SQL

TOPICS INDEX

1. **DATA & DATA TYPES**
2. **CREATE DATABASE/DROP DATABASE/USE DATA BASE**
3. **CREATING TABLE/DROP TABLE**
4. **INSERT DATA**
5. **BACKUP**
6. **UPDATE**
7. **DELETE/TRUNCATE/DROP**
8. **WHERE CONDITION, <, >, =, BETWEEN, IN, AND, OR**
9. **DISTINCT**
10. **NULL (IS NULL AND IS NOT NULL**
11. **UNION,UNION ALL,INTERSECT,EXCEPT**
12. **ORDER BY**
13. **ALIAS NAME FOR COLUMN & TABLE**
14. **IDENTITY**
15. **SET OPERATERS**
16. **CONSTRAINTS**
17. **LIKE**
18. **GROUP BY**
19. **HAVING**
20. **TOP**
21. **FUNCTIONS**
    - i) **MATHEMATICAL**
    - ii) **STRING**
    - iii) **DATE**
    - iv) **AGGRIGATE**
    - v) **RANK**
22. **JOINS**
23. **SUB QUERIES**
    - i. **NORMOL SUB QUERY**
    - ii. **CORRELATED SUB QUERY**
24. **INDEX & VIEW**
25. **SYSTEM STORED PROCEDURES (SP_ HELP & SP_HELPTEXT)**

1. DATA BASE

Syntax :

```
CREATE DATABASE database_name;
```

```
CREATE DATABASE DEMO1;
CREATE DATABASE DEMO2;
ALTER DATABASE DEMO2 MODIFY NAME=DEMO3;
DROP DATABASE DEMO3;
USE DEMO1;
```

2. CREATE TABLES

Syntax :

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,.... );
```

```
CREATE TABLE EMP ( EMPNO INT , ENAME VARCHAR(10), JOB VARCHAR(9),
MGR INT, HIREDATE DATETIME, SAL INT, COMM INT, DEPTNO INT )

SELECT*FROM EMP;

INSERT INTO EMP VALUES(7369,'SMITH','CLERK',7902,'1980-12-17',800, NULL, 20)
INSERT INTO EMP VALUES(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30)
INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30)
INSERT INTO EMP VALUES(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20)
INSERT INTO EMP VALUES(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30)
INSERT INTO EMP VALUES(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30)
INSERT INTO EMP VALUES(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10)
INSERT INTO EMP VALUES(7788,'SCOTT','ANALYST',7566,'1982-12-09',3000,NULL,20)
INSERT INTO EMP VALUES(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10)
INSERT INTO EMP VALUES(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30)
INSERT INTO EMP VALUES(7876,'ADAMS','CLERK',7788,'1983-01-12',1100, NULL,20)
INSERT INTO EMP VALUES(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30)
INSERT INTO EMP VALUES(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20)
INSERT INTO EMP VALUES(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10)

CREATE TABLE DEPT ( DEPTNO INT, DNAME VARCHAR(14), LOC VARCHAR(13) )

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK')
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS')
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO')
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON')

SELECT * FROM DEPT;
```

3. CREATE BACKUP TABLES

Syntax :

```
Select * INTO BackUP_TABLE_Name from Original_TABLE_Name;

SELECT * INTO EMP1 FROM EMP;
SELECT*INTO DEPT1 FROM DEPT;

SELECT * FROM EMP1;
SELECT*FROM DEPT1;
DROP TABLE EMP1;
```

FILTERING DATA

- DISTINCT – select distinct values in one or more columns of a table.
- WHERE – filter rows in the output of a query based on one or more conditions.
- AND – combine two Boolean expressions and return true if all expressions are true.
- OR– combine two Boolean expressions and return true if either of conditions is true.
- IN – check whether a value matches any value in a list or a subquery.
- BETWEEN – test if a value is between a ranges of values.
- LIKE – check if a character string matches a specified pattern.
- Column & Table aliases –SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

4. AND, OR, IN WITH WHERE CONDITIONS

```
--AND--

SELECT * FROM EMP1
WHERE EMPNO = 7844 AND ENAME = 'FORD';

--OR--

SELECT * FROM EMP1
WHERE EMPNO=7844 OR ENAME='FORD';

--IN--
SELECT * FROM EMP1
WHERE EMPNO IN (7369,7782,7844,4);
```

```
--CREATE BACKUP TABLES--
SELECT * INTO EMP1 FROM EMP;
SELECT*INTO DEPT1 FROM DEPT;

SELECT * FROM EMP1;
SELECT*FROM DEPT1;
```

00 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
8	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
11	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

```
--AND--

SELECT * FROM EMP1
WHERE EMPNO = 7844 AND ENAME = 'FORD';

--OR--

SELECT * FROM EMP1
WHERE EMPNO=7844 OR ENAME='FORD';

--IN--
SELECT * FROM EMP1
WHERE EMPNO IN (7369,7782,7844,4);

SELECT * FROM EMP1
WHERE COMM IS NULL;
```

100 %

Results | Messages

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
2	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
3	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

4a BETWEEN

```
--BETWEEN--
SELECT * FROM EMP1
WHERE EMPNO BETWEEN 7400 AND 8000;
```

```
--BETWEEN--
SELECT * FROM EMP1
WHERE EMPNO BETWEEN 7400 AND 8000;
```

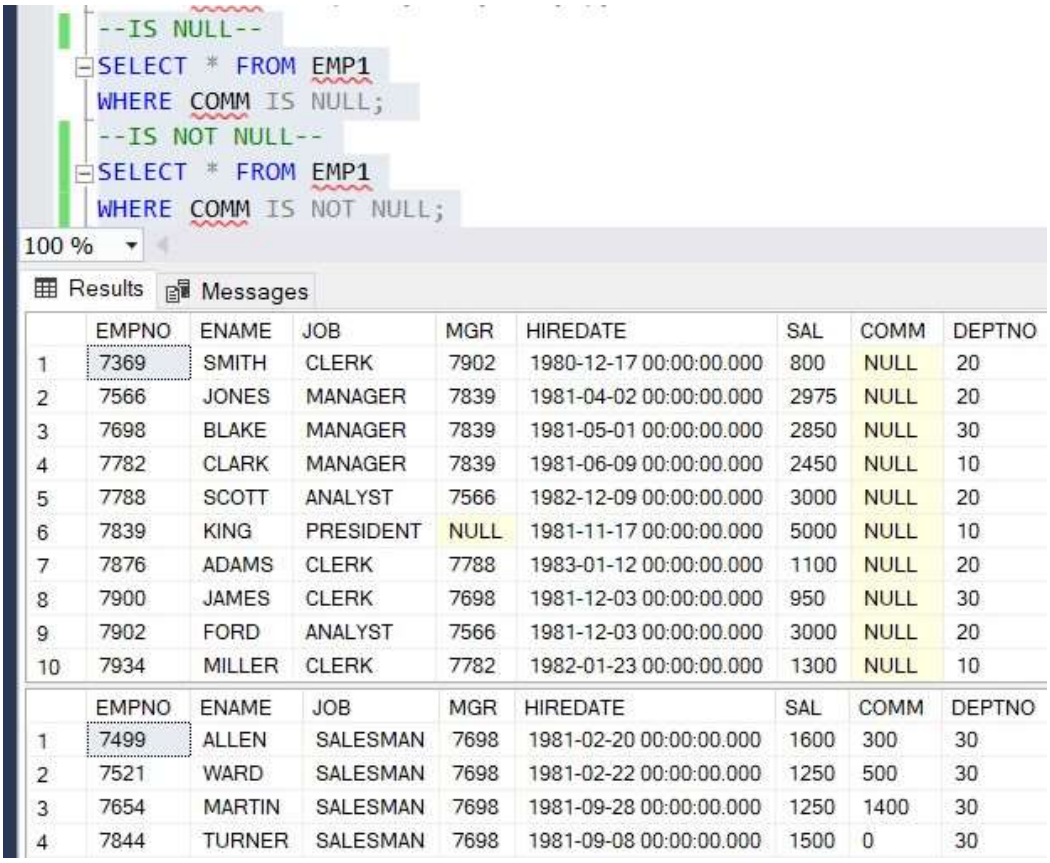
100 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
4	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
5	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
6	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
7	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
8	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
9	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
10	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
11	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
12	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
13	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

5. IS NULL & IS NOT NULL

```
--IS NULL--
SELECT * FROM EMP1
WHERE COMM IS NULL;
```



```
--IS NULL--
SELECT * FROM EMP1
WHERE COMM IS NULL;
--IS NOT NULL--
SELECT * FROM EMP1
WHERE COMM IS NOT NULL;
```

100 %

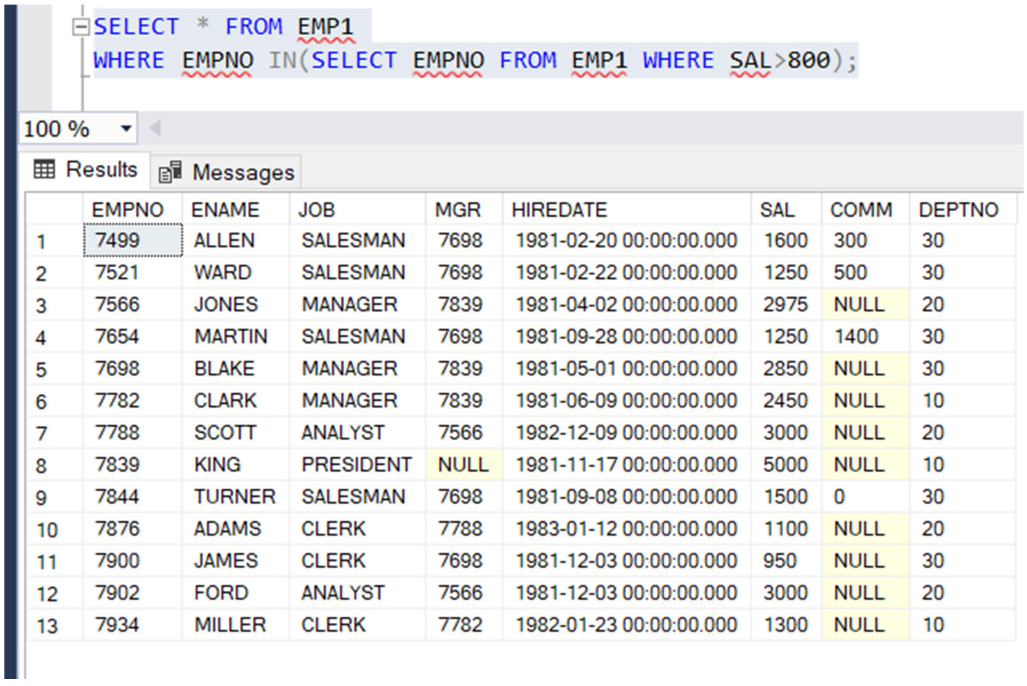
Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
3	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
4	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
5	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
6	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
7	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
8	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
9	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
10	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

5A IN WITH SUB QUERY

```
SELECT * FROM EMP1
WHERE EMPNO IN(SELECT EMPNO FROM EMP1 WHERE SAL>800);
```



```
SELECT * FROM EMP1
WHERE EMPNO IN(SELECT EMPNO FROM EMP1 WHERE SAL>800);
```

100 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
4	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
5	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
6	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
7	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
8	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
9	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
10	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
11	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
12	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
13	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

6. DISTINCT

Syntax:

```
SELECT DISTINCT column_name FROM table_name;
```

```
SELECT DISTINCT SAL,JOB FROM EMP1;
```

100 %

Results | Messages

	SAL	JOB
1	800	CLERK
2	950	CLERK
3	1100	CLERK
4	1250	SALESMAN
5	1300	CLERK
6	1500	SALESMAN
7	1600	SALESMAN
8	2450	MANAGER
9	2850	MANAGER
10	2975	MANAGER
11	3000	ANALYST
12	5000	PRESIDENT

```
SELECT DISTINCT DEPTNO FROM EMP1;
```

100 %

Results | Messages

	DEPTNO
1	10
2	20
3	30

7. ORDER BY : It will arrange the data in sequential manner (ASC/DESC)

Syntax:

```
SELECT Select_List FROM Table_Name ORDER BY Column Name [ASC/DESC];
```

```
SELECT*FROM EMP1 ORDER BY SAL DESC;
```

100 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
2	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
3	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
5	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
6	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
7	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
8	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
9	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
10	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
11	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
12	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
13	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
14	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20

```
SELECT*FROM EMP1 WHERE SAL>1500 ORDER BY SAL DESC;
```

100 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
2	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
3	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
5	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
6	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
7	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30

8. **ALIAS -**We can assign a temporary name to the column or expression while executing a SQL Server alias query. SQL Server Alias is beneficial when we want to display some specific data from a SQL Server table to label that data in a mannered way for better data representation. SQL Server Aliases are also used to increase the readability of the columns.

Syntax: Alias Column

```
SELECT column_name AS alias_name
FROM table_name;
```

Syntax: Alias Table

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

```
--ALIASING--
SELECT EMPNO AS EMPLOYYEE_ID, ENAME AS EMPLOYEE_NAME  FROM EMP1;
```

100 %

Results | Messages

	EMPLOYYEE_ID	EMPLOYEE_NAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS
12	7900	JAMES
13	7902	FORD
14	7934	MILLER

9. **DELETE**

It is used to delete particular records by giving where condition.We can even delete all records. It is little slower than the truncate.
The SQL DELETE Query is used to delete the existing records from a table.
You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax:

```
DELETE FROM table_name
WHERE [condition];
```

```
SELECT * FROM EMP1;
--DELETE--
DELETE  FROM EMP1 WHERE COMM IS NULL;
SELECT*FROM EMP1;
```

100 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
8	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
11	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

```
SELECT*FROM EMP1;

DELETE FROM EMP1
WHERE EMPNO=7844;
SELECT*FROM EMP1;
```

.00 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30

10. TRUNCATE

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself.

Syntax: `TRUNCATE TABLE  table name;`

```
TRUNCATE TABLE EMP1;
SELECT*FROM EMP1;
```

100 %

Results | Messages

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

11. DROP TABLE

The SQL DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

NOTE : You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

Syntax:
```
DROP TABLE table_name;
DROP TABLE EMP1;
```

SET OPERATORS:

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
(OR)
SELECT * FROM First
UNION
SELECT * FROM Second;
```

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example:

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

4. Except

- It combines the result of two SELECT statements. Except operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

SELECT column_name FROM table1

Except

SELECT column_name FROM table2;

Example

Except query will be:

SELECT * FROM First

Except

SELECT * FROM Second;

12. UNION :

```
SELECT * FROM STUD1;
SELECT * FROM STUD2;
SELECT * FROM STUD3;
```

STUD1

```
SELECT * FROM STUD1
UNION
SELECT * FROM STUD2
UNION
SELECT * FROM STUD3;
```

STUD2

STUD3

IT WILL COBINE 2 OR MORE
TABLES WITHOUT GIVING DUPLICATES.

RESULT

	SID	SNAME	SADD
1	1	ABC	BANG
2	2	XYZ	CHENNAI
3	3	VBN	HYD

	SID	SNAME	SADD
1	4	PQR	LUCKNOW
2	5	HJK	MYSORE
3	2	XYZ	CHENNAI

	SID	SNAME	SADD
1	1	ABC	BANG
2	4	PQR	LUCKNOW
3	6	DEF	DELHI
4	2	XYZ	CHENNAI

	SID	SNAME	SADD
1	1	ABC	BANG
2	2	XYZ	CHENNAI
3	3	VBN	HYD
4	4	PQR	LUCKN...
5	5	HJK	MYSORE
6	6	DEF	DELHI

Query executed successfully.

13. UNION ALL

```
SELECT * FROM STUD1
UNION ALL
SELECT * FROM STUD2
UNION ALL
SELECT * FROM STUD3;
```

RESULT 1

```
SELECT * FROM STUD2
UNION ALL
SELECT * FROM STUD3
UNION ALL
SELECT * FROM STUD1;
```

RESULT 2

IT WILL COBINE 2 OR
TABLES INCLUDING DUPLICATES.

100 %

Results | Messages

	SID	SNAME	SADD
2	2	XYZ	CHENNAI
3	3	VBN	HYD
4	4	PQR	LUCKNOW
5	5	HJK	MYSORE
6	2	XYZ	CHENNAI
7	1	ABC	BANG
8	4	PQR	LUCKNOW
9	6	DEF	DELHI
10	2	XYZ	CHENNAI

	SID	SNAME	SADD
1	4	PQR	LUCKNOW
2	5	HJK	MYSORE
3	2	XYZ	CHENNAI
4	1	ABC	BANG
5	4	PQR	LUCKNOW
6	6	DEF	DELHI
7	2	XYZ	CHENNAI
8	1	ABC	BANG
9	2	XYZ	CHENNAI
10	3	VBN	HYD

14. INTERSECT : IT WILL COMBINE 2 OR MORE TABLES AND RETURNS ONLY MATCHING ROWS

```
SELECT * FROM STUD1
INTERSECT
SELECT * FROM STUD2
INTERSECT
SELECT * FROM STUD3;
```

RESULT 1

```
SELECT * FROM STUD2
INTERSECT
SELECT * FROM STUD1
INTERSECT
SELECT * FROM STUD3;
```

RESULT 2

100 %

Results | Messages

	SID	SNAME	SADD
1	2	XYZ	CHENNAI

	SID	SNAME	SADD
1	2	XYZ	CHENNAI

15. EXCEPT : IT WILL RETURN ONLY

IT WILL RETURN ONLY 1 ST TABLE RECORDS
& THOSE SHOULD NOT BE IN OTHER TABLES.

```
SELECT * FROM STUD1
EXCEPT
SELECT * FROM STUD2
EXCEPT
SELECT * FROM STUD3;
```

RESULT

```
--TABLES—

SELECT * FROM STUD1
```

STUD1

```
SELECT * FROM STUD2
```

STUD2

```
SELECT * FROM STUD3;
```

STUD3

Results | Messages

	SID	SNAME	SADD
1	3	VBN	HYD

	SID	SNAME	SADD
1	1	ABC	BANG
2	2	XYZ	CHENNAI
3	3	VBN	HYD

	SID	SNAME	SADD
1	4	PQR	LUCKNOW
2	5	HJK	MYSORE
3	2	XYZ	CHENNAI

	SID	SNAME	SADD
1	1	ABC	BANG
2	4	PQR	LUCKNOW
3	6	DEF	DELHI
4	2	XYZ	CHENNAI

EXCEPT ON 2 TABLES (1-2 , 2-1)

```
SELECT * FROM STUD1
EXCEPT                          RESULT 1
SELECT * FROM STUD2;

SELECT * FROM STUD2             RESULT 2
EXCEPT
SELECT * FROM STUD1;


SELECT * FROM STUD1             STUD1

SELECT * FROM STUD2             STUD2
```

Results | Messages

	SID	SNAME	SADD
1	1	ABC	BANG
2	3	VBN	HYD

	SID	SNAME	SADD
1	4	PQR	LUCKNOW
2	5	HJK	MYSORE

	SID	SNAME	SADD
1	1	ABC	BANG
2	2	XYZ	CHENNAI
3	3	VBN	HYD

	SID	SNAME	SADD
1	4	PQR	LUCKNOW
2	5	HJK	MYSORE
3	2	XYZ	CHENNAI

16. CONSTRAINTS

SQL constraints are a set of rules implemented on tables in relational databases to dictate what data can be inserted, updated or deleted in its tables. This is done to ensure the accuracy and the reliability of information stored in the table.

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
Following are commonly used constraints available in SQL:

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

1-NOT NULL

```
CREATE TABLE CUSTOMERS1 (ID INT, NAME VARCHAR (20), SAL INT, ADDRESS VARCHAR (20), PRIMARY KEY (ID))
IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:
ALTER TABLE CUSTOMERS1
ALTER COLUMN ID INT NOT NULL;

INSERT INTO CUSTOMERS1 VALUES (1,'RAM', 1000,'HYD')
INSERT INTO CUSTOMERS1 VALUES (2,'RAJU', 2000,'BLR')
INSERT INTO CUSTOMERS1 VALUES (3,'KARTHIK', 3001,'CHENNAI'),(4,'LOKESH',2001,'BLR')
```

2-UNIQUE

```
CREATE TABLE ORDERS1 (ID INT UNIQUE, ONAME VARCHAR (20), AMOUNT INT DEFAULT 100);
IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:
ALTER TABLE ORDERS1
ADD CONSTRAINT O1_U UNIQUE (ID);
```

3-DEFAULT

```
CREATE TABLE ORDERS1 (ID INT, ONAME VARCHAR (20), AMOUNT INT DEFAULT 100);

IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:
ALTER TABLE ORDERS6
ADD CONSTRAINT DF_O6 DEFAULT 100 FOR AMOUNT;
```

4-PRIMARY KEY

```
CREATE TABLE CUSTOMERS1 (ID INT, NAME VARCHAR (20), SAL INT, ADDRESS VARCHAR (20), PRIMARY KEY (ID))
```

IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:

- BEFORE ALTERING YOU NEED TO ENSURE THAT COLUMN SHOULD BE DEFINED AS NOT NULL FOR ASIGNING PRIMARY KEY.

```
ALTER TABLE CUSTOMERS1
ALTER COLUMN ID INT NOT NULL;
ALTER TABLE CUSTOMERS1
ADD CONSTRAINT PK_O1 PRIMARY KEY(ID);
```

EX-2–GIVING NAME TO CONSTRAINT

```
CREATE TABLE CUSTOMERS2 (ID INT,NAME VARCHAR(20),SAL INT,ADDRESS VARCHAR(20), CONSTRAINT PK_CUS2
PRIMARY KEY(ID));
```

5-FOREIGN KEY

```
CREATE TABLE ORDERS1 (ID INT, ONAME VARCHAR (20), AMOUNT INT, CONSTRAINT FK_OR1 FOREIGN KEY (ID)
REFERENCES CUSTOMERS1 (ID));

IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:
CREATE TABLE ORDERS1 (ID INT, ONAME VARCHAR (20), AMOUNT INT);

ALTER TABLE ORDERS1
ADD CONSTRAINT FK_O1 FOREIGN KEY (ID) REFERENCES CUSTOMERS1 (ID);
```

6- CHECK CONSTRAINT

```
CREATE TABLE CUSTOMERS3 (ID INT NOT NULL,NAME VARCHAR(20),SAL INT CONSTRAINT CHK CHECK (SAL>1000),
ADDRESS VARCHAR(20));

IF NOT SPECIFIED AT THE TIME OF CREATION OF TABLE:
CREATE TABLE CUSTOMERS3 (ID INT NOT NULL,NAME VARCHAR(20),SAL INT,ADDRESS VARCHAR(20));
ALTER TABLE CUSTOMERS3
ADD CONSTRAINT CHK_CUS3 CHECK (SAL>1000);
```

17. LIKE

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

Syntax

The basic syntax of % and _ is as follows −

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
```

```
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

```
--LIKE
SELECT * FROM EMP1 WHERE ENAME LIKE 'B%';
SELECT * FROM EMP1 WHERE ENAME LIKE '_M%';
SELECT * FROM EMP1 WHERE ENAME LIKE '%M%';
SELECT * FROM EMP1 WHERE ENAME NOT LIKE '%M%';
SELECT * FROM EMP1 WHERE ENAME LIKE '__A%';

 SELECT * FROM EMP1 WHERE ENAME LIKE'T%';

SELECT * FROM EMP1 WHERE EMPNO LIKE '_5%';
SELECT * FROM EMP1 WHERE ENAME LIKE'%S';
SELECT * FROM EMP1 WHERE JOB LIKE 'M%';
SELECT * FROM EMP1 WHERE SAL LIKE'%50';
SELECT * FROM EMP1 WHERE COMM LIKE'%0%';

 SELECT * FROM EMP1 WHERE DEPTNO LIKE'2%';



--ONLY ONE ROW ANSWER
SELECT MAX(ENAME)AS MAX_ENAME FROM EMP1;
SELECT MIN(ENAME)AS MIN_SAL FROM EMP1;
SELECT AVG(EMPNO)AS AVG_SAL FROM EMP1;
SELECT COUNT(SAL)AS  COUNT_SAL FROM EMP1;

SELECT MAX(SAL),MIN(SAL) FROM EMP1;
SELECT MAX(SAL),DEPTNO FROM EMP1 GROUP BY DEPTNO;
SELECT COUNT(*),JOB FROM EMP1 GROUP BY JOB HAVING COUNT(*)>2;
SELECT MIN(SAL),DEPTNO FROM EMP1 GROUP BY DEPTNO HAVING MIN(SAL)<AVG(SAL);

--NOTE
--WHEN WE USE GROUP BY WHERE CONDITION WILL NOT WORK INSTEAD  HAVING WILL WORK.



EXERSCISE
1- IN A TABLE DATE OF BIRTH IS GIVEN FIND THE AGE IN YEARS
2- IN A TABLE JOINING DATE IS GIVEN FIND THE EXPERIENCE IN MONTHS

CREATE TABLE DOB(ID INT,NAME VARCHAR(20),DOB DATE);
SELECT * FROM DOB;
INSERT INTO DOB VALUES(1,'RAM','1989-02-25 00:00:00.000');
INSERT INTO DOB VALUES(2,'RAJU','2021-07-01');
INSERT INTO DOB VALUES(3,'FUTURE','2022-08-01');
SELECT GETDATE();

EXERSCISE
1- IN A TABLE DATE OF BIRTH IS GIVEN FIND THE AGE IN YEARS

SELECT *, DATEDIFF(YY,DOB,GETDATE()) AS AGE_IN_YEARS FROM DOB;
SELECT *, DATEDIFF(MM,DOB,GETDATE()) AS AGE_IN_MONTHS FROM DOB;
SELECT *, DATEDIFF(DD,DOB,GETDATE())AS AGE_IN_DAYS FROM DOB;
```

```
SELECT *, DATEDIFF(YY,DOB,GETDATE()) AS AGE_IN_YEARS FROM DOB;
SELECT *, DATEDIFF(MM,DOB,GETDATE()) AS AGE_IN_MONTHS FROM DOB;
SELECT *, DATEDIFF(DD,DOB,GETDATE())AS AGE_IN_DAYS FROM DOB;
```

99 %

Results | Messages

	ID	NAME	DOB	AGE_IN_YEARS
1	1	RAM	1989-02-25	32
2	2	RAJU	2021-07-01	0
3	3	FUTURE	2022-08-01	-1

	ID	NAME	DOB	AGE_IN_MONTHS
1	1	RAM	1989-02-25	390
2	2	RAJU	2021-07-01	1
3	3	FUTURE	2022-08-01	-12

	ID	NAME	DOB	AGE_IN_DAYS
1	1	RAM	1989-02-25	11846
2	2	RAJU	2021-07-01	32
3	3	FUTURE	2022-08-01	-364

EXERSCISE

2- IN A TABLE JOINING DATE IS GIVEN FIND THE EXPERIENCE IN MONTHS

```
SELECT *, DATEDIFF (MM, HIREDATE, GETDATE ()) AS EXPERIENCE FROM EMP1;
```

```
SELECT *, DATEDIFF(MM,HIREDATE,GETDATE()) AS EXPERIENCE_IN_MONTHS FROM EMP1;
```

99 %

Results | Messages

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	EXPERIENCE_IN_MONTHS
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20	488
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30	486
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30	486
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20	484
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30	479
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30	483
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10	482
8	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20	464
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10	477
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30	479
11	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20	463
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30	476
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20	476
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10	475

18. Group by Clause

The GROUP BY Clause is utilized in SQL with the SELECT statement to organize similar data into groups.

S. No.	Having Clause	Group By Clause
1	It is used for applying some extra condition to the query.	The Group by clause is used to group the data according to particular column or row.
2	Having cannot be used without Group by clause.	Group by can be used without having clause with the select statement.
3	The having clause can contain aggregate functions.	It cannot contain aggregate functions.
4	It restrict the query output by using some conditions	It groups the output on basis of some rows or columns.

The GROUP BY clause is a SQL command that is used to group rows that have the same values. The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
That's what it does, summarizing data from the database.
The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

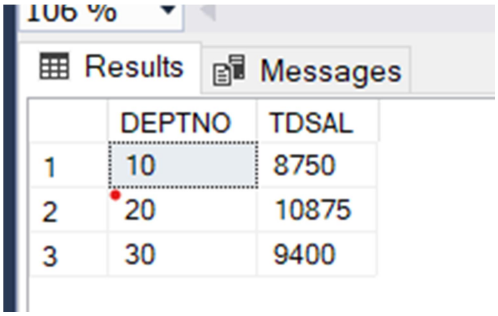
Summary;

- The GROUP BY Clause SQL is used to group rows with same values.
- The GROUP BY Clause is used together with the SQL SELECT statement.
- The SELECT statement used in the GROUP BY clause can only be used contain column names, aggregate functions, constants and expressions.
- SQL Having Clause is used to restrict the results returned by the GROUP BY clause.
- SQL GROUP BY Clause is used to collect data from multiple records and returned record set by one or more columns.

```
SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING condition];
```

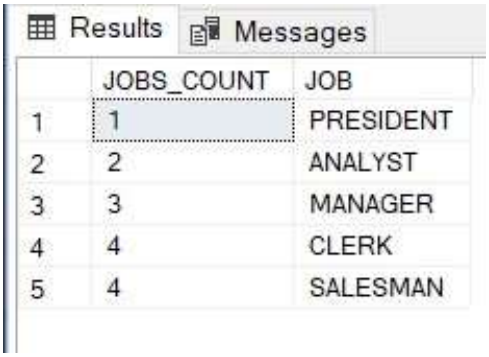
```
SELECT * FROM EMP1;
```

```
SELECT DEPTNO,SUM (SAL) AS TDSAL FROM EMP1
GROUP BY DEPTNO;
```



19. HAVING Clause

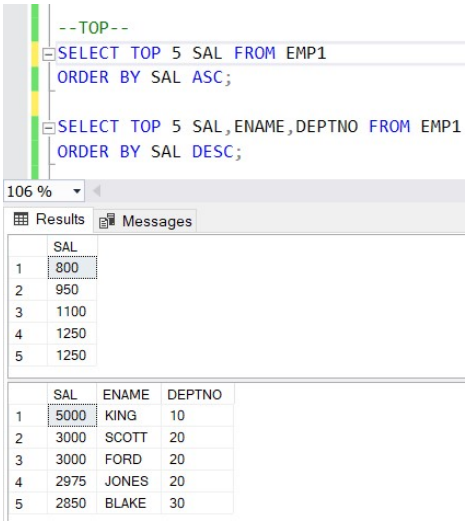
```
SELECT COUNT (*) AS JOBS_COUNT, JOB FROM EMP1
GROUP BY JOB HAVING COUNT (*)>=1
ORDER BY COUNT (*) ASC;
```



20. TOP

```
SELECT TOP 5 SAL FROM EMP1
ORDER BY SAL ASC;
```

```
SELECT TOP 5 SAL, ENAME, DEPTNO FROM EMP1
ORDER BY SAL DESC;
```



```
--TOP--
SELECT TOP 5 SAL FROM EMP1
ORDER BY SAL ASC;

SELECT TOP 5 SAL,ENAME,DEPTNO FROM EMP1
ORDER BY SAL DESC;
```

106 %

Results | Messages

	SAL
1	800
2	950
3	1100
4	1250
5	1250

	SAL	ENAME	DEPTNO
1	5000	KING	10
2	3000	SCOTT	20
3	3000	FORD	20
4	2975	JONES	20
5	2850	BLAKE	30

21. FUNCTIONS

a) **MATHEMATICAL**
b) **STRING**
c) **DATE**
d) **AGGRIGATE**
e) **RANK**

MATHEMATICAL FUNCTIONS:

```
1) This function will take input as numbers and will return output as numbers
2) ABS (NUMBER) - returns unsigned value of a given number (Number - Argument
   or Parameter.
3) SELECT ABS(-890)--O/P 890
4) SELECT ABS(0)--O/P 0
5) SELECT ABS(+52)--O/P 52
6) SELECT ABS(-890) AS A1, ABS(17) AS A2;
7) SQRT (NUMBER) - Returns square root of a given positive number.
8) SELECT SQRT(64);--O/P 8
9) SQUARE (NUMBER) - Returns square of a given value
10)   SELECT SQUARE (9);--O/P 81
11)   POWER ( NUMBER(Base), NUMBER(Exponent)) - It will find the power of a
   given number.
12)   SELECT POWER(2,3);--O/P 8
13)   SIGN ( NUMBER X )--This method returns 1 if X is positive, -1 if it is
   negative and 0 if the value of X is 0
14)   SELECT SIGN(30);---O/P 1
15)   SELECT SIGN(-30);  -- O/P -1
16)   SELECT SIGN(0);  -- O/P
17)   SELECT sign(00) O/P 0
18)   SELECT sign(-100) O/P -1
19)   SELECT sign(200) O/P +1
20)   SELECT PI() - Returns PI value
21)   SELECT SIN (NUMBER)
22)   SELECT SIN(45);
23)   SELECT ROUND ( NUMBER, NUMBER , [NUMBER])
24)   Argument1, Argument2, Argument3 (optional for TRUNCATE)
```

```
25)  By default Argument3 is ZERO
26)  SELECT ROUND(1234.5678,2,1) --- 1234.5600
27)  SELECT ROUND(1234.5678,2) ----- 1234.5700
28)  SELECT ROUND(1234.5418,2) ----- 1234.5400
29)  SELECT ROUND(1234.5638,1) ----- 1234.6000
30)  SELECT ROUND(1234.1678,1) ----- 1234.2000
31)  SELECT ROUND(1234.5678,1) ----- 1234.6000
32)  SELECT ROUND(1234.5678,0) ----- 1235.0000
33)  SELECT ROUND(1234.1678,0) ----- 1234.0000
34)  SELECT ROUND(1234.2678,0) ----- 1234.0000
35)  SELECT ROUND(1234.5678,-2)----- 1200.0000
36)  SELECT ROUND(1264.5678,-2)----- 1300.0000
37)  SELECT ROUND(5678.123,-2) ----- 5700.0000
38)  CEILING(NUMBER) - This function will increment a given number to its
   nearest integer. Based on
39)  any digit in decimal points is greater than zero.
40)  SELECT CEILING(4.1);--5
41)  SELECT CEILING (123.000) ---- 123
42)  SELECT CEILING (123.010) ---- 124
43)  SELECT CEILING (123.456) ---- 124
44)  FLOOR (NUMBER) - It decreases a nearest integer
45)  SELECT FLOOR (-123.456) ---- {-124}
46)  SELECT FLOOR(7.01);--7
47)  SELECT FLOOR(7.99);--7
```

STRING FUNCTIONS:

```
This are used to perform different operations on STRING DATA TYPE.
SELECT LEN('TEXT') - Returns number of characters.
SELECT ENAME, LEN(ENAME) AS COUNT_ENAME FROM EMP WHERE LEN(ENAME) > 5
SELECT ENAME, LEN(ENAME) AS COUNT_ENAME FROM EMP1 WHERE LEN(ENAME)<5
SELECT ENAME, LEN(ENAME) AS COUNT_ENAME FROM EMP1
SELECT LOWER('STRING') - Converts characters to Lower case.
SELECT UPPER('STRINg') - Converts characters to Upper case.
SELECT ASCII('STRING') - returns ASCII code of the given character
SELECT ASCII('A')----65
SELECT ASCII('P')----80
SELECT LEFT(STRING,NUMBER) - Extracts number of characters from left side of a
string
SELECT LEFT('COMPUTER',3) ---- COM
SELECT RIGHT( STRING, NUMBER) - Extracts number of characters from right side
of a string
SELECT RIGHT('COMPUTER',3)-----TER
SELECT SUBSTRING (STRING, STARTPOSITION, NUMBER OF CHAR'S) - It extracts
required string
from a given string based on start position and number of characters.
SELECT SUBSTRING ('COMPUTER',4,3)--PUT
SELECT SUBSTRING ('COMPUTER',2,3)--OMP

SELECT SUBSTRING('RAMESH',0,2);--R

SELECT LTRIM('STRING') - Removes blanks spaces from left side of a string.

SELECT LTRIM('   R STRING') --R STRING
SELECT RTRIM('R STRING     ')--R STRING
```

```
SELECT RTRIM('STRING')- Removes blank spaces from right side of a string.
SELECT REPLICATE('STRING', NUMBER) - Displays a given string for n times.
SELECT REPLICATE('COMM',2)--COMMCOMM
SELECT REVERSE('STRING') - It will reverse a given string.
SELECT REVERSE('COMM')--- MMOC
SELECT REPLACE (STRING1, STRING2, STRING3)
String1 ---- given string, String2 ---- search string, String3 ---- replace
string
SELECT REPLACE ('COMPUTER','UT','IR')--COMPIRER
SELECT REPLACE ('COMPUTER  UT','UT','IR')

It will search for String2 in String1 and replaces with String3 at all
occurrences of String 2 in String1
SELECT CHARINDEX(STRING1, STRING2, [NUMBER])
String1 --- Search String, String2 --- Given String, Number --- Position
It will search for string1 in string2 and returns position of string1, if it
exists else returns zero, by
default it will search from first character. It also supports to search for
character/string after a
specific position

--CHARINDEX ( expressionToFind , expressionToSearch [ , start_location ] )

SELECT CHARINDEX('R','COMPUTER',7)--8
SELECT CHARINDEX('U','COMPUTER',0)--5
SELECT CHARINDEX('U','COMPUTER',1)--5
SELECT CHARINDEX('U','COMPUTER',2)--5
SELECT CHARINDEX('U','COMPUTER',3)--5
SELECT CHARINDEX('U','COMPUTER',6)--0
SELECT CHARINDEX('U','COMPUTER');--5
SELECT CHARINDEX('R','COMPUTER',10)--0
SELECT CHARINDEX('R','COMPUTER',9)--0
SELECT CHARINDEX('R','COMPUTER',1)--8
SELECT CHARINDEX('R','COMPUTER',0)--8
SELECT REPLICATE ('A', DEPTNO) FROM EMP1;
O/P
AAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAA
```

DATE & TIME FUNCTIONS:

These functions are used to perform operations on date, time data type. When date type data is Retrieved it will be in the following format.

CENTURY: YEAR: MONTH: DATE: HOURS: MINUTES: SECONDS

SELECT GETDATE (): It returns current system DATE & TIME of the Server.

```
SELECT GETDATE()
```

ARITHMETIC OPERATIONS on data & time data type “+”, “-“ are the operations which can be used

```
for performing operations on date type data.
DATE + NUMBER = DATE
DATE - NUMBER = DATE
DATE - DATE = NUMBER [Number of days]
SELECT HIREDATE, HIREDATE+15 AS PLUS15, HIREDATE-28 AS MINUS28 FROM EMP
DAY (DATE) -------- extracts days from date
SELECT DAY ('1989-02-25')
SELECT DAY ('2025-12-09')
MONTH (DATE) ----- extracts months from date
YEAR (DATE) ------ extracts year from date
SELECT DAY(GETDATE())
SELECT MONTH(GETDATE())
SELECT YEAR(GETDATE())
DATEADD(DATEPART , NUMBER , DATE) - It is used to add number of days, months, years to a
given date.

DATEADD(IN DATEPART WHAT U NEED, NUMBER , DATE) - It is used to add number of days, months, years to
a
given date.

DATEPART can be specified with this format - DD: MM: YY: HH: MI: SS: DW
SELECT DATEADD(DAY,7,GETDATE())
SELECT DATEADD(DAY,29,'2021-1-3')
SELECT DATEADD(MI,7,'2021-08-02')

SELECT DATEADD(W,10,'1989-02-25')
SELECT DATEADD(DD,10,'1989-02-25')
SELECT DATEADD(DD,10,GETDATE())
SELECT DATEADD(MONTH,10,GETDATE())
SELECT DATEADD(MM,10,GETDATE())
SELECT DATEADD(YEARS,10,GETDATE())
SELECT DATEADD(YY,10,GETDATE())
SELECT DATEDIFF( DATEPART, STARTDATE, ENDDATE) - It returns difference between 2 dates in
terms of days, months & years.
SELECT GETDATE()
SELECT DATEPART(YY,'2021-08-01 23:30:30.573');


SELECT ENAME, DATEDIFF(DD,HIREDATE,GETDATE()) DAYS FROM EMP
SELECT DATEPART(DATEPART,DATE)- It extracts individual parts of a table.
DATEPART(DD,GETDATE()) - MM: YY: HH: MI: SS: DW
DATENAME(DATEPART,DATE) - In this function month name, day name will be extracted other
date parts providing same output.
DATENAME(MM,GETDATE()) ---- JUNE(Based on the current server date)
DATENAME(DW,GETDATE()) ---- MON
```

22. JOINS

- **Inner Join**
  - Returns rows when there is a match in both tables.
  - Inner join produces a data set that includes rows from the left table which have matching rows from the right table.

    Syntax

    The basic syntax of the INNER JOIN is as follows.

```
SELECT table1.column1, table2.column2... FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

- **Left Join**
  - Left join selects data starting from the left table and matching rows in the right table. The left join returns all rows from the left table and the matching rows from the right table. If a row in the left table does not have a matching row in the right table, the columns of the right table will have nulls.
  - The left join is also known as left outer join. The outer keyword is optional.

  Syntax

  The basic syntax of the INNER JOIN is as follows.

```
SELECT table1.column1, table2.column2... FROM table1
LEFT JOIN table2
ON table1.common_field = table2.common_field;
```

- **Right Join**
  - The right join or right outer join selects data starting from the right table. It is a reversed version of the left join.
  - The right join returns a result set that contains all rows from the right table and the matching rows in the left table. If a row in the right table that does not have a matching row in the left table, all columns in the left table will contain nulls.

  Syntax

  The basic syntax of the INNER JOIN is as follows.

```
SELECT table1.column1, table2.column2... FROM table1
LEFT JOIN table2
ON table1.common_field = table2.common_field;
```

- **Full Join**
  - The full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides where available. In case there is no match, the missing side will have NULL values.

  Syntax

  The basic syntax of a FULL JOIN is as follows

```
SELECT table1.column1, table2.column2...FROM table1
FULL JOIN table2
ON table1.common_field = table2.common_field;
```

➢ Cross Join

- The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

Syntax

The basic syntax of the CARTESIAN JOIN or the CROSS JOIN is as follows

```
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ] CROSS JOIN TABLE 2
```

➢ Self Join

- SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax

The basic syntax of SELF JOIN is as follows

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_field = b.common_field;
```

```
--Joins--
--TABLE -1--
CREATE TABLE STUDENTS_R(ID INT,FNAME VARCHAR (20),LNAME VARCHAR(20),DOB DATE,DEPT_NAME VARCHAR(20),
CONSTRAINT PK_STUR PRIMARY KEY (ID));
INSERT INTO STUDENTS_R VALUES(1,'RAM','S','1989-02-25','MECH');
INSERT INTO STUDENTS_R VALUES(2,'RAJU','P','1990-02-25','CSE');
INSERT INTO STUDENTS_R (ID,FNAME,LNAME,DEPT_NAME) VALUES (3,'VINAY','S','ECE');
INSERT INTO STUDENTS_R (ID,FNAME,LNAME,DOB) VALUES (4,'KRISHNA','R','2000-02-01');
INSERT INTO STUDENTS_R VALUES (5,'ESWAR','G','1950-01-01','CIVIL');
--TABLE -2--
CREATE TABLE DEPT_R(SID INT,DEPT_NO INT,D_NAME VARCHAR(20),COLLEGE VARCHAR(20),
CONSTRAINT PK_DEPTR PRIMARY KEY(SID));
INSERT INTO DEPT_R VALUES(1,101,'MECH','SVCET');
INSERT INTO DEPT_R VALUES(2,102,'CSE','SVCET');
INSERT INTO DEPT_R VALUES(6,103,'ECE','SVCET');
INSERT INTO DEPT_R (SID,DEPT_NO,D_NAME) VALUES(7,104,'CIVIL');

SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;
```

STUDENTS_R

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

DEPT_R

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

```
--INNER JOIN--Returns rows when there is a match in both tables.
SELECT * FROM STUDENTS_R
INNER JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

EX-1

```
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;
--INNER JOIN--Returns rows when there is a match in both tables.
SELECT * FROM STUDENTS_R
INNER JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

100 %

Results | Messages

STUDENTS_R

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

DEPT_R

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

INNER JOIN OUTPUT

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET

EX-2 BY INTER CHANGING TABLES

```
SELECT * FROM STUDENTS_R
INNER JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;

SELECT * FROM DEPT_R
INNER JOIN STUDENTS_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

100 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET

	SID	DEPT_NO	D_NAME	COLLEGE	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	101	MECH	SVCET	1	RAM	S	1989-02-25	MECH
2	2	102	CSE	SVCET	2	RAJU	P	1990-02-25	CSE

```
--LEFT JOIN--
SELECT * FROM STUDENTS_R
LEFT JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

```
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;

SELECT * FROM STUDENTS_R
LEFT JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

91 %

Results | Messages

STUDENTS_R

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

DEPT_R

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

LEFT JOIN OUTPUT

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	3	VINAY	S	NULL	ECE	NULL	NULL	NULL	NULL
4	4	KRISHNA	R	2000-02-01	NULL	NULL	NULL	NULL	NULL
5	5	ESWAR	G	1950-01-01	CIVIL	NULL	NULL	NULL	NULL

EX-2 BY INTER CHANGING TABLES

```
--LEFT JOIN--
SELECT * FROM STUDENTS_R
LEFT JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;

SELECT * FROM DEPT_R
LEFT JOIN STUDENTS_R
ON STUDENTS_R.ID = DEPT_R.SID;

SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;
```

91 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	3	VINAY	S	NULL	ECE	NULL	NULL	NULL	NULL
4	4	KRISHNA	R	2000-02-01	NULL	NULL	NULL	NULL	NULL
5	5	ESWAR	G	1950-01-01	CIVIL	NULL	NULL	NULL	NULL

	SID	DEPT_NO	D_NAME	COLLEGE	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	101	MECH	SVCET	1	RAM	S	1989-02-25	MECH
2	2	102	CSE	SVCET	2	RAJU	P	1990-02-25	CSE
3	6	103	ECE	SVCET	NULL	NULL	NULL	NULL	NULL
4	7	104	CIVIL	NULL	NULL	NULL	NULL	NULL	NULL

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

EX-3 ONLY SELECTED COLUMNS INSTEAD OF ALL, BY INTER CHANGING TABLES

```
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;

SELECT STUDENTS_R.ID,STUDENTS_R.FNAME,DEPT_R.SID,DEPT_R.DEPT_NO FROM STUDENTS_R
LEFT JOIN DEPT_R
ON STUDENTS_R.ID=DEPT_R.SID;

SELECT DEPT_R.SID,DEPT_R.DEPT_NO,STUDENTS_R.ID,STUDENTS_R.FNAME FROM DEPT_R
LEFT JOIN STUDENTS_R
ON STUDENTS_R.ID=DEPT_R.SID;
```

91 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

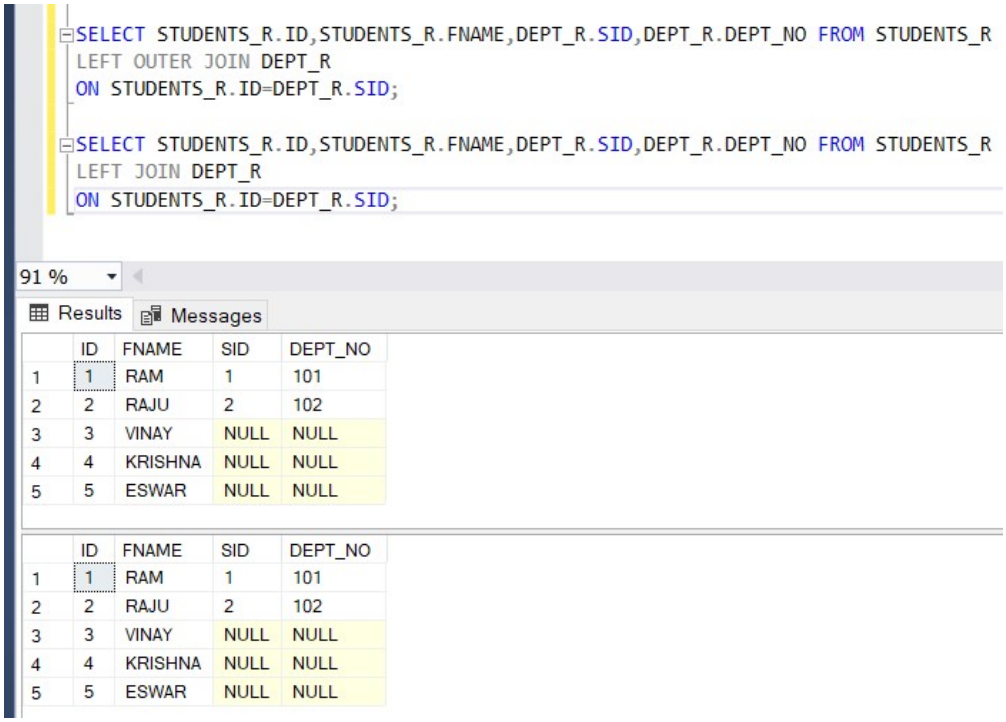
	ID	FNAME	SID	DEPT_NO
1	1	RAM	1	101
2	2	RAJU	2	102
3	3	VINAY	NULL	NULL
4	4	KRISHNA	NULL	NULL
5	5	ESWAR	NULL	NULL

	SID	DEPT_NO	ID	FNAME
1	1	101	1	RAM
2	2	102	2	RAJU
3	6	103	NULL	NULL
4	7	104	NULL	NULL

EX-4 LEFT OUTER VS LEFT JOIN BOTH ARE SAME

```
SELECT STUDENTS_R.ID,STUDENTS_R.FNAME,DEPT_R.SID,DEPT_R.DEPT_NO FROM STUDENTS_R
LEFT OUTER JOIN DEPT_R
ON STUDENTS_R.ID=DEPT_R.SID;

SELECT STUDENTS_R.ID,STUDENTS_R.FNAME,DEPT_R.SID,DEPT_R.DEPT_NO FROM STUDENTS_R
LEFT JOIN DEPT_R
ON STUDENTS_R.ID=DEPT_R.SID;
```



91 %

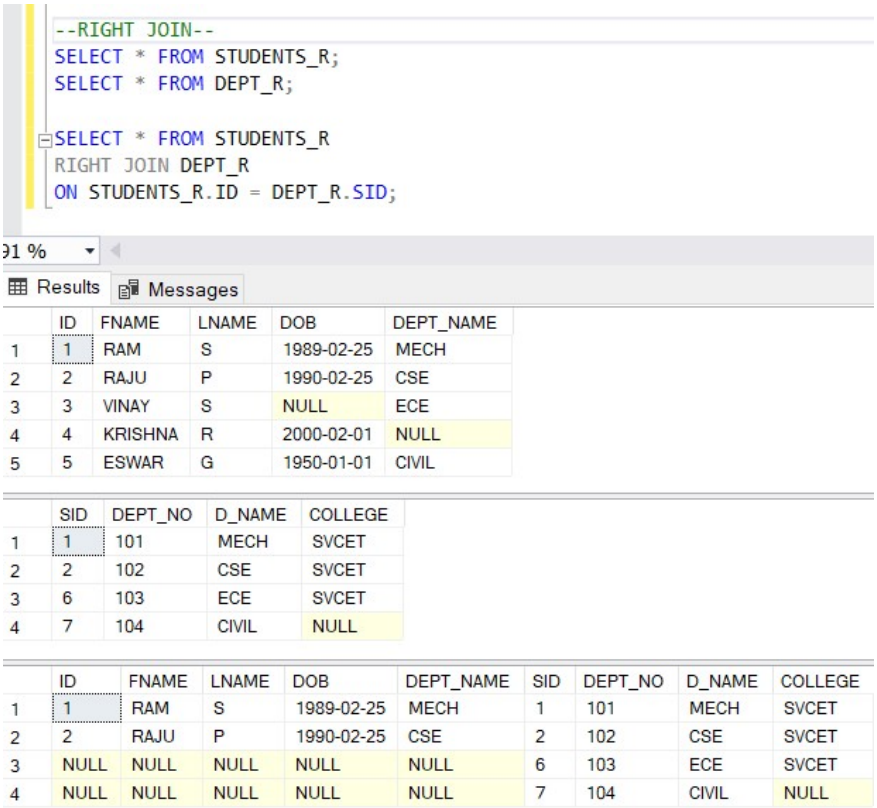
Results | Messages

	ID	FNAME	SID	DEPT_NO
1	1	RAM	1	101
2	2	RAJU	2	102
3	3	VINAY	NULL	NULL
4	4	KRISHNA	NULL	NULL
5	5	ESWAR	NULL	NULL

	ID	FNAME	SID	DEPT_NO
1	1	RAM	1	101
2	2	RAJU	2	102
3	3	VINAY	NULL	NULL
4	4	KRISHNA	NULL	NULL
5	5	ESWAR	NULL	NULL

```
--RIGHT JOIN--
SELECT * FROM STUDENTS_R
RIGHT JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

EX-1



91 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	NULL	NULL	NULL	NULL	NULL	6	103	ECE	SVCET
4	NULL	NULL	NULL	NULL	NULL	7	104	CIVIL	NULL

EX-2

BY INTERCHANGING TABLES

```
--RIGHT JOIN--
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;

SELECT * FROM STUDENTS_R
RIGHT JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;

SELECT * FROM DEPT_R
RIGHT JOIN STUDENTS_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

91 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	NULL	NULL	NULL	NULL	NULL	6	103	ECE	SVCET
4	NULL	NULL	NULL	NULL	NULL	7	104	CIVIL	NULL

	SID	DEPT_NO	D_NAME	COLLEGE	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	101	MECH	SVCET	1	RAM	S	1989-02-25	MECH
2	2	102	CSE	SVCET	2	RAJU	P	1990-02-25	CSE
3	NULL	NULL	NULL	NULL	3	VINAY	S	NULL	ECE
4	NULL	NULL	NULL	NULL	4	KRIS...	R	2000-02-01	NULL
5	NULL	NULL	NULL	NULL	5	ESWAR	G	1950-01-01	CIVIL

```
--FULL JOIN--
SELECT * FROM STUDENTS_R
FULL JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

EX-1

```
--FULL JOIN--
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;

SELECT * FROM STUDENTS_R
FULL JOIN DEPT_R
ON STUDENTS_R.ID = DEPT_R.SID;
```

100 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	3	VINAY	S	NULL	ECE	NULL	NULL	NULL	NULL
4	4	KRISHNA	R	2000-02-01	NULL	NULL	NULL	NULL	NULL
5	5	ESWAR	G	1950-01-01	CIVIL	NULL	NULL	NULL	NULL
6	NULL	NULL	NULL	NULL	NULL	6	103	ECE	SVCET
7	NULL	NULL	NULL	NULL	NULL	7	104	CIVIL	NULL

EX-2 BY INTER CHANGING TABLES

```
SELECT * FROM STUDENTS_R;
SELECT * FROM DEPT_R;

SELECT * FROM STUDENTS_R
FULL JOIN DEPT_R
ON DEPT_R.SID = STUDENTS_R.ID;

SELECT * FROM DEPT_R
FULL JOIN STUDENTS_R
ON DEPT_R.SID = STUDENTS_R.ID;
```

91 %

Results | Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	RAM	S	1989-02-25	MECH
2	2	RAJU	P	1990-02-25	CSE
3	3	VINAY	S	NULL	ECE
4	4	KRISHNA	R	2000-02-01	NULL
5	5	ESWAR	G	1950-01-01	CIVIL

	SID	DEPT_NO	D_NAME	COLLEGE
1	1	101	MECH	SVCET
2	2	102	CSE	SVCET
3	6	103	ECE	SVCET
4	7	104	CIVIL	NULL

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
3	3	VINAY	S	NULL	ECE	NULL	NULL	NULL	NULL
4	4	KRISHNA	R	2000-02-01	NULL	NULL	NULL	NULL	NULL
5	5	ESWAR	G	1950-01-01	CIVIL	NULL	NULL	NULL	NULL
6	NULL	NULL	NULL	NULL	NULL	6	103	ECE	SVCET
7	NULL	NULL	NULL	NULL	NULL	7	104	CIVIL	NULL

	SID	DEPT_NO	D_NAME	COLLEGE	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	101	MECH	SVCET	1	RAM	S	1989-02-25	MECH
2	2	102	CSE	SVCET	2	RAJU	P	1990-02-25	CSE
3	NULL	NULL	NULL	NULL	3	VINAY	S	NULL	ECE
4	NULL	NULL	NULL	NULL	4	KRIS...	R	2000-02-01	NULL
5	NULL	NULL	NULL	NULL	5	ESWAR	G	1950-01-01	CIVIL
6	6	103	ECE	SVCET	NULL	NULL	NULL	NULL	NULL
7	7	104	CIVIL	NULL	NULL	NULL	NULL	NULL	NULL

--CROSS JOIN--

```
--CROSS JOIN--
SELECT STUDENTS_R.ID,DEPT_R.SID FROM STUDENTS_R
CROSS JOIN DEPT_R;
SELECT DEPT_R.SID,STUDENTS_R.ID FROM DEPT_R
CROSS JOIN STUDENTS_R;
```

EX-1

Results M

	ID	SID
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	1	2
7	2	2
8	3	2
9	4	2
10	5	2
11	1	6
12	2	6
13	3	6
14	4	6
15	5	6
16	1	7
17	2	7
18	3	7
19	4	7
20	5	7

	SID	ID
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	2	1
7	2	2
8	2	3
9	2	4
10	2	5
11	6	1
12	6	2
13	6	3
14	6	4
15	6	5
16	7	1
17	7	2
18	7	3
19	7	4
20	7	5

EX-2

```
SELECT * FROM STUDENTS_R
CROSS JOIN DEPT_R;

SELECT * FROM DEPT_R
CROSS JOIN STUDENTS_R;
```

Results Messages

	ID	FNAME	LNAME	DOB	DEPT_NAME	SID	DEPT_NO	D_NAME	COLLEGE
1	1	RAM	S	1989-02-25	MECH	1	101	MECH	SVCET
2	2	RAJU	P	1990-02-25	CSE	1	101	MECH	SVCET
3	3	VINAY	S	NULL	ECE	1	101	MECH	SVCET
4	4	KRISHNA	R	2000-02-01	NULL	1	101	MECH	SVCET
5	5	ESWAR	G	1950-01-01	CIVIL	1	101	MECH	SVCET
6	1	RAM	S	1989-02-25	MECH	2	102	CSE	SVCET
7	2	RAJU	P	1990-02-25	CSE	2	102	CSE	SVCET
8	3	VINAY	S	NULL	ECE	2	102	CSE	SVCET
9	4	KRISHNA	R	2000-02-01	NULL	2	102	CSE	SVCET
10	5	ESWAR	G	1950-01-01	CIVIL	2	102	CSE	SVCET
11	1	RAM	S	1989-02-25	MECH	6	103	ECE	SVCET
12	2	RAJU	P	1990-02-25	CSE	6	103	ECE	SVCET
13	3	VINAY	S	NULL	ECE	6	103	ECE	SVCET
14	4	KRISHNA	R	2000-02-01	NULL	6	103	ECE	SVCET
15	5	ESWAR	G	1950-01-01	CIVIL	6	103	ECE	SVCET
16	1	RAM	S	1989-02-25	MECH	7	104	CIVIL	NULL
17	2	RAJU	P	1990-02-25	CSE	7	104	CIVIL	NULL
18	3	VINAY	S	NULL	ECE	7	104	CIVIL	NULL
19	4	KRISHNA	R	2000-02-01	NULL	7	104	CIVIL	NULL
20	5	ESWAR	G	1950-01-01	CIVIL	7	104	CIVIL	NULL

	SID	DEPT_NO	D_NAME	COLLEGE	ID	FNAME	LNAME	DOB	DEPT_NAME
1	1	101	MECH	SVCET	1	RAM	S	1989-02-25	MECH
2	1	101	MECH	SVCET	2	RAJU	P	1990-02-25	CSE
3	1	101	MECH	SVCET	3	VINAY	S	NULL	ECE
4	1	101	MECH	SVCET	4	KRISHNA	R	2000-02-01	NULL
5	1	101	MECH	SVCET	5	ESWAR	G	1950-01-01	CIVIL
6	2	102	CSE	SVCET	1	RAM	S	1989-02-25	MECH
7	2	102	CSE	SVCET	2	RAJU	P	1990-02-25	CSE
8	2	102	CSE	SVCET	3	VINAY	S	NULL	ECE
9	2	102	CSE	SVCET	4	KRISHNA	R	2000-02-01	NULL
10	2	102	CSE	SVCET	5	ESWAR	G	1950-01-01	CIVIL
11	6	103	ECE	SVCET	1	RAM	S	1989-02-25	MECH
12	6	103	ECE	SVCET	2	RAJU	P	1990-02-25	CSE
13	6	103	ECE	SVCET	3	VINAY	S	NULL	ECE
14	6	103	ECE	SVCET	4	KRISHNA	R	2000-02-01	NULL
15	6	103	ECE	SVCET	5	ESWAR	G	1950-01-01	CIVIL
16	7	104	CIVIL	NULL	1	RAM	S	1989-02-25	MECH
17	7	104	CIVIL	NULL	2	RAJU	P	1990-02-25	CSE
18	7	104	CIVIL	NULL	3	VINAY	S	NULL	ECE
19	7	104	CIVIL	NULL	4	KRISHNA	R	2000-02-01	NULL
20	7	104	CIVIL	NULL	5	ESWAR	G	1950-01-01	CIVIL

23. SUB QUERIES

NESTED QUERIES / SUB QUERIES

- A Query which is written with the other Query refers to NESTED Query.
- A Query which allows to write other query is called Outer Query or Main Query.
- A Query which is written in main query is called INNER or SUB-QUERY

➢ CLASSIFICATION OF SUB QUERIES

- Normal Sub Query
- Correlated Sub Query

A SUB-QUERY can n be written

- At WHERE Clause
- At HAVING Clause
- At FROM Clause ----- called as INLINE VIEW
- As an Expression ---- called as SCALAR SUB QUERY
- Under DML Queries

```
Q) Display EMPNO, ENAME,JOB,SAL, DEPTNO of that Employee who is being paid by Max Salary.
SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP1 WHERE SAL=(SELECT MAX(SAL) FROM EMP1);
Q) Display EMPNO, ENAME, JOB,SAL of those Employees whose salary is more than Average Salary of all
Employees.
SELECT EMPNO,ENAME,JOB,SAL FROM EMP1 WHERE SAL > (SELECT AVG(SAL) FROM EMP1);
Q) Display Ename, Job of those Employees who are working on a same job as of TURNER
SELECT ENAME,JOB FROM EMP1 WHERE JOB=(SELECT JOB FROM EMP1 WHERE ENAME='TURNER');
Q) Display ENAME,SAL of those Employees whose salary is more than average salary of Deptno 20.
SELECT ENAME,SAL FROM EMP1 WHERE SAL>(SELECT AVG(SAL) FROM EMP1 WHERE DEPTNO=20);
```

Q) Display ENAME,JOB,SAL of those Employees who are working on a same job as of "Allen" and Salary is more than Salary of "Miller".

```
SELECT ENAME,JOB,SAL FROM EMP1 WHERE JOB=((SELECT JOB FROM EMP1 WHERE ENAME ='ALLEN')) AND (SAL>
(SELECT SAL FROM EMP1 WHERE ENAME='MILLER'));
```

```
SELECT ENAME, JOB, SAL
FROM EMP1 WHERE JOB = ( SELECT JOB FROM EMP1 WHERE ENAME = 'ALLEN') AND SAL > ( SELECT SAL FROM EMP1
WHERE ENAME ='MILLER')
```

Q) Display Second Max Salary.

```
SELECT MAX(SAL) FROM EMP1 WHERE SAL<( SELECT MAX(SAL) FROM EMP1);
```

```
SELECT MAX (SAL) FROM EMP1 WHERE SAL <    ( SELECT MAX (SAL) FROM EMP1 );
SELECT ENAME, EMPNO , JOB, SAL , DEPTNO FROM EMP1
WHERE SAL=(SELECT MAX (SAL) FROM EMP1 WHERE SAL < (SELECT MAX (SAL) FROM EMP1));
```

Q) Write a Query to display First 3 Max Salaries of Employees.

```
SELECT  EMPNO,ENAME,SAL FROM EMP1 WHERE SAL >= (SELECT MAX(SAL) FROM EMP1 WHERE SAL< (SELECT
MAX(SAL) FROM EMP1 WHERE SAL<(SELECT MAX(SAL) FROM EMP1)));
```

```
SELECT EMPNO , ENAME, SAL FROM EMP1 WHERE SAL >= (SELECT MAX (SAL) FROM EMP1 WHERE SAL < (SELECT MAX
(SAL) FROM EMP1
WHERE SAL < (SELECT MAX (SAL) FROM EMP1)))
```

Q) Display DEPTNO & Average Salary of those departments whose average salary is more than the average salary of all Employees.

```
SELECT DEPTNO, AVG (SAL) FROM EMP1
HAVING AVG(SAL) > (SELECT AVG (SAL) FROM EMP1 GROUP BY DEPTNO);
```

At WHERE Clause

```
Q) Display EMPNO, ENAME,JOB,SAL, DEPTNO of that Employee who is being paid by Max Salary.

SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP1 WHERE SAL=(SELECT MAX(SAL) FROM EMP1);
```

110 %

Results | Messages

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7839	KING	PRESIDENT	5000	10

At HAVING Clause

```
Q)  Display DEPTNO & Average Salary of those departments whose average salary
is more than the average salary of all Employees.

SELECT DEPTNO, AVG(SAL) FROM EMP1 GROUP BY DEPTNO
HAVING AVG(SAL)>(SELECT AVG(SAL) FROM EMP1);
```

110 %

Results | Messages

	DEPTNO	(No column name)
1	10	2916
2	20	2175

At FROM Clause ----- called as INLINE VIEW

```
--Whole table wise top 2 sal

SELECT * FROM (SELECT DENSE_RANK() OVER (ORDER BY SAL DESC ) DNR ,* FROM EMP1) Q1 WHERE Q1.DNR IN(1,2);

--Dept wise top 2 highest sal--
SELECT * FROM (SELECT DENSE_RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) DNRP ,* FROM EMP1) Q2 WHERE Q2.DNRP IN (1,2);
```

110 %

Results | Messages

	DNR	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
2	2	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
3	2	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20

	DNRP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
2	2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
3	1	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
4	1	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
5	2	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
6	1	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
7	2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30

RANK FUNCTIONS.

- ROW_NUMBER()
- RANK()
- DENSE_RANK()

In the SQL RANK functions, we use the OVER() clause to define a set of rows in the result set. We can also use SQL PARTITION BY clause to define a subset of data in a partition. You can also use Order by clause to sort the results in a descending or ascending order.

ROW_Number () SQL RANK function

We use ROW_Number () SQL RANK function to get a unique sequential number for each row in the specified data. It gives the rank one for the first row and then increments the value by one for each row. We get different ranks for the row having similar values as well.

By default, it sorts the data in ascending order and starts assigning ranks for each row.

Ex: 1

```
SELECT ROW_NUMBER() OVER (ORDER BY SAL) RNO , * FROM EMP1;
```

```
SELECT ROW NUMBER() OVER (ORDER BY SAL) RNO , * FROM EMP1;
```

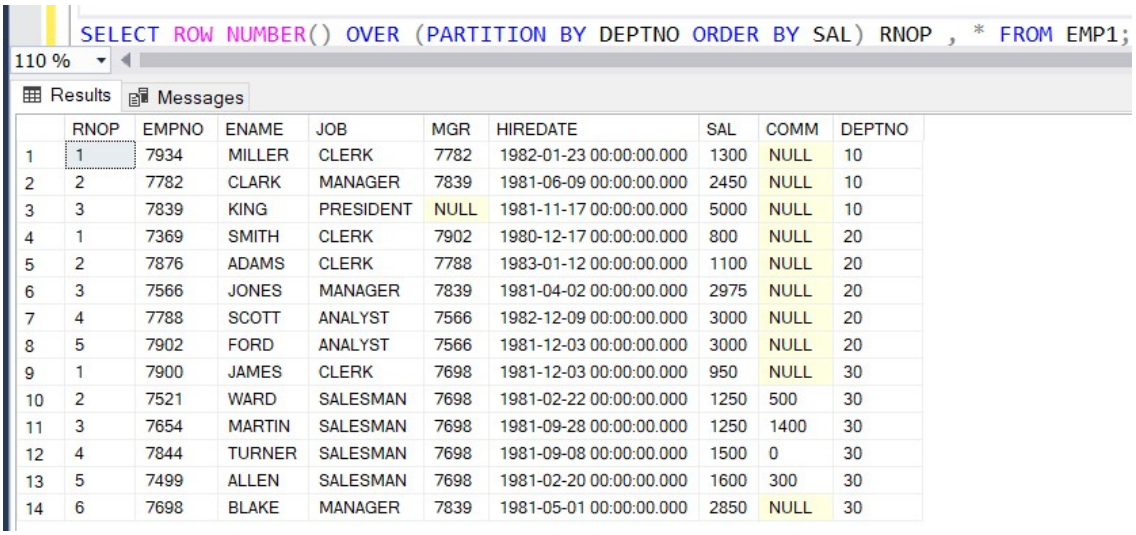
110 %

Results | Messages

	RNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	2	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
3	3	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
4	4	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
5	5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
6	6	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
7	7	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
8	8	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
9	9	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
10	10	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
11	11	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
12	12	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
13	13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
14	14	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10

EX:2

```
SELECT ROW_NUMBER() OVER (PARTITION BY DEPTNO ORDER BY SAL) RNOP , * FROM EMP1;
```



```
SELECT ROW NUMBER() OVER (PARTITION BY DEPTNO ORDER BY SAL) RNOP , * FROM EMP1;
```

110 %

Results | Messages

	RNOP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
2	2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
3	3	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
4	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
5	2	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
6	3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
7	4	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
8	5	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
9	1	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
10	2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
11	3	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
12	4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
13	5	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
14	6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30

RANK()

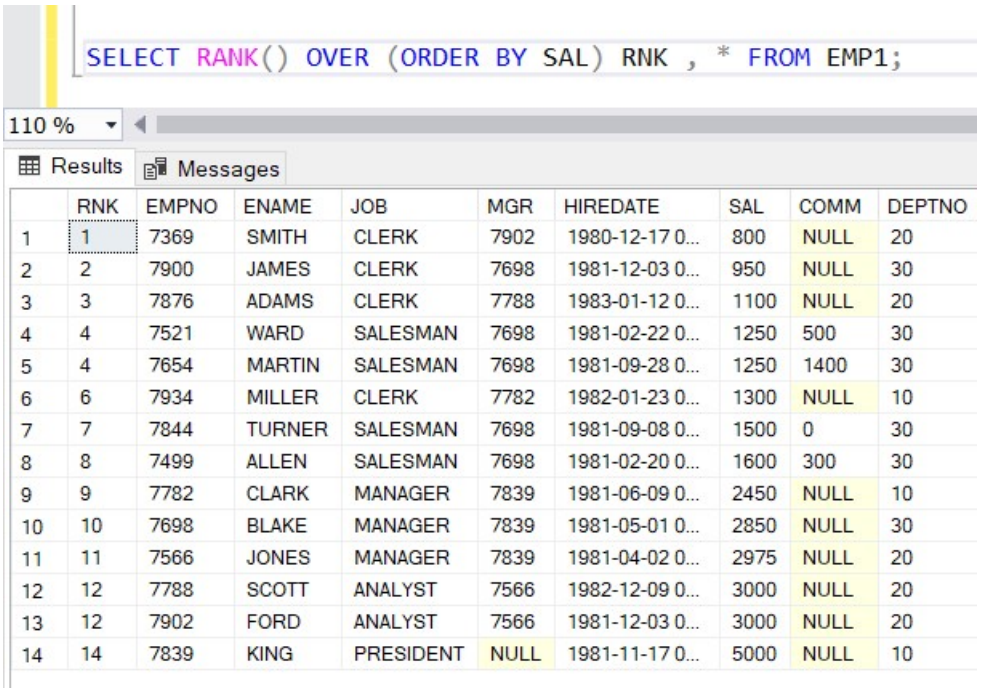
It is used to return a unique rank number for the each distinct row within the specified partition.

It starts from 1 for the first row in each partition, with the same rank for duplicate values and it leaves gaps between the ranks; this gap appears in the sequence after the duplicate values.

RANK () behaves like ROW_NUMBER () function except for the rows with equal values, where it will rank with the same rank ID and generate a gap after it.

EX : 1

```
SELECT RANK() OVER (ORDER BY SAL) RNK , * FROM EMP1;
```



```
SELECT RANK() OVER (ORDER BY SAL) RNK , * FROM EMP1;
```

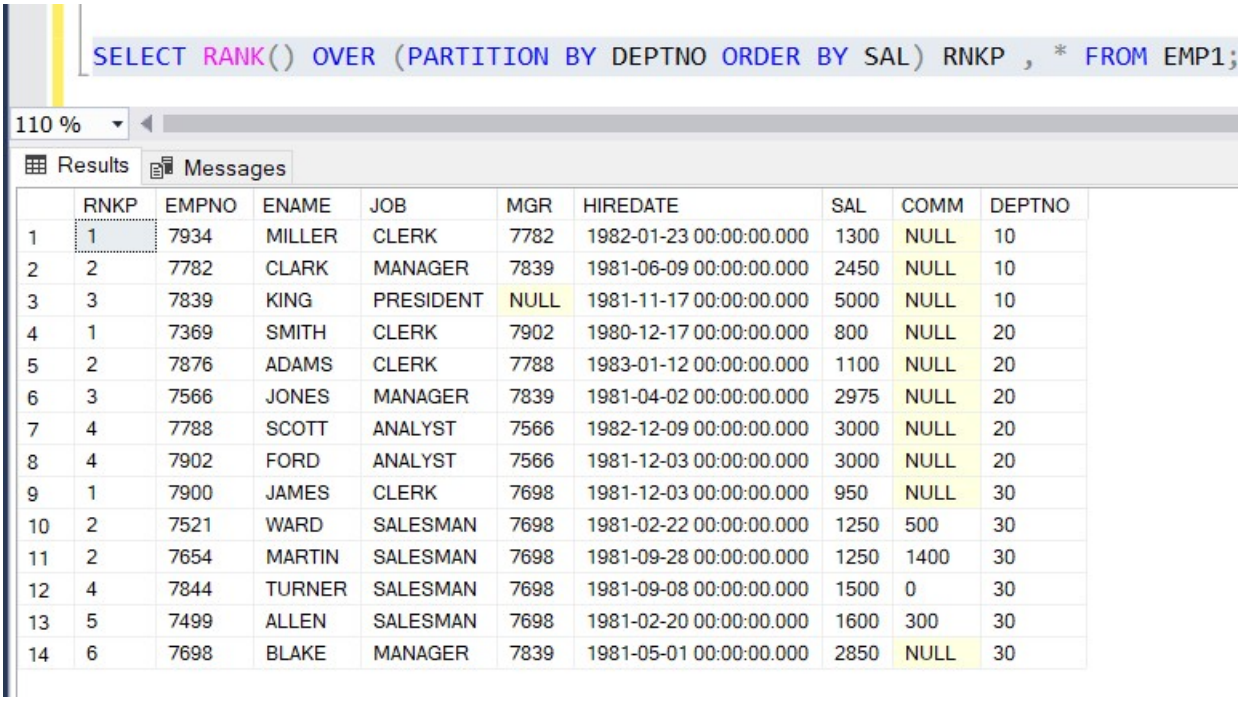
110 %

Results | Messages

	RNK	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7369	SMITH	CLERK	7902	1980-12-17 0...	800	NULL	20
2	2	7900	JAMES	CLERK	7698	1981-12-03 0...	950	NULL	30
3	3	7876	ADAMS	CLERK	7788	1983-01-12 0...	1100	NULL	20
4	4	7521	WARD	SALESMAN	7698	1981-02-22 0...	1250	500	30
5	4	7654	MARTIN	SALESMAN	7698	1981-09-28 0...	1250	1400	30
6	6	7934	MILLER	CLERK	7782	1982-01-23 0...	1300	NULL	10
7	7	7844	TURNER	SALESMAN	7698	1981-09-08 0...	1500	0	30
8	8	7499	ALLEN	SALESMAN	7698	1981-02-20 0...	1600	300	30
9	9	7782	CLARK	MANAGER	7839	1981-06-09 0...	2450	NULL	10
10	10	7698	BLAKE	MANAGER	7839	1981-05-01 0...	2850	NULL	30
11	11	7566	JONES	MANAGER	7839	1981-04-02 0...	2975	NULL	20
12	12	7788	SCOTT	ANALYST	7566	1982-12-09 0...	3000	NULL	20
13	12	7902	FORD	ANALYST	7566	1981-12-03 0...	3000	NULL	20
14	14	7839	KING	PRESIDENT	NULL	1981-11-17 0...	5000	NULL	10

EX : 2

```
SELECT RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL) RNKP , * FROM EMP1;
```



```
SELECT RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL) RNKP , * FROM EMP1;
```

110 %

Results | Messages

	RNKP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
2	2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
3	3	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
4	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
5	2	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
6	3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
7	4	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
8	4	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
9	1	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
10	2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
11	2	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
12	4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
13	5	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
14	6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30

DENSE_RANK()

It is similar to RANK() but the only difference is DENSE_RANK() does not skip any rank, i.e. leaving no gap(s) between the gap(s).

Generally, DENSE_RANK() is used to provide ranking to the records

We use DENSE_RANK() function to specify a unique rank number within the partition as per the specified column value. It is similar to the Rank function with a small difference.
In the SQL RANK function DENSE_RANK(), if we have duplicate values, SQL assigns different ranks to those rows as well. Ideally, we should get the same rank for duplicate or similar values.

RANK and DENSE_RANK

- ROW_NUMBER will always generate unique values without any gaps, even if there are ties.
- RANK can have gaps in its sequence and when values are the same, they get the same rank.
- DENSE_RANK also returns the same rank for ties, but doesn't have any gaps in the sequence.

EX : 1

```
SELECT DENSE_RANK() OVER(ORDER BY SAL) DRNK , * FROM EMP1;
```

```
SELECT DENSE_RANK() OVER(ORDER BY SAL) DRNK , * FROM EMP1;
```

110 %

Results | Messages

	DRNK	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	2	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
3	3	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
4	4	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
5	4	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
6	5	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
7	6	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
8	7	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
9	8	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
10	9	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
11	10	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
12	11	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
13	11	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
14	12	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10

EX : 2

```
SELECT DENSE_RANK() OVER ( PARTITION BY DEPTNO ORDER BY SAL) DRNKP ,* FROM EMP1;
```

SELECT DENSE_RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL) DRNKP ,* FROM EMP1;

110 %

Results | Messages

	DRNKP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
2	2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
3	3	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
4	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
5	2	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
6	3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
7	4	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
8	4	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
9	1	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
10	2	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
11	2	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
12	3	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
13	4	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
14	5	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30

SQL SUB QUERIES ALONG WITH RANK FUNCTIONS

```
--In full table , 3rd highest on

SELECT * FROM(SELECT DENSE_RANK() OVER (ORDER BY SAL DESC) DNR,* FROM EMP1) Q3 WHERE Q3.DNR IN(3);

-- Dept wise 3rd highest
SELECT * FROM (SELECT DENSE_RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) DNR, * FROM EMP1) Q4 WHERE Q4.DNR IN(3);
```

0 %

Results | Messages

	DNR	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20

	DNR	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	3	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10
	3	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
	3	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

```
-- In full table 2nd and 4th highest

SELECT * FROM (SELECT DENSE_RANK() OVER (ORDER BY SAL DESC ) DNR, * FROM EMP1) Q5 WHERE Q5.DNR IN (2,4);

-- Dept wise 2nd and 4th highest In sal

SELECT * FROM (SELECT DENSE_RANK() OVER(PARTITION BY DEPTNO ORDER BY SAL DESC) DNRP ,* FROM EMP1) Q6 WHERE Q6.DNRP IN (2,4);
```

110 %

Results Messages

	DNR	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	2	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
2	2	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
3	4	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30

	DNRP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	2	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
2	2	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
3	4	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
4	2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
5	4	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
6	4	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30

```
--Full table wise least 3rd record

SELECT * FROM (SELECT DENSE_RANK() OVER (ORDER BY SAL ASC) DNR , * FROM EMP1) Q7 WHERE Q7.DNR IN (3);
--Full table DEPT WISE wise least 3rd record
SELECT * FROM ( SELECT DENSE_RANK() OVER ( PARTITION BY DEPTNO ORDER BY SAL ASC ) DNRP , * FROM EMP1) Q6 WHERE Q6.DNRP IN (3);
```

0 %

Results Messages

	DNR	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	3	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20

	DNRP	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
	3	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
	3	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
	3	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30

```
--Dept wise emp count
/*SELECT DENSE_RANK() OVER (ORDER BY COUNT (*) ASC) DNR ,COUNT(*) AS COUNT,DEPTNO FROM EMP1 GROUP BY DEPTNO;*/
SELECT DEPTNO, COUNT(ENAME) AS DWISECOUNT FROM EMP1 GROUP BY DEPTNO;
SELECT DEPTNO,COUNT(*) AS DWISECOUNT, DENSE_RANK() OVER (ORDER BY COUNT (*) ASC) DNR FROM EMP1 GROUP BY DEPTNO;
```

110 %

Results Messages

	DEPTNO	DWISECOUNT
1	10	3
2	20	5
3	30	6

	DEPTNO	DWISECOUNT	DNR
1	10	3	1
2	20	5	2
3	30	6	3

```
--Job wise emp count
SELECT JOB,COUNT(JOB) AS COUNTDEPWISE FROM EMP1 GROUP BY JOB ORDER BY COUNT(JOB);

SELECT JOB,COUNT(*) AS JOBWISECNT,DENSE_RANK() OVER (ORDER BY COUNT(*) ASC) DNR FROM EMP1 GROUP BY JOB;
```

110 %

Results Messages

	JOB	COUNTDEPWISE
1	PRESIDENT	1
2	ANALYST	2
3	MANAGER	3
4	CLERK	4
5	SALESMAN	4

	JOB	JOBWISECNT	DNR
1	PRESIDENT	1	1
2	ANALYST	2	2
3	MANAGER	3	3
4	CLERK	4	4
5	SALESMAN	4	4

```
/*To display records like
1, 3, 5 rows (from whole table)
This you can do by row number*/

SELECT * FROM (SELECT ROW_NUMBER() OVER (ORDER BY EMPNO) RNO,* FROM EMP1) Q10 WHERE Q10.RNO%2=1;
```

110 %

Results | Messages

	RNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
3	5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
4	7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
5	9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
6	11	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
7	13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20

CORRELATED SUB QUERY

- As we all know query within a query is call sub query and it has two types one is normal sub query and another is correlated sub query.
- Normal sub query has one inner query and one outer query
- Normal sub query can be written at where clause, from clause, having clause and as an expression.
- In Normal sub query Inner query will execute 1st and that result will become the input for outer query.
- But in Co related sub query - both inner and outer queries execution will be in parallel

A correlated subquery is a subquery that uses the values of the outer query. In other words, it depends on the outer query for its values. Because of this dependency, a correlated subquery cannot be executed independently as a simple subquery.

Moreover, a correlated subquery is executed repeatedly, once for each row evaluated by the outer query. The correlated subquery is also known as a repeating subquery.

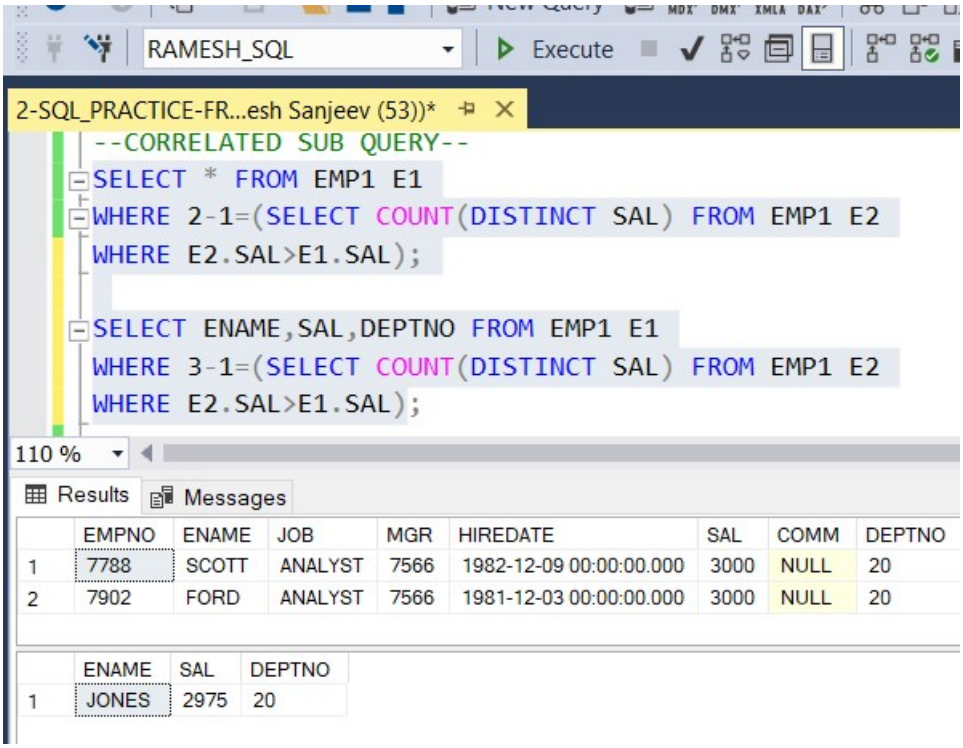
Syntax:

SELECT * FROM EMP e1/ SELECT ENAME, SAL FROM EMP1 E1
WHERE N-1= (SELECT COUNT (DISTINCT salary) FROM EMP e2
WHERE e2.sal > e1.sal)
(In this above e1 and e2 are same table as EMP — with alias names)

In the above example Put N value as 2 and get the 2nd highest and put the N value as 3 and get the 3 rd highest

```
--CORRELATED SUB QUERY--
SELECT * FROM EMP1 E1
WHERE 2-1=(SELECT COUNT(DISTINCT SAL) FROM EMP1 E2
WHERE E2.SAL>E1.SAL);

SELECT ENAME,SAL,DEPTNO FROM EMP1 E1
WHERE 3-1=(SELECT COUNT(DISTINCT SAL) FROM EMP1 E2
WHERE E2.SAL>E1.SAL);
```



24. INDEX

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

CLASSIFIED IN THREE TYPES

- Clustered Index
- Non Clustered Index
- Unique

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CLUSTERED INDEX

- It will Alter the physical representation of rows in a table.
- A table can have only One Clustered Index.
- It will always arrange the data of a table in Ascending order
- Data pages and Index pages will be stored at one level.
- Index should be created for a column where more than one search value is available

NON CLUSTERED INDEX

- This index will not Alter the physical requirement of rows in a table.
- A table can have 249 Non Clustered Index's.
- Data is not arranged in Order.
- It is a default index created.
- Data pages & index pages are stored at different level.

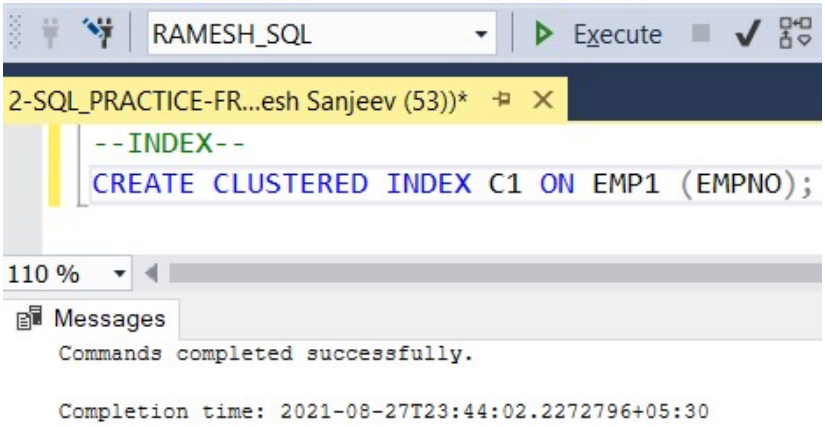
UNIQUE INDEX

- This index can be created only on those columns which contains unique data.
- This index is automatically created when unique constraint is created on a column.

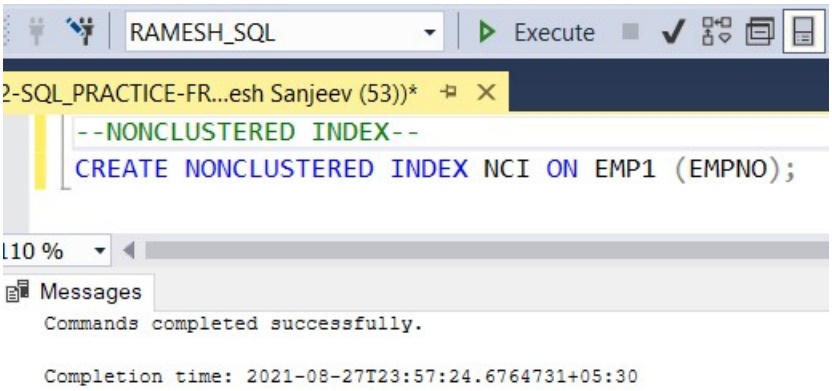
```
CREATE UNIQUE INDEX INDX4 ON DEPT DEPTNO
```

**SQL SERVER creates an Index automatically for two Constraints Primary Key & Unique key.

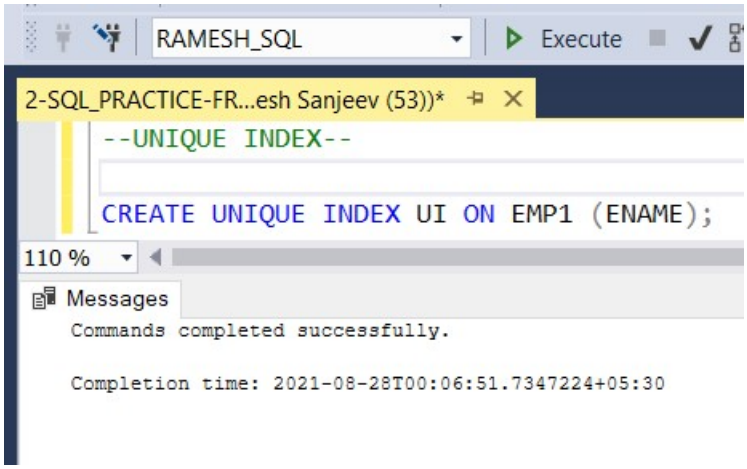
```
--INDEX--
CREATE CLUSTERED INDEX C1 ON EMP1 (EMPNO);
```




```
--NONCLUSTERED INDEX--
CREATE NONCLUSTERED INDEX NCI ON EMP1 (EMPNO);
```




```
--UNIQUE INDEX--
CREATE UNIQUE INDEX UI ON EMP1 (ENAME);
```



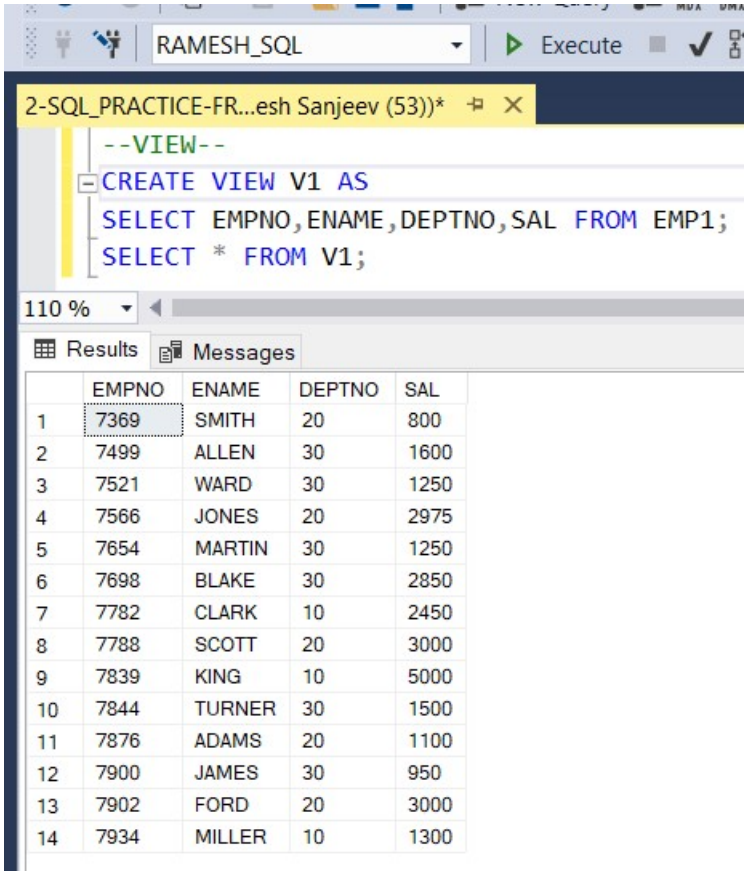
25. VIEWS

- View is a window of virtual table which will allow related users to view and modify related data.
- Any modifications are made through a view will reflect on base table and vice versa , which leads to data consistency
- A view is a database object which resides in database server and stores only select query in compiled format hence it is called stored query.

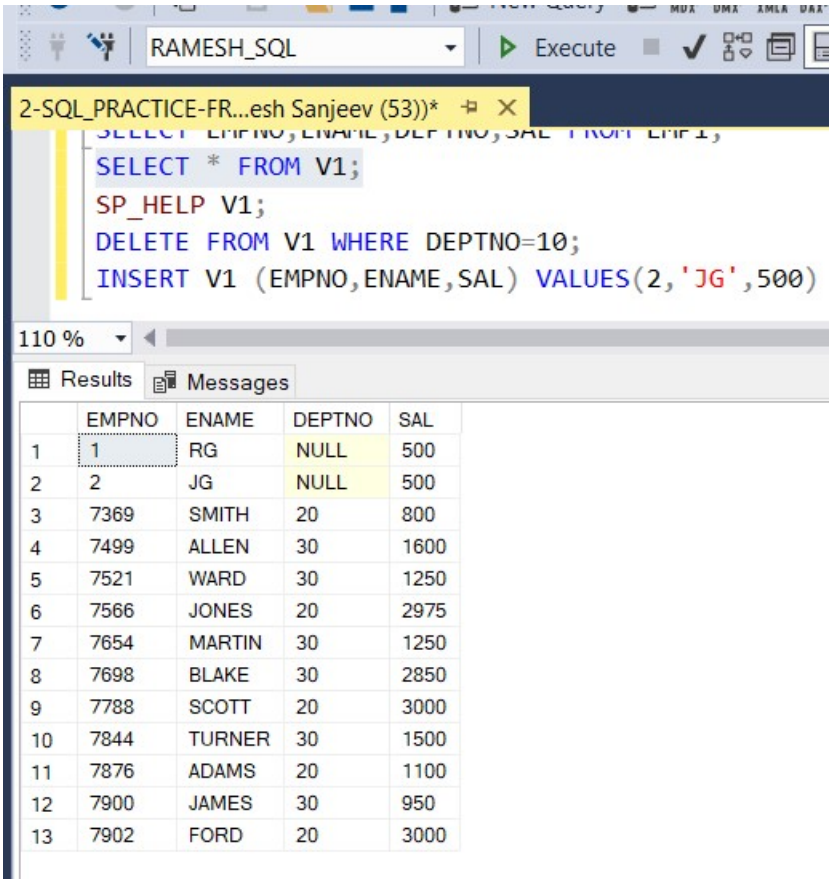
CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
CREATE VIEW V1 AS
SELECT EMPNO,ENAME,DEPTNO,SAL FROM EMP1;
SELECT * FROM V1;
SP_HELP V1;
DELETE FROM V1 WHERE DEPTNO=10;
INSERT V1 (EMPNO,ENAME,SAL) VALUES(2,'JG',500);
```



	EMPNO	ENAME	DEPTNO	SAL
1	7369	SMITH	20	800
2	7499	ALLEN	30	1600
3	7521	WARD	30	1250
4	7566	JONES	20	2975
5	7654	MARTIN	30	1250
6	7698	BLAKE	30	2850
7	7782	CLARK	10	2450
8	7788	SCOTT	20	3000
9	7839	KING	10	5000
10	7844	TURNER	30	1500
11	7876	ADAMS	20	1100
12	7900	JAMES	30	950
13	7902	FORD	20	3000
14	7934	MILLER	10	1300



	EMPNO	ENAME	DEPTNO	SAL
1	1	RG	NULL	500
2	2	JG	NULL	500
3	7369	SMITH	20	800
4	7499	ALLEN	30	1600
5	7521	WARD	30	1250
6	7566	JONES	20	2975
7	7654	MARTIN	30	1250
8	7698	BLAKE	30	2850
9	7788	SCOTT	20	3000
10	7844	TURNER	30	1500
11	7876	ADAMS	20	1100
12	7900	JAMES	30	950
13	7902	FORD	20	3000

Encrypted View:

- When a view is encrypted it does not support to display the text which is stored in a view.
- SQL Server does not support decryption of view.
- To create this view, it should be associated by with encryption
- The query which is stored in encrypted view can be altered

Schema Binding View:

Rules to be followed for creating schema binding view.

- Schema binding view will not support to store select query with * Selection operator.
- It is must that table name specified in select query should be preceded by schema name
- The advantage of creating schema binding views is that it will restrict the dropping of the base objects

```
CREATE VIEW V9 WITH ENCRYPTION
AS SELECT * FROM EMP1;


CREATE VIEW V10 WITH SCHEMABINDING
AS SELECT * FROM EMP1;
```