Practical 2

Aim :- Implementation of Analytical queries like Roll_Up, 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)

Display the table

Code :-

```
select * from Employee;
```

Output :-

```
SQL> select * from Employee_Finny
  2  ;

    EMP_NO     DEP_NO BDATE         SALARY       COMM JOB
---------- ---------- --------- ---------- ---------- ----------------------
       108         20 08-SEP-99      71000        700 Clerk
       101         10 12-JAN-00      80000       2000 Clerk
       102         10 12-APR-01     450000       1000 Clerk
       103         10 15-DEC-99      50000        200 Clerk
       104         20 27-MAR-98      65000       3000 Manager
       105         20 16-MAR-03      55000        500 Manager
       106         20 17-JUN-91      44000        400 Manager
       107         10 21-AUG-95      58000       7000 Manager

8 rows selected.
```

RollUp :-

Code :-

```
SELECT dep_no,job,count(*),sum(salary)
from Employee_Finny
group by rollup(dep_no,job);
```

Output :-

```
SQL> SELECT dep_no,job,count(*),sum(salary)
  2  from Employee_Finny
  3  group by rollup(dep_no,job);

    DEP_NO JOB                                                 COUNT(*) SUM(SALARY)
---------- -------------------------------------------------- ---------- -----------
        10 Clerk                                                       3      580000
        10 Manager                                                     1       58000
        10                                                             4      638000
        20 Clerk                                                       1       71000
        20 Manager                                                     3      164000
        20                                                             4      235000
                                                                       8      873000

7 rows selected.
```

Cube :-

Code :- SELECT dep_no,job,count(*),sum(salary)
from Employee_Finny
group by cube(dep_no,job);

Output :-

```
SQL> SELECT dep_no,job,count(*),sum(salary)
  2  from Employee_Finny
  3  group by cube(dep_no,job);

    DEP_NO JOB                                                 COUNT(*) SUM(SALARY)
---------- --------------------------------------------------- ---------- -----------
                                                                        8      873000
           Clerk                                                        4      651000
           Manager                                                      4      222000
        10                                                              4      638000
        10 Clerk                                                        3      580000
        10 Manager                                                      1       58000
        20                                                              4      235000
        20 Clerk                                                        1       71000
        20 Manager                                                      3      164000

9 rows selected.
```

Rank :-

Code :-
select emp_no,dep_no,salary,comm,
rank() over(partition by dep_no order by salary)as Rank from Employee_Finny;

Output :-

```
SQL> select emp_no,dep_no,salary,comm,
  2  rank() over(partition by dep_no order by salary)as Rank from Employee_Finny;

    EMP_NO     DEP_NO     SALARY       COMM       RANK
---------- ---------- ---------- ---------- ----------
       103         10      50000        200          1
       107         10      58000       7000          2
       101         10      80000       2000          3
       102         10     450000       1000          4
       106         20      44000        400          1
       105         20      55000        500          2
       104         20      65000       3000          3
       108         20      71000        700          4

8 rows selected.
```

Code :-

```
update Employee_Finny set salary=22000 where emp_no=103;
select * from Employee_Finny;
```

Output :-

```
SQL>   update Employee_Finny set salary=58000 where emp_no=101;

1 row updated.

SQL> select * from Employee_Finny;

    EMP_NO     DEP_NO BDATE         SALARY       COMM JOB
---------- ---------- --------- ---------- ---------- ------------------
       108         20 08-SEP-99      71000        700 Clerk
       101         10 12-JAN-00      58000       2000 Clerk
       102         10 12-APR-01     450000       1000 Clerk
       103         10 15-DEC-99      50000        200 Clerk
       104         20 27-MAR-98      65000       3000 Manager
       105         20 16-MAR-03      55000        500 Manager
       106         20 17-JUN-91      44000        400 Manager
       107         10 21-AUG-95      58000       7000 Manager

8 rows selected.
```

Code :-

```
   select emp_no,dep_no,salary,comm,
rank() over(partition by dep_no order by salary)as Rank from Employee_Finny;
```

Output :-

```
SQL>    select emp_no,dep_no,salary,comm,
  2  rank() over(partition by dep_no order by salary)as Rank from Employee_Finny;

    EMP_NO     DEP_NO     SALARY       COMM       RANK
---------- ---------- ---------- ---------- ----------
       103         10      50000        200          1
       101         10      58000       2000          2
       107         10      58000       7000          2
       102         10     450000       1000          4
       106         20      44000        400          1
       105         20      55000        500          2
       104         20      65000       3000          3
       108         20      71000        700          4

8 rows selected.
```

Dense rank :-

Code :-

```
select emp_no,dep_no,salary,comm,
dense_rank() over(partition by dep_no order by salary)as Rank from
Employee_Finny;
```

Output :-

```
SQL> select emp_no,dep_no,salary,comm,
  2  dense_rank() over(partition by dep_no order by salary)as Rank from
  3  Employee_Finny;

    EMP_NO     DEP_NO     SALARY       COMM       RANK
---------- ---------- ---------- ---------- ----------
       103         10      50000        200          1
       101         10      58000       2000          2
       107         10      58000       7000          2
       102         10     450000       1000          3
       106         20      44000        400          1
       105         20      55000        500          2
       104         20      65000       3000          3
       108         20      71000        700          4

8 rows selected.
```

Lead :-

Code :-

```
select emp_no,bdate,
lead(bdate,1) over(order by bdate) as "next"
from Employee _Finny;
```

Output :-

```
SQL> select emp_no,bdate,
  2  lead(bdate,1) over(order by bdate) as "next"
  3  from Employee_Finny;

    EMP_NO BDATE     next
---------- --------- ---------
       106 17-JUN-91 21-AUG-95
       107 21-AUG-95 27-MAR-98
       104 27-MAR-98 08-SEP-99
       108 08-SEP-99 15-DEC-99
       103 15-DEC-99 12-JAN-00
       101 12-JAN-00 12-APR-01
       102 12-APR-01 16-MAR-03
       105 16-MAR-03

8 rows selected.
```

Code :-

```
select emp_no,bdate,
lead(bdate,1) over(order by bdate) as "next"
from Employee_Finny where dep_no=10;
```

Output :-

```
SQL> select emp_no,bdate,
  2  lead(bdate,1) over(order by bdate) as "next"
  3  from Employee_Finny where dep_no=10;

    EMP_NO BDATE     next
---------- --------- ---------
       107 21-AUG-95 15-DEC-99
       103 15-DEC-99 12-JAN-00
       101 12-JAN-00 12-APR-01
       102 12-APR-01
```

Lag :-

Code :-

```
select emp_no,bdate,
lag(bdate,1) over(order by bdate) as "Previous"
from Employee_Finny ;
```

Output :-

```
SQL> select emp_no,bdate,
  2  lag(bdate,1) over(order by bdate) as "Previous"
  3  from Employee_Finny ;

    EMP_NO BDATE     Previous
---------- --------- ---------
       106 17-JUN-91
       107 21-AUG-95 17-JUN-91
       104 27-MAR-98 21-AUG-95
       108 08-SEP-99 27-MAR-98
       103 15-DEC-99 08-SEP-99
       101 12-JAN-00 15-DEC-99
       102 12-APR-01 12-JAN-00
       105 16-MAR-03 12-APR-01

8 rows selected.
```

Code :-

```
select emp_no,bdate,
lag(bdate,1) over(order by bdate) as "Previous"
from Employee_Finny where dep_no=10;
```

Output :-

```
SQL> select emp_no,bdate,
  2  lag(bdate,1) over(order by bdate) as "Previous"
  3  from Employee_Finny where dep_no=10;

    EMP_NO BDATE     Previous
---------- --------- ---------
       107 21-AUG-95
       103 15-DEC-99 21-AUG-95
       101 12-JAN-00 15-DEC-99
       102 12-APR-01 12-JAN-00
```

First :-

Code :-

```
  select dep_no,salary,
max(salary)keep(DENSE_RANK FIRST ORDER BY salary desc)
over(PARTITION BY dep_no)"max"
from Employee_Finny;
```

Output :-

```
SQL>   select dep_no,salary,
  2  max(salary)keep(DENSE_RANK FIRST ORDER BY salary desc)
  3  over(PARTITION BY dep_no)"max"
  4  from Employee_Finny;

    DEP_NO     SALARY        max
---------- ---------- ----------
        10      58000     450000
        10      50000     450000
        10     450000     450000
        10      80000     450000
        20      44000      71000
        20      55000      71000
        20      71000      71000
        20      65000      71000

8 rows selected.
```

Last :-

Code :-

```
select dep_no,salary,
min(salary)keep(DENSE_RANK LAST ORDER BY salary desc)
over(PARTITION BY dep_no)"min"
from Employee_Finny;
```

Output :-

```
SQL>   select dep_no,salary,
  2  min(salary)keep(DENSE_RANK LAST ORDER BY salary desc)
  3  over(PARTITION BY dep_no)"min"
  4  from Employee_Finny;

    DEP_NO     SALARY        min
---------- ---------- ----------
        10      58000      50000
        10      50000      50000
        10     450000      50000
        10      80000      50000
        20      44000      44000
        20      55000      44000
        20      71000      44000
        20      65000      44000

8 rows selected.
```

Conclusion :- Successfully Implemented all analytical queries.