

STRUCTURED QUERY LANGUAGE



BY



SQUIRRELS

INSPIRE. LEARN. GROW.

TOPICS TO BE COVERED:

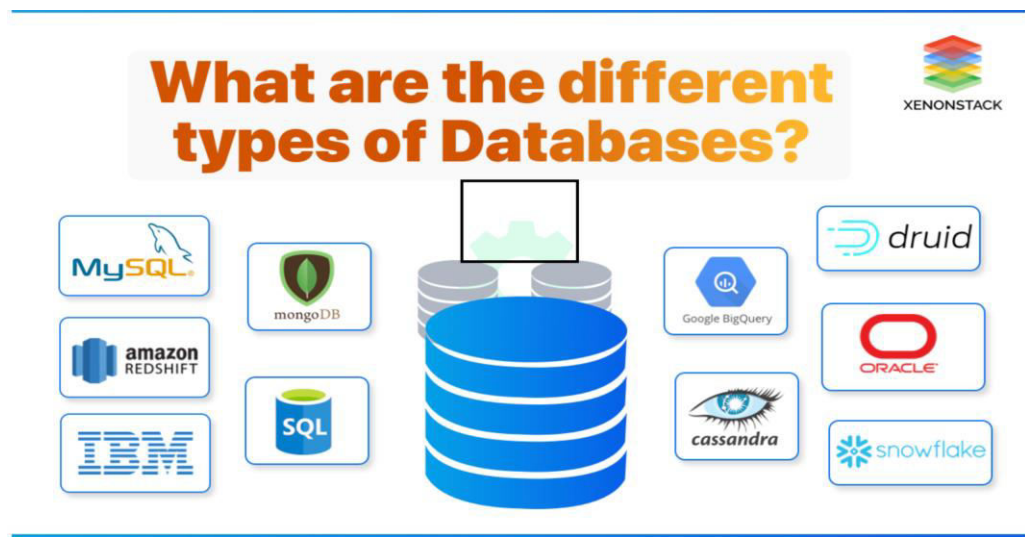
1. WHAT IS A DATABASE
2. DATA TYPES IN SQL
3. TYPES OF SQL COMMANDS- PRACTICAL APPROACH
4. DISTINCT KEYWORD & COUNT KEYWORD
5. KEYS & CONSTRAINTS
6. WHERE CLAUSE
7. LIMIT CLAUSE
8. ORDER BY CLAUSE
10. GROUP BY CLAUSE
11. HAVING BY CLAUSE
12. GENERAL ORDER OF THE CLAUSE
13. JOINS
14. UNION
15. VIEWS & STORED PROCEDURES
15. FUNCTIONS AGGREGATE FUNCTIONS
16. STRING FUNCTIONS
17. SUB QUERY IN SQL/WITH CLAUSE
18. COMMON TABLE EXPRESSIONS/WITH CLAUSE
19. WINDOWS FUNCTIONS
20. CASE WHEN



1. WHAT IS A DATABASE?

A database is a structured collection of data that is organised in a way to easily retrieve, manage, and update information.

1.1 TYPES OF DATABASES:



1. Relational Databases:

In Relational database or RDMS A relational database is the most used database. It contains several tables, and each table has its primary key.

Due to a collection of an organised set of tables, data can be accessed easily in RDBMS.

Example: MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle Database.

Column-family stores (Cassandra),

Graph databases (Neo4j).

2. Non-Relational database or NoSQL Databases:

Suitable for handling large amounts of unstructured or semi-structured data.

Types: Document-oriented (MongoDB),

Key-value stores (Redis),

Column-family stores (Cassandra),

Graph databases (Neo4j).

2. DATA TYPES IN SQL:

A data type is a type/category or attribute which defines the type of data in the table.

A table is composed of columns, rows,

In MySQL there are three main data types: string, numeric, and date and time.

2.1 DATATYPE:

- **CHAR:** represents a single fixed character data type but CHAR when definite with size as CHAR(SIZE) hold the data upto the mentioned fixed length.
Eg: CHAR (50)

- **VARCHAR:** character Data type with variable length, where you must define the length of the character and the upon defining maximum size it stores up to the defined size.

VARCHAR (50), here we can store up to 50.

INT: Keyword INT is used to define integer/numeric value data type.

- **FLOAT:** For storing decimal values
- **DATE AND TIME:** Date and time data type consists of

There are highly used data types, although there are numerous data types.

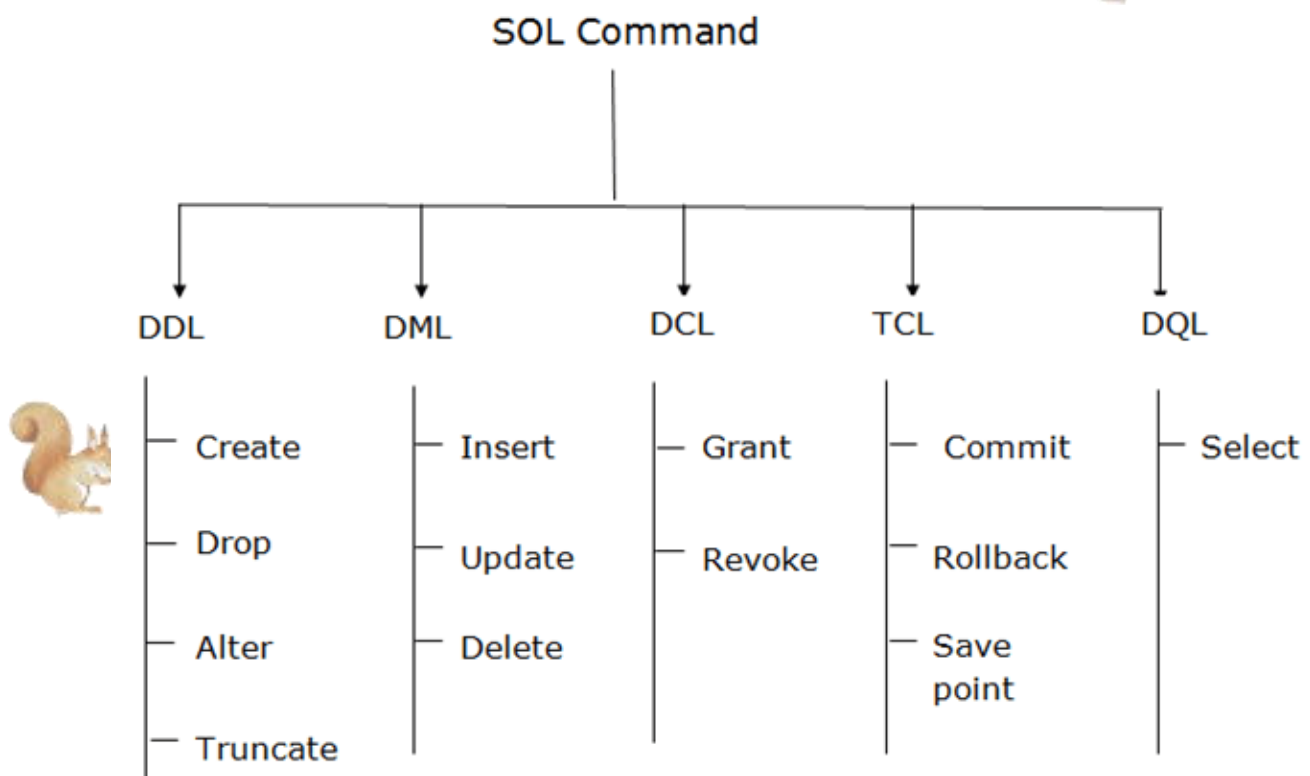
3.TYPES OF SQL COMMANDS:

SQL or Structured Query Language is used to operate on the data stored in a database. To query or interact with the database to extract information we use SQL commands or statements composed of syntax.

SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



3.1.1 DATA DEFINITION LANGUAGE: Data definition

language is used for performing operations like delete, alter/modify the data, create the structure of the table. All the commands of DDL are auto committed, which means it permanently saves all the changes in the database.

Here are some commands that come under DDL:

1.CREATE COMMAND: SQL CREATE TABLE statement is used to create tables in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

SYNTAX:

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

The column parameters specify the names of the columns of the table. The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

2. ALTER COMMAND: The ALTER table statement is used to add, delete, or modify columns in an existing table. The ALTER table statement is also used to add and drop various constraints on an existing table

SYNTAX:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

3.DROP COMMAND: It is used to delete both the structure and record stored in the table.

SYNTAX:


```
DROP TABLE table_name;
```

4.TRUNCATE COMMAND:

A truncate SQL statement is used to remove all rows (complete data) from a table. It is like the DELETE statement without the WHERE clause. The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

SYNTAX:

```
TRUNCATE TABLE table_name;
```



TRUNCATE TABLE Vs DELETE TABLE: The DELETE statement is used when we want to remove some or all the records from the table, while the TRUNCATE statement will delete entire rows from a table.

TRUNCATE TABLE Vs DROP TABLE: Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

3.1.2 DATA MANIPULATION LANGUAGE:

Commands such as Insert, Update, Delete are a part of DML

INSERT COMMAND:

1.It is used to insert data into the table, records can be inserted into the single row, or multiple rows.

2.Each value in the records we are inserting in a table using this statement should be of the same datatype as the respective column and satisfy the constraints of the column (if any). The values passed using an insert statement should match the number of columns in the table or, the number of columns mentioned in the current query. If any of these conditions are not satisfied, this statement generates an error.

SYNTAX:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

UPDATE COMMAND:

Update command is used to modify data in the table in columns or rows.

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


DELETE COMMAND: The delete statement is used to delete existing record in the table.

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

3.1.3DCL COMMANDS:

DCL (Data Control Language)

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

GRANT:

This command gives users access privileges to the database.

Syntax:

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

REVOKE: This command withdraws the user's access privileges given by using the GRANT command.

Syntax:

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

3.1.4 TCL (Transaction Control Language)

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions [here](#). Hence, the following TCL commands are used to control the execution of a transaction:

BEGIN: Opens a Transaction.

COMMIT: Commits a Transaction.

ROLLBACK: Rollbacks a transaction in case of any error occurs.

SAVEPOINT: Sets a save point within a transaction.

Syntax:

```
SAVEPOINT Savepoint_name;
```

3.2 PRACTICAL APPROACH:

Let's look at the practical approach to interact with database.

1. CREATE DATABASE: Input the desired name for the database

*** SQL is not case sensitive, but generally Upper case should be used for better visibility ***

SYNTAX for creating a database TEMP1(Any name can be chosen)

```
CREATE DATABASE TEMP1;
```

***** Using ; is mandatory, it acts as a full stop for the sql command

2. DELETE :


To delete database: Drop command deletes the entire table along with the design/structure of the table .

```
DROP DATABASE TEMP1;
```

```
1 create database TEMP1;
2 DROP DATABASE TEMP1;
3
4
5
6 CREATE DATABASE TEMP2;
7 DROP DATABASE TEMP2;
8
9 CREATE DATABASE COLLEGE
```

3.3 TABLE RELATED QUERIES:

**** Now, before creating table, let's call the database first where we want to create our table by using USE command, followed by create table command:



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

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

In simple words, we have defined the schema of the table or designed the table.

1.CREATING A TABLE:

```
CREATE TABLE student(  
rollno INT PRIMARY KEY,  
name VARCHAR(50)  
);
```

2.SELECT COMMAND:

Select command to view entire data in the table use select command

```
SELECT * from TABLE_NAME;
```

To select data from columns, enter data base

Ex: Consider a table Customers,

To extract all the data from CustomerName, City run the command as below:

```
SELECT CustomerName,City FROM Customers;
```

3. INSERT INTO COMMAND:

The SQL INSERT INTO Statement is used to add new rows of data into a table in the database. Almost all the RDBMS provide this SQL query to add the records in database tables.

Each value in the records we are inserting in a table using this statement should be of the same datatype as the respective column and satisfy the constraints of the column (if any). The values passed using an insert statement should match the number of columns in the table or, the number of columns mentioned in the current query. If any of these conditions are not satisfied, this statement generates an error

SYNTAX:

There are two basic syntaxes of the SQL INSERT INTO statement which are shown below:

1. With Column names

```
INSERT INTO TABLE_NAME (column1, column2...columnN)
VALUES
(value1, value2...valueN);
```

EXAMPLE: Consider a table studentsdata with columns rollno and name, marks

```
CREATE TABLE studentsdata;
```

```
INSERT INTO studentsdata
    (rollno, name, marks)
```

```
VALUES
```

```
(1, "ADITI", 23),
```

```
(2, "PRIYANKA", 85),
```

```
(3, "ZUNAID",87),
```

```
(4, "ADITI", 90),
```

```
(5, "YAMINI", 78),
```

```
(6, "ANI", 67),
```

```
(7, "EKTA", 55)
```

```
;
```

OUTPUT:

```
1|ADITI|23
```

```
2|PRIYANKA|85
```

```
3|ZUNAID|87
```

```
4|ADITI|90
```

```
5|YAMINI|78
```

```
6|ANI|67
```

```
7|EKTA|55
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

2. Without column names.

Make sure the order of the values is in the same order as the columns in the table.

```
INSERT INTO TABLENAME
```

```
VALUES(value1, value2.....valueN);
```

4. UPDATE COMMAND:

Update command is used to update existing data in the row/rows,

"Set" Keyword is used to modify the old data into new data.

SYNTAX:

```
UPDATE TableName,
```

```
Set col1 = value 1, col2 = value2
```

```
WHERE Condition;
```

Example,

If the table studentsdata has rollno, name, city, grade, marks and we want to upgrade grade "A" to grade "O" , we will run the following query

```
SELECT studentsdata,
```

```
SET Grade = "O"
```

```
WHERE Grade = "A";
```

5.ALTER COMMAND:

Alter command is used to alter or modify the design of the schema or table , we can perform certain actions like Add column, drop column, rename column,

Let's add a column into the table:

SYNTAX:

```
ALTER TABLE Table_name
```

```
ADD Column_name datatype constraint;
```

6. RENAME TABLE:

SYNTAX:

```
ALTER TABLE Table_name
```

```
RENAME TO new_table_name;
```

7. DROP COMMAND

Let's drop a column

```
ALTER TABLE Table_name  
DROP Column_name;
```

8. CHANGE Column(rename:)

```
ALTER TABLE table_name  
CHANGE COLUMN old_name new_name new_datatype new_constraint
```

9.MODIFY COMMAND (column-modify datatype/constraint):

```
ALTER TABLE table_name  
MODIFY col_name new_datatype new_constraint
```

10.TRUNCATE COMMAND:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE table_name studentsdata;  
This will give blank values under the columns
```

11.DELETE COMMAND:

As the name suggests, the delete command is used to delete the data from the existing rows.

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

Consider the table `studentsdata`, where we want to delete the record of the student whose marks are less than 60, the command will be written as follows:

Now, let's write the code for deleting the rows of students whose marks are less than 60

BEFORE :

1|ADITI|23

2|PRIYANKA|85

3|ZUNAID|87

4|ADITI|90

5|YAMINI|78

6|ANI|67

7|EKTA|55



Now, the command and result will be:

Here the students with less than marks 60 will be deleted.

```
DELETE FROM studentsdata WHERE marks < 60;
```

OUTPUT :

2|PRIYANKA|85

3|ZUNAID|87

4|ADITI|90

5|YAMINI|78

6|ANI|67

4.DISTINCT KEYWORD IN SELECT STATEMENT:

Using the distinct keyword gives the unique value for instead of the repeating values

For example, here we will get only distinct, different or unique values.

In the example below we have 21 distinct records, here only if we use the keyword COUNT, it will give is the count of distinct records.

```
SELECT Count(DISTINCT) Country FROM Customers;
```

Here, we get 21 distinct records,

whereas in SELECT COUNTRY FROM Customers, give a total of 91 records.

Number of Records: 21



Country
Argentina
Austria
Belgium
Brazil
Canada
Denmark
Finland
France

Number of Records: 91

Country
Germany
Mexico
Mexico
UK
Sweden
Germany
France
Spain

5.KEYS IN SQL:

Keys in SQL refer to the special columns in tables inside the database, there are 2 important keys in the table

1. PRIMARY KEY
2. FOREIGN KEY

1. PRIMARY KEY: In simple terms, PRIMARY KEYS = UNIQUE + NOT NULL

It is a column or set of columns which uniquely identifies each row,

Primary key should contain the unique values and no null values, or in other words, the column which is selected as the primary key should have all the unique values and no null values.

Eg , customer_ID, phone number in a table

```
CREATE TABLE Temp1(
```

```
Id INT,
```

```
name VARCHAR(50),
```

```
city VARCHAR(50),
```

```
age INT,
```

```
PRIMARY KEY(Id, name)
```

```
);
```

We can make a combination of 2 columns as the primary key because the values within the columns can be duplicated but the combination can't be.

2.FOREIGN KEY:

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

It helps establish the connection between both the tables.

There can be multiple foreign keys

They can be duplicate and null values.

** There are various other keys such as Super key, Candidate keys in SQL,

```
CREATE TABLE Temp(
```

```
  Cust_id INT,
```

```
  FOREIGN KEY(cust_id) references CUSTOMER(id)
```

Here cust_id = Foreign key of the new table, the one mentioned before reference keyword.

CUSTOMER(id) = Table customer and id is the primary key of the customer table

Primary key is a type of candidate key

5.1 CONSTRAINTS:

Constraints in SQL are used to specify rules or conditions in a table, we do that by using Keywords.

There are multiple constraints in SQL but let's look at the most important constraint:

1. **NOT NULL:** Where we define not null means the columns cannot have the null value. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.
2. **UNIQUE:** As the name says, all the values in the column should be unique or different, no duplicate values
3. **PRIMARY KEY:** Where we are keeping the values as unique and not

null

4. **FOREIGN KEY:** It is used to define the foreign key for a table and establish the relationship.
5. **CHECK:** This is used to limit the values inside the column
6. **DEFAULT:** Is used to set default values within the columns.

```
SALARY INT default 25000
```

6.WHERE CLAUSE IN SQL :

The WHERE clause is used to filter records.

It is used to extract only those records that fulfil a specified condition or an extra condition or define few conditions.

Where clause is used with the select statement

SYNTAX:

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

Here, conditions will be defined by the user.

Example,

Consider a table with roll no, name and marks of the students in the table students data,

now to find out students with the marks above 20 use the command as follows:

```
SELECT * FROM studentsdata WHERE marks > 40;
```

In the table `studentsdata`, the data is as follows:

```
1|ADITI|23
2|PRIYANKA|85
3|ZUNAID|87
4|ADITI|90
5|YAMINI|78
```

Now, executing our select statement with the where clause for students with marks above 40, it will yield the result as follows:

OUTPUT:

```
2|PRIYANKA|85
3|ZUNAID|87
4|ADITI|90
5|YAMINI|78
```

For marks > greater than 80, it will give the remaining 3 records instead of 4, removing the data for Yamini as well.

```
2|PRIYANKA|85
3|ZUNAID|87
4|ADITI|90
```

7. OPERATORS:

Operators used in SQL are like the mathematical operators.

Where Clause

Using Operators in WHERE

Arithmetic Operators : +(addition) , -(subtraction), *(multiplication), /(division), %(modulus)

Comparison Operators : = (equal to), != (not equal to), > , >=, <, <=

Logical Operators : AND, OR , NOT, IN, BETWEEN, ALL, LIKE, ANY

Bitwise Operators : & (Bitwise AND), | (Bitwise OR)

Here: % gives the remainder as the result from the division

Example for AND, OR operator

Consider the table student data with roll no, name, marks, city ,

AND operator: It is used to execute both the conditions in the statement

```
SELECT * FROM studentsdata WHERE marks > 80 AND city ="Mumbai"
```

This gives the data for students with marks greater than 80 and belonging only from mumbai.

OR operator: With the usage of OR keyword, either of the conditions, if satisfied yields the result.

```
SELECT * FROM studentsdata WHERE marks > 90 OR city ="Mumbai"
```

8.LIMIT CLAUSE IN SQL :

Limit clause in SQL is used to set an upper limit on a number of rows to be returned, in other simple words, limit is used to specify the number of records we want to return. It gives and counts the record from the Top .

SYNTAX:

```
SELECT Col1,Col2 FROM table_name  
LIMIT number
```

In Table containing multiple rows, if we want to extract top 4 records put LIMIT 4

```
SELECT * FROM studentsdata LIMIT 4;
```

```
1|ADITI|23
```

```
2|PRIYANKA|85
```

```
3|ZUNAID|87
```

```
4|ADITI|90
```

For extracting top 3 records of the students where marks of the students are less than 60, The command and output will be as follows:

```
SELECT *  
FROM studentsdata  
WHERE marks<60  
LIMIT 3;
```

9.ORDER BY CLAUSE IN SQL:

The MYSQL ORDER BY Clause is used to sort the records in ascending or descending order at times depending upon the data

Syntax:

```
SELECT *  
FROM tables  
WHERE conditions  
ORDER BY expression [ ASC | DESC ];
```

1.WHERE conditions: It is optional. It specifies conditions that must be fulfilled for the records to be selected.

2.ASC: It is optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).

3.DESC: It is also optional. It sorts the result set in descending order by expression.

ORDER BY: WITHOUT USING ASC/DESC ATTRIBUTE

- If you use MySQL ORDER BY clause without specifying the ASC and DESC modifier then by default you will get the result in ascending

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
ORDER BY officer_name;
```

order.

10.GROUP BY CLAUSE:

The GROUP BY Clause is used to collect data from multiple records and group the result by one or more columns as a summary , it actually groups and work on columns that have same values .

Therefore, the GROUP BY clause is typically used with aggregate functions such as SUM(), COUNT(), AVG(), MAX(), or MIN() etc.

SYNTAX:

```
SELECT column_name(s)  
FROM table_name  
WHERE condition  
GROUP BY column_name(s)  
ORDER BY column_name(s);
```

Example: To count the number of students in the city, here we will use the COUNT() function, as the GROUPBY is used mostly with aggregate functions, The approach here is that it will group students by every city and will give the count of number of students in each city based on roll no. and will result in the column of the same.

```
Select City, Count(rollno)
FROM studentsdata
GROUP BY City
```

city	count(rollno)
Delhi	3
Mumbai	2
Pune	1

11.HAVING CLAUSE IN SQL:

The SQL having clause is used to **apply filter** on the “**result of the GROUP BY**” based on specific condition, or in other words,

HAVING is used to filter the results of groups created by GROUP BY.

- The WHERE clause places conditions on the selected columns, whereas the HAVING clause place conditions on groups created by the GROUP BY clause.

SYNTAX:

```
SELECT column name(s)
FROM table name
WHERE condition(s)
GROUP BY columns name(s)
Having condition(s)
```

EXAMPLE:

Taking a table as follows:

Customer_ID	Amount	Mode	Payment_date
1	60	Cash	2020-09-14
2	30	Credit Card	2020-04-27
3	90	Credit Card	2020-02-12
4	40	Debit Card	2021-11-20
5	40	Mobile Payment	2021-08-25
6	10	Cash	2021-05-28

Here, please note, "Where" Clause is particularly used with SELECT statement,

HAVING is executed always universally after the GROUP BY condition.

```

SELECT mode, COUNT (amount) AS total
FROM Payment
GROUP BY mode
HAVING COUNT (amount) >= 3
ORDER BY total DESC

```

12. GENERAL ORDER OF THE CLAUSES

General Order

```
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;
```



13. JOINS IN SQL:

Joins in SQL are used to combine two or more rows in a table based on a column related between them to extract data from those tables.

In very simple words, they are used to combine the common information from 2 or more multiple tables

To join 2 or more tables, we need to have a common column or in simple words, common data among those tables or related columns.

EXAMPLE:

Let's take table A and table B, where the table are as follows:

TABLE A:

Emp_ID	Name
109	Elvin
23	Riya

TABLE B:

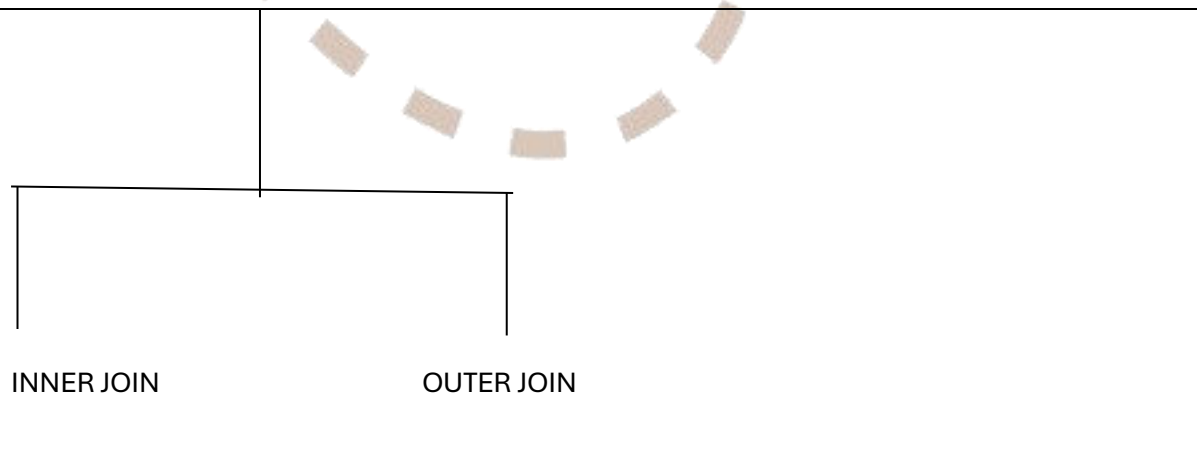
Emp_ID	Salary
109	3000
231	5000
454	11000

Now, here in both the tables A and B respectively, we can see that there are 2 columns

Emp_ID, which is a common column

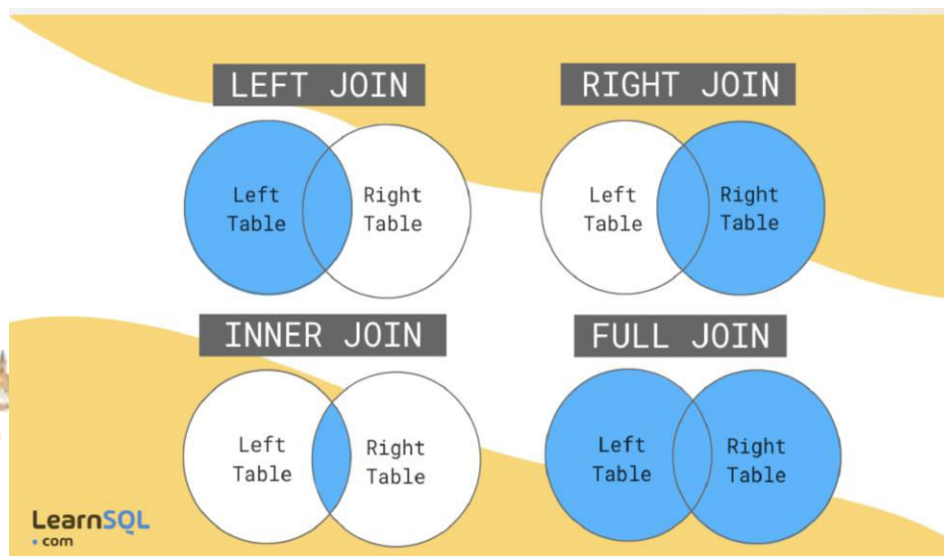
Name and salary are different columns.

13.1 TYPES OF JOINS:



Left Join	Right Join Full Join
Inner Join, the right join, left join and full join fall under the category of OUTER JOINS.	

The following illustration shows the venn diagram representation of the joins:



13.1.2 INNER JOIN:

Inner join is used when we want the common data from both the tables, in other words it returns the record that have matching values in both the tables,

SYNTAX:

```
SELECT column(s)
FROM tableA
INNER JOIN table B
```


ON tableA.col_name=tableB.col_name;

This means we are doing joins on table A and table B, based on the columns from table A and table B where the values for both the columns are the same.

We can even use table B first here and then table A in the syntax because we are fetching only the common data or overlapping values

EXAMPLE:

Consider TABLE A Customers:

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Now, the common column = customer_id in both the tables,

The query to get first name, age and item corresponding to common customers or records from both the tables will be as follows:

```
SELECT *
FROM Customers
INNER JOIN Orders
on Customers.customer_id=Orders.Customer_id;
```

Result : Now see, we only got the common records, the customer_id

customer_id	first_name	age	item
4	John	25	Keyboard
4	John	25	Mouse
3	David	22	Monitor
1	John	31	Keyboard
2	Robert	22	Mousepad

PLEASE NOTE: The result shows the records here for customer id 1, 2, 3, 4 but not for 5 as it is not there in table B which is customers, as in orders table there are 4 id 1,2,3,4 and in customer

13.1.2LEFT JOIN:

A LEFT JOIN returns all rows from the left table and common data from the right table(tableB or table2).

If no matching rows have been found in the right table, the query will return NULL values for those columns.

Considering 2 tables A and B , where A table is left table and table B

SYNTAX :

```
SELECT column(s)
FROM table A
```

LEFT JOIN table B

ON tableA.column_name=tableB.column_names

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

SELECT *

FROM Customers

LEFT JOIN Orders

on Customers.customer_id=Orders.Customer_id;

RESULT :

customer_id	first_name	last_name	age	country	order_id	item
1	John	Doe	31	USA	4	Keyboard
2	Robert	Luna	22	USA	5	Mousepad
3	David	Robinson	22	UK	3	Monitor
4	John	Reinhardt	25	UK	1	Keyboard
4	John	Reinhardt	25	UK	2	Mouse
5	Betty	Doe	28	UAE		

So, here all the records which are there in the left table and the overlapping or common record in the right table will be fetched in the output , so as you can see customers 1 to 5 are fetched from both the tables, the common customer_id = 1,2,3,4,4, but as 5 is present in left table it will give 5 too .

13.1.3 RIGHT JOIN:

A RIGHT JOIN returns all rows from the right table (Table B) and the matching rows from the left table (Table A). If there are no matching rows in the left table, it will return NULL values for those columns.

```
SELECT column(s)
```

```
FROM tableA
```

```
RIGHT JOIN tableB
```

```
ON tableA.column_name = tableB.column_name;
```

The result includes all records from the right table, regardless of whether there is a match in the left table.

If there is no match, the corresponding columns from the left table will contain NULL.

EXAMPLE:

There are 2 tables Orders and employees, now we will apply right join to get the column details.

Here we are selecting columns such as order id, last name and first name for those particular id. We have other columns such as ShipperId and BirthDate, Photo.

But we only want a table with EmployeeID and First Name and Last Name,

As First and Last name falls under second table or table B we will apply the right join.

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

The query to fetch the employee id along with first and last name will be as follows:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Number of Records: 197

OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret
10251	Leverling	Janet
10252	Peacock	Margaret
10253	Leverling	Janet
10254	Buchanan	Steven
10255	Dodsworth	Anne
10256	Leverling	Janet
10257	Peacock	Margaret
10258	Davolio	Nancy
10259	Peacock	Margaret
10260	Peacock	Margaret
10261	Peacock	Margaret
10262	Callahan	Laura
10263	Dodsworth	Anne
10264	Suyama	Michael

13.1.4. FULL OUTER JOIN OR FULL JOIN:

Let's take MySQL here, which doesn't support key word, FULL JOIN, hence we use the keyword UNION.

In very simple terms: In MySQL , we perform the full outer join as follows:

FULL JOIN = LEFT JOIN + UNION + RIGHT JOIN , whereas for other Databases the Full Join works.

This gives us the data of both the tables and union by default gives us the unique data.

Now , let;s look at the syntax with example for full outer join :

EXAMPLE : Take 2 tables customers and Order :

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

CUSTOMERS

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

ORDER :

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

RESULT :

Number of Records: 197

OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret
10251	Leverling	Janet
10252	Peacock	Margaret
10253	Leverling	Janet
10254	Buchanan	Steven

10268	Callahan	Laura
10269	Buchanan	Steven
10270	Davolio	Nancy
10271	Suyama	Michael
10272	Suyama	Michael
10273	Leverling	Janet
10274	Suyama	Michael
10275	Davolio	Nancy
10276	Callahan	Laura
10277	Fuller	Andrew

13.2 EXCLUSIVE JOINS:

LEFT EXCLUSIVE JOIN:

FOR left and right exclusive joins, we use the join key words along with "WHERE" Condition along with "IS NULL".

We use exclusive joins only when we want the values of values of table belonging completely to table A but do not overlap with table B.

It should exclusively belong to Table A or the left table.

In left exclusive joins, the null values of B are present in table A, and not null values of B are present in table A, hence we will only get the values of table A, not even the overlapping values of A and B.

Although there is no specific So the syntax, goes like this:

Example: There are 2 tables A = student and table B = course. Now the LEFT EXCLUSIVE JOIN looks as follows:

```
SELECT *  
FROM STUDENT  
LEFT JOIN COURSE  
ON student.id = course.id  
WHERE course.id IS NULL
```

RIGHT EXCLUSIVE JOIN:

Here also, we will use the "WHERE" condition along with "IS NULL". So here, the right exclusive join will give the values of "Only right table or table B" but neither the overlapping values of table A and B, nor the values of table A.

13.5 SELF JOIN:

A **self join** in SQL is a join where a table is joined with itself. This is useful when you want to compare rows within the same table. You can think of it as treating the same table as two different tables for the purpose of the join.

Self joins are often used to find relationships within the same dataset, such as hierarchical relationships or comparing rows based on certain criteria.

SYNTAX:

```
SELECT A.column1, A.column2, B.column1, B.column2
```

```
FROM table_name A
```

```
JOIN table_name B
```

```
ON A.common_column = B.common_column;
```

Consider a Table: Employees

EmployeeID	FirstName	LastName	ManagerID
------------	-----------	----------	-----------

1	Alice	Smith	NULL
---	-------	-------	------

2	Bob	Johnson	1
---	-----	---------	---

3	Charlie	Brown	1
---	---------	-------	---

4	David	Wilson	2
---	-------	--------	---

5	Eva	Davis	2
---	-----	-------	---

In this example, the ManagerID column indicates the manager for each employee. An employee can manage other employees, so we can use a self join to find employees and their managers.

Query

To get a list of employees along with their managers' names, you would write:

```
SELECT
```

```
A.FirstName AS EmployeeFirstName,
```

```
A.LastName AS EmployeeLastName,
```

```
B.FirstName AS ManagerFirstName,  
B.LastName AS ManagerLastName  
FROM Employees A  
JOIN Employees B ON A.ManagerID = B.EmployeeID;
```

Result

The result of this query would look like this:

EmployeeFirstNa me	EmployeeLastNa me	ManagerFirstNa me	ManagerLastNa me
Bob	Johnson	Alice	Smith
Charlie	Brown	Alice	Smith
David	Wilson	Bob	Johnson
Eva	Davis	Bob	Johnson

Explanation of the Result

- Each employee is listed with their corresponding manager's first and last names.
- For instance, Bob and Charlie both report to Alice, while David and Eva report to Bob.

14. UNION:

Unions in SQL are used to join two or more select statements.

Using this statement gives all the unique values or records

Every select statement using union should have same number of columns

Columns have same data type,

The order of the columns in every select statement should be same.

```
SELECT column_name(s) FROM table1
```

```
UNION
```

```
SELECT column_name(s) FROM table2;
```

UNION ALL:

Unions in SQL are used to join two or more select statements and gives even the duplicate values.

Both Union and Union All can be applied on same or different tables.

15. VIEWS & STORED PROCEDURE:

Stored procedures is a set of SQL instructions stored as units that can be used over and over again for reusability and code optimization.

A stored procedure is a precompiled set of SQL statements that can perform operations like data retrieval, updates, or business logic.

It can accept parameters, contain control-of-flow logic (like IF statements), and perform multiple operations in one call.

VIEW :

A view is essentially a virtual table created by a query that pulls data from one or more tables.

It's used to simplify complex queries, provide a specific data representation, or encapsulate complex joins and calculations.

Views can be used in SELECT statements just like a regular table.

There are various advantages of having stored procedures

1.Security:

The database administrator can give required permission to required people to leverage or access the stored procedure.

2.Reusability:

Once the stored procedure is written it is not required to write the same code again and again, this eliminates needless rewrites of the same code, decreases code inconsistency, and allows the access and execution of code by any user or application possessing the necessary permissions.

3.Easier maintenance

In Simple language, whenever there are any changes in the application, only

the procedures are updated, and we don't need to change the application tier,

Thus it saves time,

SYNTAX FOR STORED PROCEDURES:

```
CREATE PROCEDURE procedure name
AS
BEGIN
.
.
.
.
END
```

16.FUNCTIONS:

Functions in SQL are pre-built units or sub-programs or are special commands that perform specific tasks on data such as calculations and return results.

They help us manipulate and analyse information in a database.

We can use pre-defined functions or can define our own functions, we need to call the functions to use them or leverage their functionality.

They accept input parameters, perform actions and then give the results.

Functions can be classified as:

1. System defined functions
2. User defined functions

System defined functions are built-in functions where the job or task is pre-defined to be executed on data like Count(). Max() , Min() where we can easily make out Count() stands for counting , Max() stands for maximum ,Min() stands for Minimum, there are various types of string functions which we will see ahead ,

CALLING A FUNCTION:

Now, what do we mean by calling a function , to use that function which is already built in or even the ones we as users define, we need to call them , in SQL we use certain syntax for the same

• 16.1. Aggregate Function

These functions perform calculations on a set of values and return a single value.

These functions are often used with **"GroupBY"** and **"Select" statement**

- COUNT(): Counts the number of rows.
- SUM(): Adds up all the values in a column.
- AVG(): Calculates the average of a set of values.
- MAX() : Finds the largest value.
- MIN() : Finds the smallest value.
- ROUND(): Rounds a number to a specified number of decimal places.

16.2. Scalar Functions

These functions operate on a single value and return a single value.

- UPPER(): Converts text to uppercase.
- LOWER() : Converts text to lowercase.
- LENGTH(): Returns the number of characters in a string.

16.3. String Functions

Let's have a look at some of the MYSQL string functions These manipulate string values (text) or in simple words are used to work on strings.

- `CONCAT()`: Combines two or more strings into one.
- `SUBSTRING()`: Extracts a part of a string.
- `TRIM()`: Removes extra spaces from the beginning and end of a string.
- `FORMAT()`: Is used to set the format of the field.
- `REPLACE()`: Replaces all occurrences of a string within a substring, with a new string.

Please note : In data analysis, these functions all together mentioned in `Scalar()` and `string()` will be widely used .



16.4. Date Functions

These deal with date and time values.

- `NOW()`: Gets the current date and time.
- `DATEADD()`: Adds a specified time interval to a date.
- `DATEDIFF()`: Calculates the difference between two dates.

16.5. Conversion Functions

These convert data from one type to another.

- CAST(): Changes a value from one data type to another.
- CONVERT(): Similar to CAST, but with more options for formatting.

16.6 ISNULL () :

This is one of the most important functions in SQL, it is used to check whether an expression is NULL or not.

The return type is 1, if the value is null otherwise it's 0.

SYNTAX:

```
SELECT column1, column 2, column 3,  
FROM Table Name  
WHERE column IS NULL;
```

NOT NULL()

Alternatively,

We can find Records which are not null using the IS NOT NULL() as a Constraint or as a function.

Example : We have a Table Persons with columns ::

We have a Table Persons with columns ::

Persons

ID	NAME	BIRTH_DATE
1	ZARON	1995-02-10
2		1995-11-03
3	RANJVIJAY	1990- 10 - 23
4	SHARIQ	

Now, supposedly , we want to find the date of null values from date of birth, we will go ahead as following :

SELECT * from Persons

WHERE BIRTH_DATE IS NULL:

It will fetch the record for "ID = 4, SHARIQ, Where the date of birth is missing as column BIRTH_DATE indicates date of birth.

17.SUB- QUERY IN SQL:

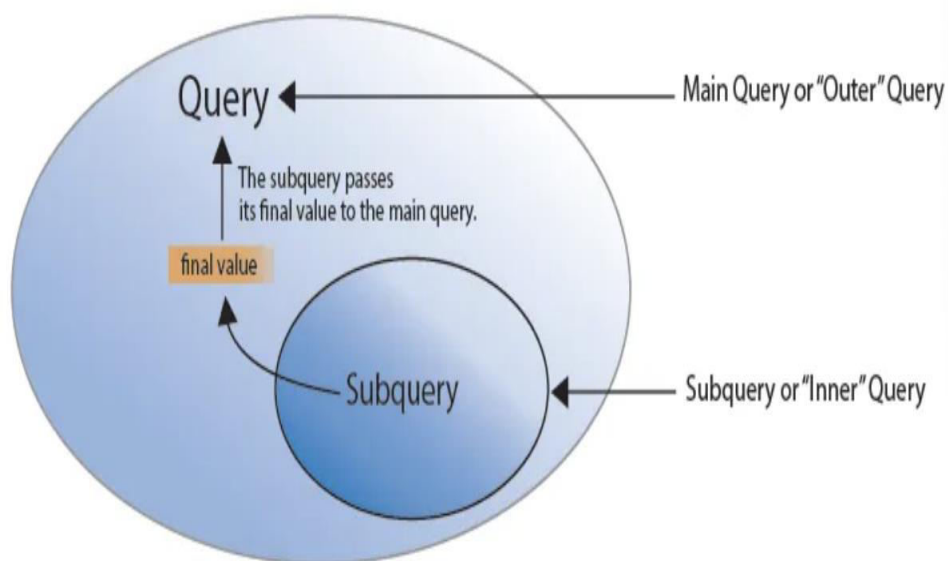
What is a Subquery?

A **subquery** is a small query inside a larger query. It's like asking a question within a question! You use it to get some information that helps answer the main question. It is also known as inner query or nested query as we have query defined inside or within the query /

Why Use a Subquery?

You might use a subquery when you need to find something that helps you filter or get more data in your main query.

A diagrammatic representation of what Query and Sub-query would look like.



```
SELECT column_name
```

```
FROM table_name
```

```
WHERE column_name IN (SELECT column_name FROM another_table  
WHERE condition);
```

Types of Subqueries

1.Single-Row Subquery

- Returns a single value (one row and one column).
- Commonly used with comparison operators (=, <, >, etc.).

Example:

```
SELECT StudentName  
FROM Students  
WHERE Score = (SELECT MAX(Score) FROM Students);
```

Explanation: This finds the student with the highest score.

2.Multi-Row Subquery

- Returns multiple rows.
- Used with the IN, ANY, or ALL operators.

Example:

```
SELECT StudentName  
FROM Students  
WHERE StudentID IN (SELECT StudentID FROM PassedStudents);
```

Explanation: This finds all students whose IDs are listed in the PassedStudents table.

3.Correlated Subquery

Refers to a column from the outer query.

Executes once for each row processed by the outer query.

Example:

```
SELECT StudentName  
FROM Students AS s  
WHERE Score > (SELECT AVG(Score) FROM Students AS s2 WHERE  
s2.StudentID = s.StudentID);
```

Explanation: This finds students who scored above the average score. The inner query calculates the average for each student in the outer query.

4.Exists Subquery

Checks for the existence of rows returned by the subquery.

Often used with the EXISTS keyword.

```
SELECT StudentName  
FROM Students s  
WHERE EXISTS (SELECT 1 FROM PassedStudents p WHERE p.StudentID  
= s.StudentID);
```

Explanation: This finds students who have passed by checking if their ID exists in the PassedStudents table.

Inline Views (or Derived Tables)

Treats a subquery as a temporary table.


Allows you to use the results as if they were a table.

Example:

```
SELECT avgScore.StudentName, avgScore.AvgScore
FROM (SELECT StudentName, AVG(Score) AS AvgScore FROM Students
GROUP BY StudentName) AS avgScore
WHERE avgScore.AvgScore > 80;
```

Explanation: This finds students with an average score above 80, using a subquery as a temporary table.

Key Points About Subqueries

 Nesting: You can nest subqueries within each other. For example, a subquery can be part of another subquery.

Performance: Subqueries can be less efficient than joins for large datasets. It's essential to consider performance when designing your queries.

Readability: Sometimes, using joins instead of subqueries can make your SQL statements clearer and easier to understand.

Scalability: Complex subqueries might become harder to maintain. In such cases, breaking them down into smaller parts or using views can help.

18.COMMON TABLE EXPRESSIONS/WITH CLAUSE

.Common Table Expressions (CTEs) in SQL are like temporary tables that are created when we run a query and use for the query calculations .

2. They are better than sub-queries, as instead of sub-query one can use WITH Clause, they query looks more clean meaning more optimized easy to read and understand.

3. In some scenarios the performance is also good .

Key Points About CTEs:

CTE's aren't stored anywhere , once the query is done , they are deleted.

WITH CLAUSE is used to write a CTE expression.

SYNTAX / EXAMPLE

```
WITH CTE_Name AS (  
  QUERY .....)  
SELECT *  
FROM CTE_Name;
```

Example:

Suppose we have a table of sales:

Sales_ID	Product	Amount
1	Apples	100
2	Bananas	200
3	Cherries	150

```
WITH TotalSales AS (  
    SELECT SUM(Amount) AS Total  
    FROM Sales  
)  
SELECT Total  
FROM TotalSales;
```

Now, that we need to find total sales for all products Using a CTE, it will be as follows:

Breakdown of the Example

1. CTE Definition: `WITH TotalSales AS (...)` creates a temporary table named `TotalSales` that calculates the total sales amount.
2. Main Query: The `SELECT` statement retrieves the total from the `TotalSales` CTE.

19. WINDOWS FUNCTIONS :



Window Functions

Aggregate	Value	Ranking
<ul style="list-style-type: none">• AVG()• MAX()• MIN()• SUM()• COUNT()	<ul style="list-style-type: none">• ROW_NUMBER()• RANK()• DENSE_RANK()• PERCENT_RANK()• NTILE()	<ul style="list-style-type: none">• LAG()• LEAD()• FIRST_VALUE()• LAST_VALUE()• NTH_VALUE()

A window function makes a calculation across multiple rows that are related to the current row.

Or in extremely simple words , Window functions in SQL are special types of functions that allow you to perform calculations across a set of rows related to the current row without grouping the rows into a single output.

They are useful for analysing data while still keeping the individual rows intact.

For example, a window function allows you to calculate:

- Running totals (i.e. sum values from all the rows before the current row)
- 7-day moving averages (i.e. average values from 7 rows before the current row)
- Rankings
- Windows can be defined in the SELECT section of the query.
- Windows function applies aggregate , ranking , analytic functions over a particular window.
- NOTE : “**OVER**” Clause is used with windows function to define that window.

SYNTAX :

```
SELECT Column_name,  
       window_function() OVER(  
         PARTITION BY partition_expression  
         ORDER BY order_expression  
         Row or range clause  
       ) AS window_column_alias  
FROM table_name
```

Window_function = aggregate function , ranking function,
analytics function

Syntax:

OVER() contains

PARTITION BY : On which columns you want to apply partition.

ORDER BY : The columns according to which we want to sort the data.

19.1 RANKING FUNCTIONS:

Ranking Functions help determine the position of rows in a dataset (like RANK(), DENSE_RANK(), ROW_NUMBER(), PERCENT RANK())

Let's look at a very simple data and infer :

new_id	ROW_NUMBER	RANK	DENSE_RANK	PERCENT_RANK
100	1	1	1	0
200	2	2	2	0.166
200	3	2	2	0.166
300	4	4	3	0.5
500	5	5	4	0.666
500	6	5	4	0.666
700	7	7	5	1

Try to make out from the table how ranks are given .

1. RANK():

Assigns a unique rank to each row within a partition. If there are ties or duplicate values, it assigns the same rank but skips the next rank(s).

In the context of ranking functions in SQL, "ties" refer to situations where two or more rows have the same value in the column or columns used for ranking. When this happens, the ranking functions need to decide how to assign ranks to these rows.

Example: If two rows are tied for rank 1, the next rank assigned will be 3.

So here in table as 200 is getting repeated and 200 on row 2 is correctly and naturally marked as 2, but for rank function column in 200 mentioned in row 3, we have ranked marked as 2 itself, instead naturally and universally the counting should have been marked as 3 and 3 is skipped and in the next row as well 3 is skipped , instead it is counted as 4.

SYNTAX:

```
RANK() OVER (PARTITION BY partition_column ORDER BY  
order_column [ASC|DESC])
```

- **PARTITION BY:** Optional. Divides the result set into partitions to which the rank is applied.
- **ORDER BY:** Required. Specifies the order in which the ranking is assigned.

2. DENSE_RANK():

It is similar to RANK(), but it doesn't skip ranks. If there are ties, the next rank will be the immediate next integer.

Example: If two rows are tied for rank 1, the next rank assigned will be 2.

Here for 200 in row 2 it is marked as 2, and for that of row 3 is marked as 2, but for the next one the rank assigned is 3. It did not skip to mention rank 3.

```
DENSE_RANK() OVER (PARTITION BY partition_column ORDER  
BY order_column [ASC|DESC])
```

SYNTAX:

3. ROW_NUMBER():

This is pretty simple, Assigns a unique sequential integer to each row, starting at 1. It does not consider ties. Example: Every row will have a unique number, even if some values are the same.

SYNTAX:

```
DENSE_RANK() OVER (PARTITION BY partition_column ORDER  
BY order_column [ASC|DESC])
```

4. PERCENT RANK():

The PERCENT_RANK() function in SQL is a ranking function that calculates the relative rank of a row within a partition as a percentage of the total number of rows in that partition. It is often used to understand how a particular value compares to others in a dataset.

PERCENT_RANK() computes the relative rank of a row in a result set as follows:

$$\text{PERCENT_RANK} = \frac{(\text{Rank} - 1)}{(\text{Total Rows} - 1)}$$

As in the last column percentage rank has been calculated

SYNTAX:

```
PERCENT_RANK() OVER (PARTITION BY partition_column  
ORDER BY order_column [ASC|DESC])
```

Here is an example of using all the functions,

EXAMPLE :

```
SELECT
```

```
Student,  
Score,  
RANK() OVER (ORDER BY Score DESC) AS Rank,  
DENSE_RANK() OVER (ORDER BY Score DESC) AS DenseRank,  
ROW_NUMBER() OVER (ORDER BY Score DESC) AS RowNum,  
PERCENT_RANK() OVER (ORDER BY Score DESC) AS PercentRank  
FROM Students;
```

19.2 Analytic Functions :

The LAG(), LEAD(), FIRST_VALUE(), and LAST_VALUE() functions are analytic functions used to access data from previous or subsequent rows within a result set. These functions are useful for comparing values across rows and analyzing trends over time.

new_id	FIRST_VALUE	LAST_VALUE	LEAD	LAG
100	100	100	200	null
200	100	200	200	100
200	100	200	300	200
300	100	300	500	200
500	100	500	500	300
500	100	500	700	500
700	100	700	null	500

LAG () and LEAD() are positional functions. These are very useful in creating reports, because they can refer to data from rows above or below the current row

1.LAG ()

Definition: LAG() function allows access to a value stored in a different row above the current row.

If we look at the first row here in the column new_id , we see 100 ,and for LAG() function to identify and compare , we have nothing above 100, so the value here is null.

Whereas , in the second row the value is 200 , hence the result of comparison between 100 and 200, we get 100, the comparison here is done from previous values or the data in the row above.

```
LAG(column_name, offset, default_value) OVER (PARTITION BY  
partition_column ORDER BY order_column)
```

LAG() takes three arguments:

Expression : the name of the column or an expression from which the value is obtained,

column_name: The column you want to retrieve from the previous row.

offset: The number of rows back from the current row (default is 1).

default_value: A value to return if the offset goes beyond the available rows (default is NULL).

if you do not specify the offset in the LAG() or LEAD() functions, it defaults to 1. This means that it will refer to the immediately

preceding row for LAG() and the immediately following row for LEAD().

2. LEAD ()

Definition: The LEAD() function allows you to access data from the rows below.

column_name: The column you want to retrieve from the previous row.

offset: The number of rows back from the current row (default is 1).

default_value: A value to return if the offset goes beyond the available rows (default is NULL).

```
LEAD(column_name, offset, default_value) OVER (PARTITION BY  
partition_column ORDER BY order_column)
```



IT MOVES FORWARD INSTEAD OF BACKWARDS.

PRO TIP :

Just like LAG(), LEAD() is a window function and requires an OVER clause. And as with LAG(), LEAD() must be accompanied by an ORDER BY in the OVER clause.

3. FIRST_VALUE()

Definition: The FIRST_VALUE() function retrieves the first value in a specified column within the window frame of the current row.

OR

Retrieves the first value in a specified order.

```
FIRST_VALUE(column_name) OVER (PARTITION BY
```

```
partition_column ORDER BY order_column)
```

4. LAST_VALUE()

Definition: The LAST_VALUE() function retrieves the last value in a specified column within the window frame of the current row.

Retrieves the last value in a specified order, requiring a frame definition.

```
LAST_VALUE(column_name) OVER (PARTITION BY  
partition_column ORDER BY order_column ROWS BETWEEN  
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
```



20. CASE WHEN:

The CASE WHEN expression in SQL is a powerful tool for implementing conditional logic directly within your queries. It allows you to evaluate conditions and return specific values based on whether those conditions are met.

It's like an If-Else Statement which are known as conditional statements.

Syntax :

```
CASE  
  
    WHEN condition1 THEN result1
```

```
WHEN condition2 THEN result2  
  
...  
  
ELSE default_result  
  
END
```

The explanation of the syntax is as follows :

WHEN: Specifies the condition to evaluate.

THEN: Specifies the result to return if the corresponding WHEN condition is true.

ELSE: (Optional) Specifies the default result if none of the conditions are met.

END: Marks the end of the CASE expression.

Conditional Logic: Enables complex decision-making directly within SQL queries.

Flexibility: Can be used in SELECT, ORDER BY, WHERE, and HAVING clauses.

Readability: Makes queries easier to understand by explicitly stating conditions and corresponding outcomes.

EXAMPLE :

Suppose we have a table called Employees with columns for EmployeeID, Name, and Salary and you want to categorize employees based on their salary:

- "High" for salaries above \$80,000
- "Medium" for salaries between \$50,000 and \$80,000
- "Low" for salaries below \$50,000

Here's how you can use the CASE WHEN expression in a query:

```
SELECT
```

```
EmployeeID,  
Name,  
Salary,  
CASE  
    WHEN Salary > 80000 THEN 'High'  
    WHEN Salary BETWEEN 50000 AND 80000 THEN 'Medium'  
    ELSE 'Low'  
END AS SalaryCategory  
FROM Employees;
```

