# PIZZA HUTS

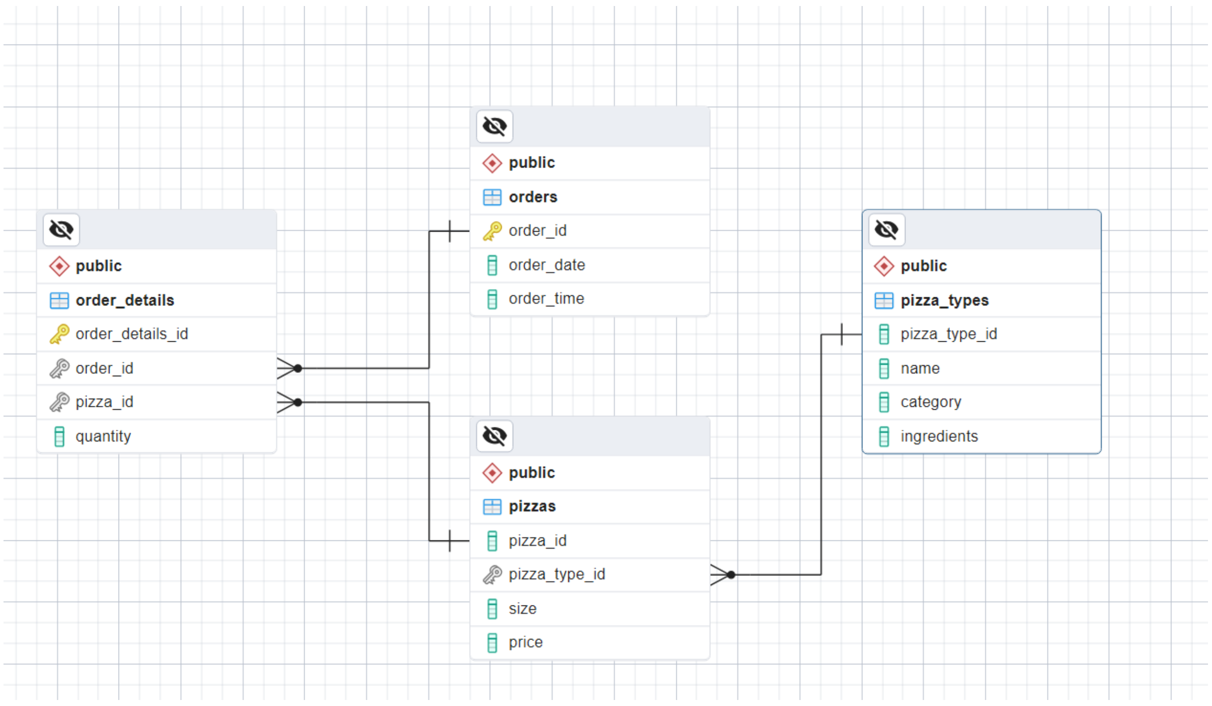
## Introduction to analysis

This Data-Analysis project was developed by utilizing PostgreSQL database and Structured Query Language to query the database to generate useful business insights.
Pizzahut sales data, which was available in CSV format was extracted and loaded too the PostgreSQL database in a structured manner for the analysis by leveraging pgAdmin4 software.
A new database named pizza hut was created and tables with proper schema and keys were structured and available sales data was imported using wizard as a part of the project.

## Methodology used in the analysis

pgAdmin4 was utilized for creating PostgreSQL database, structuring the available csv data into relational format and for querying the data insights. Basic to advanced SQL concepts like complex joins, aggregations, subqueries, window functions were leveraged and utilized for comprehensive analysis

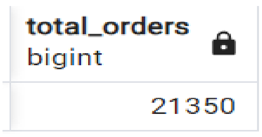
## Schema Visualization of the Database



## Analysis and Insights.

1) Retrieve the total number of orders placed.

```
select count (*) as total_orders from orders;
```



```
total_orders
bigint
       21350
```

2) Calculate the total revenue generated from pizza sales.

```
Select SUM (od.quantity * p.price) AS total_revenue
from pizzas p
join order_details od
on p.pizza_id = od.pizza_id;
```

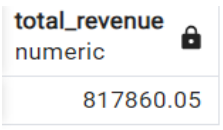
• SELECT: This clause specifies what columns or expressions you want to retrieve. In this case, it selects the total revenue calculated as the sum of quantity times price.

• SUM **(od.quantity * p.price)**: This is the expression used to calculate the revenue for each pizza order. It multiplies the quantity of each pizza ordered (od.quantity) by its price (p.price). Then, SUM () function adds up all these individual revenue values to get the total revenue.

• FROM **pizzas p**: This specifies the table pizzas as the source of pizza information. It assigns an alias p to the pizzas table so that we can refer to it with the shorthand p.

• JOIN **order_details od ON p.pizza_id = od.pizza_id**: This part of the query joins the pizzas table (p) with the order_details table (od) based on the pizza_id column. This ensures that each pizza in the pizzas table is matched with its corresponding order details in the order_details table.

- p.pizza_id refers to the pizza ID column in the pizzas table.
- od.pizza_id refers to the pizza ID column in the order_details table.

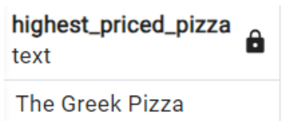


```
total_revenue
numeric
    817860.05
```

3) Identify the highest-priced pizza.

```
SELECT pt.name AS highest_priced_pizza
FROM pizza_types pt
JOIN pizzas p ON pt.pizza_type_id = p.pizza_type_id
ORDER BY p.price DESC
LIMIT 1;
```

1. **SELECT**: Tt selects the name of the pizza type with the highest price.
2. **pt.name AS highest_priced_pizza**: This renames the column name from the pizza_types table as highest_priced_pizza, which is the alias used for the column in the result set.
3. **FROM pizza_types pt**: This specifies the source table as pizza_types. It assigns the alias pt to the pizza_types table for easier reference.
4. **JOIN pizzas p ON pt.pizza_type_id = p.pizza_type_id**: This part of the query joins the pizza_types table (pt) with the pizzas table (p) based on the pizza_type_id column. This ensures that each pizza type is matched with its corresponding pizzas.
   - pt.pizza_type_id refers to the pizza type ID column in the pizza_types table.
   - p.pizza_type_id refers to the pizza type ID column in the pizzas table.
5. **ORDER BY p.price DESC**: This clause orders the result set by the price column in descending order. This ensures that the pizza with the highest price comes first in the result set.
6. **LIMIT 1**: This limits the result set to only one row, which corresponds to the pizza type with the highest price. Without this clause, the query would return all pizza types ordered by price, and we only want the top one.



| highest_priced_pizza text |
|---|
| The Greek Pizza |

4) Identify the most common pizza size ordered.

```
select pizza_size as most_common_pizza_size
from (
        select p.size as pizza_size,
              count(p.size) as total_count
        from order_details od
        join pizzas p on od.pizza_id = p.pizza_id
        group by p.size
        order by total_count desc
       limit 1)
limit 1;
```

1. Select Couse-

This part of the query selects `pizza_size` from the result of the subquery. The alias `most_common_pizza_size` is given to `pizza_size` for clarity.

2. Subquery

- **`select p.size as pizza_size, count(p.size) as total_count`**:
  - This selects two columns: the `size` of the pizza from the `pizzas` table and the count of each size (`total_count`). The `size` is aliased as `pizza_size`.
- **`from order_details od join pizzas p on od.pizza_id = p.pizza_id`**:

    - This performs an inner join between the `order_details` table (`od`) and the `pizzas` table (`p`). The join condition is `od.pizza_id = p.pizza_id`, meaning the query matches each order detail with the corresponding pizza.
- **`group by p.size`**:
    - This groups the result by `p.size`, so the count (`count(p.size)`) is calculated for each pizza size.
- **`order by total_count desc`**:
    - This orders the results by `total_count` in descending order, so the most common pizza size (the one with the highest count) comes first.
- **`limit 1`**:
    - This limits the result to just the top row, which is the most common pizza size based on the count.

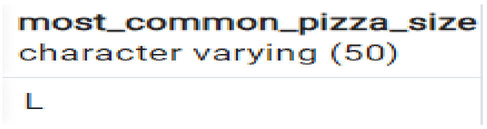
**3.** Final limit 1

- This is somewhat redundant in this context because the subquery already uses `limit 1`, ensuring only one row is returned. Including `limit 1` again in the outer query is not necessary but does not harm the query. It essentially reasserts that only one result should be returned.

## Summary

The query can be summarized as follows:

1. **Join** the `order_details` and `pizzas` tables on the `pizza_id`.
2. **Group** the joined results by `pizza_size` and **count** the occurrences of each size.
3. **Order** the counts in descending order to get the most common size first.
4. **Limit** the result to the top row to get the most common pizza size.
5. **Select** the most common pizza size from the subquery result.



```
most_common_pizza_size
character varying (50)
L
```

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

```
select pt.name as pizza_type, sum(quantity) as total_quantity
from order_details od
join pizzas p on od.pizza_id = p.pizza_id
join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
group by pt.name
order by total_quantity desc
limit 5;
```

## 1. SELECT Clause

- **`pt.name as pizza_type`**:
  - This selects the name of the pizza type from the `pizza_types` table and aliases it as `pizza_type` for clarity.
- **`sum(quantity) as total_quantity`**:
  - This calculates the total quantity ordered for each pizza type by summing the `quantity` column from the `order_details` table and aliases it as `total_quantity`.

## 2. FROM Clause

- **`from order_details od`**:
  - This specifies the `order_details` table, aliased as `od`.
- **`join pizzas p on od.pizza_id = p.pizza_id`**:
  - This performs an inner join between the `order_details` table (`od`) and the `pizzas` table (`p`). The join condition is `od.pizza_id = p.pizza_id`, matching each order detail with the corresponding pizza.
- **`join pizza_types pt on p.pizza_type_id = pt.pizza_type_id`**:
  - This performs another inner join, this time between the `pizzas` table (`p`) and the `pizza_types` table (`pt`). The join condition is `p.pizza_type_id = pt.pizza_type_id`, matching each pizza with its type.

## 3. GROUP BY Clause

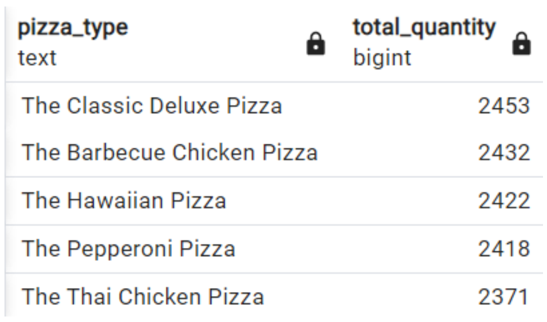
- **`group by pt.name`**:
  - This groups the result by the `name` of the pizza type (`pt.name`). It means the aggregate functions (in this case, `sum(quantity)`) are calculated for each unique pizza type.

## 4. ORDER BY Clause

- **`order by total_quantity desc`**:
  - This orders the results by the `total_quantity` in descending order. The most ordered pizza types (those with the highest total quantity) come first.

## 5. LIMIT Clause---`limit 5`:

  - This limits the result to the top 5 rows. Essentially, it retrieves the top 5 pizza types with the highest total quantity ordered.



| pizza_type<br>text | total_quantity<br>bigint |
|---|---|
| The Classic Deluxe Pizza | 2453 |
| The Barbecue Chicken Pizza | 2432 |
| The Hawaiian Pizza | 2422 |
| The Pepperoni Pizza | 2418 |
| The Thai Chicken Pizza | 2371 |

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

```
select pt.category as category,sum(quantity) as total_order_quantity
from order_details od
join pizzas p on od.pizza_id = p.pizza_id
join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
group by pt.category
order by total_order_quantity desc;
```

**1. SELECT Clause**

- **`pt.category as category`**:
  - This selects the `category` column from the `pizza_types` table and aliases it as `category` for clarity.
- **`sum(quantity) as total_order_quantity`**:
  - This calculates the total quantity ordered for each category by summing the `quantity` column from the `order_details` table and aliases it as `total_order_quantity`.

**3. FROM Clause**

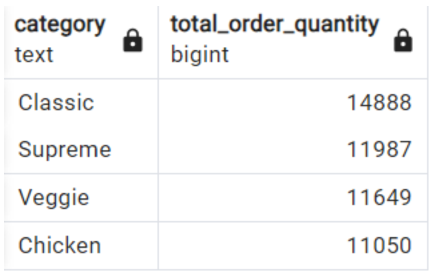
- **`from order_details od`**:
  1. This specifies the `order_details` table, aliased as `od`.
- **`join pizzas p on od.pizza_id = p.pizza_id`**:
  1. This performs an inner join between the `order_details` table (`od`) and the `pizzas` table (`p`). The join condition is `od.pizza_id = p.pizza_id`, matching each order detail with the corresponding pizza.
- **`join pizza_types pt on p.pizza_type_id = pt.pizza_type_id`**:
  1. This performs another inner join, this time between the `pizzas` table (`p`) and the `pizza_types` table (`pt`). The join condition is `p.pizza_type_id = pt.pizza_type_id`, matching each pizza with its type.

**3. GROUP BY Clause**

- **`group by pt.category`**:
  - This groups the result by the `category` of the pizza type (`pt.category`). It means the aggregate functions (in this case, `sum(quantity)`) are calculated for each unique pizza category.

**4. ORDER BY Clause**

- **`order by total_order_quantity desc`**:
  - This orders the results by the `total_order_quantity` in descending order. The pizza categories with the highest total quantities come first.



| category<br>text | total_order_quantity<br>bigint |
|---|---|
| Classic | 14888 |
| Supreme | 11987 |
| Veggie | 11649 |
| Chicken | 11050 |

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

```
select extract (hour from order_time) as hour,
        count(order_id) AS orders_count from orders
group by extract (hour from order_time)
order by hour;
```

**1. SELECT Clause**

- **`extract (hour from order_time) as hour`**:
  - The `extract` function is used to extract the hour component from the `order_time` column. This gives the hour during which the order was placed (from 0 to 23).
  - This extracted hour is aliased as `hour` for clarity in the results.
- **`count(order_id) as orders_count`**:
  - The `count` function counts the number of `order_id` entries for each hour, which gives the number of orders placed during each hour.
  - This count is aliased as `orders_count` to indicate that it represents the number of orders.

**2. FROM Clause**

- **`from orders`**:
  - This specifies that the data should be selected from the `orders` table.

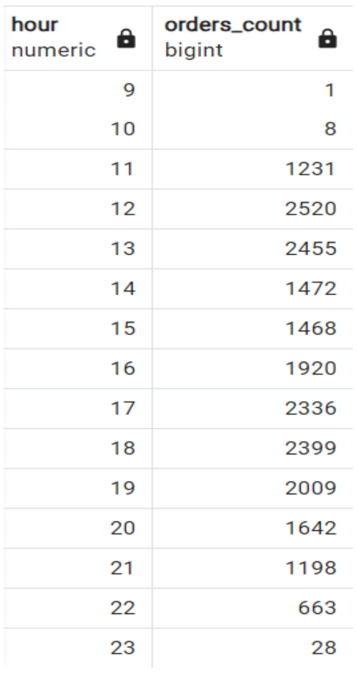
**3. GROUP BY Clause**

- **`group by extract (hour from order_time)`**:
  - This groups the results by the hour extracted from the `order_time` column. It ensures that the count of orders is calculated for each unique hour value.

**4. ORDER BY Clause**

**`order by hour`**:

- This orders the grouped results by the `hour` in ascending order. The result will display the count of orders for each hour in chronological order from 0 to 23.



| hour<br>numeric | orders_count<br>bigint |
|---|---|
| 9 | 1 |
| 10 | 8 |
| 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 |

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

```
select pt.category as category, sum(od.quantity) as total_quantity_ordered
from order_details od
join pizzas p on od.pizza_id = p.pizza_id
join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
group by pt.category
```

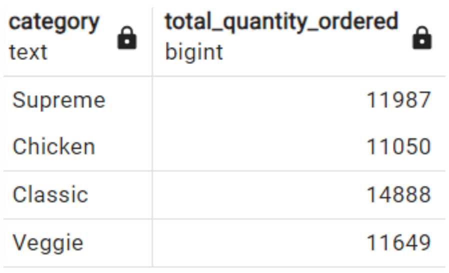
- The `order_details` table (`od`) is joined with the `pizzas` table (`p`) using `pizza_id` to link each order detail with the corresponding pizza.
- The `pizzas` table (`p`) is then joined with the `pizza_types` table (`pt`) using `pizza_type_id` to link each pizza with its type.

- Group **by Category**:

  - The results are grouped by `pt.category`, so each unique pizza category is considered.

- Calculate **Total Quantity**:

  - For each pizza category, the total quantity ordered (`sum(od.quantity)`) is calculated.



| category<br>text | total_quantity_ordered<br>bigint |
|---|---|
| Supreme | 11987 |
| Chicken | 11050 |
| Classic | 14888 |
| Veggie | 11649 |

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

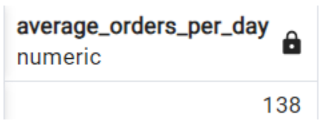
```
select
        round(avg(sum_orders_per_day)) as average_orders_per_day
from (
    select o.order_date as day, sum(od.quantity) as sum_orders_per_day
    from orders o
    join order_details od on o.order_id = od.order_id
    group by o.order_date
   )
```

## 1. Outer Query

- **`round(avg(sum_orders_per_day)) as average_orders_per_day`**:
    - `avg(sum_orders_per_day)`: This calculates the average of the daily total orders calculated in the subquery.
    - `round(...)`: This rounds the average to the nearest integer.
    - The result is aliased as `average_orders_per_day`.

## 2. Subquery

- **`select o.order_date as day, sum(od.quantity) as sum_orders_per_day`**:
    - `o.order_date as day`: This selects the `order_date` from the `orders` table and aliases it as `day` for clarity.
    - `sum(od.quantity) as sum_orders_per_day`: This calculates the total quantity of orders for each day by summing the `quantity` column from the `order_details` table. It is aliased as `sum_orders_per_day`.
- **`from orders o join order_details od on o.order_id = od.order_id`**:
    - This performs an inner join between the `orders` table (`o`) and the `order_details` table (`od`). The join condition is `o.order_id = od.order_id`, matching each order with its corresponding order details.
- **`group by o.order_date`**:
    - This groups the results by the `order_date`, so the sum of quantities is calculated for each unique day.



| average_orders_per_day<br>numeric |
|---|
| 138 |

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

```
select pt.name as pizza_type, sum (od.quantity * p.price) as total_revenue
from order_details od
join pizzas p on od.pizza_id = p.pizza_id
join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
group by pt.name
order by total_revenue desc
limit 3;
```

**1. SELECT Clause**

- 
  - This selects the `name` column from the `pizza_types` table and aliases it as `pizza_type` for clarity.
- **`sum (od.quantity * p.price) as total_revenue`**:
  - This calculates the total revenue for each pizza type by multiplying the `quantity` of each order (from `order_details`) by the `price` of the pizza (from `pizzas`), and then summing these values. It is aliased as `total_revenue`.

**2. FROM Clause**

- **`from order_details od`**:
  - This specifies the `order_details` table, aliased as `od`.
- **`join pizzas p on od.pizza_id = p.pizza_id`**:
  - This performs an inner join between the `order_details` table (`od`) and the `pizzas` table (`p`). The join condition is `od.pizza_id = p.pizza_id`, matching each order detail with the corresponding pizza.
- **`join pizza_types pt on p.pizza_type_id = pt.pizza_type_id`**:
  - This performs another inner join, this time between the `pizzas` table (`p`) and the `pizza_types` table (`pt`). The join condition is `p.pizza_type_id = pt.pizza_type_id`, matching each pizza with its type.

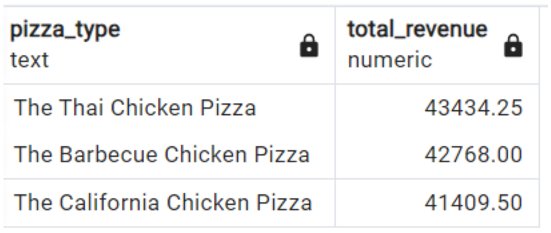
**3. GROUP BY Clause**

- **`group by pt.name`**:
  - This groups the result by the `name` of the pizza type (`pt.name`). It means the aggregate function (in this case, `sum (od.quantity * p.price)`) is calculated for each unique pizza type.

**4. ORDER BY Clause**

- **`order by total_revenue desc`**:
  - This orders the results by the `total_revenue` in descending order. The pizza types with the highest revenue come first.

**5. LIMIT Clause- limit 3**

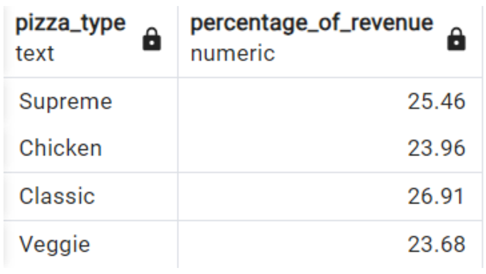


| pizza_type<br>text | total_revenue<br>numeric |
|---|---|
| The Thai Chicken Pizza | 43434.25 |
| The Barbecue Chicken Pizza | 42768.00 |
| The California Chicken Pizza | 41409.50 |

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

```
Select pt.category as pizza_type, round(sum(od.quantity * p.price)/(
Select sum (od.quantity * p.price) from order_details od
  join pizzas p on od.pizza_id = p.pizza_id)*100,2) as percentage_of_revenue
from order_details od
join pizzas p on od.pizza_id = p.pizza_id
join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
group by pt.category
```

1. **Revenue Calculation**:
   - `SUM (od.quantity * p.price)`: For each pizza type category, this calculates the total revenue by multiplying the quantity of each pizza sold (`od.quantity`) by its price (`p.price`) and summing these values.
2. **Total Revenue Calculation**:
   - The subquery computes the total revenue across all pizzas by summing the product of `quantity` and `price` for every order detail.
3. **Percentage Calculation**:
   - `SUM (od.quantity * p.price) / (SELECT SUM(...)) * 100`: The revenue for each pizza type category is divided by the total revenue (obtained from the subquery) and then multiplied by 100 to get the percentage.
   - `ROUND (..., 2)`: The result is rounded to two decimal places for better readability.



| pizza_type<br>text | percentage_of_revenue<br>numeric |
|---|---|
| Supreme | 25.46 |
| Chicken | 23.96 |
| Classic | 26.91 |
| Veggie | 23.68 |

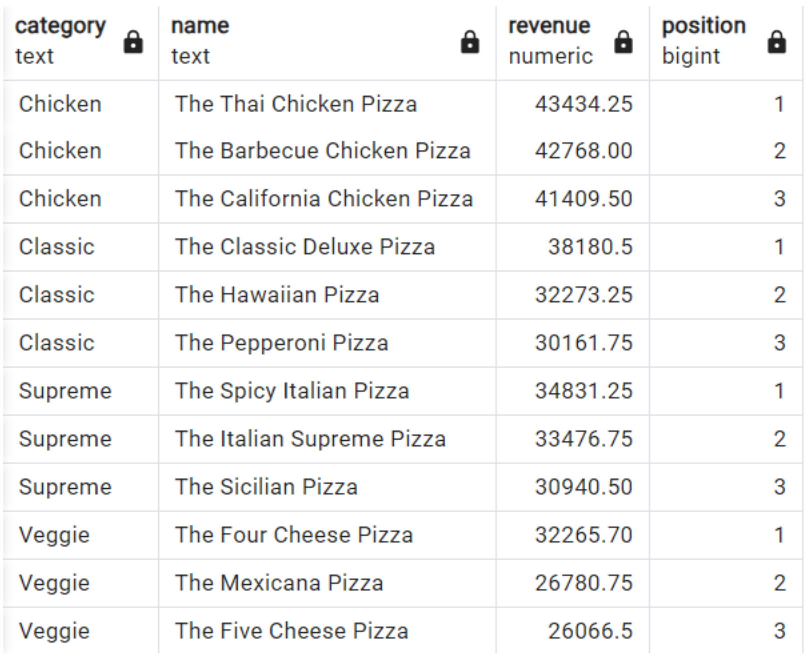
## 12. Analyze the cumulative revenue generated over time.

```
Select order_date, sum(revenue) over (order by order_date) as cumulative_revenue
from (select o.order_date as order_date, sum(p.price * od.quantity) as revenue
        from pizzas p
        join order_details od on p.pizza_id = od.pizza_id
        join orders o on od.order_id = o.order_id
        group by o.order_date )
order by order_date;
```

1. **Revenue Calculation for Each Order Date**:
   - The inner subquery calculates the total revenue for each order date by joining the relevant tables (`pizzas`, `order_details`, and `orders`), grouping by `order_date`, and summing the product of `price` and `quantity`.
2. **Cumulative Revenue Calculation**:
   - The outer query takes the revenue results from the subquery and calculates a running total of revenue ordered by `order_date` using the `SUM () OVER (ORDER BY order_date)` window function.

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

```
select category,name,revenue,position
from (
        select
                pt.category as category, pt.name as name,
                sum (od.quantity * p.price) as revenue,
                rank () over (partition by pt.category order by sum(od.quantity *
p.price) desc) as position
        from order_details od
        join pizzas p on od.pizza_id = p.pizza_id
        join pizza_types pt on p.pizza_type_id = pt.pizza_type_id
        group by pt.category, pt.name
        order by category, revenue desc
        )
where position in (1,2,3);
```



| category | name | revenue | position |
|---|---|---|---|
| text | text | numeric | bigint |
| Chicken | The Thai Chicken Pizza | 43434.25 | 1 |
| Chicken | The Barbecue Chicken Pizza | 42768.00 | 2 |
| Chicken | The California Chicken Pizza | 41409.50 | 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 | 30940.50 | 3 |
| Veggie | The Four Cheese Pizza | 32265.70 | 1 |
| Veggie | The Mexicana Pizza | 26780.75 | 2 |
| Veggie | The Five Cheese Pizza | 26066.5 | 3 |