**Aim :- Implementation of Analytical queries like RollUp, Cube, First, Last, Lead,** **Lag, Rank, Dense Rank, etc.**

**Analytical Queries :-**

Analytic functions compute an aggregate value based on a group of rows. They differ from aggregate functions in that they return multiple rows for each group. The group of rows is called a window and is defined by the analytic\_clause. For each row, a sliding window of rows is defined. The window determines the range of rows used to perform the calculations for the current row. Window sizes can be based on either a physical number of rows or a logical interval such as time.

Analytic functions are the last set of operations performed in a query except for the final ORDER BY clause. All joins and all WHERE, GROUP BY, and HAVING clauses are completed before the analytic functions are processed. Therefore, analytic functions can appear only in the select list or ORDER BY clause.

Analytic functions are commonly used to compute cumulative, moving, centered, and reporting aggregates.

**ROLLUP :-**

ROLLUP enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total. ROLLUP is a simple extension to the GROUP BY clause, so its syntax is extremely easy to use. The ROLLUP extension is highly efficient, adding minimal overhead to a query.

Syntax :-

*SELECT ... GROUP BY*

*ROLLUP(grouping\_column\_reference\_list)*

**CUBE :-**

CUBE enables a SELECT statement to calculate subtotals for all possible combinations of a group of dimensions. It also calculates a grand total. This is the set of information typically needed for all cross-tabular reports, so CUBE can calculate a cross-tabular report with a single SELECT statement. Like ROLLUP, CUBE is a

simple extension to the GROUP BY clause, and its syntax is also easy to learn.

Syntax :-

*SELECT ... GROUP BY*

*CUBE (grouping\_column\_reference\_list)*

**FIRST :-**

The FIRST functions can be used to return the first value from an ordered sequence. Say we want to display the salary of each employee, along with the highest within their department we may use something like.

Syntax :-

*Function( ) KEEP (DENSE\_RANK FIRST ORDER BY <expr>) OVER (<partitioning\_clause>)*

**LAST :-**

The LAST functions can be used to return the last value from an ordered sequence. Say we want to display the salary of each employee, along with the lowest within their department we may use something like.

Syntax :-

*Function( ) KEEP (DENSE\_RANK LAST ORDER BY <expr>) OVER (<partitioning\_clause>)*

**LEAD :-**

The LEAD function is used to return data from rows further down result set.

Syntax :-

*LEAD*

*{ ( value\_expr [, offset [, default]] ) [ { RESPECT | IGNORE } NULLS ] | ( value\_expr [ { RESPECT | IGNORE } NULLS ] [, offset [, default]] )*

*}*

*OVER ([ query\_partition\_clause ] order\_by\_clause)*

**LAG :-**

The LAG function is used to access data from a previous row.

Syntax :-

*LAG*

*{ ( value\_expr [, offset [, default]]) [ { RESPECT | IGNORE } NULLS ] | ( value\_expr [ { RESPECT | IGNORE } NULLS ] [, offset [, default]] )*

*}*

*OVER ([ query\_partition\_clause ] order\_by\_clause)*

**RANK :-**

Let's assume we want to assign a sequential order, or rank, to people within a department based on salary, we might use the RANK function like this.

The basic description for the RANK analytic function is shown below. The analytic clause is described in more detail here.

Syntax :-

*RANK() OVER ([ query\_partition\_clause ] order\_by\_clause)*

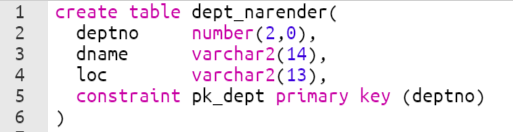
**DENSE RANK :-**

The DENSE\_RANK function acts like the RANK function except that it assigns consecutive ranks, so this is not like olympic medaling. The basic description for the DENSE\_RANK analytic function is shown below. The analytic clause is described in more detail here.

Syntax :-

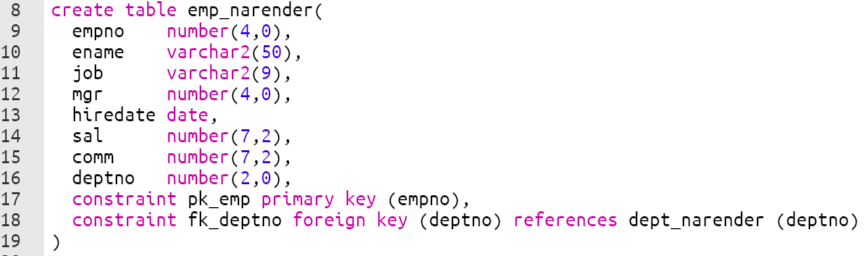
*DENSE\_RANK() OVER([ query\_partition\_clause ] order\_by\_clause)*

**Creating table departments [dept\_narender]:**



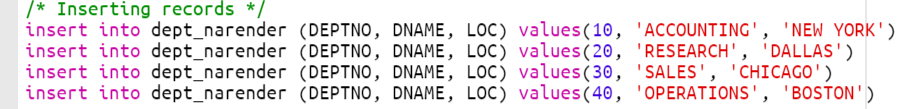


**Creating table employees [emp\_narender]:**



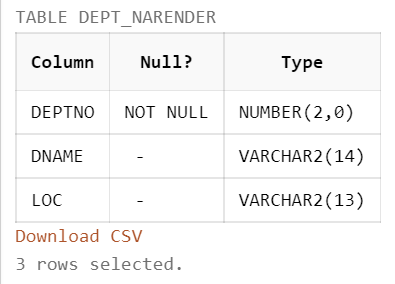


**Inserting Records in department table:**



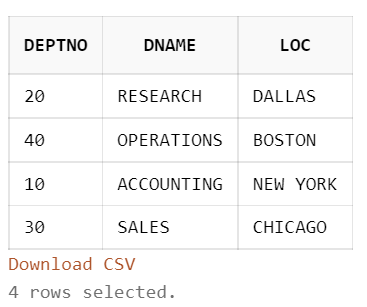
**Checking Description of department table:**



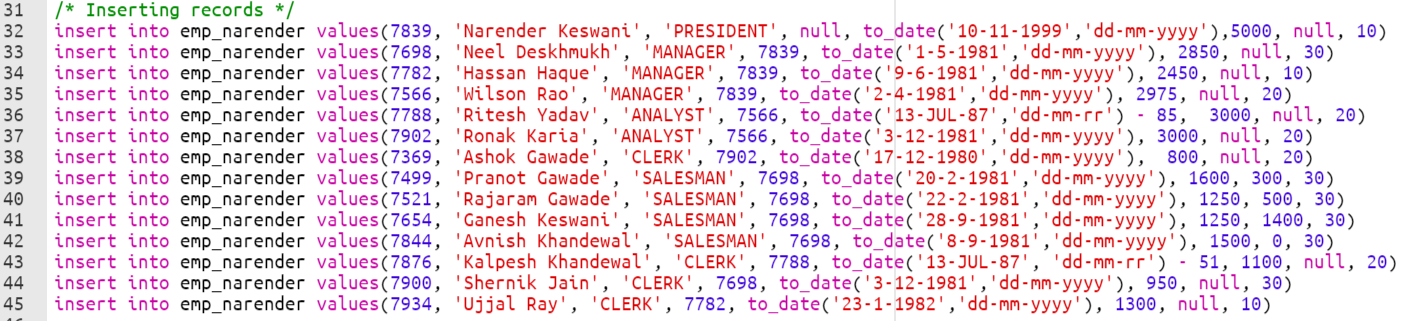


**Viewing all the records of the department table:**



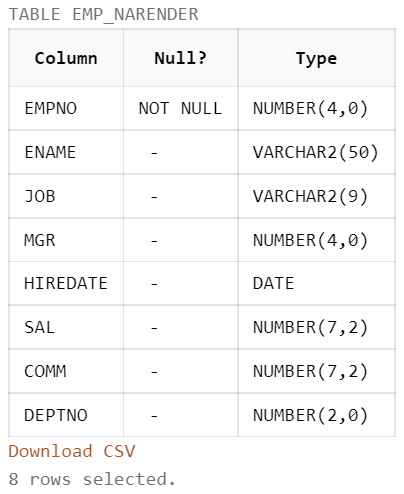


**Inserting records in employee table:**



**Viewing description of employee table:**





**Viewing all the records of the employee table:**



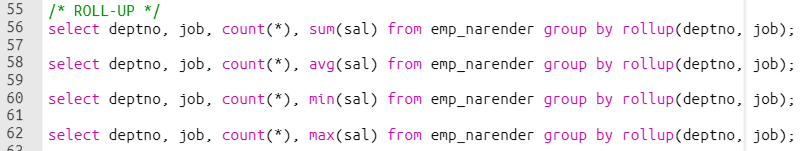


**Viewing combining records of the employee and department table based on the joins:**

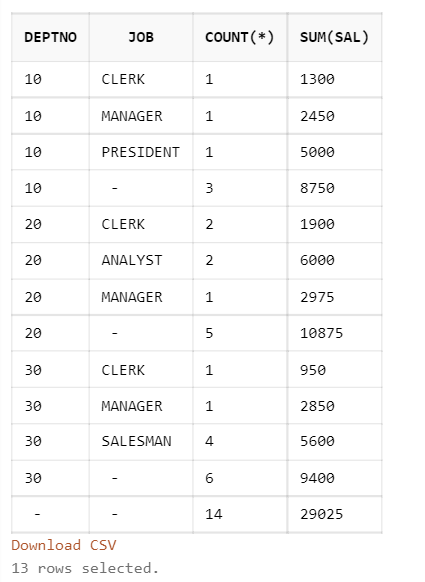




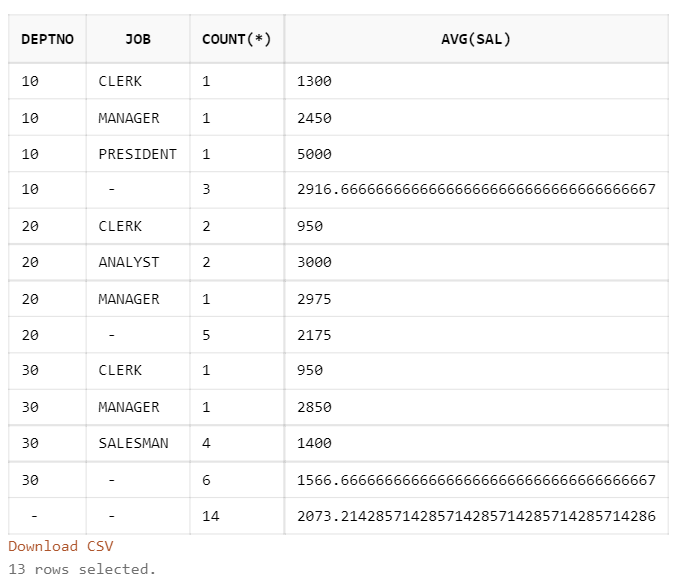
1. **ROLL -UP:**



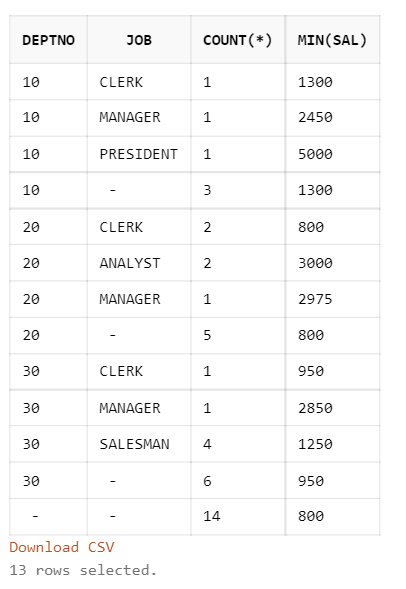
**EXAMPLE OF SUM:**



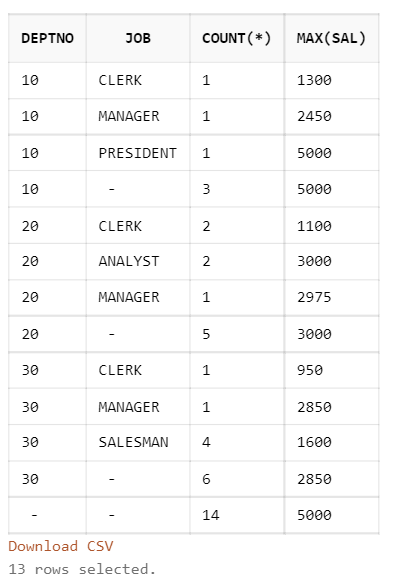
**EXAMPLE OF AVG:**



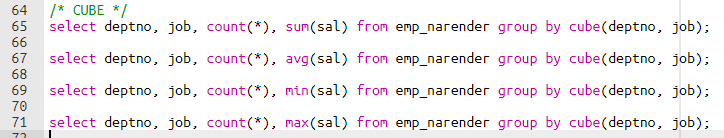
**EXAMPLE OF MIN:**

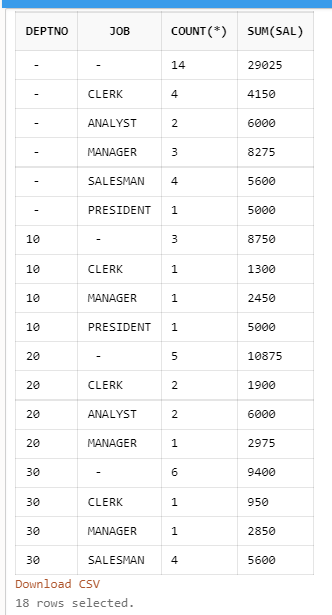


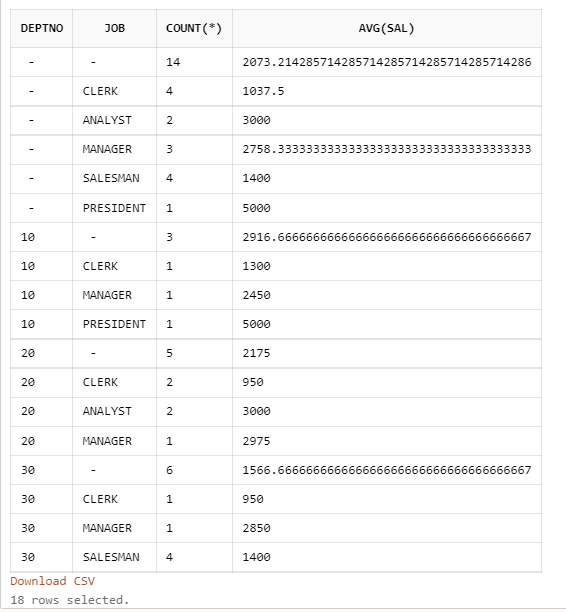
**EXAMPLE OF MAX:**

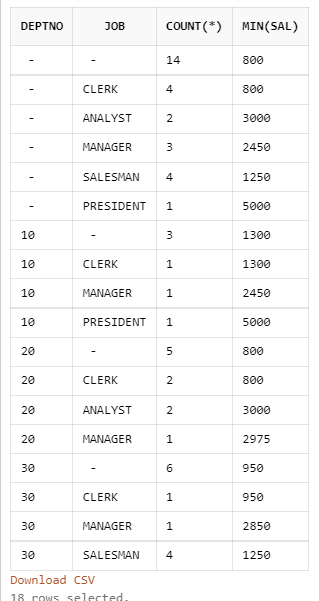


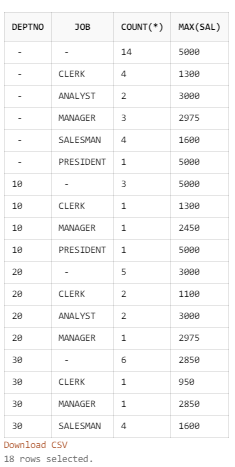
1. **CUBE:**





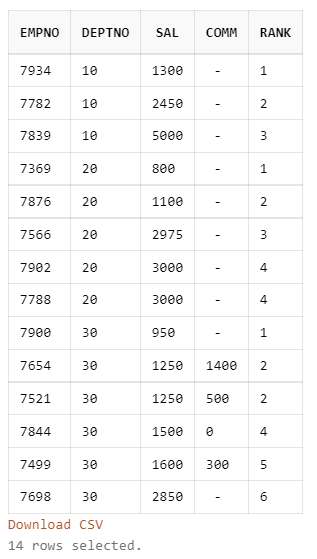






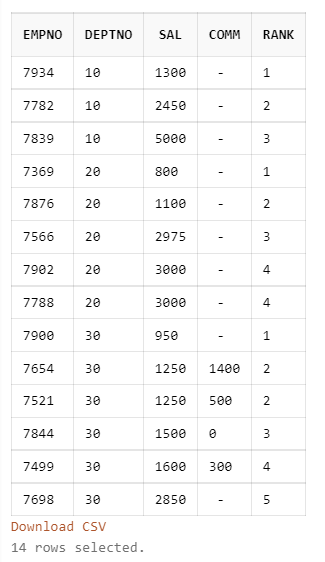
1. **RANK:**



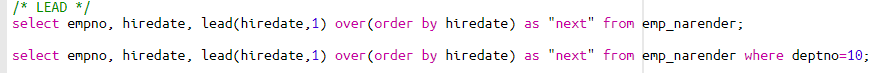


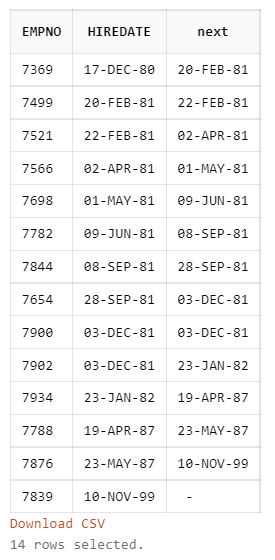
1. **DENSE-RANK:**

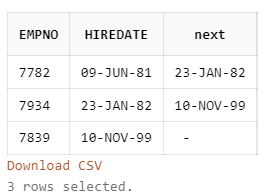




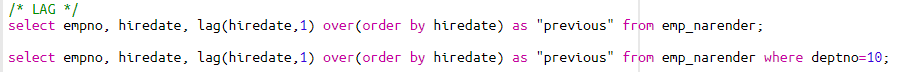
1. **LEAD:**

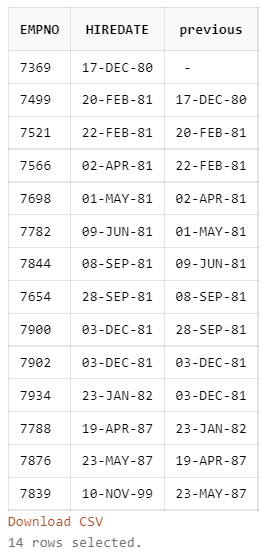


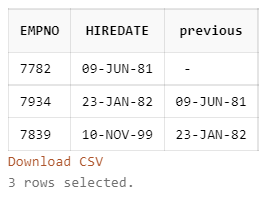




1. **LAG:**

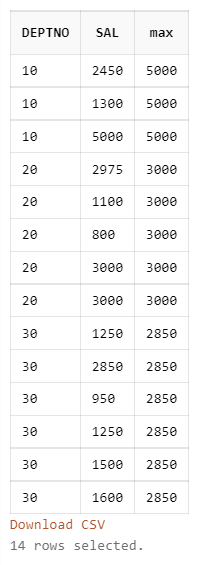






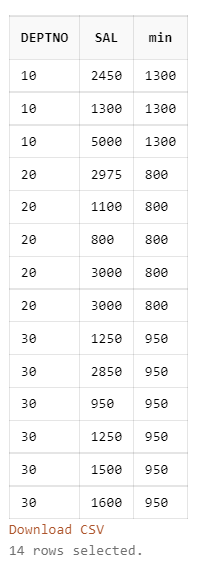
1. **FIRST:**





1. **LAST:**





**CONCLUSION:**

I have learned the implementation of Analytical queries like RollUp, Cube, First, Last, Lead, Lag, Rank, Dense Rank, etc.