DBMS NOTES

Data -

"data" refers to any organized collection of facts or information that is stored and managed within the database.

Information -

“information” means meaningful data in database .

Sources Of Data -

1.Telecom - Telecom, short for telecommunications, refers to the transmission of information, such as voice, data, and video, over long distances using electronic means.

2.Automobile - Automobiles are powered by internal combustion engines, electric motors, or a combination of both.

3.Banking - Banks play a crucial role in the economy by offering various services to individuals, businesses, and governments.

4.Ecom – "E-commerce," short for electronic commerce, refers to the buying and selling of goods or services over the internet or other electronic networks.

5. sensor – A sensor is a device or component that detects or measures physical properties, conditions, or events and converts them into electrical signals or other readable output**. …… etc**

Unit of Data –

1 byte = 8 bit

1 kb = 1024 byte

1 mb = 1024 kb

1 GB = 1024 mb

1 TB =1024 GB

1 PB =1024 TB

1 EB = 1024 PB

1YB =1024 EB

1ZB = 1024 YB

Types Of Data –

1. Structured Data –

Structured data is organized in a structured format according to a predefined schema or data model, making it easy to store, retrieve, and analyze.

2. Unstructured Data / Non structured Data –

Non-structured data, also known as unstructured data, refers to data that does not have a predefined data model or organized format

3. Semi structured Data –

Semi-structured data is a type of data that does not conform to the rigid structure of traditional relational databases but has some semblance of structure. It contains elements of both structured and unstructured data, exhibiting characteristics of both.

DBMS-

DBMS stands for Database Management System. It's software that allows users to interact with a database. A database is a structured collection of data, typically stored and accessed electronically from a computer system.

Based On The Types Of Data there are different types of DBMS

Structure Data-RDBMS – Relational DBMS

mysql , Oracle , Postgre , DB2 , msserver etc…

RDBMS –

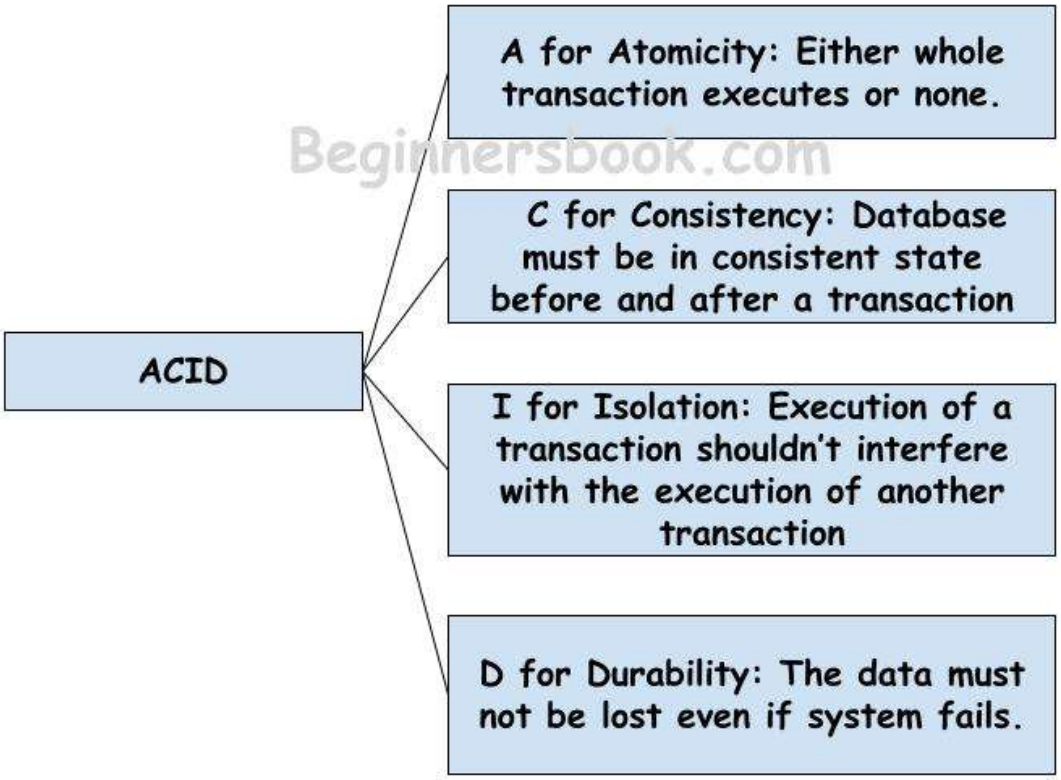
RDBMS stands for Relational Database Management System. It's a type of database management system that organizes data into tables, where each table consists of rows and columns.

Columns – Store Specific info for all Records.

Rows – it is also known as Record / Tuple .

ACID –

ACID stands for Atomicity, Consistency, Isolation, and Durability. These are the four key properties that ensure the reliability and integrity of transactions in a database system:



There are 3 Commands in SQL –

- **DDL (Data definition language)**
  1. **Create**
  2. **Alter**
  3. **Drop**
  4. **Truncate**

- **DML (Data Manipulation language)**
  1.insert
  2.update
  3.delete

- **DRL/DSL/DQL (Data Retrieval language)(Data Selective language)(Data Query language)**
  1.select

SQL is not case sensitive language.

List all databases –

```sql
SHOW DATABASES;
```

Create databases –

```sql
CREATE DATABASE database_name;
```

Use database-

```sql
USE database_name;
```

How to check in which database we are present –

```sql
SELECT DATABASE();
```

Show tables in Database –

```sql
SHOW TABLES;
```

Describe table –

```sql
DESC table_name;
```

Insert data in table –

```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Show table -

```sql
SELECT * FROM table_name;
```

Datatypes –

Data types in SQL refer to the specific type of data that can be stored in a column of a table.

1.Int –

"int" data type represents integer values. Integers are whole numbers without any decimal points.

2. Varchar() –

The "VARCHAR" data type is commonly used in database management systems (DBMS) like MySQL, PostgreSQL, Oracle, SQL Server, etc. It stands for "Variable Character" and is used to store character strings of variable length.

3. Boolean –

It represents a binary value that can either be true or false. It's particularly useful for conditions, comparisons.

4. Float –

Floats are used to represent numbers with decimal points, allowing for fractional values.

5. Date –

The date data type represents a specific calendar date, typically including the day, month, and year.

6.Timestamp –

Timestamps represent a specific moment in time, including both the date and time components.

Constraints –

constraints are rules or conditions applied to columns or tables to enforce data integrity and ensure that data follows certain rules or conditions.

1.Not null –

It will not allow null values

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    HireDate DATE NOT NULL
);
```

2.unique-

It not accept duplicate values

```sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE,
    StudentName VARCHAR(50),
    DateOfBirth DATE,
    CONSTRAINT uc_email UNIQUE (Email)
);
```

3.primary key –

It is the Combination Of Not null and uniq constraint

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    HireDate DATE
);
```

4.default –

When defining a DEFAULT constraint for a column, you specify a default value that will be used if no value is provided for that column in an INSERT statement.

```sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE DEFAULT CURRENT_DATE
);
```

5.foreign key –

Foreign keys define relationships between tables within a database. These relationships establish how data in one table relates to data in another table, creating a parent-child relationship between the tables.

Parent table should have primary key to create foreign key.

```sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

DDL – 1.create 2. Alter 3. Drop 4 . truncate

DML – 1. insert 2. update 3. Delete

DRL / DQL / DSL – 1.select

Create Table -

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    HireDate DATE
);
```

Alter Table –

The "ALTER TABLE" statement in SQL is used to modify the structure of an existing table. It allows you to perform various operations on a table, such as adding, modifying, or dropping columns, as well as adding or dropping constraints.

Rename Table –

```sql
ALTER TABLE Employees
RENAME TO Staff;
```

Add column -

```sql
ALTER TABLE Employees
ADD COLUMN Department VARCHAR(50);
```

Drop Existing Column –

```sql
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

Drop Constraints –

```sql
ALTER TABLE TableName
DROP CONSTRAINT ConstraintName;
```

Add Constraints –

```sql
ALTER TABLE TableName
ADD CONSTRAINT PK_ConstraintName PRIMARY KEY (ColumnList);
```

Change Datatype Of Column –

```sql
ALTER TABLE students
ALTER COLUMN age VARCHAR(3);
```

Truncate –

The TRUNCATE TABLE statement is used to remove all rows from a table quickly

```sql
TRUNCATE TABLE TableName;
```

Drop –

DROP command is used to remove database objects such as tables, indexes, views, or stored procedures.

```sql
DROP TABLE TableName;
```

Note - All DDL Commands are autocommit (autosave) means we cannot restore it

DML Command –

Update –

Used to modify existing data in a table.

```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- **Delete –**

Used to remove rows from a table.

```sql
DELETE FROM table_name
WHERE condition;
```

- **Insert –**

Used to add new rows of data into a table

```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Difference between delete , truncate and drop -

S. No	DELETE	TRUNCATE	DROP
1.	Delete is a DML command.	Truncate is a DDL command.	Drop is a DDL command.
2.	Delete is used to delete the records in the table.	Truncate is used to delete the data and keep the table structure as it is.	Drop is used to drop the table data as well as table structure.
3.	Delete statement use the where clause to delete particular records.	Truncate can't use where clause to delete particulate data.	Drop can't use where clause.
4.	Delete statement can be rollback before the commit.	Truncate statement can't rollback	Drop statement can't rollback.
5.	Delete is slower as compare to Truncate.	Truncate is faster as compare to Delete.	Drop removes the table from the database.
6.	Syntax: - delete from table_name;	Syntax: - Truncate Table table_name ;	Syntax: - Drop Table table_name;

DRL Command –

It's a subset of SQL (Structured Query Language) used for querying data from a database. DQL commands are primarily focused on retrieving data.

- **Select –**

Used to retrieve data from one or more tables.

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- **Alias –**

An "alias" is a temporary name assigned to a table or a column in a query. Aliases are often used to make column names more readable or to shorten table names in complex queries.

```sql
SELECT first_name AS "First Name", last_name AS "Last Name"
FROM employees;
```

AS Keyword is optional it is not Mandatory

Table: students

name	class	fee
Ali	MCS	0
Asad	BSCS	0
Arshad	BSCS	0
Akram	MCS	2000
Shamil	MCS	1000

SQL Query

```
SELECT name AS n, fee AS f
FROM students;
```

Output

n	f
Ali	0
Asad	0
Arshad	0
Akram	2000
Shamil	1000

- **Where Clause –**

"WHERE" clause is used to filter rows from a table or tables based on a specified condition or set of conditions.

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SELECT: Specifies the columns you want to retrieve.

FROM: Specifies the table from which you want to retrieve the data.

WHERE: Specifies the condition(s) that must be met for a row to be included in the result set.

- **Operator –**

- **Arithmetic Operator – (+ , - , / , % , *)**

“ + “ operator -

Query-

select eid,ename,did,sal,sal+5000 salwithbonus from emp;

Output –

eid	ename	did	sal	salwithbonus
1	John	101	50000.00	55000.00
2	Alice	102	60000.00	65000.00
3	Bob	103	55000.00	60000.00
4	Carol	101	52000.00	57000.00
5	David	102	58000.00	63000.00
6	Emma	103	51000.00	56000.00

“ - “operator –

Query –

select eid,ename,did,sal,sal-2000 salwithtax from emp;

Output –

eid	ename	did	sal	salwithtax
1	John	101	50000.00	48000.00
2	Alice	102	60000.00	58000.00
3	Bob	103	55000.00	53000.00
4	Carol	101	52000.00	50000.00
5	David	102	58000.00	56000.00
6	Emma	103	51000.00	49000.00

" * " operator -

Query –

```
select eid,ename,did,sal,sal*12 annual_sal from emp;
```

Output –

eid	ename	did	sal	annual_sal
1	John	101	50000.00	600000.00
2	Alice	102	60000.00	720000.00
3	Bob	103	55000.00	660000.00
4	Carol	101	52000.00	624000.00
5	David	102	58000.00	696000.00
6	Emma	103	51000.00	612000.00

" / " operator -

Query –

```
select eid,ename,did,sal,sal/2 halfmonthsal from emp;
```

Output –

eid	ename	did	sal	halfmonthsal
1	John	101	50000.00	25000.000000
2	Alice	102	60000.00	30000.000000
3	Bob	103	55000.00	27500.000000
4	Carol	101	52000.00	26000.000000
5	David	102	58000.00	29000.000000
6	Emma	103	51000.00	25500.000000

" % " operator -

```
select * from emp where eid%2=0;
```

Output –

eid	ename	did	sal
2	Alice	102	60000.00
4	Carol	101	52000.00
6	Emma	103	51000.00
8	Grace	102	59000.00
10	Ivy	101	51000.00
12	Karen	103	52000.00
14	Molly	102	60000.00
16	Oliver	101	52000.00

- **Relational Operator /Comparison Operator -**

" < " operator -

Input –

eid	ename	did	sal
1	John	101	50000.00
2	Alice	102	60000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
14	Molly	102	60000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

Query –

```
select * from emp where sal < 60000;
```

Output –

eid	ename	did	sal
1	John	101	50000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

“ <= “ operator -

Query –

```
select * from emp where sal <= 60000;
```

Output –

eid	ename	did	sal
1	John	101	50000.00
2	Alice	102	60000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
14	Molly	102	60000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

" > " operator –

Query –

```
select * from emp where sal > 50000;
```

Output –

eid	ename	did	sal
2	Alice	102	60000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
14	Molly	102	60000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

“ >= “ operator -

Query –

```
select * from emp where sal >= 60000;
```

Output –

```
+-----+-------+------+----------+
| eid | ename | did  | sal      |
+-----+-------+------+----------+
|   2 | Alice |  102 | 60000.00 |
|  14 | Molly |  102 | 60000.00 |
+-----+-------+------+----------+
```

" = " operator -

Query –

```
select * from emp where sal = 60000;
```

Output –

```
+-----+-------+------+----------+
| eid | ename | did  | sal      |
+-----+-------+------+----------+
|   2 | Alice |  102 | 60000.00 |
|  14 | Molly |  102 | 60000.00 |
+-----+-------+------+----------+
```

" != " operator -

Query –

```
select * from emp where sal != 60000;
```

Output –

eid	ename	did	sal
1	John	101	50000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

- **Logical Operator – (AND , OR , NOT)**

Note - in logical AND if both condition true then only record get in result in Logical OR if any one of the condition True then we grt record in result.

And operator –

Input –

eid	ename	did	sal
1	John	101	50000.00
2	Alice	102	60000.00
3	Bob	103	55000.00
4	Carol	101	52000.00
5	David	102	58000.00
6	Emma	103	51000.00
7	Frank	101	53000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
10	Ivy	101	51000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
13	Larry	101	54000.00
14	Molly	102	60000.00
15	Nancy	103	53000.00
16	Oliver	101	52000.00
17	Pam	102	59000.00

Query –

```
Select  *  from emp where sal > 50000 and did = 101 ;
```

Output –

eid	ename	did	sal
4	Carol	101	52000.00
7	Frank	101	53000.00
10	Ivy	101	51000.00
13	Larry	101	54000.00
16	Oliver	101	52000.00

Or operator –

Query –

```
Select * from emp where sal > 60000 or did = 101 ;
```

Output –

eid	ename	did	sal
1	John	101	50000.00
4	Carol	101	52000.00
7	Frank	101	53000.00
10	Ivy	101	51000.00
13	Larry	101	54000.00
16	Oliver	101	52000.00

Not operator –

Query –

Select * from emp where not did = 101 ;

Output –

eid	ename	did	sal
2	Alice	102	60000.00
3	Bob	103	55000.00
5	David	102	58000.00
6	Emma	103	51000.00
8	Grace	102	59000.00
9	Henry	103	54000.00
11	Jack	102	57000.00
12	Karen	103	52000.00
14	Molly	102	60000.00
15	Nancy	103	53000.00
17	Pam	102	59000.00

- **Identity Operator – (is , is not)**

  is operator –

  Input –

empid	empname	sal
1	John Doe	50000.00
2	Jane Smith	60000.00
3	Alice Johnson	55000.00
4	Bob Williams	52000.00
5	Emily Davis	58000.00
6	neeta	NULL

Query –

Select * from emp2 where sal is null ;

Output –

empid	empname	sal
6	neeta	NULL

Is not operator –

Query –

Select * from emp2 where sal is not null ;

Output –

empid	empname	sal
1	John Doe	50000.00
2	Jane Smith	60000.00
3	Alice Johnson	55000.00
4	Bob Williams	52000.00
5	Emily Davis	58000.00

- **Membership Operator – (IN , NOT IN)**

in operator –

Input –

eid	ename	mgrid
1	Karan	NULL
2	Rohan	1
3	Gokul	1
4	Preet	2
5	Neeta	2
6	peter	3
7	bond	3
8	John	5
9	AKANKSHA	2
10	sanket	2
12	ajit	2

Query-

```
Select * from emp where eid in (1,6,7,9);
```

Output –

eid	ename	mgrid
1	Karan	NULL
6	peter	3
7	bond	3
9	AKANKSHA	2

Not in operator –

Query –

Select * from emp where eid not in (1,6,7,9,10);

Output –

eid	ename	mgrid
2	Rohan	1
3	Gokul	1
4	Preet	2
5	Neeta	2
8	John	5
12	ajit	2

Like Operator –

The pattern you provide can include wildcard characters to represent different types of matches

- % (percent sign): Matches any sequence of characters (including zero characters).
- _ (underscore): Matches any single character.

Input –

eid	ename	mgrid
1	Karan	NULL
2	Rohan	1
3	Gokul	1
4	Preet	2
5	Neeta	2
6	peter	3
7	bond	3
8	John	5
9	AKANKSHA	2
10	sanket	2
12	ajit	2

Query –

```
select * from emp where ename like 'K%';
```

Output –

eid	ename	mgrid
1	Karan	NULL

Query –

Select * from emp where ename like ‘ _ _ a%’ ;

Output –

eid	ename	mgrid
9	AKANKSHA	2

Query –

Select * from emp where ename like '_ _ _ _';

Output –

eid	ename	mgrid
7	bond	3
8	John	5

Between .... And

The BETWEEN operator in SQL is used to filter the result set based on a range of values. It's commonly used in conjunction with the AND operator.

Input –

empid	empname	sal
1	John Doe	50000.00
2	Jane Smith	60000.00
3	Alice Johnson	55000.00
4	Bob Williams	52000.00
5	Emily Davis	58000.00

Query –

```
select * from emp2 where sal between 51000 and 56000;
```

Output –

```
+-------+---------------+----------+
| empid | empname       | sal      |
+-------+---------------+----------+
|     3 | Alice Johnson | 55000.00 |
|     4 | Bob Williams  | 52000.00 |
+-------+---------------+----------+
```

- **Limit Clause –**

The LIMIT clause in SQL is used to restrict the number of rows returned by a query. It's particularly useful when you want to retrieve a subset of rows from a larger result set.

The LIMIT clause is typically used at the end of a SQL query

Syntax –

Select col from < table name > limit n ; // n number of lines

Select col from <table name> limit n , m // after nth record we are getting m number of record

Input –

```
+------+---------------+--------+
| eid  | ename         | mgrid  |
+------+---------------+--------+
|    1 | Karan         |   NULL |
|    2 | Rohan         |      1 |
|    3 | Gokul         |      1 |
|    4 | Preet         |      2 |
|    5 | Neeta         |      2 |
|    6 | peter         |      3 |
|    7 | bond          |      3 |
|    8 | John          |      5 |
|    9 | AKANKSHA      |      2 |
|   10 |    sanket     |      2 |
|   12 |    ajit       |      2 |
```

Query –

```
select * from emp limit 5;
```

Output –

```
+------+-------+-------+
| eid  | ename | mgrid |
+------+-------+-------+
|    1 | Karan |  NULL |
|    2 | Rohan |     1 |
|    3 | Gokul |     1 |
|    4 | Preet |     2 |
|    5 | Neeta |     2 |
+------+-------+-------+
```

Query –

Select * from emp limit 7,3 ;

Output –

```
+------+--------------+-------+
| eid  | ename        | mgrid |
+------+--------------+-------+
|    8 | John         |     5 |
|    9 | AKANKSHA     |     2 |
|   10 |    sanket    |     2 |
+------+--------------+-------+
```

- **Order By Clause –**

The ORDER BY clause in SQL is used to sort the result set returned by a query based on one or more columns. It allows you to specify the order in which the rows should appear in the final result set.
Specifies the sort order.
Note - ASC for ascending (default) and DESC for descending.

Syntax –
Select <col> from <table_name> order by <col> <order type> ;

Input –

eid	ename	sal
1	karan	25000
2	Rakesh	35000
3	Rohan	35000

Query –
Select eid , ename , sal from employee order by sal desc , eid desc ;

Output –

eid	ename	sal
3	Rohan	35000
2	Rakesh	35000
1	karan	25000

Query –

Select eid , ename , sal from employee order by sal asc , eid desc ;

Output –

eid	ename	sal
1	karan	25000
3	Rohan	35000
2	Rakesh	35000

- ### Case Statement –

The CASE statement in SQL is a conditional expression that allows you to perform logic within a query. It's useful for creating custom conditions and generating calculated fields based on those conditions.
The CASE statement can be used in various SQL clauses, such as SELECT, WHERE, ORDER BY, and others.

Syntax –

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    -- more conditions and results as needed
    ELSE default_result
END
```

Input –

eid	ename	did	city
1	karan	10	Pune
2	Rohan	10	Nagar
3	Ganesh	20	Pune
4	Rahul	20	Surat
5	Peter	10	NULL

Query –

```
select *,
case
when city='Pune' then 1000
when city='Nagar' then 500
when city='Surat' then 2000
else 0
end bonus
from emp1;
```

Output –

eid	ename	did	city	bonus
1	karan	10	Pune	1000
2	Rohan	10	Nagar	500
3	Ganesh	20	Pune	1000
4	Rahul	20	Surat	2000
5	Peter	10	NULL	0

- **Join –**

Join is used to Retrive data from more than one tables.
It is used to combine rows from two or more tables based on a related column between them.

Note - to perform join we need atleast one common values column in both table

- **Inner Join –**

An INNER JOIN in SQL is used to retrieve records from two or more tables where there is a match between the columns specified in the JOIN condition. It returns only the rows that have matching values in both tables.

```sql
SELECT columns
FROM table1
INNER JOIN table2 ON table1.column = table2.column;
```

SELECT -
specifies the columns you want to retrieve from the combined result.
FROM –
specifies the first table you're querying.
INNER JOIN –
specifies that you're performing an inner join.
ON -
specifies the condition for joining the tables, usually based on a related column between them.

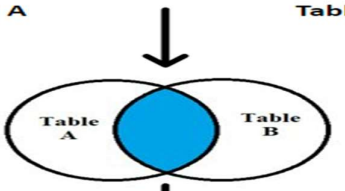
Query –
Select Student id, name ,department from table1 as t1 inner join table2 as t2 on t1.studentid = t2.studentid ;

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Table A

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

Table B



Student ID	Name	Department
1004	D	Mathematics

- **Left Join –**

A LEFT JOIN (or LEFT OUTER JOIN) in SQL is used to retrieve all records from the left table, and the matched records from the right table. If there are no matching records in the right table, NULL values are returned for the columns of the right table.

```sql
SELECT columns
FROM table1
LEFT JOIN table2 ON table1.column = table2.column;
```

SELECT -

specifies the columns you want to retrieve from the combined result.

FROM -

specifies the first table you're querying.

LEFT JOIN –

specifies that you're performing a left join.

ON -

specifies the condition for joining the tables, usually based on a related column between them .

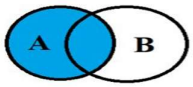
Query –

Select Student id, name ,department from table1 as t1 left join table2 as t2 on t1.studentid = t2.studentid ;

Student ID	Name
1001	A
1002	B
1003	C
1004	D

+

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science



Student ID	Name	Department
1001	A	NULL
1002	B	NULL
1003	C	NULL
1004	D	Mathematics

- **Right join –**

A RIGHT JOIN (or RIGHT OUTER JOIN) in SQL is similar to a LEFT JOIN, but it focuses on the right table. It returns all records from the right table and the matched records from the left table. If there are no matching records in the left table, NULL values are returned for the columns of the left table.

```sql
SELECT columns
FROM table1
RIGHT JOIN table2 ON table1.column = table2.column;
```

SELECT -

specifies the columns you want to retrieve from the combined result.

FROM -

specifies the first table you're querying.

LEFT JOIN -

specifies that you're performing a left join.

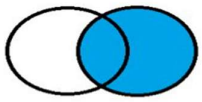
ON -

specifies the condition for joining the tables, usually based on a related column between them.

Query –

Select Student id, name ,department from table1 as t1 right join table2 as t2 on t1.studentid = t2.studentid ;

Student ID	Name
1001	A
1002	B
1003	C
1004	D



Student ID	Name	Department
1004	D	Mathematics
1005	NULL	Mathematics
1006	NULL	History
1007	NULL	Physics
1008	NULL	Computer Science

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

- **Full Outer Join –**

A FULL OUTER JOIN in SQL is a method used to combine rows from two tables based on a related column, returning all rows from both tables. It includes all records from both the left and right tables, and if there are no matching records, it fills in NULL values for the columns from the non-matching table.

```sql
SELECT columns
FROM table1
FULL OUTER JOIN table2 ON table1.column = table2.column;
```

SELECT –

specifies the columns you want to retrieve from the combined result.

FROM -

specifies the first table you're querying.

FULL OUTER JOIN -

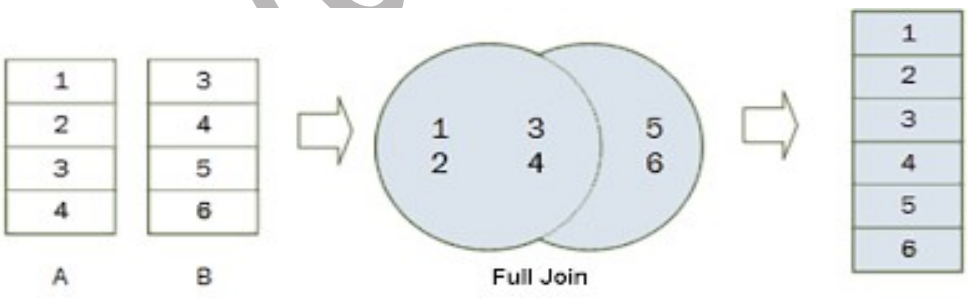
specifies that you're performing a full outer join, combining all records from both tables.

ON -

specifies the condition for joining the tables, usually based on a related column between them.

Query –

Select * from A full join B on A.id = B.id ;



- **Cross Join –**

A CROSS JOIN in SQL is used to combine every row from one table with every row from another table, resulting in a Cartesian product. It does not require

any condition to be specified for the join; instead, it simply joins each row of the first table with every row of the second table.

```sql
SELECT columns
FROM table1
CROSS JOIN table2;
```

SELECT –

specifies the columns you want to retrieve from the combined result.

FROM –

specifies the first table you're querying.

CROSS JOIN –

indicates that you're performing a cross join, combining every row from table1 with every row from table2.

Query –

Select * from employee cross join department ;

Employee

Emp_ID	Emp_Name	Salary
100	Reena	20000
101	Mathew	40000
102	Amar	30000

Department

Dept_Id	Dept_Name
1	Furniture
2	Technology

CROSS JOIN

Emp_ID	Emp_Name	Salary	Dept_Id	Dept_Name
100	Reena	20000	1	Furniture
100	Reena	20000	2	Technology
101	Mathew	40000	1	Furniture
101	Mathew	40000	2	Technology
102	Amar	30000	1	Furniture
102	Amar	30000	2	Technology

- **Self Join –**

A self-join in SQL is when a table is joined with itself. This is commonly done when you have a table with a hierarchical structure or when you need to compare rows within the same table.

sql Copy code

```sql
SELECT e.employee_id, e.employee_name, m.employee_name AS manager_name
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id;
```

Query –

```
mysql> select * from employees;
+------+----------+------+
| eid  | ename    | mid  |
+------+----------+------+
|    1 | pooja    | NULL |
|    2 | ajit     |    1 |
|    3 | sanket   |    1 |
|    4 | priyanka |    2 |
|    5 | mayuri   |    3 |
|    6 | neeta    |    3 |
+------+----------+------+
6 rows in set (0.00 sec)
```

```
mysql> select e.eid ,e.ename,m.ename as manager from employees e left join employees m on m.eid=e.mid;
+------+----------+---------+
| eid  | ename    | manager |
+------+----------+---------+
|    1 | pooja    | NULL    |
|    2 | ajit     | pooja   |
|    3 | sanket   | pooja   |
|    4 | priyanka | ajit    |
|    5 | mayuri   | sanket  |
|    6 | neeta    | sanket  |
+------+----------+---------+
6 rows in set (0.00 sec)
```

Join more than 2 tables

Query -

```
mysql> select * from address;
+-------+------+------------------+
| addid | padd | eadd             |
+-------+------+------------------+
| ad101 | ABC  | rohan@gmail.com  |
| ad102 | PIP  | rakesh@gmail.com |
+-------+------+------------------+
```

```
mysql> select * from order_status;
+------+------------+
| oid  | status     |
+------+------------+
|   11 | DELIVERED  |
|   12 | IN TRANSIT |
+------+------------+
```

```
mysql> select * from orders;
+-----+------+-------------+----------+-------+
| oid | cid  | description | amount   | addid |
+-----+------+-------------+----------+-------+
|  11 |    1 | mobile      | 25000.00 | ad101 |
|  12 |    1 | laptop      | 50000.00 | ad102 |
+-----+------+-------------+----------+-------+
```

```
mysql> select * from customers;
+-----+--------+-------+-------+
| cid | cname  | city  | addid |
+-----+--------+-------+-------+
|   1 | rohan  | pune  | ad101 |
|   2 | rakesh | nagar | ad102 |
+-----+--------+-------+-------+
```

Answer –

Select customers.cid, cname, city, orders.oid, description, amount, status, padd, eadd
from customers
left join orders on customers.cid=orders.cid
left join order_status on orders.oid=order_status.oid
left join address on customers.addid=address.addid;

```
+-----+--------+-------+------+-------------+----------+------------+------+------------------+
| cid | cname  | city  | oid  | description | amount   | status     | padd | eadd             |
+-----+--------+-------+------+-------------+----------+------------+------+------------------+
|   1 | rohan  | pune  |   11 | mobile      | 25000.00 | DELIVERED  | ABC  | rohan@gmail.com  |
|   1 | rohan  | pune  |   12 | laptop      | 50000.00 | IN TRANSIT | ABC  | rohan@gmail.com  |
|   2 | rakesh | nagar | NULL | NULL        |     NULL | NULL       | PIP  | rakesh@gmail.com |
+-----+--------+-------+------+-------------+----------+------------+------+------------------+
```

Functions –

Functions are named blocks of code that perform specific tasks. They take input, do something with it, and then produce output.

Scalar Function:

A scalar function operates on individual values and returns a single result. It takes input parameters and performs operations like mathematical calculations, string manipulations, or date/time conversions.

Group Function:

A group function operates on groups of rows and returns a single result for each group. It's often used with the GROUP BY clause to perform calculations on subsets of data.

1 . Upper( ) –

Syntax - upper(Column_name) ;

The UPPER() function in SQL is used to convert a string to uppercase. It takes a string as input and returns a new string where all characters are in uppercase.

Input –

eid	ename	mgrid
1	Karan	NULL
2	Rohan	1
3	Gokul	1
4	Preet	2
5	Neeta	2
6	peter	3
7	bond	3
8	John	5

Query –

```
select eid,upper(ename) as ename from emp;
```

Output –

eid	ename	mgrid
1	KARAN	NULL
2	ROHAN	1
3	GOKUL	1
4	PREET	2
5	NEETA	2
6	PETER	3
7	BOND	3
8	JOHN	5

3. Lower( ) –

Syntax – lower(Column_name) ;

The LOWER() function in SQL is the counterpart to the UPPER() function. It's used to convert a string to lowercase.

Input –

eid	ename	mgrid
1	KARAN	NULL
2	ROHAN	1
3	GOKUL	1
4	PREET	2
5	NEETA	2
6	PETER	3
7	BOND	3
8	JOHN	5

Query –

```
select eid,lower(ename) as ename from emp;
```

Output –

```
+------+-------+
| eid  | ename |
+------+-------+
|    1 | karan |
|    2 | rohan |
|    3 | gokul |
|    4 | preet |
|    5 | neeta |
|    6 | peter |
|    7 | bond  |
|    8 | john  |
|    9 | nick  |
+------+-------+
```

3. Length( ) –

Syntax - length(Column_name) ;

The LENGTH() function in SQL is used to determine the length of a string. It returns the number of characters in the specified string.

Input –

```
+------+-------+-
| eid  | ename |
+------+-------+-
|    1 | KARAN |
|    2 | ROHAN |
|    3 | GOKUL |
|    4 | PREET |
|    5 | NEETA |
|    6 | PETER |
|    7 | BOND  |
|    8 | JOHN  |
|    9 | NICK  |
+------+-------+-
```

Query –

```
select eid,ename,length(ename) from emp;
```

Output –

eid	ename	length(ename)
1	KARAN	5
2	ROHAN	5
3	GOKUL	5
4	PREET	5
5	NEETA	5
6	PETER	5
7	BOND	4
8	JOHN	4
9	NICK	4

4. **Trim( ) -**

Syntax - trim(Column_name);

The TRIM() function in SQL is used to remove unwanted before and after spaces from value

Input –

cid	cname
1	akanksha
2	karan
3	rahul sharma
4	sandip sharma
5	priya

Query –

```
select cid,trim(cname) from cust1;
```

Output –

```
+------+---------------+
| cid  | trim(cname)   |
+------+---------------+
|    1 | akanksha      |
|    2 | karan         |
|    3 | rahul sharma  |
|    4 | sandip sharma |
|    5 | priya         |
```

LTRIM()

Syntax - ltrim(Column_name);

The LTRIM () function in SQL is used to remove only left sides before values space

Input –

```
+------+----------------+
| cid  | cname          |
+------+----------------+
|    1 |    akanksha    |
|    2 | karan          |
|    3 |  rahul sharma  |
|    4 | sandip sharma  |
|    5 | priya          |
+------+----------------+
```

Query –

```
select cid,ltrim(cname) from cust1;
```

Output –

```
+------+-----------------+
| cid  | ltrim(cname)    |
+------+-----------------+
|    1 | akanksha        |
|    2 | karan           |
|    3 | rahul sharma    |
|    4 | sandip sharma   |
|    5 | priya           |
+------+-----------------+
```

RTRIM( )–

Syntax – rtrim(Column_name);

The RTRIM() function in SQL is used to remove only right sides after values space

Input –

```
+------+-----------------+
| cid  | cname           |
+------+-----------------+
|    1 |    akanksha     |
|    2 | karan           |
|    3 |  rahul sharma   |
|    4 | sandip sharma   |
|    5 | priya           |
```

Query –

```
select cid,rtrim(cname) from cust1;
```

Output –

```
+------+----------------+
| cid  | rtrim(cname)   |
+------+----------------+
|    1 |    akanksha    |
|    2 | karan          |
|    3 |  rahul sharma  |
|    4 | sandip sharma  |
|    5 | priya          |
+------+----------------+
```

5.repeat( )–

Syntax - repeat(column _name , n):

The REPEAT() function in SQL is used to repeat a string a specified number of times. It takes two arguments: the string to be repeated and the number of times to repeat it.

Input –

```
+------+-------+
| eid  | ename |
+------+-------+
|    1 | karan |
|    2 | rohan |
|    3 | gokul |
|    4 | preet |
|    5 | neeta |
|    6 | peter |
|    7 | bond  |
|    8 | john  |
|    9 | nick  |
+------+-------+
```

Query –

```
select repeat(eid,3) from emp;
```

Output –

```
+---------------+
| repeat(eid,3) |
+---------------+
| 111           |
| 222           |
| 333           |
| 444           |
| 555           |
| 666           |
| 777           |
| 888           |
| 999           |
+---------------+
```

6.Replace( ) –

Syntax – replace(x , y)

The REPLACE() function in SQL is used to replace all occurrences of a specified substring within a string with another substring.

x:old value you wanted to replace

y:new value with you wanted to replace

Input –

```
+------+-----------------+
| cid  | cname           |
+------+-----------------+
|    1 |    akanksha     |
|    2 | karan           |
|    3 |  rahul sharma   |
|    4 | sandip sharma   |
|    5 | priya           |
+------+-----------------+
```

Query –

```
select cid,cname,replace(cname,'a','m') as replace_name from cust1;
```

Output –

```
+------+-----------------+-----------------+
| cid  | cname           | replace_name    |
+------+-----------------+-----------------+
|    1 |    akanksha     |    mkmnkshm     |
|    2 | karan           | kmrmn           |
|    3 |  rahul sharma   |  rmhul shmrmm   |
|    4 | sandip sharma   | smndip shmrmm   |
|    5 | priya           | priym           |
+------+-----------------+-----------------+
```

7.concat( ) –

The CONCAT() function in SQL is used to concatenate two or more strings together. It takes multiple string arguments and returns a single string that combines all the input strings.

Syntax – concat(col_name,col_name) ;

Input –

```
+------+--------+--------+--------+
| sid  | fname  | lname  | city   |
+------+--------+--------+--------+
|    1 | Vikas  | Kumar  | Pune   |
|    2 | Rajesh | Kale   | Surat  |
|    3 | Tushar | sharma | Mumbai |
|    4 | Vipul  | Dere   | Pune   |
|    5 | Rock   | seen   | CA     |
|    5 | Rock   | seen   | CA     |
+------+--------+--------+--------+
```

Query –

```
select sid,concat(fname,lname) as full_name,city from sub;
```

Output –

```
+------+--------+--------+--------+---------------+
| sid  | fname  | lname  | city   | full_name     |
+------+--------+--------+--------+---------------+
|    1 | Vikas  | Kumar  | Pune   | Vikas Kumar   |
|    2 | Rajesh | Kale   | Surat  | Rajesh Kale   |
|    3 | Tushar | sharma | Mumbai | Tushar sharma |
|    4 | Vipul  | Dere   | Pune   | Vipul Dere    |
|    5 | Rock   | seen   | CA     | Rock seen     |
|    5 | Rock   | seen   | CA     | Rock seen     |
+------+--------+--------+--------+---------------+
```

8.substr( ) –

Syntax – substr(column_name, start_index, total_char)

The SUBSTR() function in SQL is used to extract a substring from a string. it supports +ve and -ve indexing .It takes three arguments: the string from which to extract the substring, the starting position of the substring, and the length of the substring (optional).

For example –

```
K   a  r  a  n
1   2  3  4  5
-5 -4 -3 -2 -1
```

Input –

sid	fname	lname	city
1	Vikas	Kumar	Pune
2	Rajesh	Kale	Surat
3	Tushar	sharma	Mumbai
4	Vipul	Dere	Pune
5	Rock	seen	CA
5	Rock	seen	CA

Query –

```
select sid,fname,substr(fname,1,1) from sub;
```

Output –

sid	fname	substr(fname,1,1)
1	Vikas	V
2	Rajesh	R
3	Tushar	T
4	Vipul	V
5	Rock	R
5	Rock	R

Query –

```
select sid,concat(substr(fname,1,1),' ',substr(lname,1,1)) full_initial ,substr(city,1,3) city_3_char from sub;
```

Output –

sid	full_initial	city_3_char
1	V K	Pun
2	R K	Sur
3	T s	Mum
4	V D	Pun
5	R s	CA
5	R s	CA

9.substring_index() –

the SUBSTRING_INDEX() function is used to extract a substring from a string before or after a specified delimiter. It takes three arguments: the original string, the delimiter, and the occurrence number.

Syntax - substring_index(col_name,seperator,select value index)

Input –

sid	fname	lname	city	full_name
1	Vikas	Kumar	Pune	Vikas Kumar
2	Rajesh	Kale	Surat	Rajesh Kale
3	Tushar	sharma	Mumbai	Tushar sharma
4	Vipul	Dere	Pune	Vipul Dere
5	Rock	seen	CA	Rock seen
5	Rock	seen	CA	Rock seen

Query –

```
select substring_index(full_name,' ',1) as firstname from sub;
```

Output –

```
+------------+
  firstname  |
+------------+
  Vikas      |
  Rajesh     |
  Tushar     |
  Vipul      |
  Rock       |
  Rock       |
+------------+
```

Query –

```
select substring_index(full_name,' ',-1) as lastname from sub;
```

Output –

```
+----------+
| lastname |
+----------+
| Kumar    |
| Kale     |
| sharma   |
| Dere     |
| seen     |
| seen     |
+----------+
```

10.distinct( ) –

Syntax – distinct(column_name);

The DISTINCT keyword in SQL is used to return unique values in a result set. It eliminates duplicate rows from the result set, ensuring that

each row is unique. distinct function gives only unique values from the table

Input –

```
+------+--------+
| roll | name   |
+------+--------+
|    1 | Karan  |
|    1 | Karan  |
|    2 | Rajesh |
|    2 | Rajesh |
|    3 | Ganesh |
|    1 | Rakesh |
+------+--------+
```

Query –

```
select distinct * from std;
```

Output –

```
+------+--------+
| roll | name   |
+------+--------+
|    1 | Karan  |
|    2 | Rajesh |
|    3 | Ganesh |
|    1 | Rakesh |
+------+--------+
```

Query –

```
select distinct roll from std;
```

Output –

```
+------+
| roll |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

11.format( ) –

Syntax – format(column_name , n)

N – (n indicates that how much digit we want after decimal)

FORMAT () function is indeed used to format values as strings. It allows you to specify a format pattern for numbers, dates, times, and other data types.

Input –

```
120.2356
```

Query –

```
select pid,pname,format(price,2) from sales;
```

Output –

```
+---------+
| price   |
+---------+
| 120.24  |
+---------+
```

12. Round( ) –

the ROUND() function is used to round a numeric value to a specified number of decimal places. It takes two arguments: the numeric value to be rounded and the number of decimal places.

Input –

```
120.2356
```

Query –

```
select round(120.2356  )  as price;
```

Output –

```
+-------+
| price |
+-------+
|   120 |
+-------+
```

13.coalesce –

The COALESCE() function in SQL is used to return the first non-null value in a list of expressions. It takes multiple arguments and returns the first non-null value. If all expressions are null, it returns null.

Input –

```
+------+--------+-------+--------+-------+
| cid  | cname  | pcity | ccity  | fcity |
+------+--------+-------+--------+-------+
|    1 | Rohan  | Pune  | Nagar  | Surat |
|    2 | Rajesh | NULL  | Mumbai | Pune  |
|    3 | Nilesh | NULL  | NULL   | Delhi |
|    4 | Kishor | NULL  | NULL   | NULL  |
+------+--------+-------+--------+-------+
```

Query-

```
select cid,cname,coalesce(pcity,ccity,fcity,'City Not PRESENT') as city from cust;
```

Output-

```
+------+--------+-------+
| cid  | cname  | city  |
+------+--------+-------+
|    1 | Rohan  | Pune  |
|    2 | Rajesh | Mumbai|
|    3 | Nilesh | Delhi |
|    4 | Kishor | city not present|
+------+--------+-------+
```

Date Function –

The DATE function is used to create a date value from a specified year, month, and day.

1.current_date( ) –

Returns the current date.

Syntax - select current_date();

Output –

```
+----------------+
| current_date() |
+----------------+
| 2024-05-03     |
+----------------+
```

2.current_time( ) –

Returns the current time.

Syntax – select current_time();

Output –

```
+----------------+
| current_time() |
+----------------+
| 08:42:57       |
+----------------+
```

3.current_timestamp( ) –

Returns the current date and time.

Syntax – select current_timestamp( );

Output –

```
+---------------------+
| current_timestamp() |
+---------------------+
| 2024-05-03 08:38:47 |
+---------------------+
```

4. now( ) –

NOW() is a function used to retrieve the current date and time . Returns the current date and time.

Syntax – select now( ) ;

Output –

```
+---------------------+
| now()               |
+---------------------+
| 2024-05-03 08:39:34 |
+---------------------+
```

5. year( ) -

Extract year from date or timestamp .

Syntax – select year( ) ;

Input –

```
+------+--------+------+------------+---------------------+---------+
| eid  | ename  | did  | doj        | rcd                 | act_flg |
+------+--------+------+------------+---------------------+---------+
|    1 | Rajesh |   10 | 2020-01-31 | 2024-05-03 09:00:24 |       1 |
|    2 | Roha   |   20 | 2021-05-06 | 2024-05-03 09:00:24 |       1 |
|    3 | Neel   |   10 | 2023-12-31 | 2024-05-03 09:00:24 |       1 |
+------+--------+------+------------+---------------------+---------+
```

Query –

```
select eid,ename,year(doj) as joining_year from emp;
```

Output -

```
+-------+---------+--------------+
| eid   | ename   | joining_year |
+-------+---------+--------------+
|     1 | Rajesh  |         2020 |
|     2 | Roha    |         2021 |
|     3 | Neel    |         2023 |
+-------+---------+--------------+
```

6. month( ) –

Extract month from date or timestamp .

Syntax – select month( ) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Syntax –

select eid,ename,doj,month(doj) as joining_month from emp;

Output –

eid	ename	doj	joining_month
1	Rajesh	2020-01-31	1
2	Roha	2021-05-06	5
3	Neel	2023-12-31	12

7.day( ) –

Extract day from date or timestamp .

Syntax – select day( ) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

select eid,ename,doj,day(doj) as joining_day from emp;

Output –

eid	ename	doj	joining_day
1	Rajesh	2020-01-31	31
2	Roha	2021-05-06	6
3	Neel	2023-12-31	31

8.dayname( ) -

Extract day name from date or timestamp .

Syntax – select dayname( ) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

select eid,ename,rcd,dayname(rcd) from emp;

Output –

eid	ename	rcd	dayname(rcd)
1	Rajesh	2024-05-03 09:00:24	Friday
2	Roha	2024-05-03 09:00:24	Friday
3	Neel	2024-05-03 09:00:24	Friday

9. hour( ) –

Extract hour from date or timestamp .

Syntax – select hour( ) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

```
select eid,ename,rcd,hour(rcd) from emp;
```

Output –

eid	ename	rcd	hour(rcd)
1	Rajesh	2024-05-03 09:00:24	9
2	Roha	2024-05-03 09:00:24	9
3	Neel	2024-05-03 09:00:24	9

10. minute( ) –

Extract minute from date or timestamp .

Syntax – select minute( ) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

```
select eid,ename,rcd,minute(rcd) from emp;
```

Output –

eid	ename	rcd	minute(rcd)
1	Rajesh	2024-05-03 09:00:24	0
2	Roha	2024-05-03 09:00:24	0
3	Neel	2024-05-03 09:00:24	0

11.second( ) –

Extract second from date or timestamp .

Syntax – select second( ) ;

Input -

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

```
select eid,ename,rcd,second(rcd) from emp;
```

Output –

eid	ename	rcd	second(rcd)
1	Rajesh	2024-05-03 09:00:24	24
2	Roha	2024-05-03 09:00:24	24
3	Neel	2024-05-03 09:00:24	24

12. date_format( ) –

Date_format( ) formats a date as specified . when we want to change formatting of date then we use this function .

Syntax – date_format(col , format) ;

Col - The date to be formatted

Format - The format to use

%y : year YY

%Y : year YYYY

%m : month

%d : day n

%D : day nth

%a : dayname Sat

%b : monthname Jan

%W : dayname Sunday

%H : hour

%i : minute

%s : second

%f : microsecond

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

Select eid,ename,rcd,Date_Format (rcd, “%W. %D %b ‘%Y %H: %i %s %f “) change_rcd from emp ;

Output –

eid	ename	rcd	change_rcd
1	Rajesh	2024-05-03 09:00:24	Friday. 3rd May'2024 09:00:24.000000
2	Roha	2024-05-03 09:00:24	Friday. 3rd May'2024 09:00:24.000000
3	Neel	2024-05-03 09:00:24	Friday. 3rd May'2024 09:00:24.000000

13.date_add( ) –

DATE_ADD() is a function commonly used in SQL databases to add a specified time interval to a date.

Syntax – date_add(col_name,interval <value> <unit>)

values : n

unit : day , week , month , year

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

select eid,ename,doj,date_add(doj,interval 2 day) from emp;

Output –

eid	ename	doj	date_add(doj,interval 2 day)
1	Rajesh	2020-01-31	2020-02-02
2	Roha	2021-05-06	2021-05-08
3	Neel	2023-12-31	2024-01-02

Query –

```
select eid,ename,doj,date_add(doj,interval 1 month) from emp;
```

Output –

```
+------+--------+------------+--------------------------------+
| eid  | ename  | doj        | date_add(doj,interval 1 month) |
+------+--------+------------+--------------------------------+
|    1 | Rajesh | 2020-01-31 | 2020-02-29                     |
|    2 | Roha   | 2021-05-06 | 2021-06-06                     |
|    3 | Neel   | 2023-12-31 | 2024-01-31                     |
+------+--------+------------+--------------------------------+
```

Query –

```
select eid,ename,doj,date_add(doj,interval 1 year) from emp;
```

Output –

```
+------+--------+------------+-------------------------------+
| eid  | ename  | doj        | date_add(doj,interval 1 year) |
+------+--------+------------+-------------------------------+
|    1 | Rajesh | 2020-01-31 | 2021-01-31                    |
|    2 | Roha   | 2021-05-06 | 2022-05-06                    |
|    3 | Neel   | 2023-12-31 | 2024-12-31                    |
+------+--------+------------+-------------------------------+
```

Query –

```
select eid,ename,doj,date_add(doj,interval 2 week) from emp;
```

Output –

```
+------+--------+------------+--------------------------------+
| eid  | ename  | doj        | date_add(doj,interval 2 week)  |
+------+--------+------------+--------------------------------+
|    1 | Rajesh | 2020-01-31 | 2020-02-14                     |
|    2 | Roha   | 2021-05-06 | 2021-05-20                     |
|    3 | Neel   | 2023-12-31 | 2024-01-14                     |
+------+--------+------------+--------------------------------+
```

14 . date_sub( ) –

DATE_SUB() is a function commonly used in SQL databases to substract a specified time interval to a date.

Syntax – date_sub(col_name,interval <value> <unit>)

values : n

unit : day , week , month , year

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

```
select eid,ename,doj,date_sub(doj,interval 1 month) from emp;
```

Output –

eid	ename	doj	date_sub(doj,interval 1 month)
1	Rajesh	2020-01-31	2019-12-31
2	Roha	2021-05-06	2021-04-06
3	Neel	2023-12-31	2023-11-30

Query –

```
select eid,ename,doj,date_sub(doj,interval 1 year) from emp;
```

Output –

eid	ename	doj	date_sub(doj,interval 1 year)
1	Rajesh	2020-01-31	2019-01-31
2	Roha	2021-05-06	2020-05-06
3	Neel	2023-12-31	2022-12-31

Query –

```
select eid,ename,doj,date_sub(doj,interval 2 day) from emp;
```

Output –

```
+------+--------+------------+--------------------------------+
| eid  | ename  | doj        | date_sub(doj,interval 2 day)   |
+------+--------+------------+--------------------------------+
|    1 | Rajesh | 2020-01-31 | 2020-01-29                     |
|    2 | Roha   | 2021-05-06 | 2021-05-04                     |
|    3 | Neel   | 2023-12-31 | 2023-12-29                     |
+------+--------+------------+--------------------------------+
```

Query –

```
select eid,ename,doj,date_sub(doj,interval 2 week) from emp;
```

Output –

```
+------+--------+------------+--------------------------------+
| eid  | ename  | doj        | date_sub(doj,interval 2 week)  |
+------+--------+------------+--------------------------------+
|    1 | Rajesh | 2020-01-31 | 2020-01-17                     |
|    2 | Roha   | 2021-05-06 | 2021-04-22                     |
|    3 | Neel   | 2023-12-31 | 2023-12-17                     |
+------+--------+------------+--------------------------------+
```

15. last_day( ) –

LAST_DAY() is a function that returns the last day of the month for a given date. It's commonly used to find the end date of a month.

Syntax – last_day(date) ;

Input –

```
+------+--------+------+------------+---------------------+----------+
| eid  | ename  | did  | doj        | rcd                 | act_flg  |
+------+--------+------+------------+---------------------+----------+
|    1 | Rajesh |   10 | 2020-01-31 | 2024-05-03 09:00:24 |        1 |
|    2 | Roha   |   20 | 2021-05-06 | 2024-05-03 09:00:24 |        1 |
|    3 | Neel   |   10 | 2023-12-31 | 2024-05-03 09:00:24 |        1 |
+------+--------+------+------------+---------------------+----------+
```

Query –

select eid,ename,doj,last_day(doj) from emp;

Output –

eid	ename	doj	last_day(doj)
1	Rajesh	2020-01-31	2020-01-31
2	Roha	2021-05-06	2021-05-31
3	Neel	2023-12-31	2023-12-31

16.datediff( ) –

DATEDIFF() function in SQL is used to find the difference between two dates. It returns the number of days between two dates.

Syntax - DATEDIFF(end_date, start_date) ;

Input –

eid	ename	did	doj	rcd	act_flg
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1

Query –

select eid,ename, datediff(rcd,doj) from emp;

Output –

eid	ename	datediff(rcd,doj)
1	Rajesh	1554
2	Roha	1093
3	Neel	124

17 . Str_to_date( ) –

STR_TO_DATE() function is used in MySQL to convert a string into a date value, according to a specified format. This function is particularly useful when you have date values stored as strings in a non-standard format and you need to convert them to a proper date type for manipulation or comparison.

Syntax - str_to_date(col_name,input date format) ;

Col_name – Column that represents the date.

Input date format - format is the format of the date in the string.

Input –

eid	ename	did	doj	rcd	act_flg	exit_date	sal
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1	12022022	20000
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1	12022022	25000
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1	12022022	30000

Query –

```
select eid,ename,year(str_to_date(exit_date,"%d%m%Y")) from emp;
```

Output –

eid	ename	year(str_to_date(exit_date,"%d%m%Y"))
1	Rajesh	2022
2	Roha	2022
3	Neel	2022

Group Function –

"group functions" in SQL, it likely means aggregate functions that operate on groups of rows and return a single result for each group

1.max( ) –

Finds the maximum value in a group.

Input –

eid	ename	did	doj	rcd	act_flg	exit_date	sal
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1	12022022	20000
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1	12022022	25000
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1	12022022	30000

Query –

```
select max(sal) from emp;
```

Output –

max(sal)
30000

2.min( ) –

Finds the minimum value in a group.

Input –

eid	ename	did	doj	rcd	act_flg	exit_date	sal
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1	12022022	20000
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1	12022022	25000
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1	12022022	30000

Query –

Select min (sal) from emp;

Output –

```
+-----------+
| min(sal)  |
+-----------+
|     20000 |
+-----------+
```

3.sum( ) –

Calculates the sum of values in a group.

Input –

```
+------+--------+------+------------+---------------------+---------+-----------+-------+
| eid  | ename  | did  | doj        | rcd                 | act_flg | exit_date | sal   |
+------+--------+------+------------+---------------------+---------+-----------+-------+
|    1 | Rajesh |   10 | 2020-01-31 | 2024-05-03 09:00:24 |       1 | 12022022  | 20000 |
|    2 | Roha   |   20 | 2021-05-06 | 2024-05-03 09:00:24 |       1 | 12022022  | 25000 |
|    3 | Neel   |   10 | 2023-12-31 | 2024-05-03 09:00:24 |       1 | 12022022  | 30000 |
+------+--------+------+------------+---------------------+---------+-----------+-------+
```

Query –

Select sum (sal) from emp ;

Output –

```
+-----------+
| sum(sal)  |
+-----------+
|     75000 |
+-----------+
```

4.avg( ) –

Computes the average of values in a group.

Input –

eid	ename	did	doj	rcd	act_flg	exit_date	sal
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1	12022022	20000
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1	12022022	25000
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1	12022022	30000

Query –

select avg(sal) from emp;

Output –

avg(sal)
25000.0000

5.count( ) –

Counts the number of rows in a group

Input –

eid	ename	did	doj	rcd	act_flg	exit_date	sal
1	Rajesh	10	2020-01-31	2024-05-03 09:00:24	1	12022022	20000
2	Roha	20	2021-05-06	2024-05-03 09:00:24	1	12022022	25000
3	Neel	10	2023-12-31	2024-05-03 09:00:24	1	12022022	30000

Query –

select count(sal) from emp;

Output –

count(sal)
3

Query –

select count(*) from emp;

Output –

```
+----------+
| count(*) |
+----------+
|        3 |
+----------+
```

Query –

select count(1) from emp;

Output –

```
+----------+
| count(1) |
+----------+
|        3 |
+----------+
```

Group by clause –

The "GROUP BY" clause is a feature in SQL (Structured Query Language) used to group rows of data based on common values in one or more columns.

When you have a table with lots of rows, the GROUP BY clause lets you group those rows based on the values in one or more columns. For example, if you have a table of sales transactions with columns for product names and sales amounts, you can use GROUP BY to group the transactions by product name.

IMP Points –

1. only group by columns we can select

2. with group by group function is mandatory

3. we can use more than one column for group by

4. if you wanted to filter group by result we use having

Input –

roll	name	marks	class
1	Karan	75	5
2	Rohan	56	5
3	Reet	87	5
4	Preet	67	5
1	Shree	98	6
2	Shri	45	6
3	Rakesh	60	6
4	Roha	57	6

Query –

```
select class,count(*) from std group by class;
```

Output –

class	count(*)
5	4
6	4

Query -

```
select class,avg(marks) from std group by class;
```

Output –

```
+-------+-------------+
| class | avg(marks)  |
+-------+-------------+
|     5 |     71.2500 |
|     6 |     65.0000 |
+-------+-------------+
```

Query –

```
select class,sum(marks) from std group by class;
```

Output –

```
+-------+-------------+
| class | sum(marks)  |
+-------+-------------+
|     5 |         285 |
|     6 |         260 |
+-------+-------------+
```

Query –

```
select class,max(marks) from std group by class;
```

Output –

```
+-------+-------------+
| class | max(marks)  |
+-------+-------------+
|     5 |          87 |
|     6 |          98 |
+-------+-------------+
```

Query –

select class,min(marks) from std group by class;

Output –

```
+-------+-------------+
| class | min(marks)  |
+-------+-------------+
|     5 |          56 |
|     6 |          45 |
+-------+-------------+
```

Query –

select class,max(marks),min(marks),sum(marks),avg(marks),count(*) from std group by class;

Output –

```
+-------+------------+------------+------------+------------+----------+
| class | max(marks) | min(marks) | sum(marks) | avg(marks) | count(*) |
+-------+------------+------------+------------+------------+----------+
|     5 |         87 |         56 |        285 |    71.2500 |        4 |
|     6 |         98 |         45 |        260 |    65.0000 |        4 |
+-------+------------+------------+------------+------------+----------+
```

Having Clause –

The "HAVING" clause in SQL works alongside the "GROUP BY" clause to filter groups of rows based on specified conditions.

After you've used the "GROUP BY" clause to group rows based on certain criteria, the "HAVING" clause allows you to specify conditions for those groups. It filters the grouped data based on aggregate conditions.

Input –

oid	cid	amt	status
1	101	100	D
2	101	200	I
3	101	100	D
4	101	200	I
5	101	500	Pre.
6	102	150	D
7	102	250	D
8	103	300	I
9	104	400	D
10	105	200	D

Query –

```
select cid,status,count(oid) from sales group by cid,status having status='D';
```

Output –

cid	status	count(oid)
101	D	2
102	D	2
104	D	1
105	D	1

Query –

select cid,count(oid) from sales group by cid,status having status='D' and cid=101;

Output –

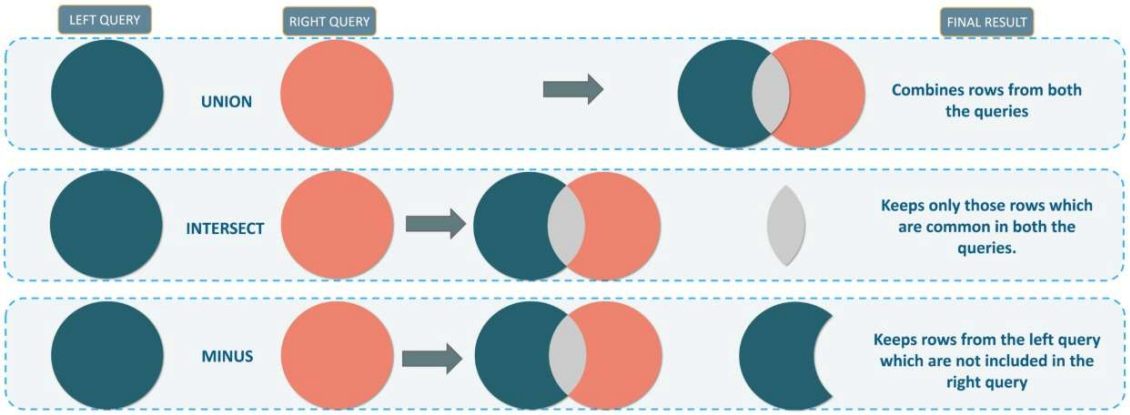
```
+------+------------+
| cid  | count(oid) |
+------+------------+
|  101 |          2 |
+------+------------+
```

Set Operator –

Set operators in SQL are used to combine the results of two or more SELECT statements into a single result set.

To merge 2 or more tables we use set operators

Note - Both tables should have same schema and data types and seq of columns



- **UNION ALL:**

Merge data from both table

The UNION ALL operator also combines the results of two or more SELECT statements into a single result set. However, it retains all rows, including duplicates, from the combined result set .

Input –

```
 select * from emp_mumbai;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

```
 select * from emp_pune;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
+------+--------+-------+
```

Query –

```
select * from emp_pune
UNION ALL
select * from emp_mumbai;
```

Output –

```
+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

- **Union –**

Merge data from tables and remove duplicate

The UNION operator combines the results of two or more SELECT statements into a single result set. It removes duplicate rows from the combined result set.

Input –

```
 select * from emp_mumbai;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

```
 select * from emp_pune;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
+------+--------+-------+
```

Query –

```
select * from emp_pune
 UNION
 select * from emp_mumbai;
```

Output –

```
+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

- **Intersect –**

matching/common / same records from both tables

The INTERSECT operator returns only the rows that appear in both result sets of two SELECT statements. It effectively finds the common elements between the two result sets.

Input –

```
 select * from emp_mumbai;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

```
 select * from emp_pune;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
+------+--------+-------+
```

Query –

select * from emp_pune

INTERSECT

select * from emp_mumbai;

Output –

```
       EID ENAME                                                    SAL
---------- ---------------------------------------------------- ----------
         1 Karan                                                 25000
         2 Rajesh                                                35000
```

- **MINUS –**

minus will give non matching / non common record from first table

It effectively finds the set difference between the two result sets.

Input –

```
 select * from emp_mumbai;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    4 | Peter  | 36000 |
+------+--------+-------+
```

```
 select * from emp_pune;

+------+--------+-------+
| eid  | ename  | sal   |
+------+--------+-------+
|    1 | Karan  | 25000 |
|    2 | Rajesh | 35000 |
|    3 | Ganesh | 45000 |
+------+--------+-------+
```

Query –

```
select * from emp_pune
MINUS
select * from emp_mumbai;
```

Output –

EID	ENAME	SAL
3	Ganesh	45000

Query –

```
select * from emp_mumbai
MINUS
select * from emp_pune;
```

Output –

EID	ENAME	SAL
4	Peter	36000

If Schema is not same how to do set Operation?

Input -

ID	NAME	SAL	CITY
1	sanket	30000	
2	ajit	40000	
3	priyanka	50000	

ID	NAME	SAL	MOB
1	sanket	30000	
2	ajit	40000	
3	akanksha	37000	

In this scenario, the schemas of the two tables are different. To perform the UNION operation, we need to align the schemas.
In the given example, the UNION operator is used to find the unique records between the two tables, emp1 and emp2, based on certain columns.

Query –

```
select id,name,sal,city,null as mob from emp1
union
select id,name,sal,null as city,mob from emp2;
```

emp1 and emp2 tables:

Both tables have columns id, name, sal, city (in emp1), and mob (in emp2).

Selecting columns:

The query selects specific columns from each table:
From emp1: id, name, sal, city (with NULL as mob).
From emp2: id, name, sal, mob (with NULL as city).

UNION operation:

The UNION operator shows only unique records .

Output –

ID	NAME	SAL	CITY	MOB
1	sanket	30000		
2	ajit	40000		
3	akanksha	37000		
3	priyanka	50000		

Result:

emp1 contains records for employees "sanket", "ajit", and "priyanka".

emp2 contains records for employees "sanket", "ajit", and "akanksha".

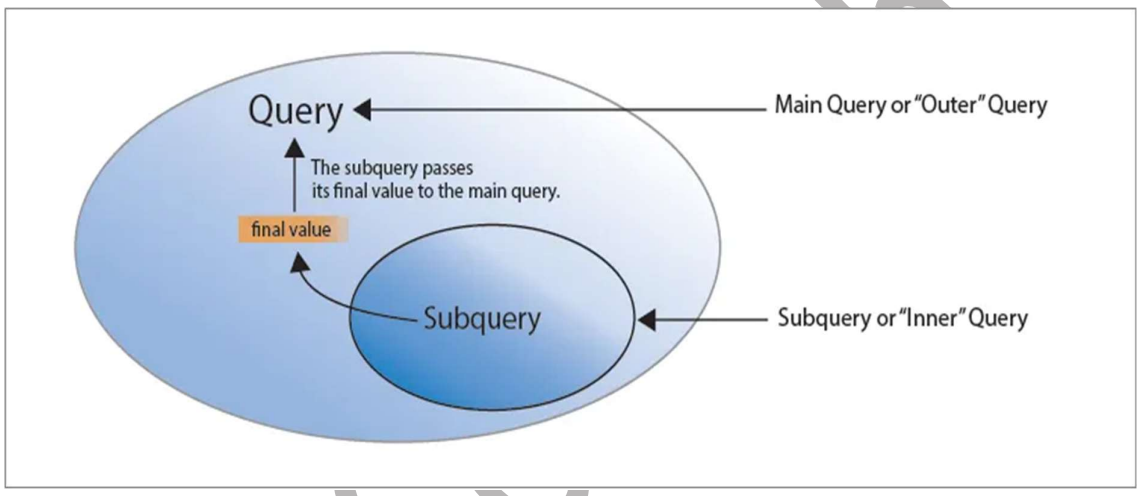
The union of these two sets results in only the common records, which are "sanket" and "ajit", “akanksha” and “Priyanka .

- **Subquery –**

  Query within query

  Subquery is also known as inner query or nested query.

  Subqueries are used to return data that will be used in the main query's execution.



Input –

```
select * from emp ;
```

EID	ENAME	DID	SAL
1	John	101	50000
2	Alice	102	60000
3	Bob	103	55000
4	Emily	104	62000
5	David	105	53000
6	Samantha	101	58000
7	Michael	102	65000
8	Emma	103	59000
9	James	104	63000
10	Sophia	105	54000

```
select * from dept ;

       DID DNAME
---------- -------------
       101 HR
       102 Finance
       103 Marketing
       104 Engineering
       105 Sales
       104 Engineering
       102 Finance
       103 Marketing
       104 Engineering
       105 Sales
```

Query –

Select * from emp where did in(select did from dept);

Output –

```
       EID ENAME                DID        SAL
---------- ---------- ---------- ----------
         6 Samantha            101      58000
         1 John                101      50000
         7 Michael             102      65000
         2 Alice               102      60000
         8 Emma                103      59000
         3 Bob                 103      55000
         9 James               104      63000
         4 Emily               104      62000
        10 Sophia              105      54000
         5 David               105      53000
```

Query –

Select * from emp where did in(101,103);

Output –

EID	ENAME	DID	SAL
1	John	101	50000
3	Bob	103	55000
6	Samantha	101	58000
8	Emma	103	59000

Query –

Select ename,dname from (select * from emp inner join dept on emp.did =dept.did)A;

Output –

ENAME	DNAME
Samantha	HR
John	HR
Michael	Finance
Alice	Finance
Emma	Marketing
Bob	Marketing
James	Engineering
Emily	Engineering
Sophia	Sales
David	Sales
James	Engineering
Emily	Engineering
Michael	Finance
Alice	Finance
Emma	Marketing
Bob	Marketing
James	Engineering
Emily	Engineering
Sophia	Sales
David	Sales

- **Windowing Function –**

  Windowing function in SQL allow to perform calculations across a set of rows related to the current row,known as the "window" without collapsing the result set.

  It typically includes 'PARTITION BY' and 'ORDER BY' clauses

  'PARTITION BY' divides the result set into partitions or groups.

  'ORDER BY' specifies the order of rows within each partition.

  Types of windowing functions-

  1. ROW_NUMBER()
  2. RANK()
  3. DENSE_RANK()
  4. LAG()
  5. LEAD()

ROW_NUMBER( ) –

This is used for giving numbering for all rows from 1....to onward

Input –

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query –

select eid,ename,did,sal,row_number() over(order by sal desc) as rn from empw;

Output –

EID	ENAME	DID	SAL	RN
6	Peter	20	40000	1
7	Bond	20	40000	2
5	Mukesh	20	35000	3
4	Neel	10	31000	4
2	Rohan	10	26000	5
3	Rakesh	10	26000	6
1	Karan	10	25000	7

Query –

select eid,ename,did,sal,row_number() over(partition by did order by sal desc) as rn from empw;

Output –

EID	ENAME	DID	SAL	RN
4	Neel	10	31000	1
2	Rohan	10	26000	2
3	Rakesh	10	26000	3
1	Karan	10	25000	4
7	Bond	20	40000	1
6	Peter	20	40000	2
5	Mukesh	20	35000	3

Input –

```
NAME
----------
Pune
Pune
Pune
Pune
Pune
Mumbai
Mumbai
Mumbai
Mumbai
Mumbai
Pune
Pune
Mumbai
Mumbai
Mumbai
```

Query-

select name from (select name,row_number() over(partition by name order by name) as rn from city)A order by rn asc,name desc;

Output-

```
NAME
----------
Pune
Mumbai
Pune
Mumbai
Pune
Mumbai
Pune
Mumbai
Pune
Mumbai
Pune
Mumbai
Pune
Mumbai
Mumbai
```

RANK() -

it gives rank or number for each row if the same value in next row it gives same rank as above and it skips the number

Input-

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query-

```
select eid,ename,did,sal,rank() over(order by sal desc) as rnk from empw;
```

Output-

EID	ENAME	DID	SAL	RNK
6	Peter	20	40000	1
7	Bond	20	40000	1
5	Mukesh	20	35000	3
4	Neel	10	31000	4
2	Rohan	10	26000	5
3	Rakesh	10	26000	5
1	Karan	10	25000	7

Query-

select eid,ename,did,sal,rank() over(partition by did order by sal desc) as rnk from empw;

Output –

EID	ENAME	DID	SAL	RNK
4	Neel	10	31000	1
2	Rohan	10	26000	2
3	Rakesh	10	26000	2
1	Karan	10	25000	4
7	Bond	20	40000	1
6	Peter	20	40000	1
5	Mukesh	20	35000	3

DENSE_RANK( ) –

it assigns the rank with numbering wise or assigns same rank for the same value in immediate rows

Input –

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query-

select eid,ename,did,sal,dense_rank() over(order by sal desc) as drnk from empw;

Output-

EID	ENAME	DID	SAL	DRNK
6	Peter	20	40000	1
7	Bond	20	40000	1
5	Mukesh	20	35000	2
4	Neel	10	31000	3
2	Rohan	10	26000	4
3	Rakesh	10	26000	4
1	Karan	10	25000	5

Query -

select * from (select eid,ename,did,sal,dense_rank() over(order by sal) as drn from empw)A where drn in (3,5,7,9);

Output –

EID	ENAME	DID	SAL	DRN
4	Neel	10	31000	3
6	Peter	20	40000	5
7	Bond	20	40000	5

Input –

PID	PNAME	TRN_DATE	QTY	TOTALAMT
1	Mobile	01-MAY-24	5	100
2	TV	02-MAY-24	3	75
3	Laptop	03-MAY-24	8	200
4	Tablet	04-MAY-24	2	50
5	Speaker	05-MAY-24	6	150
1	Mobile	06-MAY-24	4	80
2	TV	07-MAY-24	7	175
3	Laptop	08-MAY-24	3	75
4	Tablet	09-MAY-24	5	125
5	Speaker	10-MAY-24	2	50
1	Mobile	11-MAY-24	6	120
2	TV	12-MAY-24	4	100
3	Laptop	13-MAY-24	9	225
4	Tablet	14-MAY-24	3	75
5	Speaker	15-MAY-24	7	175
1	Mobile	16-MAY-24	3	60
2	TV	17-MAY-24	8	200
3	Laptop	18-MAY-24	5	125
4	Tablet	19-MAY-24	4	100
5	Speaker	20-MAY-24	6	150

Query –

```
Select  *  from ( selectpid,pname,TRN_DATE,QTY,TOTALAMT,
dense_rank() over(partition by pid order by QTY desc)as drn from sales)
A where drn=1;
```

Output –

PID	PNAME	TRN_DATE	QTY	TOTALAMT	DRN
1	Mobile	11-MAY-24	6	120	1
2	TV	17-MAY-24	8	200	1
3	Laptop	13-MAY-24	9	225	1
4	Tablet	09-MAY-24	5	125	1
5	Speaker	15-MAY-24	7	175	1

Query –

select * from (select pid,pname,trn_date,qty,totalamt,dense_rank() over(partition by pid order by totalamt asc) as drn from sales)A where drn=2;

Output –

PID	PNAME	TRN_DATE	QTY	TOTALAMT	DRN
1	Mobile	06-MAY-24	4	80	2
2	TV	12-MAY-24	4	100	2
3	Laptop	18-MAY-24	5	125	2
4	Tablet	14-MAY-24	3	75	2
5	Speaker	05-MAY-24	6	150	2
5	Speaker	20-MAY-24	6	150	2

LAG( ) –

it prefers the previous row to give value, it will write the value from previous choose row

Input –

id	pname	score
1	Sachin	100
2	Sachin	120
3	Sachin	250
4	Sachin	120
1	Rohit	200
2	Rohit	130
3	Rohit	122
5	Rohit	124
1	MS	220
2	MS	120
3	MS	100
5	MS	100

Query –

select id,pname,score,LAG(score,1,0) over(partition by pname order by id) pre_score from player_scores;

Output –

ID	PNAME	SCORE	PRE_SCORE
1	MS	220	0
2	MS	120	220
3	MS	100	120
5	MS	100	100
1	Rohit	200	0
2	Rohit	130	200
3	Rohit	122	130
5	Rohit	124	122
1	Sachin	100	0
2	Sachin	120	100
3	Sachin	250	120
4	Sachin	120	250

LEAD( ) –

it prefers the next row to give value, it will write the value from next choose row

Input –

id	pname	score
1	Sachin	100
2	Sachin	120
3	Sachin	250
4	Sachin	120
1	Rohit	200
2	Rohit	130
3	Rohit	122
5	Rohit	124
1	MS	220
2	MS	120
3	MS	100
5	MS	100

Query –

select id,pname,score,LEAD(score,1,0) over(partition by pname order by id) pre_score from player_scores;

Output –

ID	PNAME	SCORE	PRE_SCORE
1	MS	220	120
2	MS	120	100
3	MS	100	100
5	MS	100	0
1	Rohit	200	130
2	Rohit	130	122
3	Rohit	122	124
5	Rohit	124	0
1	Sachin	100	120
2	Sachin	120	250
3	Sachin	250	120
4	Sachin	120	0

Query –

select eid,ename,did,sal,row_number() over(order by sal desc) as rn,rank() over(order by sal desc) as rnk,dense_rank() over(order by sal desc) as drnk from empw ;

Output –

EID	ENAME	DID	SAL	RN	RNK	DRNK
6	Peter	20	40000	1	1	1
7	Bond	20	40000	2	1	1
5	Mukesh	20	35000	3	3	2
4	Neel	10	31000	4	4	3
2	Rohan	10	26000	5	5	4
3	Rakesh	10	26000	6	5	4
1	Karan	10	25000	7	7	5

CTE (Common Table Expression) –

It is temporary result set that you can reference within a SELECT,INSERT,UPDATE, or DELETE statement in SQL
Its like creating a virtual table that exists only for the duration of the query.

It is a function used for making or for considering temporary tables which is used for the next query purpose .

Syntax –

with <consider table name/alise> as (any select query) select <col name>, <function> from <table name>

group by <column name>;

Input –

```
select * from emp ;
```

EID	ENAME	DID	SAL
1	John	101	50000
2	Alice	102	60000
3	Bob	103	55000
4	Emily	104	62000
5	David	105	53000
6	Samantha	101	58000
7	Michael	102	65000
8	Emma	103	59000
9	James	104	63000
10	Sophia	105	54000

```
select * from dept ;
```

DID	DNAME
101	HR
102	Finance
103	Marketing
104	Engineering
105	Sales
104	Engineering
102	Finance
103	Marketing
104	Engineering
105	Sales

Query –

```
With temp as (select eid,ename,did from emp where did=101)
Select did,count(*) from temp group by did;
```

Output –

DID	COUNT(*)
101	2

Query –

```
With A as(select * from emp inner join dept on emp.did=dept.did) ,
B as (select * from emp inner join dept on emp.did=dept.did) ,
C as (select * from emp inner join dept on emp.did=dept.did)
Select ename,dname from A union all select ename,dname from B
union all select ename,dname from C;
```

Output –

ENAME	DNAME
Samantha	HR
John	HR
Michael	Finance
Alice	Finance
Emma	Marketing
Bob	Marketing
James	Engineering
Emily	Engineering
Sophia	Sales
David	Sales
James	Engineering

ENAME	DNAME
Emily	Engineering
Michael	Finance
Alice	Finance
Emma	Marketing
Bob	Marketing
James	Engineering
Emily	Engineering
Sophia	Sales
David	Sales
Samantha	HR
John	HR

ENAME	DNAME
Michael	Finance
Alice	Finance
Emma	Marketing
Bob	Marketing
James	Engineering
Emily	Engineering
Sophia	Sales
David	Sales
James	Engineering
Emily	Engineering

```
Sophia      Sales
David       Sales
James       Engineering
Emily       Engineering
Michael     Finance

ENAME       DNAME
----------  -------------
Alice       Finance
Emma        Marketing
Bob         Marketing
James       Engineering
Emily       Engineering
Sophia      Sales
David       Sales
Samantha    HR
John        HR
Michael     Finance
Alice       Finance

ENAME       DNAME
----------  -------------
Emma        Marketing
Bob         Marketing
James       Engineering
Emily       Engineering
Sophia      Sales
David       Sales
James       Engineering
Emily       Engineering
Michael     Finance
Alice       Finance
Emma        Marketing

Bob         Marketing
James       Engineering
Emily       Engineering
Sophia      Sales
David       Sales
```

VIEW –

It means we can create a table as view with selected columns data

There are 3 types of views –

1.simple view

2.complex view

3. Materialized view

Why we create view ?

1.Security

2.To hide Complexity

Simple view –

in this we create view as select statement or may be join two or more table but it not perform complex opration like joins opration,group by,windowing function,oprators etc

Syntax –

create view <table name> as <select statment>;

Input -

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query –

create view empw1 as select eid,ename,did from empw where did=10;

Output –

```
+------+--------+------+
| eid  | ename  | did  |
+------+--------+------+
|    1 | Karan  |   10 |
|    2 | Rohan  |   10 |
|    3 | Rakesh |   10 |
|    4 | Neel   |   10 |
+------+--------+------+
```

Complex View –

in this we create view as select statement & it perform complex opration like joins opration,group by,windowing function,oprators etc

Syntax –

create view <table name> as <select statment>;

Input –

```
+------+--------+-------+------+
| eid  | ename  | sal   | did  |
+------+--------+-------+------+
|    1 | Karan  | 25000 |   10 |
|    2 | Rohan  | 26000 |   10 |
|    3 | Rakesh | 26000 |   10 |
|    4 | Neel   | 31000 |   10 |
|    5 | Mukesh | 35000 |   20 |
|    6 | Peter  | 40000 |   20 |
|    7 | Bond   | 40000 |   20 |
+------+--------+-------+------+
```

```
+------+-------+
| ddid | dname |
+------+-------+
|   10 | IT    |
|   20 | RD    |
|   50 | DM    |
+------+-------+
```

Query –

create view emp_dtl as select eid,ename,dname from empw inner join dept on empw.did=dept.ddid;

Output –

eid	ename	dname
1	Karan	IT
2	Rohan	IT
3	Rakesh	IT
4	Neel	IT
5	Mukesh	RD
6	Peter	RD
7	Bond	RD

Materialized view –

It's special type of view which holds view query , result into the memory so performance will get improve for next time query run

It Supported only in oracle

Syntax –

create materialized view <table name> as <select statment>;

Input –

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query –

create materialized view d1 as select did,count(*) from empw group by did;

Output –

```
       DID   COUNT(*)
---------- ----------
        20          3
        10          4
```

Creating tables by using diff statement types –

1.create new table as select statement

Syntax –

create <table name> as select statment;

Input –

```
+------+--------+-------+------+
| eid  | ename  | sal   | did  |
+------+--------+-------+------+
|    1 | Karan  | 25000 |   10 |
|    2 | Rohan  | 26000 |   10 |
|    3 | Rakesh | 26000 |   10 |
|    4 | Neel   | 31000 |   10 |
|    5 | Mukesh | 35000 |   20 |
|    6 | Peter  | 40000 |   20 |
|    7 | Bond   | 40000 |   20 |
+------+--------+-------+------+
```

Query –

Create talbe emp1 as select eid,ename from emp;

Output –

```
       EID ENAME
---------- ----------
         1 Karan
         2 Rohan
         3 Rakesh
         4 Neel
         5 Mukesh
         6 Peter
         7 Bond
```

2.Creating Similar table from existing table –

Syntax –

Create <table name> as select * from <table name>;

Input –

```
+------+--------+-------+------+
| eid  | ename  | sal   | did  |
+------+--------+-------+------+
|    1 | Karan  | 25000 |   10 |
|    2 | Rohan  | 26000 |   10 |
|    3 | Rakesh | 26000 |   10 |
|    4 | Neel   | 31000 |   10 |
|    5 | Mukesh | 35000 |   20 |
|    6 | Peter  | 40000 |   20 |
|    7 | Bond   | 40000 |   20 |
+------+--------+-------+------+
```

Query –

Create emp1 as select * from emp;

Output –

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

3.Insert data from existing table with select statement –

Syntax –

Insert into <table_name> select statement ;

Input

EID	ENAME
1	Karan
2	Rohan
3	Rakesh
4	Neel
5	Mukesh
6	Peter
7	Bond

Query –

insert into emp1(eid,ename) select eid,ename from empw where eid in(1,2,3,4);

Output –

```
       EID ENAME
---------- ----------
         1 Karan
         2 Rohan
         3 Rakesh
         4 Neel
         5 Mukesh
         6 Peter
         7 Bond
         1 Karan
         2 Rohan
         3 Rakesh
         4 Neel
```

4.How to create similar table with only column not included data –

Syntax –

create <table name> as select statment <any fault condtion>;

Input –

eid	ename	sal	did
1	Karan	25000	10
2	Rohan	26000	10
3	Rakesh	26000	10
4	Neel	31000	10
5	Mukesh	35000	20
6	Peter	40000	20
7	Bond	40000	20

Query –

Create tabl emp as select * from emp1 where 1=2;

Output –

Empty set ;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EID                                                NUMBER(38)
 ENAME                                              VARCHAR2(10)
 SAL                                                NUMBER(38)
 DID                                                NUMBER(38)
```

5.How to create not exists table –

Syntax-

create table if not exists <table name>(creating column name & datatypes);

Query –

Create table if not exists emp_new(eid int , ename varchar(10));

Output –

Empty set ;

```
desc emp_new;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| eid   | int(11)     | YES  |     | NULL    |       |
| ename | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

SQL SCRIPT –

A file which having sql statements written in it and saved with .sql extensions . if we execute scripts it will run all queries defined in script .

Example – file name save as = myscript.sql

```
create database rhushi;

use rhushi;

create table emp (eid int,ename varchar(15),sal int);

insert into emp values(1,'rhushi',45000);
insert into emp values(2,'mukund',35000);
insert into emp values(3,'rakesh',25000);
insert into emp values(4,'sham',15000);

commit;
```

For Execute – Source D:/folder name/myscript.sql

Normalization –

Normalization is a database design technique used to organize tables and minimize redundancy.
It ensures that each table in a database serves a single purpose and that data is logically stored to reduce data duplication and maintain data integrity.

Reducing data redundancy: By dividing data into multiple tables and eliminating redundant data, normalization minimizes storage space and ensures consistency.

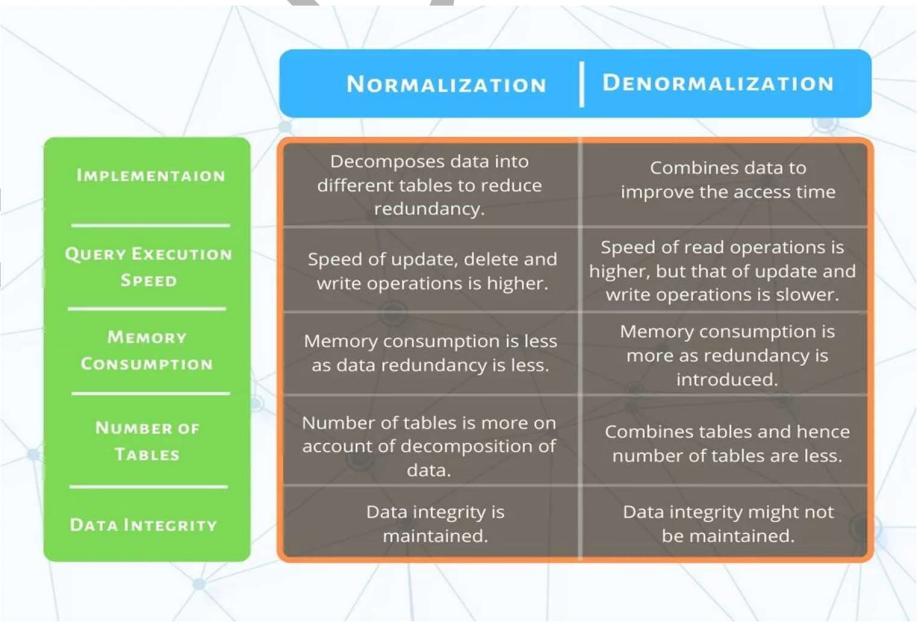
Improving data integrity: By organizing data logically and adhering to normalization rules, data integrity is maintained, reducing the likelihood of anomalies.

Simplifying updates: With normalized tables, updates to the database are simpler and less error-prone because data is stored in a structured manner.

Denormalization –

Denormalization is a database optimization technique where redundancy is intentionally introduced into the data model.
This is done to improve query performance by reducing the need for joining multiple tables.

Denormalization involves adding redundant data or aggregating data from multiple tables into a single table. By doing so, complex queries that would require joins across multiple tables can be simplified, resulting in faster query execution.



	Normalization	Denormalization
Implementaion	Decomposes data into different tables to reduce redundancy.	Combines data to improve the access time
Query Execution Speed	Speed of update, delete and write operations is higher.	Speed of read operations is higher, but that of update and write operations is slower.
Memory Consumption	Memory consumption is less as data redundancy is less.	Memory consumption is more as redundancy is introduced.
Number of Tables	Number of tables is more on account of decomposition of data.	Combines tables and hence number of tables are less.
Data Integrity	Data integrity is maintained.	Data integrity might not be maintained.