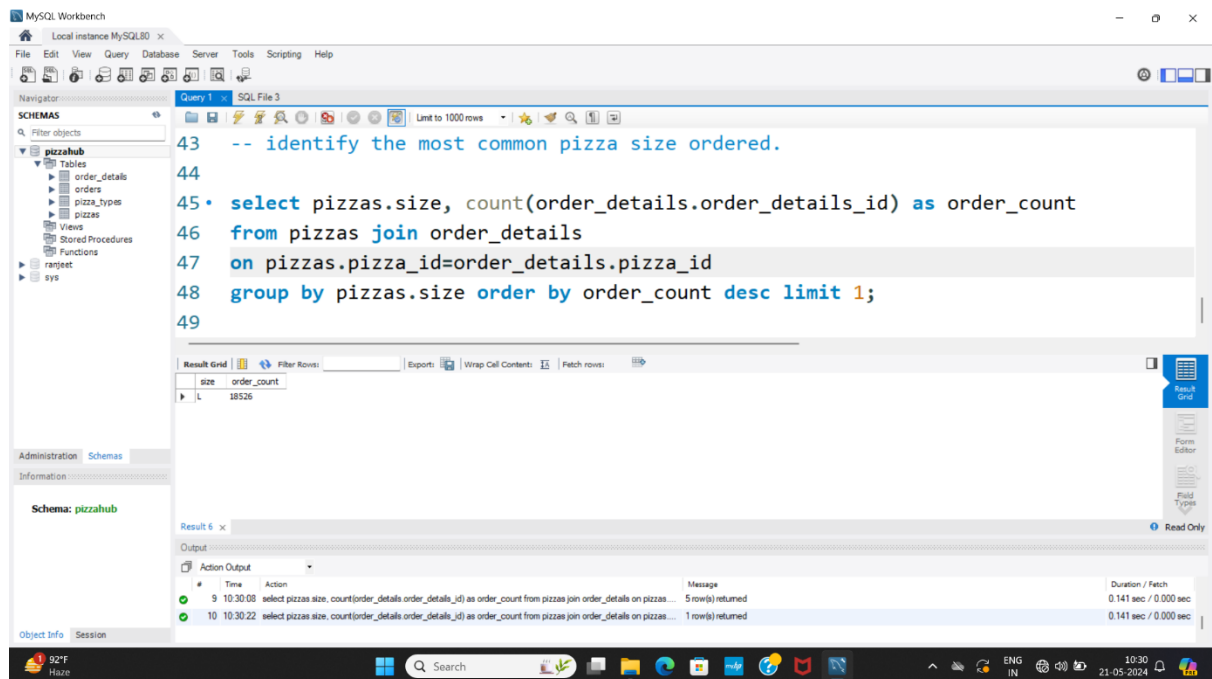
```
36
37  -- Identify the highest-priced pizza.
38
39 • select pizza_types.name, pizzas.price from pizza_types join pizzas
40 on pizza_types.pizza_type_id = pizzas.pizza_type_id
41 order by pizzas.price desc limit 1;
42
```

Result output

| | | |
|-------------|-----------------|-------|
| Result Grid | | |
| | name | price |
| 1 | The Greek Pizza | 35.95 |

4. identify the most common pizza size ordered.

SQL workbench window



SQL Query

```
Query 1 x SQL File 3
Limit to 1000 rows

43 -- identify the most common pizza size ordered.
44
45 • select pizzas.size, count(order_details.order_details_id) as order_count
46 from pizzas join order_details
47 on pizzas.pizza_id=order_details.pizza_id
48 group by pizzas.size order by order_count desc limit 1;
49
```

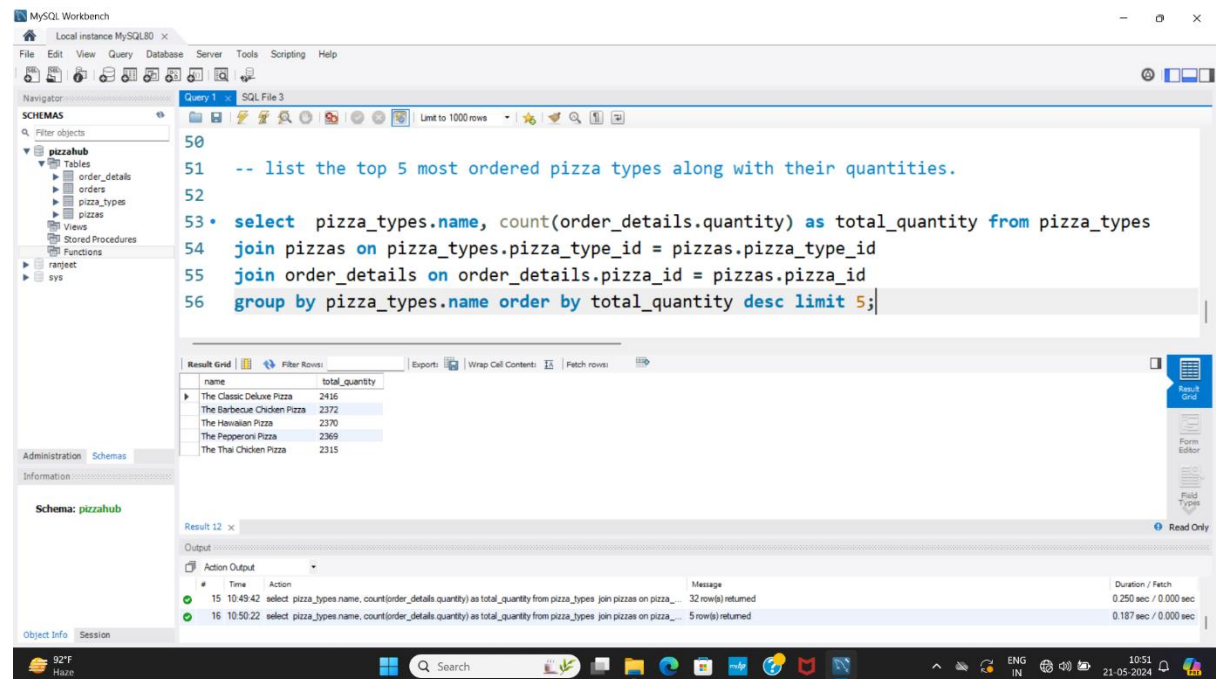
Result output

The close-up shows the 'Result Grid' tab with the following data:

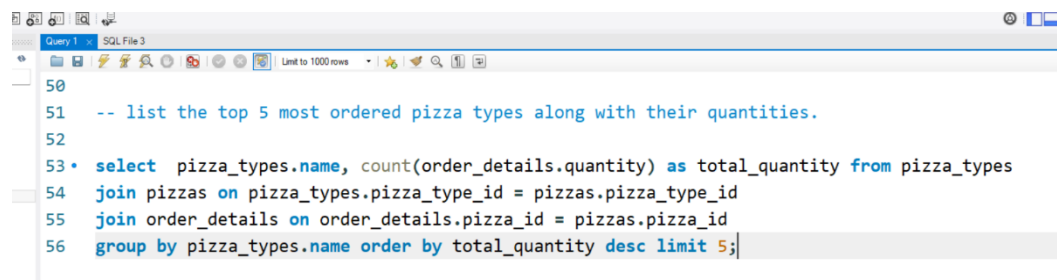
| | size | order_count |
|---|------|-------------|
| ▶ | L | 18526 |

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

SQL workbench window



SQL Query



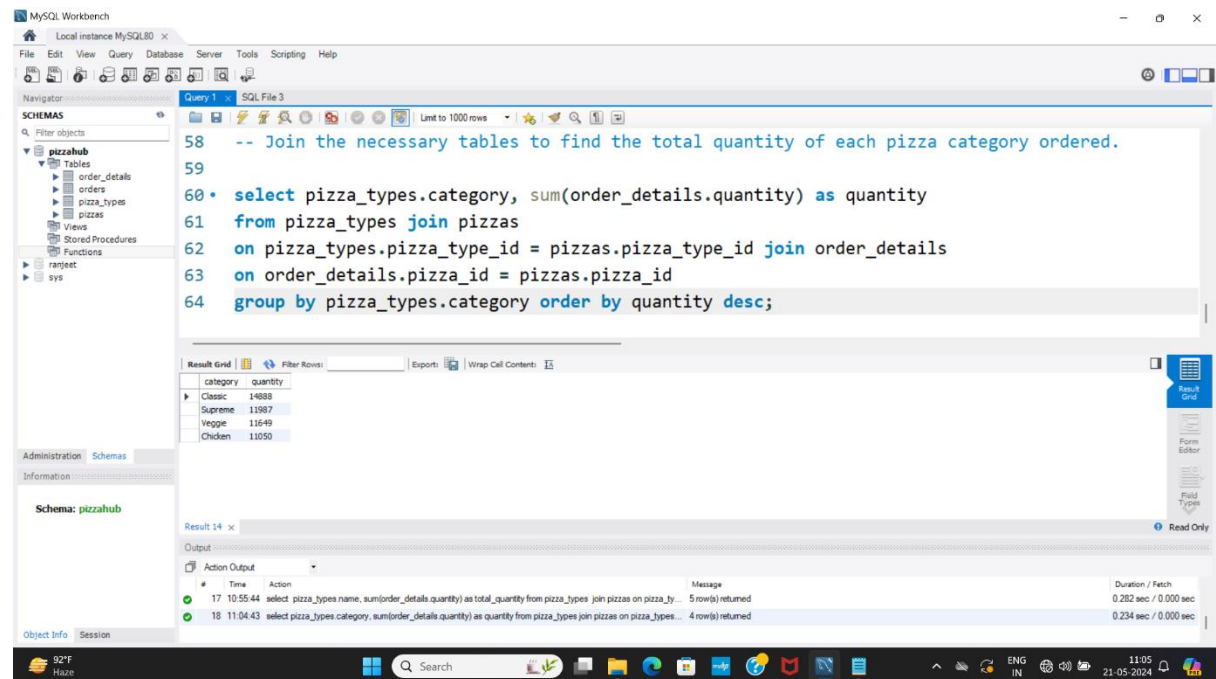
Result output

The screenshot shows the 'Result Grid' with the following data:

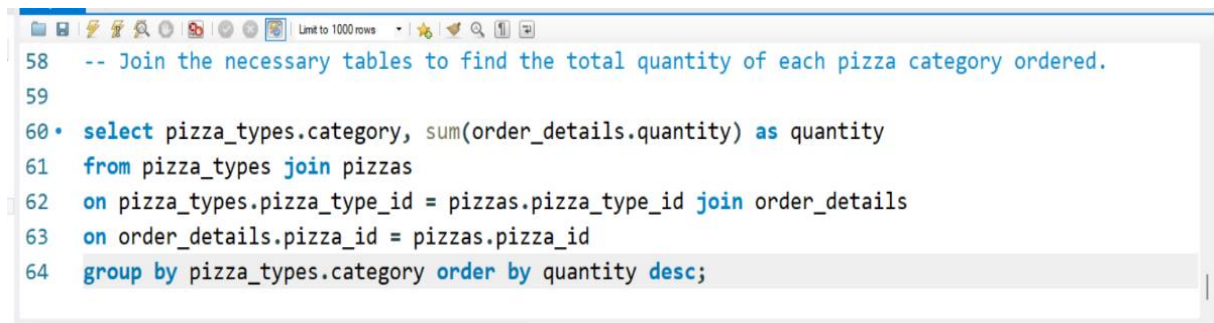
| name | total_quantity |
|----------------------------|----------------|
| The Classic Deluxe Pizza | 2416 |
| The Barbecue Chicken Pizza | 2372 |
| The Hawaiian Pizza | 2370 |
| The Pepperoni Pizza | 2369 |
| The Thai Chicken Pizza | 2315 |

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

SQL workbench window



SQL Query



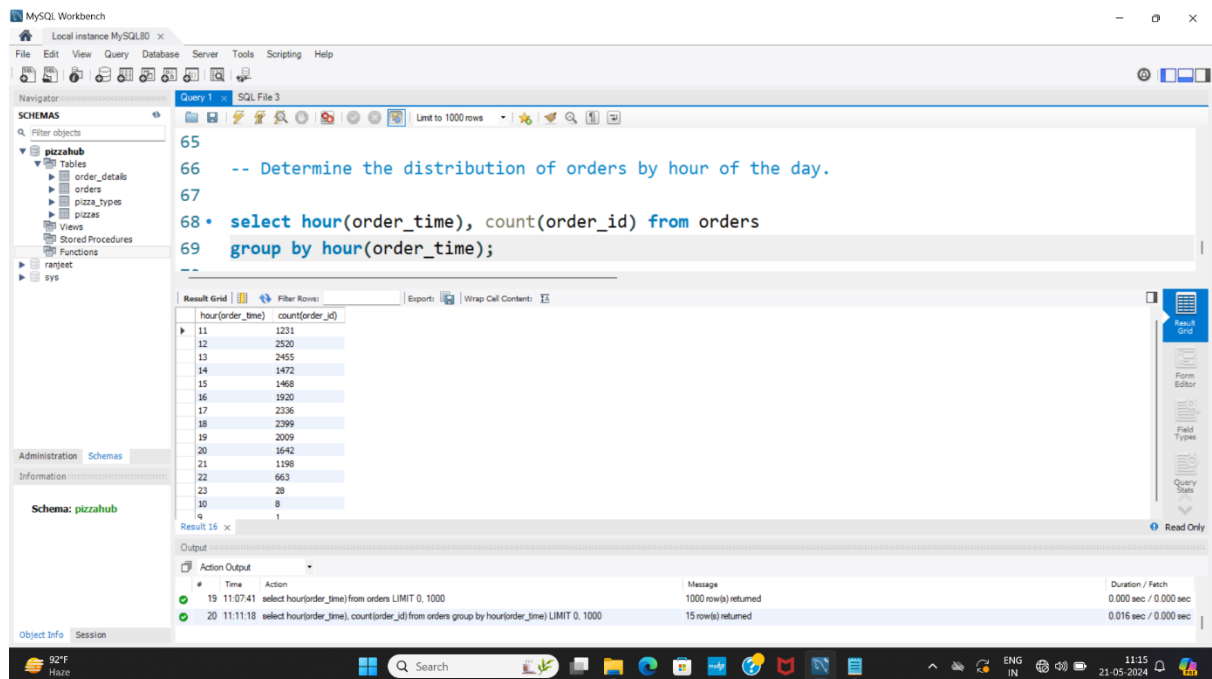
Result output

This image is a close-up of the 'Result Grid' from the previous screenshot, showing the query output. The table has two columns: 'category' and 'quantity'.

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

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

SQL workbench window



SQL Query

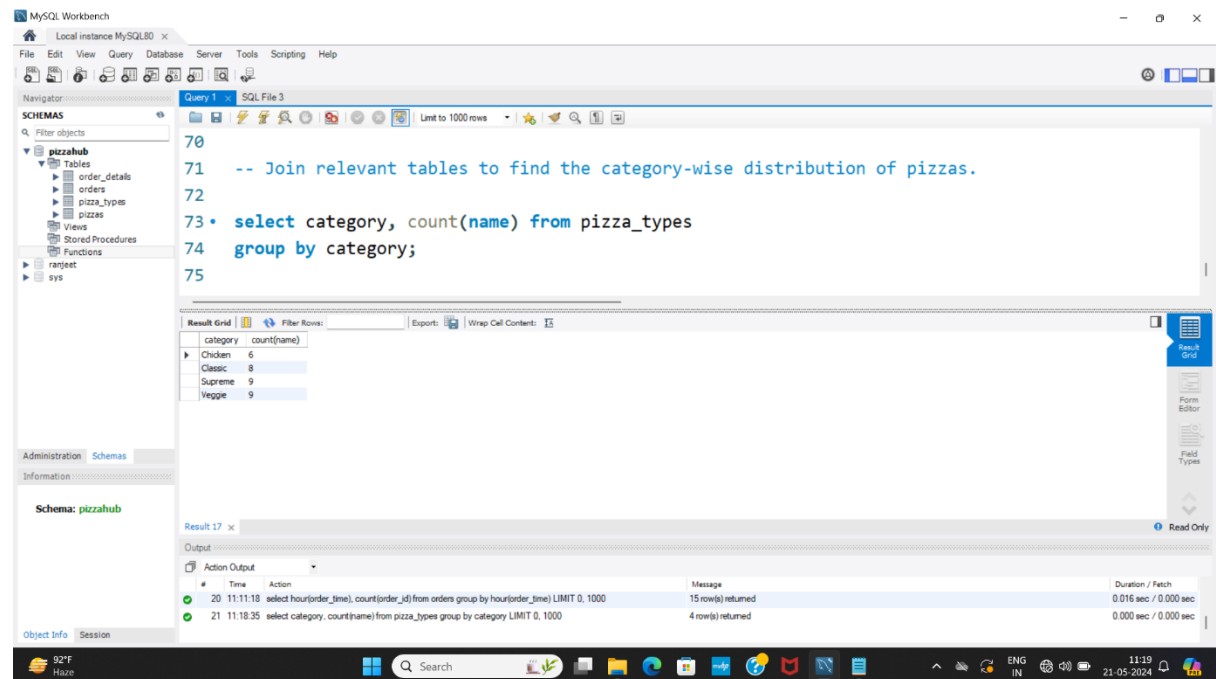
```
65
66 -- Determine the distribution of orders by hour of the day.
67
68 • select hour(order_time), count(order_id) from orders
69   group by hour(order_time);
--
```

Result output

| hour(order_time) | count(order_id) |
|------------------|-----------------|
| 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 |

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

SQL workbench window



SQL Query

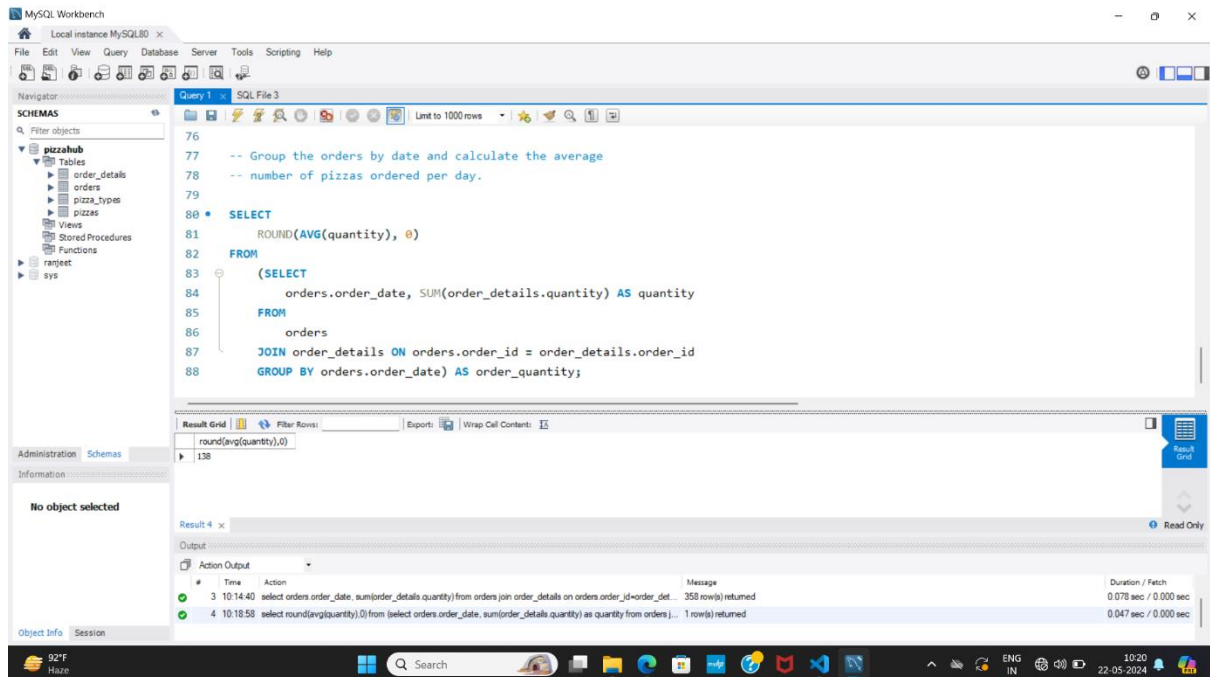
```
70
71 -- Join relevant tables to find the category-wise distribution of pizzas.
72
73 • select category, count(name) from pizza_types
74   group by category;
75
```

Result output

| Result Grid | | Filter Rows |
|-------------|----------|-------------|
| | category | count(name) |
| ▶ | Chicken | 6 |
| | Classic | 8 |
| | Supreme | 9 |
| | Veggie | 9 |

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

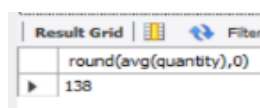
SQL workbench window



SQL Query

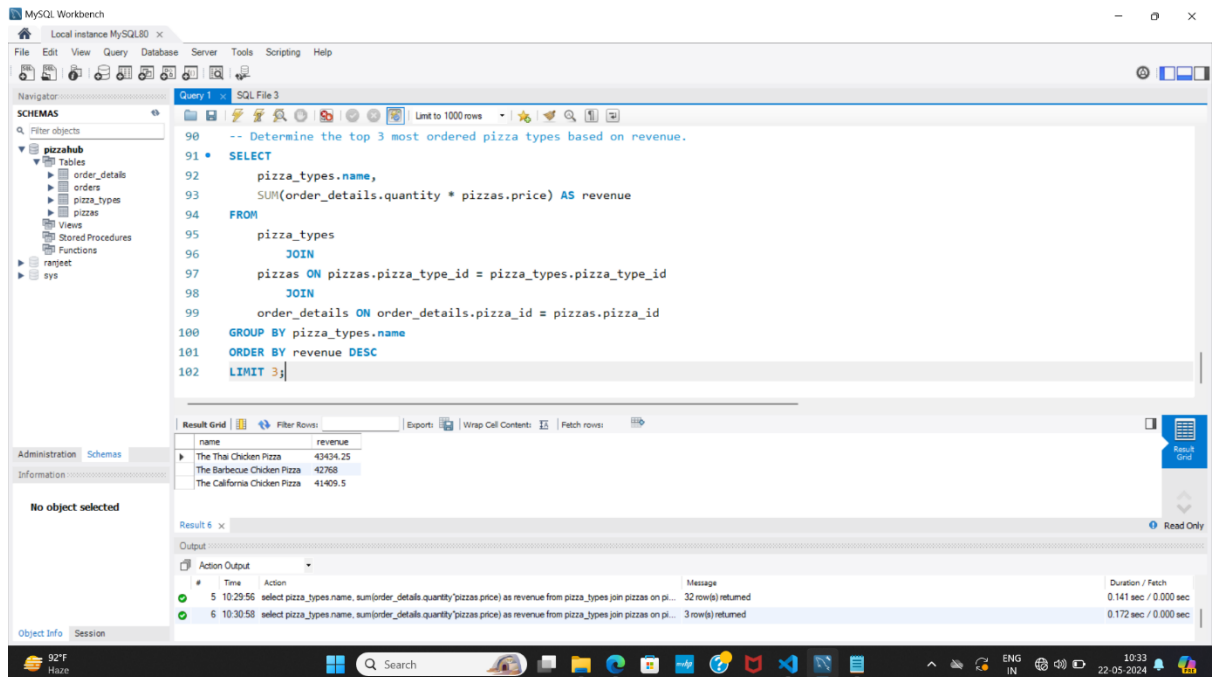
```
76
77 -- Group the orders by date and calculate the average
78 -- number of pizzas ordered per day.
79
80 • SELECT
81     ROUND(AVG(quantity), 0)
82 FROM
83     (SELECT
84         orders.order_date, SUM(order_details.quantity) AS quantity
85     FROM
86         orders
87     JOIN order_details ON orders.order_id = order_details.order_id
88     GROUP BY orders.order_date) AS order_quantity;
```

Result output

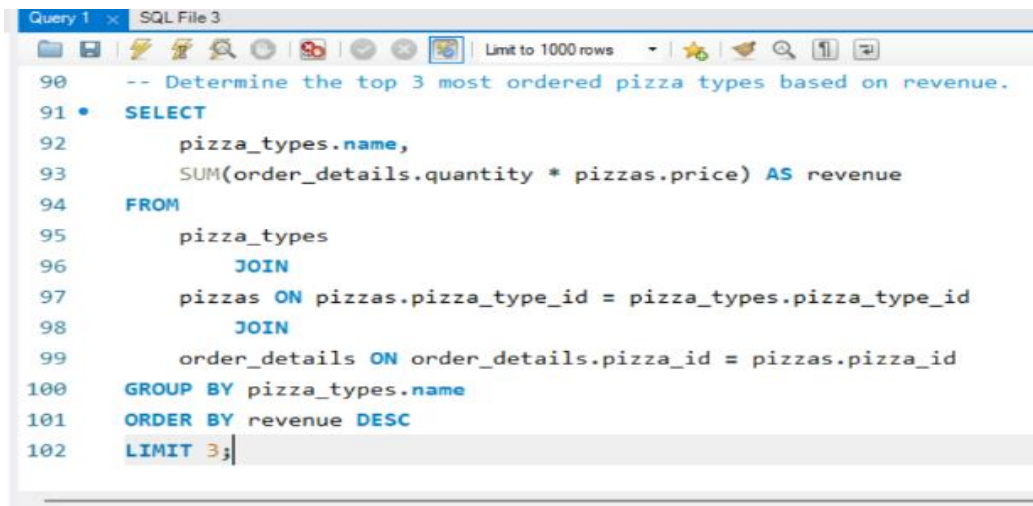


- Determine the top 3 most ordered pizza types based on revenue.

SQL workbench window



SQL Query



Result output

The image shows a close-up of the result output table in MySQL Workbench. The table has two columns: 'name' and 'revenue'.

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

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

SQL workbench window

The screenshot shows the MySQL Workbench interface. On the left is the 'SCHEMAS' pane with a tree view of the database structure, including tables like 'order_details', 'pizzas', and 'pizza_types'. The main editor displays a SQL query (Query 1) that calculates the percentage contribution of each pizza type to total revenue. The query uses a subquery to find total sales and joins it with 'order_details', 'pizzas', and 'pizza_types' tables. The results are shown in a table with columns 'category' and 'revenue'.

```
-- Calculate the percentage contribution of each pizza type to total revenue.
SELECT
  pizza_types.category,
  (SUM(order_details.quantity * pizzas.price) / (SELECT
    ROUND(SUM(order_details.quantity * pizzas.price),
    2) AS total_sales
  FROM
    order_details
    JOIN
      pizzas ON pizzas.pizza_id = order_details.pizza_id)) * 100 AS revenue
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.category
ORDER BY revenue;
```

| category | revenue |
|----------|--------------------|
| Veggie | 23.682590927384577 |
| Chicken | 23.955137556847287 |
| Supreme | 25.45631126009862 |
| Classic | 26.90596025566967 |

SQL Query

This block shows a close-up of the SQL query text from the previous screenshot. The query is designed to calculate the percentage contribution of each pizza type to the total revenue. It uses a subquery to determine the total sales and then joins this with the 'order_details', 'pizzas', and 'pizza_types' tables to calculate the revenue for each category, ordered by revenue.

```
-- Calculate the percentage contribution of each pizza type to total revenue.
SELECT
  pizza_types.category,
  (SUM(order_details.quantity * pizzas.price) / (SELECT
    ROUND(SUM(order_details.quantity * pizzas.price),
    2) AS total_sales
  FROM
    order_details
    JOIN
      pizzas ON pizzas.pizza_id = order_details.pizza_id)) * 100 AS revenue
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.category
ORDER BY revenue;
```

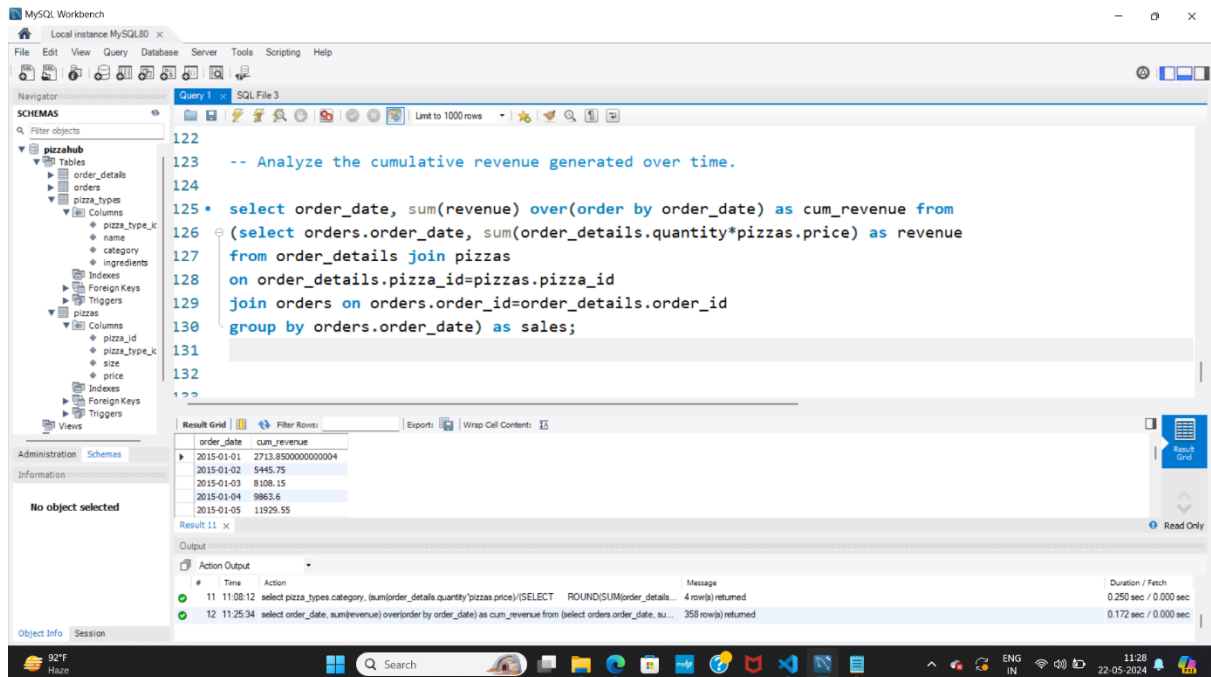
Result output

The 'Result Grid' pane displays the output of the SQL query. It shows a table with two columns: 'category' and 'revenue'. The results are sorted by revenue in descending order.

| category | revenue |
|----------|--------------------|
| Veggie | 23.682590927384577 |
| Chicken | 23.955137556847287 |
| Supreme | 25.45631126009862 |
| Classic | 26.90596025566967 |

12. Analyze the cumulative revenue generated over time.

SQL workbench window



SQL Query

```
22
23 -- Analyze the cumulative revenue generated over time.
24
25 • select order_date, sum(revenue) over(order by order_date) as cum_revenue from
26 (select orders.order_date, sum(order_details.quantity*pizzas.price) as revenue
27 from order_details join pizzas
28 on order_details.pizza_id=pizzas.pizza_id
29 join orders on orders.order_id=order_details.order_id
30 group by orders.order_date) as sales;
31
```

Result output

This is a close-up view of the 'Result Grid' from the MySQL Workbench screenshot. It shows a table with two columns: 'order_date' and 'cum_revenue'. The table contains five rows of data, with the first row highlighted. The 'cum_revenue' values are displayed in scientific notation for the first row.

| order_date | cum_revenue |
|------------|-----------------------|
| 2015-01-01 | 2713.8500000000000004 |
| 2015-01-02 | 5445.75 |
| 2015-01-03 | 8108.15 |
| 2015-01-04 | 9863.6 |
| 2015-01-05 | 11929.55 |

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

SQL workbench window

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'pizza' selected. The main editor shows a SQL query (Query 1) that determines the top 3 most ordered pizza types based on revenue for each category. The query uses a subquery to calculate revenue and a window function to rank them. The 'Result Grid' shows the output of the query, displaying columns: category, name, revenue, and rn. The results are sorted by category and then by revenue in descending order, with the top 3 results for each category highlighted. The 'Output' pane at the bottom shows the execution details of the query.




```
133 -- Determine the top 3 most ordered pizza types based on revenue for each pizza category.
134 • select category, name, revenue, rn from
135   (select category, name, revenue,
136    rank() over(partition by category order by revenue desc) as rn from
137    (select pizza_types.category, pizza_types.name,
138     sum(order_details.quantity*pizzas.price) as revenue
139   from pizza_types join pizzas
140   on pizza_types.pizza_type_id=pizzas.pizza_type_id
141   join order_details on order_details.pizza_id=pizzas.pizza_id
142   group by pizza_types.category, pizza_types.name) as a) as b where rn<=3;
```

| category | name | revenue | rn |
|----------|------------------------------|----------|----|
| Chicken | The Thai Chicken Pizza | 43434.25 | 1 |
| Chicken | The Barbecue Chicken Pizza | 42768 | 2 |
| Chicken | The California Chicken Pizza | 41409.5 | 3 |
| Classic | The Classic Deluxe Pizza | 38180.5 | 1 |
| Classic | The Hawaiian Pizza | 32273.25 | 2 |
| Classic | The Pepporoni Pizza | 30161.75 | 3 |
| Supreme | The Spicy Italian Pizza | 34831.25 | 1 |
| Supreme | The Italian Supreme Pizza | 33476.75 | 2 |
| Supreme | The Sicilian Pizza | 30940.5 | 3 |

SQL Query

```
133 -- Determine the top 3 most ordered pizza types based on revenue for each pizza category.
134 • select category, name, revenue, rn from
135   (select category, name, revenue,
136    rank() over(partition by category order by revenue desc) as rn from
137    (select pizza_types.category, pizza_types.name,
138     sum(order_details.quantity*pizzas.price) as revenue
139   from pizza_types join pizzas
140   on pizza_types.pizza_type_id=pizzas.pizza_type_id
141   join order_details on order_details.pizza_id=pizzas.pizza_id
142   group by pizza_types.category, pizza_types.name) as a) as b where rn<=3;
```


Result output

| Result Grid   Filter Rows: <input type="text"/> Exports:  Write | | | | |
|--|----------|------------------------------|----------|---|
| | category | name | revenue | m |
| ▶ | Chicken | The Thai Chicken Pizza | 43434.25 | 1 |
| | Chicken | The Barbecue Chicken Pizza | 42768 | 2 |
| | Chicken | The California Chicken Pizza | 41409.5 | 3 |
| | Classic | The Classic Deluxe Pizza | 38180.5 | 1 |
| | Classic | The Hawaiian Pizza | 32273.25 | 2 |
| | Classic | The Pepperoni Pizza | 30161.75 | 3 |
| | Supreme | The Spicy Italian Pizza | 34831.25 | 1 |
| | Supreme | The Italian Supreme Pizza | 33476.75 | 2 |
| | Supreme | The Sicilian Pizza | 30940.5 | 3 |
| | Supreme | The Sicilian Pizza | 30940.5 | 3 |

Result 14 