** SQL is a standard language for storing, manipulating and retrieving data in databases.

→ SQL is used to perform operations on the records stored in the database.

→ SQL is not a database system, but it is a query language.

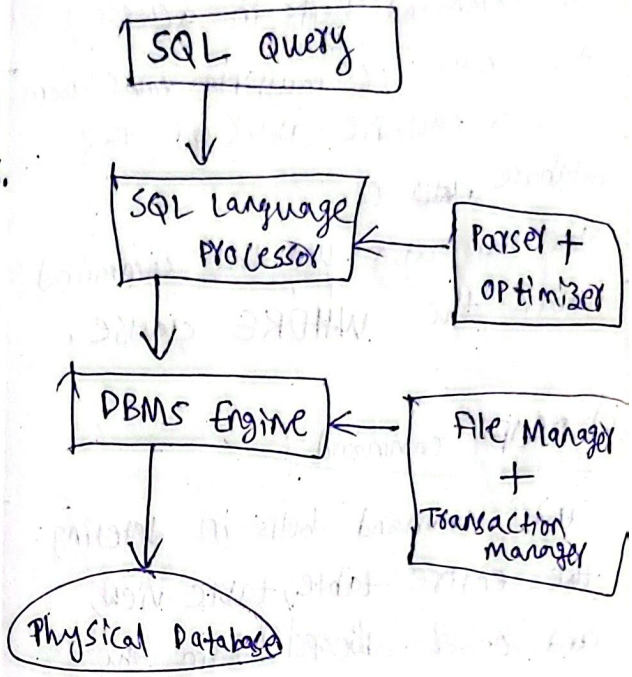
→ Suppose you want to perform the queries of SQL language on the stored data in database, you are required to install database management systems, for example MySQL, MongoDB etc--

→ structured data (data which is ~~stored~~ stored in the form of tables)

SQL contains the following four components in its process:

(i) Query Dispatcher
(ii) Optimization Engines
(iii) Classic Query Engine.
(iv) SQL query Engine, etc..

SQL architecture:



(1) CREATE command:

→ This command helps in creating the new database, new table, table view and other objects of database.

(2) UPDATE command:

→ This command helps in updating or changing the stored data in the database.

(3) DELETE command:

→ This command helps in removing or erasing the saved records from the database tables.

→ It erases single or multiple tuples from the tables of the databases

(4) SELECT command :-

→ This command helps in accessing the single or multiple rows from one or multiple tables of the database.

→ We can also use this command with the WHERE clause.

(5) DROP command :-

→ This command helps in deleting the entire table, table view, and other objects from the database.

6) INSERT command :

→ This command helps in inserting the data or records into the database tables.

→ We can easily insert the records in single as well as multiple rows of the table.

SQL

→ It is a relational database management system.

→ Databases are vertically scalable

→ Databases are in the form of tables.

→ Schema of SQL databases is predefined, fixed and static.

→ It follows ACID model
(Atomicity, consistency, isolation, durability)

→ not best for storing hierarchial data.

of SQL database systems.

No - SQL :

→ It is a non-relational or distributed database management system.

→ Databases are horizontally scalable

→ Databases are in the form of key-value, documents and graphs.

→ Schema of No-SQL databases is a dynamic schema for unstructured data.

→ It follows the BASE model.
(Basically available, Soft state, Eventual consistency)

→ Redis, Cassandra, MongoDB, BigTable are examples of No-SQL database systems.

Advantages of SQL

→ No Programming needed

→ High speed query processing

→ Standardized Language

→ Portability

→ Interactive language.

Disadvantages of SQL

→ Cost

→ Interface is complex

→ Partial Database control.

SQL Syntax

→ The syntax of SQL is a unique set of rules and guidelines.

→ syntax of SQL is not a case-sensitive.

→ ANSI → American National Standards Institution.

SQL Statement:

→ SQL statement tells the database what operation you want to perform on the structured data and what information you would like to access from database.

Example of SQL statements

```
SELECT "column_name" FROM "table_name";
```

→ SQL statement starts with any of the SQL keywords and ends with the semicolon(;).

SQL Commands and statements

(1) SELECT statement

→ This SQL statement reads the data from the SQL database and shows it as the output to the database user.

Syntax:

```
SELECT column_name1, column_name2, ...
       ....., column_nameN
[FROM table_name]
[WHERE condition]
[ORDER BY order_column_name1 [ASC|DESC],
       ....];
```

Example:

```
SELECT Emp_ID, First_Name, Last_Name,
       Salary, City
FROM Employee_details
WHERE Salary = 100000
ORDER BY Last_Name
```

→ This example shows the Emp_ID, First_Name, Last_Name, -, ... of those employees from the Employee_details table whose Salary is 100000. The o/p shows all the specified details according to the ascending alphabetical order of Last_Name

(2) UPDATE Statement

→ This SQL statement changes or modifies the stored data in the SQL database.

Syntax:

```
UPDATE table_name
SET column_name1 = new_value_1, column_name2 = new_value_2, ..., column_nameN = new_value_N
[WHERE CONDITION];
```

Example:-

```
UPDATE Employee_details
SET Salary = 100000
WHERE EMP_ID = 10;
```

3) DELETE statement

→ This SQL statement deletes the stored data from the SQL database.

Syntax:

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

Example:

```
DELETE FROM Employee_details
WHERE first_Name = 'Sumit';
```

4) CREATE TABLE statement

→ This SQL statement creates the new table in the SQL database.

Syntax

```
CREATE TABLE table_name
(
column_name1 datatype[column1 constraints(s)],
|| 2 || [ || 2 || (s)],
---,
---,
column_nameN data_type[columnN constraints
PRIMARY KEY (one or more col)
);
```

Example:

```
CREATE TABLE Employee_details(
Emp_Id NUMBER(4) NOT NULL,
First_name VARCHAR(30),
Last_name VARCHAR(30),
Salary Money,
City VARCHAR(30),
PRIMARY KEY (Emp_Id)
);
```

(5) ALTER TABLE statement:

→ This SQL statement adds, deletes, modifies the columns of the table in SQL database.

Syntax:-

```
ALTER TABLE table_name ADD column_name datatype [(size)];
```

→ The above SQL alter statement adds the column with its datatype in the existing database table.

ALTER TABLE table-name MODIFY column_name column_datatype [(size)];

→ The above 'SQL alter statement' renames the old column name to the new column name of the existing database table.

ALTER TABLE table_name DROP COLUMN column_name

→ The above SQL alter statement deletes the column of the existing database table.

Example:

ALTER TABLE Employee_details ADD designation VARCHAR(18);

→ This example adds the new field whose name is Designation with size 18 in the Employee_details table of the SQL database.

(6) DROP TABLE Statement:-

→ This SQL statement deletes or removes the table and the structure, views, permissions, and triggers associated with that table.

Syntax:

DROP TABLE [IF EXISTS] table_name1, table_name2, ..., table_nameN;

→ The above syntax of the drop statement deletes specified tables completely if they exist in the database.

Example:-

DROP TABLE Employee_details;

→ This example drops the Employee_details table if it exists in the SQL database. This removes the complete information if available in the table.

(7) CREATE DATABASE Statement

→ This SQL statement creates the new database in the database management system.

Syntax:

```
CREATE DATABASE database_name;
```

Example:

```
CREATE DATABASE Company;
```

→ The above example creates the company database in the system.

(8) DROP DATABASE statement:-

→ This SQL statement deletes the existing database with all the data tables and views from the database management system.

Syntax:-

```
DROP DATABASE database_name;
```

Example:-

```
DROP DATABASE Company;
```

→ The above example deletes the Company database from the system.

(9) INSERT INTO statement:-

→ This SQL statement inserts the data or records in the existing table of the SQL database. This statement can easily insert single and multiple records in a single query statement.

Syntax of inserting a single record

```
INSERT INTO table_name
(
column_name1,
column_name2, ---,
column_nameN
)
VALUES
(value_1,
value_2, ---,
value_N
);
```

Example of inserting a single record:

```
INSERT INTO Employee_details
(
Emp_ID,
First_Name,
Last_name,
Salary,
City,
)
VALUES
(101,
Akhil,
Sharma,
40000,
Bangalore
);
```

Syntax of inserting a multiple records in a single query:

```
INSERT INTO table_name
(column_name1, column_name2, ---, column_nameN)
VALUES (value_1, value_2, --, value_N),
       (value1, '', ---, '') ,
       - - - - - - - ;
```

Example:

```
INSERT INTO Employee_details
(Emp_ID, First_name, Last_name, Salary, City)
VALUES (101, Amit, Gupta, 50000, mumbai),
       (101, John, Agg~, '', Calcw),
       (101, sidhu, Moose, 55K, mumbai);
```

(10) TRUNCATE TABLE statement

→ This SQL statement deletes all the stored records from the table of the SQL database.

Syntax

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Employee_details;
```

→ This example deletes the record of all employees from the Employee_details table of the database.

(11) DESCRIBE statement

→ This SQL statement tells something about the specified table or view in the query.

Syntax:

```
DESCRIBE table_name|view_name;
```

Example:

```
DESCRIBE Employee_details;
```

→ This example explains the structure and other details about the Employee_details table.

(12) DISTINCT Clause:

→ This SQL statement shows the distinct values from the specified columns of the database table. This statement is used with the SELECT keyword.

Syntax

```
SELECT DISTINCT column_name1, column_
                name2, - - -
FROM table_name;
```

Example:

```
SELECT DISTINCT City, Salary
FROM Employee_details;
```

(13) COMMIT statement

→ This SQL statement saves the changes permanently, which are done in the transaction of the SQL database.

Syntax :-

```
COMMIT
```

Example

```
DELETE FROM Employee_details
WHERE salary = 30000;
COMMIT;
```

→ This example deletes the records of those employees whose salary is 30000 and then saves the changes permanently in the database.

(14) ROLLBACK statement:

→ This SQL statement undo the transactions and operations which are not yet saved to the SQL database.

Syntax :-

```
ROLLBACK
```

Example :

```
DELETE FROM Employee_details
WHERE City = Mumbai;
ROLLBACK;
```

→ This example deletes the records of those employees whose city is Mumbai and then undo the changes in the database.

(15) CREATE INDEX statement:

→ This SQL statement creates the new index in the SQL database table.

Syntax

```
CREATE INDEX Index_name
ON table_name (column_name1, column_name2, ----, column_nameN);
```

Example:-

```
CREATE INDEX idx_First_Name
ON employee_details (First_Name);
```

→ This example creates an index idx_First_Name on the First_Name column of the Employee_details table.

(16) DROP INDEX statement:

→ This statement deletes the existing index of the SQL database table.

Syntax :

```
DROP INDEX index_name;
```

Example :

```
DROP INDEX idx_First_Name;
```

→ This example deletes the index SQL

(17) USE statement:

→ This SQL statement selects the existing SQL database.

→ Before performing the operations on the database table, you have to select the database from the multiple existing databases.

Syntax

```
USE database_name;
```

Example:

```
USE Company;
```

→ This example uses the Company database

SQL Data Types:

→ Data types are used to represent the nature of the data that can be stored in the database table.

Data types mainly classified into three categories for every database.

- string Data types
- Numeric " "
- Date & time " "

MySQL String Data Types:

CHAR(size)

→ It is used to specify a fixed length string that can contain numbers, letters and special characters.

→ Its size can be 0 to 255 characters.

→ Default is 1.

VARCHAR(size)

→ It is " " " a variable length " " " "

→ It size can be from 0 to 65535 characters.

BINARY(size)

→ It is equal to CHAR() but stores binary byte strings. ~~Its size~~

→ Its size parameter specifies the column length in the bytes.

→ Default is 1.

VARBINARY(size)

→ It is equal to VARCHAR() but stores " " " .

→ " " " " maximum column length in bytes.

TEXT (size)

→ It holds a string that can contain a maximum length of 255 characters.

TINYTEXT

→ It holds a string with a maximum length of 255 characters.

MEDIUM TEXT:

→ It holds a string with a maximum length of 16,777,215.

LONG TEXT:

→ It holds a string with a maximum length of 4,294,967,295 characters.

ENUM (val1, val2, val3, ...)

→ It is used when a string object having only one value, chosen from a list of possible values.
→ It contains 65535 values in an ENUM list.
→ If you insert a value that is not in the list, a blank value will be inserted.

SET (val1, val2, val3, ...)

→ It is used to specify a string that can have 0 or more values, chosen from a list of possible values.
→ You can list up to 64 values at one time in a SET list.

BLOB (size)

→ It is used for BLOBs (Binary Large objects).
→ It can hold up to 65,535 bytes.

MySQL Numeric Data Types:

BIT (size)

→ It is used for a bit-value type.
→ The no. of bits per value is specified in size.
→ Its size can be 1 to 64.
→ The default value is 1.

INT (size)

→ It is used for integer value.
→ Its signed range varies from -2147483648 (-2^{31}) to 2147483647 (2^{31}) and unsigned range varies from 0 to $2^{32}-1$.
→ The size parameter specifies the max display width that is 255.

INTEGER (size)

→ It is equal to INT (size)

FLOAT (size, d)

→ It is used to specify a floating point number.
→ Its size parameter specifies the total

→ The no. of digits after decimal point is specified by 'd' parameter.

FLOAT(P)

→ It is used to specify a floating point number.

→ MySQL used P Parameter to determine whether to use FLOAT or DOUBLE.

→ If P is between 0 to 24, the data type becomes FLOAT().

→ If P is from 25 to 53, the data type becomes DOUBLE()

DOUBLE(size,d)

→ It is a normal size floating point number.

→ Its size parameter specifies the total number of digits.

DECIMAL(size,d)

→ It is used to specify fixed point number.

→ The maximum value for size is 65, and the default value is 10.

→ The maximum value for d is 30, & the default value is 0.

DEC(size,d)

→ It is equal to DECIMAL(size,d).

BOOL

→ It is used to specify Boolean values true and false.

→ Zero is considered as false,

→ non-zero " " " True.

MySQL Date and Time Data Types

DATE:

→ It is used to specify date format YYYY-MM-DD.

→ Its supported range is from '1000-01-01' to '9999-12-31'.

DATETIME(fsp)

→ It is used to specify date & time combination.

→ Its format is YYYY-MM-DD hh:mm:ss.

→ It supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

TIMESTAMP(fsp)

→ It is used to specify the timestamp.

→ Its value is stored as the no of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC).

→ Its format is YYYY-MM-DD hh:mm:ss

TIME(fsp)

→ It is used to specify the time format.

→ Its format is hh:mm:ss.

→ Its supported range is from '-838:59:59' to '838:59:59'

YEAR

→ It is used to specify a year in four-digit format.

→ values allowed in four digit format is from 1901 to 2155

SQL Operators

→ The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query.

→ In SQL, an operator can be a

Syntax of Unary SQL operator

Operand1 SQL-operator Operand2

Note

SQL operators are used for filtering the tables data by a specific condition in the SQL statement.

Precedence of SQL Operators

→ The precedence of SQL operators is the sequence in which the SQL evaluates the different operators in the same expression.

In the following table, the operators at the top have high precedence, and the operators that appear at the bottom have low precedence.

SQL Operator symbols	Operators
**	Exponential operator
~, -	Identity operator, Negation
+, -, ||	Addition(+), subtraction(-), String concatenation operator.
=, !=, <>, <=, >=, LIKE, IS NULL, IN, BETWEEN	Comparison Operators

NOT	Logical Negation operator
&& or AND	Conjunction operator
OR	Inclusion operator

Example

```
UPDATE employee
SET salary = 20-3*5 WHERE
                Emp_id=5;
```

→ Above example, salary is assigned 5.

Types of Operators

(1) SQL Arithmetic Operators
(2) " Comparison "
(3) " Logical "
(4) " Set "
(5) " Bit-wise "
(6) " Unary "

(1) SQL Arithmetic Operators :-

(i) SQL Addition operator (+)
(ii) " Subtraction " (-)
(iii) " Multiplication " (*)
(iv) " division " (/)
(v) " Modulus " (%).

(i) SQL Addition Operator (+)

→ It performs the addition on the numerical data of the database table.

Syntax

```
SELECT operand1 + operand2;
```

EmpID	EmpName	EmpSalary	Emp Monthly bonus
101	Tej	25000	4000
102	Raj	30000	200

Suppose we want to add 20,000 to salary of each employee then.

Ex)

```
SELECT Emp_Salary + 20000 as
Emp_New_salary FROM Employee_details;
```

Suppose we want to add the Salary and monthly bonus columns then

```
SELECT Emp_Salary + Emp_MonthlyBonus
as Emp_Total_Salary FROM Employee_
details;
```

(iv) SQL division operator (/)

syntax :

```
SELECT operand1 / operand2;
```

(v) SQL Modulus Operator (%)

syntax :

```
SELECT operand1 % operand2;
```

(ii) SQL subtraction operator (-)

syntax :-

```
SELECT operand1 - operand2;
```

example is same as above, but in place of '+', replace with '-'.

(iii) SQL multiplication operator (*)

syntax :

```
SELECT operand1 * operand2;
```

Ex :-

If we want to multiply the Emp_Id column to Emp_Salary column of that employee whose Emp_Id is 202, then

query :

```
SELECT Emp_Id * Emp_Salary as Emp_
Id * Emp_Salary FROM Employee_details
WHERE Emp_Id = 202;
```

(2) SQL Comparison Operators

→ these operators in SQL compare two different data of SQL table.

(i) SQL Equal Operator (=)
(ii) " Not " " (!=)
(iii) " Greater than " (>)
(iv) " " " Equals to " (>=)
(v) " Less than " (<)
(vi) " " " Equals to " (<=)

(i) SQL Equal operator (=)

→ This operator is highly used in SQL queries.

→ The Equal Operators in SQL shows only data that matches the specified value in the query.

Ex :

```
SELECT * FROM Employee_details
WHERE Emp_Salary = 30000;
```

→ In this example, we used the SQL equal operator with WHERE clause for getting the records of those employees whose salary is 30000

(ii) SQL Equal Not Operator (!=)

→ This operator shows only those data that do not match the query's specified value.

Ex :

```
SELECT * FROM Employee_details
WHERE Emp_Salary != 45000;
```

→ In this example, we used the SQL equal not operator with WHERE clause for getting the records of those employees whose Salary is not 45000.

(iii) SQL Greater Than operator (>)

→ This operator shows only those which are greater than the value of the right-hand of

```
SELECT * FROM Employee_deta
WHERE Emp_Id > 202;
```

→ Here, SQL greater than ope displays the records of thos

Ex:-

```
SELECT * FROM Employee_details
  WHERE Emp_Salary = 30000;
```

→ In this example, we used the SQL equal operator with WHERE clause for getting the records of those employees whose salary is 30000

(ii) SQL Equal Not Operator (!=)

→ This operator shows only those data that do not match the query's specified value.

Ex:-

```
SELECT * FROM Employee_details
  WHERE Emp_Salary != 45000;
```

→ In this example, we used the SQL equal not operator with WHERE clause for getting the records of those employees whose salary is not 45000.

(iii) SQL Greater than operator (>)

→ this operator shows only those data which are greater than the value of the right-hand operand.

```
SELECT * FROM Employee_details
  WHERE Emp_Id > 202;
```

→ Here, SQL greater than operator [illegible] employees from the above table whose Employee Id is greater than 202.

(iv) SQL Greater Than Equals to operator (>=)

→ This shows those data from the table which are greater than or equal to the value of right-hand operand.

Ex:-

```
SELECT * FROM Employee_details
  WHERE Emp_Id >= 202;
```

(v) SQL Less Than Operator (<)

Ex:-

```
SELECT * FROM Employee_details
  WHERE Emp_Id < 204;
```

(vi) SQL Less than Equal to (<=)

Ex:-

```
SELECT * FROM Employee_details
  WHERE Emp_Id <= 204;
```

(3) SQL Logical Operators:

→ The Logical operators in SQL perform the Boolean operations, which give two results TRUE and FALSE.

→ These operators provide TRUE value if both operands match the logical condition.

(i) SQL ALL Operator
(ii) " AND "
(iii) " OR "
(iv) " BETWEEN "
(v) " IN "
(vi) " NOT "
(vii) " ANY "
(viii) " LIKE "

1) SQL ALL operator

→ This operator compares the specified value to all the values of a column from the subquery in the SQL database.

This operator is always used with the following statement.

1. SELECT
2. HAVING, and
3. WHERE.

Syntax:

```
SELECT column_name1,.. column_nameN
FROM table_name WHERE column comparison_operator ALL (SELECT column
                                                      FROM tablename2)
```

Ex:

→ If we want to access the employee id & employee names of those employees from the table whose salaries are greater than the salary of employees who lives in Jaipur city, then we have to type following query in SQL

```
SELECT Emp_Id, Emp_Name FROM
Employee_details WHERE Emp_Salary > ALL(
SELECT Emp_Salary FROM Employee_details
  WHERE Emp_City = Jaipur)
```

→ Here we used the SQL ALL operator with greater than the operator

SQL AND operator

→ This operator in SQL would show the record from the database table if all the conditions seperated by the AND operator evaluated to TRUE.

→ It is also known as the conjunctive operator and is used with the WHERE clause.

Syntax:

```
SELECT column1,..., columnN FROM
table_name WHERE condition1 AND
condition2 AND ... conditionN;
```

Ex:-

```
SELECT * FROM Employee_details
WHERE Emp_Salary = 25000 AND
Emp_City = 'Delhi';
```

→ Here, SQL AND operator with WHERE clause shows the record of employees whose salary is 25000 and the city is Delhi.

SQL OR Operator:

→ The OR operator in SQL shows the record from the table if any of the conditions seperated by the OR operator evaluates to TRUE.

→ It is

Syntax:

```
SELECT columns,---,columnN FROM
table_name WHERE condition1 OR
condition2 OR condition3.
```

SQL BETWEEN Operator

→ This operator shows the records within the range mentioned in the SQL query.

→ This operator operates on the numbers, characters, & date/time operands.

→ If there is No value in the shows NULL value.

Syntax

```
SELECT column_names,--,columnNameN
FROM table_name WHERE
column_name BETWEEN value1 AND
value2;
```

Ex:-

```
SELECT* FROM employee_details
WHERE Emp_Salary BETWEEN 30000
AND 45000;
```

SQL IN Operator:

→ The IN operator in SQL allow database users to specify two or more values in a WHERE clause.

→ This operator minimizes the requirement of multiple OR conditions.

→ This operator returns those rows whose values match with any value of the given list.

Syntax

```
SELECT column_names,--,columnNameN
FROM table_Name
WHERE column_name IN (list_of_
values);
```

Ex:-

→ Suppose, we want to show all the information of those employees from the Employee_details table whose Employee Id is 202, 204, & 205.

query:-

```
SELECT * FROM Employee_details
WHERE Emp_Id IN (202,204,205);
```

query

```
SELECT * FROM Employee_details
WHERE Emp_Id NOT IN (202,205);
```

→ This shows all the information of those employees whose emp_Id is not equal to 202 & 205.

SQL NOT operator

→ This operator shows the record from the table if the condition evaluates to false.

→ It is always used with the WHERE clause.

Syntax:-

```
SELECT column1,...,columnN
FROM table_name
WHERE NOT condition;
```

Ex:-

```
SELECT * FROM Employee_details
WHERE NOT Emp_City = 'Delhi';
```

→ This shows all the information of Employees whose city is not Delhi.

Ex:-

```
SELECT * FROM Employee_details
WHERE NOT Emp_City = 'Delhi'
AND NOT Emp_City = 'Chandigarh';
```

SQL ANY Operator:-

→ This shows the records when any of the values returned by the sub-query meet the condition.

→ The ANY logical operator must match at least one record in the inner query and must be preceded by any SQL comparision operator.

Syntax:-

```
SELECT column1,...,columnN
FROM table_name
WHERE column_name comparison_operator
ANY (SELECT column_name FROM
table_name WHERE condition(s));
```

SQL LIKE Operator:

→ This shows those record from the table which match with the given pattern specified in the sub-query.

→ The percentage (%) sign is a wildcard which is used in conjunction with this logical operator.

This operator is used in the WHERE clause with the following three statements:

(i) SELECT statement
(ii) UPDATE "
(iii) DELETE "

Syntax:

```
SELECT column_name,...,column_nameN
FROM table_name WHERE column_name
  LIKE pattern;
```

Ex:-
→ If we want to show all the information of those employees whose name starts with "S".

query:

```
SELECT * FROM Employee_details
WHERE Emp_Name LIKE 'S%';
```

Ex:-
→ If we want show information regarding whose name ends with Y.

query:

```
SELECT * FROM Employee_details
WHERE Emp_Name LIKE '%Y';
```

→ If we want show information whose name starts with S & ends with Y ;

query:

```
SELECT * FROM Employee_details
WHERE EmpName LIKE 'S%Y';
```

(4) SQL set operators

(i) UNION

→ This operators combine a similar type of data from two or more SQL database tables.

→ It mixes the result, which is extracted from two or more SQL queries, into a single result.

→ Set operators combine more than one select statement in a single query and return a specific result set.

following are the various SQL operators which are performed on the "similar data" stored in the two SQL database tables!

(i) SQL Union operator
(ii) " Union ALL "
(iii) " Intersect "
(iv) " Minus "

(i) SQL Union Operator :-

→ This operator combines the result of two or more SELECT statements and provides the single output.

→ The datatype & the no-of columns must be the same for each SELECT statement used with the UNION operator.

→ This operator does not show the duplicate records in the output table.

Syntax :-

```
SELECT column1, ..., columnN FROM table_name1 WHERE [condition];
UNION
SELECT column1, ..., columnN FROM table_name2 WHERE [condition];
```

Ex)

```
SELECT Emp_ID, Emp_Name FROM Employee_details
UNION
SELECT Emp_ID, Emp_Name FROM Employee_details2;
```

SQL Union ALL operator :-

→ This is same as UNION operator, but the only difference is that it also shows the same record.

Syntax :-

```
SELECT column1, ..., columnN FROM table1 [WHERE condition]
UNION ALL
SELECT column1, ..., columnN FROM table2 [WHERE condition];
```

Ex)

→ If we want to see the employee name of each employee of both tables in a single output.

query)

```
SELECT Emp_Name FROM Employee_details1
UNION ALL
SELECT Emp_Name FROM Employee_details2;
```

SQL Intersect Operator:

→ This shows the common record from two or more SELECT statements.

→ The data type and the no. of columns must be the same for each SELECT statement used with the INTERSECT operator.

Syntax:-

```
SELECT column1, ..., columnN FROM table_name1
[WHERE condition]
INTERSECT
SELECT column1, ..., columnN FROM table_name2
[WHERE condition];
```

Ex:-
→ Suppose we want to see a common record of the employee from both the tables in a single output.

query:-

```
SELECT Emp_Name FROM Employee_details1
INTERSECT
SELECT Emp_Name FROM Employee_details2;
```

SQL Minus Operator:-

→ This operator combines the result of two or more SELECT statements & shows only the results from the first select statement that are not returned by the second select statement.

Syntax:-

```
SELECT column1, ..., columnN FROM table1
[WHERE conditions]
MINUS
SELECT column1, ..., columnN FROM table2
[WHERE conditions];
```

Ex:-
→ Suppose, we want to see the name of employees from the first result set after the combination of both tables.

query:-

```
SELECT Emp_Name FROM Employee_details1
MINUS
SELECT Emp_Name FROM Employee_details2;
```

(5) SQL Unary Operators

→ These operators perform the unary operations on the single data of SQL table, i.e., these operators operate only on one operand.

→ These types of operators can be easily operated on the numeric data value of the SQL table.

→ These are performed on the numeric data

(i) SQL Unary positive operator

(ii) " " Negative "

(iii) " " Bitwise NOT "

(i) SQL Unary Positive Operator:-

→ The SQL positive (+) operator makes the numeric value of the SQL table positive.

Syntax:-

```
SELECT +(column1), ..., +(columnN)
FROM table_name [WHERE conditions];
```

Ex:-

→ Suppose we want to see the salary of each employee as positive

query:-

```
SELECT +Emp_Salary FROM Employee_details;
```

(ii) SQL Unary Negative Operator:-

→ The SQL Negative (-) operator makes the numeric value of the SQL table negative.

Syntax:

```
SELECT -(column1), ..., -(columnN)
FROM table_name [WHERE conditions];
```

Ex:-

```
SELECT -Emp_Salary FROM Employee_details;
```

Ex:-

→ Suppose we want to see the salary of those employees as negative whose city is Kolkata.

Then query:-

```
SELECT -Emp_Salary FROM Employee_details WHERE Emp_City = 'Kolkata';
```

SQL Bitwise NOT operator:-

→ This operator provides the one's complement of the single numeric operand.

→ This operand turns each bit of numeric value.

→ If a bit of any numerical value is 001100, then this operator turns these bits into 110011.

Syntax:

```
SELECT ~(column1), ..., ~(columnN)
FROM table_name [WHERE conditions];
```

Ex:-

```
SELECT ~stu_marks FROM Employee_details;
```

6) SQL Bitwise Operators :

→ These operators perform the bit operations on the integer values.

→ To understand the performance of Bitwise operators, you just knew the basics of Boolean algebra.

(i) Bitwise AND (&)
(ii) Bitwise OR (|).

(i) Bitwise AND (&)

→ This performs logical AND operation on the given integer values.

→ This operator checks each bit of a value with the corresponding bit of another value.

Syntax :

```
SELECT column1&column2&....&columnN
FROM table_name [WHERE conditions];
```

Ex:-

→ Suppose, we want to perform the Bitwise AND operator b/w both the columns of table

Then query

```
SELECT column1&column2 FROM
                    Table_AND;
```

→ When we use the Bitwise AND operator in SQL, then SQL converts the value of both columns in binary format, and the AND operation is performed on the converted bits.

→ After that, SQL converts the resultant bits into user understandable format, i.e., decimal format.

(ii) Bitwise OR (|) :-

→ The Bitwise OR operator performs the logical OR operation on the given integer values.

→ This operator checks each bit of a value with the corresponding bit of another value.

Syntax :-

```
SELECT column1|column2|....|columnN
FROM tablename [WHERE conditions];
```

Ex:-

```
SELECT column1|column2
FROM Table_OR
```

SQL Create Database :-

→ It is a first step for storing the structured data in the database.

Syntax :-

```
CREATE DATABASE Database_Name;
```

Ex :-

```
CREATE DATABASE student;
```

O/P :-

Database created successfully.

→ You can also verify that your database is created in SQL or not.

query :-

```
SHOW DATABASE;
```

→ SQL does not allow developers to create the database with the existing database name.

→ If you want to create another student database in same database system, then it shows as

Can't create database 'student'; database exists

→ If you want to replace existing student database, then

```
REPLACE DATABASE student;
```

SQL DROP Database :-

→ The SQL DROP Database statement deletes the existing database permanently from the database system.

→ This statement deletes all the views and tables if stored in the database, so be careful while using this query.

We should remember some points before removing database.

* This statement deletes all the data from the database. If you want to restore the deleted data in the future, you should keep the backup of data of that database which you want to delete.

* Another most important point is that you cannot delete that database from the system which is currently in use by another database user.

→ If you do so, then the drop statement shows the following error on screen.

O/P :-

Cannot drop database 'nameofdatabase' because it is currently in use.

Syntax

```
DROP DATABASE Database_Name;
```

→ We can drop multiple databases

Syntax :-

```
DROP DATABASE Database1, Database2;
```

Example :-

```
SHOW DATABASES;
```

→ If student database is present in o/p then

```
DROP DATABASE student;
```

→ If student database doesn't exist in database system. Then

Can't drop database 'student'; database doesn't exist.

SQL RENAME Database

→ This statement is used to change the name of the existing data

Syntax :-

In SQL

```
ALTER DATABASE oldDatabase name
MODIFY NAME = new databasena
```

In MySQL :-

```
RENAME DATABASE old_database_n
TO new_database_name;
```

Syntax

DROP DATABASE Database_Name;

→ We can drop multiple databases

Syntax:

DROP DATABASE Database1, Database2, ...;

Example:

ALTER DATABASE student
MODIFY NAME = College;

→ This query will change the name of the database from ~~Department to Company~~ student to college.

Example:

SHOW DATABASES;

→ If student database is present in o/p then

DROP DATABASE student;

→ If student database doesn't exist in database system. Then

Can't drop database 'student'; database doesn't exist.

SQL SELECT Database:

→ Any database user & administrator can easily select the particular database from the current database server using the USE statement in SQL.

Syntax:

USE database_name;

ex:-

SHOW DATABASES;

→ If Hospital database is shown in o/p then

query

USE Hospital;

SQL RENAME Database:

→ This statement is used to change the name of the existing database.

Syntax:-

In SQL

ALTER DATABASE old_database_name
MODIFY NAME = new_database_name;

In MySQL:

RENAME DATABASE old_database_name
TO new_database_name;

SQL Table

→ Table is a collection of data organized in terms of rows and columns.

→ In DBMS terms, table is known as relation and row as tuple.

SQL TABLE Variable :-

→ It is used to create, modify, rename, copy and delete tables.

→ Table variables are used to store a set of records.

Syntax :

```
Create table "table_name"
("column1" "data type",
"column2" "data type",
 ...
"columnN" "data type");
```

→ Table variable cannot be used as an input or an output parameter.

SQL CREATE TABLE :

→ used to create table in database

syntax :

same as above syntax

SQL :

```
CREATE TABLE STUDENTS(
ID INT      NOT NULL,
NAME VARCHAR(20)  ,
AGE INT      ,
ADDRESS CHAR(25),
PRIMARY KEY (ID)
);
```

MySQL

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(255),
LastName varchar(255),
Email varchar(255),
AddressLine varchar(255),
City varchar(255),
);
```

Create a table using another Table :

→ we can create a copy of an existing table using the create table command

→ The new column gets the same column signature as the old table.

→ we can select all columns or some specific columns.

→ If we create a new table using an old table, the new table will be filled with the existing value from the old table.

Syntax:

```
CREATE TABLE table_name AS
SELECT column1, column2, ...
FROM oldtablename WHERE ....;
```

Ex:

→ The following query creates a copy of the employee table.

```
CREATE TABLE EmployeeCopy AS
SELECT EmployeeID, FirstName, Email
FROM Employee;
```

SQL DROP TABLE:

→ This statement is used to delete a table definition & all data from a table.

Syntax:

```
DROP TABLE "table_name";
```

SQL DELETE TABLE:

→ This statement is used to delete rows from a table.

→ If you want to remove specific row from a table you should use WHERE condition.

Syntax:-

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

Difference b/w TRUNCATE and DELETE statements

→ DELETE statement only deletes the rows from the table based on condition defined by WHERE clause & delete all the rows when condition is not specified. But it doesn't free the space containing by the table.

→ TRUNCATE is used to delete all the rows from the table and free the containing space.

Syntax:

```
TRUNCATE TABLE Employee;
```

Difference b/w DROP and TRUNCATE statements:-

when you drop a table

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints " " "
- Access privileges " " "

→ When you TRUNCATE a table, the table structure remains the same, so you will not face any of the above problems.

SQL RENAME TABLE :

→ The RENAME TABLE & ALTER TABLE syntax help in changing the name of the table.

Syntax :

```
RENAME oldtablename TO newtablename;
```

Ex:-

```
RENAME Cars TO Car_2021_Details;
```

Syntax of ALTER TABLE statement

```
ALTER TABLE oldtablename
  RENAME TO newtablename;
```

Ex:-

```
ALTER TABLE Bikes RENAME TO
        Bikes Details;
```

SQL TRUNCATE TABLE :-

→ This statement is used to remove all rows (complete data) from a table.

→ It is similar to DELETE statement with NO "WHERE" clause.

TRUNCATE VS DELETE :

→ Truncate table is faster and uses lesser resources than DELETE TABLE command.

TRUNCATE VS DROP :

→ Drop table command can also be used to delete complete table but it deletes table structure too.

→ TRUNCATE TABLE didn't delete the structure of the table.

Syntax :

```
TRUNCATE TABLE tablename;
```

Ex:-

```
TRUNCATE TABLE Employee;
```

SQL COPY TABLE :

→ If you want to copy the data of one SQL table into another SQL table in same SQL server then it is possible by using the SELECT INTO statement in SQL.

→ The SELECT INTO statement in SQL copies the content from one existing table into new table.

Syntax :-

```
SELECT * INTO newtablename
  FROM oldtablename ;
```

Example :-

```
SELECT* INTO CarDetails FROM Cars;
```

Ex :-

```
SELECT Car_Name, Car_Color INTO Car_Color
    FROM Cars ;
```

Syntax of SELECT INTO with WHERE clause :-

```
SELECT* INTO newtablename
  FROM oldtablename
  WHERE [condition];
```

Ex :-

```
SELECT * INTO Black_Car_Details
FROM Cars
WHERE Car_Color = 'Black';
```

Ex :-

```
SELECT * INTO Emp_Salary_4000
FROM Cars
WHERE Emp_Salary > 40000;
```

SQL TEMP Table :

→ The concept of temporary table is introduced by SQL server.

→ Temporary tables can be created at run-time and can do all kinds of operations that a normal table can do.

→ These temporary tables are created inside tempdb database.

There are two types of temp tables based on the behaviour and scope.

1. Local Temp Variable
2. Global " "

Local Temp Variable :-

→ Local temp tables are only available at current connection time.

→ It is automatically deleted when user disconnects from instances.

→ It is started with hash (#) sign.

Ex :-

```
CREATE TABLE #local temp table(
  user id int,
  Username varchar (50),
  User address varchar(150)
  );
```

Global Temp Variable

→ Global Temp table name starts with double hash (##).

→ Once, this table is created it is like a permanent table.

→ It is always ready for all users & not deleted until the total connection is withdrawn.

Ex)-

```
CREATE TABLE ##new global temp table(
    User id int,
    User name varchar(50),
    User address varchar(150)
    )
```

SQL ALTER TABLE :-

→ This statement allows you to add, modify and delete columns of an existing table.

→ This statement also allows database users to add and remove various SQL constraints on the existing tables.

→ Any user can also change the name of the table using this statements.

→ You can easily add single or multiple columns using ADD keyword.

Syntax :-

```
ALTER TABLE table_name
ADD column_name1 column_definition;
```

→ If you want to add multiple Columns

Syntax :-

```
ALTER TABLE table_name
ADD(column_name1 column_definition,
column_name2 column_definition,
. . .
column_nameN column_definition);
```

Ex)-

```
ALTER TABLE Cars
ADD Car_Model varchar(20);
```

Ex)-

```
ALTER TABLE Employee
ADD(Emp_ContactNo. Number(13),
    Emp_EmailID varchar(50));
```

ALTER TABLE MODIFY COLUMN statement:-

→ MODIFY keyword is used for changing the column definition of the existing table.

Syntax :-

```
ALTER TABLE table_name
MODIFY columnname column-definition;
```

→ for modifying multiple columns.

```
ALTER TABLE tablename
MODIFY (column_name1 columndefinition,
column_name2 column-definition,
---
columnnameN column-definition);
```

Syntax :-

```
ALTER TABLE table_name
DROP Column column_name;
```

Ex:-

```
ALTER TABLE Cars
DROP COLUMN Car_Color;
```

ALTER TABLE RENAME:-

Syntax :-

```
ALTER TABLE table_name
RENAME COLUMN old-name to new-name;
```

Ex:-

```
ALTER TABLE Cars
RENAME COLUMN Car_Colors to
Colors;
```

SELECT statement :-

Syntax :-

```
SELECT column1, ..., columnN
FROM Table_name;
```

Ex:-

```
CREATE TABLE Student_Records
(
Student_id Int PRIMARY KEY,
First_Name VARCHAR(20),
Address VARCHAR(11),
Age Int NOT NULL,
percentage Int (11),
Grade VARCHAR(10)
);
```

→ To insert records of students

```
INSERT INTO Student VALUES
(201, Akash, Delhi, 18.89, A2),
(202, ---- A1),
(207, vivek, Goa, 20, 82, B2);
```

→ The following SQL query displays all the values of each column

SELECT * FROM student_Records;

OUT).

Student_ID	First_name	Address	Age
201	Akash	Delhi	18

SELECT Student_Id, Age, Percentage, Grade
FROM Employee;

Select statement with WHERE clause.

Syntax):

SELECT * FROM tablename WHERE [condition];

Select statement with GROUP BY clause!

→ It is used with the SELECT statement to show the common data of the column from the table!

Syntax!

SELECT column1, column aggregate_function_name(column_name2)
FROM table_name
GROUP BY column_name1

Ex!)

Creating Car_details table

CREATE TABLE Cars_Details
(
Car_number INT PRIMARY KEY,
Car_Name VARCHAR(50),
Car_Price INT NOT NULL,
Car_Amount INT (1)
);

→ INSERT query inserts the record of cars in Car_Details table!

INSERT INTO Car_Details(Car_Number, Car_Name, Car_Amount, Car_Price)
VALUES(2578, Greta, 3, 1500000),
(9258, - -, - -, ---),
(- ———),
(- - —— -);

SELECT * FROM Car_Details;

Car_no	Car_name	Car_Amount	Car_Price
2578	Greta	3	1000000
9258	Audi	2	900000
8233	Venue	6	900000
6214	Nexon	7	1000000

SELECT COUNT(Car_Name), Car_Price
FROM Cars_Details
GROUP BY Car_Price;

OP!

Count(Car_Name	Car_Price
2	1000000
2	900 000

SELECT Statement with HAVING clause

→ The HAVING clause in the SELECT statement creates a selection in those groups which are defined by the GROUP BY clause.

Syntax!

```
SELECT column,--,columnN aggregatefunction(column name)
FROM table_name
GROUP BY column_name
HAVING;
```

Employee_HAVING table

EmployeID	Emp_Name	Emp_Salary	Emp_City
201	Jone	20000	Goa
202	Basant	40000	Delhi
203	Rashet	80000	Jaipur
204	Tej	20000	Goa
205	Raj	50000	Delhi

Query!

→ The following query shows the total salary of those employees having more than 50000 from the above Employee_Having table.

```
SELECT SUM(Emp_Salary), Emp_City
FROM Employee_Having
GROUP BY Emp_City
HAVING SUM(Emp_Salary) > 50000
```

OP!-

SUM(Emp_Salary)	Emp_City
90000	Delhi
80000	Jaipur

SELECT statement with ORDER BY Clause!

→ This clause with the SQL SELECT statement shows the records or rows in a sorted manner.

→ This clause arranges the values in both ascending and descending order.

→ Few databases systems arrange the values of column in ascending order by default.

Syntax!-

```
SELECT column1,---,columnN
FROM table_name
WHERE [condition]
ORDER BY [column1--columnN ASC|desc]
```

Ex:-

```
SELECT * FROM Employee_order
ORDER BY Emp_Salary DESC;
```

O/P:-

emp_id	emp_name	emp_Salary	emp_city
204	X	90000	Goa
203	Y	80000	Jaipur
205	Z	50000	Delhi
201	A	20000	Goa
202	B	15000	Delhi

SQL SELECT UNIQUE:-

→ Actually, there is no difference between DISTINCT and UNIQUE.

→ SELECT UNIQUE statement is used to retrieve a unique or distinct element from the table.

Syntax:-

```
SELECT UNIQUE column_name
FROM table_name;
```

Note:-

In SQL

SELECT UNIQUE } Both are same.
SELECT DISTINCT }

Ex:-

home_town
Lucknow
Varanasi
Lucknow

```
SELECT DISTINCT, home_town
FROM students
```

O/P:-

home_town
Lucknow
Varanasi

SQL SELECT COUNT:

→ The SQL COUNT() is a function that returns the no. of records of the table in the o/p

→ This statement is used with the SELECT statement.

Syntax:-

```
SELECT COUNT (column_name)
FROM table_name;
```

Ex:-

Bike_name	Bike_color	Bike_Cost
Pulsar	Black	185,000
Apache	Black	NULL
KTM RC	Red	90,000
Royal Enfield	white	NULL
Livo	Black	80,000
KTM DUKE	Red	195,000

Suppose, you want to count the total no-of bike colors from Bike Table.

```
SELECT COUNT(Bike_Color)
AS TotalBikeColor
FROM Bikes;
```

op!

TotalBikeColor
6

query

```
SELECT COUNT(Bike_Cost)
AS TotalBikeCost FROM Bikes;
```

op!

TotalBikeCost
4

→ The op of this query is 4, bez there are two NULL values in the column.

SELECT Count(*) function in SQL!

Syntax!-

```
SELECT COUNT(*)
FROM table_name;
```

→ The count(*) function in SQL shows all the NULL and Non-NULL records present in the table.

Count() with WHERE clause :-

Syntax!-

```
SELECT COUNT (column_name)
FROM table_name
WHERE [condition];
```

Ex!-

Bike_Color
Black
Black
Red
White
Red
Black

```
SELECT COUNT(Bike_Name)
AS TotalBikeBlackColor
FROM Bikes
WHERE Bike_Color = 'Black';
```

op!

TotalBikeBlackColor
3

Count function with DISTINCT keyword

→ The DISTINCT keyword with the COUNT function shows only the numbers of unique rows of a column.

Syntax!-

```
SELECT COUNT(DISTINCT column_name)
FROM table_name
WHERE [condition];
```

Col
Car_Color
White
BLACK
Red
White
White
Red

SELECT COUNT(DISTINCT Car_Color)
AS Unique_Car_Color FROM Cars;

O/P

Unique_Car_Color
3

SQL SELECT TOP

→ This statement shows the limited no. of records or rows from the database table.

→ The TOP clause in the statement specifies how many rows are returned.

Note:-

All the databases systems do not support the TOP keyword for selecting the limited no. of records.

Oracle supports the ROWNUM keyword, and MySQL supports the LIMIT keyword.

Syntax of TOP Clause in SQL

SELECT TOP number | percent column_name(s)
FROM table_name
WHERE [condition];

→ Suppose we want to show the first three Names & Color of Car query:-

SELECT TOP 3 Car_Name, Car_Color
FROM Cars;

→ Suppose we want to show all columns of 4 students

SELECT TOP 4 * FROM student;

SELECT TOP 4 * FROM Employee
WHERE EMP_City = Goa;

SELECT TOP 50 PERCENT *
FROM Bikes;

Syntax of LIMIT Clause in MySQL

SELECT column1, ..., columnN
FROM table_name
LIMIT value;

Ex!

→ Suppose, you want to show the first three records of Cars

query!

```
SELECT * FROM Cars LIMIT 3;
```

Syntax of ROWNUM Keyword in oracle

```
SELECT column1,...,columnN
FROM table_name
WHERE ROWNUM <= value;
```

query!

```
SELECT * FROM Cars WHERE
        ROWNUM <=3;
```

SQL SELECT FIRST!-

→ The SQL first() function is used to return the first value of the selected column.

Syntax

```
SELECT FIRST (column_name)
FROM table_name;
```

→ It is only works on MS Access but not supported by Oracle, MySQL etc...

SQL LAST!-

```
SELECT LAST (Student_Name)
AS Last_student
FROM student_Details;
```

O/P!-

Last_Student
Mohit

Syntax of LIMIT Clause in MySQL

```
SELECT column_Name
FROM Table_name
ORDER BY Column_Name DESC
LIMIT 1;
```

Ex! Emp_City

Delhi
Lucknow
Kolkata
Goa

```
SELECT Emp_City FROM Employee
ORDER BY Emp_City DESC LIMIT 1;
```

O/P!-

Goa.

SQL SELECT RANDOM :-

→ This function returns the random row.

→ It can be used in online exam to display the random questions.

Random row selection with MySQL :-

```
SELECT column FROM table
ORDER BY RAND()
LIMIT 1
```

SQL SELECT IN :

→ SQL IN is an operator used in a SQL query to help reduce the need to use multiple SQL "OR" conditions.

→ It is used in SELECT, INSERT, UPDATE or DELETE statement.

Syntax

```
Expression IN (val1, val2, ..., valn);
```

Ex:-

```
SELECT *
FROM students
WHERE student_name IN (Amit, Raghav, Rajeev)
```

SQL SELECT from Multiple Tables :

→ This statement is used to retrieve fields from multiple tables.

→ To do so, we need to use join query

Example :-

```
SELECT orders.order_id, suppliers.name
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id
ORDER BY order_id;
```

Let us take 3 tables

Customer1

Cus_id	Name1
1	Jack
2	Jill

Customer2

Cus_id	Name2
1	Sandy
2	Venus

Product table

P_id	Cus_id	P_name
1	1	Laptop
2	2	Phone
3	P1	Pen
4	P2	Notebook

```
SELECT P.P_id, P.Cus_id, P.P_name,
                        C1.name1, C2.name2
FROM product AS P
LEFT JOIN customer1 AS C1
ON P.Cus_id = C1.Cus_id
LEFT JOIN customer2 AS C2
ON P.Cus_id = C2.Cus_id ;
```

o/p:

P_id	Cus_id	P_name	P_name1	P_name2
1	1	Laptop	Jack	NULL
2	2	Phone	Jill	NULL
3	P1	Pen	NULL	Sandy
4	P2	Notebook	NULL	Venus

SQL SELECT DATE :-

→ This is used to retrieve a date from a database.

→ If you want to find a particular date from a database, you can use this statement.

Suppose we want to get all the records after '2013-12-12'.

query:

```
SELECT * FROM table_name
WHERE your date-column >= '2013-12-12'
```

Suppose we want to get records after '2013-12-12' & before '2013-12-13' date

query:

```
SELECT * FROM table-name
WHERE your date-column < '2013-12-13'
AND your date-column >= '2013-12-12'
```

→ query:

```
SELECT * FROM table_name
WHERE yourdate BETWEEN '2012-12-12'
AND '2013-12-12'
```

→ If you are looking for one date in particular you can use. You should change the date parameter into the acceptable form.

query:

```
SELECT * FROM table_name
WHERE cast (datediff (day, 0, yourdate)
        as datetime) = '2012-12-12'.
```

SQL SELECT SUM :-

→ It returns the summed value of an expression.

Syntax:

```
SELECT SUM (expression)
FROM tables
WHERE conditions;
```

→ expression may be numeric cell or formula

query:

```
SELECT SUM(DISTINCT salary)
AS "Total Salary"
FROM employees
WHERE salary > 20000;
```

SUM with GROUP BY:

query:

```
SELECT department, SUM(sales)
AS "Total Sales"
FROM order_details
GROUP BY department;
```

SQL SELECT NULL :-

→ NULL values are used to represent missing unknown data.

Ex:

SUR_NAME	NAME	MARKS
Tej	seema	
Raj	karan	5.5
Taj	Rana	
Ravi	Ranbir	6.2

query:

```
SELECT SUR_NAME, NAME, MARKS
FROM STUDENTS
WHERE MARKS IS NULL
```

OR:

SURNAME	NAME	MARKS
Tej	seema	
Taj	Rana	

query:

```
SELECT .
FROM .
WHERE MARKS IS NOT NULL
```

→ displays the values that are not null.

SQL WHERE

→ It is a DML statement.

→ It is not mandatory clause of DML statements. But it can be used to limit the no. of rows.

→ WHERE clause is used in SELECT, UPDATE, DELETE statement etc...

SQL AND :-

Syntax:

```
SELECT columns
FROM tables
WHERE condition1 AND condition2;
```

Ex:

```
SELECT * FROM emp
WHERE department = "IT" AND
Location = "Chennai";
```

AND with UPDATE statement in mysql

```
UPDATE emp SET location = "Delhi"
```

AND with DELETE statement:

mySQL:

```
DELETE FROM emp
WHERE Last_Name = 'Jain' AND
      Location = 'Bangalore';
```

SQL OR

Syntax:

```
SELECT columns
FROM tables
WHERE condition1 OR condition2;
```

SQL WITH:

→ This is used to provide a sub-query block which can be referenced in several places within the main SQL query.

Syntax using a single sub-query alias.

```
WITH <alias_name> AS (sql_sub-query)
SELECT column_list FROM <alias_name>
[WHERE <join_condition>]
```

(table name)

Syntax for multiple sub-query aliases:

```
WITH alias_name_A> AS (sql_sub-query)
<alias_name_B> AS (sql_sub-query_from_aliasname_A
  OR sql_subquery statement)
SELECT <column_list>
FROM <alias_name_A>, <alias_name_B>, [table name]
```

SQL SELECT AS:

→ This is used to assign a new name temporarily to a table column or even a table.

Syntax:

```
SELECT column1 AS new_column_name,
       column2 AS
FROM Table_Name;
```

query:

```
SELECT day_of_order AS 'Date',
       Customer AS 'Client',
       Product, Quantity
FROM orders;
```

* Note:

SQL AS is the same as SQL ALIAS

query:

```
SELECT Student_Name AS Student,
       AVG(Student_percentage) AS Avgpercen
FROM students
```

OR:

Student	Average_percentage
Rena More	88.400

query :-

```
SELECT student_RollNo AS 'Roll No',
  CONCAT(student_Phono, ',', studentAhometown) AS student Info
FROM students;
```

→ Here concat() function combines two different columns.

O/P :-

Roll No	Mobile number
1	8342248572, Lucknow
2	123456, Delhi
3		
4	8124567, surat
5	

Using temporary name

query :-

```
SELECT s.student_RollNo, s.studentname
FROM students AS s
WHERE s.student_RollNo = 3;
```

O/P :-

st_RollNo	stu_name		
3	Kashish Gola	pune	Pune

HAVING Clause in SQL

→ This clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

→ WHERE clause uses condition for filtering records before any groupings are made.

→ HAVING clause uses condition for filtering values from a group.

HAVING

→ Used to fetch data/values from a groups according to conditions.

→ Always executed with the GROUP BY clause.

→ Can include aggregate functions in a query or statement.

→ Used to filter groups.

→ Implement this clause in column operation.

→ It is a post-filter.

WHERE :

→ Used to fetch data/values from tables according to conditions.

→ executed without GROUP BY clause

→ Cannot include aggregate functions.

→ used to filter single row in a

→ Implements this clause in row operations.

→ It is a pre-filter.

Syntax of HAVING clause :

```
SELECT column_name, column_name aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Ex :-

query

```
SELECT SUM(Emp_salary), Emp_City
FROM Employee
GROUPBY Emp_city;
```

O/P :-

SUM(Emp_salary)	Emp_city
4000	Goa
9000	Delhi
8000	Jaipur

query

```
SELECT SUM( ), ...
FROM ''
GROUP BY ...;
HAVING SUM(Emp_salary) ...
```

O/P :-

Emp_Sal	Emp_City
9000	Delhi
8000	Jaipur

→ Implements this clause in row operations.

→ It is a pre-filter.

Syntax of HAVING clause:

```
SELECT column_1, column_N aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Ex:-

query

```
SELECT SUM(Emp_Salary), Emp_City
FROM Employee
GROUP BY Emp_City;
```

OR

SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

query

```
SELECT SUM( ),
FROM ;;
GROUP BY --;
HAVING SUM(Emp_Salary) > 5000
```

OR

Emp_Sal	Emp_City
9000	Delhi
8000	Jaipur

SQL ORDER BY Clause :-

→ Whenever we want to sort the records based on the columns stored in the tables, of ~~SQL database~~ then ORDER BY is used.

Syntax to store records in ascending order;

```
SELECT column1, ..., columnN
FROM TableName
ORDER BY ColumnName ASC;
```

Syntax to store records in descending order;

```
SELECT column1, ..., columnN
FROM TableName
ORDER BY ColumnName DESC;
```

Ex:-

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

SQL ORDER BY RANDOM

→ If you want the resulting record to be ordered randomly, we use this.

syntax:-

```
SELECT column
FROM table
ORDER BY RAND();
```

query:-

```
SELECT * FROM items
ORDER BY RAND() LIMIT 1;
```

→ only one row will display randomly.

→ If you run this query, different o/p will show for every run.

SQL SORTING ON MULTIPLE COLUMNS:-

```
SELECT * FROM customers
ORDER BY country, Customer_Name;
```

SQL INSERT STATEMENT

→ It is used to insert a single or multiple records in a table.

1. By SQL insert into statement
   1. By specifying column names
   2. without " " "
2. By SQL insert into select statem

(1) Inserting data directly into a ta

→ you can insert a row in the table by using SQL INSERT INTO command.

no need to specify the column nam

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

Specifying both the column name

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

(2) INSERT data through SELECT statemen

syntax:-

```
INSERT INTO table_name
[(column1, ..., columnN)]
SELECT column1, ..., columnN
FROM table_name [WHERE condition];
```

Note: when you add a new row, you should make sure that data type of the value and the column should be matched.

SQL INSERT Multiple Rows :- SQL UPDATE :-

Ex:-

→ To create a table in database, first we need to select the database in which we want to create a table.

query

```
USE dbs;
```

→ Then we will write a query to create a table named student in the selected database 'dbs'.

query :

```
CREATE TABLE student(
    ID INT, Name VARCHAR(20),
    Percentage INT, ...);
```

→ student table is created.

→ Single query to insert multiple rows.

```
INSERT INTO student (ID, Name, Percentage,
                     location, DOB)
VALUES (1, "Teja", 75, "Delhi", 01-01-1998),
(2, "Dev", 75, "Pune", 1-1-2), (------), (---)
(--), - - - ; (--));
```

→ This command is used to modify the data that is already in the database.

Syntax :-

```
UPDATE table_name
SET [column_name1 = value1, ...
     column_nameN = valueN]
WHERE condition
```

SQL Update with JOIN :

→ This means we will update one table using another table and join condition.

Syntax of sql update query with JOIN statement:

```
UPDATE customer_table
INNER JOIN
Customer_table
ON customer_table.rel_cust_name =
   Customer_table.cust_id
SET customer_table.rel_cust_name =
   Customer_table.cust_id
```

Ex:-

table1 :

	Col1	Col2	Col3
1	1	11	First
2	11	12	second
3	21	13	third
4	31	14	fourth

	Col1	Col2	Col3
1	1	21	Two-one
2	11	22	Two-Two
3	21	23	Two-three
4	31	24	Two-four

query:-

```
UPDATE table1
SET Col2 = t2.Col2,
    Col3 = t2.Col3
FROM table1 t1
INNER JOIN table2 t2 ON t1.Col1 = t2.Col1
WHERE t1.Col1 IN (21,31)
```

O/P:-

```
SELECT * FROM table1
```

	Col1	Col2	Col3
1	1	11	First
2	11	12	second
3	21	23	Two-three
4	31	24	Two-four

SQL UPDATE DATE:-

Syntax:

```
UPDATE table
SET Column_Name = 'YYYY-MM-DD HH:MM:SS'
WHERE Id = Value
```

Ex:

```
UPDATE table
SET EndDate = '2014-03-16 00:00:00.000'
WHERE H = 1
```

SQL DELETE:-

→ It is used to delete rows from a table.

Syntax

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

Ex:-

```
DELETE FROM EMPLOYEE
WHERE ID = 101;
```

→ The row gets deleted where id is 101.

query

```
DELETE FROM EMPLOYEE;
```

O/P:

ID	EMP_NAME	CITY	SALARY

→ This query deletes all the information but table remains same.

SQL DELETE DUPLICATE ROWS:-

Syntax:

```
SELECT DISTINCT Column1, ---, ColumnN
FROM table_name
WHERE condition;
```

Q1:-

```
SELECT DISTINCT Percentage
 FROM students
ORDER BY Percentage;
```

SQL DELETE VIEW:

SQL VIEW

→ A view is a result of a stored query on the data.

→ SQL view can be created by a SQL query by joining one or more table.

Syntax:

```
CREATE VIEW view_name AS
SELECT columns
 FROM tables
 WHERE conditions;
```

→ If you want to delete a SQL view, query:-

```
DROP VIEW view_name
```

SQL DELETE JOIN :-

→ It is not a very easy process, Sometimes, we need to update or delete records on the basis of complex WHERE clauses.

→ It need three tables, used to operate an SQL Syntax for DELETE JOIN,

Syntax :- for deleting JOIN

```
DELETE [target table]
FROM [table 1]
 INNER JOIN [table 2]
ON [table1.[joining column] =
     [table2.[joing column]
WHERE [condition]
```

Syntax for Updating JOIN

```
UPDATE [target table]
SET [target column] = [new value]
FROM [table 1]
    INNER JOIN [table 2]
ON [table1.[joining co
WHERE
```

SQL JOIN

→ JOIN means "to combine two or more tables".

There are 5 types of JOINS

(i) INNER JOIN
(ii) LEFT OUTER "
(iii) RIGHT " "
(iv) FULL " "
(v) CROSS JOIN

The joining of two tables is based on common field between them.

SQL LEFT JOIN

→ This join returns all the values from the left table and it also includes matching values from right table, if there are no matching join value it returns NULL.

Syntax:

```
SELECT table1.column1, table2.column...
FROM table1
LEFT JOIN table2
ON table1.column_field = table2.column_field;
```

Ex:

CUSTOMER TABLE

ID	NAME	Age	Salary
1	X	51	56K
2	Y	21	25K
3	Z	24	31K
4	A	23	32K
5	B	23	42K

ORDER TABLE:

O_ID	DATE	CUST_ID	Amount
001	20-01-2012	2	3K
002	12-02-2012	2	2K
003	22-03-2012	3	4K
	11-04-2012	4	5K

Join these two tables with LEFT JOIN

query:

```
SELECT ID, NAME, AMOUNT, DATE
FROM CUSTOMER
LEFT JOIN ORDER
ON CUSTOMER.ID = ORDER.CUST_ID;
```

O/P:

ID	NAME	AMOUNT	DATE
1	X	NULL	NULL
2	Y	3K	20-01-2012
2	Y	2K	12-02-2012
3	Z	4K	22-03-2012
4	A	5K	11-04-2012
5	B	NULL	NULL

SQL RIGHT JOIN :-

→ This join returns all the values from the rows of right table, it also includes the matched values from left table but if there is no matching in both tables, it returns NULL.