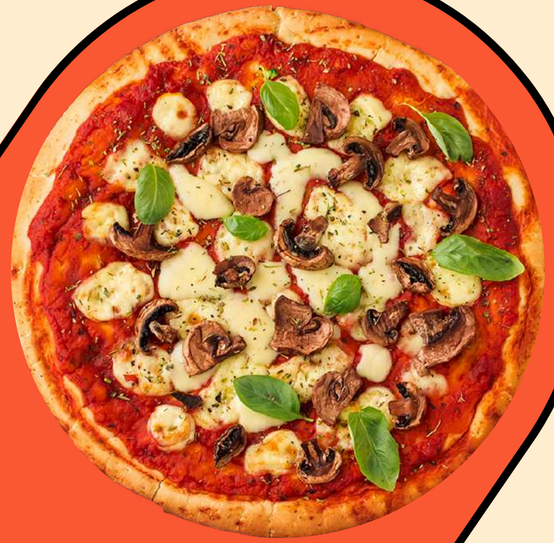
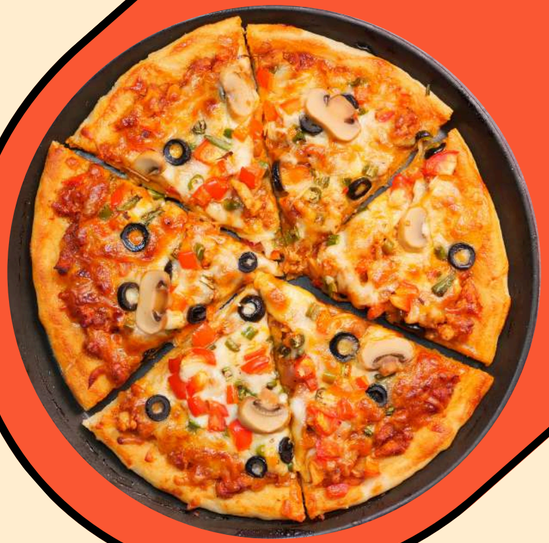
PIZZA



PIZZA SALES DATA



PIZZA SALES ANALYSIS (MYSQL)

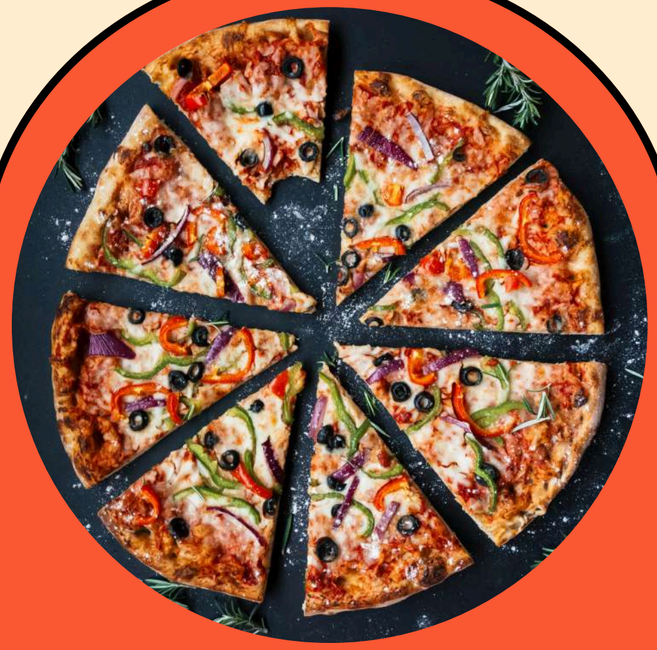
Project Overview

This project involves a comprehensive analysis of a pizza restaurant's sales data using MySQL. The goal was to extract actionable insights regarding sales performance, customer preferences, and operational efficiency. By querying a relational database, I analyzed key metrics such as total revenue, order distribution, and top-selling pizza categories.

DATABASE SCHEMA

The analysis was performed on the pizzahut database, which consists of four primary tables:

- orders: Contains details of each order (ID, date, and time).
- order_details: Contains the specific items within each order (quantity and pizza ID).
- pizzas: Contains technical details like pizza size and price.
- pizza_types: Contains descriptive details like name and category (e.g., Classic, Veggie).



-- RETRIEVE THE TOTAL NUMBER OF ORDERS PLACED.

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

	total_orders
▶	21350

-- CALCULATE THE TOTAL REVENUE GENERATED FROM PIZZA SALES.

```
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
```

Result Grid

	total_sales
▶	817860.05

-- IDENTIFY THE HIGHEST-PRICED PIZZA.

```sql
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;
```

Result Grid | Filter Rows:

name	price
The Greek Pizza	35.95

-- IDENTIFY THE MOST COMMON PIZZA SIZE ORDERED.

```sql
SELECT
    pizzas.size,
    COUNT(order_details.order_detail_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;
```

Result Grid | Filter Rows:

size	order_count
L	18526
M	15385
S	14137
XL	544
XXL	28

-- LIST THE TOP 5 MOST ORDERED PIZZA TYPES ALONG WITH THEIR QUANTITIES.

```sql
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;
```

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.

```sql
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;
```

Result Grid | Filter Ro

category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

-- DETERMINE THE DISTRIBUTION OF ORDERS BY HOUR OF THE DAY

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

Result Grid | Filter Ro

hour	order_count
11	1231
12	2520
13	2455
14	1472
15	1468
16	1920
17	2336
18	2399
19	2009
20	1642
21	1198
22	663
23	28
10	8
9	1

-- JOIN RELEVANT TABLES TO FIND THE CATEGORY-WISE DISTRIBUTION OF PIZZAS.

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

Result Grid | Filter Rows:

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

-- DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE.

```
    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;
```

Result Grid | Filter Rows:

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

-- DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE.

```
    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;
```

Result Grid | Filter Rows:

name	revenue
The Thai Chicken Pizza	43434.25
The Barbecue Chicken Pizza	42768
The California Chicken Pizza	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
                    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;
```

Result Grid | Filt

category	revenue
Classic	26.91
Supreme	25.46
Chicken	23.96
Veggie	23.68

-- ANALYZE THE CUMULATIVE REVENUE GENERATED OVER TIME.

```
select order_date,
sum(revenue) over(order by order_date) as cum_revenue
from
(select orders.order_date,
sum(order_details.quantity*pizzas.price) as revenue
from order_details join pizzas
on order_details.pizza_id = pizzas.pizza_id
join orders
on orders.order_id = order_details.order_id
group by orders.order_date) as sales;
```

Result Grid | Filter Rows:

order_date	cum_revenue
2015-11-18	722646.1000000002
2015-11-19	725341.0000000002
2015-11-20	727729.1000000002
2015-11-21	729813.0500000002
2015-11-22	731181.7500000001
2015-11-23	733646.9000000001
2015-11-24	735876.9500000002
2015-11-25	738240.2000000002
2015-11-26	742646.1500000001
2015-11-27	747068.6000000001
2015-11-28	749036.6500000001
2015-11-29	750935.6500000001
2015-11-30	753158.9000000001
2015-12-01	755235.6000000001
2015-12-02	757449.7000000001

-- DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE FOR EACH PIZZA CATEGORY.

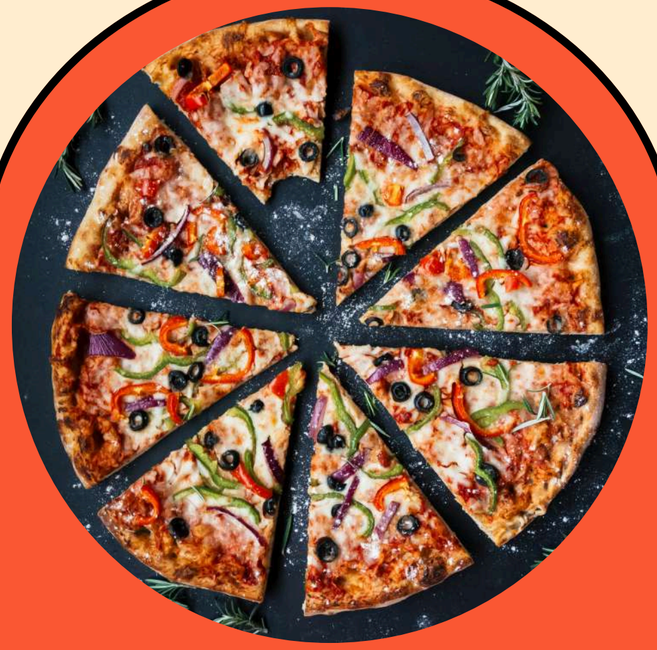
```sql
select category, name, revenue,
rank() over(partition by category order by revenue desc) as rn
from
(SELECT
    pizza_types.category,
    pizza_types.name,
    SUM((order_details.quantity) * pizzas.price) AS revenue
FROM
    pizza_types
        JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
        JOIN
    order_details ON order_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category , pizza_types.name) as a;
```

sult Grid | Filter Rows: | Export: | Wrap

category	name	revenue	rn
Chicken	The Thai Chicken Pizza	43434.25	1
Chicken	The Barbecue Chicken Pizza	42768	2
Chicken	The California Chicken Pizza	41409.5	3
Chicken	The Southwest Chicken Pizza	34705.75	4
Chicken	The Chicken Alfredo Pizza	16900.25	5
Chicken	The Chicken Pesto Pizza	16701.75	6
Classic	The Classic Deluxe Pizza	38180.5	1
Classic	The Hawaiian Pizza	32273.25	2
Classic	The Pepperoni Pizza	30161.75	3
Classic	The Greek Pizza	28454.100000000013	4
Classic	The Italian Capocollo Pizza	25094	5

TECHNICAL SKILLS DEMONSTRATED

- Joins: Using INNER JOIN to connect multiple tables (up to 3 tables in a single query).
- Aggregations: Proficient use of SUM(), COUNT(), AVG(), and ROUND().
- Grouping & Filtering: Advanced use of GROUP BY, ORDER BY, and LIMIT.
- Subqueries: Nesting queries to perform complex calculations like average daily sales.
- Window Functions: Implementing RANK() and OVER(PARTITION BY) for granular data ranking.




THANK YOU