

Assignment No	3
Title	Analytical Queries
Objective	RollUp, Cube, rank, dense_rank, row_number, LEAD, LAG, FIRST, LAST Implementation
Roll No	MCA2511

Aim: Implementation of Analytical Queries like Roll_up, CUBE, First, Last, Lead, Lag, Row_number, Rank and Dense Rank, LEAD, LAG, FIRST, LAST.

1. Creating table: Employee table and Inserting values in the Table.

Query:

```
create table Employee (
    EmpNo number (6),
    Name varchar(10),
    Position varchar(6),
    Manager number(6),
    JoinDate date,
    Salary number(7),
    Deptno number(3));
```

```
SQL> create table Employee (
  2  EmpNo number (6),
  3  Name varchar(10),
  4  Position varchar(6),
  5  Manager number(6),
  6  JoinDate date,
  7  Salary number(7),
  8  Deptno number(3));
```

Table created.

2. Insert Value in table

Query:

```
INSERT INTO Employee
VALUES(1,'abc','xyz',201,TO_DATE('15-MAR-2006','dd-MON-yyyy'),40000,101);
INSERT INTO Employee
VALUES(2,'e2','p1',201,TO_DATE('15-MAY-2016','dd-MON-yyyy'),30000,101);
```

```
INSERT INTO Employee
VALUES(3,'e3','p2',201,TO_DATE('28-JUN-2005','dd-MON-yyyy'),20000,103);
INSERT INTO Employee
VALUES(4,'e4','p1',201,TO_DATE('5-MAR-1972','dd-MON-yyyy'),15000,102);
INSERT INTO Employee
VALUES(5,'e5','p3',201,TO_DATE('25-APR-2016','dd-MON-yyyy'),35000,103);
```

```
SQL> INSERT INTO Employee VALUES(1, 'abc', 'xyz', 201, TO_DATE('15-MAR-2006', 'dd-MON-yyyy'), 40000, 101);
1 row created.

SQL> INSERT INTO Employee VALUES(2, 'e2', 'p1', 201, TO_DATE('15-MAY-2016', 'dd-MON-yyyy'), 30000, 101);
1 row created.

SQL> INSERT INTO Employee VALUES(3, 'e3', 'p2', 201, TO_DATE('28-JUN-2005', 'dd-MON-yyyy'), 20000, 103);
1 row created.

SQL> INSERT INTO Employee VALUES(4, 'e4', 'p1', 201, TO_DATE('5-MAR-1972', 'dd-MON-yyyy'), 15000, 102);
1 row created.

SQL> INSERT INTO Employee VALUES(5, 'e5', 'p3', 201, TO_DATE('25-APR-2016', 'dd-MON-yyyy'), 35000, 103);
1 row created.
```

Display data

Query: select * from Employee;

```
SQL> select * from Employee;
```

EMPNO	NAME	POSITI	MANAGER	JOINDATE	SALARY	DEPTNO
1	abc	xyz	201	15-MAR-06	40000	101
2	e2	p1	201	15-MAY-16	30000	101
3	e3	p2	201	28-JUN-05	20000	103
4	e4	p1	201	05-MAR-72	15000	102
5	e5	p3	201	25-APR-16	35000	103

3. Display Salary without RollUp.

Query:

```
select DeptNo, sum(Salary)
as total from Employee
Group by DeptNo
Order by DeptNo;
```

```
SQL> select DeptNo, sum(Salary)
  2  as total from Employee
  3  Group by DeptNo
  4  Order by DeptNo;
```

DEPTNO	TOTAL
101	70000
102	15000
103	55000

4. Display Salary with RollUp

Query:

```
select DeptNo, sum(Salary)
as total from Employee
Group by rollup(DeptNo)
Order by DeptNo;
```

```
SQL> select DeptNo, sum(Salary)
  2  as total from Employee
  3  Group by rollup(DeptNo)
  4  Order by DeptNo;
```

DEPTNO	TOTAL
101	70000
102	15000
103	55000
	140000

Q. Difference between RollUp & without RollUp.

Ans : The ROLLUP operator adds an extra row to show the grand total of all grouped values.

Without ROLLUP, we get totals per DeptNo only.

With ROLLUP, we get totals per DeptNo and a final row showing the overall total (with DeptNo = NULL).

5. Multiple columns without RollUp:

Query:

```
select DeptNo, Position, sum(Salary)
as total from Employee
Group by DeptNo, Position
Order by 1,2;
```

```
SQL> select DeptNo, Position, sum(Salary)
  2  as total from Employee
  3  Group by DeptNo, Position
  4  Order by 1,2;
```

DEPTNO	POSITI	TOTAL
101	p1	30000
101	xyz	40000
102	p1	15000
103	p2	20000
103	p3	35000

6. Multiple columns with RollUp:

Query:

```
select DeptNo, Position, sum(Salary)
as total from Employee
group by rollup(DeptNo, Position)
order by 1,2;
```

```
SQL> INSERT INTO Employee VALUES(6, 'abc2', 'xyz', 201, TO_DATE('25-JAN-2006', 'dd-MON-yyyy'), 5000, 101);
1 row created.
```

```
SQL> select DeptNo, Position, sum(Salary)
  2  as total from Employee
  3  group by rollup(DeptNo, Position)
  4  order by 1,2;
```

DEPTNO	POSITI	TOTAL
101	p1	30000
101	xyz	45000
101		75000
102	p1	15000
102		15000
103	p2	20000
103	p3	35000
103		55000
		145000

```
9 rows selected.
```

7. Display Salary without Cube.

Query:

```
select DeptNo, sum(Salary)
as total from Employee
group by cube(DeptNo)
order by DeptNo;
```

```
SQL> select DeptNo, sum(Salary)
  2  as total from Employee
  3  group by cube(DeptNo)
  4  order by DeptNo;
```

DEPTNO	TOTAL
101	75000
102	15000
103	55000
	145000

8. Display multiple Columns without Cube:

Query:

```
select DeptNo, Position, sum(Salary)
as total from Employee
Group by DeptNo, Position
Order by 1,2;
```

```
SQL> select DeptNo, Position, sum(Salary)
  2  as total from Employee
  3  Group by DeptNo, Position
  4  Order by 1,2;
```

DEPTNO	POSITION	TOTAL
101	p1	30000
101	xyz	45000
102	p1	15000
103	p2	20000
103	p3	35000

9. Display multiple Columns with Cube:

Query:

```
select DeptNo, Position, sum(Salary)
as total from Employee
Group by cube(DeptNo, Position)
Order by 1,2;
```

```
SQL> select DeptNo, Position, sum(Salary)
  2  as total from Employee
  3  Group by cube(DeptNo, Position)
  4  Order by 1,2;
```

DEPTNO	POSITION	TOTAL
101	p1	30000
101	xyz	45000
101		75000
102	p1	15000
102		15000
103	p2	20000
103	p3	35000
103		55000
	p1	45000
	p2	20000
	p3	35000

DEPTNO	POSITION	TOTAL
	xyz	45000
		145000

13 rows selected.

Q: Difference between GROUP BY and GROUP BY CUBE?

Ans:

GROUP BY shows totals for each specific group (e.g., each DeptNo and Position).

GROUP BY CUBE shows:

- Totals for each group
- Subtotals for each column (like total per DeptNo or per Position)
- A grand total for all data

So, CUBE gives more detailed summary data than normal GROUP BY.

10. Display Salary with rank().

Query:

```
select Salary, rank()
over(order by Salary)
RANK from Employee;
```

```
SQL> select Salary, rank()
  2 over(order by Salary)
  3 RANK from Employee;
```

SALARY	RANK
5000	1
15000	2
20000	3
30000	4
35000	5
40000	6

6 rows selected.

11. Display multiple Columns with rank():

Query:

```
select EmpNo, DeptNo, Salary, JoinDate, rank()
over(order by Salary)
RANK from Employee;
```

```
SQL> select EmpNo, DeptNo, Salary, JoinDate, rank()
  2 over(order by Salary)
  3 RANK from Employee;
```

EMPNO	DEPTNO	SALARY	JOINDATE	RANK
6	101	5000	25-JAN-06	1
4	102	15000	05-MAR-72	2
3	103	20000	28-JUN-05	3
2	101	30000	15-MAY-16	4
5	103	35000	25-APR-16	5
1	101	40000	15-MAR-06	6

6 rows selected.

12. Display Salary with dense_rank().

Query:

```
select Salary, dense_rank()  
over(order by Salary)  
DENSE_RANK from Employee;
```

```
SQL> select Salary, dense_rank()  
2 over(order by Salary)  
3 DENSE_RANK from Employee;
```

SALARY	DENSE_RANK
5000	1
15000	2
20000	3
30000	4
35000	5
40000	6

SALARY	DENSE_RANK
5000	1
15000	2
20000	3
30000	4
35000	5
40000	6

6 rows selected.

13. Insert new values & display dense_rank again.

Query:

```
INSERT INTO Employee
VALUES(7,'e6','p3',201,TO_DATE('25-APR-2016','dd-MON-yyyy'),15000,101);
select Salary, dense_rank()
over(order by Salary)
DENSE_RANK from Employee;

SQL> INSERT INTO Employee VALUES(7,'e6','p3',201,TO_DATE('25-APR-2016','dd-MON-yyyy'),15000,101);
1 row created.

SQL> select Salary, dense_rank()
  2 over(order by Salary)
  3 DENSE_RANK from Employee;

  SALARY DENSE_RANK
----- -----
    5000          1
   15000         2
   15000         2
   20000         3
   30000         4
   35000         5
   40000         6

7 rows selected.

SQL> select Salary, rank()
  2 over(order by Salary)
  3 RANK from Employee;

  SALARY      RANK
----- -----
    5000          1
   15000         2
   15000         2
   20000         4
   30000         5
   35000         6
   40000         7
```

Q. Difference between rank() & dense_rank()

Ans:

RANK() skips ranks if there's a tie.

DENSE_RANK() does not skip; it gives continuous ranking.

14. Display EmpNo using row_number.

Query:

```
select EmpNo, row_number()  
over(order by Salary)  
ROW_NUMBER from Employee;
```

```
SQL> select EmpNo, row_number()  
2 over(order by Salary)  
3 ROW_NUMBER from Employee;
```

EMPNO	ROW_NUMBER
6	1
4	2
7	3
3	4
2	5
5	6
1	7

```
7 rows selected.
```

15. Display Multiple columns using LEAD().

Query:

```
SQL> select * from Employee;
```

EMPNO	NAME	POSITI	MANAGER	JOINDATE	SALARY	DEPTNO
1	abc	xyz	201	15-MAR-06	40000	101
2	e2	p1	201	15-MAY-16	30000	101
3	e3	p2	201	28-JUN-05	20000	103
4	e4	p1	201	05-MAR-72	15000	102
5	e5	p3	201	25-APR-16	35000	103
7	e6	p3	201	25-APR-16	15000	101
8	e7	p2	202	12-JAN-18	28000	102
9	e8	p1	202	03-SEP-15	32000	101
10	e9	p3	201	20-NOV-19	36000	103
11	e10	xyz	203	17-FEB-14	41000	104
12	e11	p1	202	29-MAY-11	29000	101
EMPNO	NAME	POSITI	MANAGER	JOINDATE	SALARY	DEPTNO
13	e12	p2	203	08-AUG-17	22000	102
14	e13	p3	202	19-JUL-13	37000	104
15	e14	xyz	201	06-DEC-09	43000	103

```
select DeptNo, EmpNo, Salary,
LEAD(Salary,1,0)
over(partition by DeptNo order by Salary desc)
next_low_sal from Employee
Where DeptNo in(101,102) order
by DeptNo, Salary desc;
```

```
SQL> select DeptNo, EmpNo, Salary,
  2 LEAD(Salary,1,0)
  3 over(partition by DeptNo order by Salary desc)
  4 next_low_sal from Employee
  5 Where DeptNo in(101,102) order
  6 by DeptNo, Salary desc;
```

DEPTNO	EMPNO	SALARY	NEXT_LOW_SAL
101	1	40000	32000
101	9	32000	30000
101	2	30000	29000
101	12	29000	15000
101	7	15000	0
102	8	28000	22000
102	13	22000	15000
102	4	15000	0

8 rows selected.

16. Display columns using LAG().

Query:

```
select DeptNo, EmpNo, Salary,  
LAG(Salary,1,0)  
over(partition by DeptNo order by Salary desc)  
next_high_sal from Employee  
Where DeptNo in(101,102) order  
by DeptNo, Salary desc;
```

```
SQL> select DeptNo, EmpNo, Salary,  
2 LAG(Salary,1,0)  
3 over(partition by DeptNo order by Salary desc)  
4 next_high_sal from Employee  
5 Where DeptNo in(101,102) order  
6 by DeptNo, Salary desc;
```

DEPTNO	EMPNO	SALARY	NEXT_HIGH_SAL
101	1	40000	0
101	9	32000	40000
101	2	30000	32000
101	12	29000	30000
101	7	15000	29000
102	8	28000	0
102	13	22000	28000
102	4	15000	22000

8 rows selected.

17. use of FIRST function.

Query:

```
select DeptNo, EmpNo, Salary,
min(Salary)keep(dense_rank FIRST order by Salary)
over(partition by DeptNo) as lowest from Employee
order by DeptNo, Salary;
```

```
SQL> select DeptNo, EmpNo, Salary,
2 min(Salary)keep(dense_rank FIRST order by Salary)
3 over(partition by DeptNo) as lowest from Employee
4 order by DeptNo, Salary;
```

DEPTNO	EMPNO	SALARY	LOWEST
101	7	15000	15000
101	12	29000	15000
101	2	30000	15000
101	9	32000	15000
101	1	40000	15000
102	4	15000	15000
102	13	22000	15000
102	8	28000	15000
103	3	20000	20000
103	5	35000	20000
103	10	36000	20000

DEPTNO	EMPNO	SALARY	LOWEST
103	15	43000	20000
104	14	37000	37000
104	11	41000	37000

14 rows selected.

18. use of LAST function.

Query:

```
select DeptNo, EmpNo, Salary,
min(Salary)keep(dense_rank LAST order by Salary)
over(partition by DeptNo) as lowest from Employee
order by DeptNo, Salary;
```

```
SQL> select DeptNo, EmpNo, Salary,
2 min(Salary)keep(dense_rank LAST order by Salary)
3 over(partition by DeptNo) as lowest from Employee
4 order by DeptNo, Salary;
```

DEPTNO	EMPNO	SALARY	LOWEST
101	7	15000	40000
101	12	29000	40000
101	2	30000	40000
101	9	32000	40000
101	1	40000	40000
102	4	15000	28000
102	13	22000	28000
102	8	28000	28000
103	3	20000	43000
103	5	35000	43000
103	10	36000	43000

DEPTNO	EMPNO	SALARY	LOWEST
103	15	43000	43000
104	14	37000	41000
104	11	41000	41000

14 rows selected.