What is SQL?

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS.

You can easily create and manipulate the database, access and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

If you want to get a job in the field of data science, then it is the most important query language to learn. Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back-end.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database management systems.
- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.
- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

History of SQL

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd. They both developed the SQL at the San Jose Research laboratory of IBM Corporation in 1970.

At the end of the 1970s, relational software Inc. developed their own first SQL using the concepts of E.F. Codd, Raymond Boyce, and Donald Chamberlin. This SQL was totally based on RDBMS. Relational Software Inc., which is now known as Oracle Corporation, introduced the Oracle V2 in June 1979, which is the first implementation of SQL language. This Oracle V2 version operates on VAX computers.

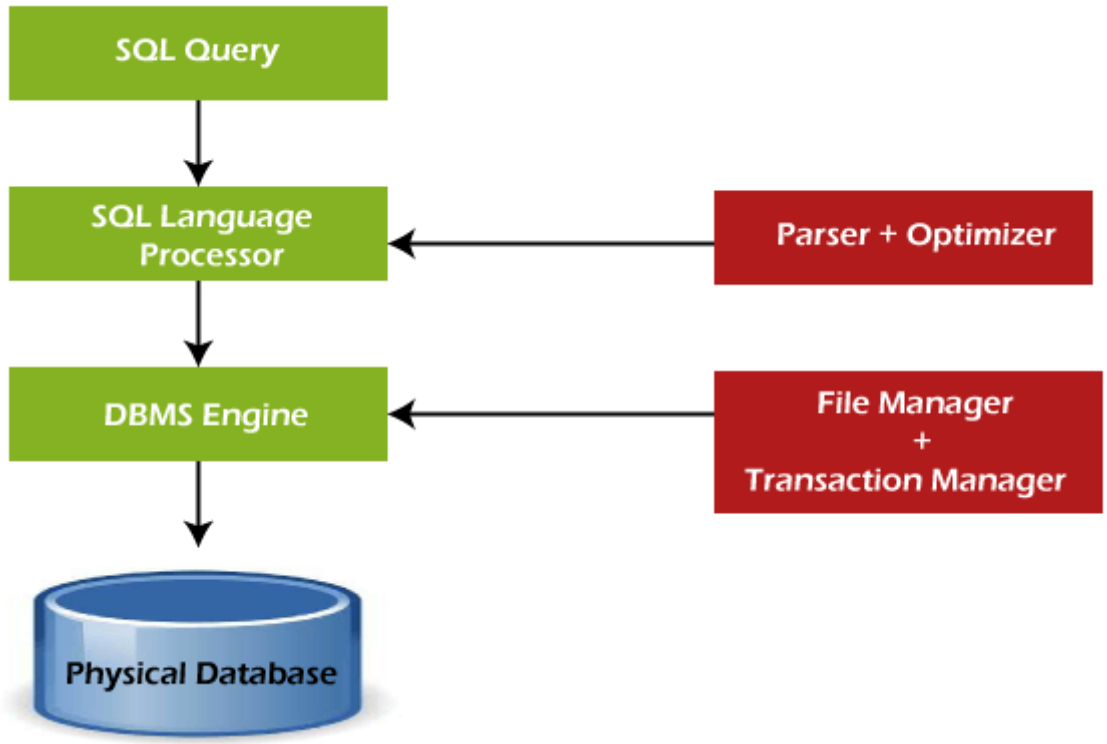
Process of SQL

When we are executing the command of SQL on any Relational database management system, then the system automatically finds the best routine to carry out our request, and the SQL engine determines how to interpret that particular command.

Structured Query Language contains the following four components in its process:

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine allows data professionals and users to maintain non-SQL queries. The architecture of SQL is shown in the following diagram:



Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command
3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

CREATE Command

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

UPDATE Command

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

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 database.

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.

DROP Command

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

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 vs No-SQL

The following table describes the differences between the SQL and NoSQL, which are necessary to understand:

SQL	No-SQL
1. SQL is a relational database management system.	1. While No-SQL is a non-relational or distributed database management system.
2. The query language used in this database system is a structured query language.	2. The query language used in the No-SQL database systems is a non-declarative query language.

3. The schema of SQL databases is predefined, fixed, and static.	3. The schema of No-SQL databases is a dynamic schema for unstructured data.
4. These databases are vertically scalable.	4. These databases are horizontally scalable.
5. The database type of SQL is in the form of tables, i.e., in the form of rows and columns.	5. The database type of No-SQL is in the form of documents, key-value, and graphs.
6. It follows the ACID model.	6. It follows the BASE model.
7. Complex queries are easily managed in the SQL database.	7. NoSQL databases cannot handle complex queries.
8. This database is not the best choice for storing hierarchical data.	8. While No-SQL database is a perfect option for storing hierarchical data.
9. All SQL databases require object-relational mapping.	9. Many No-SQL databases do not require object-relational mapping.
10. Gauges, CircleCI, Hootsuite, etc., are the top enterprises that are using this query language.	10. Airbnb, Uber, and Kickstarter are the top enterprises that are using this query language.
11. SQLite, Ms-SQL, Oracle, PostgreSQL, and MySQL are examples of SQL database systems.	11. Redis, MongoDB, Hbase, BigTable, CouchDB, and Cassandra are examples of NoSQL database systems.

Advantages of SQL

SQL provides various advantages which make it more popular in the field of data science. It is a perfect query language which allows data professionals and users to communicate with the database. Following are the best advantages or benefits of Structured Query Language:

1. No programming needed

SQL does not require a large number of coding lines for managing the database systems. We can easily access and maintain the database by using simple SQL syntactical rules. These simple rules make the SQL user-friendly.

2. High-Speed Query Processing

A large amount of data is accessed quickly and efficiently from the database by using SQL queries. Insertion, deletion, and updation operations on data are also performed in less time.

3. Standardized Language

SQL follows the long-established standards of ISO and ANSI, which offer a uniform platform across the globe to all its users.

4. Portability

The structured query language can be easily used in desktop computers, laptops, tablets, and even smartphones. It can also be used with other applications according to the user's requirements.

5. Interactive language

We can easily learn and understand the SQL language. We can also use this language for communicating with the database because it is a simple query language. This language is also used for receiving the answers to complex queries in a few seconds.

6. More than one Data View

The SQL language also helps in making the multiple views of the database structure for the different database users.

Disadvantages of SQL

With the advantages of SQL, it also has some disadvantages, which are as follows:

1. Cost

The operation cost of some SQL versions is high. That's why some programmers cannot use the Structured Query Language.

2. Interface is Complex

Another big disadvantage is that the interface of Structured query language is difficult, which makes it difficult for SQL users to use and manage it.

3. Partial Database control

The business rules are hidden. So, the data professionals and users who are using this query language cannot have full database control.

SQL Syntax

When you want to do some operations on the data in the database, then you must have to write the query in the predefined syntax of SQL.

The syntax of the structured query language is a unique set of rules and guidelines, which is not case-sensitive. Its Syntax is defined and maintained by the ISO and ANSI standards.

Following are some most important points about the SQL syntax which are to remember:

- You can write the keywords of SQL in both uppercase and lowercase, but writing the SQL keywords in uppercase improves the readability of the SQL query.
- SQL statements or syntax are dependent on text lines. We can place a single SQL statement on one or multiple text lines.

- You can perform most of the action in a database with SQL statements.
- SQL syntax depends on relational algebra and tuple relational calculus.

SQL Statement

SQL statements tell the database what operation you want to perform on the structured data and what information you would like to access from the database.

The statements of SQL are very simple and easy to use and understand. They are like plain English but with a particular syntax.

Simple Example of SQL statement:

1. **SELECT** "column_name" **FROM** "table_name";

Each SQL statement begins with any of the SQL keywords and ends with the semicolon (;). The semicolon is used in the SQL for separating the multiple Sql statements which are going to execute in the same call. In this SQL tutorial, we will use the semicolon (;) at the end of each SQL query or statement.

Most Important SQL Commands and Statements

1. Select Statement
2. Update Statement
3. Delete Statement
4. Create Table Statement
5. Alter Table Statement
6. Drop Table Statement
7. Create Database Statement
8. Drop Database Statement
9. Insert into Statement
10. Truncate Table Statement
11. Describe Statement
12. Distinct Clause
13. Commit Statement
14. Rollback Statement
15. Create Index Statement
16. Drop Index Statement
17. Use Statement

Let's discuss each statement in short one by one with syntax and one example:

1. SELECT Statement

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

Syntax of SELECT Statement:

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

Example of SELECT Statement:

```
1. SELECT Emp_ID, First_Name, Last_Name, Salary, City
2. FROM Employee_details
3. WHERE Salary = 100000
4. ORDER BY Last_Name
```

This example shows the **Emp_ID, First_Name, Last_Name, Salary, and City** of those employees from the **Employee_details** table whose **Salary** is **100000**. The output shows all the specified details according to the ascending alphabetical order of **Last_Name**.

3. UPDATE Statement

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

Syntax of UPDATE Statement:

```
1. UPDATE table_name
2. SET column_name1 = new_value_1, column_name2 = new_value_2, ...., column_nameN = new_val
   ue_N
3. [ WHERE  CONDITION ];
```

Example of UPDATE Statement:

```
1. UPDATE Employee_details
2. SET Salary = 100000
3. WHERE Emp_ID = 10;
```

This example changes the **Salary** of those employees of the **Employee_detail**s table whose **Emp_ID** is **10** in the table.

3. DELETE Statement

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

Syntax of DELETE Statement:

```
1. DELETE FROM table_name
2. [ WHERE CONDITION ];
```

Example of DELETE Statement:

```
1. DELETE FROM Employee_details
2. WHERE First_Name = 'Sumit';
```

This example deletes the record of those employees from the **Employee_details** table whose **First_Name** is **Sumit** in the table.

4. CREATE TABLE Statement

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

Syntax of CREATE TABLE Statement:

```
1. CREATE TABLE table_name
2. (
3. column_name1 data_type [column1 constraint(s)],
4. column_name2 data_type [column2 constraint(s)],
5. column_nameN data_type [columnN constraint(s)],
6. PRIMARY KEY(one or more col)
7. );
```

Example of CREATE TABLE Statement:

```
1. CREATE TABLE Employee_details(
2.     Emp_Id NUMBER(4) NOT NULL,
3.     First_name VARCHAR(30),
4.     Last_name VARCHAR(30),
5.     Salary Money,
6.     City VARCHAR(30),
7.     PRIMARY KEY (Emp_Id)
8. );
```

This example creates the table **Employee_details** with five columns or fields in the SQL database. The fields in the table are **Emp_Id, First_Name, Last_Name, Salary,** and **City**. The **Emp_Id** column in the table acts as a **primary key**, which means that the Emp_Id column cannot contain duplicate values and null values.

5. ALTER TABLE Statement

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

Syntax of ALTER TABLE Statement:

```
1. 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.

```
1. 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.

```
1. ALTER TABLE table_name DROP COLUMN column_name;
```

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

Example of ALTER TABLE Statement:

```
1. ALTER TABLE Employee_details
2. 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 of DROP TABLE Statement:

```
1. DROP TABLE [ IF EXISTS ]
2. table_name1, table_name2, ……, table_nameN;
```

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

Example of DROP TABLE Statement:

```
1. 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 of CREATE DATABASE Statement:

```
1. CREATE DATABASE database_name;
```

Example of CREATE DATABASE Statement:

```
1. 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 of DROP DATABASE Statement:

```
1. DROP DATABASE database_name;
```

Example of DROP DATABASE Statement:

```
1. 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 insert a single record:

```
1. INSERT INTO table_name
2. (
3. column_name1,
4. column_name2, ….,
5. column_nameN
6. )
7. VALUES
8. (value_1,
9. value_2, …..,
10. value_N
11. );
```

Example of insert a single record:

```
1. INSERT INTO Employee_details
2. (
3. Emp_ID,
4. First_name,
5. Last_name,
6. Salary,
7. City
8. )
9. VALUES
10. (101,
11. Akhil,
12. Sharma,
13. 40000,
14. Bangalore
15. );
```

This example inserts **101** in the first column, **Akhil** in the second column, **Sharma** in the third column, **40000** in the fourth column, and **Bangalore** in the last column of the table **Employee_details**.

Syntax of inserting a multiple records in a single query:

```
1. INSERT INTO table_name
2. ( column_name1, column_name2, …., column_nameN)
3. VALUES (value_1, value_2, ….., value_N), (value_1, value_2, ….., value_N),….;
```

Example of inserting multiple records in a single query:

```
1. INSERT INTO Employee_details
2. ( Emp_ID, First_name, Last_name, Salary, City )
3. VALUES (101, Amit, Gupta, 50000, Mumbai), (101,  John, Aggarwal, 45000, Calcutta), (101, Sidh
   u, Arora, 55000, Mumbai);
```

This example inserts the records of three employees in the **Employee_details** table in the single query statement.

10. TRUNCATE TABLE Statement

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

Syntax of TRUNCATE TABLE Statement:

```
1. TRUNCATE TABLE table_name;
```

Example of TRUNCATE TABLE Statement:

```
1. 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 of DESCRIBE Statement:

```
1. DESCRIBE table_name | view_name;
```

Example of DESCRIBE Statement:

```
1. 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 of DISTINCT Clause:

```
1. SELECT DISTINCT column_name1, column_name2, ...
2. FROM table_name;
```

Example of DISTINCT Clause:

```
1. SELECT DISTINCT City, Salary
2. FROM Employee_details;
```

This example shows the distinct values of the **City** and **Salary** column from the **Employee_details** table.

13. COMMIT Statement

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

Syntax of COMMIT Statement:

1. **COMMIT**

Example of COMMIT Statement:

```
1. DELETE FROM Employee_details
2. WHERE salary = 30000;
3. 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 of ROLLBACK Statement:

```
1. ROLLBACK
```

Example of ROLLBACK Statement:

```
1. DELETE FROM Employee_details
2. WHERE City = Mumbai;
3. 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 of CREATE INDEX Statement:

```
1. CREATE INDEX index_name
2. ON table_name ( column_name1, column_name2, …, column_nameN );
```

Example of CREATE INDEX Statement:

```
1. CREATE INDEX idx_First_Name
2. 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 SQL statement deletes the existing index of the SQL database table.

Syntax of DROP INDEX Statement:

```
1. DROP INDEX index_name;
```

Example of DROP INDEX Statement:

```
1. DROP INDEX idx_First_Name;
```

This example deletes the index **idx_First_Name** from the SQL database.

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 of USE Statement:

1. USE database_name;

Example of USE DATABASE Statement:

2. 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. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.

Data types mainly classified into three categories for every database.

- String Data types
- Numeric Data types
- Date and time Data types

Data Types in MySQL, SQL Server and Oracle Databases

MySQL Data Types

A list of data types used in MySQL database. This is based on MySQL 8.0.

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 used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the 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.

MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	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 number 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 the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. 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 number of digits. The number of digits after the 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 to24, 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. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by **d** parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for **d** is 30, and 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, and nonzero values are considered as 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 and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its 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 number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
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 from 1901 to 2155, and 0000.

SQL Server Data Types

SQL Server String Data Type

char(n)	It is a fixed width character string data type. Its size can be up to 8000 characters.
varchar(n)	It is a variable width character string data type. Its size can be up to 8000 characters.
varchar(max)	It is a variable width character string data types. Its size can be up to 1,073,741,824 characters.
text	It is a variable width character string data type. Its size can be up to 2GB of text data.
nchar	It is a fixed width Unicode string data type. Its size can be up to 4000 characters.
nvarchar	It is a variable width Unicode string data type. Its size can be up to 4000 characters.

ntext	It is a variable width Unicode string data type. Its size can be up to 2GB of text data.
binary(n)	It is a fixed width Binary string data type. Its size can be up to 8000 bytes.
varbinary	It is a variable width Binary string data type. Its size can be up to 8000 bytes.
image	It is also a variable width Binary string data type. Its size can be up to 2GB.

SQL Server Numeric Data Types

bit	It is an integer that can be 0, 1 or null.
tinyint	It allows whole numbers from 0 to 255.
Smallint	It allows whole numbers between -32,768 and 32,767.
Int	It allows whole numbers between -2,147,483,648 and 2,147,483,647.
bigint	It allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.
float(n)	It is used to specify floating precision number data from -1.79E+308 to 1.79E+308. The n parameter indicates whether the field should hold the 4 or 8 bytes. Default value of n is 53.
real	It is a floating precision number data from -3.40E+38 to 3.40E+38.
money	It is used to specify monetary data from -922,337,233,685,477.5808 to 922,337,203,685,477.5807.

SQL Server Date and Time Data Type

datetime	It is used to specify date and time combination. It supports range from January 1, 1753, to December 31, 9999 with an accuracy of 3.33 milliseconds.
datetime2	It is used to specify date and time combination. It supports range from January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds
date	It is used to store date only. It supports range from January 1, 0001 to December 31, 9999
time	It stores time only to an accuracy of 100 nanoseconds

timestamp	It stores a unique number when a new row gets created or modified. The time stamp value is based upon an internal clock and does not correspond to real time. Each table may contain only one-time stamp variable.

SQL Server Other Data Types

Sql_variant	It is used for various data types except for text, timestamp, and ntext. It stores up to 8000 bytes of data.
XML	It stores XML formatted data. Maximum 2GB.
cursor	It stores a reference to a cursor used for database operations.
table	It stores result set for later processing.
uniqueidentifier	It stores GUID (Globally unique identifier).

Oracle Data Types

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)
NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.

FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.
LONG RAW	It is used to specify variable length raw binary data. Its range up to 2^{31}-1 bytes or 2 GB, per row.

SQL Operators

Every database administrator and user uses SQL queries for manipulating and accessing the data of database tables and views.

The manipulation and retrieving of the data are performed with the help of reserved words and characters, which are used to perform arithmetic operations, logical operations, comparison operations, compound operations, etc.

What is SQL Operator?

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 either be a unary or binary operator. The unary operator uses only one operand for performing the unary operation, whereas the binary operator uses two operands for performing the binary operation.

Syntax of Unary SQL Operator

1. Operator SQL_Operand

Syntax of Unary SQL Operator

1. Operand1 SQL_Operator Operand2

> ***Note: SQL operators are used for filtering the table's data by a specific condition in the SQL statement.***

What is the Precedence of SQL Operator?

The precedence of SQL operators is the sequence in which the SQL evaluates the different operators in the same expression. Structured Query Language evaluates those operators first, which have high precedence. 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
**	Exponentiation operator
+, -	Identity operator, Negation operator
*, /	Multiplication operator, Division operator
+, -, ||	Addition (plus) operator, subtraction (minus) operator, String Concatenation operator
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison Operators
NOT	Logical negation operator
&& or AND	Conjunction operator
OR	Inclusion operator

For Example,

```
1. UPDATE employee
2. SET salary = 20 - 3 * 5 WHERE Emp_Id = 5;
```

In the above SQL example, salary is assigned 5, not 85, because the * (Multiplication)

Operator has higher precedence than the - (subtraction) operator, so it first gets multiplied with 3*5 and then subtracts from 20.

Types of Operator

SQL operators are categorized in the following categories:

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

Let's discuss each operator with their types.

SQL Arithmetic Operators

The **Arithmetic Operators** perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.

Following are the various arithmetic operators performed on the SQL data:

1. SQL Addition Operator (+)
2. SQL Subtraction Operator (-)
3. SQL Multiplication Operator (+)
4. SQL Division Operator (-)
5. SQL Modulus Operator (+)

SQL Addition Operator (+)

The **Addition Operator** in SQL performs the addition on the numerical data of the database table. In SQL, we can easily add the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also add the numbers to the existing numbers of the specific column.

Syntax of SQL Addition Operator:

```
1. SELECT operand1 + operand2;
```

Let's understand the below example which explains how to execute Addition Operator in SQL query:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_Monthlybonus.**

Emp Id	Emp Name	Emp Salary	Emp Monthlybonus
101	Tushar	25000	4000
102	Anuj	30000	200

- Suppose, we want to add **20,000** to the salary of each employee specified in the table. Then, we have to write the following query in the SQL:

```
1. SELECT Emp_Salary + 20000 as Emp_New_Salary FROM Employee_details;
```

In this query, we have performed the SQL addition operation on the single column of the given table.

- Suppose, we want to add the Salary and monthly bonus columns of the above table, then we have to write the following query in SQL:

```
1. SELECT Emp_Salary + Emp_Monthlybonus as Emp_Total_Salary FROM Employee_details;
```

In this query, we have added two columns with each other of the above table.

SQL Subtraction Operator (-)

The Subtraction Operator in SQL performs the subtraction on the numerical data of the database table. In SQL, we can easily subtract the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also subtract the number from the existing number of the specific table column.

Syntax of SQL Subtraction Operator:

```
1. SELECT operand1 - operand2;
```

Let's understand the below example which explains how to execute Subtraction Operator in SQL query:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_Monthlybonus.**

Emp Id	Emp Name	Emp Salary	Penalty
201	Abhay	25000	200
202	Sumit	30000	500

- Suppose we want to subtract 5,000 from the salary of each employee given in the **Employee_details** table. Then, we have to write the following query in the SQL:

```
1. SELECT Emp_Salary - 5000 as Emp_New_Salary FROM Employee_details;
```

In this query, we have performed the SQL subtraction operation on the single column of the given table.

- If we want to subtract the penalty from the salary of each employee, then we have to write the following query in SQL:

```
1. SELECT Emp_Salary - Penalty as Emp_Total_Salary FROM Employee_details;
```

The SQL Multiplication Operator (*)

Multiplication Operator in SQL performs the Multiplication on the numerical data of the database table. In SQL, we can easily multiply the numerical values of two columns of the same table by specifying both the column names as the first and second operand.

Syntax of SQL Multiplication Operator:

```
1. SELECT operand1 * operand2;
```

Let's understand the below example which explains how to execute Multiplication Operator in SQL query:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_Monthlybonus.**

Emp Id	Emp Name	Emp Salary	Penalty
201	Abhay	25000	200
202	Sumit	30000	500

- Suppose, we want to double the salary of each employee given in the **Employee_details** table. Then, we have to write the following query in the SQL:

```
1. SELECT Emp_Salary * 2 as Emp_New_Salary FROM Employee_details;
```

In this query, we have performed the SQL multiplication operation on the single column of the given table.

- If we want to multiply **the Emp_Id** column to **Emp_Salary** column of that employee whose **Emp_Id** is **202,** then we have to write the following query in SQL:

```
1. SELECT Emp_Id * Emp_Salary as Emp_Id * Emp_Salary FROM Employee_details WHERE Emp_Id = 202;
```

In this query, we have multiplied the values of two columns by using the WHERE clause.

SQL Division Operator (/)

The Division Operator in SQL divides the operand on the left side by the operand on the right side.

Syntax of SQL Division Operator:

```
1. SELECT operand1 / operand2;
```

In SQL, we can also divide the numerical values of one column by another column of the same table by specifying both column names as the first and second operand.

We can also perform the division operation on the stored numbers in the column of the SQL table.

Let's understand the below example which explains how to execute Division Operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	25000
202	Sumit	30000

- Suppose, we want to half the salary of each employee given in the Employee_details table. For this operation, we have to write the following query in the SQL:

```
1. SELECT Emp_Salary / 2 as Emp_New_Salary FROM Employee_details;
```

In this query, we have performed the SQL division operation on the single column of the given table.

SQL Modulus Operator (%)

The Modulus Operator in SQL provides the remainder when the operand on the left side is divided by the operand on the right side.

Syntax of SQL Modulus Operator:

```
1. SELECT operand1 % operand2;
```

Let's understand the below example which explains how to execute Modulus Operator in SQL query:

This example consists of a **Division** table, which has three columns **Number, First_operand, and Second_operand.**

Number	First operand	Second operand
1	56	4
2	32	8
3	89	9
4	18	10
5	10	5

- If we want to get the remainder by dividing the numbers of First_operand column by the numbers of Second_operand column, then we have to write the following query in SQL:

```
1. SELECT First_operand % Second_operand as Remainder FROM Employee_details;
```

SQL Comparison Operators

The **Comparison Operators** in SQL compare two different data of SQL table and check whether they are the same, greater, and lesser. The SQL comparison operators are used with the WHERE clause in the SQL queries

Following are the various comparison operators which are performed on the data stored in the SQL database tables:

1. SQL Equal Operator (=)
2. SQL Not Equal Operator (!=)
3. SQL Greater Than Operator (>)
4. SQL Greater Than Equals to Operator (>=)
5. SQL Less Than Operator (<)\
6. SQL Less Than Equals to Operator (<=)

SQL Equal Operator (=)

This operator is highly used in SQL queries. The **Equal Operator** in SQL shows only data that matches the specified value in the query.

This operator returns TRUE records from the database table if the value of both operands specified in the query is matched.

Let's understand the below example which explains how to execute Equal Operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	30000
202	Ankit	40000
203	Bheem	30000
204	Ram	29000
205	Sumit	30000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose salary is 30000. Then, we have to write the following query in the SQL database:

```
1. 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.

SQL Equal Not Operator (!=)

The **Equal Not Operator** in SQL shows only those data that do not match the query's specified value.

This operator returns those records or rows from the database views and tables if the value of both operands specified in the query is not matched with each other.

Let's understand the below example which explains how to execute Equal Not Operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	45000
202	Ankit	45000
203	Bheem	30000
204	Ram	29000
205	Sumit	29000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose salary is not 45000. Then, we have to write the following query in the SQL database:

```
1. 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.

SQL Greater Than Operator (>)

The **Greater Than Operator** in SQL shows only those data which are greater than the value of the right-hand operand.

Let's understand the below example which explains how to execute Greater ThanOperator (>) in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	45000
202	Ankit	45000

203	Bheem	30000
204	Ram	29000
205	Sumit	29000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose employee id is greater than 202. Then, we have to write the following query in the SQL database:

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

Here, SQL greater than operator displays the records of those employees from the above table whose Employee Id is greater than 202.

SQL Greater Than Equals to Operator (>=)

The **Greater Than Equals to Operator** in SQL shows those data from the table which are greater than and equal to the value of the right-hand operand.

Let's understand the below example which explains how to execute greater than equals to the operator (>=) in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	45000
202	Ankit	45000
203	Bheem	30000
204	Ram	29000
205	Sumit	29000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose employee id is greater than and equals to 202. For this, we have to write the following query in the SQL database:

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

Here,**'SQL greater than equals to operator'** with WHERE clause displays the rows of those employees from the table whose Employee Id is greater than and equals to 202.

SQL Less Than Operator (<)

The **Less Than Operator** in SQL shows only those data from the database tables which are less than the value of the right-side operand.

This comparison operator checks that the left side operand is lesser than the right side operand. If the condition becomes true, then this operator in SQL displays the data which is less than the value of the right-side operand.

Let's understand the below example which explains how to execute less than operator (<) in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary
201	Abhay	45000
202	Ankit	45000
203	Bheem	30000
204	Ram	29000
205	Sumit	29000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose employee id is less than 204. For this, we have to write the following query in the SQL database:

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

Here,**SQL less than operator** with WHERE clause displays the records of those employees from the above table whose Employee Id is less than 204.

SQL Less Than Equals to Operator (<=)

The **Less Than Equals to Operator** in SQL shows those data from the table which are lesser and equal to the value of the right-side operand.

This comparison operator checks that the left side operand is lesser and equal to the right side operand.

Let's understand the below example which explains how to execute less than equals to the operator (<=) in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, and Emp_Salary.**

Emp Id	Emp Name	Emp Salary

201	Abhay	45000
202	Ankit	45000
203	Bheem	30000
204	Ram	29000
205	Sumit	29000

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose employee id is less and equals **203.** For this, we have to write the following query in the SQL database:

```
1. SELECT * FROM Employee_details WHERE Emp_Id <= 203;
```

Here, SQL **less than equals to the operator** with WHERE clause displays the rows of those employees from the table whose Employee Id is less than and equals 202.

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.

Following are the various logical operators which are performed on the data stored in the SQL database tables:

1. SQL ALL operator
2. SQL AND operator
3. SQL OR operator
4. SQL BETWEEN operator
5. SQL IN operator
6. SQL NOT operator
7. SQL ANY operator
8. SQL LIKE operator

SQL ALL Operator

The ALL operator in SQL compares the specified value to all the values of a column from the sub-query in the SQL database.

This operator is always used with the following statement:

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

Syntax of ALL operator:

1. SELECT column_Name1, ..., column_NameN FROM table_Name WHERE column Comparison_operator ALL (SELECT column FROM tablename2)

Let's understand the below example which explains how to execute ALL logical operators in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Abhay	25000	Gurgaon
202	Ankit	45000	Delhi
203	Bheem	30000	Jaipur
204	Ram	29000	Mumbai
205	Sumit	40000	Kolkata

- If we want to access the employee id and 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 the following query in SQL.

1. 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

The **AND operator** in SQL would show the record from the database table if all the conditions separated by the AND operator evaluated to True. It is also known as the conjunctive operator and is used with the WHERE clause.

Syntax of AND operator:

1. SELECT column1, ..., columnN FROM table_Name WHERE condition1 AND condition2 AND condition3 AND ..... AND conditionN;

Let's understand the below example which explains how to execute AND logical operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City

201	Abhay	25000	Delhi
202	Ankit	45000	Chandigarh
203	Bheem	30000	Delhi
204	Ram	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to access all the records of those employees from the **Employee_details** table whose salary is 25000 and the city is Delhi. For this, we have to write the following query in SQL:

```
1. SELECT * FROM Employee_details WHERE Emp_Salary = 25000 OR 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 separated by the OR operator evaluates to True. It is also known as the conjunctive operator and is used with the WHERE clause.

Syntax of OR operator:

```
1. SELECT column1, ...., columnN FROM table_Name WHERE condition1 OR condition2 OR condition3 OR ..... OR conditionN;
```

Let's understand the below example which explains how to execute OR logical operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Abhay	25000	Delhi
202	Ankit	45000	Chandigarh
203	Bheem	30000	Delhi
204	Ram	25000	Delhi
205	Sumit	40000	Kolkata

- If we want to access all the records of those employees from the **Employee_details** table whose salary is 25000 or the city is Delhi. For this, we have to write the following query in SQL:

1. SELECT * FROM Employee_details WHERE Emp_Salary = 25000 OR Emp_City = 'Delhi';

Here, **SQL OR operator** with WHERE clause shows the record of employees whose salary is 25000 or the city is Delhi.

SQL BETWEEN Operator

The **BETWEEN operator** in SQL shows the record within the range mentioned in the SQL query. This operator operates on the numbers, characters, and date/time operands.

If there is no value in the given range, then this operator shows NULL value.

Syntax of BETWEEN operator:

1. SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WHERE column_nameBETWEEN value1 and value2;

Let's understand the below example which explains how to execute BETWEEN logical operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Abhay	25000	Delhi
202	Ankit	45000	Chandigarh
203	Bheem	30000	Delhi
204	Ram	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to access all the information of those employees from the **Employee_details** table who is having salaries between 20000 and 40000. For this, we have to write the following query in SQL:

1. SELECT * FROM Employee_details WHERE Emp_Salary BETWEEN 30000 AND 45000;

Here, we used the **SQL BETWEEN operator** with the Emp_Salary field.

SQL IN Operator

The **IN operator** in SQL allows database users to specify two or more values in a WHERE clause. This logical operator minimizes the requirement of multiple OR conditions.

This operator makes the query easier to learn and understand. This operator returns those rows whose values match with any value of the given list.

Syntax of IN operator:

1. SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WHERE column_nam e IN (list_of_values);

Let's understand the below example which explains how to execute IN logical operator in SQL query:

This example consists of an **Employee_details** table, which has three columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Abhay	25000	Delhi
202	Ankit	45000	Chandigarh
203	Bheem	30000	Delhi
204	Ram	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to show all the information of those employees from the **Employee_details** table whose **Employee Id** is 202, 204, and 205. For this, we have to write the following query in SQL:

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

Here, we used the **SQL IN operator** with the Emp_Id column.

- Suppose, we want to show all the information of those employees from the **Employee_details** table whose **Employee Id** is not equal to 202 and 205. For this, we have to write the following query in SQL:

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

Here, we used the **SQL NOT IN operator** with the Emp_Id column.

SQL NOT Operator

The **NOT operator** in SQL shows the record from the table if the condition evaluates to false. It is always used with the WHERE clause.

Syntax of NOT operator:

1. SELECT column1, column2 ...., columnN FROM table_Name WHERE NOT condition;

Let's understand the below example which explains how to execute NOT logical operator in SQL query:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name,**

Emp_Salary, and Emp_City.

Emp Id	Emp Name	Emp Salary	Emp City
201	Abhay	25000	Delhi
202	Ankit	45000	Chandigarh
203	Bheem	30000	Delhi
204	Ram	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to show all the information of those employees from the **Employee_details** table whose Cityis not Delhi. For this, we have to write the following query in SQL:

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

In this example, we used the **SQL NOT operator** with the Emp_City column.

- Suppose, we want to show all the information of those employees from the **Employee_details** table whose Cityis not Delhi and Chandigarh. For this, we have to write the following query in SQL:

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

In this example, we used the **SQL NOT operator** with the Emp_City column.

SQL ANY Operator

The **ANY operator** in SQL 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 comparison operator.

Syntax of ANY operator:

```
1. SELECT column1, column2 ...., columnN FROM table_Name WHERE column_name comparison_operat
   or ANY ( SELECT column_name FROM table_name WHERE condition(s)) ;
```

SQL LIKE Operator

The **LIKE operator** in SQL shows those records 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:

1. SELECT statement
2. UPDATE statement

3. DELETE statement

Syntax of LIKE operator:

1. SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WHERE colum n_name LIKE pattern;

Let's understand the below example which explains how to execute LIKE logical operator in SQL query:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Chandigarh
203	Saket	30000	Delhi
204	Abhay	25000	Delhi
205	Sumit	40000	Kolkata

- If we want to show all the information of those employees from the **Employee_details** whose name starts with ''s''. For this, we have to write the following query in SQL:

1. SELECT * FROM Employee_details WHERE Emp_Name LIKE 's%' ;

In this example, we used the SQL LIKE operator with **Emp_Name** column because we want to access the record of those employees whose name starts with s.

- If we want to show all the information of those employees from the **Employee_details**whose name ends with ''y''. For this, we have to write the following query in SQL:

1. SELECT * FROM Employee_details WHERE Emp_Name LIKE '%y' ;

- If we want to show all the information of those employees from the **Employee_details**whose name starts with ''S'' and ends with ''y''. For this, we have to write the following query in SQL:

1. SELECT * FROM Employee_details WHERE Emp_Name LIKE 'S%y' ;

SQL Set Operators

The **Set Operators** in SQL 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 set operators which are performed on the similar data stored in the two SQL database tables:

1. SQL Union Operator
2. SQL Union ALL Operator
3. SQL Intersect Operator
4. SQL Minus Operator

SQL Union Operator

The SQL Union Operator combines the result of two or more SELECT statements and provides the single output.

The data type and the number 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 of UNION Set operator:

1. SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
2. UNION
3. SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];

Let's understand the below example which explains how to execute Union operator in Structured Query Language:

In this example, **we used two tables.** Both tables have four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Delhi
203	Saket	30000	Aligarh

Table: Employee_details1

Emp Id	Emp Name	Emp Salary	Emp City
203	Saket	30000	Aligarh
204	Saurabh	40000	Delhi
205	Ram	30000	Kerala
201	Sanjay	25000	Delhi

Table: Employee_details2

- Suppose, we want to see the employee name and employee id of each employee from both tables in a single output. For this, we have to write the following query in SQL:

```
1. SELECT Emp_ID, Emp_Name FROM Employee_details1
2. UNION
3. SELECT Emp_ID, Emp_Name FROM Employee_details2 ;
```

SQL Union ALL Operator

The SQL Union Operator is the same as the UNION operator, but the only difference is that it also shows the same record.

Syntax of UNION ALL Set operator:

```
1. SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
2. UNION ALL
3. SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];
```

Let's understand the below example which explains how to execute Union ALL operator in Structured Query Language:

In this example, **we used two tables.** Both tables have four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Delhi
203	Saket	30000	Aligarh

Table: Employee_details1

Emp Id	Emp Name	Emp Salary	Emp City
203	Saket	30000	Aligarh
204	Saurabh	40000	Delhi
205	Ram	30000	Kerala
201	Sanjay	25000	Delhi

Table: Employee_details2

- If we want to see the employee name of each employee of both tables in a single output. For this, we have to write the following query in SQL:

```
1. SELECT Emp_Name FROM Employee_details1
2. UNION ALL
3. SELECT Emp_Name FROM Employee_details2 ;
```

SQL Intersect Operator

The SQL Intersect Operator shows the common record from two or more SELECT statements. The data type and the number of columns must be the same for each SELECT statement used with the INTERSECT operator.

Syntax of INTERSECT Set operator:

```
1. SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
2. INTERSECT
3. SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];
```

Let's understand the below example which explains how to execute INTERSECT operator in Structured Query Language:

In this example, **we used two tables.** Both tables have four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Delhi
203	Saket	30000	Aligarh

Table: Employee_details1

Emp Id	Emp Name	Emp Salary	Emp City
203	Saket	30000	Aligarh
204	Saurabh	40000	Delhi
205	Ram	30000	Kerala
201	Sanjay	25000	Delhi

Table: Employee_details2

Suppose, we want to see a common record of the employee from both the tables in a single output. For this, we have to write the following query in SQL:

1. SELECT Emp_Name FROM Employee_details1
2. INTERSECT
3. SELECT Emp_Name FROM Employee_details2 ;

SQL Minus Operator

The SQL Minus Operator combines the result of two or more SELECT statements and shows only the results from the first data set.

Syntax of MINUS operator:

1. SELECT column1, column2 ...., columnN FROM First_tablename [WHERE conditions]
2. MINUS
3. SELECT column1, column2 ...., columnN FROM Second_tablename [WHERE conditions];

Let's understand the below example which explains how to execute INTERSECT operator in Structured Query Language:

In this example, **we used two tables.** Both tables have four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Delhi
203	Saket	30000	Aligarh

Table: Employee_details1

Emp Id	Emp Name	Emp Salary	Emp City
203	Saket	30000	Aligarh
204	Saurabh	40000	Delhi
205	Ram	30000	Kerala
201	Sanjay	25000	Delhi

Table: Employee_details2

Suppose, we want to see the name of employees from the first result set after the combination of both tables. For this, we have to write the following query in SQL:

```
1. SELECT Emp_Name FROM Employee_details1
2. MINUS
3. SELECT Emp_Name FROM Employee_details2 ;
```

SQL Unary Operators

The **Unary Operators** in SQL perform the unary operations on the single data of the 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.

Following are the various unary operators which are performed on the numeric data stored in the SQL table:

1. SQL Unary Positive Operator
2. SQL Unary Negative Operator
3. SQL Unary Bitwise NOT Operator

SQL Unary Positive Operator

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

Syntax of Unary Positive Operator

```
1. SELECT +(column1), +(column2) ...., +(columnN) FROM table_Name [WHERE conditions] ;
```

Let's understand the below example which explains how to execute a Positive unary operator on the data of SQL table:

This example consists of an**Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Chandigarh
203	Saket	30000	Delhi
204	Abhay	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to see the salary of each employee as positive from the Employee_details table. For this, we have to write the following query in SQL:

```
1. SELECT  +Emp_Salary Employee_details ;
```

SQL Unary Negative Operator

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

Syntax of Unary Negative Operator

```
1. SELECT -(column_Name1), -(column_Name2) ...., -
   (column_NameN) FROM table_Name [WHERE conditions] ;
```

Let's understand the below example which explains how to execute Negative unary operator on the data of SQL table:

This example consists of an **Employee_details** table, which has four columns **Emp_Id, Emp_Name, Emp_Salary, and Emp_City.**

Emp Id	Emp Name	Emp Salary	Emp City
201	Sanjay	25000	Delhi
202	Ajay	45000	Chandigarh
203	Saket	30000	Delhi
204	Abhay	25000	Delhi
205	Sumit	40000	Kolkata

- Suppose, we want to see the salary of each employee as negative from the Employee_details table. For this, we have to write the following query in SQL:

```
1. SELECT -Emp_Salary Employee_details ;
```

- Suppose, we want to see the salary of those employees as negative whose city is Kolkatain the Employee_details table. For this, we have to write the following query in SQL:

```
1. SELECT -Emp_Salary Employee_details WHERE Emp_City = 'Kolkata';
```

SQL Bitwise NOT Operator

The SQL Bitwise NOT operator provides the one's complement of the single numeric operand. This operator turns each bit of numeric value. If the bit of any numerical value is 001100, then this operator turns these bits into 110011.

Syntax of Bitwise NOT Operator

```
1. SELECT ~(column1), ~(column2) ...., ~(columnN) FROM table_Name [WHERE conditions] ;
```

Let's understand the below example which explains how to execute the Bitwise NOT operator on the data of SQL table:

This example consists of a**Student_details** table, which has four columns **Roll_No, Stu_Name, Stu_Marks, and Stu_City.**

Emp Id	Stu Name	Stu Marks	Stu City
101	Sanjay	85	Delhi
102	Ajay	97	Chandigarh
103	Saket	45	Delhi
104	Abhay	68	Delhi
105	Sumit	60	Kolkata

If we want to perform the Bitwise Not operator on the marks column of **Student_details,** we have to write the following query in SQL:

```
1. SELECT ~Stu_Marks Employee_details ;
```

SQL Bitwise Operators

The **Bitwise Operators** in SQL perform the bit operations on the Integer values. To understand the performance of Bitwise operators, you just knew the basics of Boolean algebra.

Following are the two important logical operators which are performed on the data stored in the SQL database tables:

1. Bitwise AND (&)
2. Bitwise OR(|)

Bitwise AND (&)

The Bitwise AND operator performs the logical AND operation on the given Integer values. This operator checks each bit of a value with the corresponding bit of another value.

Syntax of Bitwise AND Operator

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

Let's understand the below example which explains how to execute Bitwise AND operator on the data of SQL table:

This example consists of the following table, which has two columns. Each column holds numerical values. When we use the Bitwise AND operator in SQL, then SQL converts the values 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.

Column1	Column2
1	1
2	5
3	4
4	2
5	3

- Suppose, we want to perform the Bitwise AND operator between both the columns of the above table. For this, we have to write the following query in SQL:

```
1. SELECT Column1 & Column2 From TABLE_AND ;
```

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 of Bitwise OR Operator

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

Let's understand the below example which explains how to execute Bitwise OR operator on the data of SQL table:

This example consists of a table that has two columns. Each column holds numerical values.

When we used the Bitwise OR operator in SQL, then SQL converts the values of both columns in binary format, and the OR operation is performed on the binary bits. After that, SQL converts the resultant binary bits into user understandable format, i.e., decimal format.

Column1	Column2
1	1
2	5
3	4
4	2

5	3

- Suppose, we want to perform the Bitwise OR operator between both the columns of the above table. For this, we have to write the following query in SQL:

```
1. SELECT Column1 | Column2 From TABLE_OR ;
```