Pizza Sales Analysis with SQL

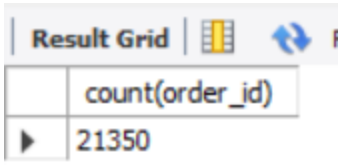
The Pizza Sales Analysis Report provides a comprehensive overview of the sales performance of different types of pizzas over a specified period. This report is designed to help business stakeholders, such as restaurant managers and marketing teams, understand which pizza categories and specific pizza types are the most popular and generate the highest revenue.

Purpose: To analyze the sales data of various pizza types to identify trends, top-selling items, and revenue contributions.
Data Sources: Sales data is aggregated from multiple tables including order_details, orders, pizzas, and pizza_types to provide a holistic view of pizza sales.
Time Period: The report covers sales data over a specific time frame, such as a month, quarter, or year, depending on the user's needs.

Q.1 Retrieve the total number of orders placed.
Ans:

```
select count(order_id) from orders;
```

Output:



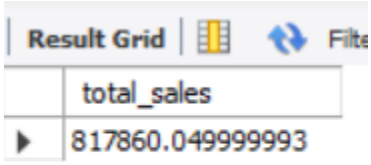
Breakdown of the Query:

- **SELECT**: This keyword is used to specify the data that you want to retrieve from the database.
- **COUNT(order_id)**: This function counts the number of non-NULL values in the order_id column. Since order_id is typically a primary key and hence unique and non-NULL in most tables, this count will effectively represent the total number of rows in the orders table.
- **FROM orders**: This specifies the table from which to retrieve the data, in this case, the orders table.

Q.2 Calculate the total revenue generated from pizza sales.
Ans:

```
SELECT
   SUM(order_details.quantity * pizzas.price) AS total_sales
FROM
   order_details
      JOIN
   pizzas ON pizzas.pizza_id = order_details.pizza_id
```

Output:



Breakdown of the Query:

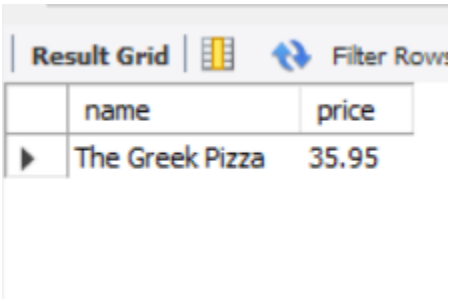
- **SELECT**: Specifies the data you want to retrieve from the database.
- **SUM(order_details.quantity * pizzas.price) AS total_sales**:
    - **SUM**: This is an aggregate function that calculates the total sum of the values generated by the expression within it.
    - **order_details.quantity * pizzas.price**: This expression multiplies the quantity of each pizza ordered (order_details.quantity) by the price of that

pizza (pizzas.price). This gives the total sales value for each order line (i.e., the revenue generated from each type of pizza in each order).
    - **AS total_sales**: This clause assigns an alias total_sales to the result of the SUM function, naming the column in the output that contains the total sales revenue.
- **FROM order_details**: Specifies the order_details table as the main table from which to retrieve data. This table likely contains details about each order, including the pizza_id and the quantity of each pizza ordered.
- **JOIN pizzas ON pizzas.pizza_id = order_details.pizza_id**:
    - **JOIN**: This keyword is used to combine rows from two or more tables based on a related column between them.
    - **pizzas**: This is the table that contains the price information for each pizza (pizza_id and price).
    - **ON pizzas.pizza_id = order_details.pizza_id**: This clause specifies the condition for the join, indicating that rows from the order_details table should be matched with rows from the pizzas table where the pizza_id is the same in both tables.

Q.3 Identify the highest-priced pizza.
Ans

```
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;
```

Output:



Breakdown of the Query:

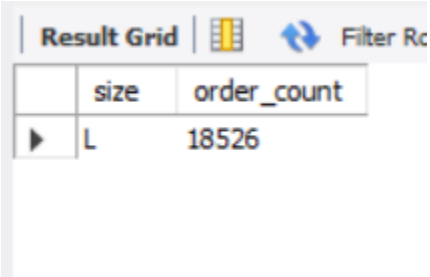
- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **pizza_types.name, pizzas.price**:
    - **pizza_types.name**: Retrieves the name of the pizza type from the pizza_types table.
    - **pizzas.price**: Retrieves the price of the pizza from the pizzas table.

- **FROM pizza_types**: Specifies the pizza_types table as the primary table from which to retrieve data.
- **JOIN pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id**:
    - **JOIN**: This keyword is used to combine rows from two tables based on a related column between them.
    - **pizzas**: This table contains the price information for each pizza.
    - **ON pizza_types.pizza_type_id = pizzas.pizza_type_id**: This clause specifies the condition for the join. It means that each row in the pizza_types table is matched with rows in the pizzas table where the pizza_type_id matches. This allows you to associate each pizza with its corresponding type.
- **ORDER BY pizzas.price DESC**:
    - **ORDER BY**: This clause sorts the result set.
    - **pizzas.price DESC**: Sorts the rows in descending order based on the price column from the pizzas table. This means the most expensive pizza will be listed first.
- **LIMIT 1**:
    - **LIMIT**: Restricts the number of rows returned by the query.
    - **1**: Means that only the first row of the result set will be returned, which corresponds to the pizza with the highest price due to the descending sort order applied earlier.

Q.4 Identify the most common pizza size ordered.
Ans

```
SELECT

   pizzas.size,

   COUNT(order_details.order_details_id) AS order_count

FROM

   pizzas

      JOIN

   order_details ON pizzas.pizza_id = order_details.pizza_id

GROUP BY pizzas.size

ORDER BY order_count DESC

LIMIT 1;
```

Output:



	size	order_count
▶	L	18526

Breakdown of the Query:

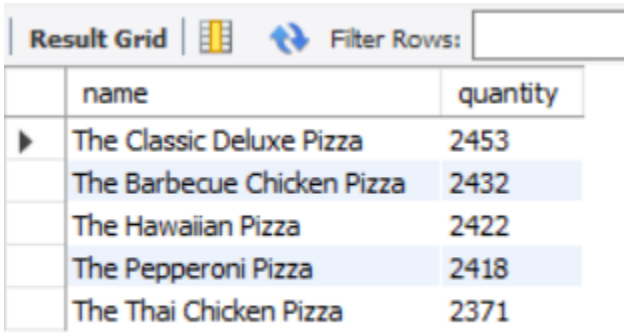
- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **pizzas.size**: Retrieves the size of the pizza from the pizzas table. The size column might represent various pizza sizes, such as "Small," "Medium," "Large," etc.
- **COUNT(order_details.order_details_id) AS order_count**:
    - **COUNT(order_details.order_details_id)**: This aggregate function counts the number of non-NULL entries in the order_details_id column of the order_details table for each group of pizza sizes. Essentially, it counts how many times each size of pizza was ordered.
    - **AS order_count**: This clause gives the result of the COUNT function an alias (order_count), which represents the total number of orders for each pizza size.
- **FROM pizzas**: Specifies the pizzas table as the main table to retrieve data from.
- **JOIN order_details ON pizzas.pizza_id = order_details.pizza_id**:
    - **JOIN**: Combines rows from the pizzas and order_details tables based on a related column between them.
    - **order_details**: This table contains information about each order, such as order_details_id and pizza_id.
    - **ON pizzas.pizza_id = order_details.pizza_id**: This clause specifies the condition for the join, matching each row from the pizzas table with rows in the order_details table where the pizza_id is the same. This allows you to link each pizza size with its corresponding order details.
- **GROUP BY pizzas.size**:
    - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by pizzas.size, which means that all rows with the same pizza size are grouped together.
    - By grouping by pizzas.size, the COUNT function will calculate the number of orders for each size.
- **ORDER BY order_count DESC**:
    - **ORDER BY**: This clause sorts the result set.
    - **order_count DESC**: Sorts the rows in descending order based on the order_count column, meaning the pizza size with the highest number of orders will appear first.
- **LIMIT 1**:

- **LIMIT**: Restricts the number of rows returned by the query.
- **1**: Ensures that only the first row of the result set is returned, which corresponds to the pizza size with the highest order count.

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

```
SELECT
   pizza_types.name, SUM(order_details.quantity) AS quantity
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.name
ORDER BY quantity DESC
LIMIT 5;
```

Output:



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

Breakdown of the Query:

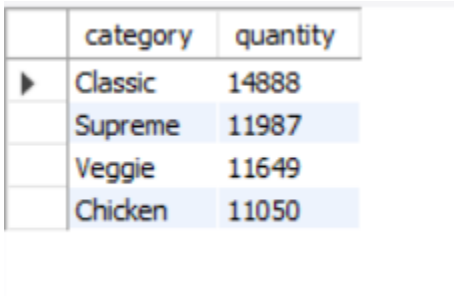
- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **pizza_types.name**: Retrieves the name of the pizza type from the pizza_types table. This column represents the specific type of pizza (e.g., "Pepperoni," "Margherita").
- **SUM(order_details.quantity) AS quantity**:
  - **SUM(order_details.quantity)**: This aggregate function calculates the total sum of the quantity column from the order_details table. This represents the total number of each pizza type that has been ordered across all orders.

    - **AS quantity**: This clause gives the result of the SUM function an alias (quantity), naming the column in the output that contains the total quantity ordered for each pizza type.
- **FROM pizza_types**: Specifies the pizza_types table as the primary table to retrieve data from.
- **JOIN pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id**:
    - **JOIN**: Combines rows from the pizza_types and pizzas tables based on a related column between them.
    - **pizzas**: This table contains information about each pizza, including its pizza_type_id, which links it to the pizza_types table.
    - **ON pizza_types.pizza_type_id = pizzas.pizza_type_id**: This clause specifies the condition for the join, matching each row from the pizza_types table with rows in the pizzas table where the pizza_type_id is the same. This allows you to associate each pizza with its corresponding type.
- **JOIN order_details ON order_details.pizza_id = pizzas.pizza_id**:
    - **JOIN**: Combines rows from the pizzas and order_details tables based on a related column between them.
    - **order_details**: This table contains details about each order, including the pizza_id and the quantity of each pizza ordered.
    - **ON order_details.pizza_id = pizzas.pizza_id**: This clause specifies the condition for the join, matching each row from the order_details table with rows in the pizzas table where the pizza_id is the same. This allows you to link each pizza with its corresponding order details.
- **GROUP BY pizza_types.name**:
    - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by pizza_types.name, meaning that all rows with the same pizza type name are grouped together.
    - By grouping by pizza_types.name, the SUM function will calculate the total quantity ordered for each pizza type.
- **ORDER BY quantity DESC**:
    - **ORDER BY**: This clause sorts the result set.
    - **quantity DESC**: Sorts the rows in descending order based on the quantity column. This means the pizza type with the highest total quantity ordered will appear first.
- **LIMIT 5**:
    - **LIMIT**: Restricts the number of rows returned by the query.
    - **5**: Ensures that only the first 5 rows of the result set are returned, which correspond to the top 5 pizza types with the highest quantities ordered.

Q.6 Join the necessary tables to find the total quantity of each pizza category ordered.
Ans:
SELECT

```
    pizza_types.category,
    SUM(order_details.quantity) AS quantity
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
ORDER BY quantity DESC;
```

Output:



category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

Breakdown of the Query:

- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **pizza_types.category**: Retrieves the category of the pizza from the pizza_types table. The category column represents different categories of pizza (e.g., "Vegetarian," "Meat," "Specialty").
- **SUM(order_details.quantity) AS quantity**:
    - **SUM(order_details.quantity)**: This aggregate function calculates the total sum of the quantity column from the order_details table for each group of pizza categories. This represents the total number of pizzas ordered in each category.
    - **AS quantity**: This clause assigns an alias (quantity) to the result of the SUM function, naming the column in the output that contains the total quantity ordered for each pizza category.
- **FROM pizza_types**: Specifies the pizza_types table as the primary table to retrieve data from.
- **JOIN pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id**:
    - **JOIN**: Combines rows from the pizza_types and pizzas tables based on a related column between them.
    - **pizzas**: This table contains information about each pizza, including its pizza_type_id, which links it to the pizza_types table.
    - **ON pizza_types.pizza_type_id = pizzas.pizza_type_id**: This clause specifies the condition for the join, matching each row from the pizza_types table with rows in the pizzas table where the pizza_type_id is

the same. This allows you to associate each pizza with its corresponding type.

- **JOIN order_details ON order_details.pizza_id = pizzas.pizza_id**:
  - **JOIN**: Combines rows from the pizzas and order_details tables based on a related column between them.
  - **order_details**: This table contains details about each order, including the pizza_id and the quantity of each pizza ordered.
  - **ON order_details.pizza_id = pizzas.pizza_id**: This clause specifies the condition for the join, matching each row from the order_details table with rows in the pizzas table where the pizza_id is the same. This allows you to link each pizza with its corresponding order details.
- **GROUP BY pizza_types.category**:
  - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by pizza_types.category, meaning that all rows with the same pizza category are grouped together.
  - By grouping by pizza_types.category, the SUM function will calculate the total quantity ordered for each category.
- **ORDER BY quantity DESC**:
  - **ORDER BY**: This clause sorts the result set.
  - **quantity DESC**: Sorts the rows in descending order based on the quantity column. This means the pizza category with the highest total quantity ordered will appear first.

Q. 7 Determine the distribution of orders by hour of the day.
Ans:

```
SELECT
   HOUR(order_time) AS hour, COUNT(order_id)
FROM
   orders
GROUP BY HOUR(order_time)
```

Output:

hour	COUNT(order_id)
9	1
10	8
11	1231
12	2520
13	2455
14	1472
15	1468
16	1920
17	2336

Breakdown of the Query:

- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **HOUR(order_time) AS hour**:
    - **HOUR(order_time)**: This function extracts the hour part from the order_time column in the orders table. The order_time column presumably contains the time at which each order was placed.
    - **AS hour**: This clause gives an alias (hour) to the result of the HOUR function, naming the column in the output that contains the hour extracted from the order_time.
- **COUNT(order_id)**:
    - **COUNT(order_id)**: This aggregate function counts the number of non-NULL entries in the order_id column for each group of hours. This represents the total number of orders placed during each hour of the day.
- **FROM orders**: Specifies the orders table as the source of data to retrieve.
- **GROUP BY HOUR(order_time)**:
    - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by the hour extracted from the order_time column. This means that all rows with the same hour value are grouped together.
    - **HOUR(order_time)**: The GROUP BY clause uses the hour value extracted from order_time to group the data. This allows the COUNT function to calculate the number of orders for each specific hour.

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

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

Output:

	category	COUNT(name)
▶	Chicken	6
	Classic	8
	Supreme	9
	Veggie	9

Breakdown of the Query:

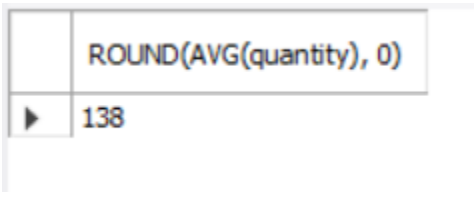
- **SELECT**: Specifies the columns that you want to retrieve from the database.

- **category**: Retrieves the category column from the pizza_types table. This column represents different categories of pizza (e.g., "Vegetarian," "Meat," "Specialty").
- **COUNT(name)**:
    - **COUNT(name)**: This aggregate function counts the number of non-NULL entries in the name column for each group of categories. This represents the total number of distinct pizza types within each category.
    - Since the COUNT function is used on the name column, it counts how many pizza names exist for each category.
- **FROM pizza_types**: Specifies the pizza_types table as the source of data to retrieve.
- **GROUP BY category**:
    - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by the category column, which means that all rows with the same category value are grouped together.
    - By grouping by category, the COUNT function will calculate the number of pizza names (or types) within each category.

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

```
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:



	ROUND(AVG(quantity), 0)
▶	138

Breakdown of the Query:

- **Outer Query**:
    - **SELECT ROUND(AVG(quantity), 0)**:
        - **AVG(quantity)**: This aggregate function calculates the average of the quantity column from the subquery results, which represents the daily total number of pizzas ordered.

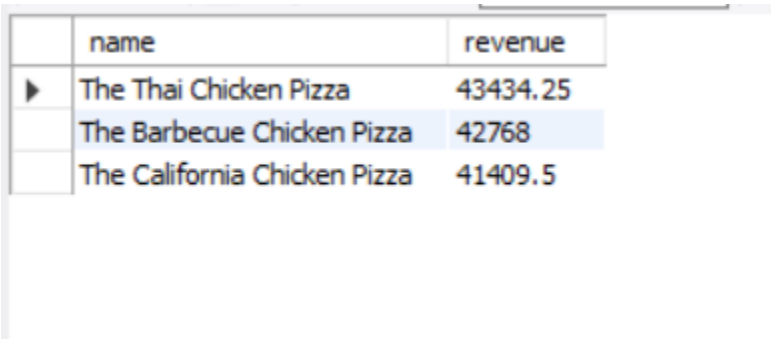
        - **ROUND(..., 0)**: The ROUND function rounds the average to the nearest whole number (0 decimal places). This is often done to simplify the data for reporting or analysis purposes.
- **FROM (...) AS order_quantity**:
    - This specifies that the outer query is selecting data from a subquery, aliased as order_quantity. The subquery provides the detailed daily totals of pizzas ordered, which are then averaged by the outer query.
- **Subquery**:
    - **SELECT orders.order_date, SUM(order_details.quantity) AS quantity**:
        - **orders.order_date**: Selects the date of each order from the orders table. This is used to group orders by day.
        - **SUM(order_details.quantity) AS quantity**: The SUM function calculates the total number of pizzas ordered (quantity) for each day. This total represents all the pizzas ordered on a given date.
    - **FROM orders**: Specifies the orders table as the source for the main query data.
    - **JOIN order_details ON orders.order_id = order_details.order_id**:
        - **JOIN**: Combines rows from the orders and order_details tables based on a related column between them.
        - **order_details**: Contains the details of each order, including the order_id and the quantity of each pizza ordered.
        - **ON orders.order_id = order_details.order_id**: This clause specifies the condition for the join, matching each order in the orders table with its corresponding details in the order_details table using the order_id.
    - **GROUP BY orders.order_date**:
        - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by the order_date, meaning all orders on the same date are grouped together.
        - This allows the SUM function to compute the total quantity of pizzas ordered per day.

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

```
SELECT
   pizza_types.name,
   SUM(order_details.quantity * pizzas.price) 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.name
```

```
ORDER BY revenue DESC
LIMIT 3;
```

Output:



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

Breakdown of the Query:

- **SELECT**: Specifies the columns that you want to retrieve from the database.
- **pizza_types.name**:
  - Retrieves the name column from the pizza_types table, which represents the name of each type of pizza.
- **SUM(order_details.quantity * pizzas.price) AS revenue**:
  - **order_details.quantity**: Represents the number of units of a particular pizza ordered in each entry.
  - **pizzas.price**: The price of each pizza from the pizzas table.
  - **order_details.quantity * pizzas.price**: Calculates the revenue generated by each order line by multiplying the quantity of pizzas ordered by their price.
  - **SUM(...)**: The aggregate function SUM calculates the total revenue generated for each pizza type across all orders.
  - **AS revenue**: Assigns the alias revenue to the calculated total revenue for readability in the output.
- **FROM pizza_types**: Specifies the pizza_types table as the source for the initial dataset.
- **JOIN pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id**:
  - **JOIN**: Combines rows from the pizza_types and pizzas tables based on a related column between them.
  - **ON pizzas.pizza_type_id = pizza_types.pizza_type_id**: This join condition matches each row from the pizza_types table with corresponding rows in the pizzas table using the pizza_type_id. This relates each pizza with its specific type.
- **JOIN order_details ON order_details.pizza_id = pizzas.pizza_id**:
  - **JOIN**: Combines rows from the pizzas and order_details tables based on a related column between them.
  - **ON order_details.pizza_id = pizzas.pizza_id**: This join condition matches each row from the order_details table with corresponding rows in

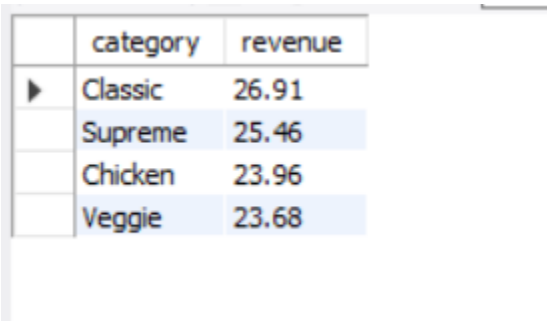
the pizzas table using the pizza_id. This allows linking each pizza order detail to its respective pizza.

- **GROUP BY pizza_types.name**:
  - **GROUP BY**: Groups the result set by the specified column(s). Here, it groups the data by pizza_types.name, which means that all rows corresponding to the same pizza type are grouped together.
  - This allows the SUM function to calculate the total revenue for each specific pizza type.
- **ORDER BY revenue DESC**:
  - **ORDER BY**: This clause sorts the result set.
  - **revenue DESC**: Sorts the rows in descending order based on the revenue column, meaning the pizza types with the highest revenue appear first.
- **LIMIT 3**:
  - **LIMIT**: Restricts the number of rows returned by the query.
  - **3**: Ensures that only the top three rows (pizza types with the highest revenue) are returned.

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

```
select pizza_types.category,
round((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) )*100,2) 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 order by revenue desc;
```

Output:



category	revenue
Classic	26.91
Supreme	25.46
Chicken	23.96
Veggie	23.68

Breakdown of the Query:

- **SELECT**: Specifies the columns that you want to retrieve from the database.

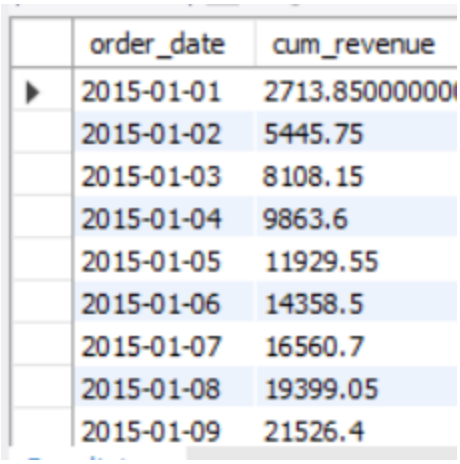
- **pizza_types.category**:
    - Retrieves the category column from the pizza_types table, which represents different categories of pizzas (e.g., "Vegetarian," "Meat," "Specialty").
- **ROUND((SUM(order_details.quantity * pizzas.price) / (...) * 100, 2) AS revenue_percentage**:
    - **SUM(order_details.quantity * pizzas.price)**:
        - **order_details.quantity**: Represents the number of units of a particular pizza ordered in each entry.
        - **pizzas.price**: The price of each pizza from the pizzas table.
        - **order_details.quantity * pizzas.price**: Calculates the revenue generated by each order line by multiplying the quantity of pizzas ordered by their price.
        - **SUM(...)**: This aggregate function calculates the total revenue generated for each pizza category.
    - **(SELECT ROUND(SUM(order_details.quantity * pizzas.price), 2) AS total_sales FROM ...)**:
        - This subquery calculates the total revenue from all pizza orders by summing the product of quantity and price across all orders.
        - **ROUND(SUM(...), 2)**: Rounds the total sales to two decimal places to provide a precise total revenue value.
    - **SUM(...) / (SELECT ...)**: Divides the total revenue for each category by the total revenue for all categories to calculate the revenue share of each category.
    - *** 100**: Converts the revenue share into a percentage.
    - **ROUND(..., 2)**: Rounds the percentage to two decimal places for a cleaner output.
    - **AS revenue_percentage**: Gives an alias to the calculated revenue percentage column for readability.
- **FROM pizza_types**: Specifies the pizza_types table as the source for the initial dataset.
- **JOIN pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id**:
    - **JOIN**: Combines rows from the pizza_types and pizzas tables based on a related column between them.
    - **ON pizzas.pizza_type_id = pizza_types.pizza_type_id**: This join condition matches each row from the pizza_types table with corresponding rows in the pizzas table using the pizza_type_id.
- **JOIN order_details ON order_details.pizza_id = pizzas.pizza_id**:
    - **JOIN**: Combines rows from the pizzas and order_details tables based on a related column between them.
    - **ON order_details.pizza_id = pizzas.pizza_id**: This join condition matches each row from the order_details table with corresponding rows in the pizzas table using the pizza_id.
- **GROUP BY pizza_types.category**:

    - **GROUP BY**: Groups the result set by one or more columns. Here, it groups the data by pizza_types.category, meaning all rows corresponding to the same pizza category are grouped together.
    - This allows the SUM function to calculate the total revenue for each specific pizza category.
- **ORDER BY revenue_percentage DESC**:
    - **ORDER BY**: This clause sorts the result set.
    - **revenue_percentage DESC**: Sorts the rows in descending order based on the revenue_percentage column, meaning the pizza categories with the highest revenue percentage appear first.

Q. 12 Analyze the cumulative revenue generated over time.
Ans:

```
SELECT
   s1.order_date,
   (SELECT SUM(s2.revenue)
    FROM
    (
       SELECT
          o.order_date,
          SUM(od.quantity * p.price) AS revenue
       FROM
          order_details od
       JOIN pizzas p ON od.pizza_id = p.pizza_id
       JOIN orders o ON o.order_id = od.order_id
       GROUP BY o.order_date
    ) s2
    WHERE s2.order_date <= s1.order_date
   ) AS cum_revenue
FROM
(
   SELECT
      o.order_date,
      SUM(od.quantity * p.price) AS revenue
   FROM
      order_details od
   JOIN pizzas p ON od.pizza_id = p.pizza_id
   JOIN orders o ON o.order_id = od.order_id
   GROUP BY o.order_date
) s1;
```

Output:



order_date	cum_revenue
2015-01-01	2713.85000000
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

Breakdown of the Query:

- **Outer Query**:
  - **SELECT s1.order_date,**:
    - This selects the order_date from the outer subquery (s1). This date represents each unique day on which orders were placed.
  - **(SELECT SUM(s2.revenue) ...) AS cum_revenue**:
    - This is a correlated subquery that calculates the cumulative revenue up to and including each order_date.
- **Inner Subquery (s2)**:
  - **SELECT o.order_date, SUM(od.quantity * p.price) AS revenue**:
    - **o.order_date**: Selects the order date from the orders table.
    - **SUM(od.quantity * p.price) AS revenue**: Calculates the total revenue for each order date by multiplying the quantity of each pizza ordered (od.quantity) by the price of the pizza (p.price). The SUM function adds up the revenues for all pizzas ordered on that date.
  - **FROM order_details od**: Indicates the order_details table as the source of detailed order information.
  - **JOIN pizzas p ON od.pizza_id = p.pizza_id**:
    - **JOIN**: This joins the order_details table with the pizzas table based on the pizza_id. This allows access to pizza prices to calculate revenue.
  - **JOIN orders o ON o.order_id = od.order_id**:
    - **JOIN**: This joins the orders table with the order_details table based on order_id. This is needed to access the order_date for grouping.
  - **GROUP BY o.order_date**:
    - **GROUP BY**: Groups the results by each unique order_date to compute the total revenue for each date.
- **Correlated Subquery Condition (WHERE s2.order_date <= s1.order_date)**:
  - This condition is key to the cumulative calculation. It specifies that the subquery should sum the revenues for all dates up to and including the

order_date from the outer query (s1.order_date). This effectively calculates the running total (cumulative revenue) up to each date.

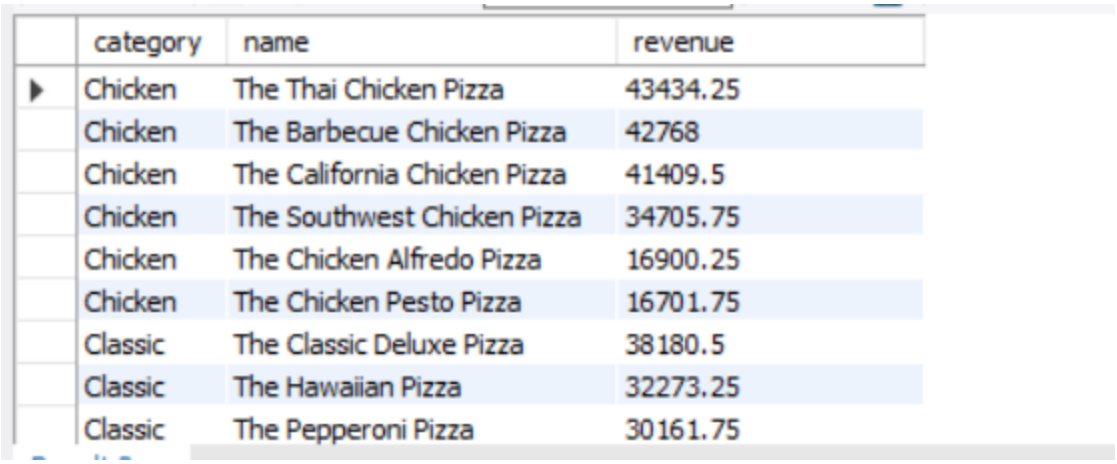
- **Main Subquery (s1)**:
  - **SELECT o.order_date, SUM(od.quantity * p.price) AS revenue**:
    - Similar to the s2 subquery, this calculates the total revenue per order date.
  - **FROM order_details od ... GROUP BY o.order_date**:
    - As explained above, this calculates the daily total revenue.

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

Ans:

```
SELECT
   pt.category,
   pt.name,
   SUM(od.quantity * p.price) AS revenue
FROM
   pizza_types pt
JOIN pizzas p ON pt.pizza_type_id = p.pizza_type_id
JOIN order_details od ON od.pizza_id = p.pizza_id
GROUP BY pt.category, pt.name
HAVING SUM(od.quantity * p.price) IN (
   SELECT
      revenue
   FROM (
      SELECT
         pt2.category,
         pt2.name,
         SUM(od2.quantity * p2.price) AS revenue
      FROM
         pizza_types pt2
      JOIN pizzas p2 ON pt2.pizza_type_id = p2.pizza_type_id
      JOIN order_details od2 ON od2.pizza_id = p2.pizza_id
      GROUP BY pt2.category, pt2.name
      ORDER BY revenue DESC
      LIMIT 3
   ) AS top_revenues
)
ORDER BY pt.category, revenue DESC;
```

Output:



category	name	revenue
Chicken	The Thai Chicken Pizza	43434.25
Chicken	The Barbecue Chicken Pizza	42768
Chicken	The California Chicken Pizza	41409.5
Chicken	The Southwest Chicken Pizza	34705.75
Chicken	The Chicken Alfredo Pizza	16900.25
Chicken	The Chicken Pesto Pizza	16701.75
Classic	The Classic Deluxe Pizza	38180.5
Classic	The Hawaiian Pizza	32273.25
Classic	The Pepperoni Pizza	30161.75

Breakdown of the Query:

- **SELECT pt.category, pt.name, SUM(od.quantity * p.price) AS revenue**:
  - **pt.category**: Selects the category of the pizza from the pizza_types table.
  - **pt.name**: Selects the name of the pizza type from the pizza_types table.
  - **SUM(od.quantity * p.price) AS revenue**: Calculates the total revenue for each pizza type by multiplying the quantity ordered (od.quantity) by the price of the pizza (p.price) and summing these values.
- **FROM pizza_types pt**:
  - **pt** is an alias for the pizza_types table, which contains information about different pizza types.
- **JOIN pizzas p ON pt.pizza_type_id = p.pizza_type_id**:
  - Joins the pizzas table with the pizza_types table based on the pizza_type_id to link each pizza with its corresponding type.
- **JOIN order_details od ON od.pizza_id = p.pizza_id**:
  - Joins the order_details table with the pizzas table based on pizza_id to access the quantities ordered for each pizza.
- **GROUP BY pt.category, pt.name**:
  - Groups the results by both category and name to calculate the total revenue for each pizza type.
- **HAVING SUM(od.quantity * p.price) IN (...)**:
  - The HAVING clause filters the grouped results to include only those rows where the total revenue matches any of the top 3 revenues determined by the subquery.
- **Subquery (IN clause)**:
  - **SELECT revenue FROM (...) AS top_revenues**:
    - Retrieves the revenue values calculated in the innermost subquery.
  - **Innermost Subquery**:
    - **SELECT pt2.category, pt2.name, SUM(od2.quantity * p2.price) AS revenue**:
      - Similar to the outer query, but with different aliases (pt2, p2, od2). This calculates the revenue for each pizza type.

- **GROUP BY pt2.category, pt2.name**:
  - Groups by pizza type to aggregate the revenue for each.
- **ORDER BY revenue DESC**:
  - Orders the result by revenue in descending order to identify the top revenue-generating pizzas.
- **LIMIT 3**:
  - Limits the result set to the top 3 pizzas based on revenue.

Summary

This analysis focuses on exploring various dimensions of sales and customer data. By identifying trends in customer numbers, revenue generation by pizza types, and peak times, it provides a detailed view of operational performance. This information helps owners formulate targeted questions and explore specific areas for improvement.