1. Retrieve the total number of orders placed.

>>> **Query:**

```
SELECT
   COUNT(quantity) AS total_orders
FROM
   order_details;
```

- **Explanation:**
  - This query returns the **total number of ordered line Placed** in the order_details table.

2. Calculate the total revenue generated from pizza sales.

>>> **Query:**

```
SELECT
        ROUND(SUM(p.price * o.quantity), 2) AS total_revenue
     FROM
        pizzas p
          JOIN
        order_details o ON p.pizza_id = o.pizza_id;
```

- **Explanation:**
  - This query calculates the **total revenue generated from all pizza orders**.
    It multiplies the price of each pizza by the quantity ordered, sums all those values, and rounds the final result to two decimal places.

3. Identify the highest-priced pizza.

>>> **Query:**

```
SELECT
   p.name, pi.price
FROM
   pizza_types p
     JOIN
   pizzas pi ON p.pizza_type_id = pi.pizza_type_id
ORDER BY pi.price DESC
```

```
LIMIT 1;
```

- **Explanation:**
  This query joins the pizza_types and pizzas tables to match each pizza with its type, then sorts all pizzas by price in descending order. Finally, it returns the single most expensive pizza along with its price.

4. Identify the most common pizza size ordered.

>>> **Query:**

```
SELECT
   p.size, COUNT(o.quantity) AS common_pizza_ordered
FROM
   pizzas p
      JOIN
   order_details o ON p.pizza_id = o.pizza_id
GROUP BY p.size
ORDER BY common_pizza_ordered DESC
LIMIT 1;
```

- **Explanation:**
  - This query finds which pizza size is ordered the most. It groups all orders by pizza size, counts how many times each size appears in the order records, and then returns the single most frequently ordered size by sorting the results in descending order.

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

>>> **Query:**

```
SELECT
   p.name, SUM(o.quantity) AS total_orders
FROM
   pizza_types p
      JOIN
   pizzas pi ON p.pizza_type_id = pi.pizza_type_id
      JOIN
   order_details o ON o.pizza_id = pi.pizza_id
GROUP BY p.name
ORDER BY total_orders DESC
```

```
LIMIT 5;
```

- **Explanation:**
    - This query identifies the five most-ordered pizza types. It joins `pizza_types`, `pizzas`, and `order_details`, groups the data by pizza name, calculates the total quantity ordered for each type, and then returns the top five based on highest order count.

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

>>> **Query:**

```
SELECT
   p.category, SUM(quantity) AS total_quantity
FROM
   pizza_types p
      JOIN
   pizzas pi ON p.pizza_type_id = pi.pizza_type_id
      JOIN
   order_details o ON o.pizza_id = pi.pizza_id
GROUP BY p.category
ORDER BY total_quantity;
```

- **Explanation:**
    - This query calculates the **total quantity of pizzas sold in each category**. It joins the pizza type, pizza, and order details tables, groups the data by category, and sums the quantities. The results are then ordered from the lowest to highest total quantity.

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

>>> **Query:**

```
SELECT
   HOUR(order_time) AS hours, COUNT(order_id) AS count_order
FROM
   orders
GROUP BY HOUR(order_time);
```

- **Explanation**:

    - This query shows **how many orders were placed in each hour of the day**. It extracts the hour from the `order_time`, groups all orders by that hour, and counts how many orders occurred in each time slot.

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

>>> **Query:**

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

- **Explanation:**
    - This query counts how many **pizza types** exist in each **category**. It groups the rows in the `pizza_types` table by category and returns the total number of pizza names within each group.

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

>>> **Query:**

```
SELECT
    AVG(pizza) AS daily_average
FROM
    (SELECT
        DATE(orders.order_date) AS date, SUM(o.quantity) AS pizza
    FROM
        order_details o
    JOIN orders ON o.order_id = orders.order_id
    GROUP BY DATE(orders.order_date)) t1;
```

- **Explanation:**
    - This query calculates the **average number of pizzas sold per day**. It first finds how many pizzas were ordered on each date, then takes the average of those daily totals to produce the final daily average.

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

>>> **Query:**

```
SELECT
   ROUND(SUM(p.price * o.quantity), 2) AS total_revenue,
   pi.name
FROM
   pizzas p
      JOIN
   order_details o ON p.pizza_id = o.pizza_id
      JOIN
   pizza_types pi ON pi.pizza_type_id = p.pizza_type_id
GROUP BY pi.name
ORDER BY total_revenue DESC
LIMIT 3;
```

- **Explanation:**
    - This query finds the **top three pizzas that generated the highest revenue**. It calculates revenue by multiplying each pizza's price with the ordered quantity, groups the totals by pizza name, and sorts them in descending order to return the top three earners.

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

>>> **Query:**

```
SELECT
   pi.category AS category_name,
   (SUM(p.price * o.quantity) / (SELECT
         ROUND(SUM(p.price * o.quantity), 2) AS total_revenue
      FROM
         pizzas p
            JOIN
         order_details o ON p.pizza_id = o.pizza_id)) * 100 AS total_rev
FROM
   pizzas p
```

```
    JOIN
  order_details o ON p.pizza_id = o.pizza_id
    JOIN
  pizza_types pi ON pi.pizza_type_id = p.pizza_type_id
GROUP BY pi.category
ORDER BY total_rev DESC;
```

- **Explanation:**
  - This query calculates **what percentage of total revenue each pizza category contributes**. It sums the revenue for each category, divides it by the overall revenue from all pizzas, multiplies by 100, and orders the results from highest to lowest contribution.

12.Analyze the cumulative revenue generated over time.

>>> Query :

```
select t1.*,sum(total_revenue) over(order by order_date) as cum_rev from
(SELECT
  oo.order_date,SUM(p.price * o.quantity) as total_revenue
  from order_details o join pizzas p on o.pizza_id = p.pizza_id
  join orders oo on  oo.order_id = o.order_id
  group by oo.order_date)t1;
```

- **Explanation:**
  - This query calculates the **daily revenue** and then adds a **running cumulative revenue** across dates. The inner query gets total revenue for each order date, and the outer query applies a window function to compute the cumulative sum in chronological order.

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

>>> **Query:**

```
select * from
(select *, rank() over(partition by category order by revenue desc) as rn from
(SELECT
  ROUND(SUM(p.price * o.quantity), 2) AS revenue,pi.category,pi.name
FROM
  pizzas p
  JOIN

order_details o ON p.pizza_id = o.pizza_id

join pizza_types pi on pi.pizza_type_id = p.pizza_type_id

group by pi.category,pi.name) t1) t2  where rn <= 3;
```

- **Explanation:**
    - This query finds the **top 3 highest-revenue pizzas within each category**. It first calculates revenue per pizza, then uses a window `RANK()` function partitioned by category to rank pizzas by revenue. Finally, it filters the results to return only those with rank ≤ 3.