Agenda

- Group by with Rollup
- Grouping
- Window Functions
- ROW_NUMBER(), RANK(), DENSE_RANK(), LEAD(), LAG()
- Moving Window
- ~~Common Table Expression (CTE)~~
- ~~Recursive CTE~~

Group by with Roll up

- It provides the super aggegrate summary of the group operations.

```
-- display deptwise count of employees
SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno;

-- display deptwise count of employees and also display count of total employees
in organization
-- deptno      COUNT(empno)
-- 10             3
-- 20             5
-- 30             6
-- NULL          14
SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno;
SELECT COUNT(empno) FROM emp;

SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno
UNION
SELECT NULL,COUNT(empno) FROM emp;

SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno WITH ROLLUP;
```

Grouping

- GROUP BY queries that include a WITH ROLLUP modifier produces super-aggregate output rows where NULL represents the set of all values.
- The GROUPING() function enables you to distinguish NULL values for super-aggregate rows from NULL values in regular grouped rows.
- The GROUPING() function returns 1 indicating sub/grand aggregation on that column

Window Functions

- Aggregate(Group) functions operate on group of rows and generates summary (fewer rows).
- Window functions also operate on group of rows, but not reduce number of rows.
- Windowing enable dividing data into multiple partitions, sorting each partition and perform window operations on each row.

- Window functions are of two types

1. Aggegrate Functions
    - Can be used with or without windowing.
    - SUM(), AVG(), MAX(), MIN(), COUNT(),..
2. Non Aggegrate Functions
    - Can be used with windowing only
    - ROW_NUMBER(), RANK(), DENSE_RANK(), FIRST_VALUE(), LAST_VALUE(), LEAD(), LAG(),..

- windowing is done with the help of over() clause
- SELECT window_function(...) OVER(window specification), col1,col2,col3 FROM table
- In OVER(window specification) the window specification can be

1. empty -> consider the entire col as single partation
2. PARTITION BY columns
3. ORDER BY columns ASC | DESCs
4. ROWS | RANGE BETWEEN frame_start AND frame_end

Windowing on Aggegrate Functions

```
-- display all emps with empno,name,sal,total sal of all emps
SELECT ename,sal,(SELECT SUM(sal) FROM emp) AS total_sal FROM emp;

SELECT ename,sal,SUM(sal) OVER() AS total_sal FROM emp;
-- here windowing includes all the rows as a single window

-- display empno, ename, sal of each emp along with total sal of all emps in his
dept.
SELECT ename,sal,(SELECT SUM(sal) FROM emp e1 WHERE e1.deptno = e2.deptno ) AS
dept_sal FROM emp e2;

SELECT ename,sal,SUM(sal) OVER(PARTITION BY deptno) AS dept_sal FROM emp;
-- here windowing inclues deptwise groups as a window
```

Windowing on Non Aggegrate Functions

1. ROW_NUMBER()

- Assigns a sequential integer to every row within its partition
- works with Partition BY which provides numbering to every row in that partation
- ORDER BY affects the order in which rows are numbered. Without ORDER BY, row numbering is nondeterministic.

```
SELECT empno,ename,sal,deptno FROM emp;
SELECT ROW_NUMBER() OVER () AS rn,empno,ename,sal,deptno FROM emp;
SELECT ROW_NUMBER() OVER (PARTITION BY deptno) AS rn,empno,ename,sal,deptno FROM
emp;
```

```
SELECT ROW_NUMBER() OVER (ORDER BY sal) AS rn,empno,ename,sal,deptno FROM emp;
SELECT ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal) AS
rn,empno,ename,sal,deptno FROM emp;
```

2. RANK()

- Assigns a rank to every row within its partition based on the ORDER BY clause.
- It assigns the same rank to the rows with equal values.
- If two or more rows have the same rank, then there will be gaps in the sequence of ranked values.
- It is designed to provide the rank value equal to the no of rows in the partation and hence gaps are added.

```
SELECT empno,ename,sal,deptno FROM emp;
SELECT RANK() OVER () AS rnk,empno,ename,sal,deptno FROM emp;
SELECT RANK() OVER (PARTITION BY deptno) AS rnk,empno,ename,sal,deptno FROM emp;
SELECT RANK() OVER (ORDER BY sal DESC) AS rnk,empno,ename,sal,deptno FROM emp;
SELECT RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS
rnk,empno,ename,sal,deptno FROM emp;
```

3. DENSE_RANK()

- similar the the RANK()
- however if two or more rows have the same rank, then there will not be gaps in the sequence of ranked values.

```
SELECT empno,ename,sal,deptno FROM emp;
SELECT DENSE_RANK() OVER () AS dns_rnk,empno,ename,sal,deptno FROM emp;
SELECT DENSE_RANK() OVER (PARTITION BY deptno) AS dns_rnk,empno,ename,sal,deptno
FROM emp;
SELECT DENSE_RANK() OVER (ORDER BY sal DESC) AS dns_rnk,empno,ename,sal,deptno
FROM emp;
SELECT DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS
dns_rnk,empno,ename,sal,deptno FROM emp;
```

4. LEAD(), LAG()

- USed to find difference between consecutive entries

```
SELECT empno,ename,sal,LAG(sal) OVER (ORDER BY sal) AS previous,LEAD(sal)
OVER(ORDER BY sal) AS next FROM emp;

-- we can create an alis for the window that we create
SELECT empno,ename,sal,LAG(sal) OVER (wnd) AS previous,LEAD(sal) OVER(wnd) AS next
FROM emp WINDOW wnd AS (ORDER BY sal);

--display the salary difference between all the employees working in same type of
```

```
job
```

Moving Window

- ROWS | RANGE BETWEEN frame_start AND frame_end
- It is also called as window frame
- frame_start
    1. UNBOUNDED PRECEDING: The window starts inthe first row of the partition
    2. CURRENT ROW: The window starts in the current row
    3. N PRECEDING or M FOLLOWING
- frame_end
    1. UNBOUNDED FOLLOWING : The window starts inthe first row of the partition
    2. CURRENT ROW: The window starts in the current row
    3. N PRECEDING or M FOLLOWING
- By default the frame selected is RANGE UNBOUNDED PRECEDING AND CURRENT ROW
- ROWS: The frame is defined by beginning and ending row positions. Offsets are differences in row numbers from the current row number.
- RANGE: The frame is defined by rows within a value range (value given in order by). Offsets are differences in row values from the current row value.

```
DROP TABLE IF EXISTS transactions;
CREATE TABLE transactions (accid INT, txdate DATETIME, amount DOUBLE);
INSERT INTO transactions VALUES
(1, '2000-01-01', 1000),
(1, '2000-01-02', 2000),
(1, '2000-01-03', -500),
(1, '2000-01-04', -300),
(1, '2000-01-05', 4000),
(1, '2000-01-06', -2000),
(1, '2000-01-07', -200),
(2, '2000-01-02', 3000),
(2, '2000-01-04', 2000),
(2, '2000-01-06', -1000),
(3, '2000-01-01', 2000),
(3, '2000-01-03', -1000),
(3, '2000-01-05', 500);

SELECT * FROM transactions;
```