STRUCTURED QUERY LANGUAGE (SQL)

PART - 3

www.linkedin.com/in/sushmithadogga

4.9 Joins:

Using a JOIN clause we can combine rows from two or more tables, based on a common column between them.

The different types of joins include:

- Inner join
- Left outer join
- Right outer join
- Full outer join

4.9.1 Inner Join:

Using inner join we can pick only those records that are matching in both the tables.

```
SELECT column_name(s)

FROM tb1

INNER JOIN tb2

ON tb1.column_name = tb2.column_name;
```

Example:

```
SELECT student.ID, department.Department_name

FROM student

INNER JOIN department ON student.ID = department.Department_ID;
```

4.9.2 Left Join:

Using the left join we can pick all records from the left table (tb1), and the matching records from the right table (tb2).The LEFT JOIN keyword returns all records from the left table (student), even if there are no matches in the right table (department).

```
SELECT column_name(s)

FROM tb1

LEFT JOIN tb2

ON tb1.column_name = tb2.column_name;
```

Example:

```
SELECT student.ID, department.Department_name

FROM student

LEFT JOIN department ON student.ID = department.Department_ID;
```

4.9.3 Right Join:

Using the right join we can pick all records from the right table (tb2), and the matching records from the left table (tb1).The RIGHT JOIN keyword returns all records from the right table (department), even if there are no matches in the left table (student).

```
SELECT column_name(s)

FROM tb1

RIGHT JOIN tb2

ON tb1.column_name = tb2.column_name;
```

Example:

```
SELECT student.ID, department.Department_name

FROM student

RIGHT JOIN department ON student.ID = department.Department_ID;
```

4.9.4 Full Outer Join:

Using full outer join we can pick all the records if there is a match in left (tb1) or right (tb2) table records.

```
SELECT column_name(s)

FROM tb1

FULL OUTER JOIN tb2

ON tb1.column_name = tb2.column_name;
```

Example:

```
SELECT student.ID, department.Department_name

FROM student

FULL OUTER JOIN department ON student.ID = department.Department_ID

WHERE student.Class = "Batch A";

Without the where clause, the full outer join returns large result-sets.
```

4.10 Union:

Using the UNION operator we can combine the result-set of two or more SELECT statements.It returns only the distinct values.

```
SELECT column_name(s) FROM tb1
UNION
SELECT column_name(s) FROM tb2;
```

To filter the records we can use, UNION with WHERE :

Example:

```
SELECT ID FROM student
WHERE Class='Batch A'
UNION
SELECT Department_ID FROM department
WHERE Department_name='ECE';
```

To include duplicates as well, we use UNION ALL :

```
SELECT column_name(s)FROM tb1

UNION ALL
```

```
SELECT column_name(s)FROM tb2
```

4.11 Except:

Using the EXCEPT clause we can combine two SELECT statements and return rows only from the first SELECT statement which are not available in the second SELECT statement.

```
SELECT column_name(s)FROM tb1

EXCEPT

SELECT column_name(s)FROM tb2
```

4.12 Merge:

Merge statement is a combination of INSERT, UPDATE, DELETE . When we want to merge a **Source table** and a **Target table** , then with the help of a MERGE statement, all the three operations (INSERT, UPDATE, DELETE) can be performed at once.

```
-- FOR INSERT

WHEN NOT MATCHED BY tb1 THEN

INSERT (column_names) VALUES (tb2.column_names)

-- FOR UPDATE

WHEN MATCHED THEN UPDATE SET

tb1.column_name = tb2.column_name,

tb1.column_name = tb2.column_name

-- FOR DELETE

WHEN NOT MATCHED BY tb2 THEN

DELETE
```

4.13 Exists:

Using the EXISTS operator, we can test the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records.

```
SELECT column_name(s)FROM tb1

WHERE EXISTS

(SELECT column_name FROM tb1 WHERE condition);
```

4.14 Window functions:

Using Window functions we calculate an aggregate value over a particular window(set of rows). They return multiple rows for each group.
OVER clause is used to define the windows. It not only partitions the rows into windows using PARTITION BY but also orders the rows within these partitions using ORDER BY

Window functions include:

- Ranking Window Functions (RANK, DENSE_RANK, ROW_NUMBER)
- Aggregate Window Functions (SUM, AVG, MAX, MIN, COUNT, STDDEV, VARIANCE)
- Positional Window Functions(LAG/LEAD)

4.14.1 Ranking Window Functions :

Ranking functions return a ranking value for each row in a partition.

- **RANK():**
  Every row in every partition gets a rank from the rank function. Rows with the same value are given the same rank, and the first row is given rank 1. One rank value will be skipped for the rank that comes after two values with the same value.

```
SELECT Studentname, Subject, Marks,
RANK() OVER(PARTITION BY Studentname ORDER BY Marks DESC) Ranking
FROM student
ORDER BY Studentname, Ranking;
```

- **DENSE_RANK():**
  It assigns rank to each row within partition. The first row is assigned rank 1 and rows having the same value have the same rank. For the next rank after two same rank, consecutive integers are used, no rank is skipped.

```
SELECT Studentname, Subject, Marks,
DENSE_RANK() OVER(PARTITION BY Studentname ORDER BY Marks DESC) Ranking
FROM student
ORDER BY Ranking;
```

- **ROW_NUMBER():**
  It assigns consecutive integers to all the rows within the partition. Within a partition, no two rows can have the same row number.

```
SELECT Studentname, Subject, Marks,
ROW_NUMBER() OVER(PARTITION BY Studentname ORDER BY Marks) rownum
FROM student;
```

4.14.2 Aggregate Window Function :

Various aggregate functions like SUM(), COUNT(), AVERAGE(), MAX(), MIN() applied over a particular window (set of rows) are called aggregate window functions.

Example:

```
SELECT Name, Age, Department, Salary,
AVERAGE(Salary) OVER( PARTITION BY Department ORDER BY Age) AS
Avg_Salary
FROM employee ;
```

4.14.3 Positional Window Function :

The LAG() function allows access to a value stored in a different row above the current row. The row above may be adjacent or some number of rows above, as sorted by a specified column or set of columns.

```
SELECT seller_name, sale_value,
LAG(sale_value) OVER(ORDER BY sale_value) as previous_sale_value
FROM sale;
```

The LEAD() function allows access to a value stored in a different row below the current row. The row below may be adjacent or some number of rows below, as sorted by a specified column or set of columns.

```
SELECT seller_name, sale_value,
LEAD(sale_value) OVER(ORDER BY sale_value) as next_sale_value
FROM sale ;
```

4.15 Case :

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.If there is no ELSE part and no conditions are true, it returns NULL.

```
CASE

    WHEN condition1 THEN result1

    WHEN condition2 THEN result2

    WHEN conditionN THEN resultN

    ELSE result

END;
```