Use of DISTINCT, LIMIT, ORDER BY & Query Optimization in SQL

1. Practical Use of DISTINCT

Q1. Unique Products: Write a SQL query that will return a list of unique products ordered by customers (i.e., without duplicates)
Returns all unique product names without duplicates.

```
SELECT DISTINCT product_name FROM orders;
```

Expected Output:

product_name
Laptop
Phone
Tablet

Q2. Unique Customers: Write a SQL query to find out how many unique products were ordered by each customer. The result should show the customer's ID and the number of distinct products they've ordered
Counts how many different products each customer has ordered using COUNT with DISTINCT.

```
SELECT customer_id, COUNT(DISTINCT product_name) AS unique_product_count
FROM orders
GROUP BY customer_id;
```

Expected Output:

customer_id	unique_products
200	2
201	1
202	1
203	1

204	1

Q3. Distinct Product Count: Write a query to count the number of distinct products ordered on the platform. This should return a single number

Returns the total number of unique products ordered on the platform.

```
SELECT COUNT(DISTINCT product_name) AS total_unique_products
FROM orders;
```

Expected Output:

total_unique_products
3

Q4. Sorting by Most Recent Orders: Write a SQL query that returns the most recent distinct products ordered, sorted by the order date in descending order. Limit the result to the top 3 most recent products.

Fetches the latest unique products by sorting orders by date and limiting the output.

```
SELECT DISTINCT product_name
FROM orders
ORDER BY order_date DESC
LIMIT 3;
```

Expected Output:

product_name
Laptop
Phone
Tablet

2. Combining DISTINCT, LIMIT, and ORDER BY

Q1. Top Products in the Last Month: Write a SQL query to return the top 2 most ordered distinct products from the last month. Sort the results by order date in descending order and limit the output to the top 2.

Filters last month's orders, removes duplicates, sorts by recency, and limits the output.

```
SELECT DISTINCT product_name
FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL 1 MONTH
ORDER BY order_date DESC
LIMIT 2;
```

Expected Output:

product_name	total_orders
Laptop	3
Phone	3

Q2. Unique Products for Specific Customer: Write a SQL query to return the distinct products ordered by customer 200, sorted by the order date in descending order. Limit the result to 3 products

Returns distinct products ordered by customer 200, sorted by most recent orders.

```
SELECT DISTINCT product_name
FROM orders
WHERE customer_id = 200
ORDER BY order_date DESC
LIMIT 3;
```

Expected Output:

product_name
Tablet
Laptop

Q3. Top N Products: Write a SQL query to retrieve the top 5 most ordered products based on the number of distinct orders, sorted by product name in ascending order. Limit the result to the top 5

Uses aggregation instead of raw DISTINCT to rank products efficiently.

```
SELECT product_name, COUNT(DISTINCT order_id) AS total_orders
FROM orders
GROUP BY product_name
ORDER BY total_orders DESC
LIMIT 5;
```

Expected Output:

product_name	total_orders
Laptop	3
Phone	3
Tablet	2

Q4. Unique Orders for Each Product: Write a SQL query to count the distinct number of orders placed for each product. Sort the results by the number of distinct orders in descending order.
Counts how many unique orders exist per product and sorts by popularity.

```
SELECT product_name, COUNT(DISTINCT order_id) AS order_count
FROM orders
GROUP BY product_name
ORDER BY order_count DESC;
```

Expected Output:

product_name	order_count
Laptop	3
Phone	3
Tablet	2

3. Optimizing Queries with DISTINCT and Indexing

Q1. Optimizing Query with DISTINCT: Given a large dataset, write a SQL query to retrieve the distinct products ordered in the last month. Suggest an optimization strategy using indexes. What columns would you index to make this query faster?
This query retrieves distinct products ordered in the last month. An index on (order_date, product_name) helps reduce full-table scans and speeds up filtering and deduplication.

```
CREATE INDEX idx_orders_date_product
ON orders(order_date, product_name);

SELECT DISTINCT product_name
FROM orders
```

```
WHERE order_date >= CURRENT_DATE - INTERVAL 1 MONTH;
```

Expected Output:

product_name
Laptop
Phone
Tablet

Q2. Performance Consideration: Why is using DISTINCT on large datasets computationally expensive? What impact does it have on query performance?
DISTINCT is computationally expensive because the database must sort or hash rows to remove duplicates. On large datasets, this increases CPU usage, memory usage, and temporary disk I/O, leading to slower query execution.

Q3. Efficient Query Writing: Write a SQL query that retrieves only the top 3 distinct products ordered by customer 200. Use LIMIT and ORDER BY efficiently, and explain why the query is optimized for performance.
Filtering by customer_id first drastically reduces rows before DISTINCT and ORDER BY are applied, making the query more efficient.

```
SELECT DISTINCT product_name
FROM orders
WHERE customer_id = 200
ORDER BY order_date DESC
LIMIT 3;
```

Expected Output:

product_name
Tablet
Laptop

4. Query Optimization and Analysis with DISTINCT, LIMIT, ORDER BY

Q1. Execution Plan Analysis: Write a SQL query that returns the most popular products ordered in the last 30 s. Use DISTINCT, LIMIT, and ORDER BY to fetch the top 10 products. Use the EXPLAIN keyword to analyze the execution plan and identify potential performance issues.
EXPLAIN shows how the query is executed. Indexes on order_date and product_name reduce sorting and grouping cost.

```
EXPLAIN
SELECT product_name, COUNT(*) AS total_orders
FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL 30 DAY
GROUP BY product_name
ORDER BY total_orders DESC
LIMIT 10;
```

Expected Output:

product_name	total_orders
Laptop	3
Phone	3
Tablet	2

Q2. Optimizing Sorting and Filtering: If the database grows even further, which column(s) would you recommend indexing to improve the speed of queries involving ORDER BY, DISTINCT, and LIMIT ?
Indexing order_date improves WHERE filtering, while product_name indexing optimizes GROUP BY and ORDER BY operations. Composite indexes give best performance.

Q3. Alternative Query Approaches: Write an optimized version of a query that retrieves the top 5 most ordered products in the last 30 s. Discuss why your query is more efficient than using DISTINCT without optimization.
Using GROUP BY with MAX(order_date) avoids the overhead of DISTINCT and scales better on large datasets.

```
SELECT product_name
FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL 30 DAY
GROUP BY product_name
ORDER BY MAX(order_date) DESC
```

```
LIMIT 5;
```

Expected Output:

product_name
Tablet
Phone
Laptop

5. Real-World Scenario and Complex Query Creation

Q1. Complex Query Creation: Write a SQL query that retrieves the top 10 most recent distinct products ordered, sorted by the order date in descending order. Make sure to limit the result to 10 products.
This query retrieves the most recent order date per product and limits results efficiently.

```
SELECT product_name, MAX(order_date) AS last_order_date
FROM orders
GROUP BY product_name
ORDER BY last_order_date DESC
LIMIT 10;
```

Expected Output:

product_name	last_order_date
Tablet	2025-02-02
Phone	2025-02-01
Laptop	2025-01-20

Q2. Query Optimization: Discuss how you would optimize the query if the "orders" table had millions of rows. What indexing strategies would you apply to ensure efficient query execution?
For millions of rows, use composite indexes on (product_name, order_date, customer_id). This minimizes disk I/O and speeds up grouping and sorting.

Q3. Additional Enhancements: Imagine that the query needs to be enhanced to show the customer who made the most recent purchase for each product. How would you modify your query to include this information?

This enhanced query joins aggregated results back to the main table to fetch the customer who placed the most recent order for each product.

```
SELECT o.product_name, o.customer_id, o.order_date
FROM orders o
JOIN (
   SELECT product_name, MAX(order_date) AS max_date
   FROM orders
   GROUP BY product_name
) t
ON o.product_name = t.product_name
AND o.order_date = t.max_date
ORDER BY o.order_date DESC
LIMIT 10;
```

Expected Output:

product_name	customer_id	order_date
Tablet	206	2025-02-02
Phone	205	2025-02-01
Laptop	204	2025-01-20