PIZZA SALES REPORT | SQL DATA ANALYSIS PROJECT : 🚀

🔍 **OVERVIEW :** Analyzed a pizza restaurant's sales dataset using SQL to extract actionable insights and help improve business performance.

Comprehensive SQL analysis of pizza sales data to derive actionable business insights . Perfect for showcasing data analysis and SQL skills !

- ANKIT KUMAR

LARANA PIZZA

🔧 Tech Stack :

- 🗃️Databas : SQL (MySQL/PostgreSQL)
  -
- 📊Visualization: Power BI/Tableau (optional)
- 📂 Data: Sales, orders, pizza details

📊 Key Insights :

✅ 💰 Revenue Trends: Analyzed daily, weekly & monthly sales
✅ 🍕 Top Pizzas: Identified best & worst sellers by category
✅ ⏱️ Order Timing: Peak hours & days for maximum sales
✅ 📦 Avg. Order Value: Calculated customer spending patterns

🛠️ Skills Demonstrated :

✔️ SQL Queries: JOIN, GROUP BY, subqueries, aggregations (SUM, AVG)
✔️ **Data Modeling**: Database design with PK/FK relationships
✔️ **Business** Intelligence: Transforming raw data into insights

INSTRUCTIONS

LARANA PIZZA

DATABASE NAME : PIZZAHUT

```
create database pizzahut;
```

```
create table orders (
order_id int not null,
order_date date not null,
order_time time not null,
primary key(order_id) ) ;
```

```
create table order_details (
order_details_id int not null,
order_id int not null,
pizza_id text not null,
quantity int not null,
primary key(order_details_id) ) ;
```

Tables:

1.orders

- Stores information about each individual order.
- Columns:
  - order_id (Primary Key): Unique identifier for each order.
  - order_date: Date the order was placed.
  - order_time: Time the order was placed.

2.order_details

- Contains detailed information for each item in an order.
- Columns:
  - order_details_id (Primary Key): Unique ID for each line item.
  - order_id: Foreign key linking to the orders table.
  - pizza_id: Identifier for the specific pizza ordered.
  - quantity: Number of units of the pizza ordered.

PURPOSE :

This schema forms the foundation for performing queries and analysis such as:

- Tracking total sales and revenue
- Identifying best-selling pizzas
- Analyzing order patterns by date and time
- Supporting visualization and dashboard reporting

LARANA PIZZA

BASICS QUESTIONS :

Question 1 . > Retrieve the total number of orders placed.

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

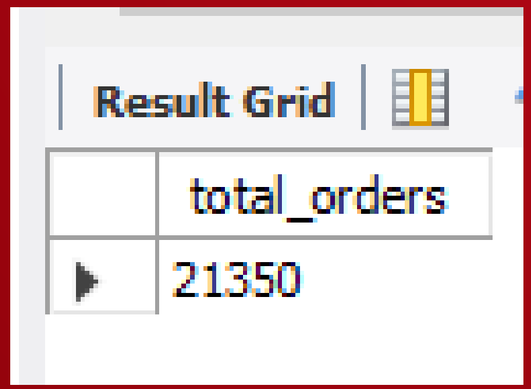
SUMMARY :

- This query is used to find how many total orders have been placed in the orders table. It's a basic and essential KPI (Key Performance Indicator) in sales analysis.

EXPLANATION :

- SELECT: This tells the database you want to retrieve data.
- COUNT(order_id): This is an aggregate function that counts how many non-null order_id values exist in the orders table. Since order_id is the primary key, every row will have a unique and non-null order_id, so you're effectively counting the total number of orders.
- AS total_orders: This gives a custom name (alias) to the result column. Instead of seeing something like count(order_id), it will show as total_orders in the output.

RESULT :



LARANA PIZZA

BASICS Questiion 2 . > Calculate the total revenue generated from pizza sales.

```
/*select
sum(order_details.quantity * pizzas.price) as total_sales
from order_details
join pizzas
 on pizzas.pizza_id = order_details.pizza_id*/
```

✅ 1. JOIN CLAUSE :

- This joins the order_details table with the pizzas table.
- It matches each pizza ordered (from order_details) with its price (from pizzas) using the common column pizza_id.

```
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
```

✅ 2. Multiplication for Revenue :

- This calculates the revenue for each line item, i.e., number of pizzas ordered × price per pizza.

✅ 3. SUM Function :

- Adds up the total revenue from all order lines (all pizzas sold).

✅ 4. ROUND Function:

- Rounds the total revenue to 2 decimal places for clean currency formatting (e.g., ₹15345.75 instead of ₹15345.754999...).

✅ 5. Alias:

- Renames the output column to total_sales for better readability.

✅ OUTPUT :

Result Grid

total_sales
817860.05

LARANA PIZZA

BASICS Question 3 . > 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;

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;
```

🔍 QUERY PURPOSE :

TO FIND THE HIGHEST-PRICED PIZZA.

⚙️ HOW IT WORKS :

- JOINS PIZZA_TYPES AND PIZZAS TABLES USING PIZZA_TYPE_ID.
- SELECTS THE PIZZA NAME AND ITS PRICE.
- SORTS BY PRICE IN DESCENDING ORDER.
- USES LIMIT 1 TO RETURN ONLY THE MOST EXPENSIVE PIZZA.

BASICS Question 4 . > Identify the most common pizza size ordered.

```
select quantity, count(order_details_id) from order_details group by quantity;
select pizzas.size, count(order_details.order_details_id) from pizzas join order_details on pizzas.pizza_id = order_details.pizza_id group by pizzas.size;
```

```
/* 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; */
```

```
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;
```

🔍 QUERY PURPOSE :

To find the **most commonly ordered pizza size**.

⚙️ How It Works:

- Joins `pizzas` and `order_details` using `pizza_id`.
- Groups data by `pizzas.size`.
- Counts how many times each size was ordered.
- Sorts results in **descending** order of count.
- `LIMIT 1` returns the **most ordered size**.

OUTPUT :

Result Grid | Filter

	size	order_count
▶	L	18526

LARANA PIZZA

- BASICS Question 5 . > List the top 5 most ordered pizza types along with their quantities.

```
select pizza_types.name, order_details.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;
```

```
/* 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 ;*/
```

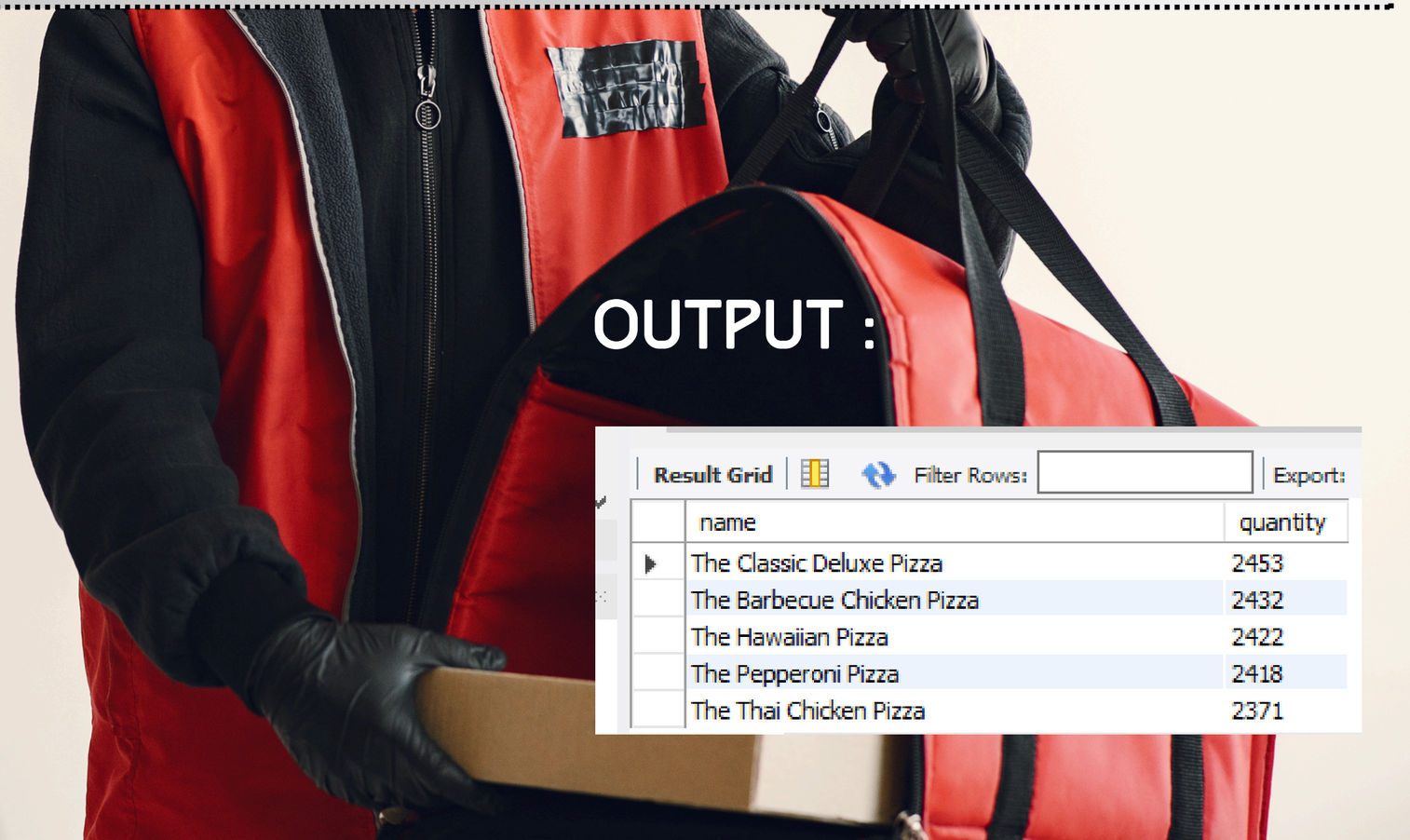
```
select pizza_types.name, sum(order_details.quantity) as quantity from pizza_types join pizzas on pizza_types.pizza_type_id = pizzas.pizza_type_id join order_details
on order_details.pizza_id = pizzas.pizza_id group by pizza_types.name order by quantity desc limit 5;
```

🔍 **QUERY PURPOSE :**

To find the top 5 most ordered pizza types by total quantity sold.

⚙️ **HOW IT WORKS :**

- Joins pizza_types, pizzas, and order_details using pizza_type_id and pizza_id.
- Groups results by pizza type name.
- Uses SUM(order_details.quantity) to get total quantity ordered for each type.
- Sorts by quantity in descending order.
- LIMIT 5 returns the top 5 pizza types.



OUTPUT :

Result Grid | Filter Rows: | Export:

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

LARANA PIZZA

Intermediate Questions :

Question . > 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;
```

🔍 **QUERY PURPOSE :** To find the total quantity of pizzas ordered, grouped by pizza category (like "Classic", "Veggie", "Meat", etc.).

OUTPUT :

Result Grid | Filter

category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

⚙️ Step-by-Step Breakdown :

1. **Joins:**
   - `pizza_types` is joined with `pizzas` via `pizza_type_id` to get category info.
   - `pizzas` is joined with `order_details` via `pizza_id` to get quantity data.
2. **SUM(order_details.quantity):**
   - Adds up how many pizzas were ordered for each category.
3. **GROUP BY pizza_types.category:**
   - Groups the results by category so each row represents a different pizza category.
4. **ORDER BY quantity DESC:**
   - Sorts the result from the **highest to lowest** quantity ordered.

🧠 Summary :

This query helps you understand which pizza category is most popular based on total orders — great for marketing and inventory decisions!

Intermediate Question 2 . > Determine the distribution of orders by hour of the day.

🔍 Query Purpose :

To analyze the distribution of orders by hour — i.e., to see at what times of the day customers place the most orders.

```
-- select * from orders;
-- select hour(order_time) from orders;
select hour(order_time), count(order_id) from orders group by hour(order_time);
```

⚙️ Step-by-Step Breakdown :

✅ **HOUR(order_time):**

- **Extracts the hour part (0 to 23) from the order_time field.**
  - **For example, if order_time = '18:45:00', then HOUR(order_time) = 18.**

✅ **COUNT(order_id):**

- **Counts how many orders were placed in each hour.**

✅ **GROUP BY HOUR(order_time):**

- **Groups the orders by each hour of the day.**

OUTPUT :

Result Grid | Filter Rows:

hour(order_time)	count(order_id)
17	2336
18	2399
19	2009
20	1642
21	1198

🧠 **Summary:**

This query helps you understand peak business hours, which is super useful for:

- **Staff scheduling**
- **Marketing promotions**
- **Operational planning**

Intermediate Question 3 . > Join relevant tables to find the category-wise distribution of pizzas.

```
-- select * from pizza_types;
select category , count(name) from pizza_types group by category;
```

📌 SUMMARY :

This query gives you the distribution of available pizza types per category, which helps in understanding the menu structure or variety offered under each type.

🔍 *Query Purpose :*

To find out how many different pizzas exist in each category, like "Veggie", "Meat", "Classic", etc.

⚙️ Step-by-Step Breakdown:

✅ category:

- Comes from the pizza_types table. It indicates the type or style of pizza (e.g., Classic, Veggie, etc.).

✅ COUNT(name):

- Counts how many unique pizza names belong to each category.
- Since name refers to individual pizza types (like Margherita, BBQ Chicken), this tells you how many pizza options each category includes.

✅ GROUP BY category:

- Groups the pizza types by their category to count how many are in each one.

Result Grid | Filter Row

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

Intermediate Question 4 . > Group the orders by date and calculate the average number of pizzas ordered per day.

🔍 QUERY PURPOSE:

To calculate the average number of pizzas ordered per day.

```
select * from orders;
select orders.order_date, sum(order_details.quantity) from orders join order_details on orders.order_id = order_details.order_id group by orders.order_date;
```

```
/* AVERAGE OF PER DAY
select avg(quantity) 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; */
```

```
-- Round VALUE per day pizzas - orders
select round(avg(quantity),0) as pizzas_perDAY_orderd 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;
```

⚙️ HOW IT WORKS:

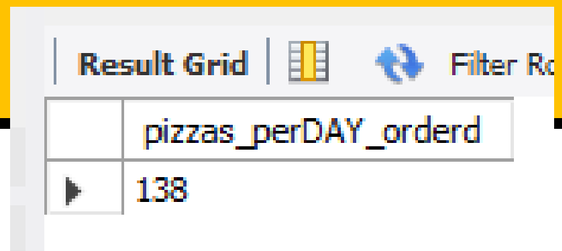
✅ Step-by-Step:

1. Inner Query:
   - Groups orders by order_date.
   - Calculates the total pizzas (SUM(quantity)) for each day.
2. Outer Query:
   - Takes the daily totals and calculates the average (AVG) across all days.
   - ROUND(..., 0) rounds it to a whole number.

🧠 Summary:

This query shows how many pizzas, on average, were ordered each day, helping you understand daily sales trends.

OUTPUT :



Result Grid | Filter Ro

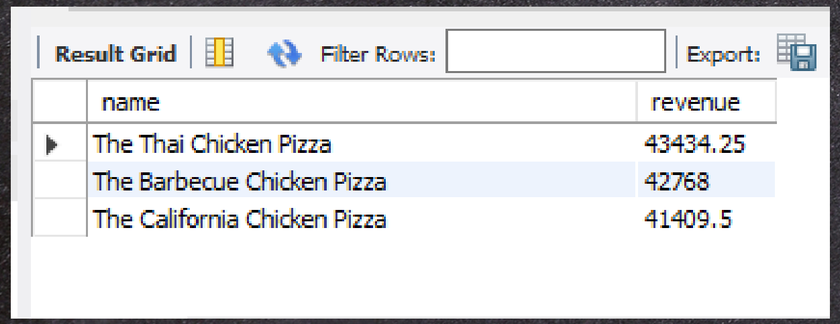
	pizzas_perDAY_orderd
▶	138

Intermediate Question 5 . > Determine the top 3 most ordered pizza types based on revenue.

```
-- select * from pizza_types;
/* select pizza_types.name, 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; */
```

```
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 :



Result Grid | Filter Rows: | Export:

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

🔍 QUERY PURPOSE:

To find the top 3 most ordered pizza types based on total revenue generated.

🧠 Summary:

This query tells you which pizza types make the most money, helping in business decisions like promotions, pricing, or stocking.

⚙️ How It Works:

- Joins 3 Tables:
- pizza_types gives pizza names.
- pizzas gives price info.
- order_details gives quantity ordered.
- Calculates Revenue:
- order_details.quantity * pizzas.price gives revenue per item.
- SUM(...) adds up total revenue per pizza type.
- Groups by Pizza Name:
- Each row represents a unique pizza type with total revenue.
- Sorts by Revenue (Descending):
- Highest-earning pizzas come first.
- Limits to Top 3:
- Only shows the top 3 pizza types by revenue.

ANKIT

```
Advanced Questions :
Advanced Questions 1 : > Calculate the percentage contribution of each pizza type to total revenue.
```

🔍 Query Purpose: To calculate the percentage of total revenue contributed by each pizza category (like Classic, Veggie, Meat).

```
/* select pizza_types.category, 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 order by revenue desc ; */
```

```
/* select pizza_types.category, (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 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 ; */
```

🧠 What This Query Does:

```
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:

Result Grid | Filter

category	revenue
Classic	26.91
Supreme	25.46
Chicken	23.96
Veggie	23.68

⚙️ How It Works Step-by-Step:

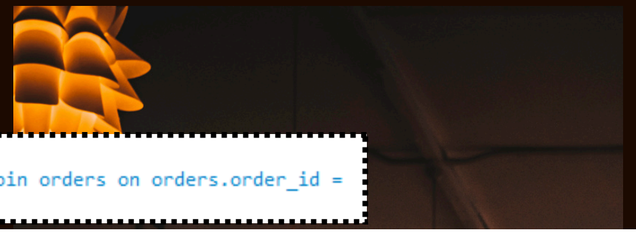
- Joins:
- Combines pizza_types, pizzas, and order_details to get category, price, and quantity.
- Revenue per Category:
- SUM(order_details.quantity * pizzas.price) calculates total revenue for each category.
- Total Revenue (Subquery):
- The subquery calculates the total revenue from all pizzas.
- Percentage Calculation:
- Divides category revenue by total revenue, multiplies by 100 to get percentage.
- ROUND(..., 2) keeps it to 2 decimal places.
- Grouping:
- GROUP BY pizza_types.category groups the results per category.
- Ordering:
- ORDER BY revenue DESC shows the highest contributors first.

✅ Summary:

Advanced Questions . 2 > Analyze the cumulative revenue generated over time.

🔍 Query Purpose:

To calculate cumulative revenue over time, i.e., how total revenue builds up day by day.



```
-- select * from orders;
/* 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; */
```

```
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;
```

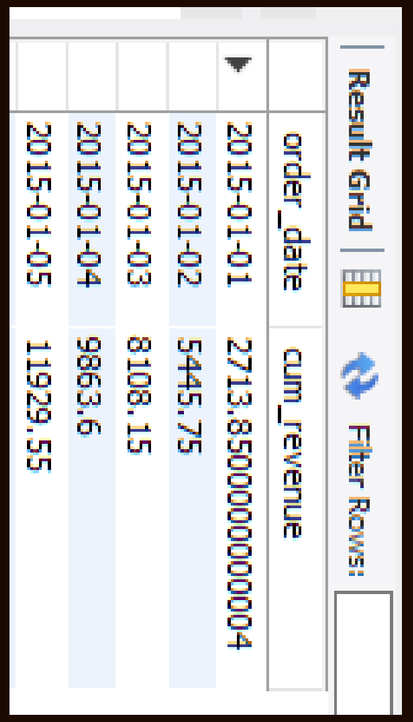
⚙️ Step-by-Step Breakdown:

- Inner Query (Named sales):
- Joins orders, order_details, and pizzas.
- Groups by order_date.
- Calculates daily revenue using:
- SUM(order_details.quantity * pizzas.price).
- Outer Query:
- Uses a window function:
- SUM(revenue) OVER (ORDER BY order_date)
- This calculates the running total of revenue (i.e., cumulative revenue) sorted by date.

✅ Summary:

This query gives a day-by-day growth of total revenue, helping you track sales performance over time — useful for trend analysis, forecasting, or visual dashboards.

✅OUTPUT :



Result Grid | Filter Rows:

order_date	cum_revenue
2015-01-01	2713.8500000000004
2015-01-02	5445.75
2015-01-03	8108.15
2015-01-04	9863.6
2015-01-05	11929.55

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

🔍 Query Purpose: To find the top 3 most revenue-generating pizza types in each category.

```
-- select * from pizza_types;

/* 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; */
```

```
/* 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; */
```

```
select name, revenue from
(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) as B where rn <= 3;
```

⚙️ Step-by-Step Breakdown:

- Inner Query (A):
- Joins pizza_types, pizzas, and order_details.
- Groups by category and pizza name.
- Calculates revenue per pizza using SUM(quantity * price).
- Middle Layer:
- Uses RANK() with PARTITION BY category to assign a ranking within each category based on revenue (ORDER BY revenue DESC).
- Outer Query:
- Filters to get only the top 3 (rn <= 3) pizzas per category.

Result Grid | Filter Rows: | Export: | Wrap

OUTPUT :

name	revenue
The Thai Chicken Pizza	43434.25
The Barbecue Chicken Pizza	42768
The California Chicken Pizza	41409.5
The Classic Deluxe Pizza	38180.5
The Hawaiian Pizza	32273.25

✅ Summary:

- This query gives a category-wise breakdown of the top-selling pizzas by revenue — perfect for targeted promotions, menu optimization, or insights into customer preferences. 🍕💰📊

LARANA PIZZA

THANK YOU!