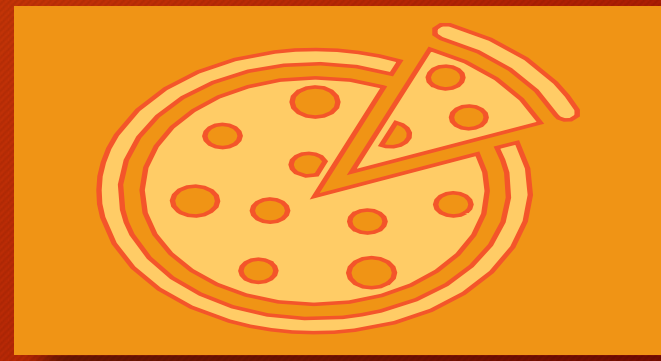
Pizza Sales Analysis Using MySQL



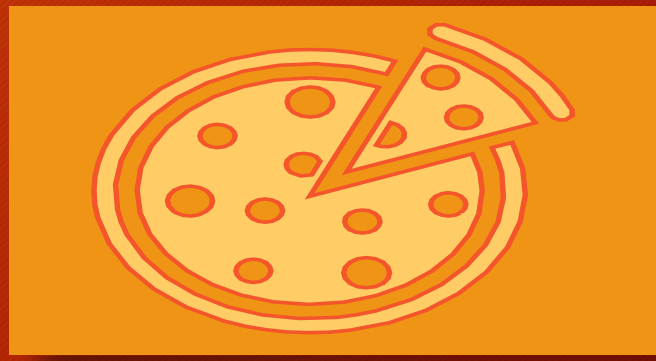
Leveraging MySQL for Pizza Sales Data Insights

By: Sundram Kumar

Business Objective

The business objective of this project is to analyse the sales data from the past year to identify key trends, patterns, and customer preferences. The goal is to use these insights to make data-driven decisions that will help the pizza store optimize its operations, improve marketing strategies, and enhance overall sales for the upcoming year.

Data Collection and Acquisition



Data Source and Structure

I got this dataset from Kaggle, the data contains the 4 table namely; order_details, orders, pizza_types, pizzas.

- **order_details** contains columns namely; order_details_id, order_id, pizza_id, and quantity.
- **orders** contains columns namely; order_id, order_date, and order_time.
- **pizza_types** contains columns namely; pizza_type_id, name, category, and ingredients.
- **pizzas** contains columns namely; pizza_id, pizza_type_id, size, and price.

Data Dictionary

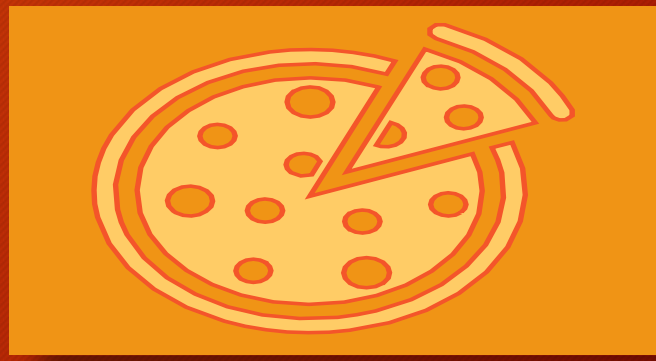
The data dictionary provides detailed insights into the meaning and structure of the dataset. This pizza sales dataset consists of 12 key features:

- order_id: Unique identifier for each order placed by a table
- order_details_id: Unique identifier for each pizza placed within each order (pizzas of the same type and size are kept in the same row, and the quantity increases)
- pizza_id: Unique key identifier that ties the pizza ordered to its details, like size and price
- quantity: Quantity ordered for each pizza of the same type and size
- order_date: Date the order was placed (entered into the system prior to cooking & serving)

Data Dictionary

- order_time: Time the order was placed (entered into the system prior to cooking & serving)
- price: Price of the pizza in USD
- pizza_size: Size of the pizza (Small, Medium, Large, X Large, or XX Large)
- pizza_category: Unique key identifier that ties the pizza ordered to its details, like size and price
- pizza_ingredients: ingredients used in the pizza as shown in the menu (they all include Mozzarella Cheese, even if not specified; and they all include Tomato Sauce, unless another sauce is specified)
- pizza_name: Name of the pizza as shown in the menu

Data Cleaning and Transformation



Data Cleaning

Check the first 10 rows to make sure it imported well.

For order_details Table:

```
1 •  SELECT * FROM order_details
2    LIMIT 10;
```

Output

order_details_id	order_id	pizza_id	quantity
1	1	hawaiian_m	1
2	2	classic_dlx_m	1
3	2	five_cheese_l	1
4	2	ital_supr_l	1
5	2	mexicana_m	1
6	2	thai_ckn_l	1
7	3	ital_supr_m	1
8	3	prsc_argla_l	1
9	4	ital_supr_m	1
10	5	ital_supr_m	1
NULL	NULL	NULL	NULL

Reviewing the first 10 rows of data without encountering any errors indicates successful importation and suggests that our data was imported accurately. This initial validation step confirms the integrity of the imported data and ensures that further analysis can proceed smoothly.

Data Cleaning

Check the first 10 rows to make sure it imported well.

For orders Table:

```
1 •  SELECT * FROM orders
2    LIMIT 10;
```

Output

	order_id	order_date	order_time
▸	1	2015-01-01	11:38:36
	2	2015-01-01	11:57:40
	3	2015-01-01	12:12:28
	4	2015-01-01	12:16:31
	5	2015-01-01	12:21:30
	6	2015-01-01	12:29:36
	7	2015-01-01	12:50:37
	8	2015-01-01	12:51:37
	9	2015-01-01	12:52:01
	10	2015-01-01	13:00:15
	NULL	NULL	NULL

Reviewing the first 10 rows of data without encountering any errors indicates successful importation and suggests that our data was imported accurately. This initial validation step confirms the integrity of the imported data and ensures that further analysis can proceed smoothly.

Data Cleaning

Check the first 10 rows to make sure it imported well.

For pizza_types Table:

```
1 • SELECT * FROM pizza_types
2   LIMIT 10;
```

Output

pizza_type_id	name	category	ingredients
bbq_ckn	The Barbecue Chicken Pi...	Chicken	Barbecued Chicken, Red ...
cali_ckn	The California Chicken Pi...	Chicken	Chicken, Artichoke, Spin...
ckn_alfredo	The Chicken Alfredo Pizza	Chicken	Chicken, Red Onions, Re...
ckn_pesto	The Chicken Pesto Pizza	Chicken	Chicken, Tomatoes, Red...
southw_ckn	The Southwest Chicken ...	Chicken	Chicken, Tomatoes, Red...
thai_ckn	The Thai Chicken Pizza	Chicken	Chicken, Pineapple, Tom...
big_meat	The Big Meat Pizza	Classic	Bacon, Pepperoni, Italia...
classic_dlx	The Classic Deluxe Pizza	Classic	Pepperoni, Mushrooms, ...
hawaiian	The Hawaiian Pizza	Classic	Sliced Ham, Pineapple, ...
ital_cpcllo	The Italian Capocollo Piz...	Classic	Capocollo, Red Peppers, ...

Reviewing the first 10 rows of data without encountering any errors indicates successful importation and suggests that our data was imported accurately. This initial validation step confirms the integrity of the imported data and ensures that further analysis can proceed smoothly.

Data Cleaning

Check the first 10 rows to make sure it imported well.

For pizzas Table:

```
1 •  SELECT * FROM pizzas
2    LIMIT 10;
```

Output

pizza_id	pizza_type_id	size	price
bbq_ckn_s	bbq_ckn	S	12.75
bbq_ckn_m	bbq_ckn	M	16.75
bbq_ckn_l	bbq_ckn	L	20.75
cali_ckn_s	cali_ckn	S	12.75
cali_ckn_m	cali_ckn	M	16.75
cali_ckn_l	cali_ckn	L	20.75
ckn_alfredo_s	ckn_alfredo	S	12.75
ckn_alfredo_m	ckn_alfredo	M	16.75
ckn_alfredo_l	ckn_alfredo	L	20.75
ckn_pesto_s	ckn_pesto	S	12.75

Reviewing the first 10 rows of data without encountering any errors indicates successful importation and suggests that our data was imported accurately. This initial validation step confirms the integrity of the imported data and ensures that further analysis can proceed smoothly.

Data Cleaning

Total rows in our dataset.

For order_details Table:

```
1 •  SELECT COUNT(*) AS No_Of_Rows FROM order_details;
```

Output

No_Of_Rows
48620

Our order_details table comprises a total of 48,620 rows.

For orders Table:

```
1    SELECT COUNT(*) AS No_Of_Rows FROM orders;
```

Output

No_Of_Rows
21350

Our orders table comprises a total of 21,350 rows.

Data Cleaning

Total rows in our dataset.

For pizza_types Table:

```
1    SELECT COUNT(*) AS No_Of_Rows FROM pizza_types;
```

Output

No_Of_Rows
32

Our pizza_types table comprises a total of 32 rows.

For pizzas Table:

```
1    SELECT COUNT(*) AS No_Of_Rows FROM pizzas;
```

Output

No_Of_Rows
96

Our pizzas table comprises a total of 96 rows.

Checking For Missing Values

For order_details Table

```
1 • SELECT
2       *
3   FROM
4       order_details
5   WHERE
6       order_details_id IS NULL
7           OR order_id IS NULL
8           OR pizza_id IS NULL
9           OR quantity IS NULL;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

order_details_id	order_id	pizza_id	quantity
NULL	NULL	NULL	NULL

For orders Table:

```
1 • SELECT
2       *
3   FROM
4       orders
5   WHERE
6       order_id IS NULL OR order_date IS NULL
7           OR order_time IS NULL;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

order_id	order_date	order_time
NULL	NULL	NULL

Checking For Missing Values

For pizza_types Table:

```
1 • SELECT
2       *
3   FROM
4       pizza_types
5   WHERE
6       pizza_type_id IS NULL OR name IS NULL
7           OR category IS NULL
8           OR ingredients IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pizza_type_id	name	category	ingredients

For pizzas Table:

```
1 • SELECT
2       *
3   FROM
4       pizzas
5   WHERE
6       pizza_id IS NULL
7           OR pizza_type_id IS NULL
8           OR size IS NULL
9           OR price IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pizza_id	pizza_type_id	size	price

Upon review, our analysis indicated blank results, signifying the absence of missing values within our dataset. This reassures us of the completeness and reliability of our data.

Checking For Duplicates

For order_details Table

```
1 •  SELECT order_details_id, COUNT(order_details_id) as COUNT
2    FROM order_details
3    GROUP BY order_details_id
4    HAVING COUNT(order_details_id)>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_details_id	COUNT

For orders Table:

```
1 •  SELECT order_id, COUNT(order_id) as COUNT
2    FROM orders
3    GROUP BY order_id
4    HAVING COUNT(order_id)>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	COUNT

The query returns an empty table, showing there are no duplicate rows.

Checking For Duplicates

For pizza_types Table:

```
1 •  SELECT pizza_type_id, COUNT(pizza_type_id) as COUNT
2    FROM pizza_types
3    GROUP BY pizza_type_id
4    HAVING COUNT(pizza_type_id)>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pizza_type_id	COUNT

For pizzas Table:

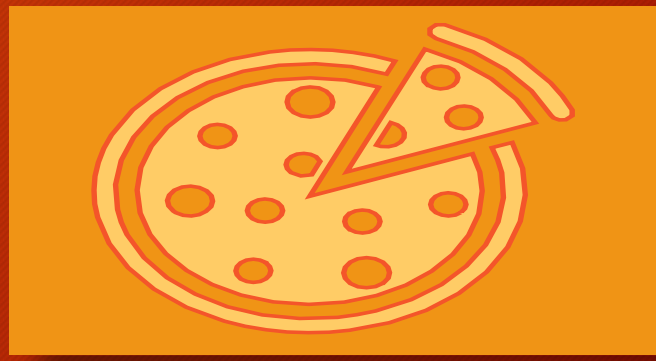
```
1 •  SELECT pizza_id, COUNT(pizza_id) as COUNT
2    FROM pizzas
3    GROUP BY pizza_id
4    HAVING COUNT(pizza_id)>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pizza_id	COUNT

The query returns an empty table, showing there are no duplicate rows.

Data Analysis



Retrieve the total number of orders placed.

To determine the total number of orders, I utilize the “COUNT” function to count the distinct values in the order_id column from the table.

```
1 • SELECT
2       COUNT(order_id) AS Total_orders
3   FROM
4       orders;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total_orders
21350

A total of 21,350 unique orders were recorded.

Calculate the total revenue generated from pizza sales.

To determine the overall revenue, I utilize the ‘SUM’ function to aggregate the values in the price column, followed by applying the ‘ROUND’ function to ensure it’s rounded to the nearest whole number.

```
1 •  SELECT
2        ROUND(SUM((order_details.quantity * pizzas.price)),
3                2) AS Total_Revenue
4    FROM
5        order_details
6            JOIN
7        pizzas ON order_details.pizza_id = pizzas.pizza_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total_Revenue
817860.05

The pizza sales have generated a total revenue of $817,860.05

Calculate the average order value of store.

This query computes the average order value by dividing the total revenue (sum of total_price) by the count of unique order IDs. The result is then formatted as a decimal with 10 digits in total, and 2 decimal places.

```
1 • SELECT
2       ROUND((SUM(order_details.quantity * pizzas.price) / COUNT(DISTINCT order_details.order_id)),
3               2) AS Avg_Order_Value
4   FROM
5       order_details
6           JOIN
7       pizzas ON order_details.pizza_id = pizzas.pizza_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Avg_Order_Value
38.31

The average order value is $38.31

Identify the highest-priced pizza.

To address this query, I'll utilize the "ORDER BY" function with the "DESC" feature to obtain the highest priced pizza.

```
1 • SELECT
2       pizza_types.name, pizzas.price
3   FROM
4       pizzas
5           JOIN
6       pizza_types ON pizzas.pizza_type_id = pizza_types.pizza_type_id
7   ORDER BY pizzas.price DESC
8   LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

name	price
The Greek Pizza	35.95

The highest priced pizza is The Greek Pizza ($35.95).

Identify the most common pizza size ordered.

I've utilized the "COUNT" function to determine the distinct number of order IDs for each pizza size. Then, I've grouped the results by pizza size and sorted them in descending order based on the total number of orders placed within each size.

```sql
SELECT
    pizzas.size,
    COUNT(DISTINCT order_details.order_id) AS Total_no_orders
FROM
    pizzas
        JOIN
    order_details ON pizzas.pizza_id = order_details.pizza_id
GROUP BY pizzas.size
ORDER BY COUNT(order_details.order_id) DESC
LIMIT 1;
```

Output

size	Total_no_orders
L	12736

The most common pizza size ordered is Large - L (12,736).

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

I've utilized the "SUM" function to determine the total number of quantity sold for each pizza. Then, I've grouped the results by pizza name & sorted them in descending order based on the total number of quantity sold within each pizza type.

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

Output

name	Total_Quantity_Sold
The Classic Deluxe Pizza	2453
The Barbecue Chicken Pi...	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.

I used the “SUM” function to calculate the total quantity of pizzas sold for each pizza category by joining the required table.Then, I grouped the results by pizza category and arranged them in descending order based on the total quantity of pizzas sold.

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

Output

category	Total_Quantity_Sold
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

Sales Trend

To provide insight into busiest days and times, I conducted two distinct queries to showcase weekdays and hours separately. This approach aims to offer a clear visualization of when the establishment experiences the highest levels of activity, aiding the owner in optimizing staffing schedules and resource allocation strategies.

I utilized the “DAYNAME” function to extract the day from the order_date column, enabling me to discern the distribution of orders throughout the week. Subsequently, I computed the total number of orders and calculated the average number of orders per day by dividing the total number of orders by the number of days. This information was grouped by the day of the week and ordered based on the count of orders. Below are the queries I employed for this analysis:

Sales Trend By Day Of Week

Throughout the year, Friday emerges as the busiest day of the week, with the business recording the highest pizza sales on Fridays. Friday witnessed a total of 3538 orders filled, averaging 71 orders per day. This is followed by Thursday and Saturday, with total orders of 3239 and 3158, and average daily orders of 62 and 61, respectively.

```sql
SELECT
    DAYNAME(orders.order_date) AS Day_Of_Week,
    COUNT(DISTINCT orders.order_id) AS Total_order,
    ROUND((COUNT(DISTINCT order_id) / COUNT(DISTINCT order_date)),
            0) AS Average_daily_order
FROM
    orders
GROUP BY DAYNAME(orders.order_date)
ORDER BY Total_order DESC;
```

Output

Day_Of_Week	Total_order	Average_daily_order
Friday	3538	71
Thursday	3239	62
Saturday	3158	61
Wednesday	3024	58
Tuesday	2973	57
Monday	2794	58
Sunday	2624	50

Sales Trend By Hour Of Day

To extract only the hour from the order_time column, I utilized the “HOUR ” function. Below is the query utilized for this purpose:

```
1 • SELECT
2       HOUR(order_time) AS Hour, COUNT(order_id) AS Order_Count
3   FROM
4       orders
5   GROUP BY (HOUR(order_time))
6   ORDER BY Order_Count DESC;
```

Output

Hour	Order_Count
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

The busiest times for pizza orders are typically around “12pm”, “1pm”, “5pm”, “6pm”, and “7pm”, indicating that the business experiences its highest order volumes during lunch and dinner hours.

Sales Trend By Month

Moreover, I took the initiative to identify the busiest month for orders. This additional analysis will provide valuable insights for our dashboard.

```sql
SELECT
    MONTHNAME(orders.order_date) AS Month,
    COUNT(DISTINCT orders.order_id) AS Total_Order
FROM
    orders
GROUP BY MONTHNAME(orders.order_date)
ORDER By Total_Order DESC;
```

Output

Month	Total_Order
July	1935
May	1853
January	1845
August	1841
March	1840
April	1799
November	1792
June	1773
February	1685
December	1680
September	1661
October	1646

July stands out as the month with the highest number of orders, totaling 1935.

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

I used the “COUNT” function to calculate the total number of pizzas in each category. Then, I grouped the results by pizza category to identity the category-wise distribution of pizzas.

```
1 •  SELECT
2        category, COUNT(name) As No_Of_Pizza
3    FROM
4        pizza_types
5    GROUP BY category;
```

Output

	category	No_Of_Pizza
▸	Chicken	6
	Classic	8
	Supreme	9
	Veggie	9

Supreme and Veggie has most no. of varieties followed by Classic.

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

```sql
SELECT
    category, COUNT(name) As No_Of_Pizza
FROM
    pizza_types
GROUP BY category;SELECT
    ROUND(AVG(Quantity), 0) AS Avg_No_Of_Order_Per_Day
FROM
    (SELECT
        orders.order_date, SUM(order_details.quantity) AS Quantity
    FROM
        order_details
    JOIN orders ON order_details.order_id = orders.order_id
    GROUP BY orders.order_date) AS Quantity_Sold;
```

Output

Avg_No_Of_Order_Per_Day
138

Average number of pizza ordered per day is 138.

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

To address this query, I'll utilize the "SUM" function to compute the revenue by joining the required table. Additionally, I'll employ the "GROUP BY" statement to group data by pizza name, and the "ORDER BY" statement with the "DESC" feature to obtain the Top 3 result.

```sql
SELECT
    pizza_types.name,
    SUM(order_details.quantity * pizzas.price) AS Total_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.name
ORDER BY Total_Revenue DESC
LIMIT 3;
```

Output

name	Total_Revenue
The Thai Chicken Pizza	43434.25
The Barbecue Chicken Pi...	42768
The California Chicken Pi...	41409.5

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

```sql
SELECT
    pizza_types.category,
    (ROUND((SUM(order_details.quantity * pizzas.price) / (SELECT
                SUM(order_details.quantity * pizzas.price)
            FROM
                order_details
                    JOIN
                pizzas ON order_details.pizza_id = pizzas.pizza_id) * 100),
        2)) AS Percentage_Contribution
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;
```

Output

category	Percentage_Contribution
Classic	26.91
Veggie	23.68
Supreme	25.46
Chicken	23.96

The category that generated the highest percentage is Classic with 26.91% of total sales and then followed by Supreme with 25.46%.

Analyze the cumulative revenue generated over time.

```sql
SELECT order_date,
ROUND(sum(Revenue) OVER (ORDER BY order_date),2) AS Cumulative_Revenue
FROM
    (SELECT orders.order_date, SUM(order_details.quantity*pizzas.price) AS Revenue
    FROM order_details
    JOIN orders
    ON order_details.order_id = orders.order_id
    JOIN pizzas
    ON order_details.pizza_id = pizzas.pizza_id
    GROUP BY orders.order_date) AS Daily_Revenue;
```

Output

order_date	Cumulative_Revenue
2015-01-01	2713.85
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	23990.35
2015-01-11	25862.65
2015-01-12	27781.7
2015-01-13	29831.3
2015-01-14	32358.7
2015-01-15	34343.5
2015-01-16	36937.65
2015-01-17	39001.75

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

To address this query, I'll utilize the CTE in which I used the "SUM" function to calculate the total revenue by category then I used "RANK" function to rank top3 pizzas in each category. After that, I grouped the results by pizza category and name.

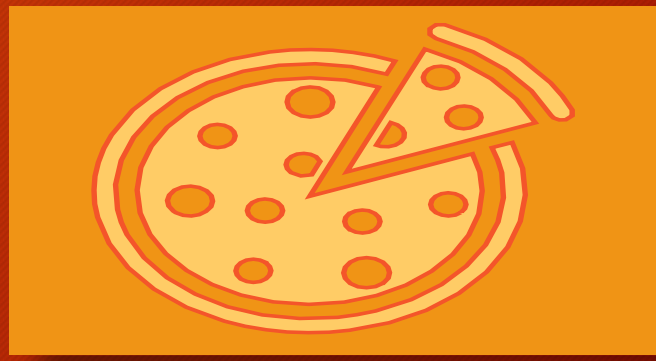
```sql
WITH my_cte AS(
SELECT pizza_types.category, pizza_types.name,
sum(order_details.quantity*pizzas.price) AS Revenue,
RANK() OVER(PARTITION BY category ORDER BY SUM(order_details.quantity*pizzas.price) DESC) AS rn
FROM pizza_types
JOIN pizzas
ON pizza_types.pizza_type_id = pizzas.pizza_type_id
JOIN order_details
ON pizzas.pizza_id = order_details.pizza_id
GROUP BY pizza_types.category, pizza_types.name)

SELECT category, name, Revenue FROM my_cte
WHERE rn <= 3;
```

Output

category	name	Revenue
Chicken	The Thai Chicken Pizza	43434.25
Chicken	The Barbecue Chicken Pi...	42768
Chicken	The California Chicken Pi...	41409.5
Classic	The Classic Deluxe Pizza	38180.5
Classic	The Hawaiian Pizza	32273.25
Classic	The Pepperoni Pizza	30161.75
Supreme	The Spicy Italian Pizza	34831.25
Supreme	The Italian Supreme Pizza	33476.75
Supreme	The Sicilian Pizza	30940.5
Veggie	The Four Cheese Pizza	32265.70000000065
Veggie	The Mexicana Pizza	26780.75
Veggie	The Five Cheese Pizza	26066.5

Findings and Recommendation



Findings

- The dataset comprises a total of 21,350 unique orders, with an average order value of $ 38.31, contributing to a total revenue of $817,860.
- The Total quantity of pizza sold as derived from this dataset is 49,574.
- The day of the week with the most orders is Friday with 3538 total orders.
- July stands out as the month with the highest number of orders, totaling 1935 and most orders are at the beginning of the year.
- The best-selling pizza is The Classic Deluxe Pizza, with 2,453 orders, while The Brie Carrie Pizza ranks as the worst-selling pizza, with only 490 orders.
- Classic and Supreme categories are the pizza categories with the contribution of 26.91% and 25.46% of total sales respectively.

Recommendations

Based on my comprehensive analysis and insights, here are the conclusions drawn and future recommendations aimed at enhancing the success of the store:

- Launch targeted promotions or special deals specifically on Sundays and Mondays to incentivize purchases and boost sales on these slower days.
- Consider promoting more diverse pizza options within the Chicken and Veggie categories to stimulate sales.
- Implement targeted marketing strategies to increase awareness and demand for XL and XXL pizza sizes.
- Analyze customer preferences and behaviors to understand why sales decrease drastically in August to October, exploring potential factors such as seasonal changes, competitor activity, or economic trends.
- Leverage data analytics tools to conduct deeper analysis of customer behavior and preferences, identifying opportunities for targeted marketing efforts and personalized promotions.

THANK YOU