SQL INTERVIEW PREPARATION PART 4

WINDOWS FUNCTIONS

Window functions perform calculations across a set of table rows that are related to the current row, but unlike aggregate functions, they do not group the rows into a single output. These are often used with the OVER() clause to define the window or subset of rows for computation.

1. ROW_NUMBER()

Assigns a unique number to each row within a partition.

Example Table: Employees

EmployeeID	Department	Salary
1	HR	5000
2	HR	4500
3	IT	7000
4	IT	7200

Query:

```
SELECT EmployeeID, Department, Salary,
ROW_NUMBER() OVER(PARTITION BY Department ORDER BY Salary DESC) AS RowNum
FROM Employees;
```

Result:

EmployeeID	Department	Salary	RowNum
1	HR	5000	1
2	HR	4500	2
4	IT	7200	1
3	IT	7000	2

2. RANK()

Assigns a rank to rows in a partition, with gaps for tied values.

Query:

```
SELECT EmployeeID, Department, Salary,
RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS Rank
FROM Employees;
```

Result:

EmployeeID	Department	Salary	Rank
1	HR	5000	1
2	HR	4500	2
4	IT	7200	1
3	IT	7000	2

3. DENSE_RANK()

Similar to RANK() but without gaps for tied values.

Query:

```
SELECT EmployeeID, Department, Salary,
DENSE_RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS DenseRank
FROM Employees;
```

Result:

EmployeeID	Department	Salary	DenseRank
1	HR	5000	1
2	HR	4500	2
4	IT	7200	1
3	IT	7000	2

4. NTILE()

Divides rows into a specified number of buckets and assigns a bucket number.

Query:

```
SELECT EmployeeID, Department, Salary,
NTILE(2) OVER(ORDER BY Salary DESC) AS Bucket
FROM Employees;
```

Result:

EmployeeID	Department	Salary	Bucket
4	IT	7200	1
3	IT	7000	1
1	HR	5000	2
2	HR	4500	2

5. LAG()

Fetches the value of a column from the previous row.

Query:

```
SELECT EmployeeID, Department, Salary,
LAG(Salary) OVER(ORDER BY Salary DESC) AS PrevSalary
FROM Employees;
```

Result:

EmployeeID	Department	Salary	PrevSalary
4	IT	7200	NULL
3	IT	7000	7200
1	HR	5000	7000
2	HR	4500	5000

6. LEAD()

Fetches the value of a column from the next row.

Query:

```
SELECT EmployeeID, Department, Salary,
LEAD(Salary) OVER(ORDER BY Salary DESC) AS NextSalary
FROM Employees;
```

Result:

EmployeeID	Department	Salary	NextSalary
4	IT	7200	7000
3	IT	7000	5000
1	HR	5000	4500
2	HR	4500	NULL

7. SUM()

Calculates a running total within a partition.

Query:

```
SELECT EmployeeID, Department, Salary,
SUM(Salary) OVER(PARTITION BY Department ORDER BY Salary) AS RunningTotal
FROM Employees;
```

Result:

EmployeeID	Department	Salary	RunningTotal
2	HR	4500	4500
1	HR	5000	9500
3	IT	7000	7000

EmployeeID	Department	Salary	RunningTotal
4	IT	7200	14200

8. AVG(), MIN(), MAX()

Calculates average, minimum, or maximum over a partition.

Query:

```
SELECT EmployeeID, Department, Salary,
AVG(Salary) OVER(PARTITION BY Department) AS AvgSalary,
MIN(Salary) OVER(PARTITION BY Department) AS MinSalary,
MAX(Salary) OVER(PARTITION BY Department) AS MaxSalary
FROM Employees;
```

Result:

EmployeeID	Department	Salary	AvgSalary	MinSalary	MaxSalary
1	HR	5000	4750	4500	5000
2	HR	4500	4750	4500	5000
3	IT	7000	7100	7000	7200
4	IT	7200	7100	7000	7200

Difference Between ROW_NUMBER(), RANK(), and DENSE_RANK()

These SQL window functions are used to assign rankings or numbers to rows within a partition of data. The key differences lie in how they handle tied values.

1. ROW_NUMBER()

Assigns a unique number to each row in a partition, even for tied values.

- **No gaps in numbering.**
- **Tied values get unique ranks.**

2. RANK()

Assigns a rank to rows within a partition.

- **Tied values get the same rank.**
- **Gaps appear in numbering for tied values.**

3. DENSE_RANK()

Assigns a rank to rows within a partition.

- **Tied values get the same rank.**
- **No gaps in numbering.**

Example Table: Sales

SaleID	Product	Amount
1	A	500
2	B	300
3	C	500
4	D	200
5	E	300

Query

```
SELECT SaleID, Product, Amount,
ROW_NUMBER() OVER(ORDER BY Amount DESC) AS RowNumber,
RANK() OVER(ORDER BY Amount DESC) AS Rank,
DENSE_RANK() OVER(ORDER BY Amount DESC) AS DenseRank
FROM Sales;
```

Result

SaleID	Product	Amount	RowNumber	Rank	DenseRank
1	A	500	1	1	1
3	C	500	2	1	1
2	B	300	3	3	2
5	E	300	4	3	2
4	D	200	5	5	3

Key Differences

1. **ROW_NUMBER()**
    - No ties; assigns a unique number to each row.
    - Example: RowNumber 3 and 4 for tied values of 300.
2. **RANK()**
    - Tied values get the same rank, but skips numbers for ties.
    - Example: Rank 3 (for 300), skips 4, and next rank is 5.
3. **DENSE_RANK()**
    - Tied values get the same rank, but no gaps in numbering.
    - Example: DenseRank 2 (for 300), next rank is 3.

Conclusion

- **Use ROW_NUMBER()** when you need unique sequential numbers.
- **Use RANK()** when you need rankings with gaps for ties.
- **Use DENSE_RANK()** when you need rankings without gaps for ties.