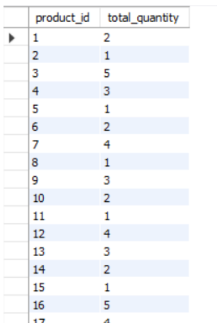
Basic Queries:

1) Write a query to retrieve the total quantity of each product ordered.

```
SELECT product_id, SUM(quantity) AS total_quantity

FROM Orders

GROUP BY product_id;
```

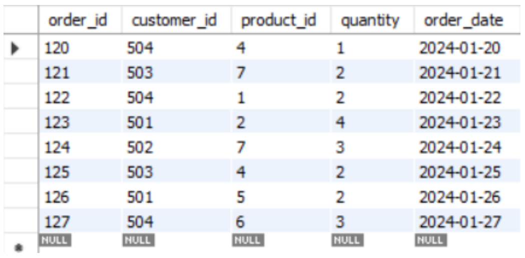


product_id	total_quantity
1	2
2	1
3	5
4	3
5	1
6	2
7	4
8	1
9	3
10	2
11	1
12	4
13	3
14	2
15	1
16	5
17	4

Inference: This query shows the total quantity ordered for each product, helping to identify the most demanded products.

2) Write a query to list all orders placed in the last 7 days.

```
SELECT *

FROM Orders

WHERE order_date >= (

   SELECT MAX(order_date)

   FROM Orders

) - 7;
```

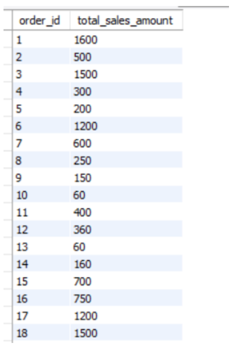


order_id	customer_id	product_id	quantity	order_date
120	504	4	1	2024-01-20
121	503	7	2	2024-01-21
122	504	1	2	2024-01-22
123	501	2	4	2024-01-23
124	502	7	3	2024-01-24
125	503	4	2	2024-01-25
126	501	5	2	2024-01-26
127	504	6	3	2024-01-27
NULL	NULL	NULL	NULL	NULL

Inference: This query retrieves all orders placed within the last 7 days by comparing the order dates with the most recent order date in the database.

3) Write a query to find the total sales amount for each order by multiplying the quantity by the product price.

```
SELECT o.order_id, o.quantity * p.price AS total_sales_amount

FROM Orders o

JOIN Products p ON o.product_id = p.product_id;
```



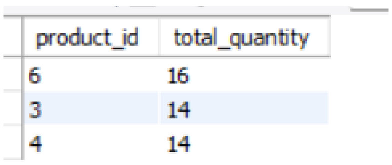
order_id	total_sales_amount
1	1600
2	500
3	1500
4	300
5	200
6	1200
7	600
8	250
9	150
10	60
11	400
12	360
13	60
14	160
15	700
16	750
17	1200
18	1500

#Inference : This query retrieves the total sales amount for each order by multiplying the quantity of the product by its price for each item in the order.

2. Intermediate Queries:

1) Write a query to find the top 3 best-selling products by total quantity.

```
SELECT product_id, SUM(quantity) AS total_quantity

FROM Orders

GROUP BY product_id

ORDER BY total_quantity DESC

LIMIT 3;
```

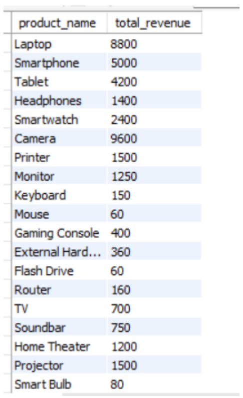


product_id	total_quantity
6	16
3	14
4	14

Inference: This query identifies the top 3 products with the highest total quantities ordered by summing up the quantities and sorting them in descending order.

2) Write a query to find the total revenue generated by each product.

```
SELECT p.product_name, SUM(o.quantity * p.price) AS total_revenue

FROM Orders o

JOIN Products p ON o.product_id = p.product_id

GROUP BY p.product_name;
```

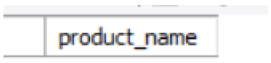


product_name	total_revenue
Laptop	8800
Smartphone	5000
Tablet	4200
Headphones	1400
Smartwatch	2400
Camera	9600
Printer	1500
Monitor	1250
Keyboard	150
Mouse	60
Gaming Console	400
External Hard...	360
Flash Drive	60
Router	160
TV	700
Soundbar	750
Home Theater	1200
Projector	1500
Smart Bulb	80

Inference: This query calculates the total revenue generated by each product by multiplying the quantity ordered by the product price and then grouping the results by product name.

3) Write a query to list the products that have never been ordered.

```
SELECT p.product_name

FROM Products p

LEFT JOIN Orders o ON p.product_id = o.product_id

WHERE o.product_id IS NULL;
```



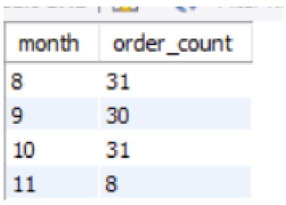
product_name

Inference: This query retrieves the names of products that have never been ordered by checking for products with no matching entries in the Orders table.

3. Date & Time Queries:

1)Write a query to count how many orders were placed in each month of 2023.

```
SELECT MONTH(order_date) AS month, COUNT(*) AS order_count

FROM Orders

WHERE YEAR(order_date) = 2023

GROUP BY MONTH(order_date);
```

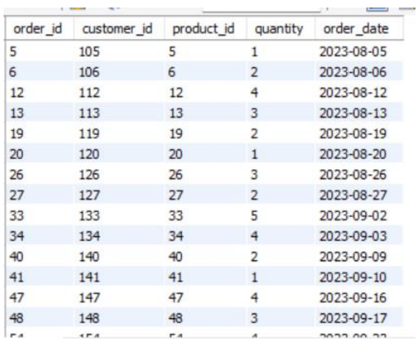


month	order_count
8	31
9	30
10	31
11	8

# Inference : This query counts the number of orders placed each month in 2023, grouping the results by month to show the order count for each.

2)Write a query to find all orders placed on weekends (Saturday and Sunday).

```
SELECT *

FROM Orders

WHERE DAYOFWEEK(order_date) IN (1, 7);
```



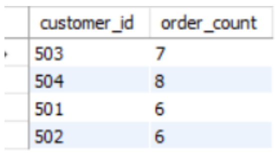
order_id	customer_id	product_id	quantity	order_date
5	105	5	1	2023-08-05
6	106	6	2	2023-08-06
12	112	12	4	2023-08-12
13	113	13	3	2023-08-13
19	119	19	2	2023-08-19
20	120	20	1	2023-08-20
26	126	26	3	2023-08-26
27	127	27	2	2023-08-27
33	133	33	5	2023-09-02
34	134	34	4	2023-09-03
40	140	40	2	2023-09-09
41	141	41	1	2023-09-10
47	147	47	4	2023-09-16
48	148	48	3	2023-09-17

#Inference : This query retrieves all orders that were placed on weekends, specifically on Sundays (1) and Saturdays (7).

4. Customer-focused Queries:

1)Write a query to find customers who have placed more than 5 orders.

```
SELECT customer_id, COUNT(order_id) AS order_count

FROM Orders

GROUP BY customer_id

HAVING COUNT(order_id) > 5;
```

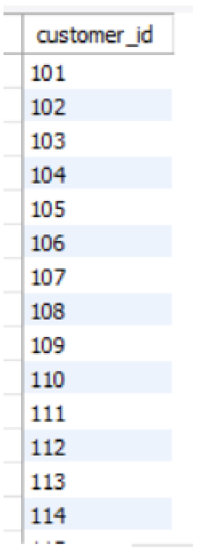


customer_id	order_count
503	7
504	8
501	6
502	6

Inference : This query lists customers who have placed more than five orders, showing their customer IDs along with the total number of orders they've made.

2)Write a query to find the customers who placed an order within the first 30 days of the year.

```
SELECT DISTINCT customer_id

FROM Orders

WHERE order_date <= (SELECT MIN(order_date) FROM Orders) + 29 ;
```



customer_id
101
102
103
104
105
106
107
108
109
110
111
112
113
114

Inference : This query identifies customers who placed an order within the first 30 days, listing their unique customer IDs.

3) Suggest how to optimize the Orders table by adding relevant indexes. Explain your choice.

```
CREATE INDEX idx_order_date ON Orders(order_date);
CREATE INDEX idx_customer_id ON Orders(customer_id);

CREATE INDEX idx_product_id ON Orders(product_id);
```

Inference: Adding indexes on frequently queried columns (like order_date, customer_id, and product_id) speeds up query performance, particularly for large datasets.