EDA OF PIZZA SALES USING SQL

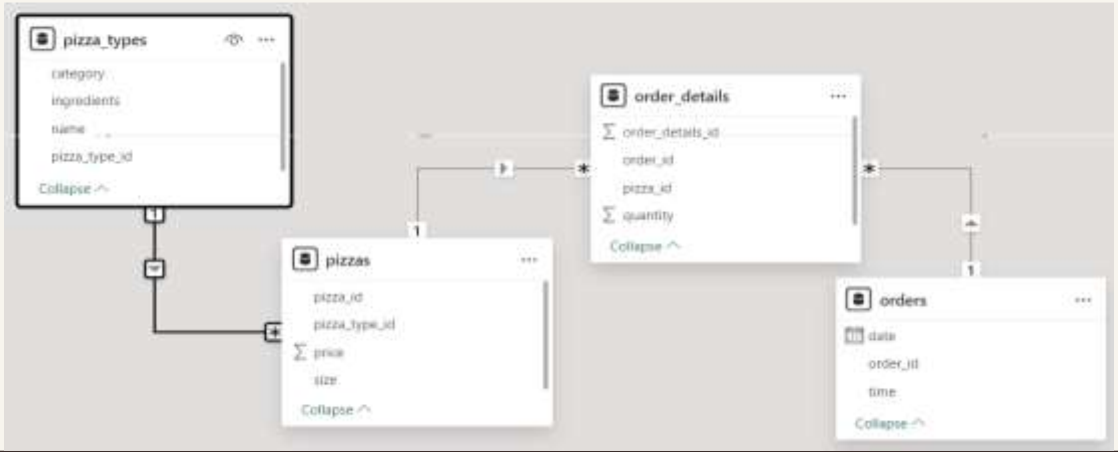
Naga Pavan Kumar

HELLO

This is Naga Pavan kumar in this project I tried to answer the questions relate to the pizza sales in 2015 using the SQL queries.

The data set contains the files

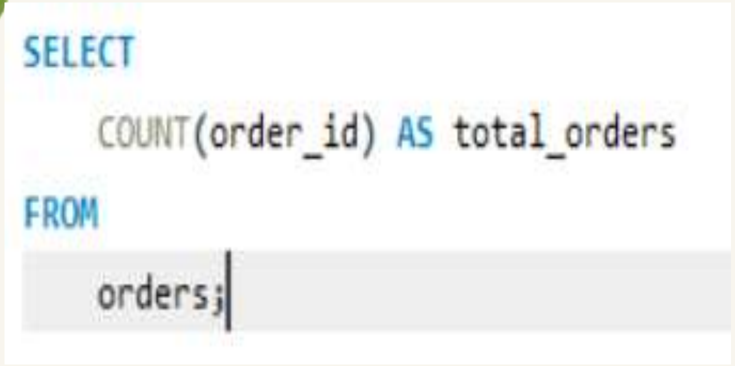
- pizzas.csv: Contains information about pizza types.
- pizza_types.csv: Provides details about pizza categories and prices.
- orders.csv: Includes order information (order IDs, timestamps, etc.).
- orders_details.csv: Contains transaction details (order items, quantities, and amounts).
- These are having the schema as



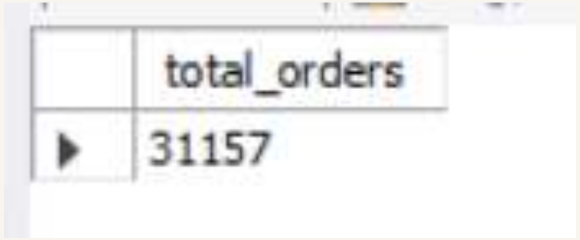
Questions to be answered

- 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.
- 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.
- 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.



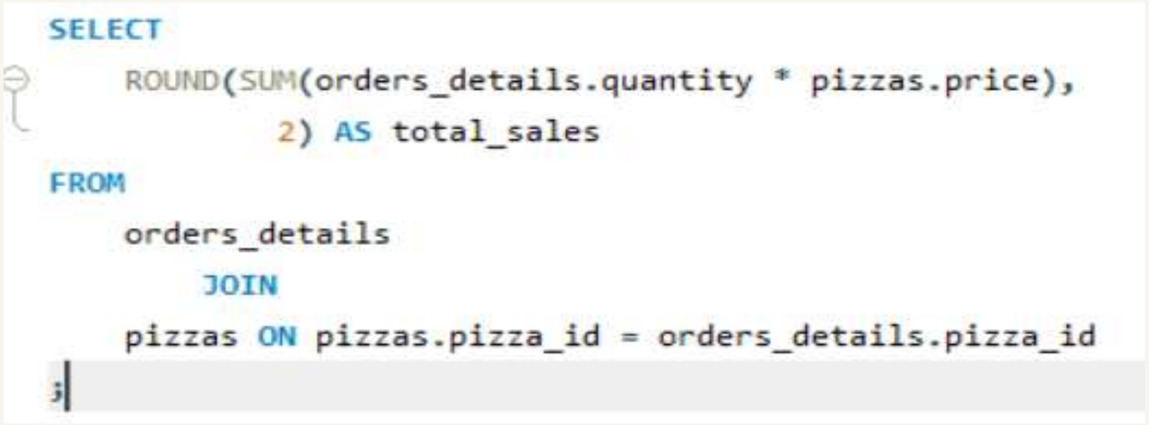
```
SELECT
    COUNT(order_id) AS total_orders
FROM
    orders;
```



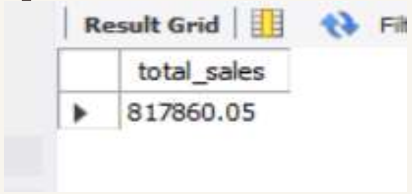
```
  total_orders
▶ 31157
```

The sql aggregate function count is used for the order_id column in the data set to find the total number of orders

Calculate the total revenue generated from pizza sales.



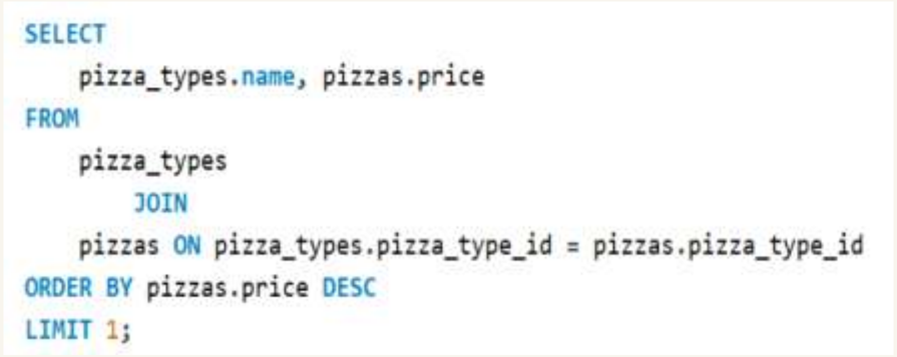
```
SELECT
    ROUND(SUM(orders_details.quantity * pizzas.price),
            2) AS total_sales
FROM
    orders_details
        JOIN
    pizzas ON pizzas.pizza_id = orders_details.pizza_id
;
```



```
total_sales
817860.05
```

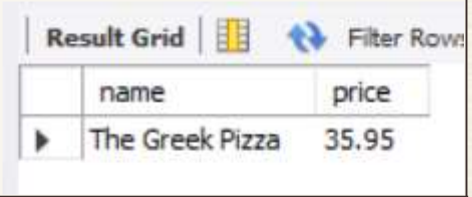
- To solve this question the revenue is obtained by multiplying quantity and price but as these two columns are not in the same table we should apply a join based pizza_id which is common column in the two tables.
- And used the ROUND to limit the result to 2 decimal places.

Identify the highest-priced pizza.



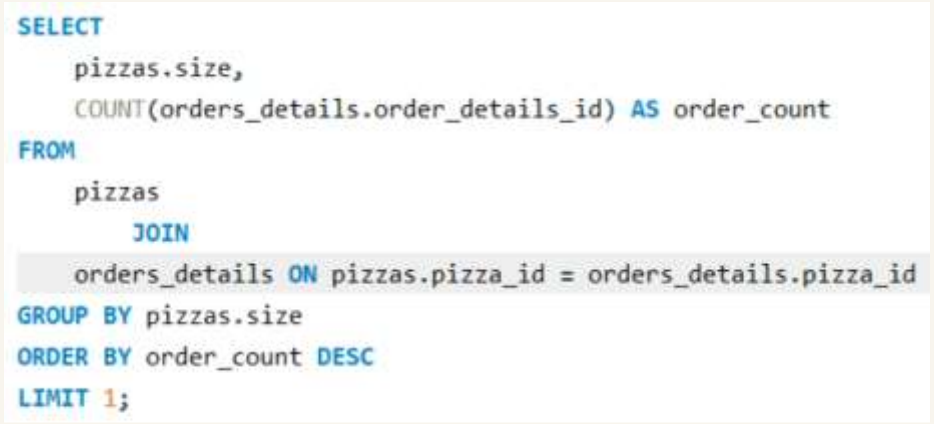
```
SELECT
    pizza_types.name, pizzas.price
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
ORDER BY pizzas.price DESC
LIMIT 1;
```

To solve this same JOIN is applied as the name and price where in the pizza_types and pizzas tables and the order by is applied to align by descending using DESC and the LIMIT is applied as 1 to get the highest priced pizza.



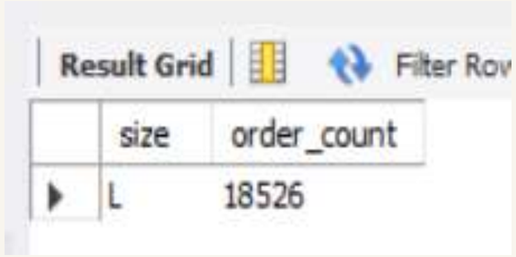
name	price
The Greek Pizza	35.95

Identify the most common pizza size ordered.



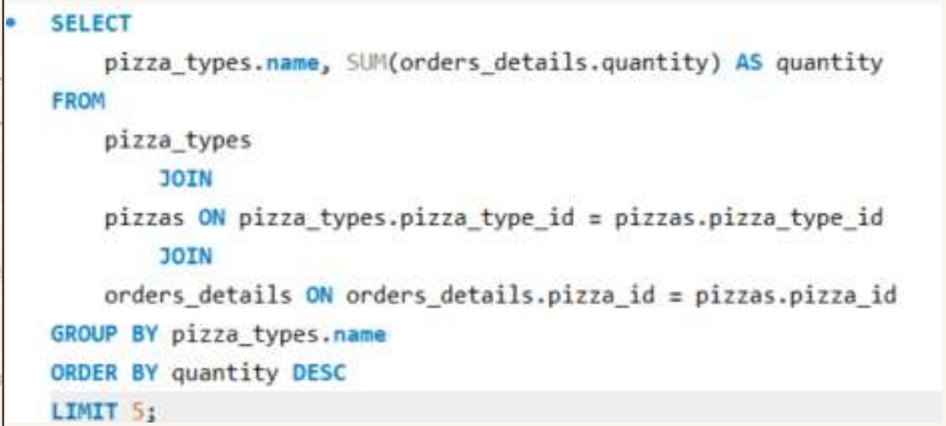
```
SELECT
    pizzas.size,
    COUNT(orders_details.order_details_id) AS order_count
FROM
    pizzas
        JOIN
    orders_details ON pizzas.pizza_id = orders_details.pizza_id
GROUP BY pizzas.size
ORDER BY order_count DESC
LIMIT 1;
```

To solve the question the aggregate function count is used on the order_details_id to know the order count and then as the size is pizza size is in pizzas JOIN is applied based on the pizza_id then the details where GROUP BY size, ORDER BY count using DESC and LIMIT by 1.



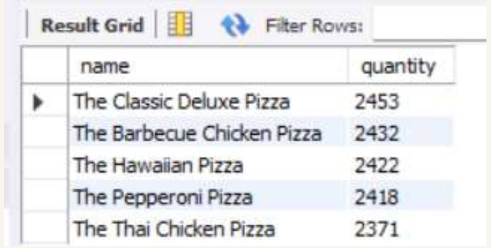
size	order_count
L	18526

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



```
SELECT
    pizza_types.name, SUM(orders_details.quantity) AS quantity
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
        JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.name
ORDER BY quantity DESC
LIMIT 5;
```

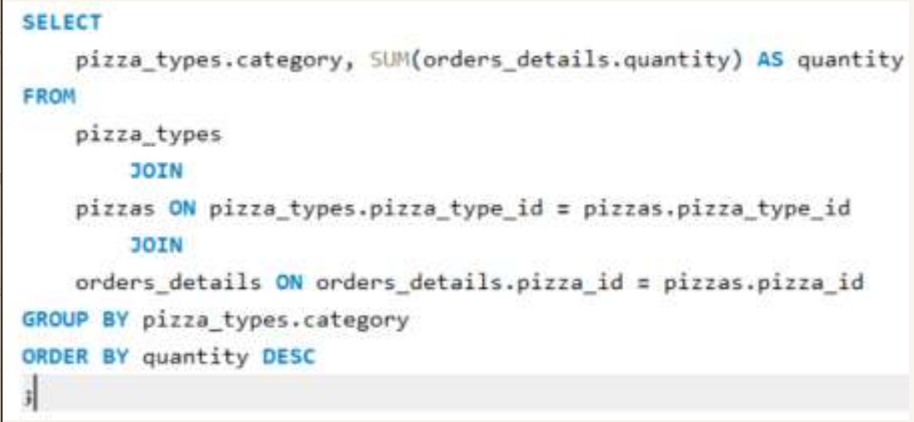
To solve these the data required is in two unconnected tables but can be joined as the pizzas have both pizza_id and pizza_type_id can be joined by two INNER JOINS and GROUP BY name and ORDER BY quantity and LIMIT by 5.



Result Grid | Filter Rows:

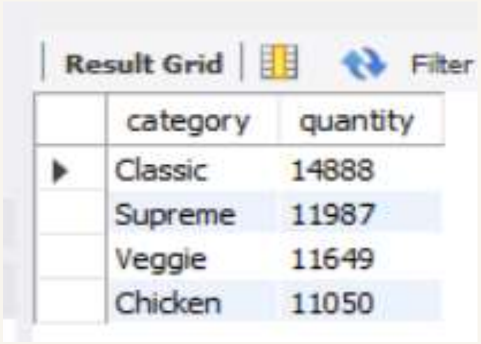
name	quantity
The Classic Deluxe Pizza	2453
The Barbecue Chicken Pizza	2432
The Hawaiian Pizza	2422
The Pepperoni Pizza	2418
The Thai Chicken Pizza	2371

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



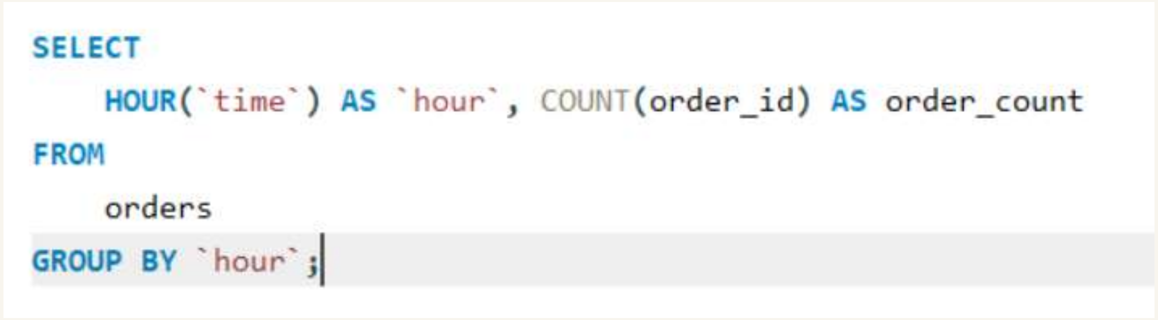
```
SELECT
    pizza_types.category, SUM(orders_details.quantity) AS quantity
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
        JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category
ORDER BY quantity DESC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

To solve these the data required is in two unconnected tables but can be be joined as the pizzas have both pizza_id and pizza_type_id can be joined by two INNER JOINS and GROUP BY category and ORDER BY quantity.



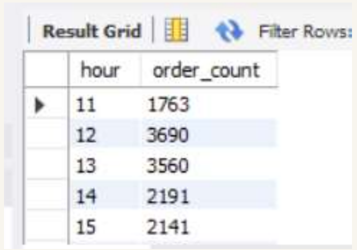
category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

Determine the distribution of orders by hour of the day



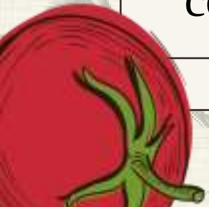
```
SELECT
    HOUR(`time`) AS `hour`, COUNT(order_id) AS order_count
FROM
    orders
GROUP BY `hour`;
```

The HOUR command in the sql gives the hour part of the date time and the order count by order_id COUNT and the results are grouped by the use of the hour here **backticks** are applied to the name hour as it is the command in the sql

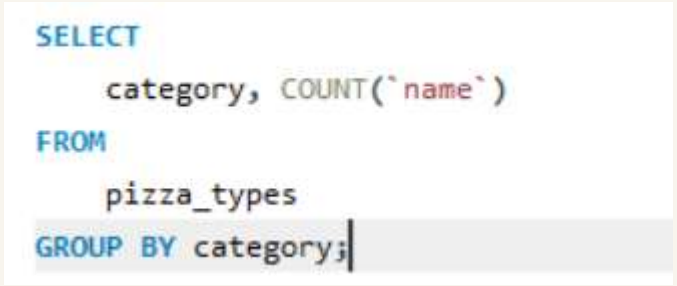


Result Grid | Filter Rows:

```
hour  order_count
11    1763
12    3690
13    3560
14    2191
15    2141
```

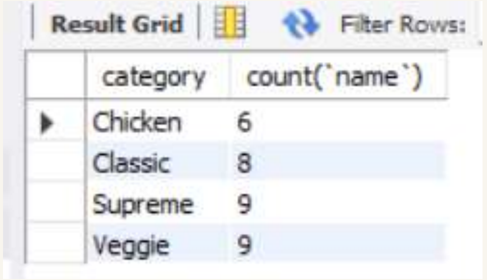


Join relevant tables to find the category-wise distribution of pizzas.



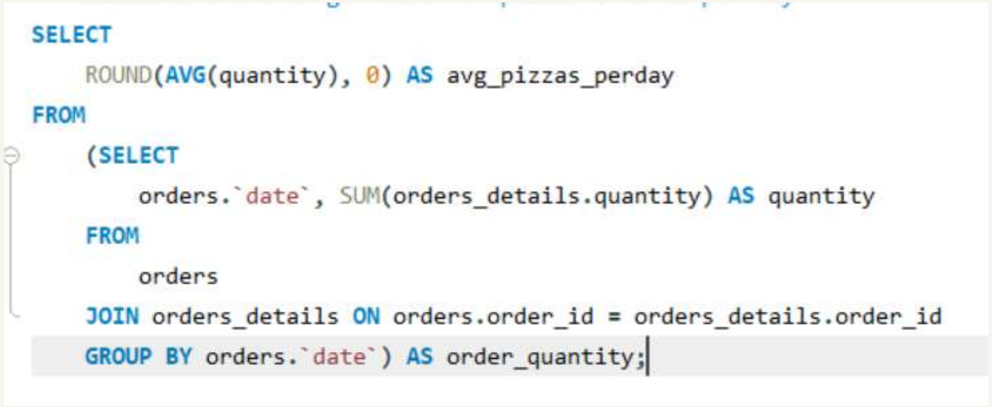
```
SELECT
    category, COUNT(`name`)
FROM
    pizza_types
GROUP BY category;
```

To solve this the COUNT is applied to names of the pizzas and GROUP BY to the category from the table pizza_types



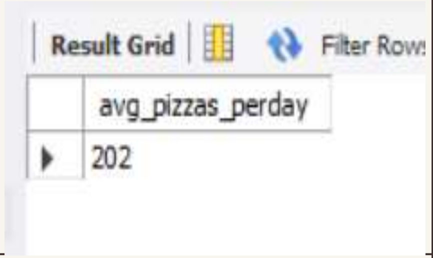
category	count(`name`)
Chicken	6
Classic	8
Supreme	9
Veggie	9

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



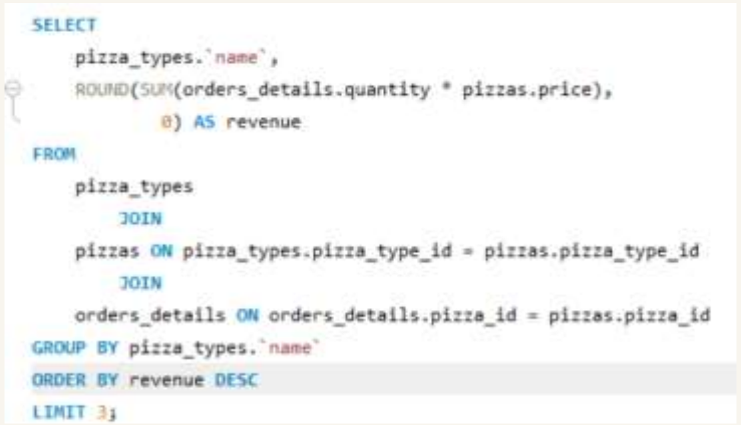
```
SELECT
    ROUND(AVG(quantity), 0) AS avg_pizzas_perday
FROM
    (SELECT
        orders.`date`, SUM(orders_details.quantity) AS quantity
    FROM
        orders
    JOIN orders_details ON orders.order_id = orders_details.order_id
    GROUP BY orders.`date`) AS order_quantity;
```

To get the average pizzas ordered by date first the AVG quantity was taken by rounding off to 0 then a subquery was written to get the order quantity by date and sum of quanity was taken joined by order_id and GROUP BY date. from the orders and order_details tables

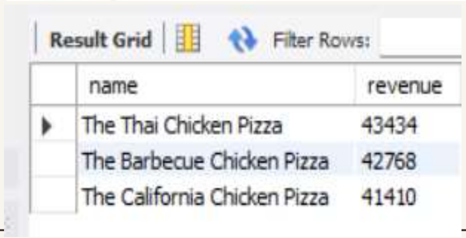


```
avg_pizzas_perday
202
```

Determine the top 3 most ordered pizza types based on revenue



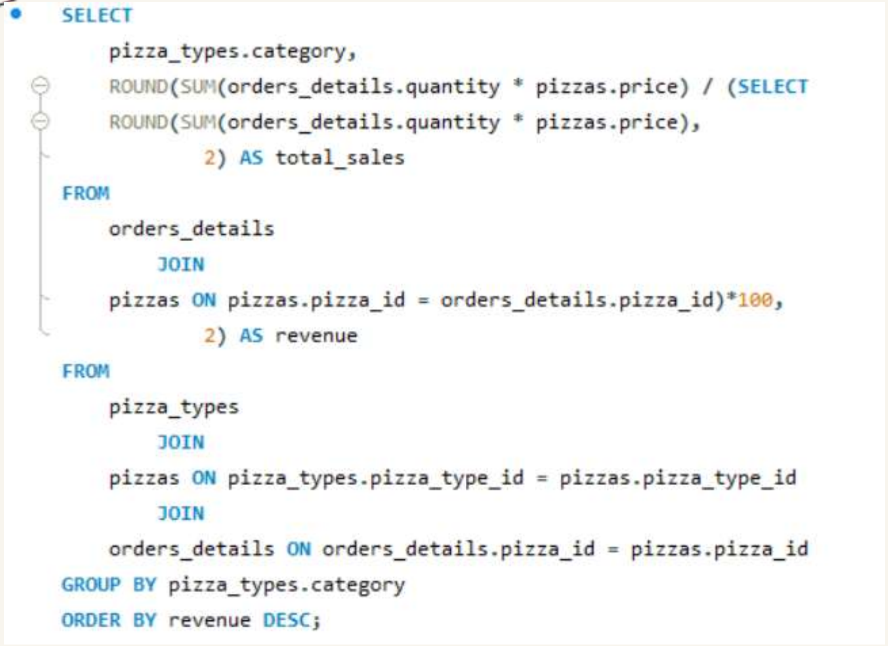
```
SELECT
    pizza_types.`name`,
    ROUND(SUM(orders_details.quantity * pizzas.price),
            0) AS revenue
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
        JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.`name`
ORDER BY revenue DESC
LIMIT 3;
```



```
  name                          revenue
▶ The Thai Chicken Pizza        43434
  The Barbecue Chicken Pizza    42768
  The California Chicken Pizza  41410
```

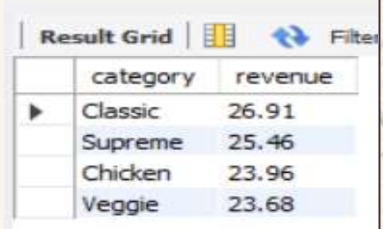
To get this the revenue and the pizza types has to be compared which are in two uncounted tables pizza_types and the order_details but can be joined by the pizzas table by the use of two INNER JOINS because it has pizza_id and pizza_type_id and then the data is GROUP BY pizza_types and ORDER BY revenue DESC with LIMIT as 3.

Calculate the percentage contribution of each pizza type to total revenue



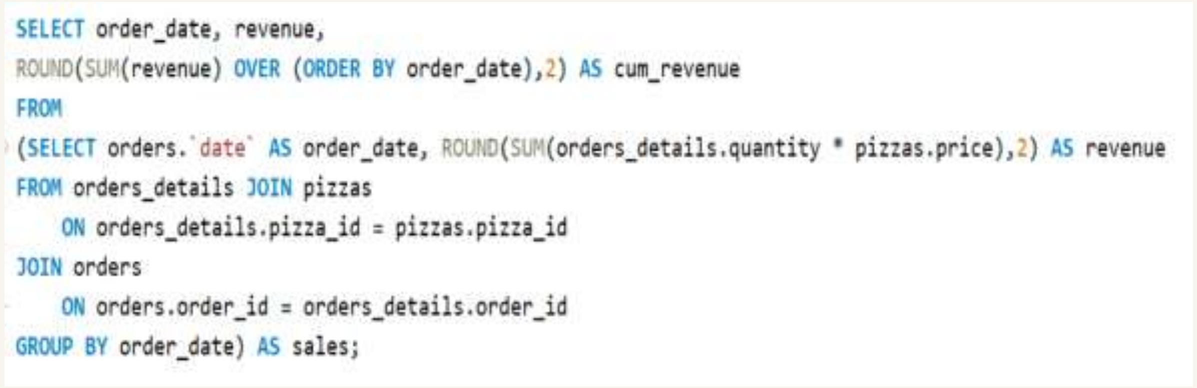
```
SELECT
    pizza_types.category,
    ROUND(SUM(orders_details.quantity * pizzas.price) / (SELECT
    ROUND(SUM(orders_details.quantity * pizzas.price),
            2) AS total_sales
FROM
    orders_details
        JOIN
    pizzas ON pizzas.pizza_id = orders_details.pizza_id)*100,
            2) AS revenue
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
        JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category
ORDER BY revenue DESC;
```

The contribution of each pizza type can be known by writing a query to get the price and dividing it by the revenue (which is obtained by sub query in the code) and then pizza_types and order details were joined by the pizzas table by the use of two INNER JOINS because it has pizza_id and pizza_type_id and then the data is GROUP BY pizza_types.category and ORDER BY revenue DESC.



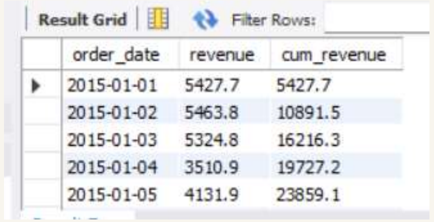
category	revenue
Classic	26.91
Supreme	25.46
Chicken	23.96
Veggie	23.68

Analyze the cumulative revenue generated over time



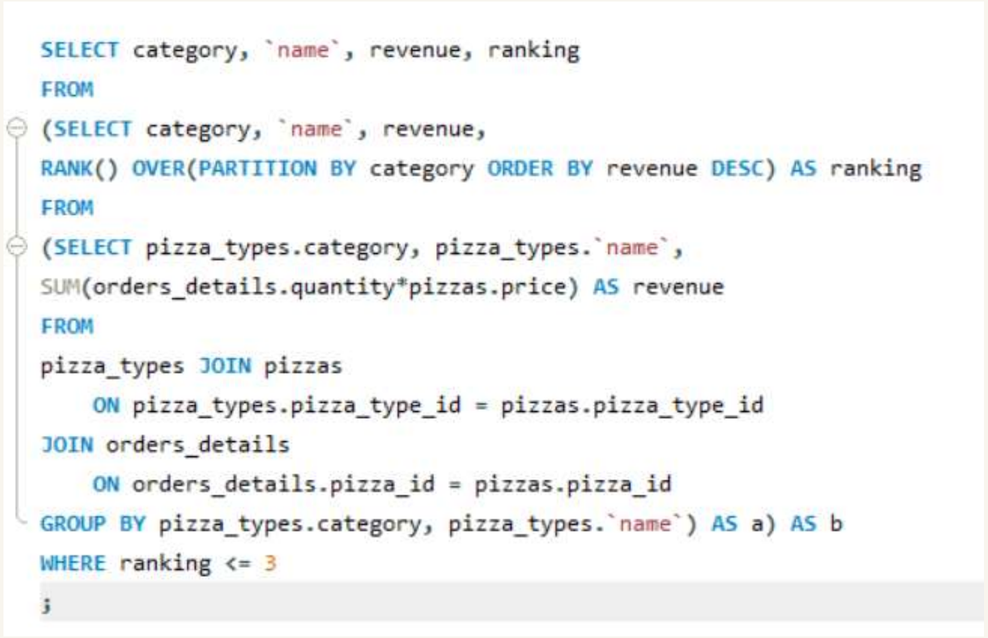
```
SELECT order_date, revenue,
ROUND(SUM(revenue) OVER (ORDER BY order_date),2) AS cum_revenue
FROM
(SELECT orders.`date` AS order_date, ROUND(SUM(orders_details.quantity * pizzas.price),2) AS revenue
FROM orders_details JOIN pizzas
    ON orders_details.pizza_id = pizzas.pizza_id
JOIN orders
    ON orders.order_id = orders_details.order_id
GROUP BY order_date) AS sales;
```

The cumulative sum of the revenue can be obtained by the sum of revenue over the ordered by order_date and the of two INNER JOINS using the order details table as the common point.



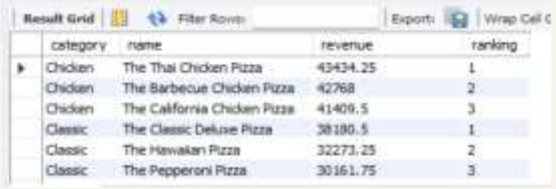
order_date	revenue	cum_revenue
2015-01-01	5427.7	5427.7
2015-01-02	5463.8	10891.5
2015-01-03	5324.8	16216.3
2015-01-04	3510.9	19727.2
2015-01-05	4131.9	23859.1

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



```
SELECT category, `name`, revenue, ranking
FROM
(SELECT category, `name`, revenue,
RANK() OVER(PARTITION BY category ORDER BY revenue DESC) AS ranking
FROM
(SELECT pizza_types.category, pizza_types.`name`,
SUM(orders_details.quantity*pizzas.price) AS revenue
FROM
pizza_types JOIN pizzas
    ON pizza_types.pizza_type_id = pizzas.pizza_type_id
JOIN orders_details
    ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category, pizza_types.`name`) AS a) AS b
WHERE ranking <= 3
;
```

The required information for this query is pizza type, pizza and revenue generated by each pizza and finally we need to rank them so for the info we use TWO INNER JOINS and group them by both categoryand nem of the pizza and finally the ranking is provided by the use of the (OVER, PARTITION BY(using category), ORDER BY (using revenue)) these all are done by the use of sub query in the FROM section.



category	name	revenue	ranking
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

THANKS!