SQL (STRUCTURED QUERY LANGUAGE)

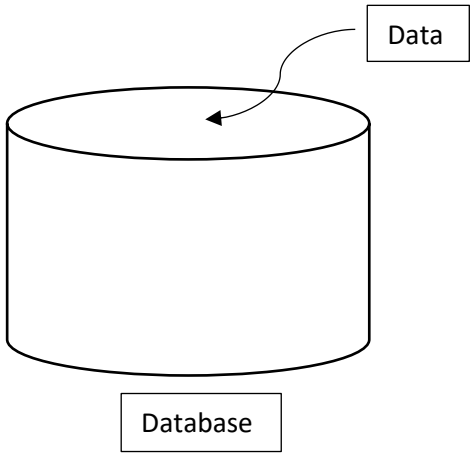
SQL stands for Structured Query Language, and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updating, insertion and deletion of data from a database.

Data

It is a raw fact which describes the attributes/properties of an entity.
Raw fact→unchanged
Entity→Object (living or non-living)

Database

It is place or medium which is used to store the data in a systematic and organized manner.



In a database, we do some frequent operations.

1) CREATE/INSERT
2) READ/RETRIEVE
3) UPDATE/MODIFY
4) DELETE/DROP

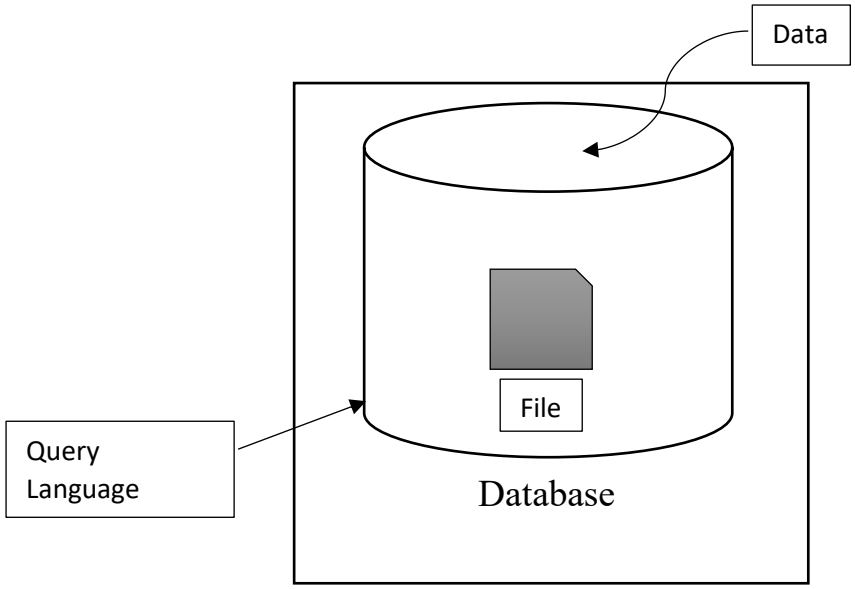
These are formally called as "**CRUD**" operation.

DBMS (Database Management System)

DBMS is a software which is used to maintain and manage the database.

- Security and authorization are the 2 important features provided by DBMS.
- In DBMS, we can store the data in the form of files.
- We use Query Language to communicate with DBMS.

DBMS acts as an interface between the data and the database.

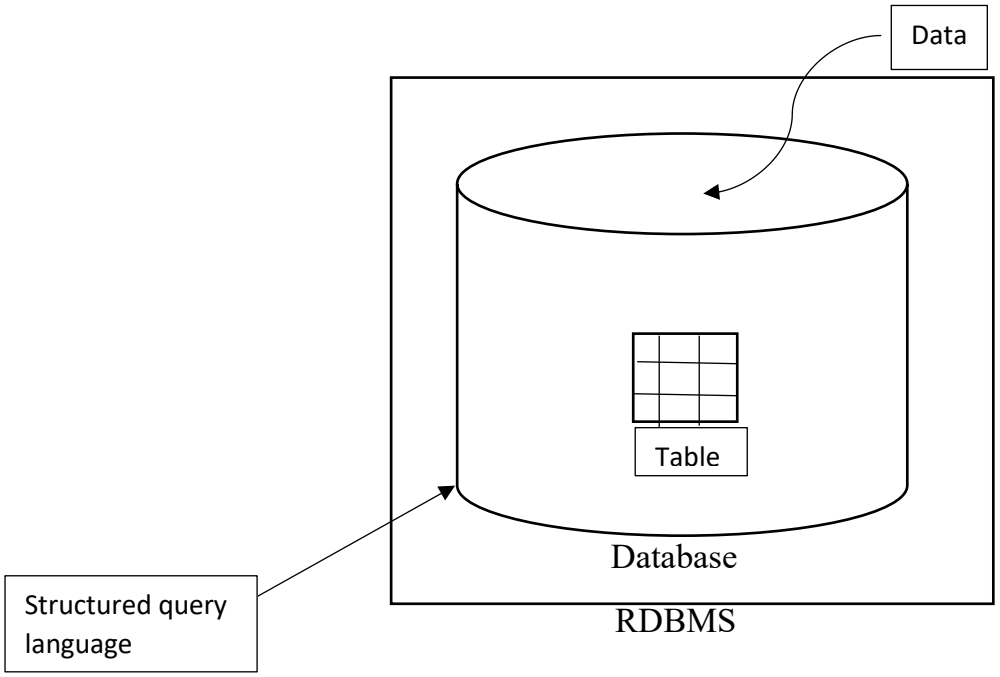


DBMS

RDMS (Relational Database Management System)

It is a type of DBMS software which is used to maintain and manage the database.

- Any DBMS software which follows relational model will be considered as RDBMS.
- Security and authorization are the 2 main important features provided by RDBMS.
- In RDBMS, we store the data in table format.
- We use Structured Query language to communicate or interact with RDBMS.



RDBMS

Difference between DBMS and RDBMS

DBMS	RDBMS
1. Data will be stored in file format.	1. Data will be stored in table format.
2. We use Query language to communicate with DBMS.	2. We use Structured query language to communicate with RDBMS.
3. Little bit difficult to store and access the data as compared to RDBMS.	3. Easy to store and access the data.

Relational model

- Relational model is invented by a data scientist E F Codd (Edgar Frank Codd).
- He gave the main rule as,
  In RDBMS, we have to store the data in Table format.

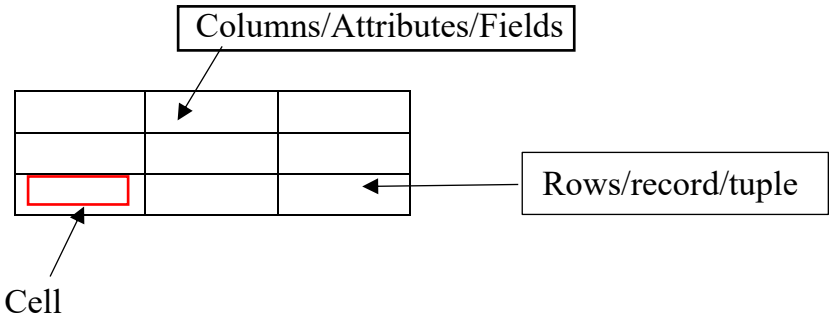


Table: A logical organization of rows and columns is called as table

Cell: The smallest unit of the table which is used to store the data is known as Cell. The intersection of rows and columns creates a cell.

Rules of E F Codd

1. The data enter into a cell should be a single valued data.
2. In RDBMS, we store the data in table format including meta data.
   Metadata: The data which describes another data is known as meta data.
   Meta data are stored in meta tables which is automatically generated by the compiler.
3. According to E F Codd, we can store the data in multiple tables, if required we can establish the connection between 2 tables.
4. We can validate the data by 2 steps
   1. By assigning datatypes.
   2. By assigning constraints.

Note: Datatypes are mandatory and constraints are optional.

DATATYPES

Datatypes are used to find out what type or kind of data that is given for a particular column.

Types of datatypes

1. CHAR
2. VARCHAR/VARCHAR2
3. DATE
4. NUMBER
5. LARGE OBJECTS
    a. CHARACTER LARGE OBJECTS
    b. BINARY LARGE OBJECTS

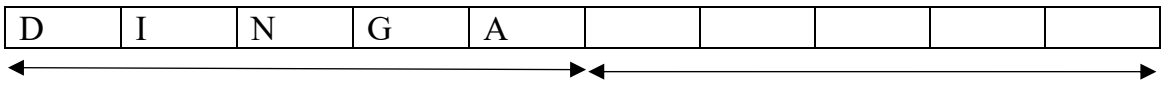
1. CHAR Datatype

- It is used to store the characters.
- It accepts the following
  'a-z', 'A-Z', '0-9', Special characters and Alphanumeric characters (Combination of alphabet and numbers).
- We have to write characters within ('') single quotes.

SYNTAX: CHAR(SIZE)

- Whenever we use char data type we have to mention the size for it.
- The maximum size of char data type is 2000ch.
- Char datatype follows "**Fixed length memory allocation**".

Example: CHAR (10)



Used memory Unused memory/Memory wastage

2. VARCHAR/VARCHAR2

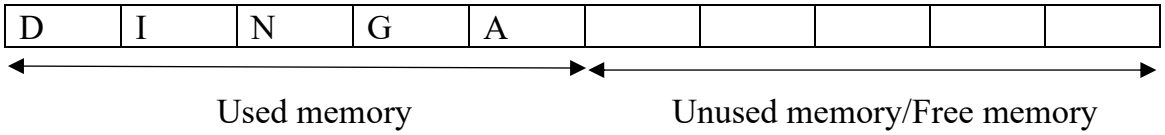
- It is used to store the characters.
- It accepts the following
  'a-z', 'A-Z', '0-9', Special characters and Alphanumeric characters (Combination of alphabet and numbers).
- We have to write characters within ('') single quotes.

SYNTAX: VARCHAR(SIZE)

- Whenever we use varchar data type we have to mention the size for it.
- The maximum size of char data type is 2000ch.

- Char datatype follows "**Variable length memory allocation**".

Example: CHAR (10)

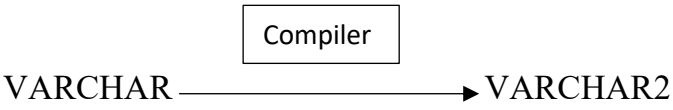


VARCHAR2

It is an updated version of VARCHAR datatype. The maximum size of VARCHAR2 is 4000ch.

Syntax: VARCHAR2(SIZE)

- VARCHAR is outdated, if you assign VARCHAR to any column, compiler will convert that into VARCHAR2 only.



3.DATE

- It is used to store the date kind of data.
- It will accept oracle-based date formats are
  1. 'DD-MON-YYYY'
     Example: '14-MAR-2024'
  2. 'DD-MON-YY'
     Example: '14-MAR-24'

Syntax: DATE

4.NUMBER

Number data type is used to accept only the numerical values that is given for a column.

Syntax: NUMBER (PRECISION, [SCALE])

PRECISION: It is used to determine the total number of digits present in the value.

The range of precision is 1 to 38.

SCALE: It is used to determine the total number of decimal digits present in the value.

The range of scale is from -84 to 127.

Scale is not mandatory and default value of scale is 0.

Example:

Case 1: Without scale value.

NUMBER (6)

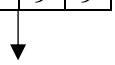
9	9	9	9	9	9

The maximum value that we can store here is 9,99,999 and we cannot store more than 9,99,999.

Case 2: When precision is more than scale.

NUMBER (4,2)

9	9	9	9



99.99 is the maximum value that we can store here.

Case 3: When precision is equal to scale.

NUMBER (3,3)

9	9	9

0.999 is the maximum value that we can store here.

Case 4: When precision is less than scale.

NUMBER (2,4)

Difference between scale and precision=S-P=4-2=2.

0	0	9	9

0.0099 is the maximum value that we can store here.

5.LARGE OBJECTS

1) CHARACTER LARGE OBJECTS:
   These are used to store characters up to 4GB of size.

   Syntax: CLOB

2) BINARY LARGE OBJECTS
   These are used to store Binary numbers of images, documents, MP3 & MP4 files etc up to 4GB of size.

   Syntax: BLOB

CONSTRAINTS

Constraints are the additional validation that is given for a column.

Types of constraints

1. UNIQUE
2. NOT NULL
3. CHECK
4. PRIMARY KEY
5. FOREIGN KEY

STUDENT TABLE

SID	SNAME	BRANCH	PHONE NO	AGE	TID
101	DINGA	MECH	1234567890	18	T3
102	DINGI	ECE	9876504321	22	T1
103	PENGA	CSE	8765432109	21	T2

TEACHERS TABLE

TID	TNAME	SALARY
T1	SMITH	25000
T2	MILLER	30000
T3	ADAMS	40000

1.UNIQUE

UNIQUE is a constraint which is given for a column where the column should not accept duplicate or repeated value.

Example: UNIQUE(SID), UNIQUE (PHONE NO)

2.NOT NULL

NOT NULL is a constraint which is given for a column where the column should not accept null values.

NULL: NULL is an empty space or nothing.

Whenever we create a column in the table we have to assign either null constraint or NOT NULL constraint.

"0" is not a NULL value.

Example: SID NOT NULL

SNAME NOT NULL

3.CHECK

Check is an extra validation which depends on the condition

If the condition is true then the value will be accepted and if the condition is false then the value will be rejected.

Example: CHECK(LENGTH(NUMBER)=10)

CHECK(AGE>=18)

4.PRIMARY KEY

Primary key is a constraint which is used to identify the records uniquely from the table.

Characteristics of PRIMARY KEY

1. Primary key cannot accept duplicate or repeated values.
2. Primary key cannot accept null values.
3. Primary key is a combination of UNIQUE and NOT NULL constraints.
4. In a table, we can have only one primary key.
5. Primary key is not mandatory but it is recommended to have one in a table.

Example: PRIMARY KEY(SID)

PRIMARY KEY(TID)

5.FOREIGN KEY

Foreign key is used to establish the connection between 2 tables. Foreign key is also known as REFERENTIAL INTEGRITY CONSTRAINT.

Characteristics of FOREIGN KEY

1. Foreign key can accept duplicate or repeated values.
2. Foreign key can accept null values.
3. Foreign key is not a combination of unique and not null constraints.
4. In a table, we can have any number of foreign keys.
5. To be a foreign key it should be primary key in its own table.
6. Foreign key present in child table, but it actually belongs to parent table.

Example: FOREIGN KEY(TID)

OVERVIEW OF SQL STATEMENTS

1. Data definition Language (DDL)
2. Data manipulation Language (DML)
3. Transaction control Language (TCL)
4. Data Control Language (DCL)
5. Data query Language (DQL)

1.Data definition Language (DDL)

It is used to do operations on the structure of the table.

1. CREATE: It is used to create a new table.
2. RENAME: It is used to rename the existing table.
3. ALTER: It is used to alter the structure of the table.
4. TRUNCATE: It is used to delete all the records permanently from the table.
5. DROP: It is used to remove the table from the database.

2.Data manipulation Language (DML)

It is used to do operations on the records of the table.

1. INSERT: It is used to insert the records into the table.
2. UPDATE: It is used to update the records of the table. We can update any particular record as well as all the records.
3. DELETE: It is used to delete the records of the table. We can delete any particular record as well as all the records.

3.Transaction control Language (TCL)

It is used to control the transactions.

1. COMMIT: It is used to save the transactions.
2. SAVEPOINT: It is used to create checkpoints.
3. ROLLBACK: It is used to undo the transactions.

4.Data control Language (DCL)

It used to give/ take back access from the user.

1. GRANT: It is used to give restricted permission to user.
2. REVOKE: It is used to take back given permission from the user.

5.Data Query Language (DQL)

It used to fetch the data from the database.

1. SELECT
2. PROJECTION
3. SELECTION

4. JOINS

DATA QUERY LANGUAGE

In this language, we learn how to fetch the data from the database.

Statements in DQL

1. SELECT
2. PROJECTION
3. SELECTION
4. JOINS

1.PROJECTION

PROJECTION is used to retrieve the data from the table by selecting only columns.

Syntax:

```
SELECT */[DISTINCT] COL_NAME/EXPRESSION [ALIAS]
FROM TABLE_NAME;
```

Order of execution

1. FROM Clause
2. SELECT Clause

Note: SQL is not a case sensitive Language but the literals are case sensitive.

1.FROM clause starts the execution.

2.For FROM Clause we have to pass table name as an argument.

3.FROM clause will go to the database and search for the given table, if the table is present in the database then the table will be kept under execution else FROM clause will stop the execution and gives you an error message.

4.After the successful execution of from clause next SELECT clause starts the execution.

5.For select clause we have to pass *(ASTERISK), COLUMN_NAME or Expression as an argument.

6.SELECT Clause is used to select the data and display it.

7.SELECT Clause is responsible for the result table.

ASTERISK (*):

It is used to select all the columns present in the table. We can also say that it is used to display the entire table.

SEMI COLON:

It is used for end of statement.

Examples

1.WAQTD Display details of employees.

```
SELECT *
FROM EMP;
```

2.WAQTD Display details of departments.

```
SELECT *
FROM DEPT;
```

3.WAQTD Names of all the employees.

```
SELECT ENAME
FROM EMP;
```

4.WAQTD Employee names and their salaries.

```
SELECT ENAME, SAL
FROM EMP;
```

5.WAQTD Salaries, Hire date, Deptno, Commission and Names of all the employees.

```
SELECT SAL, HIREDATE, DEPTNO, COMM, ENAME
FROM EMP;
```

6.WAQTD Dept Names, Locations.

```
SELECT DNAME, LOC
FROM DEPT;
```

Note: Whenever we want to display any columns or expression along with the table in the result we have to use the following syntax in the place of asterisk (*).

Syntax:

```
TABLE_NAME. *
```

7.WAQTD Details of all the employees along with the employee numbers.

```
SELECT EMP.*, EMPNO
FROM EMP;
```

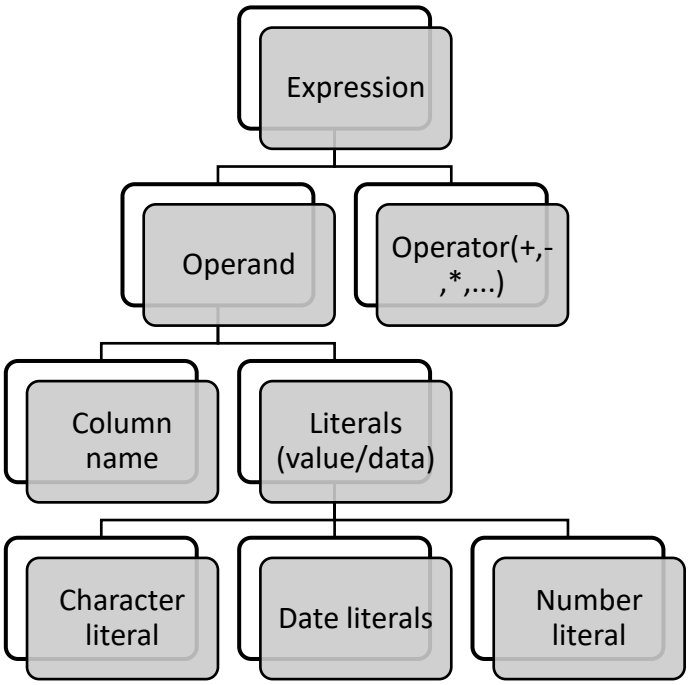
8.WAQTD Details of the departments along with DEPT Names and Locations.

SELECT DEPT.*, DNAME, LOC

FROM DEPT;

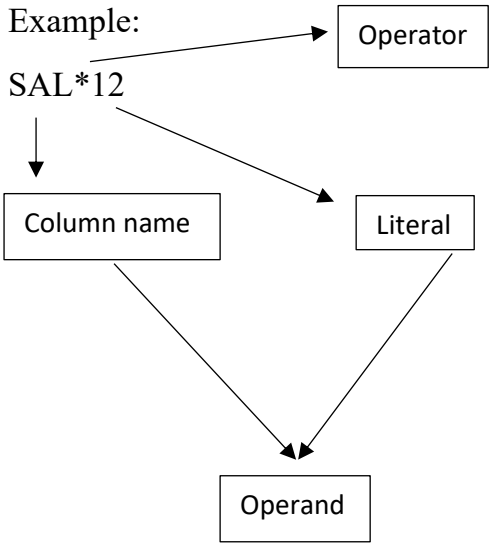
EXPRESSION

The statement which gives the result is known as expression.



Note: CHARACTER AND DATE LITERALS MUST BE ENCLOSED WITHIN SINGLE QUOTES (‘’)

Example:



1.WAQTD annual salaries employees.

SELECT SAL*12

FROM EMP;

ALIAS

Alias is an alternative name which is given for a column/expression.

Alias name can be used to with or without using the keyword AS.

Alias name can be a single word or a string, String should be enclosed within (“ ”) double quotes or enclosed within double quotes.

1.WAQTD Annual salary of employees (By using ALIAS name).

SELECT SAL*12 AS ANNUAL_SALARY

FROM EMP;

OR

SELECT SAL*12 “ANNUAL_SALARY”

FROM EMP;

OR

SELECT SAL*12 “ANNUAL_SALARY”

FROM EMP;

2.WAQTD to give ALIAS name to Sal column.

SELECT SAL SALARY

FROM EMP;

DISTINCT Clause

Distinct clause is used to remove the duplicate records present in the table.

Distinct clause has to be used as the first argument in select clause.

Whenever we pass multiple arguments to distinct clause it always removes the combination of columns that are duplicated.

Example 1:

EMPNO	ENAME	SAL	DEPTNO
1	ALLEN	2000	20
2	BLAKE	3000	10
3	SMITH	9000	30
4	SCOTT	2000	20

5	JAMES	4000	30
6	BLAKE	3000	10

1.WAQTD different dept no's present in employee table.

SELECT DISTINCT DEPTNO

FROM EMP;

DEPTNO
10
20
30

2.WAQTD Different salaries and dept no's present in employee table.

SELECT DISTINCT SAL, DEPTNO

FROM EMP;

SAL	DEPTNO
2000	20
3000	10
9000	30
4000	30

SELECTION

The retrieval of data from the table by selecting both columns and records is known as Selection.

Syntax:

```
SELECT */[DISTINCT] COL_NAME/EXPRESSION [ALIAS]
FROM TABLE_NAME
WHERE <filter condition>;
```

Order of execution

1.FROM

2.WHERE – Row by row

3.SELECT – Row by row

- WHERE clause is used to filter the records.
- WHERE clause executes row by row.
- WHERE clause executes after the execution of FROM clause.
- For WHERE clause we have to pass filter condition as an argument.
- For WHERE clause we can pass multiple filter conditions by using logical operators.

Example 1:

EMPNO	ENAME	SAL	DEPTNO
1	ALLEN	2000	20
2	BLAKE	3000	10
3	SMITH	9000	30
4	SCOTT	2000	20
5	JAMES	4000	30
6	BLAKE	3000	10

WAQTD Employee names and their dept no's where the employees should be working in dept no 20.

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO=20;
```

Execution

EMPNO	ENAME	SAL	DEPTNO
1	ALLEN	2000	20
2	BLAKE	3000	10
3	SMITH	9000	30
4	SCOTT	2000	20
5	JAMES	4000	30
6	BLAKE	3000	10

Output of FROM Clause

EMPNO	ENAME	SAL	DEPTNO
1	ALLEN	2000	20
4	SCOTT	2000	20

Output of WHERE Clause

ENAME	DEPTNO
ALLEN	20
SCOTT	20

Output of SELECT Clause

2.WAQTD Details of an employee whose name is SMITH.

```
SELECT *
FROM EMP
WHERE ENAME='SMITH';
```

3.WAQTD Employee names and their salary where the employees should be earning more than 1250.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL>1250;
```

Questions on WHERE clause

1.WAQTD THE ANNUAL SALARY OF THE EMPLOYEES WHOSE NAME IS ALLEN.

2.WAQTD NAMES OF THE EMPLOYEES WORKING AS CLERK.

3.WAQTD SALARY OF THE EMPLOYEES WORKING AS SALESMAN.

4.WAQTD DETAILS OF THE EMPLOYEES WHO EARNS MORE THAN 2000.

5.WAQTD DETAILS OF THE EMPLOYEES WHOSE NAME IS JONES.

6.WAQTD DETAILS OF THE EMPLOYEES WHO HAS HIRED AFTER 01-JAN-81.

7.WAQTD NAME AND SALARY ALONG WITH THEIR ANNUAL SALARY IF THE ANNUAL SALARY IS MORE THAN 12000.

8.WAQTD EMPNO OF THE EMPLOYEE WHO ARE WORKING IN DEPTNO 30.

9.WAQTD NAME AND HIREDATE IF THEY ARE HIRED BEFORE 1981.

10.WAQTD DETAILS OF THE EMPLOYEES WORKING AS MANAGER.

OPERATORS IN SQL

1. Arithmetic operator (+, -, *, /, ……)
2. Relational operator (<, <=, >, >=)
3. Comparison operator (=, !=)
4. Concatenation operator (||)
5. Logical operator (AND, OR, NOT)
6. Special operator (IN, NOT IN, BETWEEN, NOT BETWEEN, NOT BETWEEN, IS, IS NOT, LIKE, NOT LIKE)
7. Subquery operator (ALL, ANY, EXIST, NOT EXIST)

CONCATENATION OPERATOR

CONCATENATION operator is used to join the given two strings or columns.

Example:

1.WAQTD to join employee names and salary column.

```
SELECT ENAME||SAL
FROM EMP;
```

Example format 1:

EMPLOYEE SMITH SALARY IS 800

```
SELECT 'EMPLOYEE'||' '||ENAME||' '||'SALARY IS'||SAL
FROM EMP;
```

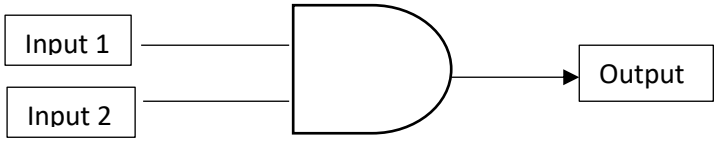
Example format 2:

THE EMPLOYEE SMITH DEPARTMENT NO IS 20.

```
SELECT 'THE EMPLOYEE'||' '||ENAME||' '||'DEPARTMENT NUMBER IS'||'
'||DEPTNO
FROM EMP;
```

LOGICAL OPERATOR

1. AND: AND operator returns true only if all the conditions are true. AND operator is used between the conditions.



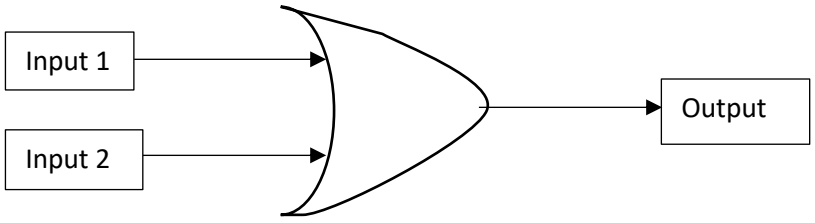
TRUTH TABLE

INPUT 1	INPUT 2	OUTPUT
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

AND Operator follows Binary multiplication.

INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

2. OR: OR operator returns true even if any of the conditions any one of the conditions is true. OR operator is used between the conditions.



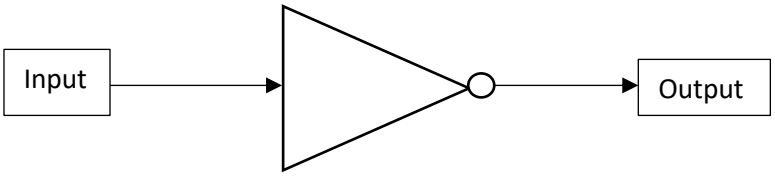
TRUTH TABLE

INPUT 1	INPUT 2	OUTPUT
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

OR operator follows Binary addition.

INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

3. NOT: NOT operator is used as negation. NOT operator will accept only one input. If we pass true, it will return false Similarly if we pass false, it will return true.



TRUE→FALSE

FALSE→TRUE

0→1

1→0

<u>Questions on logical operators</u>

1.WAQTD DETAILS OF THE EMPLOYEES WORKING AS CLERK AND EARNING LESS THAN 1500.

2.WAQTD DETAILS OF THE EMPLOYEES ALONG WITH THE ANNUAL SALRIES IF THEY ARE WORKING IN DEPT 30 AS SALESMAN AND THEIR ANNUAL SALRY HAS TO BE GREATER THAN 14000.

3.WAQTD ALL THE DETAILS OF THE EMPLOYEES WORKING IN DEPT 30 OR AS AN ANALYST.

4.WAQTD NAMES OF THE EMPLOYEES WHOSE SALAEY IS LESS THAN 1100 AND THEIR DESIGNATION IS CLERK.

5.WAQTD NAME AND SAL, ANNUAL SALARY AND DEPT NO IF DEPTNO IS 20 EARNING MORE THAN 1100 AND ANNUAL SALARY EXCEEDS 12000.

6.WAQTD EMPNO AND NAMES OF THE EMPLOYEES WORKING AS MANAGER IN DEPT 20.

7.WAQTD NAME AND HIREDATE OF EMPLOYEES WORKING AS MANAGER IN DEPT 30.

8.WAQTD DETAILS OF EMPLOYEES WORKING IN DEPT 20 OR 30.

9.WAQTD DETAILS OF EMPLOYEES WORKING AS ANALYST IN DEPT NO 10.

10.WAQTD DETAILS OF EMPLOYEES WORKING AS PRESIDENT WITH SALARY OF 4000.

SPECIAL OPERATORS

1. **IN OPERATOR**
   IN operator is similar to = operator. IN operator is a multi-value operator which can accept multiple values at the right-hand side.
   Syntax:
   COL_NAME/EXPRESSION IN (V1, V2, V3, ……, Vn)

Example 1: WAQTD Employee names and their dept no where the employees should be working in dept no 10 or 20.

Normal method

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO=10 OR DEPTNO=20;
```

By using IN operator

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO IN (10,20);
```

2. **NOT IN OPERATOR**
   NOT IN operator is similar to != operator. NOT IN operator is similar to IN operator where it rejects the value instead of selecting it.
   Syntax:
   COL_NAME/EXPRESSION NOT IN (V1, V2, V3, ……, Vn)

Example 1: WAQTD Employee names and their dept no where the employees should not be working in dept no 10 or 20.

Normal method

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO! =10 AND DEPTNO! =20;
```

By using IN operator

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO NOT IN (10,20);
```

3. **BETWEEN OPERATOR**

BETWEEN operator is used whenever we have range of values. BETWEEN operator works including ranges. The ranges cannot be interchanged.

Syntax:

```
COL_NAME/EXPRESSION BETWEEN lower range AND higher range;
```

Example 1:

WAQTD Employee names and their hire date where the employees should be hired after 1981 and before 1983.

Normal method

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE>'31-DEC-81' AND HIREDATE<'01-JAN-83';
OR
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE>='01-JAN-82' AND HIREDATE<='31-DEC-82';
```

By using BETWEEN Operator

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-82' AND '31-DEC-82';
```

2.WAQTD Employee names and their salaries where the employees should not be earning more than 3000 and less than 5000.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 3001 AND 4999;
```

4. **NOT BETWEEN OPERATOR**

NOT BETWEEN Operator is similar to BETWEEN operator where it rejects the value instead of selecting it.

Syntax:

```
COL_NAME/EXPRESSION NOT BETWEEN lower range AND upper range;
```

Example 1:

WAQTD Employee names and their hire date where the employees should not be hired in the year 1982.

Normal method

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE<='31-DEC-81' AND HIREDATE<='01-JAN-83';
```

OR

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE>'01-JAN-82' AND HIREDATE<'31-DEC-82';
```

By using BETWEEN Operator

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE NOT BETWEEN '01-JAN-82' AND '31-DEC-82';
```

5. **IS OPEARTOR**

IS Operator is used to compare with only NULL values.

Syntax:

```
COL_NAME/EXPRESSION IS NULL;
```

Example 1:

WAQTD details of the employees who is not earning COMMISSION.

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

2.WAQTD details of the employees who is not reporting to any managers.

```
SELECT *
FROM EMP
WHERE MGR IS NULL;
```

6. **IS NOT OPERATOR**

IS NOT Operator is similar to IS operator where it rejects the vale instead of selecting it.

Syntax:

COL_NAME/EXPRESSION IS NOT NULL;

Example 1:

WAQTD details of the employees who is earning COMMISSION.

```
SELECT *
FROM EMP
WHERE COMM IS NOT NULL;
```

2.WAQTD details of the employees who is reporting to managers.

```
SELECT *
FROM EMP
WHERE MGR IS NOT NULL;
```

7. LIKE OPERATOR

LIKE operator is used to perform pattern matching.

To achieve pattern matching we use the following special characters.

% - It can accept any no of characters, any number of times and no characters.

_ (underscore) – It can accept exactly one character which can be any character.

Note: % and _ are also known as wild characters.

Example 1:

WAQTD emp names whose name starts with char A.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE 'A%';
```

2.WAQTD emp names whose name ends with char R.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '%R';
```

3.WAQTD emp names whose name has char A in it.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '%A%';
```

4.WAQTD emp names whose name has character A as the third letter in it.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '_ _A%';
```

5.WAQTD emp names whose name has char R as the third letter and total characters in the name should be 6.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '_ _ R _ _ _';
```

8. NOT LIKE OPERATOR

NOT LIKE operator is similar to LIKE operator where it rejects the value instead of selecting it.

Example 1:

WAQTD emp names whose does not name start with char A.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE 'A%';
```

2.WAQTD emp names whose name does not end with char R.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE '%R';
```

Questions on special operator

1.WAQTD names, dept no's and jobs of the employees working as clerks in dept no 10 or 20.

2.WAQTD details of employees working as CLERK or MANAGER in dept.

3.WAQTD names of the employees working in dept no 10 or 20 and their names should not have character A int it.

4. WAQTD details of the employees with Emp no 7902, 7839.

5.WAQTD details of the employees working as MANAGER or SALESMAN or CLERK.

6.WAQTD names of the employees hired after 81 and before 87 and their name should not start with char A.

7.WAQTD details of the employees earning more than 1250 but less than 3000.

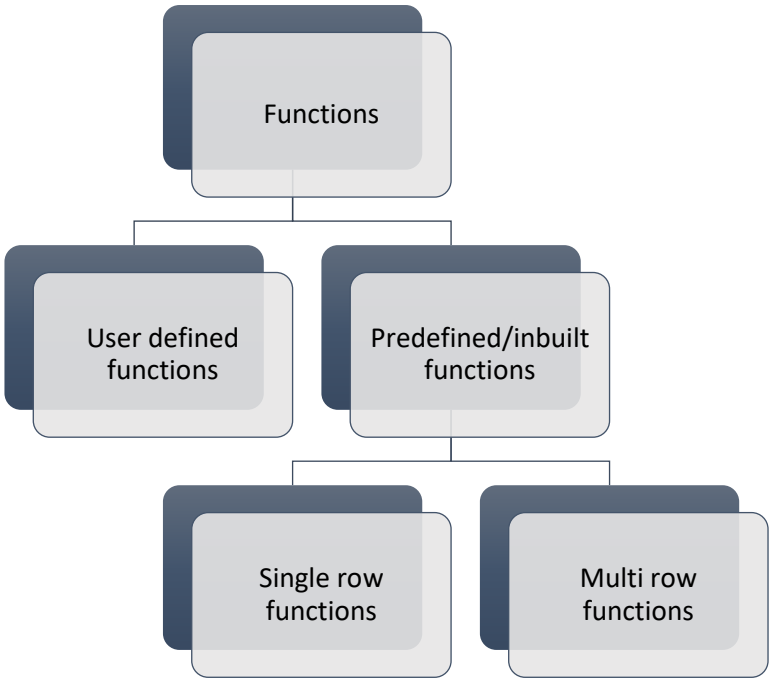
8.WAQTD names of the employees hired after 81 into DEPT 10 or 30.

9. WAQTD names of the employees along with annual salary for the employees working as MANAGER or CLERK into dept 10 or 30.

10.WAQTD all the details of the employees along with the ANNUAL SALARY if salary between 1000 and 4000 and annual salary more than 15000.

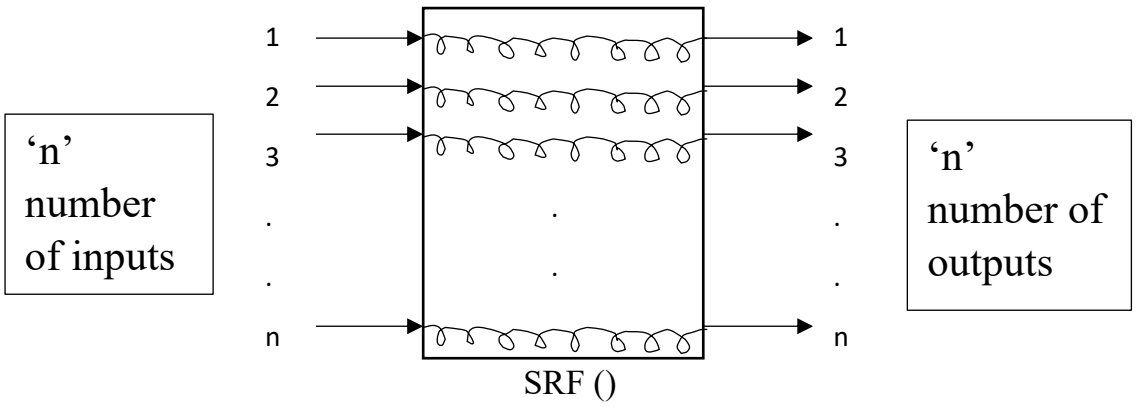
FUNCTIONS

Functions are the block of code which is used to perform a particular task.



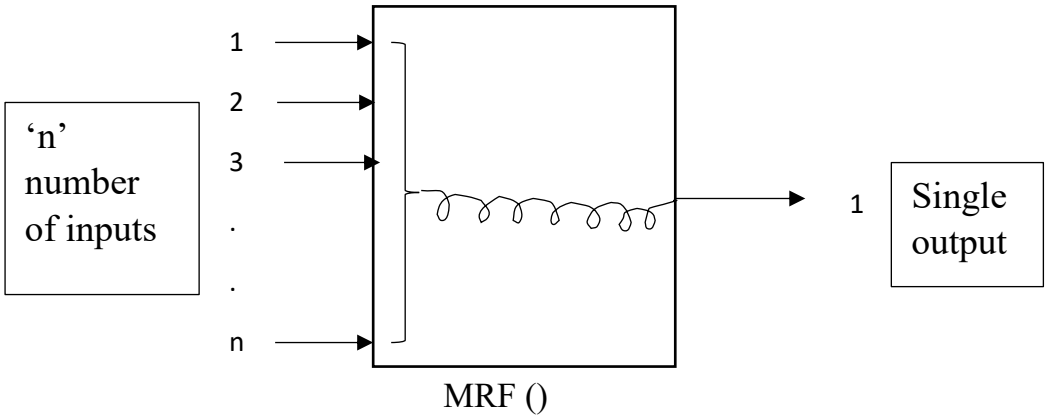
1. **SINGLE ROW FUNCTIONS**
    - Single row functions executes row by row.
    - It takes an input, starts the execution and gives you an output and then it goes for next input.

- If we pass ‘n’ number of inputs to a single row functions, it will return ‘n’ number of outputs.



2. MULTI ROW FUNCTIONS

1. Multi row functions are also known as GROUP Functions/ Aggregate functions.
2. It takes all the inputs at once aggregates them and starts the executions, gives you only one output.
3. If you pass ‘n’ number of inputs to the multi row functions, it will return only one output.



LIST OF MULTI ROW FUNCTIONS

1. MIN ()
2. MAX ()
3. SUM ()
4. AVG ()
5. COUNT ()

Rules of using MRF ()

- Multi row functions can accept only a single argument that can be COL_NAME or an expression.
- MAX () and MIN () can accept all the datatype columns.
- SUM () and AVG () can accept only numbered datatype columns.
- Multi row functions will ignore NULL values.
- We cannot use multi row functions in WHERE clause.
- We cannot use any columns along with multi row functions in SELECT clause.
- COUNT () is the only multi row function which accepts * as an argument.

Examples:

1.WAQTD no of employees, maximum salary, minimum commission, average salary and total commission of all the employees.

```
SELECT COUNT (*), MAX(SAL), MIN(COMM), AVG(SAL), SUM(COMM)
FROM EMP;
```

2.WAQTD no of employees getting salary less than 2000 in dept no 10.

```
SELECT COUNT (*)
FROM EMP
WHERE SAL<2000 AND DEPTNO=10;
```

3.WAQTD total salary needed to pay employees working as CLERK.

```
SELECT SUM(SAL)
FROM EMP
WHERE JOB='CLERK';
```

4.WAQTD average salary needed to pay all the employees.

```
SELECT AVG(SAL)
FROM EMP;
```

5.WAQTD no of employees having 'A' as their first character.

```
SELECT COUNT (*)
FROM EMP
WHERE ENAME LIKE 'A%';
```

6.WAQTD no of employees working as clerk or manager.

```
SELECT COUNT (*)
```

FROM EMP

WHERE JOB IN ('CLERK','MANAGER');

7.WAQTD total salary needed to pay employees hired in February.

SELECT SUM(SAL)

FROM EMP

WHERE HIREDATE LIKE '%FEB%';

8.WAQTD no of employees reporting to 7839.

SELECT COUNT (*)

FROM EMP

WHERE MGR IN 7839.

9.WAQTD no of employees getting commission in dept no 30.

SELECT COUNT (*)

FROM EMP

WHERE COMM IS NOT NULL AND DEPTNO=30;

10.WAQTD no of employees having character A in their names.

SELECT COUNT (*)

FROM EMP

WHERE ENAME LIKE '%A%';

GROUP BY CLAUSE

Syntax:

```
SELECT GROUP FUNCTION/GROUP BY EXPRESSION
FROM EMP
WHERE <filter condition>
GROUP BY COLUMN_NAME/EXPRESSION;
```

Order of execution

1.FROM

2.WHERE CLAUSE – row by row

3.GROUP BY CLAUSE – row by row

4.SELECT CLAUSE – group by group

1.GROUP BY clause is used to group the records.

2.GROUP BY clause executes row by row.

3.After the execution of GROUP BY clause it will create groups.

4.Any clause which executes after the execution of GROUP BY clause will be executed group by group.

5.We can use GROUP BY expression along with the multi row function in SELECT clause.

GROUP BY EXPRESSION

Any column name or expression which is written in inside GROUP BY clause is known GROUP BY EXPRESSION.

Example

1.WAQTD maximum and minimum salary of the employees where the employees should not be earning rupees 4000 in each dept.

```
SELECT MAX(SAL), MIN(SAL)

FROM EMP

WHERE SAL NOT IN 4000

GROUP BY DEPTNO;
```

2.WAQTD no of employees and job of all the employees present in each job.

```
SELECT COUNT (*), JOB

FROM EMP

GROUP BY JOB;
```

Questions on GROUP BY clause

1.WAQTD no of employees working in each dept except 'PRESIDENT'.

2.WAQTD total salary needed to pay all the employees in each dept.

3.WAQTD no of employees working as manager in each dept.

4.WAQTD average salary needed to pay all the employees in each dept excluding the employees of dept no 20.

5.WAQTD no of employees having char A in their names in each job.

6.WAQTD no of employees and average salary needed to pay the employees whose salary is greater than 2000 in each dept.

7.WAQTD no of employees and total salary given to all the salesman in each dept.

8.WAQTD no of employees with their maximum salary in each job.

9.WAQTD maximum salaries given to all employee working in each dept.

10.WAQTD number of times the salary are present in employee table.

HAVING clause:

Syntax:

```
SELECT GROUP FUNCTION/GROUP BY EXPRESSION
FROM TABLE
WHERE <filter condition>
GROUP BY COLUMN_NAME/EXPRESSION
HAVING <group filter condition>;
```

Order of execution

1. **FROM**
2. **WHERE – row by row**
3. **GROUP BY – row by row**
4. **HAVING – group by group**
5. **SELECT – group by group**

1.HAVING clause is used to filter the groups.

2.HAVING clause executes group by group.

3.HAVING clause executes after the execution of GROUP BY clause.

4.since HAVING clause executes after the GROUP BY clause, we have to write group filter condition.

5.HAVING clause cannot be used without using GROUP by clause.

6.For HAVING clause, we can pass multi row functions as an argument.

Examples:

1.WAQTD department numbers if there are at least 2 employees working in each department except the employee who earns 4000.

```
SELECT DEPTNO
FROM EMP
WHERE SAL NOT IN 4000
GROUP BY DEPTNO
HAVING COUNT (*)>=2;
```

2.WAQTD maximum salary and minimum salary of all employees where the employees should be earning more than 1250, their maximum salary should more than 2900 and their minimum salary should be less than 1500 in each dept.

```
SELECT MAX(SAL), MIN(SAL)
FROM EMP
WHERE SAL>1250
GROUP BY DEPTNO
HAVING MAX(SAL)<2900 AND MIN(SAL)>1500;
```

Questions on HAVING clause

1.WAQTD dept no and no of employees working in each department if there are at least 2 clerks in each dept.

```
SELECT DEPTNO, COUNT (*)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT (*)>=2;
```

2.WAQTD dept no and total salary needed to pay all the employees in each department if there are at least 4 employees in each dept.

```
SELECT DEPTNO, SUM(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT (*)>=4;
```

3.WAQTD no of employees earning salary more than 1200 in each job and the total salary needed to pay employees of each job must exceed 3500.

```
SELECT COUNT (*), JOB
FROM EMP
WHERE SAL>1200
GROUP BY JOB
HAVING SUM(SAL)>3500;
```

4.WAQTD dept no and number of employees working only if there are 2 employees working in each dept as MANAGER.

```
SELECT DEPTNO, COUNT (*)
```

```
FROM EMP
WHERE JOB='MANAGER'
GROUP BY DEPTNO
HAVING COUNT (*)>=2;
```

5. WAQTD job and maximum salary of employees in each job if the maximum salary exceeds 2600.

```
SELECT JOB, MAX(SAL)
FROM EMP
GROUP BY JOB
HAVING MAX(SAL)>2600;
```

6.AWQTD the salaries which are duplicated in emp table.

```
SELECT SAL
FROM EMP
GROUP BY SAL
HAVING COUNT (*)>1;
```

7.WAQTD the hire dates which are duplicated in emp table.

```
SELECT HIREDATE
FROM EMP
GROUP BY HIREDATE
HAVING COUNT (*)>1;
```

8.WAQTD average salary of each department if average is less than 300.

```
SELECT AVG(SAL), DEPTNO
FROM EMP
GROUP BY DEPTNO
HAVING AVG(SAL)<300;
```

9.WAQTD dept no if there are at least 3 employees in each dept whose name has char 'A' or 'S'.

```
SELECT DEPTNO
FROM EMP
```

WHERE ENAME LIKE '%A%' OR ENAME LIKE '%S%'

GROUP BY DEPTNO

HAVING COUNT (*)>=3;

10.WAQTD minimum and maximum salaries of each job if minimum salary is more than 1000 and maximum salary is less than 5000.

SELECT MIN(SAL), MAX(SAL), JOB

FROM EMP

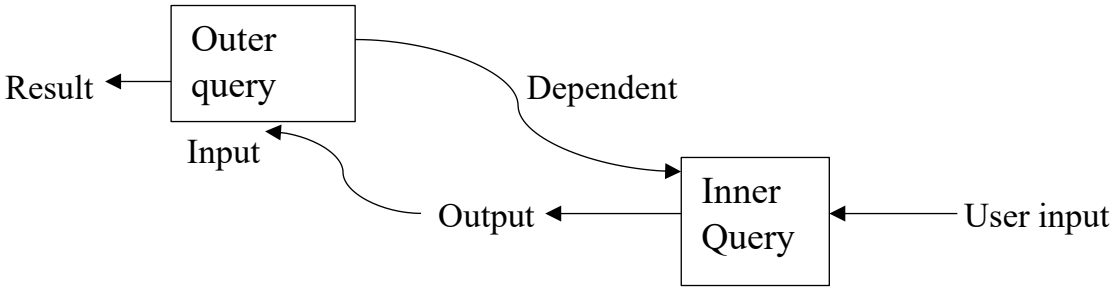
GROUP BY JOB

HAVING MIN(SAL)>1000 AND MAX(SAL)<5000;

SUBQUERY

A query which is written inside another query is known as Subquery.

Working procedure of subquery



- In a subquery we should have minimum of 2 queries.
  - Outer query
  - Inner query
- Inner query starts the execution and gives you an output.
- The output of the inner query is given as an input to the outer query.
- After getting the input from the inner query, outer query starts the execution and generates the result.
- Therefore, we can say that outer query is dependent on inner query (because the input for the outer query is given by the inner query).

Case 1: Whenever there are unknowns in the question we use subquery.

Example 1: WAQTD emp names and their salary where the employees should be earning more than SMITH.

SELECT ENAME, SAL

FROM EMP

```
WHERE SAL> (SELECT SAL
            FROM EMP
            WHERE ENAME='SMITH');
```

2.WAQTD details of emp where the employees should be working in the same designation of MILLER.

```
SELECT *
FROM EMP
WHERE JOB= (SELECT JOB
            FROM EMP
            WHERE ENAME='MILLER');
```

Questions on subquery

1.WAQTD names of the employees earning more than ADAMS.

2.WAQTD name and salary of the employees earning less than KING.

3.WAQTD name and dept no of the employees if they are working in the same dept no as JONES.

4.WAQTD name and job of all the employees working in the same designation as JAMES.

5.WAQTD emp no and name along with annual salary of the employees if their annual salary is greater than WAQRDS annual salary.

6.WAQTD name and hire date of the employees if they are hired before TURNER.

7.WAQTD name and hire date of the employees if they hired after the president.

8.WAQTD name and salary of the employees if they are earning salary less than employees whose emp is 7839.

9.WAQTD all the details of the employees if the employees are hired before MILLER.

10.WAQTD ename and empno of the employees if employees are earning more than ALLEN.

11.WAQTD employee names, emp salary and designation where the employees should be earning more than SMITH and less than KING.

12.WAQTD details of all the employees where the employees should be hired before TURNER.

13.WAQTD employee name and salary of all the employees who are earning more than MILLER but less than ALLEN.

14.WAQTD all the details of the employees working in dept 20 and working in the same designation as SMITH.

15.WAQTD all the details of the employees working as the manager in the same dept as TURNER.

Case 2: Whenever the data to be selected and condition to be executed are present in two different tables then we use subquery.

Examples

1.WAQTD employee name and their dept no where the employee should be working in accounting dept.

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE DNAME='ACCOUNTING');
```

2.WAQTD dept name and location of SMITH.

```
SELECT DNAME, LOC
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE ENAME='SMITH');
```

Questions on subquery case 2

1.WAQTD dept name of the employees whose name is SMITH.

2.WAQTD dept name and location of the employees whose name is KING.

3.WAQTD location of the employees whose employee number is 7902.

4.WAQTD dept name and location along with Dept number of the employee whose name ends with R.

5.WAQTD dept name of the employees whose designation is PRESIDENT.

6.WAQTD names of the employees working in ACCOUNTING dept.

7.WAQTD emp name and salaries of the employees who are working in location CHICAGO.

8.WAQTD details of the employees working in SALES.

9.WAQTD details of the employees along with the annual salary if the employees are working in New York.

10.WAQTD name of the employees working in OPERATIONS department.

11.WAQTD details of the employees where the employees should be working in SALES dept, should be hired before TURNER and should be working as CLERK.

12.WAQTD names of the employees earning more than SCOTT in ACCOUNTING dept.

13.WAQTD details of the employees working as manager in the location CHICAGO.

14.WAQTD details of the employees working as SALESMAN in SALES department.

15.WAQTD name and salary of the employees earning more than KING in ACCOUNTING department.

TYPES OF SUBQUERY

1.Single row subquery

- If a subquery returns exactly one output then we can say that it is single row subquery.
- For single row subquery, we can use normal operators as well as special operators.

```
SELECT *
FROM EMP                [30]
WHERE DEPTNO = (SELECT DEPTNO
   [DEPTNO=30]      FROM EMP
                    WHERE DNAME='SALES');
```

2.Multi row subquery

- If a subquery returns more than one output then we can say that it is a multirow subquery.
- For a multi row subquery, we cannot use normal operators but we can use special operators as well as subquery operators.

```
SELECT *
FROM EMP                [30, 20]
```

```
WHERE DEPTNO IN (SELECT DEPTNO
[DEPTNO IN (30,20)]  FROM EMP
                     WHERE LOC IN ('CHICAGO','DALLAS'));
```

SUBQUERY OPERATOR

1.ALL OPERATOR

ALL is a special operator which is used along with the relational operator which will allow the values present at the LHS to be compared with all the values present at the RHS.

Syntax:

COL_NAME/EXPRESSION <Relational operator> ALL (V1,V2,V3,…..Vn)

2.ANY OPERATOR

ANY is a special operator which is used along with the relational operator which will allow the values present at the LHS to be compared with any of the values present at the RHS.

Syntax:

COL_NAME/EXPRESSION <Relational operator> ANY (V1,V2,V3,…..Vn)

Examples:

1.WAQTD emp names and their salary where the emp should be earning more than the salesman.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL>ALL (SELECT SAL
               FROM EMP
               WHERE JOB IN 'SALESMAN');
```

2.WAQTD details of the employees who earns salary less than at least a CLERK.

```
SELECT *
FROM EMP
```

```
WHERE SAL<ANY (SELECT SAL
               FROM EMP
               WHERE JOB = 'CLERK');
```

3.WAQTD details of the employees who earns more than the manager and should be working ACCOUNTING or RESEARCH dept.

```
t. SELECT *
FROM EMP
WHERE SAL > ALL (SELECT SAL
                 FROM EMP
                 WHERE JOB IN 'MANAGER' AND
DEPTNO IN (SELECT DEPTNO
           FROM EMP
           WHERE DNAME IN ('ACCOUNTING','RESEARCH'));
```

Questions on SUBQUERY Operators.

1.WAQTD name of the employees earning salary more than salesman.

2.WAQTD details of the employees hired after all the CLERK.

3.WAQTD name and salary for all the employees if they are earning less than at least a MANAGER.

4.WAQTD name and hire date of the employees hired before all the MANAGERS.

5.WAQTD names of all the employees hired after all the managers and earning salary more than all the CLERK.

6.WAQTD details of the employees working as a CLERK and hired before at least a SALESMAN.

7.WAQTD details of the employees working in ACCOUNTING or SALES dept.

8.WAQTD department names of the employees with name KING and MILLER.

9.WAQTD details of the employees working in NEW YORK or CHICAGO.

10.WAQTD emp names if employees are hired after all the employees of dept no 10.

USING MULTI ROW FUNCTIONS IN SUBQUERY

Examples:

WAQTD emp names and their salary where the employee should be earning the maximum salary.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL IN (SELECT MAX(SAL)
 FROM EMP);
```

Questions on Multi row functions in subquery

1.WAQTD details of the employees earning least annual salary.

```
SELECT *
FROM EMP
WHERE SAL*12 IN (SELECT MIN(SAL*12)
                 FROM EMP);
```

2.WAQTD name and dept no of the employees earning minimum salary.

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE SAL IN (SELECT MIN(SAL)
              FROM EMP);
```

3.WAQTD name and hire date of the employees hired before all the employees.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE IN (SELECT MIN(HIREDATE)
                   FROM EMP);
```

4.WAQTD name and hire date of the employees hired at the last.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE IN (SELECT MAX(HIREDATE)
                   FROM EMP);
```

5.WAQTD name, commission of the employees who earns minimum commission.

```
SELECT ENAME, COMM
FROM EMP
WHERE COMM IN (SELECT MIN(COMM)
                FROM EMP);
```

6.WAQTD name, salary and commission of the employees earning maximum commission.

```
SELECT ENAME, SAL, COMM
FROM EMP
WHERE COMM IN (SELECT MAX(COMM)
                FROM EMP);
```

7.WAQTD details of employees who has greatest emp no.

```
SELECT *
FROM EMP
WHERE EMPNO IN (SELECT MAX(EMPNO)
                 FROM EMP);
```

8.WAQTD details of the employees having least hire date.

```
SELECT *
FROM EMP
WHERE HIREDATE IN (SELECT MIN(HIREDATE)
                    FROM EMP);
```

9.WAQTD details of the employees earning maximum annual salary.

```
SELECT *
FROM EMP
WHERE SAL*12 IN (SELECT MAX(SAL*12)
                 FROM EMP);
```

NESTED SUBQUERY

A subquery which is written inside another subquery is known as nested subquery.

We can nest up to 255 subqueries.

Example

1.WAQTD third maximum salary.

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

2.WAQTD emp names and their salary who is earning second maximum salary.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL IN (SELECT MAX(SAL)
FROM EMP
WHERE SAL< (SELECT MAX(SAL)
FROM EMP));
```

3.WAQTD third minimum salary.

```
SELECT MIN(SAL)
FROM EMP
WHERE SAL> (SELECT MIN(SAL)
FROM EMP
WHERE SAL> (SELECT MIN(SAL)
```

```
FROM EMP));
```

4.WAQTD name of the employees earning 3rd maximum salary.

```
SELECT ENAME
FROM EMP
WHERE SAL< (SELECT MAX(SAL)
FROM EMP
WHERE SAL< (SELECT MAX(SAL)
FROM EMP
WHERE SAL< (SELECT MAX(SAL)
FROM EMP)));
```

5.WAQTD dept names of an employee who is earning 2nd maximum salary.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (SELECT DEPTNO
FROM EMP
WHERE SAL IN (SELECT MAX(SAL)
FROM EMP
WHERE SAL< (SELECT MAX(SAL)
FROM EMP)));
```

6.WAQTD details of the employees who was hired second.

```
SELECT *
FROM EMP
WHERE HIREDATE IN (SELECT MIN(HIREDATE)
FROM EMP
WHERE HIREDATE> (SELECT MIN(HIREDATE)
FROM EMP));
```

7.WAQTD name of the employee hired before the last employee.

```
SELECT ENAME
FROM EMP
```

```
WHERE HIREDATE IN (SELECT MAX(HIREDATE)
FROM EMP
WHERE HIREDATE< (SELECT MAX(HIREDATE)
FROM EMP));
```

8.WAQTD location of the employees who hired first.

```
SELECT LOC
FROM EMP
WHERE DEPTNO IN (SELECT DEPTNO
FROM EMP
WHERE HIREDATE IN (SELECT MIN(HIREDATE)
FROM EMP));
```

EMPLOYEE MANAGER RELATIONSHIP

To find the employee's manager.

Step 1: Find the MGR of the employee.

Step 2: Compare the MGR with EMPNO

Step 3: Find the particular field asked in the question.

To find the employees reporting to the given manager.

Step 1: Find EMPNO of the given manager.

Step 2: Compare the EMPNO with MGR of the employees.

Step 3: Find the particular field asked in the question.

Examples

1.WAQTD SMITH's manager name.

```
SELECT ENAME
FROM EMP
WHERE EMPNO IN (SELECT MGR
FROM EMP
WHERE ENAME='SMITH');
```

2.WAQTD emp names reporting to KING.

```
SELECT ENAME
```

```
FROM EMP
WHERE MGR IN (SELECT EMPNO
FROM EMP
WHERE ENAME='KING');
```

3.WAQTD SMITH's managers manager name.

```
SELECT ENAME
FROM EMP
WHERE EMPNO IN (SELECT MGR
FROM EMP
WHERE EMPNO IN (SELECT MGR
FROM EMP
WHERE ENAME='SMITH'));
```

Questions on EMPLOYEE MANAGER RELATIONSHIP

1.WAQTD SCOTT's reporting manager.

2.WAQTD ADAM's manager's manager name.

3.WAQTD dept names of JONES manager.

4.WAQTD MILLER's manager salary.

5.WAQTD location of SMITH's managers manager.

6.WAQTD names of the employees reporting to BLAKE.

7.WAQTD number of employees reporting to KING.

8.WAQTD details of employees reporting to JONE's.

9.WAQTD name of the employees reporting to BLAKE's manager.

10.WAQTD no of employees reporting to FORD's manager.

JOINS

The retrieval of data from multiple tables simultaneously is known as JOINS.

Types of JOINS

1. CARTESIAN JOIN/CROSS JOIN
2. INNER JOIN/EQUI JOIN
3. OUTER JOIN
    a. LEFT OUTER JOIN
    b. RIGHT OUTER JOIN
    c. FULL OUTER JOIN
4. NATURAL JOIN
5. SELF JOIN

1.CARTESIAN JOIN/CROSS JOIN:

- A record from table 1 will be merged with all the records of table 2.
- The total number of columns present in the result table will be the summation of number of columns present in table 1 and table 2.
- The total number of records present in the result table will be the product of the total number of records present in table 1 and table 2.

Syntax:

ANSI: AMERICAN NATIONAL STANDARD INSTITUTE

```
SELECT COL_NAME
FROM TABLE1 CROSS JOIN TABLE2;
```

ORACLE

```
SELECT COL_NAME
FROM TABLE1, TABLE2;
```

Note: Cartesian join is not efficient.

Example

ANSI

```
SELECT *
FROM EMP CROSS JOIN DEPT;
```

ORACLE

```
SELECT *
FROM EMP, DEPT;
```

EMP

EMPNO	ENAME	DEPTNO
1	A	20
2	B	30
3	C	10

DEPT

DEPTNO	DNAME	LOC
10	D1	L1
20	D2	L2
30	D3	L3

RESULT

EMPNO	ENAME	DEPTNO	DEPTNO	DNAME	LOC
1	A	20	10	D1	L1
1	A	20	20	D2	L2
1	A	20	30	D3	L3
2	B	30	10	D1	L1
2	B	30	20	D2	L2
2	B	30	30	D3	L3
3	C	10	10	D1	L1
3	C	10	20	D2	L2
3	C	10	30	D3	L3

2.INNER JOIN

INNER JOIN is used to obtain only matched records.

JOIN CONDITION

It is a condition by which two tables are merged.

Syntax:

```
TABLE_NAME1.COL_NAME=TABLE_NAME2.COL_NAME
```

COL_NAME should be common column present in both the tables.

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME1 INNER JOIN TABLE_NAME2
ON <JOIN CONDITION>;
```

ORACLE

```
SELECT COL_NAME
FROM TABLE_NAME1, TABLE_NAME2
WHERE <JOIN CONDITION>;
```

Example:

ANSI

```
SELECT *
FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO;
```

ORACLE

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

EMP:

EMPNO	ENAME	DEPTNO
1	A	20
2	B	30
3	C	10

DEPT:

DEPTNO	DNAME	LOC
10	D1	L1
20	D2	L2
30	D3	L3

RESULT

EMPNO	ENAME	DEPTNO	DEPTNO	DNAME	LOC
1	A	20	20	D2	L2
2	B	30	30	D3	L3
3	C	10	10	D1	L1

Examples:

1.WAQTD emp names and their dept names.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

2.WAQTD emp names and their dept names where the emp should be working in dept no 20.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO AND EMP.DEPTNO=20;
```

3.WAQTD emp names, employee salary, dept name and location of employees should be earning more salary than SMITH and should be working in location DALLAS.

```
SELECT ENAME, SAL, DNAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO AND SAL> (SELECT SAL
FROM EMP
WHERE ENAME='SMITH') AND LOC='DALLAS';
```

Questions on INNER JOINS

1.WAQTD name of the employees and location for all the employees.

```
SELECT ENAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

2.WAQTD dept name and salary for all the employees working in ACCOUNTING.

```
SELECT DNAME, SAL
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO AND DNAME='ACCOUNTING';
```

3.WAQTD dept names and annual salary of all the employees whose salary is more than 2340.

```
SELECT DNAME, SAL*12
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO AND SAL>2340;
```

4.WAQTD emp name and DNAME for all the employees having char A int their dept names.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO AND ENAME LIKE '%A%';
```

5.WAQTD emp names and dept names for all the employees working as SALESMAN.

```
SELECT ENAME, DNAME
```

FROM EMP, DEPT

WHERE EMP.DEPTNO=DEPT.DEPTNO AND JOB='SALESMAN';

6.WAQTD dept name and job for all the employees whose job and dept name starts with character S.

SELECT DNAME, JOB

FROM EMP, DEPT

WHERE EMP.DEPTNO=DEPT.DEPTNO AND JOB LIKE 'S%' AND DNAME LIKE 'S%';

7.WAQTD dept name and MGR for all the employees reporting 7839.

SELECT DNAME, MGR

FROM EMP, DEPT

WHERE EMP.DEPTNO=DEPT.DEPTNO;

3.OUTER JOIN

OUTER JOIN is used to obtain only the unmatched records.

1.LEFT OUTER JOIN

LEFT OUTER JOIN is used to obtain the matched records along with unmatched records of left table.

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME1 LEFT [OUTER] JOIN TABLE_NAME2
WHERE <JOIN CONDITION>;
```

ORACLE

```
SELECT COL_NAME
FROM TABLE_NAME1, TABLE_NAME2
WHERE TABLE_NAME1.COL_NAME=TABLE_NAME2.COL_NAME (+);
```

Examples:

SELECT *

FROM EMP LEFT OUTER JOIN DEPT

ON EMP.DEPTNO=DEPT.DEPTNO;

OR

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO(+);
```

2.RIGHT OUTER JOIN

RIGHT OUTER JOIN is used to obtain the matched records along with the unmatched records of right table.

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME1 RIGHT [OUTER] JOIN TABLE_NAME2
ON <JOIN CONDITION>;
```

ORACLE

```
SELECT COL_NAME
FROM TABLE_NAME1, TABLE_NAME2
WHERE TABLE_NAME1.COL_NAME (+) =TABLE_NAME2.COL_NAME;
```

Example:

ANSI

```
SELECT *
FROM EMP FULL [OUTER] JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

ORACLE

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO (+) =DEPT.DEPTNO;
```

3.FULL OUTER JOIN

FULL OUTER JOIN is used to obtain the matched records along with the unmatched records of both the tables.

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME1 FULL [OUTER] JOIN TABLE_NAME2
ON <JOIN CONDITION>;
```

Example:

```
SELECT *
FROM EMP FULL OUTER JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO;
```

4.NATURAL JOIN

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME1 NATURAL JOIN TABLE_NAME2;
```

*While performing natural join if there are common columns present in the tables then it will give the output of inner join.

*If there are no common columns present in the tables then it will give the output of CARTESIAN/CROSS JOIN.

Note: In NATURAL JOIN, we don't use any join condition.

Example:

```
1.SELECT *
FROM EMP NATURAL JOIN DEPT;
```

It gives the output of INNER JOIN.

```
2.SELECT *
FROM EMP NATURAL JOIN SALGRADE;
```

It gives the output of CARTESIAN/CROSS JOIN.

5.SELF JOIN

Joining the same two tables or joining the table itself is known as SELF JOIN.

Syntax:

ANSI

```
SELECT COL_NAME
FROM TABLE_NAME T1 JOIN TABLE_NAME T2
ON <JOIN CONDITION>;
```

ORACLE

```
SELECT COL_NAME
FROM TABLE_NAME T1, TABLE_NAME T2
WHERE <JOIN CONDITION>;
```

Note: ALIAS is mandatory in SELF JOIN.

Examples:

1.WAQTD emp names and their managers name.

SELECT E1.ENAME, E2.ENAME MGR_NAME

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO;

2.WAQTD emp names, emp salary, managers name and managers salary where the employees should be earning more than the manager.

SELECT E1.ENAME, E1.SAL, E2.ENAME, E2.SAL

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.SAL>E2.SAL;

3.WAQTD emp names, emp annual salary, manager's name and manager's annual salary where the employee's annual salary should be greater than 1000 and manager's name should have character A in it.

SELECT E1.ENAME, E1.SAL*12 EMP_ANNUAL_SAL, E2.ENAME MGR_NAME, E2.SAL*12 MGR_ANNUAL_SAL

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.SAL*12>1000 AND E2.ENAME LIKE '%A%';

Questions on SELF JOIN

1.WAQTD name of the employees and their manager's name if employees are working as CLERK.

SELECT E1.ENAME, E2.ENAME MGR_NAME

```
FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.JOB='CLERK';
```

2.WAQTD name of the employees and manager's designation if manager's works in dept 10 or 20.

```
SELECT E1.ENAME, E2.JOB MGR_JOB

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E2.DEPTNO IN (10, 20);
```

3.WAQTD name of the employees and manager's salary if employee and manager both earn more than 2300.

```
SELECT E1.ENAME, E2.SAL MGR_SAL

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.SAL>2300 AND E2.SAL>2300;
```

4.WAQTD emp name and manager's hire date if employee was hired before 1982.

```
SELECT E1.ENAME, E2.HIREDATE MGR_HIREDATE

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.HIREDATE<'01-JAN-1982';
```

5.WAQTD emp name and manager's comm if the employee works as SALESMAN and manager works in dept 30.

```
SELECT E1.ENAME, E2.COMM MGR_COMM

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.JOB='SALESMAN' AND
E2.DEPTNO=30;
```

6.WAQTD emp names, manager's name and their salaries if employees earn more than manager.

```
SELECT E1.ENAME, E2.ENAME MGR_NAME, E2.SAL MGR_SAL

FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E1.SAL>E2.SAL;
```

7.WAQTD emp names, hire date, manager's name and manager's hire date if manager was hired before employee.

SELECT E1.ENAME, E1.HIREDATE, E2.ENAME MGR_NAME, E2.HIREDATE MGR_HIREDATE

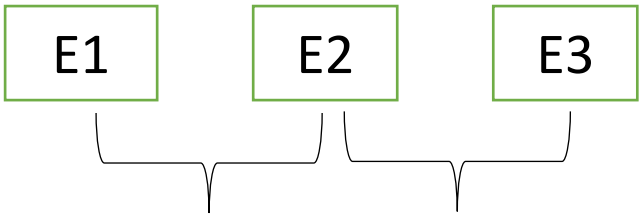
FROM EMP E1, EMP E2

WHERE E1.MGR=E2.EMPNO AND E2.HIREDATE<E1.HIREDATE;

JOINING MORE THAN TWO TABLES

Note: Whenever we join n number of tables, we have to write (n-1) number of join conditions.

1.WAQTD emp names, manager's name and manager manager's name.



E1.MGR=E2.EMPNO E2.MGR=E3.EMPNO

SELECT E1.ENAME, E2.ENAME MGR_NAME, E3.ENAME MGR's_MGR_NAME

FROM EMP E1, EMP E2, EMP E3

WHERE E1.MGR=E2.EMPNO AND E2.MGR=E3.EMPNO;

2.WAQTD emp names, emp salary, manager's name and manager's salary where the employees should be earning less than the manager's salary as well as manager's salary should be earning less than their managers.

SELECT E1.ENAME, E1.SAL, E2.ENAME MGR_NAME, E2.SAL MGR_SAL

FROM EMP E1, EMP E2, EMP E3

WHERE E1.MGR=E2.EMPNO AND E2.MGR=E3.EMPNO AND E1.SAL<E2.SAL AND E2.SAL<E3.SAL;

3.WAQTD manager's salary and manager's manager salary where the manager's salary is less than 2000 and manager's manager should be greater than 3000.

SELECT E2.SAL MGR_SAL, E3.SAL MGR_MGR_SAL

FROM EMP E1, EMP E2, EMP E3

WHERE E1.MGR=E2.EMPNO AND E2.MGR=E3.EMPNO AND E2.SAL<2000 AND E3.SAL>3000;

4.WAQTD emp names, manager's manager name and manager's manager salary where the employees should be earning less than manager's manager salary.

```
SELECT E1.ENAME, E3.ENAME MGR_MGR_NAME, E3.SAL
MGR's_MGR_SAL

FROM EMP E1, EMP E2, EMP E3

WHERE E1.MGR=E2.EMPNO AND E2.MGR=E3.EMPNO and E1.SAL<E3.SAL;
```

ORDER BY CLAUSE

Syntax:

```
SELECT COL_NAME
FROM TABLE_NAME
[WHERE <filter condition>]
[GROUP BY COL_NAME]
[HAVING <group filter condition>]
ORDER BY CLAUSE COL_NAME [ASC]/DESC;
```

Order of execution

1.FROM

2.WHERE

3.GROUP BY

4.HAVING

5.SELECT

6.ORDER BY

- ORDER BY clause is used to sort the records either in ascending or descending order.
- ORDER BY clause has to be written as the last clause in the QUERY.
- ORDER BY is the only clause which executes after the executes after the execution of SELECT clause.
- By default, ORDER BY clause sorts the records in ascending order.

Examples

1.SELECT ENAME

FROM EMP

ORDER BY ENAME;

Names will be sorted in ascending order.

2.SELECT ENAME

FROM EMP

ORDER BY ENAME DESC;

Names will be sorted in descending order.

3.SELECT ENAME, SAL

FROM EMP

ORDER BY ENAME, SAL;

Records will be sorted in ascending order of names.

4.SELECT ENAME, SAL

FROM EMP

ORDER BY SAL, ENAME;

Records will be sorted in ascending order SALARY.

5.SELECT ENAME, SAL

FROM EMP

ORDER BY SAL DESC, ENAME;

Records will be arranged in descending order of salary.

SINGLE ROW FUNCTION

1.MOD ()

2.ROUND ()

3.TRUNC ()

4.LENGTH ()

5.UPPER ()

6.LOWER ()

7.INITCAP ()

8.REVERSE ()

9.SUBSTR ()

10.INSTR ()

11.REPLACE ()

12.TO_CHAR ()

13.TO_DATE ()

14.NVL ()

Note:

DUAL is a dummy table which consists of a single record which is used to display our own values in the result.

1.MOD ()

This function is used to display the modulus of a given number.

Syntax:

```
MOD (m, n)
```

Example: MOD (5, 2) =1

```
SELECT MOD (5, 2)
FROM DUAL;
```

2.ROUND ()

This function is used to round of the given number to its nearest value.

Syntax:

```
ROUND(NUMBER)
```

Ex: ROUND (44.6) = 45

SELECT ROUND (44.6)

FROM DUAL;

3.TRUNC ()

This function is similar to ROUND function but it always rounds the given number to its lowest value.

Syntax:

TRUNC (NUMBER)

Ex: TRUNC (44.9) = 44

SELECT TRUNC (44.9)

FROM DUAL;

4.LENGTH ()

This function is used to obtain total number of CHAR's in the given string.

Syntax:

LENGTH ('STRING')

Ex: LENGTH ('QSPIDERS') = 8

SELECT LENGTH ('QSPIDERS')

FROM DUAL;

SELECT LENGTH (ENAME)

FROM EMP;

5.UPPER ()

This function is used to convert the given value into upper case.

Syntax:

UPPER ('STRING')

Ex: UPPER ('qspiders') =QSPIDERS

6.LOWER ()

This function is used to convert the given value into lower case.

Syntax:

LOWER ('STRING')

Ex: LOWER (‘QSPIDERS’) = qspiders

7.INITCAP ()

This function is used to convert the given string into initial uppercase and remaining letters in lowercase.

Syntax:

INITCAP (‘STRING’)

Ex: INITCAP (‘QSPIDERS’) = Qspiders

8.REVERSE ()

This function is used to reverse the given string.

Syntax:

REVERSE (‘STRING’)

Ex: REVERSE (‘QSPIDERS’) = SREDIPSQ

9.SUBSTR ()

This function is used to extract a part of the given string from the original string.

Syntax:

SUBSTR (‘ORIGINAL STRING’, POSITION, [LENGTH]);

Example:

1.SUBSTR (‘BANGALORE’, 2)

B A N G A L O R E

1 2 3 4 5 6 7 8 9 →

Ans: ANGALORE

2.SUBSTR (‘BANGALORE’,2,4)

B A N G A L O R E

1 2 3 4 5 6 7 8 9

→

Ans: ANGA

10.INSTR ()

This function is used to obtain the position value of the substring if it is present in the original string.

Syntax:

INSTR (‘ORIGINAL STRING’, ‘SUBSTR’, POSITION, [nth OCCURRENCE]);

Note: The default value of position and nth occurrence is 1.

Example:

1.INSTR (‘BANGALORE’, ‘N’, 1)

B A N G A L O R E

1 2 3 4 5 6 7 8 9

Ans: 3

2.INSTR (‘BANGALORE’, ‘A’, 2, 1)

B A N G A L O R E

1 2 3 4 5 6 7 8 9

Ans: 2

3. INSTR (‘BANGALORE’, ‘A’, 2, 2)

B A N G A L O R E

1 2 3 4 5 6 7 8 9

Ans: 5

4. INSTR (‘BANGALORE’, ‘P’, 2, 2)

B A N G A L O R E

1 2 3 4 5 6 7 8 9

Ans: 0

11.REPLACE ()

This function is used to replace a substring with a new string.

Syntax:

REPLACE (‘ORIGINAL STRING’, ‘SUBSTR’, ‘NEWSTRING’);

Example:

1.REPLACE (‘QSPIDERS’, ‘S’, ‘T’);

Ans: QTPIDERT

2.REPLACE (‘QSPIDERS’, ‘SP’, ‘T’);

Ans: QTIDERS

3.REPLACE (‘QSPIDERS’, ‘S’)

Ans: QPIDER

12.TO_CHAR ()

This function is used to convert the given date into string format.

Syntax:

TO_CHAR (‘DATE’, ‘FORMAT_MODEL’);

Format models

YEAR – TWENTY TWENTY FOUR

YYYY – 2024

YY – 24

MONTH – APRIL

MON – APR

MM – 04

DAY – MONDAY

DY – MON

DD – 29

D – 2

HH24 – 23

HH12 – 11

MI – 25

SS – 35

TO_DATE ()

Syntax:

TO_DATE (‘DATE’)

Example:

1.TO_CHAR (SYSDATE, ‘YEAR’)

Ans: TWENTY TWENTY-FOUR

2.TO_CHAR (SYSDATE, ‘YYYY’)

Ans: 2024

3.TO_CHAR (TO_DATE(’04-MAY-2023’), ‘MONTH’)

Ans: MAY

4.TO_CHAR (TO_DATE(’04-MAY-2023’), ‘YYYY, MON, DAY)

Ans: 2023, MAY, THURSDAY

5.WAQTD Emp names who hired on Monday.

```
SELECT ENAME
FROM EMP
WHERE TO_CHAR (HIREDATE, ‘DY’) = ‘MON’;
```

6.WAQTD Emp names who hired in the month of November.

```
SELECT ENAME
FROM EMP
WHERE TO_CHAR (HIREDATE, ‘MON’) = ‘NOV’;
```

14.NVL () - NULL VALUE LOGIC ()

NVL () is used to overcome the drawbacks of operations on NULL.

Syntax:

```
NVL (ARG1, ARG2)
```

- In arg1, we write COL_NAME which can be NULL.
- In arg2, we write a value that will be substituted if the arg1 is NULL.
- If arg1 is not NULL then the same value which is present in the arg1 will be retained.

Example:

1.WAQTD emp names and the total salary of the each and every employee separately. (Without using NVL ()).

```
ENAME, SAL+NVL (COMM,0)
FROM EMP;
```

DATA DEFINITION LANGUAGE (DDL)

It is used to do operations on the structure of the table.

1. *CREATE* – It is used to create the new table.

Syntax:

```
CREATE TABLE table_name
(
COLUMN_NAME1 DATATYPE NOT NULL/[NULL],
COLUMN_NAME2 DATATYPE NOT NULL/[NULL],
COLUMN_NAME3 DATATYPE NOT NULL/[NULL],
.
.
.
COLUMN_NAMEn DATATYPE NOT NULL/[NULL],
CONSTRAINT constraint_ref_name UNIQUE (COLUMN_NAME),
CONSTRAINT constraint_ref_name CHECK (CONDITION),
CONSTRAINT constraint_ref_name PRIMARY KEY (COLUMN_NAME),
CONSTRAINT constraint_ref_name FOREIGN KEY (COLUMN_NAME)
REFERENCES parent_table_name (COLUMN_NAME));
```

Documentation to create TEACHER table.
Table_name: TEACHER

COL_NAME	DATATYPE	NULL/NOT NULL	CONSTRAINTS	CONSTRAINT REF NAME
TID	VARCHAR (2)	NOT NULL	PRIMARY KEY	TID_PK
TNAME	VARCHAR (20)	NOT NULL	-	-
SALARY	NUMBER (7, 2)	NULL	CHECK	SAL_C

Documentation to create STUDENT table.
Table_name: STUDENT

COL_NAME	DATATYPE	NULL/NOT NULL	CONSTRAINTS	CONSTRAINT REF NAME
SID	CHAR (6)	NOT NULL	PRIMARY KEY	SID_PK
SNAME	VARCHAR (20)	NOT NULL	-	-
SBRANCH	VARCHAR (25)	NOT NULL	-	-
PHONE_NO	NUMBER (10)	-	CHECK	PH_C
TID	CHAR (2)	-	FOREIGN KEY	TID_FK

Query to create the teachers table.

```
CREATE TABLE TEACHER
(
TID VARCHAR (2) NOT NULL,
TNAME VARCHAR (20) NOT NULL,
```

SALARY NUMBER (7,2),

CONSTRAINT TID_PK PRIMARY KEY (COLUMN_NAME),

CONSTRAINT SAL_C CHECK (SALARY>0));

Query to create the STUDENT table.

CREATE TABLE STUDENT

(

SID CHAR (6) NOT NULL,

SNAME VARCHAR (20) NOT NULL,

BRANCH VARCHAR (25) NOT NULL,

PHONE_NO NUMBER (10),

TID VARCHAR (2),

CONSTARINT SID_PK PRIMARY KEY (SID),

CONSTRAINT PH_U UNIQUE (PHONE_NO),

CONSTRAINT PH_C CHECK (LENGTH(PHONE_NO) = 10),

CONSTRAINT TID_FK FOREIGN KEY (TID) REFERENCES TEACHER (TID));

To check the description of the TABLE.

DESC TABLE_NAME;

2. *RENAME*

   It is used to change the name of the existing table.

   Syntax:

   RENAME current_table_name TO new_table_name;

   Ex: RENAME STUDENT TO STU;

3. *ALTER*

   It is used to change the structure of the table.

   To add a column

   ALTER TABLE table_name

   ADD COLUMN_NAME DATATYPE NULL/NOT NULL;

   To drop a column

   ALTER TABLE table_name

   DROP COLUMN column_name;

   To change the datatype

   ALTER TABLE table_name

   MODIFY COLUMN_NAME new_datatype.

To change NOT NULL constraint

ALTER TABLE table_name
MODIFY COLUMN_NAME existing_datatype NULL/NOT NULL;

To rename the column

ALTER TABLE table_name
RENAME COLUMN current_name TO ne_name;

To modify the constraints

a. ALTER TABLE table_name
ADD CONSTRAINT constraint_ref_name UNIQUE
(COLUMN_NAME);

b. ALTER TABLE table_name
ADD CONSTRAINT constraint_ref_name CHECK (CONDITION);

c. ALTER TABLE table_name
ADD CONSTRAINT constraint_ref_name PRIMARY KEY
(COLUMN_NAME);

d. ALTER TABLE table_name
ADD CONSTRAINT constraint_ref_name FOREIGN KEY
(COLUMN_NAME);

To drop/disable/enable a constraint.

ALTER TABLE table_name
DROP/DISABLE/ENAME CONSTRAINT constraint_ref_name;

4. *TRUNCATE*

It is used to delete all the records permanently from the table, the table will not be deleted and the structure of the table remains same.

Syntax:

```
TRUNCATE TABLE table_name;
```

5. *DROP*

Syntax:

```
DROP TABLE table_name;
```

To remove the table (only in ORACLE)

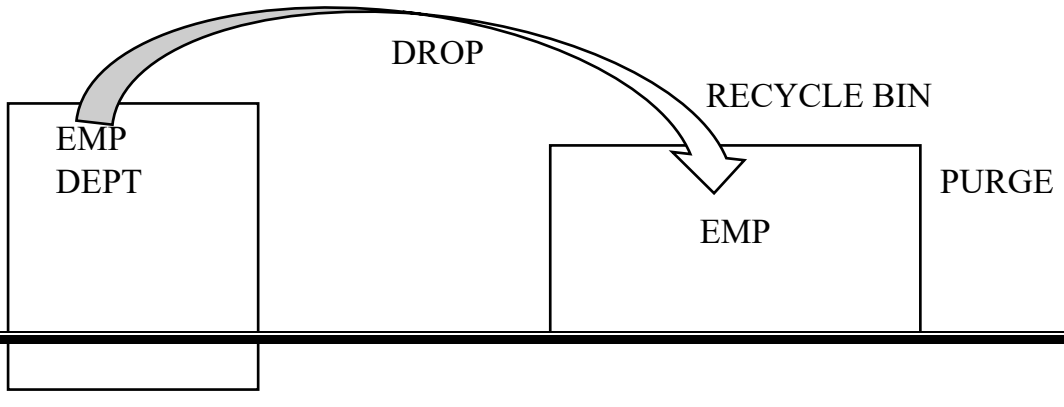
Syntax:

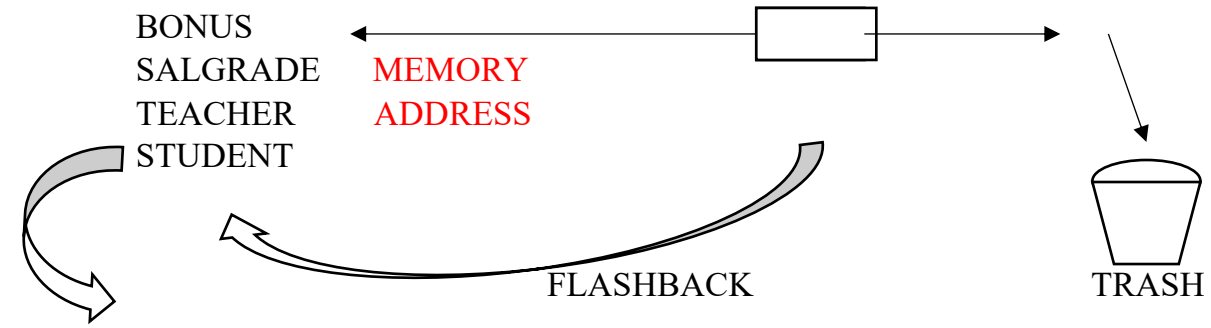
```
FLASHBACK TABLE table_name
TO BEFORE DROP
[RENAME TO new_table_name];
```

To drop the table from the recycle bin.

Syntax:

```
PURGE TABLE table_name;
```





PURGE (NOT POSSIBLE)

NOTE: DDL STATEMENTS ARE AUTO COMMIT.

DATA MANIPULATION LANGUAGE (DML)

1. *INSERT*

   This statement is used to insert a record into the table.

   Syntax:

   INSERT INTO table_name VALUES (V1, V2, V3, ……, Vn);

   Example:

```
INSERT INTO TEACHER VALUES ('T1','RAM',15000);
INSERT INTO TEACHER VALUES('T2','SURYA',30000.5);
INSERT INTO TEACHER VALUES('T3','SHYAM',65000.8);
INSERT INTO TEACHER VALUES('T4','AVANI',99999.99);

INSERT INTO STUDENT VALUES('QSP234','MALATHI','CIVIL',7867564534,'T4');
INSERT INTO STUDENT VALUES('QSP678','NEHA','ELECTRICAL',9876543215,'T2');
INSERT INTO STUDENT VALUES('QSP546','ARUN','CSE',6757654325,'T1');
INSERT INTO STUDENT VALUES('QSP237','KRISHNA','MECHANICAL',8978675645,'T3');
```

2. *UPDATE*

   It is used to update the records. We can update a particular record as well as all the records.

   Syntax:

   UPDATE table_name
   SET COL1=V1, COL2=V2,….., COLn=Vn;
   [WHERE <filter condition>];

```
UPDATE  STUDENT
SET PH_NO=8976587443
WHERE BRANCH='CSE';
```

3. *DELETE*

It is used to delete the records. We can delete a particular record as well as all the records.

Syntax:

```
DELETE
FROM table_name
[WHERE <filter condition>];
```

```
DELETE
FROM STUDENT
WHERE TID='T2';
```

TRANSACTION CONTROL LANGUAGE (TCL):

1. *COMMIT*

   It is used to save the transactions.

Syntax:

```
COMMIT;
```

2. *SAVEPOINT*

   It is used to create checkpoints.

Syntax:

```
SAVEPOINT savepoint_name;
```

3. *ROLLBACK*

   It is used to undo the transactions.

Syntax:

```
ROLLBACK;
```

Rollback to savepoint

```
ROLLBACK TO savepoint_name;
```

DATA CONSTROL LANGUAGE (DCL)

1. *GRANT*

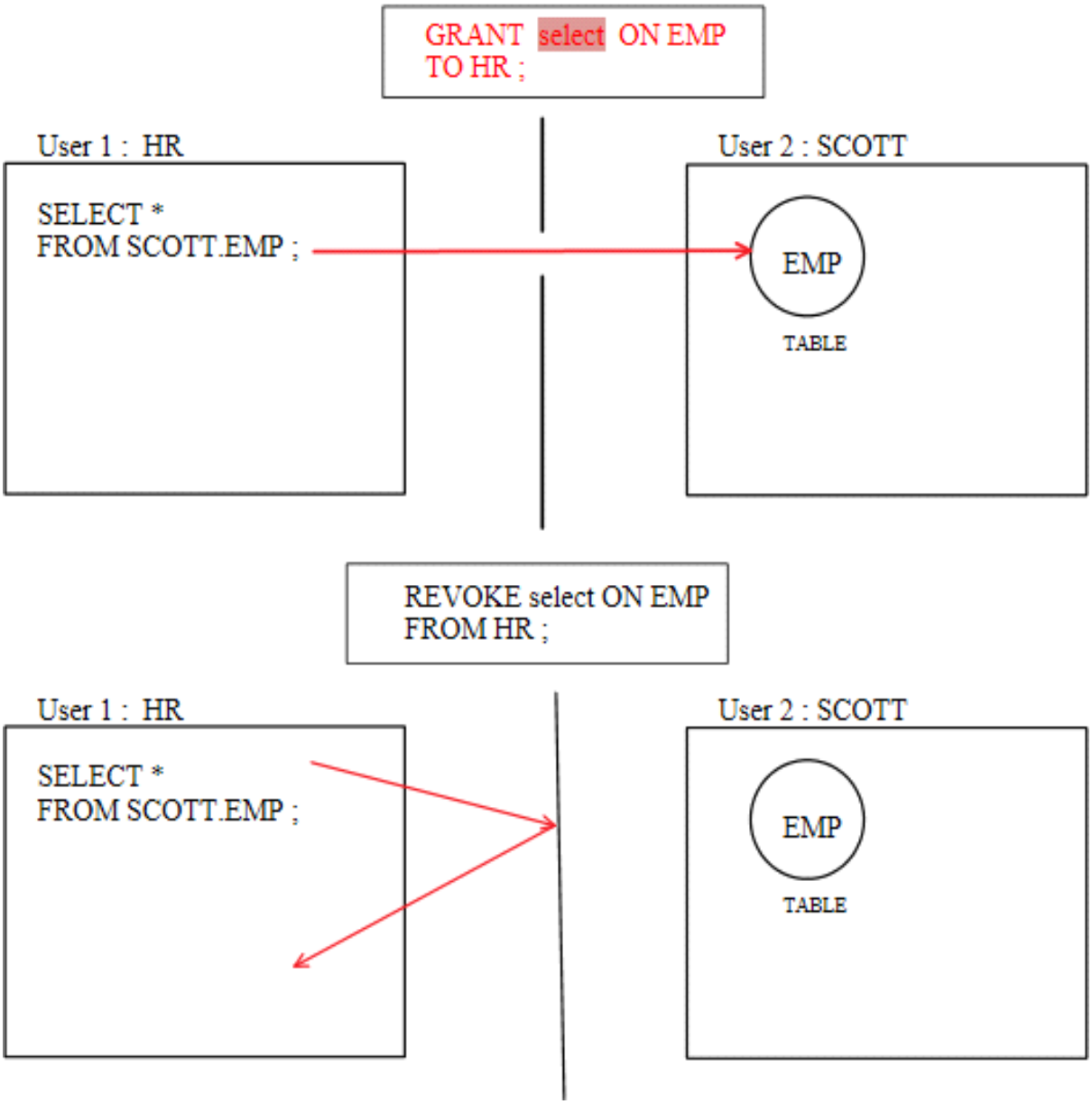
It is used to give the restricted permission to the user.

Syntax:

```
GRANT sql_statement ON table_name
TO user_name;
```

Example:

```
GRANT INSERT ON STUDENT
TO HR;
```



2. *REVOKE*

It is used to take back the given permission from the user.

Syntax:

```
REVOKE sql_statement ON table_name
FROM user_name;
```

Example:

```
REVOKE INSERT ON STUDENT
FROM HR;
```

EX: REVOKE select ON EMP
FROM HR ;

EXAMPLE :

```
SQL> CONNECT
Enter user-name: SCOTT
Enter password: *****
Connected.
SQL> GRANT DELETE ON EMP
  2  TO HR ;

Grant succeeded.

SQL> CONNECT
Enter user-name: HR
Enter password: *****
Connected.
SQL> SHOW USER
USER is "HR"
SQL> DELETE FROM SCOTT.EMP
  2  WHERE ENAME LIKE 'A%';

2 rows deleted.

SQL> SELECT *
  2  FROM SCOTT.EMP ;
FROM SCOTT.EMP
          *
ERROR at line 2:
ORA-01031: insufficient privileges
```

TRUNCATE	**DELETE**
Belongs to DDL	Belongs to DML
Removes all the records from the Table permanently.	Removes a particular record from the Table.
Auto COMMIT	Not auto COMMIT.

ATTRIBUTES

1. KEY ATTRIBUTE/ CANDIDATE KEY
   An attribute which is used to identify the records uniquely from the table is known as Key attribute/ Candidate key.
   Example: EMPNO, SID, PHONE_NO, TID
2. NON-KEY ATTRIBUTE
   All attributes except key attributes are known as Non-key attributes.
   Example: ENAME, SAL, JOB, MGR, HIREDATE etc.
3. PRIME KEY ATTRIBUTE
   Among all the key attributes one attribute chosen to be the main attribute to identify the records uniquely from the table.
   Example: SID, EMPNO, TID
4. NON-PRIME KEY ATTRIBUTE
   All key attributes except prime key attribute are known as Non-prime key attributes.
   Example: PHONE_NO
5. COMPOSITE KEY ATTRIBUTE
   It is a combination of more than one non-key attributes which is used to identify the records uniquely from the table.
   Example: (ENAME, JOB, HIREDATE)
6. SUPER KEY ATTRIBUTE
   It is a set of all the key attributes.
   Example: (SID, PHONE_NO)
7. FOREIGN KEY ATTRIBUTE
   It is an attribute which behaves as an attribute of another table to represent the relationship.
   Example: DEPTNO, TID

FUNCTIONAL DEPENDENCIES:

A relation exists such that an attribute determines another attribute uniquely, is known as functional dependency.

Types of Functional Dependencies:

1. TOTAL FUNCTIONAL DEPENDENCY
   If all attributes of a relation are determined a key attribute we call it as Total functional dependency.

   Example: let us consider a relation R contains attributes A, B, C and D in which A is key attribute.
   R→{A, B, C, D}
   A→B
   A→C

A→D
A→(B, C, D)

2. PARTIAL FUNCTIONAL DEPENDENCY

For a partial function to exist there must be a composite key relation, one of the attributes in a composite key relation determines another attribute separately then there exist a partial functional dependency.

Example: let us consider a relation R with four attributes A, B, C and D. In which (A, B) is composite key attribute.
R→{A, B, C, D} {A, B}"→Composite key attribute

(A, B)→(C, D)
B→D

3. TRANSITIVE FUNCTIONAL DEPENDENCY

An attribute determined by another non-key attribute which internally determined by key attribute then there exist a transitive functional dependency.

Example: let us consider a relation R with four attributes A, B, C and D. In which A is a key attribute.
R→{A, B, C, D}
A*→key attribute
A→B
A→D
D→C

Redundancy: The repetition of unwanted data is known as Redundancy.
Anomaly: The side effects that occur due to DML operations is known as Anomaly.

Whenever there is redundancy there is Anomaly.

Total Functional dependency	Partial functional dependency	Transitive functional dependency
No redundancy	Redundancy present	Redundancy present
No anomaly	Anomaly present	Anomaly present

NORMALIZATION

It is a process of decomposition of larger table into smaller tables in order to remove redundancies and anomalies by identifying their functional dependencies is known as Normalization.

Normal form

Any table which does not have redundancies and anomalies is known as Normal form.

Levels of normal form

1. FIRST NORMAL FORM (1NF)
2. SECOND NORMAL FORM (2NF)
3. THIRD NORMAL FORM (3NF)

NOTE: If we reduce any given table to 3NF, then the table is said to be normalized.

1.FIRST NORMAL FORM (1NF)

- No duplicate records.
- Multivalued data should not be present.

QSPIDERS

QID	NAME	COURSE
1	A	JAVA
2	B	JAVA , SQL
3	C	MT , SQL
1	A	MT

QID	NAME	COURSE1	COURSE2	COURSE3
1	A	JAVA		MT
2	B	JAVA	SQL	
3	C		SQL	MT

2.SECOND NORMAL FORM

- Table should be in 1NF.
- Table should not have partial functional dependency

Example:
Emp→{ename, sal, deptno, dname, loc}

EMP

ENAME	SAL	DEPTNO	DNAME	LOC
A	100	10	D1	L1
B	150	20	D2	L2
C	200	10	D1	L1
D	230	20	D2	L2

EMP

ENAME	SAL	DEPTNO
A	100	10
B	150	20
C	200	10
D	230	20

DEPT

DEPTNO	DNAME	LOC
10	D1	L1
20	D2	L2

{ename, deptno}→composite key attribute

Deptno→dname

Deptno→loc

Here exist a partial functional dependency.

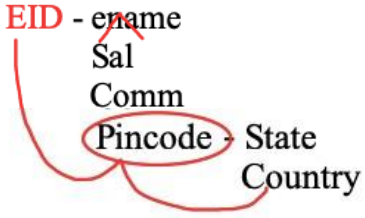
Emp→{empno, ename, sal, deptno}

Dept→{deptno, dname, loc}

3.THIRD NORMAL FORM

- Table should be in 2NF.
- Table should not have transitive functional dependency.

employee - (EID , ename , sal , comm , Pincode , state , country)



(EID , Pincode) - { ename , sal , comm , state , country }

R1 = { eid , ename , sal , comm }.
R2 = { pincode , state , country }.

3.5 NORMAL FORM (BOYCE CODD NORMAL FORM)

It is an update version of 3rd Normal form.

It is more strict than 3NF.