Enhancing Query Proficiency

Agenda

- Window functions
- Subqueries
- Order of execution in SQL

Let’s begin the discussion by answering a few questions

Querying Data with SQL: Quiz

A company wants to rank employees within each department based on salary, ensuring that employees with the same salary receive the same rank How can they achieve this?

A. COUNT() OVER (PARTITION BY department ORDER BY salary DESC)

B. ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC)

C. RANK() OVER (PARTITION BY department ORDER BY salary DESC)

D. ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary ASC)

Querying Data with SQL: Quiz

A company wants to rank employees within each department based on salary, ensuring that employees with the same salary receive the same rank How can they achieve this?

A. COUNT() OVER (PARTITION BY department ORDER BY salary DESC)

B. ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC)

C. RANK() OVER (PARTITION BY department ORDER BY salary DESC)

D. ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary ASC)

Window Function: RANK()

sales

sale_id	employee	department	sale_amount	sale_date
1	Alice	Electronics	500	2024-02-10
2	Bob	Electronics	700	2024-02-11
3	Charlie	Electronics	700	2024-02-12
4	David	Furniture	400	2024-02-10
5	Emma	Furniture	800	2024-02-11
6	Frank	Furniture	300	2024-02-12
7	Grace	Furniture	800	2024-02-13

RANK assigns a **unique rank to each row within a partition**, allowing ties by **skipping subsequent ranks**.

```
SELECT *,
  RANK() OVER(
    PARTITION BY department
    ORDER BY sale_amount DESC) AS rank
FROM sales;
```

sale_id	employee	department	sale_amount	sale_date	rank
2	Bob	Electronics	700	2024-02-11	1
3	Charlie	Electronics	700	2024-02-12	1
1	Alice	Electronics	500	2024-02-10	3
5	Emma	Furniture	800	2024-02-11	1
7	Grace	Furniture	800	2024-02-13	1
4	David	Furniture	400	2024-02-10	3
6	Frank	Furniture	300	2024-02-12	4

Window Function: ROW_NUMBER()

sales

sale_id	employee	department	sale_amount	sale_date
1	Alice	Electronics	500	2024-02-10
2	Bob	Electronics	700	2024-02-11
3	Charlie	Electronics	700	2024-02-12
4	David	Furniture	400	2024-02-10
5	Emma	Furniture	800	2024-02-11
6	Frank	Furniture	300	2024-02-12
7	Grace	Furniture	800	2024-02-13

ROW_NUMBER assigns a **unique sequential number to each row within a partition**, **without skipping numbers**, even if there are ties.

```
SELECT *,
  ROW_NUMBER() OVER(
    PARTITION BY department
    ORDER BY sale_amount DESC) AS row
FROM sales;
```

sale_id	employee	department	sale_amount	sale_date	row
2	Bob	Electronics	700	2024-02-11	1
3	Charlie	Electronics	700	2024-02-12	2
1	Alice	Electronics	500	2024-02-10	3
5	Emma	Furniture	800	2024-02-11	1
7	Grace	Furniture	800	2024-02-13	2
4	David	Furniture	400	2024-02-10	3
6	Frank	Furniture	300	2024-02-12	4

Querying Data with SQL: Quiz

A clinic wants to list each customer's next scheduled appointment. Which function should be used?

A. LEAD(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

B. LAG(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

C. RANK(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

D. NEXT(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

Querying Data with SQL: Quiz

A clinic wants to list each customer's next scheduled appointment. Which function should be used?

A. LEAD(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

B. LAG(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

C. RANK(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

D. NEXT(appointment_date) OVER (PARTITION BY customer_id ORDER BY appointment_date)

Window Function: LEAD()

sales

sale_id	employee	department	sale_amount	sale_date
1	Alice	Electronics	500	2024-02-10
2	Bob	Electronics	700	2024-02-11
3	Charlie	Electronics	700	2024-02-12
4	David	Furniture	400	2024-02-10
5	Emma	Furniture	800	2024-02-11
6	Frank	Furniture	300	2024-02-12
7	Grace	Furniture	800	2024-02-13

LEAD returns the value of a column from a **subsequent row within the same partition**, allowing forward-looking comparisons.

```
SELECT *,
  LEAD(sale_date) OVER(
    PARTITION BY department
    ORDER BY sale_date) AS next
FROM sales;
```

sale_id	employee	department	sale_amount	sale_date	next
1	Alice	Electronics	500	2024-02-10	2024-02-11
2	Bob	Electronics	700	2024-02-11	2024-02-12
3	Charlie	Electronics	700	2024-02-12	
4	David	Furniture	400	2024-02-10	2024-02-11
5	Emma	Furniture	800	2024-02-11	2024-02-12
6	Frank	Furniture	300	2024-02-12	2024-02-13
7	Grace	Furniture	800	2024-02-13	

Window Function: LAG()

sales

sale_id	employee	department	sale_amount	sale_date
1	Alice	Electronics	500	2024-02-10
2	Bob	Electronics	700	2024-02-11
3	Charlie	Electronics	700	2024-02-12
4	David	Furniture	400	2024-02-10
5	Emma	Furniture	800	2024-02-11
6	Frank	Furniture	300	2024-02-12
7	Grace	Furniture	800	2024-02-13

LAG returns the value of a column from a **preceding row within the same partition**, enabling backward-looking comparisons.

```
SELECT *,
  LAG(sale_date) OVER(
    PARTITION BY department
    ORDER BY sale_date) AS prev
FROM sales;
```

sale_id	employee	department	sale_amount	sale_date	prev
1	Alice	Electronics	500	2024-02-10	
2	Bob	Electronics	700	2024-02-11	2024-02-10
3	Charlie	Electronics	700	2024-02-12	2024-02-11
4	David	Furniture	400	2024-02-10	
5	Emma	Furniture	800	2024-02-11	2024-02-10
6	Frank	Furniture	300	2024-02-12	2024-02-11
7	Grace	Furniture	800	2024-02-13	2024-02-12

Enhancing Query Proficiency: Quiz

A company wants to find the average salary per department and then list only the departments where the average salary is above 50,000.

A

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 50000
GROUP BY department_id;
```

B

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 50000;
```

C

```
SELECT department_id, avg_salary
FROM (
    SELECT
        department_id,
        AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id)
WHERE avg_salary > 50000;
```

D

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Enhancing Query Proficiency: Quiz

A company wants to find the average salary per department and then list only the departments where the average salary is above 50,000.

A

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 50000
GROUP BY department_id;
```

B

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 50000;
```

C

```
SELECT department_id, avg_salary
FROM (
    SELECT
        department_id,
        AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id)
WHERE avg_salary > 50000;
```

D

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Subqueries in SQL

What is a Subquery?

A subquery is a **query nested inside another SQL query**

Subqueries can be used anywhere within the main query, they are commonly used within the FROM and WHERE clauses.

FROM clause as a temporary table

WHERE clause as a filter

Subqueries in SQL

FROM clause as a temporary table:

```
-- Find the third most expensive product
SELECT PRODUCT_NAME
FROM (
  SELECT
    RANK() OVER(
      ORDER BY PRODUCT_PRICE DESC) AS RNK,
    PRODUCT_NAME
  FROM PRODUCT)
WHERE RNK = 3;
```

WHERE clause as a filter:

```
-- Display the names of products whose price is
above the average price for all products
SELECT PRODUCT_NAME
FROM PRODUCT
WHERE PRODUCT_PRICE > (
  SELECT AVG(PRODUCT_PRICE)
  FROM PRODUCT);
```

Querying Data with SQL: Quiz

Which is the correct order of execution in SQL?

A. SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY

B. FROM → GROUP BY → WHERE → HAVING → SELECT → ORDER BY

C. FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

D. SELECT → GROUP BY → HAVING → ORDER BY → FROM → WHERE

Querying Data with SQL: Quiz

Which is the correct order of execution in SQL?

A. SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY

B. FROM → GROUP BY → WHERE → HAVING → SELECT → ORDER BY

C. FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

D. SELECT → GROUP BY → HAVING → ORDER BY → FROM → WHERE

Order of Execution in SQL

FROM ----> Identifies the source tables and applies JOIN operations

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation
`GROUP BY`	Groups rows into categories for aggregation

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation
`GROUP BY`	Groups rows into categories for aggregation
`HAVING`	Filters groups based on aggregate conditions

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation
`GROUP BY`	Groups rows into categories for aggregation
`HAVING`	Filters groups based on aggregate conditions
`SELECT`	Determines which columns or expressions to return

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation
`GROUP BY`	Groups rows into categories for aggregation
`HAVING`	Filters groups based on aggregate conditions
`SELECT`	Determines which columns or expressions to return
`ORDER BY`	Sorts the result set

Order of Execution in SQL

`FROM`	Identifies the source tables and applies JOIN operations
`WHERE`	Filters rows based on conditions before aggregation
`GROUP BY`	Groups rows into categories for aggregation
`HAVING`	Filters groups based on aggregate conditions
`SELECT`	Determines which columns or expressions to return
`ORDER BY`	Sorts the result set
`LIMIT`	Restricts the number of rows returned

Great Learning

Happy Learning !