SQL Tutorial

SQL tutorial provides basic and advanced concepts of SQL. Our SQL tutorial is designed for both beginners and professionals.

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

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 the database. You are required to install any database management system in your systems, for example, Oracle, MySQL, MongoDB, PostgreSQL, SQL Server, DB2, etc.

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:

- o The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.

- o SQL allows the data professionals and users to retrieve the data from the relational database management systems.

- o It also helps them to describe the structured data.

- o It allows SQL users to create, drop, and manipulate the database and its tables.

- o It also helps in creating the view, stored procedure, and functions in the relational database.

- o It allows you to define the data and modify that stored data in the relational database.

- o 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:

- o Query Dispatcher
- o Optimization Engines
- o Classic Query Engine
- o 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:
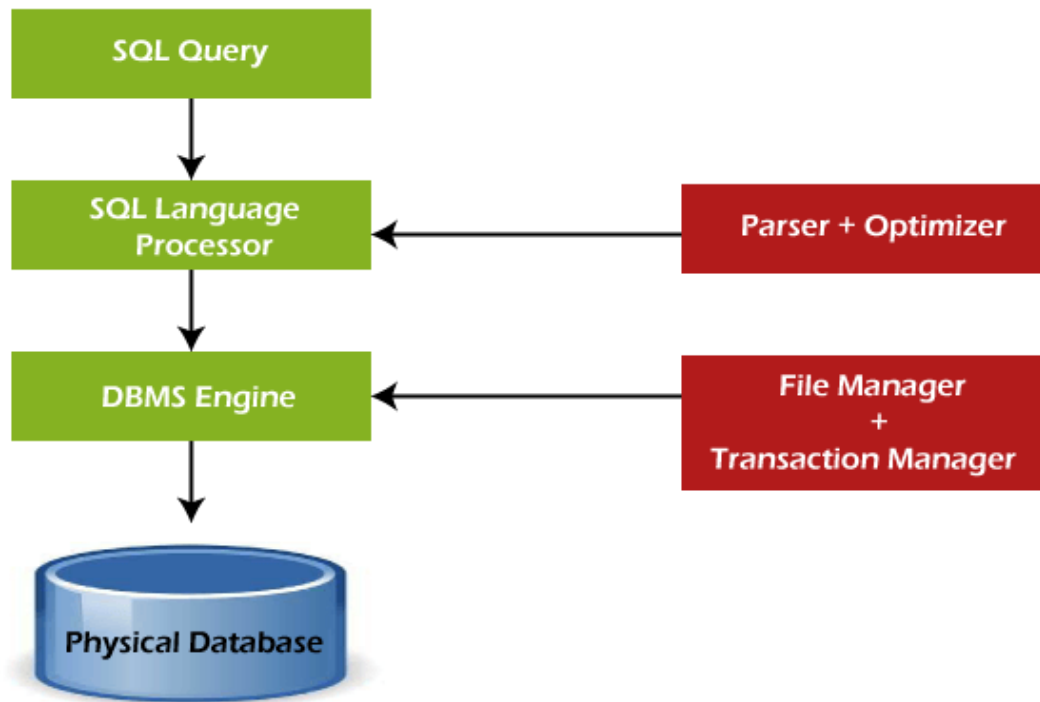
Some SQL Commands

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:

- o 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.
- o SQL statements or syntax are dependent on text lines. We can place a single SQL statement on one or multiple text lines.
- o You can perform most of the action in a database with SQL statements.
- o 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_value_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. .....
6. .....,
7. column_nameN data_type [columnN **constraint**(s)],
8. **PRIMARY KEY**(one or more col)
9.);

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, Sidhu, 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:

1. USE Company;

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.

- o   String Data types
- o   Numeric Data types

    o   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

19

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 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

- o 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.

- o 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 Employe e_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

- o 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.

- o 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;

SQL Multiplication Operator (*)

The 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

- o 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.

- o 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

- o   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

- o 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

- o 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

- o 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

- o 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

- o 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

- o 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

- o 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 |

- o 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

- o 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

- o 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_Nam e 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

      o 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 4 5000;

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_Nam e WHERE column_name 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

- o 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.

- o 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);
2.

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 |

- o 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.

- o 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_operator 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 column_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

- o 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.

o 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' ;

o 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 condition s]
2. UNION ALL
3. SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE condition s];

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

- o 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

o 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

- o 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 ;

- o 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 c onditions] ;

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 con ditions] ;

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

o   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

o 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 ;

SQL Create Database

In SQL, the 'Create Database' statement is a first step for storing the structured data in the database.

The database developers and the users use this statement in SQL for creating the new database in the database systems. It creates the database with the name which has been specified in the Create Database statement.

Syntax of Create Database statement in SQL

1. **CREATE DATABASE** Database_Name;

In this syntax, **Database_Name** specifies the name of the database which we want to create in the system. We have to type the database name in query just after the 'Create Database' keyword.

Following are the most important points which are required to learn while creating a database:

- o The database we want to create should be a simple and unique name, which can be easily identified.
- o Database name should be no more than 128 characters.

Syntax of Create Database statement in MySQL

The same command is used in MySQL to create the new database for storing the structured data.

1. **CREATE DATABASE** Database_Name;

Syntax of Create Database in Oracle

There is no need to create the database in Oracle systems. In the Oracle database, we can directly create the database tables.

Examples of Create Database statement in SQL

In this article, we took the following two examples which will help how to run and perform the Create Database query in SQL:

Example 1:

This example creates the **Student** database. To create the Student database, you have to type the following command in Structured Query Language:

1. **CREATE DATABASE** Student ;

When this query is executed successfully, then it will show the following output:

Database created successfully

You can also verify that your database is created in SQL or not by using the following query:

1. SHOW **DATABASE** ;

SQL does not allow developers to create the database with the existing database name. Suppose if you want to create another Student database in the same database system, then the Create Database statement will show the following error in the output:

```
Can't create database 'Student'; database exists
```

So, firstly you have to delete the existing database by using the Drop Statement. You can also replace the existing database with the help of Replace keyword.

If you want to replace the existing Student database, then you have to type the following SQL query:

1. **CREATE** OR REPLACE **DATABASE** Student ;

Example 2:

Suppose, we want to create the Employee database in the system.

Firstly, we have to type the following command in Structured Query Language:

1. **CREATE DATABASE** Employee ;

When this query is executed successfully, then it will show the following output:

Database created successfully

You can also check that your database is created in SQL by typing the following query:

1. SHOW **DATABASE** ;

We know that SQL does not allow developers to create the database with the existing database name.

Suppose, we want to create another Employee database in the same database system, firstly, we have to delete the existing database using a drop statement, or we have to replace the existing Employee database with the help of the 'replace' keyword.

To replace the existing Employee database with a new Employee database, we have to type the following query in SQL:

1. **CREATE** OR REPLACE **DATABASE** Employee;

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 in SQL.

Following are the most important points which are required to learn before removing the database from the database system:

- o   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.

- o   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:

1. Cannot **drop database** "name_of_the_database" because it **is** currently in use.

Syntax of Drop Database Statement in SQL

1. **DROP DATABASE** Database_Name;

In this SQL syntax, we have to specify the name of that database which we want to delete permanently from the database system. We have to write the name of the database after the DROP DATABASE keyword in every example.

We can also delete multiple databases easily by using the single DROP syntax:

1. **DROP DATABASE** Database_Name1, [Database_Name2, ..., Database_Name N] ;

Using this statement, we have no need to write multiple statements for deleting multiple databases. We can specify all the databases by using a comma in a single statement, as shown in the above syntax.

Examples of Drop Database Statement in SQL

In this article, we took the following two examples that will help how to run and perform the Drop Database query in SQL:

Example1:

Suppose, we want to delete the Student database with all its data from the database system so, firstly we have to check that the Student database exists in the system or not by using the following statement:

1. SHOW DATABASES ;

If the Student database is shown in the output, then we have to type the following query in SQL for removing the Student database:

1. **DROP DATABASE** Student;

If the Student database does not exist in the database system and we run the above query in SQL, then the query will show the following output:

```
Can't drop database 'Student'; database doesn't exist
```

Example2:

Suppose, we want to delete the College database with all its tables and views from the database system, firstly we have to check that if the College database exists in the system or not by using the following statement:

1. SHOW DATABASES;

If the College database is shown in the output, then you have to type the following query in SQL for removing the College database permanently:

1. **DROP DATABASE** College;

If the College database does not exist in the database system, and we run the above query in SQL, then this query will show the following output:

```
Can't drop database 'College'; database doesn't exist
```

SQL RENAME Database

In some situations, database users and administrators want to change the name of the database for some technical reasons. So, the **Rename Database** statement in SQL is used to change the name of the existing database.

Sometimes, the Rename Database statement is used because the developers think that the original name is not more relevant to the data of the database, or they want to give a temporary name to that database.

Syntax of Rename Database in SQL

1. **ALTER DATABASE** old_database_name **MODIFY NAME** = new_database_name;

1. **EXEC** sp_renamedb'old_database_name' , 'new_database_name'

Syntax of Rename Database in MySQL

1. RENAME **DATABASE** old_database_name **TO** new_database_name;

This syntax is used when we want to change the name of the database in MySQL.

Examples of Rename Database in SQL

In this article, we have taken the following two examples which will help you how to run and perform the Rename Database query in SQL:

Example 1:

Suppose we want to rename the Student Database. For this, we have to type the following query in SQL:

1. **ALTER DATABASE** Student **MODIFY NAME** = College ;

This query will change the name of the database from Student to College. To run this query, we must ensure that the database Student exists in the current database server. If not, then it will show an error in the output.

Example 2:

Suppose we want to rename the Department Database. For this, we have to type the following query in SQL:

1. **ALTER DATABASE** Department **MODIFY NAME** = Company ;

This query changes the name of the database from Department to Company. To run this query, we must ensure that the database Department exists in the current database server. If not, then it will show an error in the output.

SQL SELECT Database

Suppose database users and administrators want to perform some operations on tables, views, and indexes on the specific existing database in SQL. Firstly, they have to select the database on which they want to run the database queries.

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

Syntax of USE statement in SQL

1. USE database_name;

In this syntax, we have to define the name of the database after the **USE** keyword and the name of the database must be unique.

Syntax of USE statement in MySQL

1. USE database_name;

Syntax of USE statement in Oracle

There is no need to select the database in Oracle.

Examples of USE statement in SQL

In this article, we have taken the following three examples which will help you how to run and perform USE statement in SQL:

Example 1: Suppose, you want to work with the **Hospital** database. For this firstly, you have to check that if the Hospital database exists on the current database server or not by using the following query:

1. SHOW DATABASES;

If the Hospital database is shown in the output, then you have to execute the following query to select the Hospital database:

1. USE Hospital;

Example 2: Suppose, you want to work with another College database in SQL. For this firstly, you have to check that the College database exists on the current database server or not by using the following query:

1.   SHOW DATABASES;

If the College database is shown in the result, then you have to execute the following query to select the College database:

1.   USE College;

Example 3: Suppose you want to work with another School database in SQL. For this firstly, you have to check that the School database exists on the current database server or not by using the following query:

1.   SHOW DATABASES;

If the School database is shown in the result, then you have to execute the following query to select the School database:

1.   USE School;

SQL Table

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

Note: A table has a specified number of columns, but can have any number of rows.

Table is the simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of an employee table:

Employee		
EMP_NAME	**ADDRESS**	**SALARY**
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

In the above table, "Employee" is the table name, "EMP_NAME", "ADDRESS" and "SALARY" are the column names. The combination of data of multiple columns forms a row e.g. "Ankit", "Lucknow" and 15000 are the data of one row.

SQL TABLE Variable

The **SQL Table variable** is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.

It was introduced with SQL server 2000 to be an alternative of temporary tables.

It is a variable where we temporary store records and results. This is same like temp table but in the case of temp table we need to explicitly drop it.

Table variables are used to store a set of records. So declaration syntax generally looks like CREATE TABLE syntax.

1. **create table** "tablename"
2. ("column1" "data type",
3. "column2" "data type",
4. ...
5. "columnN" "data type");

When a transaction rolled back the data associated with table variable is not rolled back.

A table variable generally uses lesser resources than a temporary variable.

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

Topics of SQL TABLE Statement

SQL TABLE Variable

What TABLE variable can do?

SQL CREATE TABLE

How to create a table using SQL query>

SQL DROP TABLE

How to drop a table?

SQL DELETE TABLE

How to delete all the records of a table?

SQL RENAME TABLE

How to rename a table?

SQL TRUNCATE TABLE

How to truncate a table?

SQL COPY TABLE

How to copy a table?

SQL TEMP TABLE

What is temporary table? What are the advantage of temporary table?

SQL ALTER TABLE

How to add, modify, rename and drop column.

SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

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

Let's see the simple syntax to create the table.

1. **create table** "tablename"
2. ("column1" "data type",
3. "column2" "data type",
4. "column3" "data type",
5. ...
6. "columnN" "data type");

The data type of the columns may vary from one database to another. For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

Let us take an example to create a STUDENTS table with ID as primary key and NOT NULL are the constraint showing that these fields cannot be NULL while creating records in the table.

1. SQL> **CREATE TABLE** STUDENTS (
2. ID **INT**   NOT NULL,
3. **NAME VARCHAR** (20) NOT NULL,
4. AGE **INT**   NOT NULL,
5. ADDRESS **CHAR** (25),
6. **PRIMARY KEY** (ID)
7.);

You can verify it, if you have created the table successfully by looking at the message displayed by the SQL Server, else you can use DESC command as follows:

SQL> DESC STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			
AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

4 rows in set (0.00 sec)

Now you have the STUDENTS table available in your database and you can use to store required information related to students.

SQL CREATE TABLE Example in MySQL

Let's see the command to create a table in MySQL database.

1. **CREATE TABLE** Employee
2. (

3. EmployeeID **int**,
4. FirstName **varchar**(255),
5. LastName **varchar**(255),
6. Email **varchar**(255),
7. AddressLine **varchar**(255),
8. City **varchar**(255)
9.);

SQL CREATE TABLE Example in Oracle

Let's see the command to create a table in Oracle database.

1. **CREATE TABLE** Employee
2. (
3. EmployeeID number(10),
4. FirstName varchar2(255),
5. LastName varchar2(255),
6. Email varchar2(255),
7. AddressLine varchar2(255),
8. City varchar2(255)
9.);

SQL CREATE TABLE Example in Microsoft SQLServer

Let's see the command to create a table in SQLServer database. It is same as MySQL and Oracle.

1. **CREATE TABLE** Employee
2. (
3. EmployeeID **int**,
4. FirstName **varchar**(255),
5. LastName **varchar**(255),
6. Email **varchar**(255),
7. AddressLine **varchar**(255),
8. City **varchar**(255)
9.);

Create a Table using another table

We can create a copy of an existing table using the create table command. The new table 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.

The basic syntax for creating a table with the other table is:

1. **CREATE TABLE** table_name **AS**
2. **SELECT** column1, column2,...
3. **FROM** old_table_name **WHERE** ..... ;
4. The following SQL creates a copy **of** the employee **table**.
5. **CREATE TABLE** EmployeeCopy **AS**
6. **SELECT** EmployeeID, FirstName, Email
7. **FROM** Employee;

SQL Primary Key with CREATE TABLE Statement

The following query creates a PRIMARY KEY on the "D" column when the "Employee" table is created.

MySQL

1. **CREATE TABLE** Employee(
2. EmployeeID NOT NULL,
3. FirstName **varchar**(255) NOT NULL,
4. LastName **varchar**(255),
5. City **varchar**(255),
6. **PRIMARY KEY** (EmployeeID)
7.);

SQL Server / Oracle / MS Access

1. **CREATE TABLE** Employee(
2. EmployeeID NOT NULL **PRIMARY KEY**,
3. FirstName **varchar**(255) NOT NULL,
4. LastName **varchar**(255),
5. City **varchar**(255)

6.);

Use the following query to define a PRIMARY KEY constraints on multiple columns, and to allow naming of a PRIMARY KEY constraints.

For MySQL / SQL Server /Oracle / MS Access

1. **CREATE TABLE** Employee(
2. EmployeeID NOT NULL,
3. FirstName **varchar**(255) NOT NULL,
4. LastName **varchar**(255),
5. City **varchar**(255),
6. **CONSTRAINT**   PK_Employee **PRIMARY KEY** (EmployeeID, FirstName)
7.);

SQL DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

Let's see the syntax to drop the table from the database.

1. **DROP TABLE** "table_name";

Let us take an example:

First we verify STUDENTS table and then we would delete it from the database.

1. SQL> **DESC** STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			
AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

1. 4 rows in set (0.00 sec)

66

This shows that STUDENTS table is available in the database, so we can drop it as follows:

1. SQL>**DROP TABLE** STUDENTS;

Now, use the following command to check whether table exists or not.

1. SQL> **DESC** STUDENTS;

1. Query OK, 0 rows affected (0.01 sec)

As you can see, table is dropped so it doesn't display it.

SQL DROP TABLE Example in MySQL

Let's see the command to drop a table from the MySQL database.

1. **DROP TABLE** table_name;

SQL DROP TABLE Example in Oracle

Let's see the command to drop a table from Oracle database. It is same as MySQL.

1. **DROP TABLE** table_name;

SQL DROP TABLE Example in Microsoft SQLServer

Let's see the command to drop a table from SQLServer database. It is same as MySQL.

1. **DROP TABLE** table_name;

SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM** table_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM** table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

Let's see an "employee" table.

Emp_id	Name	Address	Salary
1	Aryan	Allahabad	22000
2	Shurabhi	Varanasi	13000
3	Pappu	Delhi	24000

Execute the following query to truncate the table:

1. **TRUNCATE TABLE** employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

TUTORIAL HUB   MY SQL

- o   Table structure will be dropped
- o   Relationship will be dropped
- o   Integrity constraints will be dropped
- o   Access privileges will also be dropped

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

SQL RENAME TABLE

In some situations, database administrators and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.

Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

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

Syntax of RENAME statement in SQL

1.   RENAME old_table _name To new_table_name ;

Examples of RENAME statement in SQL

Here, we have taken the following two different SQL examples, which will help you how to change the name of the SQL table in the database using RENAME statement:

Example 1: Let's take an example of a table named **Cars:**

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000

Swift Dezire	Red	7,95,000

Table: Cars

- o Suppose, you want to change the above table name into "Car_2021_Details". For this, you have to type the following RENAME statement in SQL:

1. RENAME Cars To Car_2021_Details ;

- o After this statement, the table "Cars" will be changed into table name "Car_2021_Details".

Example 2: Let's take an example of a table named **Employee:**

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

Table: Employee

- o Suppose, you want to change the name of the above table into the **"Coding_Employees".** For this, you have to type the following RENAME statement in SQL:

1. RENAME Employee To Coding_Employees ;

- o After this statement, the table **"Employee"** will be changed into the table name **"Coding_Employees".**

Syntax of ALTER TABLE statement in SQL

1. ALTER TABLE old_table_name RENAME TO new_table_name;

In the Syntax, we have to specify the RENAME TO keyword after the old name of the table.

Examples of ALTER TABLE statement in SQL

Here, we have taken the following three different SQL examples, which will help you how to change the name of the table in the SQL database using ALTER TABLE statement:

Example 1: Let's take an example of a table named **Bikes:**

Bike_Name	Bike_Color	Bike_Cost
KTM DUKE	Black	185,000
Royal Enfield	Black	NULL
Pulsar	Red	90,0000
Apache	White	NULL
Livo	Black	80,000
KTM RC	Red	195,000

Table : Bikes

- o Suppose, you want to change the name of the above table into "Bikes_Details" using ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Bikes RENAME TO Bikes_Details ;

After this statement, the table "Bikes" will be changed into the table name "Bikes_Details".

Example 2: Let's take an example of a table named **Student:**

Stu_ID	Stu_Name	Stu_Marks
1001	Abhay	85
1002	Ankit	75

1003	Bheem	60
1004	Ram	79
1005	Sumit	80

Table : Student

- o Suppose, you want to change the name of the above table into "MCA_Student_Details" using ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Student RENAME TO MCA_Student_Details ;

After this statement, the table "Student" will be changed into table name "MCA_Student_Details".

Example 3: Let's take an example of a table named **Employee:**

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
202	Ankit	45000	Delhi
203	Bheem	30000	Goa
204	Ram	29000	Goa
205	Sumit	40000	Delhi

Table: Employee

- o Suppose, you want to change the name of the above table into the **"Coding_Employees"** using an ALTER TABLE statement. For this, you have to type the following query in SQL:

1. ALTER TABLE Employee RENAME To Coding_Employees ;

After this statement, the table **"Employee"** will be changed into the table name **"Coding_Employees".**

SQL SELECT Statement

The SELECT statement is the most commonly used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the particular record from the particular column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

Syntax of SELECT Statement in SQL

1. **SELECT** Column_Name_1, Column_Name_2, ..., Column_Name_N **FROM** Table _Name;

In this SELECT syntax, **Column_Name_1, Column_Name_2, ..., Column_Name_N** are the name of those columns in the table whose data we want to read.

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with * asterisk sign:

1. **SELECT** * **FROM** table_name;

Examples of SELECT Statement in SQL

Here, we took the following two different SQL examples which will help you to execute the SELECT statement for retrieving the records:

Example 1:

Firstly, we have to create the new table and then insert some dummy records into it.

Use the following query to create the **Student_Records** table in SQL:

1. **CREATE TABLE** Student_Records
2. (
3. Student_Id **Int PRIMARY KEY**,
4. First_Name **VARCHAR** (20),
5. Address **VARCHAR** (20),
6. Age **Int** NOT NULL,

7. Percentage **Int** NOT NULL,
8. Grade **VARCHAR** (10)
9.) ;

The following query inserts the record of intelligent students into the **Student_Records** table:

1. **INSERT INTO** Student **VALUES** (201, Akash, Delhi, 18, 89, A2),
2. (202, Bhavesh, Kanpur, 19, 93, A1),
3. (203, Yash, Delhi, 20, 89, A2),
4. (204, Bhavna, Delhi, 19, 78, B1),
5. (05, Yatin, Lucknow, 20, 75, B1),
6. (206, Ishika, Ghaziabad, 19, 51, C1),
7. (207, Vivek, Goa, 20, 62, B2);

The following SQL query displays all the values of each column from the above Student_records table:

1. **SELECT** * **FROM** Student_Records;

The output of the above query is:

Student_ID	First_Name	Address	Age	Percentage	Grade
201	Akash	Delhi	18	89	A2
202	Bhavesh	Kanpur	19	93	A1
203	Yash	Delhi	20	89	A2
204	Bhavna	Delhi	19	78	B1
205	Yatin	Lucknow	20	75	B1
206	Ishika	Ghaziabad	19	91	C1
207	Vivek	Goa	20	80	B2

Example 2:

The following query displays the values of particular column from the above **Student_Record** table:

1. **SELECT** Student_Id, Age, Percentage, Grade **FROM** Employee;

Student_ID	Age	Percentage	Grade
201	18	89	A2
202	19	93	A1
203	20	89	A2
204	19	78	B1
205	20	75	B1
206	19	91	C1
207	20	80	B2

SELECT Statement with WHERE clause

The WHERE clause is used with SELECT statement to return only those rows from the table, which satisfy the specified condition in the query.

In SQL, the WHERE clause is not only used with SELECT, but it is also used with other SQL statements such as UPDATE, ALTER, and DELETE statements.

Syntax of SELECT Statement with WHERE clause

1. **SELECT** * **FROM** Name_of_Table **WHERE** [condition];

In the syntax, we specify the condition in the WHERE clause using SQL logical or comparison operators.

Example of SELECT Statement with WHERE clause

Firstly, we have to create the new table and then insert some dummy records into it.

Use the following query to create the **Employee_Details** table in SQL:

1. **CREATE TABLE** Employee_Details
2. (
3. Employee_ID **INT** AUTO_INCREMENT **PRIMARY KEY**,
4. Emp_Name **VARCHAR** (50),

5. Emp_City **VARCHAR** (20),
6. Emp_Salary **INT** NOT NULL,
7. Emp_Panelty **INT** NOT NULL
8.) ;

The following INSERT query inserts the record of employees into the Employee_Details table:

1. **INSERT INTO** Employee_Details (Employee_ID, Emp_Name, Emp_City, Emp_Salary, Emp_Panelty) **VALUES** (101, Anuj, Ghaziabad, 25000, 500),
2. (102, Tushar, Lucknow, 29000, 1000),
3. (103, Vivek, Kolkata, 35000, 500),
4. (104, Shivam, Goa, 22000, 500);

The following SELECT query shows the data of the **Employee_Details** table:

1. **SELECT** * **FROM** Employee_Details;

Employee_Id	Emp_Name	Emp_City	Emp_Salary	Emp_Panelty
101	Anuj	Ghaziabad	25000	500
102	Tushar	Lucknow	29000	1000
103	Vivek	Kolkata	35000	500
104	Shivam	Goa	22000	500

The following query shows the record of those employees from the above table whose Emp_Panelty is 500:

1. **SELECT** * **FROM** Employee_Details **WHERE** Emp_Panelty = 500;

This SELECT query displays the following table in result:

Employee_Id	Emp_Name	Emp_City	Emp_Salary	Emp_Panelty
101	Anuj	Ghaziabad	25000	500
103	Vivek	Kolkata	35000	500

104	Shivam	Goa	22000	500

SQL SELECT Statement with GROUP BY clause

The GROUP BY clause is used with the SELECT statement to show the common data of the column from the table:

Syntax of SELECT Statement with GROUP BY clause

1. **SELECT** column_Name_1, column_Name_2, ..., column_Name_N aggregate_function_name(column_Name2) **FROM** table_name **GROUP BY** column_Name1;

Example of SELECT Statement with GROUP BY clause

Use the following query to create the **Cars_Details** table:

1. **CREATE TABLE** Cars_Details
2. (
3. Car_Number **INT PRIMARY KEY**,
4. Car_Name **VARCHAR** (50),
5. Car_Price **INT** NOT NULL,
6. Car_AmountINT NOT NULL
7.) ;

The following INSERT query inserts the record of cars into the **Cars_Details** table:

1. **INSERT INTO** Cars_Details (Car_Number, Car_Name, Car_Amount, Car_Price)
2. **VALUES** (2578, Creta, 3, 1500000),
3. (9258, Audi, 2, 3000000),
4. (8233, Venue, 6, 900000),
5. (6214, Nexon, 7, 1000000);

The following SELECT query displays the values in the output:

1. **SELECT** * **FROM** Cars_Details;

Car_Number	Car_Name	Car_Amount	Car_Price

2578	Creta	3	1000000
9258	Audi	2	900000
8233	Venue	6	900000
6214	Nexon	7	1000000

The following SELECT with GROUP BY query lists the number of cars of the same price:

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

The output of above GROUP BY query is shown below:

Output:

Count (Car_Name)	Car_Price
2	1000000
2	900000

SQL 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 of SELECT Statement with HAVING clause

1. **SELECT** column_Name_1, column_Name_2, ..., column_Name_N aggregate_function_name(column_Name_2) **FROM** table_name **GROUP BY** column_Name1 **HAVING** ;

Example of SELECT Statement with HAVING clause

Let's create the **Employee_Having** table in SQL using the below CREATE command:

1. **CREATE TABLE** Employee_Having
2. (
3. Employee_Id **INT PRIMARY KEY**,
4. Employee_Name **VARCHAR** (50),

5. Employee_Salary **INT** NOT NULL,
6. Employee_City **VARCHAR** (50)
7.) ;

The following INSERT query inserts the record of employees into the Employee_Having table:

1. **INSERT INTO** Employee_Having (Employee_Id, Employee_Name, Employee_Salary, Employee_City)
2. **VALUES** (201, Jone, 20000, Goa),
3. (202, Basant, 40000, Delhi),
4. (203, Rashet, 80000,Jaipur),
5. (204, Aunj, 20000, Goa),
6. (205, Sumit, 50000, Delhi);

The following SELECT query shows the values of Employee_Having table in the output:

1. **SELECT** * **FROM** Employee_Having;

Employee_Id	Employee_Name	Employee_Salary	Employee_City
201	Jone	20000	Goa
202	Basant	40000	Delhi
203	Rashet	80000	Jaipur
204	Anuj	20000	Goa
205	Sumit	50000	Delhi

The following query shows the total salary of those employees having more than 5000 from the above Employee_Having table:

1. **SELECT** SUM (Employee_Salary), Employee_City **FROM** Employee_Having **GROUP BY** Employee_City **HAVING** SUM(Employee_Salary)>5000;

This HAVING query with SELECT statement shows the following table:

Output:

SUM (Employee_Salary)	Employee_City
90000	Delhi
80000	Jaipur

SELECT Statement with ORDER BY clause

The ORDER BY clause with the SQL SELECT statement shows the records or rows in a sorted manner.

The ORDER BY clause arranges the values in both ascending and descending order. Few database systems arrange the values of column in ascending order by default.

Syntax of SELECT Statement with ORDER BY clause

1. **SELECT** Column_Name_1, Column_Name_2, ..., column_Name_N **FROM** table _name **WHERE** [Condition] **ORDER BY**[column_Name_1, column_Name_2, ..., column_Name_N **asc** | **desc**];

Example of SELECT Statement with ORDER BY clause in SQL

1. **CREATE TABLE** Employee_Order
2. (
3. Id **INT** NOT NULL,
4. FirstName **VARCHAR** (50),
5. Salary **INT**,
6. City **VARCHAR** (50)
7.) ;

The following INSERT query inserts the record of employees into the Employee_Having table:

1. **INSERT INTO** Employee_Order (Id, FirstName, Salary, City)
2. **VALUES** (201, Jone, 20000, Goa),
3. (202, Basant, 15000, Delhi),
4. (203, Rashet, 80000,Jaipur),
5. (204, Aunj, 90000, Goa),
6. (205, Sumit, 50000, Delhi);

The following SELECT query shows the values of the table in the output:

1. **SELECT** * **FROM** Employee_Order;

Id	FirstName	Salary	City
201	Jone	20000	Goa
202	Basant	15000	Delhi
203	Rashet	80000	Jaipur
204	Anuj	90000	Goa
205	Sumit	50000	Delhi

The following query sorts the salary of employees in descending order from the above Employee_Order table:

1. **SELECT** * **FROM** Employee_Order **ORDER BY** Emp_Salary **DESC**;

This SQL query shows the following table in result:

Output:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
204	Anuj	90000	Goa
203	Rashet	80000	Jaipur
205	Sumit	50000	Delhi
201	Jone	20000	Goa
202	Basant	15000	Delhi

SQL SELECT UNIQUE

Actually, there is no difference between DISTINCT and UNIQUE.

SELECT UNIQUE is an old syntax which was used in oracle description but later ANSI standard defines DISTINCT as the official keyword.

After that oracle also added DISTINCT but did not withdraw the service of UNIQUE keyword for the sake of backward compatibility.

In simple words, we can say that SELECT UNIQUE statement is used to retrieve a unique or distinct element from the table.

Let's see the syntax of select unique statement.

1. **SELECT UNIQUE** column_name
2. **FROM** table_name;

SQL SELECT DISTINCT statement can also be used for the same cause.

SQL SELECT DATE

SQL SELECT DATE 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.

For example: let's see the query to get all the records after '2013-12-12'.

1. **SELECT** * **FROM**
2. **table**-**name WHERE** your **date**-**column** >= '2013-12-12'

Let's see the another query to get all the records after '2013-12-12' and before '2013-12-13' date.

1. **SELECT*** **FROM**
2. **table**-**name where** your **date**-**column** < '2013-12-13' and your **date**-**column** >= '2013-12-12'

If you want to compare the dates within the query, you should use BETWEEN operator to compare the dates.

1. **SELECT** * **FROM**
2. table_name **WHERE** yourdate BETWEEN '2012-12-12' and '2013-12-12'

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

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

SQL SELECT NULL

First of all we should know that what null value is? Null values are used to represent missing unknown data.

There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

Note: we should not compare null value with 0. They are not equivalent.

Where SQL is NULL:

How to select records with null values only? (in the marks column)

There is an example of student table:

SIR_NAME	NAME	MARKS
TYAGI	SEEMA	
SINGH	RAMAN	5.5
SHARMA	AMAR	
JAISWAL	VICKY	6.2

Let's see the query to get all the records where marks is NULL:

1. **SELECT** SIR_NAME, **NAME**, MARKS **FROM** STUDENTS
2. **WHERE** MARKS **IS** NULL

It will return the following records:

SIR_NAME	NAME	MARKS
SHARMA	AMAR	
TYAGI	SEEMA	

Where SQL is NOT NULL:

How to select records with no null values(in marks column)? Let's see the query to get all the records where marks is NOT NULL

1. **SELECT** SIR_NAME, FIRSTNAME, MARKS **FROM** STUDENTS
2. **WHERE** MARKS **IS** NOT NULL

SIR_NAME	NAME	MARKS
SINGH	RAMAN	5.5
JAISWAL	VICKY	6.2

SQL WHERE

A **WHERE clause** in SQL is a data manipulation language statement.

WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.

Actually. it filters the records. It returns only those queries which fulfill the specific conditions.

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

Let's see the syntax for sql where:

1. **SELECT** column1, **column** 2, ... **column** n
2. **FROM**   table_name

3. **WHERE** [conditions]

WHERE clause uses some conditional selection

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
< >	not equal to

SQL AND

- o The SQL **AND** condition is used in SQL query to create two or more conditions to be met.
- o It is used in SQL **SELECT, INSERT, UPDATE** and **DELETE**
- o Let's see the syntax for SQL AND:
- o SELECT columns FROM tables WHERE condition 1 AND condition 2;
- o The SQL AND condition require that both conditions should be met.
- o The SQL AND condition also can be used to join multiple tables in a SQL statement.
- o To understand this concept practically, let us see some examples.

Consider we have an employee table created into the database with the following data:

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai

4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

SQL "AND" example with "SELECT" statement

This is how an SQL "**AND**" condition can be used in the SQL **SELECT** statement.

Example 1:

Write a query to get the records from emp tables in which department of the employee is IT and location is Chennai.

Query:

1. mysql> **SELECT** ***FROM** emp **WHERE** Department = "IT" AND Location = "Chennai";

ID	First_Name	Last_Name	Department	Location
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai

In the emp table, there are three employees whose department is IT. But we have specified the AND condition according to which the employee's location should not be other than Chennai. So, there are only two employees whose department is IT and Location is Chennai.

Example 2:

Write a query to get the records from emp tables in which department of the employee is IT and location is Mumbai.

Query:

1. mysql> **SELECT** ***FROM** emp **WHERE** Department = "IT" AND Location = "Mumbai";

ID	First_Name	Last_Name	Department	Location
2	Anurag	Rajput	IT	Mumbai

In the emp table, there are three employees whose department is IT. Among these three employees, there is only one employee whose location is Mumbai. Due to the presence of the AND operator used in the query, a record must satisfy both conditions.

SQL "AND" example with "UPDATE" statement

This is how the "AND" condition can be used in the SQL UPDATE statement.

Example 1:

Write a query to update the records in emp tables in which department of the employee is Marketing, and the first name is Suraj. For that particular employee, set the updated value of the location as Delhi.

Query:

1. mysql> **UPDATE** emp **SET** Location = "Delhi" **WHERE** Department = "Marketing" AND First_Name = "Suraj";

```
mysql> UPDATE emp SET Location = "Delhi" WHERE Department = "Marketing" AND First_Name = "Suraj";
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

We will use the SELECT query to verify the updated record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi

| 6 | Roshni | Jadhav | Finance | Bangalore |
| 7 | Sandhya | Jain | Finance | Bangalore |

In the emp table, there are three employees whose department is IT. Among these three employees, there is only one employee whose location is Mumbai. Due to the presence of the AND operator used in the query, a record must satisfy both conditions.

Example 2:

Write a query to update the records in the emp table in which department of the employee is Finance and ID is 7. For that particular employee, set the updated value of the department as HR.

Query:

1. mysql> **UPDATE** emp **SET** Department = "HR" **WHERE** Department = "Finance" AND ID = 7;

```
mysql> UPDATE emp SET Department = "HR" WHERE Department = "Finance" AND ID = 7;
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

We will use the SELECT query to verify the updated record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	HR	Bangalore

In the emp table, there are two employees whose department is Finance. Among these two employees, there is only one employee whose ID is 7. Due to the presence of AND operator used in the query, a record must have the department as Finance and ID as 7.

SQL "AND" example with "DELETE" statement

This is how an SQL "AND" condition can be used in the SQL DELETE statement.

Example 1:

Write a query to delete the records from the emp table in which the last name of the employee is Jain, and the Location is Bangalore.

Query:

1. mysql> **DELETE FROM** emp **WHERE** Last_Name = 'Jain' AND Location = 'Bangalore';

```
mysql> DELETE FROM emp WHERE Last_Name = 'Jain' AND Location = 'Bangalore';
Query OK, 1 row affected (0.09 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore

There is only one record in the emp table whose last name is Jain. But still, due to the presence of AND operator, the second condition will also be checked according to

which employee's location should be Bangalore. So, only that particular record is deleted.

Example 2:

Write a query to delete the records from the emp table in which department of the employee is IT and Location is Mumbai.

Query:

1. mysql> **DELETE FROM** emp **WHERE** Department = 'IT' AND Location = 'Mumbai';

```
mysql> DELETE FROM emp WHERE Department = 'IT' AND Location = 'Mumbai';
Query OK, 1 row affected (0.11 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	Finance	Bangalore

There are three records in the emp table whose department is IT. But only one record is deleted from the emp table, which contains a total of 6 records. This happened because of the AND operator according to which the employee's location should mandatorily be Mumbai. Therefore there is only one record that satisfies both the conditions. Hence, it is deleted.

SQL OR

The SQL **OR** condition is used in SQL query to create a SQL statement where records are returned when any one condition met. It can be used in a **SELECT** statement, **INSERT** statement, **UPDATE** statement or **DELETE** statement.

Let's see the syntax for the OR condition:

1. **SELECT** columns **FROM** tables **WHERE** condition 1 OR condition 2;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

- o **SQL "OR" example with SQL SELECT**

This is how an SQL "OR" condition can be used in the SQL SELECT statement.

Example 1:

Write a query to get the records from emp tables in which department of the employee is IT or location is Chennai.

Query:

1. mysql> **SELECT** ***FROM** emp **WHERE** Department = "IT" OR Location = "Chennai";

ID	First_Name	Last_Name	Department	Location
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai

4	Pranjal	Patil	IT	Chennai

In the emp table, there are three employees whose department is IT. But there are only two records whose location is Chennai. Still, all three records are displayed. This happened because we have specified OR operator in the query, according to which the record will be considered in the result set even any one condition is met.

Example 2:

Write a query to get the records from emp tables in which department of the employee is Marketing or location is Noida.

Query:

1. mysql> **SELECT** *****FROM** emp **WHERE** Department = "Marketing" OR Location = "Noida";

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
5	Suraj	Tripathi	Marketing	Pune
7	Sandhya	Jain	Finance	Bangalore

There are two employees whose department is Marketing in the emp table, but still, three records are displayed. This happened because of the use of the OR operator in the query. Among the three records displayed above, the first two records satisfy condition 1; the second record satisfies both the conditions and the third record satisfies only condition 1. Due to the OR operator, even if anyone condition is satisfied, the record is considered in the result-set.

- o **SQL "OR" example with SQL UPDATE**

This is how the "OR" condition can be used in the SQL UPDATE statement.

Example 1:

Write a query to update the records in emp tables in which department of the employee is Marketing, or the last name is Tarle. For that particular employee, set the updated value of the location as Delhi.

Query:

1. mysql> **UPDATE** emp **SET** Location = "Delhi" **WHERE** Department = "Marketing" OR Last_Name = "Tarle";

```
mysql> UPDATE emp SET Location = "Delhi" WHERE Department = "Marketing" OR Last_Name = "Tarle";
Query OK, 3 rows affected (0.07 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

We will use the SELECT query to verify the updated record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

There are two employees whose department is 'Marketing' and one record whose last name is 'Tarle' in the emp table. Though only one condition is still met, that record is considered and updated in the table due to the OR operator.

Example 2:

Write a query to update the records in the emp table in which department of the employee is Finance, or the first name is Sandhya. For that particular employee, set the updated value of the department as HR.

Query:

1. mysql> **UPDATE** emp **SET** Department = "HR" **WHERE** Department = "Finance" OR First_Name = "Sandhya";

93

```
mysql> UPDATE emp SET Department = "HR" WHERE Department = "Finance" OR First_Name = "Sandhya";
Query OK, 2 rows affected (0.08 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

We will use the SELECT query to verify the updated record.

1. mysql> **SELECT** ***FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Delhi
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Delhi
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Delhi
6	Roshni	Jadhav	HR	Bangalore
7	Sandhya	Jain	HR	Noida

There are two employees whose department is 'Finance,' and among these two records, one record satisfies both the conditions in the emp table. However, both the records are considered and updated in the table due to the OR operator.

○ **SQL "OR" example with SQL DELETE**

This is how an SQL "OR" condition can be used in the SQL DELETE statement.

Example 1:

Write a query to delete the records from the emp table in which the last name of the employee is Jain or Location is Bangalore.

Query:

1. mysql> **DELETE FROM** emp **WHERE** Last_Name = 'Jain' OR Location = 'Banga lore';

```
mysql> DELETE FROM emp WHERE Last_Name = 'Jain' OR Location = 'Bangalore';
Query OK, 2 rows affected (0.09 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT** *****FROM** emp;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune

There is only one record in the emp table whose last name is Jain and one record whose location is Bangalore. But still, due to the presence of an OR operator, even if anyone condition is satisfied, that particular record is deleted.

Example 2:

Write a query to delete the records from the emp table in which department of the employee is marketing and Location is Delhi.

Query:

1. mysql> **DELETE FROM** emp **WHERE** Department = 'Marketing' OR Location = 'Delhi';

```
mysql> DELETE FROM emp WHERE Department = 'Marketing' OR Location = 'Delhi';
Query OK, 3 rows affected (0.07 sec)
```

We will use the SELECT query to verify the deleted record.

1. mysql> **SELECT** *****FROM** emp;

ID	First_Name	Last_Name	Department	Location
2	Anurag	Rajput	IT	Mumbai
4	Pranjal	Patil	IT	Chennai

There is only one record in the emp table whose department is Marketing and one record whose location is Delhi. But still, due to the presence of an OR operator, even if anyone condition is satisfied, that particular record is deleted.

SQL INSERT STATEMENT

SQL INSERT statement is a SQL query. It is used to insert a single or a multiple records in a table.

There are two ways to insert data in a table:

1. By SQL insert into statement
    1. By specifying column names
    2. Without specifying column names
2. By SQL insert into select statement

---

1) Inserting data directly into a table

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

There are two ways to insert values in a table.

In the first method there is no need to specify the column name where the data will be inserted, you need only their values.

1. **INSERT INTO** table_name
2. **VALUES** (value1, value2, value3....);

The second method specifies both the column name and values which you want to insert.

1. **INSERT INTO** table_name (column1, column2, column3....)
2. **VALUES** (value1, value2, value3.....);

Let's take an example of table which has five records within it.

1. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
2. **VALUES** (1, ABHIRAM, 22, ALLAHABAD);
3. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
4. **VALUES** (2, ALKA, 20, GHAZIABAD);
5. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
6. **VALUES** (3, DISHA, 21, VARANASI);
7. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
8. **VALUES** (4, ESHA, 21, DELHI);
9. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
10. **VALUES** (5, MANMEET, 23, JALANDHAR);

It will show the following table as the final result.

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI
4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR

You can create a record in CUSTOMERS table by using this syntax also.

1. **INSERT INTO** CUSTOMERS
2. **VALUES** (6, PRATIK, 24, KANPUR);

The following table will be as follow:

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI

4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR
6	PRATIK	24	KANPUR

2) Inserting data through SELECT Statement

SQL INSERT INTO SELECT Syntax

1. **INSERT INTO** table_name
2. [(column1, column2, ... **column**)]
3. **SELECT** column1, column2, ... **Column** N
4. **FROM** table_name [**WHERE** condition];

SQL INSERT Multiple Rows

Many times developers ask that is it possible to insert multiple rows into a single table in a single statement. Currently, developers have to write multiple insert statements when they insert values in a table. It is not only boring but also time-consuming.

Let us see few practical examples to understand this concept more clearly. We will use the MySQL database for writing all the queries.

Example 1:

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

1. mysql> USE dbs;

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

1. mysql> **CREATE TABLE** student(ID **INT**, **Name VARCHAR**(20), Percentage **INT**, Location **VARCHAR**(20), DateOfBirth **DATE**);

The student table is created successfully.

Now, we will write a single query to insert multiple records in the student table:

1. mysql> **INSERT INTO** student(ID, **Name**, Percentage, Location, DateOfBirth) **V ALUES**(1, "Manthan Koli", 79, "Delhi", "2003-08-20"), (2, "Dev Dixit", 75, "Pune", "1999-06-17"), (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"), (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"), (5, "Rahul Khanna", 92, "Ambala", "1996-03-04"), (6, "Pankaj Deshmukh", 67, "Kanpur", "2000-02-02"), (7, "Gaurav Kumar", 84, "Chandigarh", "1998-07-06"), (8, "Sanket Jain", 61, "Shimla", "1990-09-08"), (9, "Sahil Wagh", 90, "Kolkata", "1968-04-03"), (10, "Saurabh Singh", 54, "Kashmir", "1989-01-06");

To verify that multiple records are inserted in the student table, we will execute the SELECT query.

1. mysql> **SELECT** **FROM* student;

ID	Name	Percentage	Location	DateOfBirth

1	Manthan Koli	79	Delhi	2003-08-20
2	Dev Dixit	75	Pune	1999-06-17
3	Aakash Deshmukh	87	Mumbai	1997-09-12
4	Aaryan Jaiswal	90	Chennai	2005-10-02
5	Rahul Khanna	92	Ambala	1996-03-04
6	Pankaj Deshmukh	67	Kanpur	2000-02-02
7	Gaurav Kumar	84	Chandigarh	1998-07-06
8	Sanket Jain	61	Shimla	1990-09-08
9	Sahil Wagh	90	Kolkata	1968-04-03
10	Saurabh Singh	54	Kashmir	1989-01-06

The results show that all ten records are inserted successfully using a single query.

Example 2:

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

1. mysql> USE dbs;

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

1. mysql> **CREATE TABLE** items_tbl(ID **INT**, Item_Name **VARCHAR**(20), Item_Qu antity **INT**, Item_Price **INT**, Purchase_Date **DATE**);


```
MySQL 5.5 Command Line Client                                    —    □    ×

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE dbs;
Database changed
mysql> CREATE TABLE items_tbl(ID INT, Item_Name VARCHAR(20), Item_Quantity INT, Item_Price INT, Purchase_Date DATE);
Query OK, 0 rows affected (0.48 sec)
```

The table named items_tbl is created successfully.

Now, we will write a single query to insert multiple records in the items_tbl table:

1. mysql> **INSERT INTO** items_tbl(ID, Item_Name, Item_Quantity, Item_Price, Pur chase_Date) **VALUES**(1, "Soap", 5, 200, "2021-07-08"), (2, "Toothpaste", 2, 80, "2021-07-10"), (3, "Pen", 10, 50, "2021-07-12"), (4, "Bottle", 1, 250, "2021-07-13"), (5, "Brush", 3, 90, "2021-07-15"), (6, "Notebooks", 10, 1000, "2021-07-26"), (7, "Handkerchief", 3, 100, "2021-07-28"), (8, "Chips Packet", 5, 50, "2021-07-30"), (9, "Marker", 2, 30, "2021-08-13"), (10, "Scissors", 1, 60, "2021-08-13");

```
mysql> INSERT INTO items_tbl(ID, Item_Name, Item_Quantity, Item_Price, Purchase_Date) VALUES(1, "Soap", 5, 200, "2021-07
-08"), (2, "Toothpaste", 2, 80, "2021-07-10"), (3, "Pen", 10, 50, "2021-07-12"), (4, "Bottle", 1, 250, "2021-07-13"), (5
, "Brush", 3, 90, "2021-07-15"), (6, "Notebooks", 10, 1000, "2021-07-26"), (7, "Handkerchief", 3, 100, "2021-07-28"), (8
, "Chips Packet", 5, 50, "2021-07-30"), (9, "Marker", 2, 30, "2021-08-13"), (10, "Scissors", 1, 60, "2021-08-13");
Query OK, 10 rows affected (0.08 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

To verify that multiple records are inserted in the items_tbl table, we will execute the SELECT query.

1. mysql> **SELECT** ***FROM** items_tbl;

ID	Item_Name	Item_Quantity	Item_Price	Purchase_Date
1	Soap	5	200	2021-07-08
2	Toothpaste	2	80	2021-07-10
3	Pen	10	50	2021-07-12
4	Bottle	1	250	2021-07-13
5	Brush	3	90	2021-07-15
6	Notebooks	10	1000	2021-07-26
7	Handkerchief	3	100	2021-07-28
8	Chips Packet	5	50	2021-07-30
9	Marker	2	30	2021-08-13

10	Scissors	1	60	2021-08-13

The results show that all ten records are inserted successfully using a single query.

SQL UPDATE

The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.

SQL UPDATE statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.

The UPDATE statement can be written in following form:

1. **UPDATE** table_name **SET** [column_name1= value1,... column_nameN = value N] [**WHERE** condition]

Let's see the Syntax:**UPDATE** table_name

1. **SET** column_name = expression
2. **WHERE** conditions

Let's take an example: here we are going to update an entry in the source table.

SQL statement:

1. **UPDATE** students
2. **SET** User_Name = 'beinghuman'
3. **WHERE** Student_Id = '3'

Source Table:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	jonny

See the result after updating value:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	**beinghuman**

Updating Multiple Fields:

If you are going to update multiple fields, you should separate each field assignment with a comma.

SQL UPDATE statement for multiple fields:

1. **UPDATE** students
2. **SET** User_Name = 'beserious', First_Name = 'Johnny'
3. **WHERE** Student_Id = '3'

Result of the table is given below:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	**Johnny**	Walker	**beserious**

MYSQL SYNTAX FOR UPDATING TABLE:

1. **UPDATE** table_name
2. **SET** field1 = new-value1, field2 = new-value2,
3. [**WHERE** CLAUSE]

SQL UPDATE SELECT:

SQL UPDATE WITH SELECT QUERY:

We can use SELECT statement to update records through UPDATE statement.

SYNTAX:

1. **UPDATE** tableDestination
2. **SET** tableDestination.col = value
3. **WHERE** EXISTS (
4. **SELECT** col2.value
5. **FROM** tblSource
6. **WHERE** tblSource.join_col = tblDestination. Join_col
7. AND tblSource.**Constraint** = value)

You can also try this one -

1. **UPDATE**
2. **Table**
3. **SET**
4. **Table**.column1 = othertable.**column** 1,
5. **Table**.column2 = othertable.**column** 2
6. **FROM**
7. **Table**
8. **INNER** JOIN
9. Other_table
10. **ON**
11. **Table**.id = other_table.id

My SQL SYNTAX:

If you want to UPDATE with SELECT in My SQL, you can use this syntax:

Let's take an example having two tables. Here,

First table contains -

Cat_id, cat_name,

And the second table contains -

Rel_cat_id, rel_cat_name

SQL UPDATE COLUMN:

We can update a single or multiple columns in SQL with SQL UPDATE query.

SQL UPDATE EXAMPLE WITH UPDATING SINGLE COLUMN:

1. **UPDATE** students
2. **SET** student_id = 001
3. **WHERE** student_name = 'AJEET';

This SQL UPDATE example would update the student_id to '001' in the student table where student_name is 'AJEET'.

SQL UPDATE EXAMPLE WITH UPDATING MULTIPLE COLUMNS:

To update more than one column with a single update statement:

1. **UPDATE** students
2. **SET** student_name = 'AJEET',
3. Religion = 'HINDU'
4. **WHERE** student_name = 'RAJU';

This SQL UPDATE statement will change the student name to 'AJEET' and religion to 'HINDU' where the student name is 'RAJU'.

SQL UPDATE with JOIN

SQL UPDATE JOIN means we will update one table using another table and join condition.

Let us take an example of a customer table. I have updated customer table that contains latest customer details from another source system. I want to update the customer table with latest data. In such case, I will perform join between target table and source table using join on customer ID.

Let's see the *syntax* of SQL UPDATE query with JOIN statement.

1. **UPDATE** customer_table
2. **INNER** JOIN
3. Customer_table
4. **ON** customer_table.rel_cust_name = customer_table.cust_id
5. **SET** customer_table.rel_cust_name = customer_table.cust_name

How to use multiple tables in SQL UPDATE statement with JOIN

Let's take two tables, table 1 and table 2.

Create table1

1. **CREATE TABLE** table1 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))
2. **INSERT INTO** table1 (col1, col2, col3)
3. **SELECT** 1, 11, 'FIRST'
4. **UNION** ALL
5. **SELECT** 11,12, 'SECOND'
6. **UNION** ALL
7. **SELECT** 21, 13, 'THIRD'
8. **UNION** ALL
9. **SELECT** 31, 14, 'FOURTH'

Create table2

1. **CREATE TABLE** table2 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))
2. **INSERT INTO** table2 (col1, col2, col3)
3. **SELECT** 1, 21, 'TWO-ONE'
4. **UNION** ALL
5. **SELECT** 11, 22, 'TWO-TWO'
6. **UNION** ALL
7. **SELECT** 21, 23, 'TWO-THREE'
8. **UNION** ALL
9. **SELECT** 31, 24, 'TWO-FOUR'

Now check the content in the table.

1. **SELECT** * **FROM** table_1

1. **SELECT** * **FROM** table_2

	Col 1	Col 2	Col 3
1	1	11	First
2	11	12	Second

| 3 | 21 | **13** | **Third** |
| 4 | 31 | **14** | **Fourth** |

	Col 1	Col 2	Col 3
1	1	21	Two-One
2	11	22	Two-Two
3	21	**23**	**Two-Three**
4	31	**24**	**Two-Four**

Our requirement is that we have table 2 which has two rows where Col 1 is 21 and 31. We want to update the value from table 2 to table 1 for the rows where Col 1 is 21 and 31.

We want to also update the values of Col 2 and Col 3 only.

The most easiest and common way is to use join clause in the update statement and use multiple tables in the update statement.

1. **UPDATE table** 1
2. **SET** Col 2 = t2.Col2,
3. Col 3 = t2.Col3
4. **FROM** table1 t1
5. **INNER** JOIN **table** 2 t2 **ON** t1.Col1 = t2.col1
6. **WHERE** t1.Col1 IN (21,31)

Check the content of the table

SELECT FROM table 1

SELECT FROM table 2

	Col 1	Col 2	Col 3
1	1	11	First
2	11	12	Second

| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

	Col 1	Col 2	Col 3
1	1	21	First
2	11	22	Second
3	21	**23**	**Two-Three**
4	31	**24**	**Two-Four**

SQL UPDATE DATE

How to update a date and time field in SQL?

If you want to update a date & time field in SQL, you should use the following query.

let's see the syntax of sql update date.

1. **UPDATE table**
2. **SET** Column_Name = 'YYYY-MM-DD HH:MM:SS'
3. **WHERE** Id = value

Let us check this by an example:

Firstly we take a table in which we want to update date and time fields.

If you want to change the first row which id is 1 then you should write the following syntax:

1. **UPDATE table**
2. **SET** EndDate = '2014-03-16 00:00:00.000'
3. **WHERE** Id = 1

<pNote: you should always remember that SQL must attach default 00:00:00.000 automatically.

This query will change the date and time field of the first row in that above assumed table.

SQL DELETE

The **SQL DELETE statement** is used to delete rows from a table. Generally DELETE statement removes one or more records from a table.

SQL DELETE Syntax

Let's see the Syntax for the SQL DELETE statement:

1. **DELETE FROM** table_name [**WHERE** condition];

Here table_name is the table which has to be deleted. The *WHERE clause* in SQL DELETE statement is optional here.

SQL DELETE Example

Let us take a table, named "EMPLOYEE" table.

ID	EMP_NAME	CITY	SALARY
101	Adarsh Singh	Obra	**20000**
102	Sanjay Singh	Meerut	**21000**
103	Priyanka Sharma	Raipur	**25000**
104	Esha Singhal	Delhi	**26000**

Example of delete with WHERE clause is given below:

1. **DELETE FROM** EMPLOYEE **WHERE** ID=101;

Resulting table after the query:

ID	EMP_NAME	CITY	SALARY

102	Sanjay Singh	Meerut	**21000**
103	Priyanka Sharma	Raipur	**25000**
104	Esha Singhal	Delhi	**26000**

Another example of delete statement is given below

1. **DELETE FROM** EMPLOYEE;

Resulting table after the query:

ID	EMP_NAME	CITY	SALARY

It will delete all the records of EMPLOYEE table.

It will delete the all the records of EMPLOYEE table where ID is 101.

The WHERE clause in the SQL DELETE statement is optional and it identifies the rows in the column that gets deleted.

WHERE clause is used to prevent the deletion of all the rows in the table, If you don't use the WHERE clause you might loss all the rows.

Invalid DELETE Statement for ORACLE database

You cannot use * (asterisk) symbol to delete all the records.

1. **DELETE** * **FROM** EMPLOYEE;

Topics of SQL DELETE Statement

SQL DELETE TABLE

How to delete the table and what is the difference between DELETE and TRUNCATE statement?

SQL DELETE ROW

How to delete a row from the database?

SQL DELETE All Rows

How to delete all the rows of a table?

SQL DELETE Duplicate Rows

How to use distinct keyword to delete all the duplicate rows from the table?

SQL DELETE DATABASE

There is not used DELETE statement to delete the database. But, there is used DROP statement to delete the database.

SQL DELETE VIEW

How to delete the view from the database?

SQL DELETE JOIN

How to use delete statement with INNER JOIN?

SQL JOIN

As the name shows, JOIN means *to combine something*. In case of SQL, JOIN means **"to combine two or more tables"**.

The SQL JOIN clause takes records from two or more tables in a database and combines it together.

ANSI standard SQL defines five types of JOIN :

1. inner join,
2. left outer join,
3. right outer join,
4. full outer join, and
5. cross join.In the process of joining, rows of both tables are combined in a single table.

Why SQL JOIN is used?

If you want to access more than one table through a select statement.

If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.

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

SQL INNER JOIN also known as simple join is the most common type of join.

How to use SQL join or SQL Inner Join?

Let an example to deploy SQL JOIN process:

1.Staff table

ID	Staff_NAME	Staff_AGE	STAFF_ADDRESS	Monthley_Package
1	ARYAN	22	MUMBAI	18000
2	SUSHIL	32	DELHI	20000
3	MONTY	25	MOHALI	22000
4	AMIT	20	ALLAHABAD	12000

2.Payment table

Payment_ID	DATE	Staff_ID	AMOUNT
101	30/12/2009	1	3000.00
102	22/02/2010	3	2500.00

103	23/02/2010	4	3500.00

So if you follow this JOIN statement to join these two tables ?

1. **SELECT** Staff_ID, Staff_NAME, Staff_AGE, AMOUNT
2. **FROM** STAFF s, PAYMENT p
3. **WHERE** s.ID =p.STAFF_ID;

This will produce the result like this:

STAFF_ID	NAME	Staff_AGE	AMOUNT
3	MONTY	25	2500
1	ARYAN	22	3000
4	AMIT	25	3500
1	ARYAN	22	3000

SQL PRIMARY KEY

A column or columns is called **primary key (PK)** that *uniquely identifies each row in the table*.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

When multiple columns are used as a primary key, it is known as **composite primary key**.

In designing the composite primary key, you should use as few columns as possible. It is good for storage and performance both, the more columns you use for primary key the more storage space you require.

Inn terms of performance, less data means the database can process faster.

Points to remember for primary key:

- o Primary key enforces the entity integrity of the table.

- o Primary key always has unique data.

- o A primary key length cannot be exceeded than 900 bytes.

- o A primary key cannot have null value.

- o There can be no duplicate value for a primary key.

- o A table can contain only one primary key constraint.

When we specify a primary key constraint for a table, database engine automatically creates a unique index for the primary key column.

Main advantage of primary key:

The main advantage of this uniqueness is that we get **fast access**.

In oracle, it is not allowed for a primary key to contain more than 32 columns.

SQL primary key for one column:

The following SQL command creates a PRIMARY KEY on the "S_Id" column when the "students" table is created.

MySQL:

1. **CREATE TABLE** students
2. (
3. S_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. Address **varchar** (255),
7. City **varchar** (255),
8. **PRIMARY KEY** (S_Id)
9.)

SQL Server, Oracle, MS Access:

1. **CREATE TABLE** students

2. (
3. S_Id **int** NOT NULL **PRIMARY KEY**,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. Address **varchar** (255),
7. City **varchar** (255),
8.)

SQL primary key for multiple columns:

MySQL, SQL Server, Oracle, MS Access:

1. **CREATE TABLE** students
2. (
3. S_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. Address **varchar** (255),
7. City **varchar** (255),
8. **CONSTRAINT** pk_StudentID **PRIMARY KEY** (S_Id, LastName)
9.)

Note:you should note that in the above example there is only one PRIMARY KEY (pk_StudentID). However it is made up of two columns (S_Id and LastName).

SQL primary key on ALTER TABLE

When table is already created and you want to create a PRIMARY KEY constraint on the "S_Id" column you should use the following SQL:

Primary key on one column:

1. **ALTER TABLE** students
2. **ADD PRIMARY KEY** (S_Id)

Primary key on multiple column:

1. **ALTER TABLE** students

2. **ADD CONSTRAINT** pk_StudentID **PRIMARY KEY** (S_Id,LastName)

When you use ALTER TABLE statement to add a primary key, the primary key columns must not contain NULL values (when the table was first created).

How to DROP a PRIMARY KEY constraint?

If you want to DROP (remove) a primary key constraint, you should use following syntax:

MySQL:

1. **ALTER TABLE** students
2. **DROP PRIMARY KEY**

SQL Server / Oracle / MS Access:

1. **ALTER TABLE** students
2. **DROP CONSTRAINT** pk_StudentID

SQL FOREIGN KEY

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

In simple words you can say that, a foreign key in one table used to point primary key in another table.

Let us take an example to explain it:

Here are two tables first one is students table and second is orders table.

Here orders are given by students.

First table:

S_Id	LastName	FirstName	CITY
1	MAURYA	AJEET	ALLAHABAD
2	JAISWAL	RATAN	GHAZIABAD

3	ARORA	SAUMYA	MODINAGAR

Second table:

O_Id	OrderNo	S_Id
1	99586465	2
2	78466588	2
3	22354846	3
4	57698656	1

Here you see that "S_Id" column in the "Orders" table points to the "S_Id" column in "Students" table.

- o The "S_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
- o The "S_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The foreign key constraint is generally prevents action that destroy links between tables.

It also prevents invalid data to enter in foreign key column.

SQL FOREIGN KEY constraint ON CREATE TABLE:

(Defining a foreign key constraint on single column)

To create a foreign key on the "S_Id" column when the "Orders" table is created:

MySQL:

1. **CREATE TABLE** orders
2. (
3. O_Id **int** NOT NULL,
4. Order_No **int** NOT NULL,
5. S_Id **int**,
6. PRIMAY **KEY** (O_Id),

7. **FOREIGN KEY** (S_Id) **REFERENCES** Persons (S_Id)
8.)

SQL Server /Oracle / MS Access:

1. **CREATE TABLE** Orders
2. (
3. O_Id **int** NOT NULL PRIMAY **KEY**,
4. Order_No **int** NOT NULL,
5. S_Id **int FOREIGN KEY REFERENCES** persons (S_Id)
6.)

SQL FOREIGN KEY constraint for ALTER TABLE:

If the Order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

Defining a foreign key constraint on single column:

MySQL / SQL Server / Oracle / MS Access:

1. **ALTER TABLE** Orders
2. **ADD CONSTRAINT** fk_PerOrders
3. **FOREIGN KEY**(S_Id)
4. **REFERENCES** Students (S_Id)

DROP SYNTAX for FOREIGN KEY COSTRAINT:

If you want to drop a FOREIGN KEY constraint, use the following syntax:

MySQL:

1. **ALTER TABLE** Orders
2. ROP **FOREIGN KEY** fk_PerOrders

SQL Server / Oracle / MS Access:

1. **ALTER TABLE** Orders
2. **DROP CONSTRAINT** fk_PerOrders

Difference between primary key and foreign key in SQL:

These are some important difference between primary key and foreign key in SQL-

Primary key cannot be null on the other hand foreign key can be null.

Primary key is always unique while foreign key can be duplicated.

Primary key uniquely identify a record in a table while foreign key is a field in a table that is primary key in another table.

There is only one primary key in the table on the other hand we can have more than one foreign key in the table.

By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index for foreign key.

Unique Key in SQL

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

There is an automatically defined unique key constraint within a primary key constraint.

There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

SQL UNIQUE KEY constraint on CREATE TABLE:

If you want to create a UNIQUE constraint on the "S_Id" column when the "students" table is created, use the following SQL syntax:

SQL Server / Oracle / MS Access:

(Defining a unique key constraint on single column):

1. **CREATE TABLE** students
2. (
3. S_Id **int** NOT NULL **UNIQUE**,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255)
7.)

MySQL:

1. **CREATE TABLE** students
2. **CREATE TABLE** students
3. (
4. S_Id **int** NOT NULL,
5. LastName **varchar** (255) NOT NULL,
6. FirstName **varchar** (255),
7. City **varchar** (255),
8. **UNIQUE** (S_Id)
9.)

(Defining a unique key constraint on multiple columns):

MySQL / SQL Server / Oracle / MS Access:

1. **CREATE TABLE** students
2. (
3. S_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255),
7. **CONSTRAINT** uc_studentId **UNIQUE** (S_Id, LastName)
8.)

SQL UNIQUE KEY constraint on ALTER TABLE:

If you want to create a unique constraint on "S_Id" column when the table is already created, you should use the following SQL syntax:

(Defining a unique key constraint on single column):

MySQL / SQL Server / Oracle / MS Access:

1. **ALTER TABLE** students
2. **ADD UNIQUE** (S_Id)

(Defining a unique key constraint on multiple columns):

MySQL / SQL Server / Oracle / MS Access:

1. **ALTER TABLE** students
2. **ADD CONSTRAINT** uc_StudentId **UNIQUE** (S_Id, LastName)

DROP SYNTAX FOR A FOREIGN KEY constraint:

If you want to drop a UNIQUE constraint, use the following SQL syntax:

MySQL:

1. **ALTER TABLE** students
2. **DROP INDEX** uc_studentID

SQL Server / Oracle / MS Access:

1. **ALTER TABLE** students
2. **DROP CONSTRAINT** uc_studentID

SQL Injection

The SQL Injection is a code penetration technique that might cause loss to our database. It is one of the most practiced web hacking techniques to place malicious code in SQL statements, via webpage input. SQL injection can be used to manipulate the application's web server by malicious users.

SQL injection generally occurs when we ask a user to input their username/userID. Instead of a name or ID, the user gives us an SQL statement that we will unknowingly run on our database. For Example - we create a SELECT statement by adding a variable "demoUserID" to select a string. The variable will be fetched from user input (getRequestString).

1. demoUserI = getrequestString("UserId");
2. demoSQL = "SELECT * FROM users WHERE UserId =" +demoUserId;

Types of SQL injection attacks

SQL injections can do more harm other than passing the login algorithms. Some of the SQL injection attacks include:

- o Updating, deleting, and inserting the data: An attack can modify the cookies to poison a web application's database query.

- o It is executing commands on the server that can download and install malicious programs such as Trojans.

- o We are exporting valuable data such as credit card details, email, and passwords to the attacker's remote server.

- o Getting user login details: It is the simplest form of SQL injection. Web application typically accepts user input through a form, and the front end passes the user input to the back end database for processing.

Example of SQL Injection

We have an application based on employee records. Any employee can view only their own records by entering a unique and private employee ID. We have a field like an Employee ID. And the employee enters the following in the input field:

236893238 or 1=1

It will translate to:

1. **SELECT** * **from** EMPLOYEE **where** EMPLOYEE_ID == 236893238 or 1=1

The SQL code above is valid and will return EMPLOYEE_ID row from the EMPLOYEE table. The 1=1 will return all records for which this holds true. All the employee data is compromised; now, the malicious user can also similarly delete the employee records.

Example:

1. **SELECT** * **from** Employee **where** (Username == "" or 1=1) AND (**Password**="" or 1=1).

Now the malicious user can use the '=' operator sensibly to retrieve private and secure user information. So instead of the query mentioned above, the following query, when exhausted, retrieve protected data, not intended to be shown to users.

1. **SELECT** * **from** EMPLOYEE **where** (Employee_name =" " or 1=1) AND (**Password**=" " or 1=1)

SQL injection based on Batched SQL statements

Several databases support batched SQL statements. It is a group of two or more SQL statements separated by semicolons.

The SQL statement given below will return all rows from the Employee table, then delete the Employee_Add table.

1. **SELECT** * **From** Employee; **DROP Table** Employee_Add

How to detect SQL Injection attacks

Creating a SQL Injection attack is not difficult, but even the best and good-intentioned developers make mistakes. The detection of SQL Injection is, therefore, an essential component of creating the risk of an SQL injection attack. Web Application Firewall can detect and block basic SQL injection attacks, but we should depend on it as the sole preventive measure.

Intrusion Detection System (IDS) is both network-based and host-based. It can be tuned to detect SQL injection attacks. Network-based IDSec can monitor all connections to our database server, and flags suspicious activities. The host-based IDS can monitor web server logs and alert when something strange happens.

Impact of SQL Injection

The intruder can retrieve all the user-data present in the database, such as user details, credit card information, and social security numbers, and can also gain access to protected areas like the administrator portal. It is also possible to delete the user data from the tables. These days all the online shopping applications, bank transactions use back-end database servers. If the intruder can exploit SQL injection, the entire server is compromised.

How to prevent SQL Injection attack

o We should use user authentication to validate input from the user by pre-defining length, input type, and the input field.

o Restricting the access privileges of users and defining the amount of data any outsider can access from the database. Generally, the user cannot be granted permission to access everything in the database.