Part - 4

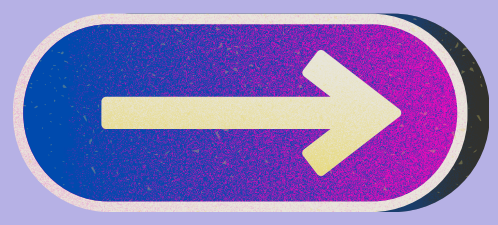
Data Analysis Interview Q & A !!!

Krishan kumar

@Krishan kumar

What are the differences between RANK, ROW_NUMBER, and DENSE_RANK in SQL?

In SQL, RANK, ROW_NUMBER, and DENSE_RANK are functions that help assign numbers to rows within a result set, but each works slightly differently.



1. RANK:

The RANK function assigns a unique rank to each distinct row within a partition of a result set. It creates gaps in the ranking sequence for rows with ties.

Syntax:

```
RANK() OVER (PARTITION BY partition_column ORDER BY order_column)
```

2. ROW_NUMBER:

The ROW_NUMBER function assigns a unique, sequential integer to each row within a partition, starting at 1 for the first row.

Syntax:

```
ROW_NUMBER() OVER (PARTITION BY partition_column ORDER BY order_column)
```

3. DENSE_RANK:

➡ **The DENSE_RANK function is similar to RANK but does not create gaps in the sequence for ties.**

➡ **Syntax:**

```
DENSE_RANK() OVER (PARTITION BY partition_column ORDER BY order_column)
```

What are the Difference among them ?

Differences:

➡ Handling Ties:

- **RANK:** **Creates gaps in the ranking sequence when rows tie. For example, if two rows share rank 1, the next rank will be 3.**

- **ROW_NUMBER:** **Does not handle ties; each row receives a unique number regardless of value.**

- **DENSE_RANK:** **No gaps in the ranking sequence for ties; if two rows are tied at rank 1, the next rank will be 2.**

I want example To understand these concepts

Example:

Consider a table Sales with columns SalesPerson and TotalSales. Here's how each function would rank the rows:

```
SELECT SalesPerson, TotalSales,
       RANK() OVER (ORDER BY TotalSales DESC) AS Rank,
       ROW_NUMBER() OVER (ORDER BY TotalSales DESC) AS RowNumber,
       DENSE_RANK() OVER (ORDER BY TotalSales DESC) AS DenseRank
FROM Sales;
```

Output:

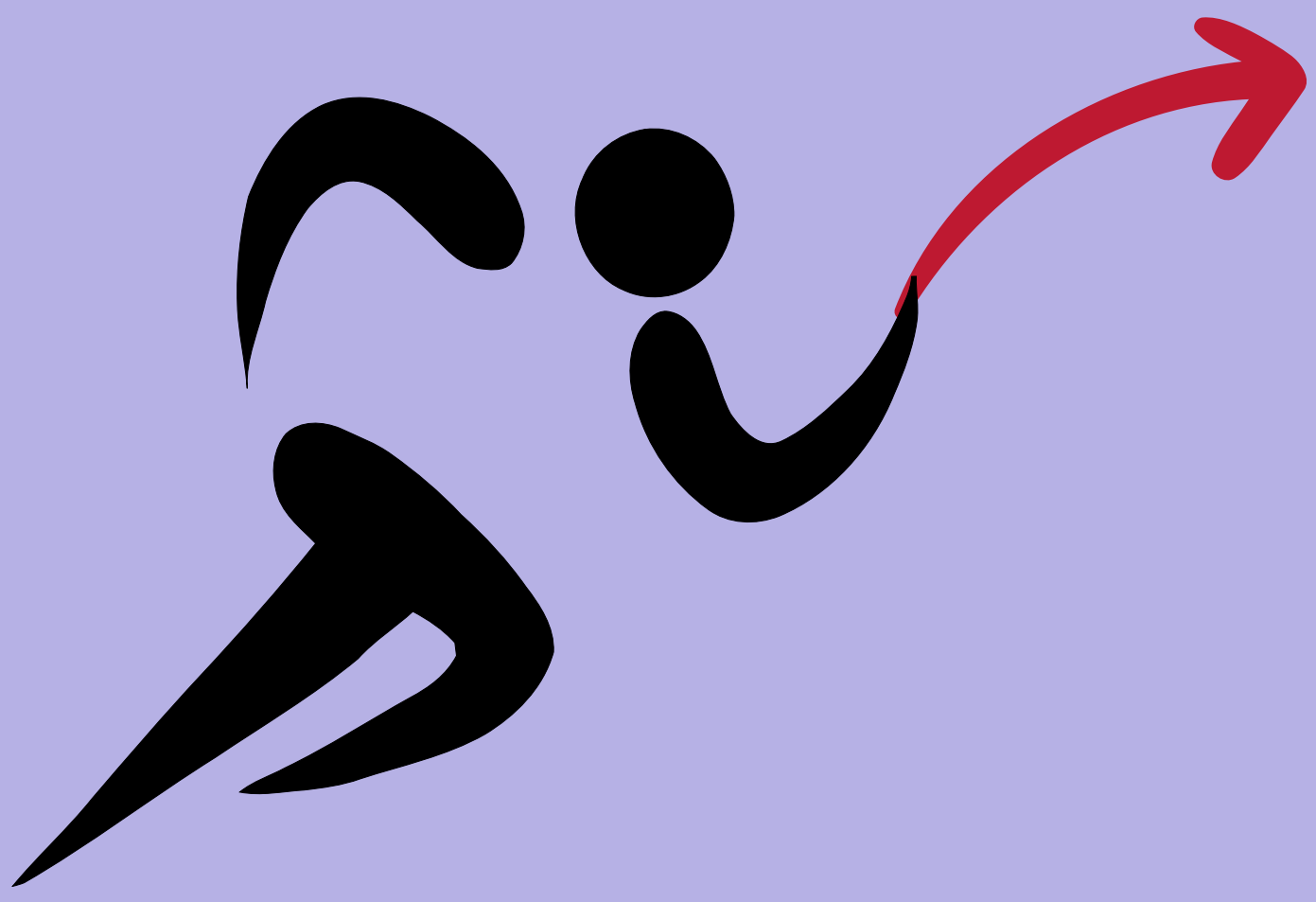
SalesPerson	TotalSales	Rank	RowNumber	DenseRank
krishna	1000	1	1	1
manish	950	2	2	2
rohan	950	2	3	2
karan	900	4	4	3

Use Cases:

- **RANK:** **Ideal for scenarios where relative standing matters, like competitions with gaps for ties.**

- **ROW_NUMBER:** **Best for generating unique row identifiers in each partition, regardless of the row values.**

- **DENSE_RANK:** **Useful when you need consecutive ranks without gaps, like prioritizing tasks in a business workflow.**



Swipe right for the jackpot of the day 😉

wanna see some

Counter Questions

1. Can you use these ranking functions without a partition?

➥ **Yes, if the PARTITION BY clause is not specified,**

The functions apply to the entire result set without partitioning.

2. When would ROW_NUMBER be more suitable than RANK or DENSE_RANK?

➡ **Use ROW_NUMBER when you need a unique identifier for each row,**

particularly in scenarios where ties are irrelevant, and you require sequential numbering.

3. Can these ranking functions improve performance?

- **While they don’t directly improve performance, these functions help in organizing and analyzing data efficiently,**

which can lead to optimized queries in certain use cases

4. Is DENSE_RANK faster than RANK?

➡ **Performance differences between DENSE_RANK and RANK are usually minimal,**

As both rely on similar partitioning and sorting operations. The choice depends more on the output structure needed.

5. Can you combine multiple ranking functions in a single query?

➡ **Yes, you can use RANK, ROW_NUMBER, and DENSE_RANK in the same query,**

often useful to compare their outputs for better insight.

you completed one interview question with me,

can you do me a favour

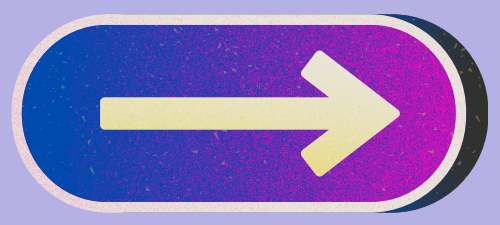
Part - 3

Data Analysis Interview Q & A !!!

Krishan kumar

@Krishan kumar

What is a CTE, and what are the benefits of using it?



CTE:

➡ **A Common Table Expression (CTE) is a temporary result set that can be referenced within a SQL statement such as**

- **SELECT,**
- **INSERT,**
- **UPDATE, or DELETE.**

CTEs make queries more readable and easier to manage by breaking them into simpler parts. They are defined using the WITH keyword.

How Do You Write a CTE?

Syntax:

➡ **The basic syntax for a CTE is as follows:**

```sql
WITH cte_name AS (
    SELECT column1, column2
    FROM table_name
    WHERE condition
)
SELECT *
FROM cte_name;
```

Example:

Let's say we need to find the average salary for employees in each department and then filter out only those departments where the average salary exceeds 5000.

```sql
WITH AvgSalaryByDept AS (
    SELECT DepartmentID, AVG(Salary) AS AvgSalary
    FROM Employees
    GROUP BY DepartmentID
)
SELECT DepartmentID, AvgSalary
FROM AvgSalaryByDept
WHERE AvgSalary > 5000;
```

What Are the Benefits of Using CTEs?

Benefits of Using CTEs

Readability and Maintainability: **CTEs make complex queries more readable by allowing you to break down lengthy nested subqueries into simpler, manageable parts.**

Avoiding Redundancy: **If a subquery is needed multiple times within a main query, using a CTE avoids writing the same logic repeatedly, making the code cleaner and easier to update.**

Recursive Queries: **CTEs support recursion, which is helpful for querying hierarchical data like organizational charts, tree structures, or family trees.**

Temporary Scope: **CTEs exist only during the execution of the query, so they do not affect the database schema.**

Performance Improvements: **Breaking a complex query into smaller CTEs can sometimes help the query planner optimize execution.**

Simplifies Nested Queries: **They reduce the need for subqueries, resulting in cleaner and more maintainable code.**

Swipe right for the jackpot of the day 😉

wanna see some

Counter Questions

1. How do CTEs compare to temporary tables?

➥ **CTEs are temporary and exist only during the query execution,**

whereas temporary tables persist until they are explicitly dropped or the session ends.

CTEs are often simpler and cleaner for modularizing complex queries.

2. What are some limitations of CTEs?

➡ **Some databases have limits on recursion depth for recursive CTEs to prevent infinite loops.**

Additionally, CTEs do not inherently optimize performance; the database's query optimizer determines the execution plan.

3. How do CTEs affect performance?

➡ **While CTEs can improve the readability and maintainability of queries,**

they do not inherently optimize performance. The performance impact depends on the database engine's query optimizer.

4. How does recursion work in CTEs?

- **Recursion in CTEs involves an initial anchor member and a recursive member that refers to the CTE itself.**

The recursion terminates when no more rows are returned by the recursive member.

5. Can a CTE be referenced multiple times in the same query?

➡ **Yes, a CTE can be referenced multiple times in a single query,**

allowing for the reuse of the defined result set without rewriting the query logic.

you completed one interview question with me,

can you do me a favour

Part - 2

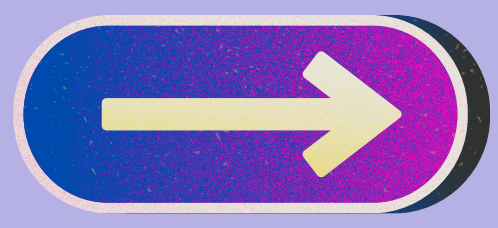
Data Analysis Interview Q & A !!!

Krishan kumar

@Krishan kumar

Differences and Similarities Between Subqueries and Joins in SQL?

Subqueries and joins are both used to combine and query data from multiple tables in SQL, but they work differently and are used in distinct scenarios



What is a Subquery?

- **A subquery is a query nested within another query and can return a single value, a list of values, or a full table.**

Syntax:

```sql
SELECT column1
FROM table1
WHERE column2 = (SELECT column2 FROM table2 WHERE condition);
```

What is a Join?

Joins combine rows from two or more tables based on related columns.

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
JOIN table2
ON table1.common_column = table2.common_column;
```

How do Subqueries and Joins Differ?

Difference between Subqueries and Joins

- **Readability:** Subqueries are easier for simple lookups, while joins are better for complex relationships.

Performance: Subqueries can be slower with large datasets; joins use indexes and optimized plans, making them more efficient.

Usage: Subqueries can appear in SELECT, WHERE, FROM, and HAVING clauses, while joins are primarily used in the FROM clause.

What are the Similarities in them ?

Similarities:

- **Purpose:** Both are used to combine data from multiple tables.

Standardization: Both follow SQL standards and are supported by most databases.

Filtering: Both can filter data based on relationships.

When Should You Use Subqueries or Joins?

When to use:

Use Subqueries: For simple lookups, modular queries, or when logic fits a nested structure.

Use Joins: For large datasets, complex relationships, or when combining multiple tables in one query.

Conclusion:
Subqueries suit simple, modular queries, while joins are ideal for complex, large-scale data relationships.

Swipe right for the surprise 🥳

Swipe right for the jackpot of the day 😉

wanna see some

Counter Questions

1. Can you use both a subquery and a join in the same SQL query?

➥ **Yes, it is possible to use both subqueries and joins together in the same query.**

For example, you can perform a join on multiple tables and use a subquery to filter the results further.

This approach can provide more control over the data retrieval process.

2. How does the performance of subqueries compare with joins?

➡ **Joins typically outperform subqueries for large datasets because they use indexed columns more efficiently, allowing the database engine to optimize execution.**

Subqueries can lead to slower performance if the inner query is executed repeatedly for each row in the outer query.

3. When should you avoid using a subquery?

➡ **Subqueries should be avoided in scenarios where they can lead to multiple evaluations of the inner query, especially with large datasets.**

It’s better to use joins in such cases as they are generally more efficient for retrieving related data across multiple tables.

4. Can subqueries be used in the HAVING clause?

➥ **Yes, subqueries can be used in the HAVING clause. This allows for complex filtering conditions on aggregated data.**
For example:

```sql
SELECT DepartmentID, COUNT(*)
FROM Employees
GROUP BY DepartmentID
HAVING COUNT(*) > (
    SELECT AVG(EmployeeCount)
    FROM (SELECT DepartmentID, COUNT(*) AS EmployeeCount FROM Employees GROUP BY
DepartmentID) AS DeptCounts
);
```

5. What are correlated subqueries, and how do they differ from regular subqueries?

➡ **Correlated subqueries refer to the outer query and execute once for each row processed by the outer query. Unlike regular subqueries, they are dependent on the outer query for their execution.**

This type of subquery can be less efficient because it may involve repeated evaluations.

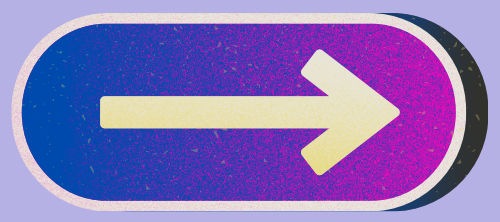
Part - 1

Data Analysis Interview Q & A !!!

Krishan kumar

@Krishan kumar

What is a subquery?



Subquery:

- **A subquery is a query nested within another SQL query. It allows you to perform more complex operations by retrieving data based on the results of another query.**

Subqueries can be particularly useful when you want to filter or aggregate data based on specific criteria defined by the inner query.

Let's understand it with an example

Example Scenario:

imagine you have two tables:

Employees :

ID	Name	DepartmentID
1	Krishna	1
2	Manisha	2
3	Radhika	1
4	Manish	3
5	Puneet	2

Departments Table:

DepartmentID	DepartmentName
1	HR
2	IT
3	Sales

Let's find the names of employees

who work in the 'IT' department.

Example of a Subquery:

Let's find the names of employees who work in the 'IT' department.

```sql
SELECT Name
FROM Employees
WHERE DepartmentID = (
            SELECT DepartmentID
            FROM Departments
            WHERE DepartmentName = 'IT'
);
```

Can you explain it with inner and outer Query🥳

Explanation:

➡️ Inner Query (Subquery):

```sql
SELECT DepartmentID
FROM Departments
WHERE DepartmentName = 'IT';
```

- **This retrieves the DepartmentID for the 'IT' department.**

➡️ Outer Query:

```sql
SELECT Name
FROM Employees
WHERE DepartmentID = (...);
```

- **This uses the result of the inner query to filter employees who belong to the 'IT' department.**

What are the benefits of Subqueries👩‍🦰

Benefits of Subqueries:

- **Modularity:** Subqueries break down complex queries into simpler, manageable parts, making them easier to understand.

- **Flexibility:** They can be used in various clauses like SELECT, WHERE, FROM, and HAVING, allowing for versatile query construction.

- **Readability:** Using subqueries can enhance the readability of SQL queries, making it easier to follow the logic of data retrieval.

Swipe right for the surprise 🥳

Swipe right for the jackpot of the day 😉

wanna see some

Counter Questions

1. What is the difference between a subquery and a JOIN?

- **A subquery is a nested query that retrieves data based on the results of another query,**

while a JOIN combines rows from two or more tables based on a related column.

Subqueries can be less efficient than JOINs for large datasets, but they can simplify complex filtering criteria.

2. Can you use a subquery in the SELECT clause?

➡ **Yes, you can use subqueries in the SELECT clause to compute values dynamically for each row returned by the outer query.**

```sql
SELECT Name,
       (SELECT DepartmentName
        FROM Departments
        WHERE DepartmentID = Employees.DepartmentID)
AS DeptName
FROM Employees;
```

3. What happens if a subquery returns more than one row?

➥ **If a subquery returns more than one row and it is used in a context where only one value is expected (like in a WHERE clause),**

it will result in an error. To handle multiple rows, you can use operators like IN instead of =:

4. Are subqueries always executed before the outer query?

➥ **Yes, subqueries are executed first, and their results are used in the outer query.**

However, SQL engines may optimize query execution plans, which can affect performance, especially with nested subqueries.

5. How do you avoid using subqueries for performance improvement?

➡ **To improve performance, you can often rewrite a subquery as a JOIN.**

This can be especially useful for large datasets, as JOINs may allow the database engine to optimize data retrieval more efficiently than subqueries.

Find This Useful

Time to hit that like button
and give it some love! 🤩

Visit my Linkedin for such amazing Content😉

in krishan kumar