Data Science and Generative AI

by SATISH @

https://sathyatech.com

Structured Query Language (SQL)

SQL is a Standard Structure QueiryLanuage

SQL

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

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

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

- You can write the keywords of SQL in both uppercase and lowercase, but writing the SQL keywords in uppercase improves the readability of the SQL query.
- SQL statements or syntax are dependent on text lines. We can place a single SQL statement on one or multiple text lines.
- You can perform most of the action in a database with SQL statements.
- SQL syntax depends on relational algebra and tuple relational calculus.

SQL Languages

SQL Languages are those languages that allow the database users to read, modify, delete and store data in the database systems.

Following are the four different types of languages or commands which are widely used in SQL queries:

1. TCL (Transaction Control Language)
2. DML (Data Manipulation Language)
3. DCL (Data Control Language)
4. DDL (Data Definition Language)
5. DRL (Data Retrival Language)

DDL (Data Definition Language)

Data Definition Languages allow users to create, modify, and destroy the schema of database objects.

We can enter the correct data in the database by applying the constraints in the DDL languages.

The DDL Languages or commands are categorized into five commands which are widely used in the SQL queries:

1. CREATE DDL Command
2. ALTER DDL Command
3. DROP DDL Command
4. TRUNCATE DDL Command
5. RENAME DDL Command

Let's discuss each DDL command with syntax and examples.

CREATE Command

This DDL command allows us to create the new table, function, stored procedure, and other database objects.

Syntax:

```
1. CREATE TABLE Name_of_Table
2.  ( Column1 datatype (Length),
3.   Column2 datatype (Length) .....);
```

Example:

The following SQL query creates the new Mobile_Details table using CREATE DDL command:

```
1. CREATE TABLE Mobile_Details
2. (
3. Mobile_Number INT NOT NULL,
4. Mobile_Name Varchar(50),
5. Manufacturing_Year INT NOT NULL,
6. Mobile_Cost INT
```

7.);

ALTER Command

This DDL command allows us to modify the structure of database objects.

Syntax:

ALTER TABLE Name_of_Table ADD Column_Name Datatype (Length of Column);

Example:

The following SQL query adds the new column in the Mobile_Details table using ALTER DDL command:

ALTER TABLE Mobile_Details ADD Mobile_Color Varchar (50);

DROP Command

This DDL command allows us to remove the table definition and data from the SQL systems.

Syntax:

DROP TABLE Name_of_Table;

Example:

DROP TABLE Mobile_Details;

TRUNCATE Command

This DDL command allows the database users to remove all the existing records from the table.

Syntax:

TRUNCATE TABLE Name_of_Table;

Example:

The following SQL query deletes all the inserted records from the Mobile_Details table using the TRUNCATE DDL command:

TRUNCATE TABLE Mobile_Details;

RENAME Command

This DDL command allows the users to change the name of the existing table.

Syntax:

RENAME Old_Table_Name TO New_Table_Name;

Example:

The following SQL query changes the name of Mobile_Details table to Mobile_Records table using the RENAME DDL command:

1. RENAME Mobile_Details TO Mobile_Records;

DML (Data Manipulation Language)

Data Manipulation languages allow database users to change the existing data of the tables.

We can use this type of language when we want to access the record, insert the new record, update the record, and delete the existing values from the tables.

Following are the four DML Languages or commands used in the SQL queries:

1. SELECT DML Command
2. INSERT DML Command
3. UPDATE DML Command
4. DELETE DML Command

Let's discuss each DML command with syntax and examples.

SELECT Command

This DML command allows us to access the stored records from the tables. We can also use the condition in the SELECT command for accessing the particular rows.

Syntax:

```
1. SELECT * FROM Name_of_Table;
```

Example:

The following SQL query shows the records of the Mobile_Records table using the SELECT DML command:

```
1. SELECT * FROM Mobile_Records;
```

INSERT Command

This DML command allows the database users to insert the new record or rows in the tables.

Syntax:

INSERT INTO Name_of_Table (Column_1, Column_2, Column_3, ...)

VALUES (Value1, Value2, Value3,.....);

Example:

The following SQL query inserts the single record of mobile into the Mobile_Records table using the INSERT DML command:

```
1. INSERT INTO Mobile_Records
2. (Mobile_number, Mobile_Name, Manufacturing_Year, Mobile_Cost, Mobile_Color)
3. VALUES (95872xxx, Apple, 2020, 95000, Black);
```

UPDATE Command

This DML command allows the database users to change the existing record or rows in the tables.

Syntax:

```
1. UPDATE Name_of_Table SET Column_Name = Value WHERE [ Condition ];
```

Example:

The following SQL query updates the values of the Mobile_Records table using the UPDATE DML command:

```
1. UPDATE Mobile_Records SET Manufacturing_Year = 2022
2. WHERE Mobile_Color = 'White';
```

DELETE Command

This DML command allows the database users to delete a particular record or row from the tables.

Syntax:

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

Example:

The following SQL query deletes the values from the Mobile_Records table using the DELETE DML command:

```
1. DELETE FROM Mobile_Records WHERE Manufacturing_Year = 2019;
```

DCL (Data Control Language)

Data Control Languages allow DBA to manage the rights and permissions on the data in the database.

Following are the two DCL Languages or commands used in the SQL queries:

1. Grant DCL Command
2. Revoke DCL Command

Let's discuss the above two DCL commands one by one with syntax and examples.

GRANT Command

This DCL command allows the database administrator to give permissions to the user for retrieving the data.

Syntax:

```
1. GRANT Name_of_Privilege ON Object TO User;
```

Example:

The following query grants the SELECT privilege on the Mobile_Records table:

```
1. GRANT SELECT ON Mobile_Records TO Ravi;
```

REVOKE Command

This DCL command allows the database administrator to remove all the permissions applied by the GRANT DCL command.

Syntax:

```
1. REVOKE Name_of_Privilege ON Object FROM User;
```

Example:

The following query removes the SELECT privilege from the Mobile_Records table:

```
1. REVOKE SELECT ON Mobile_Records FROM Ravi
```

TCL (Transaction Control Language)

Transaction Control languages maintain the SQL operations within the database. It also saves the changes made by the DML commands.

Following are the two TCL Languages or commands used in the SQL queries:

1. Commit TCL Command
2. Rollback TCL Command

Let's discuss the above TCL commands one by one with syntax and examples.

COMMIT Command

This command allows the database users to save the operations in the database.

Syntax:

```
1. COMMIT;
```

Example:

The following statements delete the record from the Mobile_Record table and commit the changes in the database:

```
1. DELETE FROM Mobile_Records WHERE Mobile_cost = 20000;
2. COMMIT;
```

Rollback Command

This command allows the database users to restore the transactions to that state which was last committed.

Syntax of Rollback command:

```
1. ROLLBACK;
```

Example of Rollback Command:

The following statements delete the record from the Mobile_Record table and rollback the changes in the database:

```
1. DELETE FROM Mobile_Records WHERE Mobile_cost = 20000;
2. ROLLBACK;
```

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.

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**.

2. 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_na
   meN = 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.

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

Data Types in MySQL, SQL Server and Oracle Databases

MySQL Data Types

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

MySQL String Data Types

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1, val2, val3,...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,val3,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

MySQL Numeric Data Types

BIT(Size)	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
INT(size)	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size

	parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by **d** parameter.
FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
DOUBLE(size, d)	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by **d** parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for **d** is 30, and the default value is 0.
DEC(size, d)	It is equal to DECIMAL(size, d).
BOOL	It is used to specify Boolean values true and false. Zero is considered as false, and nonzero values are considered as true.

MySQL Date and Time Data Types

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

SQL Server Data Types

SQL Server String Data Type

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

SQL Server Numeric Data Types

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

SQL Server Date and Time Data Type

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

date	It is used to store date only. It supports range from January 1, 0001 to December 31, 9999
time	It stores time only to an accuracy of 100 nanoseconds
timestamp	It stores a unique number when a new row gets created or modified. The time stamp value is based upon an internal clock and does not correspond to real time. Each table may contain only one-time stamp variable.

SQL Server Other Data Types

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

Oracle Data Types

Oracle String data types

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

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.

BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

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

Oracle Large Object Data Types (LOB Types)

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

SQL Operators

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

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

What is SQL Operator?

The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query. In SQL, an operator can either be a unary or binary operator. The unary operator uses only one operand for performing the unary operation, whereas the binary operator uses two operands for performing the binary operation.

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

What is the Precedence of SQL Operator?

The precedence of SQL operators is the sequence in which the SQL evaluates the different operators in the same expression. Structured Query Language evaluates those operators first, which have high precedence.

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

SQL Operator Symbols	Operators
**	Exponentiation operator
+, -	Identity operator, Negation operator
*, /	Multiplication operator, Division operator
+, -, ||	Addition (plus) operator, subtraction (minus) operator, String Concatenation operator
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison Operators
NOT	Logical negation operator
&& or AND	Conjunction operator
OR	Inclusion operator

For Example,

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

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

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

Types of Operator

SQL operators are categorized in the following categories:

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

Let's discuss each operator with their types.

SQL Arithmetic Operators

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

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

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

SQL Addition Operator (+)

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

Syntax of SQL Addition Operator:

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

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

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

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

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

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

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

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

```
1. SELECT Emp_Salary + Emp_Monthlybonus as Emp_Total_Salary FROM Employee_det
ails;
```

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

SQL Subtraction Operator (-)

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

Syntax of SQL Subtraction Operator:

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

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

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

Emp Id	Emp Name	Emp Salary	Penalty
201	Abhay	25000	200

202 Sumit 30000 500

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

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

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

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

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

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

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

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

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

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

1. SELECT Emp_Id * Emp_Salary as Emp_Id * Emp_Salary FROM Employee_details WHE RE Emp_Id = 202;

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

SQL Division Operator (/)

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

Syntax of SQL Division Operator:

1. SELECT operand1 / operand2;

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

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

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

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

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

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

1. SELECT Emp_Salary / 2 as Emp_New_Salary FROM Employee_details;

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

SQL Modulus Operator (%)

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

Syntax of SQL Modulus Operator:

1. SELECT operand1 % operand2;

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

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

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

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

1. SELECT First_operand % Second_operand as Remainder FROM Employee_details;

SQL Comparison Operators

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

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

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

SQL Equal Operator (=)

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

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

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

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

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

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

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

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

SQL Equal Not Operator (!=)

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

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

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

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

Emp Id Emp Name Emp Salary

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

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

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

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

SQL Greater Than Operator (>)

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

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

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

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

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

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

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

SQL Greater Than Equals to Operator (>=)

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

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

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

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

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

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

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

SQL Less Than Operator (<)

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

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

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

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

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

203	Bheem	30000
204	Ram	29000
205	Sumit	29000

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

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

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

SQL Less Than Equals to Operator (<=)

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

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

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

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

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

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

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

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

SQL Logical Operators

The **Logical Operators** in SQL perform the Boolean operations, which give two results **True and False.** These operators provide **True** value if both operands match the logical condition.

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

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

SQL ALL Operator

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

This operator is always used with the following statement:

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

Syntax of ALL operator:

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

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

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

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

205	Sumit	40000	Kolkata

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

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

Here, we used the **SQL ALL operator** with greater than the operator.

SQL AND Operator

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

Syntax of AND operator:

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

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

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

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

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

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

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

SQL OR Operator

The OR operator in SQL shows the record from the table if any of the conditions separated by the OR operator evaluates to True. It is also known as the conjunctive operator and is used with the WHERE clause.

Syntax of OR operator:

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

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

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

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

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

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

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

SQL BETWEEN Operator

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

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

Syntax of BETWEEN operator:

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

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

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

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

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

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

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

SQL IN Operator

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

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

Syntax of IN operator:

1. SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WH ERE 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

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

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

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

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

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

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

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

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

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

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

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

SQL ANY Operator

The **ANY operator** in SQL shows the records when any of the values returned by the sub-query meet the condition.

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

Syntax of ANY operator:

```
1. SELECT column1, column2 ...., columnN FROM table_Name WHERE column_name co
   mparison_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 WH ERE 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

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

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

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

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

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

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

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

SQL Set Operators

The **Set Operators** in SQL combine a similar type of data from two or more SQL database tables. It mixes the result, which is extracted from two or more SQL queries, into a single result.

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

Following are the various set operators which are performed on the similar data stored in the two SQL database tables:

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

SQL Union Operator

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

The data type and the number of columns must be the same for each SELECT statement used with the UNION operator. This operator does not show the duplicate records in the output table.

Syntax of UNION Set operator:

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

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

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

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

Table: Employee_details1

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

201	Sanjay	25000	Delhi

Table: Employee_details2

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

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

SQL Union ALL Operator

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

Syntax of UNION ALL Set operator:

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

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

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

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

Table: Employee_details1

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

Table: Employee_details2

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

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

SQL Intersect Operator

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

Syntax of INTERSECT Set operator:

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

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

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

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

Table: Employee_details1

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

Table: Employee_details2

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

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

SQL Minus Operator

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

Syntax of MINUS operator:

1. SELECT column1, column2 ...., columnN FROM First_tablename [WHERE conditions]

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

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

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

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

Table: Employee_details1

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

Table: Employee_details2

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

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

SQL Unary Operators

The **Unary Operators** in SQL perform the unary operations on the single data of the SQL table, i.e., these operators operate only on one operand.

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

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

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

SQL Unary Positive Operator

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

Syntax of Unary Positive Operator

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

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

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

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

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

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

SQL Unary Negative Operator

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

Syntax of Unary Negative Operator

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

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

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

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

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

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

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

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

SQL Bitwise NOT Operator

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

Syntax of Bitwise NOT Operator

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

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

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

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

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

1. SELECT ~Stu_Marks Employee_details ;

SQL Bitwise Operators

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

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

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

Bitwise AND (&)

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

Syntax of Bitwise AND Operator

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

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

This example consists of the following table, which has two columns. Each column holds numerical values.

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

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

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

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

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

Bitwise OR (|)

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

Syntax of Bitwise OR Operator

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

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

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

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

Column1 Column2

1	1
2	5
3	4
4	2
5	3

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

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

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_Nam
   e;
```

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, E
   mp_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_functio n_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_functio
   n_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, E
   mployee_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;
```

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

The **SQL DISTINCT command** is used with SELECT key word to retrieve only distinct or unique data.

In a table, there may be a chance to exist a duplicate value and sometimes we want to retrieve only unique values. In such scenarios, SQL SELECT DISTINCT statement is used.

Note: SQL SELECT UNIQUE and SQL SELECT DISTINCT statements are same.

Let's see the syntax of select distinct statement.

```
1. SELECT DISTINCT column_name ,column_name
2. FROM  table_name;
```

Let's try to understand it by the table given below:

Student_Name	Gender	Mobile_Number	HOME_TOWN
Rahul Ojha	Male	7503896532	Lucknow
Disha Rai	Female	9270568893	Varanasi
Sonoo Jaiswal	Male	9990449935	Lucknow

Here is a table of students from where we want to retrieve distinct information For example: distinct home-town.

```
1. SELECT DISTINCT home_town
2. FROM students
```

SQL SELECT COUNT

The **SQL COUNT()** is a function that returns the number of records of the table in the output.

This function is used with the SQL SELECT statement.

Let's take a simple example: If you have a record of the voters in the selected area and want to count the number of voters, then it is very difficult to do it manually, but you can do it easily by using SQL SELECT COUNT query.

Syntax of Select Count Function in SQL

1. SELECT COUNT(column_name) FROM table_name;

In the syntax, we have to specify the column's name after the COUNT keyword and the name of the table on which the Count function is to be executed.

Examples of Select Count Function in SQL

In this article, we have taken the following two SQL examples that will help you to run the Count function in the query:

Example 1: In this example, we have a table called **Bike** with three columns:

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

- Suppose, you want to count the total number of bike colors from **Bike** Table. For this operation, you have to write the following SQL statement:

1. SELECT COUNT (Bike_Color) AS TotalBikeColor FROM Bikes ;

This query will show the following output on the screen:

TotalBikeColor

6

The output of this query is six because the **Bike_Color** column does not contain any NULL value.

- Suppose, you want to count the total values of **the Bike_Cost** column from the above **Bike** Table. For this operation, you have to write the following statement in SQL:

1. SELECT COUNT (Bike_Cost) AS TotalBikeCost FROM Bikes ;

This query will show the following output on the screen:

TotalBikeCost

4

Select Count(*) Function in SQL

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

Syntax of Count (*) Function in SQL

1. SELECT COUNT(*) FROM table_name;

Example of Count (*) Function in SQL

In this example, we have the following **Bike** table with three columns:

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

- Suppose, you want to count the total number of records from the **Bike** Table. For this condition, you have to write the following statement in Structured Query Language:

1. SELECT COUNT (*) FROM Bikes ;

This query will show the following output on the screen:

Count(*)

6

SQL Count() Function With WHERE Clause

We can also use the Count() function with the WHERE clause. The Count Function with WHERE clause in the SELECT statement shows those records that matched the specified criteria.

Syntax of Count() Function With WHERE clause in SQL

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

Examples of Count Function With WHERE clause in SQL

The following two examples will help you to run the Count function with the WHERE clause in the SQL query:

Example 1: In this example, we have the following **Bike** table with three columns:

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

- Suppose, you want to count the total number of bikes whose color is black. For this, you have to type the following statement in SQL:

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

This query will show the following output on the screen:

TotalBikeBlackColor

3

Example 2: In this example, we have an **Employee_details** table with four columns:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
2001	Bheem	30000	Jaipur
2002	Ankit	45000	Delhi
2003	Sumit	40000	Delhi
2004	Ram	29000	Goa
2005	Abhay	25000	Delhi

- Suppose, you want to count the total number of those employees who belong to Delhi city. For this, you have to write the following SQL statement:

1. SELECT COUNT (Emp_Name) AS TotalEmpCity FROM Employee_details WHERE Emp_ City = 'Delhi';

This query will show the following output on the screen:

TotalEmpCity
3

SQL Count Function With DISTINCT keyword

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

Syntax of Count Function With DISTINCT keyword in SQL

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

Examples of Count Function With DISTINCT keyword in SQL

The following two examples will help you how to run the Count function with the DISTINCT keyword in the SQL query:

Example 1:

In this example, we have taken the following **Cars** table with three columns:

Car_Name	Car_Color	Car_Cost
i20	White	10,85,000
Hyundai Venue	Black	9,50,000
Swift Dezire	Red	9,00,000
Hyundai Creta	White	7,95,000
Kia Seltos	White	8,00,000

Kia Sonet	Red	10,00,000

- Suppose, you want to count the unique colors of a car from the above table. For this query, you have to write the below statement in SQL:

1. SELECT COUNT (DISTINCT Car_Color) AS Unique_Car_Color FROM Cars ;

SQL SELECT TOP

The **SELECT TOP** statement in SQL shows the limited number of records or rows from the database table. The TOP clause in the statement specifies how many rows are returned.

It shows the top N number of rows from the tables in the output. This clause is used when there are thousands of records stored in the database tables.

Let's take a simple example: If a Student table has a large amount of data about students, the select TOP statement determines how much student data will be retrieved from the given table.

Note: All the database systems do not support the TOP keyword for selecting the limited number of records. Oracle supports the ROWNUM keyword, and MySQL supports the LIMIT keyword.

Syntax of TOP Clause in SQL

1. SELECT TOP number | percent column_Name1, column_Name2, ..., column_NameN FROM table_name WHERE [Condition] ;

In the syntax, **the number** denotes the number of rows shown from the top in the output. column_Name denotes the column whose record we want to show in the output. We can also specify the condition using the WHERE clause.

Examples of TOP Clause in SQL

The following four SQL examples will help you how to use the Number and Percent in SQL TOP clause in the query:

Example 1: In this example, we have a table called **Cars** with three columns:

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

- Suppose, you want to show the first three Names and Color of Car from the above table. To do this, you have to type the following query in SQL:

```
1. SELECT TOP 3 Car_Name, Car_Color FROM Cars;
```

This query shows the following table on the screen:

Car_Name	Car_Color
Hyundai Creta	White
Hyundai Venue	White
Hyundai i20	Red

Example 2: In this example, we have a table called **Student** with three columns:

Stu_ID	Stu_Name	Stu_Marks
1001	Abhay	85
1002	Ankit	75
1003	Bheem	60
1004	Ram	79
1005	Sumit	80

- Suppose, you want to show the details of the first four students in the result from the above table. To do this, you have to type the following query in SQL:

```
1. SELECT TOP 4 * FROM Student;
```

This query shows the following table on the screen in the SQL output:

Stu_ID	Stu_Name	Stu_Marks
1001	Abhay	85
1002	Ankit	75
1003	Bheem	60
1004	Ram	79

Example 3: In this example, we have a table called **Employee** with four columns:

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

- Suppose, you want to show the details of those first four employees whose city is Goa from the above table. To do this, you have to type the following query in SQL:

1. SELECT TOP 4 * FROM Employee WHERE Emp_City = Goa ;

This query shows the following table on the screen in the SQL output:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	25000	Goa
203	Bheem	30000	Goa
204	Ram	29000	Goa

Example 4: In this example, we have a table called **Bikes** with three columns:

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

- Suppose, you want to show the 50 percent of data from the above table. To do this, you have to type the following query in SQL:

1. SELECT TOP 50 PERCENT * FROM Bikes;

This query shows the following table on the screen:

Bike_Name	Bike_Color	Bike_Cost
KTM DUKE	Black	185,000
Royal Enfield	Black	NULL
Pulsar	Red	90,0000

Syntax of LIMIT Clause in MySQL

1. SELECT column_Name1,column_Name2, ..., column_NameN FROM table_name LIM IT value;

In the syntax, we have to specify the value after the LIMIT keyword. The value denotes the number of rows to be shown from the top in the output.

Example of LIMIT Clause in MySQL

The following SQL example will help you how to use the LIMIT clause in the query. In this example, we have a table called **Cars** with three columns:

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

- Suppose, you want to show the first three records of Car using a LIMIT clause in MySQL. To do this, you have to type the following query in MySQL:

1. SELECT * FROM Cars LIMIT 3;

This query shows the following table on the screen:

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000

Syntax of ROWNUM keyword in WHERE Clause in Oracle database

1. SELECT column_Name1,column_Name2, ..., column_NameN FROM table_name WH ERE ROWNUM <= value;

In the syntax, we have to assign the value to ROWNUM in the WHERE clause. The value denotes the number of rows to be shown from the top in the output.

Example of ROWNUM keyword in WHERE Clause in Oracle

The following SQL example will help you how to use the ROWNUM keyword in the query. In this example, we have a table called **Cars** with three columns:

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

- Suppose, you want to show the first three records of Car using the ROWNUM keyword in Oracle. To do this, you have to type the following query in the Oracle database:

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

This query shows the following table on the screen:

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000

SQL SELECT FIRST

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

Let's see the syntax of sql select first() function:

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

Here a point is notable that first function is only supported by MS Access.

If you want to retrieve the first value of the "customer_name" column from the "customers" table, you need to write following query:

```
1. SELECT FIRST(customer_name) AS first_customer FROM customers;
```

Let us take the example of CUSTOMERS to examine SQL SELECT FIRST command:

Table CUSTOMERS

CUSTOMER_NAME	AGE	ADDRESS	EXPENDITURE
KAMAL SHARMA	26	GHAZIABAD	6000
ROBERT PETT	23	NEWYORK	26000
SHIKHA SRIVASTAV	22	DELHI	9000

If you want to retrieve the first value of the "customer_name" column from the "customers" table, you need to write following query:

Let's see the syntax of sql select first() function:

1. SELECT FIRST (CUSTOMER_NAME) AS first_customer FROM CUSTOMERS;
2. After that query, you will find the result:
3. KAMAL SHARMA

Note: The SELECT FIRST statement is only supported by MS Access. This statement doesn't work with other databases like Oracle, MySQL etc.

SQL SELECT LAST

The **LAST()** function in Structured Query Language shows the last value from the specified column of the table.

Note: This SQL function is only supported in Microsoft Access database. Oracle supports ORDER BY and ROWNUM keywords, and MySQL supports the LIMIT keyword for selecting the last record.

Syntax of LAST() Function

1. SELECT LAST (Field_Name) FROM Table_Name ;

In the above syntax, the LAST keyword denotes the last row to be shown from the table in the output, and the **Field_Name** denotes the column whose value we want to show.

Example of the LAST function in SQL

Example 1:

Firstly, we have to create a table and insert the data into the table in SQL.

The following SQL statement creates the **Student_Details** table with **Student_ID** as the primary key:

```
1. CREATE TABLE Student_Details
2. (
3. Student_ID INT NOT NULL,
4. Student_Name varchar(100),
5. Student_Course varchar(50),
6. Student_Age INT,
7. Student_Marks INT
8. );
```

The following SQL queries insert the record of students into the above table using INSERT INTO statement:

```
1. INSERT INTO Student_Details VALUES (101, Anuj, B.tech, 20, 88);
2. INSERT INTO Student_Details VALUES (102, Raman, MCA, 24, 98);
3. INSERT INTO Student_Details VALUES (104, Shyam, BBA, 19, 92);
4. INSERT INTO Student_Details VALUES (107, Vikash, B.tech, 20, 78);
5. INSERT INTO Student_Details VALUES (111, Monu, MBA, 21, 65);
6. INSERT INTO Student_Details VALUES (114, Jones, B.tech, 18, 93);
7. INSERT INTO Student_Details VALUES (121, Parul, BCA, 20, 97);
8. INSERT INTO Student_Details VALUES (123, Divya, B.tech, 21, 89);
9. INSERT INTO Student_Details VALUES (128, Hemant, MBA, 23, 90);
10. INSERT INTO Student_Details VALUES (130, Nidhi, BBA, 20, 88);
11. INSERT INTO Student_Details VALUES (132, Priya, MBA, 22, 99);
12. INSERT INTO Student_Details VALUES (138, Mohit, MCA, 21, 92);
```

Let's see the record of the above table using the following SELECT statement:

```
1. SELECT * FROM Student_Details;
```

Student_ID	Student_Name	Student_Course	Student_Age	Student_Marks
101	Anuj	B.tech	20	88
102	Raman	MCA	24	98
104	Shyam	BBA	19	92
107	Vikash	B.tech	20	78
111	Monu	MBA	21	65
114	Jones	B.tech	18	93
121	Parul	BCA	20	97

123	Divya	B.tech	21	89
128	Hemant	MBA	23	90
130	Nidhi	BBA	20	88
132	Priya	MBA	22	99
138	Mohit	MCA	21	92

The following query shows the last Student_Name from the above table in the output:

1. SELECT LAST (Student_Name) AS Last_Student FROM Student_Details;

Output:

Last_Student
Mohit

Syntax of LIMIT Clause in MySQL

1. SELECT column_Name FROM Table_Name ORDER BY Column_Name DESC LIMIT 1;

In this MySQL syntax, we have to specify the value 1 just after the LIMIT keyword for indicating the single row/record.

Example of LIMIT Clause in MySQL

Let's take the following Employee table to explain how to use the LIMIT clause in MySQL for accessing the last record:

Employee_Id	Emp_Name	Emp_City	Emp_Salary	Emp_Bonus
101	Anuj	Ghaziabad	35000	2000
102	Tushar	Lucknow	29000	3000
103	Vivek	Kolkata	35000	2500
104	Shivam	Goa	22000	3000

The following MySQL query shows the last value of the **Emp_City** column from the above Employee table:

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

Output:

Goa

ROWNUM keyword in Oracle

The syntax for accessing the last record from the Oracle database is given below:

1. SELECT Column_Name FROM Table_Name ORDER BY Column_Name DESC WHERE R OWNUM <=1;

In this Oracle syntax, we have to specify the ROWNUM keyword, which is less than and equal to 1. In Oracle, the ROWNUM keyword is used in the WHERE clause for retrieving the last record from the table.

Example of ROWNUM Clause in Oracle

Let's take the following Cars table to explain how to use the ROWNUM keyword in MySQL:

Car_Number	Car_Name	Car_Amount	Car_Price
2578	Creta	3	900000
9258	Audi	2	1100000
8233	Venue	6	900000
6214	Nexon	7	1000000

The following MySQL query shows the last name of the car from the **Car_Name** column of the Cars table:

1. SELECT Car_Name FROM Cars ORDER BY Car_Name DESC WHERE ROWNUM <=1;

Output:

Nexon

SQL SELECT RANDOM

The SQL SELECT RANDOM() function returns the random row. It can be used in online exam to display the random questions.

There are a lot of ways to select a random record or row from a database table. Each database server needs different SQL syntax.

If you want to select a random row with **MY SQL**:

1. SELECT column FROM table

2. ORDER BY RAND ( )
3. LIMIT 1

If you want to select a random row with **Microsoft SQL server**:

1. SELECT TOP 1 column FROM table
2. ORDER BY NEW ID()

If you want to select a random record with **ORACLE**:

1. SELECT column FROM
2. (SELECT column FROM table
3. ORDER BY dbms_random.value)
4. WHERE rownum =1

If you want to select a random row with **PostgreSQL**:

1. SELECT column FROM table
2. ORDER BY RAND()
3. LIMIT 1

SQL SELECT AS

- SQL '**AS**' is used to assign a new name temporarily to a table column or even a table.
- It makes an easy presentation of query results and allows the developer to label results more accurately without permanently renaming table columns or even the table itself.
- Let's see the syntax of select as:

1. SELECT Column_Name1 AS New_Column_Name, Column_Name2 As New_Column_Name FROM Table_Name;

Here, the Column_Name is the name of a column in the original table, and the New_Column_Name is the name assigned to a particular column only for that specific query. This means that New_Column_Name is a temporary name that will be assigned to a query.

Assigning a temporary name to the column of a table:

Let us take a table named orders, and it contains the following data:

Day_of_order	Customer	Product	Quantity
11-09-2001	Ajeet	Mobile	2
13-12-2001	Mayank	Laptop	20
26-12-2004	Balaswamy	Water cannon	35

Example:

Suppose you want to rename the 'day_of_order' column and the 'customer' column as 'Date' and 'Client', respectively.

Query:

1. SELECT day_of_order AS 'Date', Customer As 'Client', Product, Quantity FROM orders;

HAVING Clause in SQL

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement.

This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

Difference between HAVING and WHERE Clause

The difference between the WHERE and HAVING clauses in the database is the most important question asked during an IT interview.

The following table shows the comparisons between these two clauses, but the main difference is that the WHERE clause uses condition for filtering records before any groupings are made, while HAVING clause uses condition for filtering values from a group.

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.
5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. We can implements this SQL clause in column operations.	6. We can implements this SQL clause in row operations.
7. It is a post-filter.	7. It is a pre-filter.
8. It is used to filter groups.	8. It is used to filter the single record of the table.

Syntax of HAVING clause in SQL

1. SELECT column_Name1, column_Name2, ..., column_NameN aggregate_function_name(column_Name) FROM table_name GROUP BY column_Name1 HAVING condition;

Examples of HAVING clause in SQL

In this article, we have taken the following four different examples which will help you how to use the HAVING clause with different SQL aggregate functions:

Example 1: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with SUM aggregate function:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	2000	Goa
202	Ankit	4000	Delhi
203	Bheem	8000	Jaipur

204	Ram	2000	Goa
205	Sumit	5000	Delhi

If you want to add the salary of employees for each city, you have to write the following query:

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

The output of the above query shows the following output:

SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

Now, suppose that you want to show those cities whose total salary of employees is more than 5000. For this case, you have to type the following query with the HAVING clause in SQL:

1. SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City HAVING S UM(Emp_Salary)>5000;

The output of the above SQL query shows the following table in the output:

SUM(Emp_Salary)	Emp_City
9000	Delhi
8000	Jaipur

Example 2: Let's take the following **Student_details** table, which helps you to analyze the HAVING clause with the COUNT aggregate function:

Roll_No	Name	Marks	Age
1	Rithik	91	20
2	Kapil	60	19
3	Arun	82	17
4	Ram	92	18
5	Anuj	50	20
6	Suman	88	18
7	Sheetal	57	19

8	Anuj	64	20

Suppose, you want to count the number of students from the above table according to their age. For this, you have to write the following query:

1. SELECT COUNT(Roll_No), Age FROM Student_details GROUP BY Age ;

The above query will show the following output:

Count(Roll_No)	Age
3	20
2	19
1	17
2	18

Now, suppose that you want to show the age of those students whose roll number is more than and equals 2. For this case, you have to type the following query with the HAVING clause in SQL:

1. SELECT COUNT(Roll_No), Age FROM Student_details GROUP BY Age HAVING COUNT(Roll_No) >= 2 ;

The output of the above SQL query shows the following table in the output:

Count(Roll_No)	Age
3	20
2	19
2	18

Example 3: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with MIN and MAX aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	9000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding
1005	Seenu	5000	HR

1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	8000	Finance

MIN Function with HAVING Clause:

If you want to show each department and the minimum salary in each department, you have to write the following query:

1. SELECT MIN(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;

The output of the above query shows the following output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
4000	HR
3000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose minimum salary of employees is greater than 4000. For this case, you have to type the following query with the HAVING clause in SQL:

1. SELECT MIN(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept HAVING MIN(Emp_Salary) > 4000 ;

The above SQL query shows the following table in the output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
10000	Marketing

MAX Function with HAVING Clause:

In the above employee table, if you want to list each department and the maximum salary in each department. For this, you have to write the following query:

1. SELECT MAX(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;

The above query will show the following output:

MAX(Emp_Salary) Emp_Dept

9000	Finance
5000	HR
6000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose maximum salary of employees is less than 8000. For this case, you have to type the following query with the HAVING clause in SQL:

1. SELECT MAX(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept HAVING MAX(Emp_Salary) < 8000 ;

The output of the above SQL query shows the following table in the output:

MAX(Emp_Salary)	Emp_Dept
5000	HR
6000	Coding

Example 4: Let's take the following **Employee_Dept** table, which helps you to analyze the HAVING clause with AVG aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	8000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding
1005	Seenu	5000	HR
1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	6000	Finance

If you want to find the average salary of employees in each department, you have to write the following query:

1. SELECT AVG(Emp_Salary), Emp_Dept FROM Employee_Dept GROUP BY Emp_Dept;

The above query will show the following output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
4500	HR

6500	Coding
10000	Marketing

Now, suppose that you want to show those departments whose average salary is more than and equals 6500. For this case, you have to type the following query with the HAVING clause in SQL:

```
1. SELECT AVG(Emp_Salary), Emp_Dept FROM Employee_Dept GROUP BY Emp_Dept H
   AVING AVG(Emp_Salary) > 6500 ;
```

The above SQL query will show the following table in the output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
6500	Coding
10000	Marketing

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) VALUES(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");

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 = valueN] [WHERE condition]

Let's see the Syntax:

1. UPDATE table_name
2. SET column_name = expression
3. 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';
```

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

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;

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;

SQL DELETE ROW

Let us take an example of student.

Original table:

ID	STUDENT _NAME	ADDRESS
001	AJEET MAURYA	GHAZIABAD
002	RAJA KHAN	LUCKNOW
003	RAVI MALIK	DELHI

If you want to delete a student with id 003 from the student_name table, then the SQL DELETE query should be like this:

```
1. DELETE FROM student_name
2. WHERE id = 003;
```

SQL DELETE ALL ROWS

The statement SQL DELETE ALL ROWS is used to delete all rows from the table. If you want to delete all the rows from student table the query would be like,

```
1. DELETE FROM STUDENT_NAME;
```

SQL DELETE DUPLICATE ROWS

If you have got a situation that you have multiple duplicate records in a table, so at the time of fetching records from the table you should be more careful. You make sure that you are fetching unique records instead of fetching duplicate records.

To overcome with this problem we use DISTINCT keyword.

It is used along with SELECT statement to eliminate all duplicate records and fetching only unique records.

SYNTAX:

The basic syntax to eliminate duplicate records from a table is:

```
1. SELECT DISTINCT column1, column2,....columnN
2. FROM table _name
3. WHERE [conditions]
```

EXAMPLE:

Let us take an example of STUDENT table.

ROLL_NO	NAME	PERCENTAGE	ADDRESS
1	AJEET MAURYA	72.8	ALLAHABAD
2	CHANDAN SHARMA	63.5	MATHURA
3	DIVYA AGRAWAL	72.3	VARANASI
4	RAJAT KUMAR	72.3	DELHI
5	RAVI TYAGI	75.5	HAPUR

6 SONU JAISWAL 71.2 GHAZIABAD

Firstly we should check the SELECT query and see how it returns the duplicate percentage records.

1. SQL > SELECT PERCENTAGE FROM STUDENTS
2. ORDER BY PERCENTAGE;

PERCENTAGE

63.5

71.2

72.3

72.3

72.8

75.5

Now let us use SELECT query with DISTINCT keyword and see the result. This will eliminate the duplicate entry.

1. SQL > SELECT DISTINCT PERCENTAGE FROM STUDENTS
2. ORDER BY PERCENTAGE;

PERCENTAGE

63.5

71.2

72.3

72.8

75.5

SQL DELETE DATABASE

You can easily remove or delete indexes, tables and databases with the DROP statement.

The DROP index statement is:

Used to delete index in the table

DROP INDEX SYNTAX for MS Access:

1. DROP INDEX index_name ON table_name

DROP INDEX SYNTAX for MS SQL Server:

1. DROP INDEX table_name.index_name

DROP INDEX syntax for DB2/Oracle:

1. DROP INDEX index_name

DROP INDEX syntax for MySQL:

1. ALTER TABLE table_name DROP INDEX index_name

DROP DATABASE Statement:

The drop database statement is used to delete a database.

1. DROP DATABASE database_name

SQL DELETE VIEW

Before knowing about what is SQL delete view, it is important to know -

What is SQL view?

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

The SQL view is a table which does not physically exist. It is only a virtual table.

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

Syntax for SQL create view -

1. CREATE VIEW view_name AS
2. SELECT columns
3. FROM tables
4. WHERE conditions;

If you want to delete a SQL view, It is done by SQL DROP command you should use the following syntax:

SQL DROP VIEW syntax:

1. DROP VIEW view_name

SQL Server JOINS

In real life, we store our data in multiple logical tables that are linked together by a common key value in relational databases like SQL Server, Oracle, MySQL, and others. As a result, we constantly need to get data from two or more tables into the desired output based on some conditions. We can quickly achieve this type of data in SQL Server using the SQL JOIN clause. This article gives a complete overview of JOIN and its different types with an example.

The join clause allows us to **retrieve data from two or more related tables** into a meaningful result set. We can join the table using a **SELECT** statement and a **join condition**. It indicates how SQL Server can use data from one table to select rows from another table. In general, tables are related to each other using **foreign key** constraints.

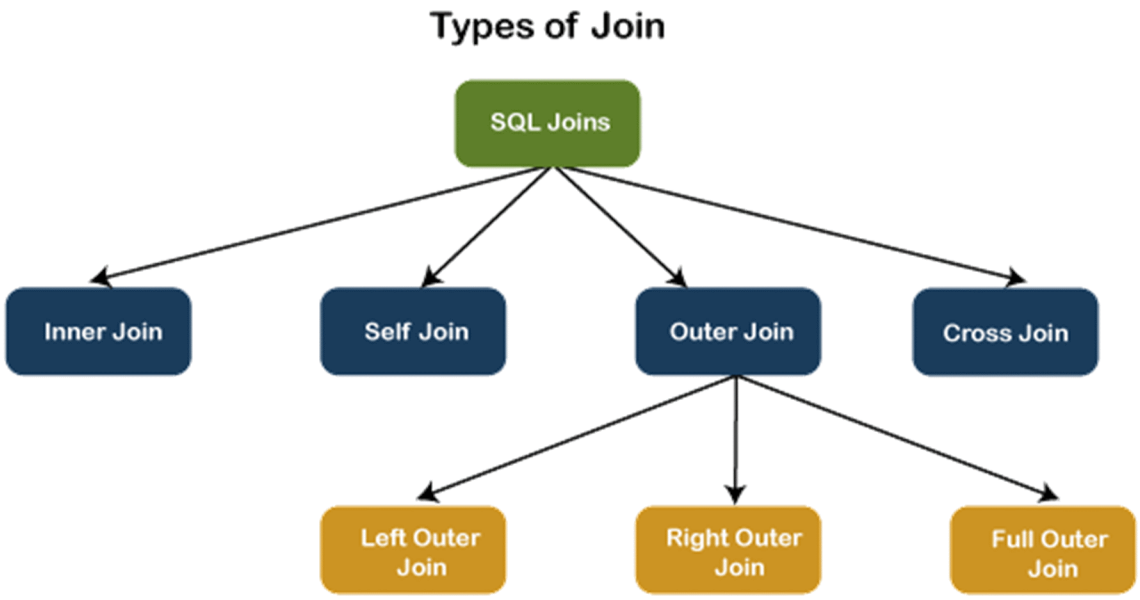
In a JOIN query, a condition indicates how two tables are related:

- Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
- Specify the logical operator to compare values from the columns like =, <, or >.

Types of JOINS in SQL Server

SQL Server mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:

1. INNER JOIN
2. SELF JOIN
3. CROSS JOIN
4. OUTER JOIN

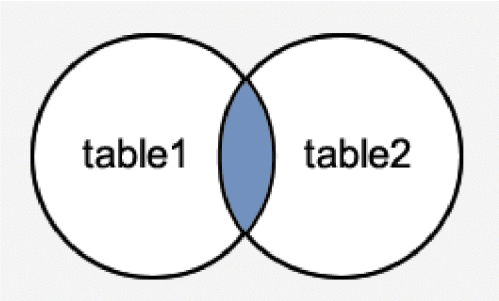


Let us discuss each of these joins in detail.

INNER JOIN

This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a **default join**. If we omit the INNER keyword with the JOIN query, we will get the same output.

The following visual representation explains how INNER JOIN returns the matching records from **table1** and **table2:**



INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

1. **SELECT** columns
2. **FROM** table1
3. **INNER** JOIN table2 **ON** condition1
4. **INNER** JOIN table3 **ON** condition2

INNER JOIN Example

Let us first create two tables "**Student**" and "**Fee**" using the following statement:

```
CREATE TABLE Student (
  id int PRIMARY KEY IDENTITY,
  admission_no varchar(45) NOT NULL,
  first_name varchar(45) NOT NULL,
  last_name varchar(45) NOT NULL,
  age int,
  city varchar(25) NOT NULL
);

CREATE TABLE Fee (
  admission_no varchar(45) NOT NULL,
  course varchar(45) NOT NULL,
  amount_paid int,
);
```

Next, we will insert some records into these tables using the below statements:

```
INSERT INTO Student (admission_no, first_name, last_name, age, city)
VALUES (3354,'Luisa', 'Evans', 13, 'Texas'),
(2135, 'Paul', 'Ward', 15, 'Alaska'),
(4321, 'Peter', 'Bennett', 14, 'California'),
(4213,'Carlos', 'Patterson', 17, 'New York'),
(5112, 'Rose', 'Huges', 16, 'Florida'),
(6113, 'Marielia', 'Simmons', 15, 'Arizona'),
(7555,'Antonio', 'Butler', 14, 'New York'),
(8345, 'Diego', 'Cox', 13, 'California');


INSERT INTO Fee (admission_no, course, amount_paid)
VALUES (3354,'Java', 20000),
(7555, 'Android', 22000),
(4321, 'Python', 18000),
(8345,'SQL', 15000),
(5112, 'Machine Learning', 30000);
```

Execute the **SELECT** statement to verify the records:

Table: Student

id	admission_no	first_name	last_name	age	city
1	3354	Luisa	Evans	13	Texas
2	2135	Paul	Ward	15	Alaska
3	4321	Peter	Bennett	14	California
4	4213	Carlos	Patterson	17	New York
5	5112	Rose	Huges	16	Florida
6	6113	Marielia	Simmons	15	Arizona
7	7555	Antonio	Butler	14	New York
8	8345	Diego	Cox	13	California

Table: Fee

admission_no	course	amount_paid
3354	Java	20000
7555	Android	22000
4321	Python	18000
8345	SQL	15000
5112	Machine Learning	30000

We can demonstrate the INNER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
INNER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this example, we have used the **admission_no column** as a join condition to get the data from both tables. Depending on this table, we can see the information of the students who have paid their fee.

SELF JOIN

A table is joined to itself using the SELF JOIN. It means that **each table row is combined with itself** and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of **table name aliases** to assign a specific name to each table's instance. The table aliases enable us to use the **table's temporary** name that we are going to use in the query. It's a useful way to extract hierarchical data and comparing rows inside a single table.

SELF JOIN Syntax

The following expression illustrates the syntax of SELF JOIN in SQL Server. It works the same as the syntax of joining two different tables. Here, we use aliases names for tables because both the table name are the same.

```
1. SELECT T1.col_name, T2.col_name...
2. FROM table1 T1, table1 T2
3. WHERE join_condition;
```

Example

We can demonstrate the SELF JOIN using the following command:

```
1. SELECT S1.first_name, S2.last_name, S2.city
2. FROM Student S1, Student S2
3. WHERE S1.id <> S2.iD AND S1.city = S2.city
4. ORDER BY S2.city;
```

This command gives the below result:

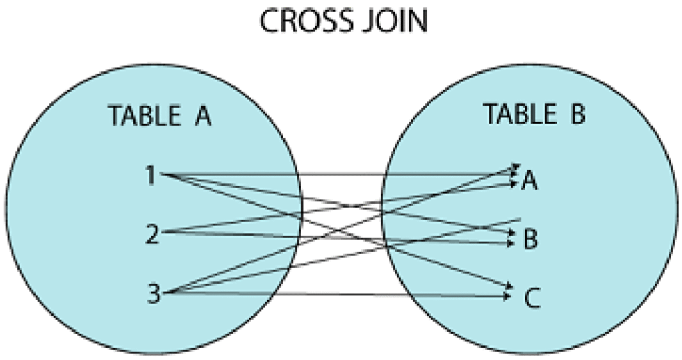
first_name	last_name	City
Peter	Cox	California
Diego	Bennett	California
Carlos	Butler	New York
Antonio	Patterson	New York

In this example, we have used the **id and city column** as a join condition to get the data from both tables.

CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as **CARTESIAN JOIN** because it produces the **Cartesian product** of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.

The below visual representation illustrates the CROSS JOIN. It will give all the records from **table1** and **table2** where each row is the combination of rows of both tables:



CROSS JOIN Syntax

The following syntax illustrates the use of CROSS JOIN in SQL Server:

```
1. SELECT column_lists
2. FROM table1
3. CROSS JOIN table2;
```

Example

We can demonstrate the CROSS JOIN using the following command:

```
1. SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.am
   ount_paid
2. FROM Student
3. CROSS JOIN Fee
4. WHERE Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

OUTER JOIN

OUTER JOIN in SQL Server returns all records from both tables that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

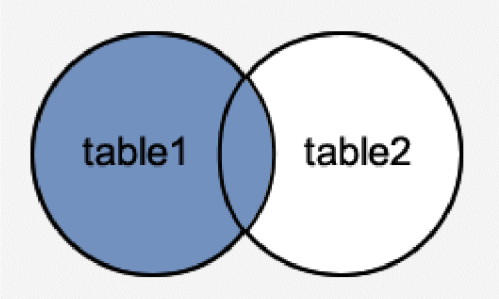
We can categories the OUTER JOIN further into three types:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN

The LEFT OUTER JOIN retrieves all the records from the left table and matching rows from the right table. It will return **NULL** when no matching record is found in the right side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.

The below visual representation illustrates the LEFT OUTER JOIN:



LEFT OUTER JOIN Syntax

The following syntax illustrates the use of LEFT OUTER JOIN in SQL Server:

```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example

We can demonstrate the LEFT OUTER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
```

3. LEFT OUTER JOIN Fee
4. ON Student.admission_no = Fee.admission_no;

This command gives the below result:

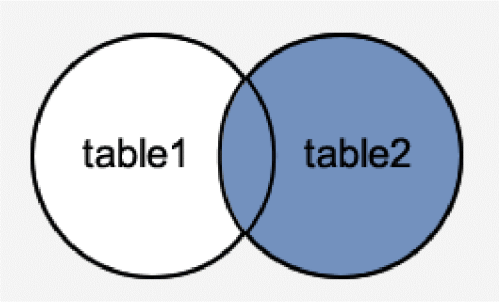
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

This output shows that the unmatched row's values are replaced with NULLs in the respective columns.

RIGHT OUTER JOIN

The RIGHT OUTER JOIN retrieves all the records from the right-hand table and matched rows from the left-hand table. It will return **NULL** when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

The below visual representation illustrates the RIGHT OUTER JOIN:



RIGHT OUTER JOIN Syntax

The following syntax illustrates the use of RIGHT OUTER JOIN in SQL Server:

1. SELECT column_lists
2. FROM table1
3. RIGHT [OUTER] JOIN table2
4. ON table1.column = table2.column;

Example

The following example explains how to use the RIGHT OUTER JOIN to get records from both tables:

```
1. SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.am
   ount_paid
2. FROM Student
3. RIGHT OUTER JOIN Fee
4. ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

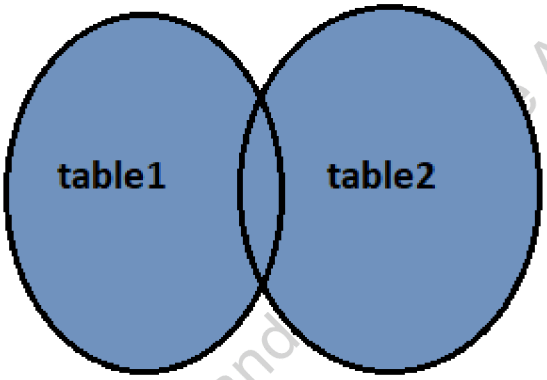
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
7555	Antonio	Butler	Android	22000
4321	Peter	Bennett	Python	18000
8345	Diego	Cox	SQL	15000
5112	Rose	Huges	Machine Learning	30000

In this output, we can see that no column has NULL values because all rows in the Fee table are available in the Student table based on the specified condition.

FULL OUTER JOIN

The FULL OUTER JOIN in SQL Server returns a result that includes all rows from both tables. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

The below visual representation illustrates the **FULL OUTER JOIN**:



FULL OUTER JOIN Syntax

The following syntax illustrates the use of FULL OUTER JOIN in SQL Server:

```
1. SELECT column_lists
2. FROM table1
3. FULL [OUTER] JOIN table2
4. ON table1.column = table2.column;
```

Example

The following example explains how to use the FULL OUTER JOIN to get records from both tables:

```
1. SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.am
   ount_paid
2. FROM Student
3. FULL OUTER JOIN Fee
4. ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

SQL PRACTICE TABLES:

```
DROP TABLE EMP

DROP TABLE DEPT

DROP TABLE BONUS

DROP TABLE SALGRADE

DROP TABLE DUMMY


CREATE TABLE EMP

(EMPNO NUMERIC(4) NOT NULL,

ENAME VARCHAR(10),

JOB VARCHAR(9),

MGR NUMERIC(4),

HIREDATE DATETIME,

SAL NUMERIC(7, 2),

COMM NUMERIC(7, 2),

DEPTNO NUMERIC(2))


INSERT INTO EMP VALUES

(7369, 'SMITH', 'CLERK', 7902, '17-DEC-1980', 800, NULL, 20)

INSERT INTO EMP VALUES

(7499, 'ALLEN', 'SALESMAN', 7698, '20-FEB-1981', 1600, 300, 30)

INSERT INTO EMP VALUES

(7521, 'WARD', 'SALESMAN', 7698, '22-FEB-1981', 1250, 500, 30)

INSERT INTO EMP VALUES
```

(7566, 'JONES', 'MANAGER', 7839, '2-APR-1981', 2975, NULL, 20)

INSERT INTO EMP VALUES

(7654, 'MARTIN', 'SALESMAN', 7698, '28-SEP-1981', 1250, 1400, 30)

INSERT INTO EMP VALUES

(7698, 'BLAKE', 'MANAGER', 7839, '1-MAY-1981', 2850, NULL, 30)

INSERT INTO EMP VALUES

(7782, 'CLARK', 'MANAGER', 7839, '9-JUN-1981', 2450, NULL, 10)

INSERT INTO EMP VALUES

(7788, 'SCOTT', 'ANALYST', 7566, '09-DEC-1982', 3000, NULL, 20)

INSERT INTO EMP VALUES

(7839, 'KING', 'PRESIDENT', NULL, '17-NOV-1981', 5000, NULL, 10)

INSERT INTO EMP VALUES

(7844, 'TURNER', 'SALESMAN', 7698, '8-SEP-1981', 1500, 0, 30)

INSERT INTO EMP VALUES

(7876, 'ADAMS', 'CLERK', 7788, '12-JAN-1983', 1100, NULL, 20)

INSERT INTO EMP VALUES

(7900, 'JAMES', 'CLERK', 7698, '3-DEC-1981', 950, NULL, 30)

INSERT INTO EMP VALUES

(7902, 'FORD', 'ANALYST', 7566, '3-DEC-1981', 3000, NULL, 20)

INSERT INTO EMP VALUES

(7934, 'MILLER', 'CLERK', 7782, '23-JAN-1982', 1300, NULL, 10)

CREATE TABLE DEPT

(DEPTNO NUMERIC(2),

```
DNAME VARCHAR(14),
LOC VARCHAR(13) )

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK')
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS')
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO')
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON')

CREATE TABLE BONUS
(ENAME VARCHAR(10),
JOB VARCHAR(9),
SAL NUMERIC,
COMM NUMERIC)

CREATE TABLE SALGRADE
(GRADE NUMERIC,
LOSAL NUMERIC,
HISAL NUMERIC)

INSERT INTO SALGRADE VALUES (1, 700, 1200)
INSERT INTO SALGRADE VALUES (2, 1201, 1400)
INSERT INTO SALGRADE VALUES (3, 1401, 2000)
INSERT INTO SALGRADE VALUES (4, 2001, 3000)
INSERT INTO SALGRADE VALUES (5, 3001, 9999)
```

CREATE TABLE DUMMY

(DUMMY NUMERIC)

INSERT INTO DUMMY VALUES (0)

select * from emp

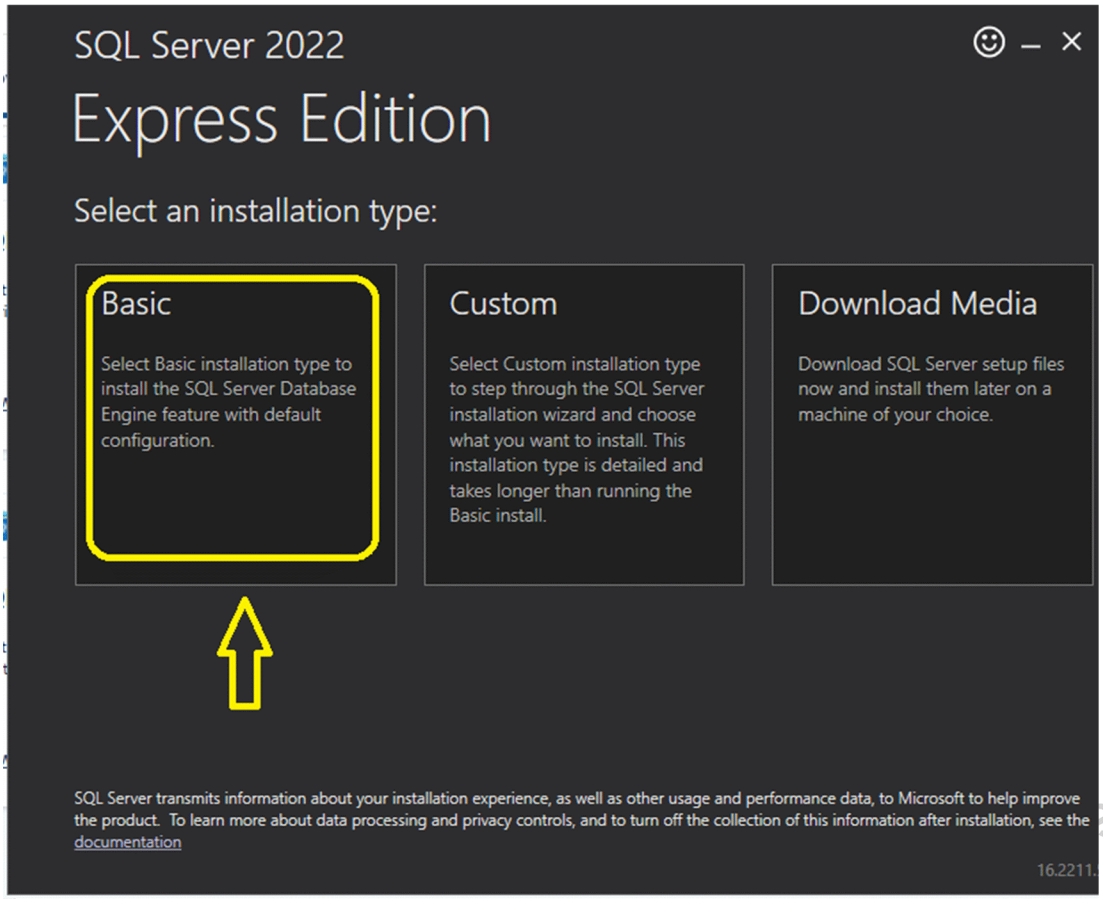
select * from dept

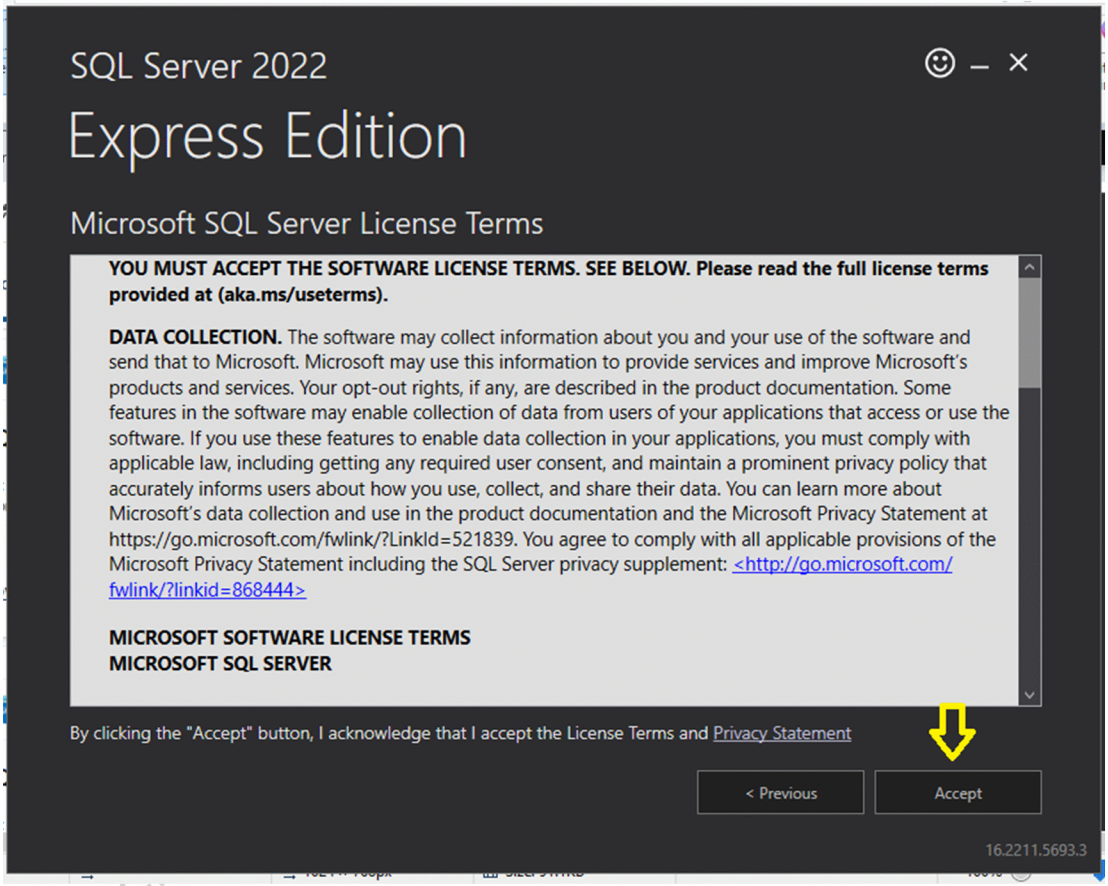
select * from salgrade

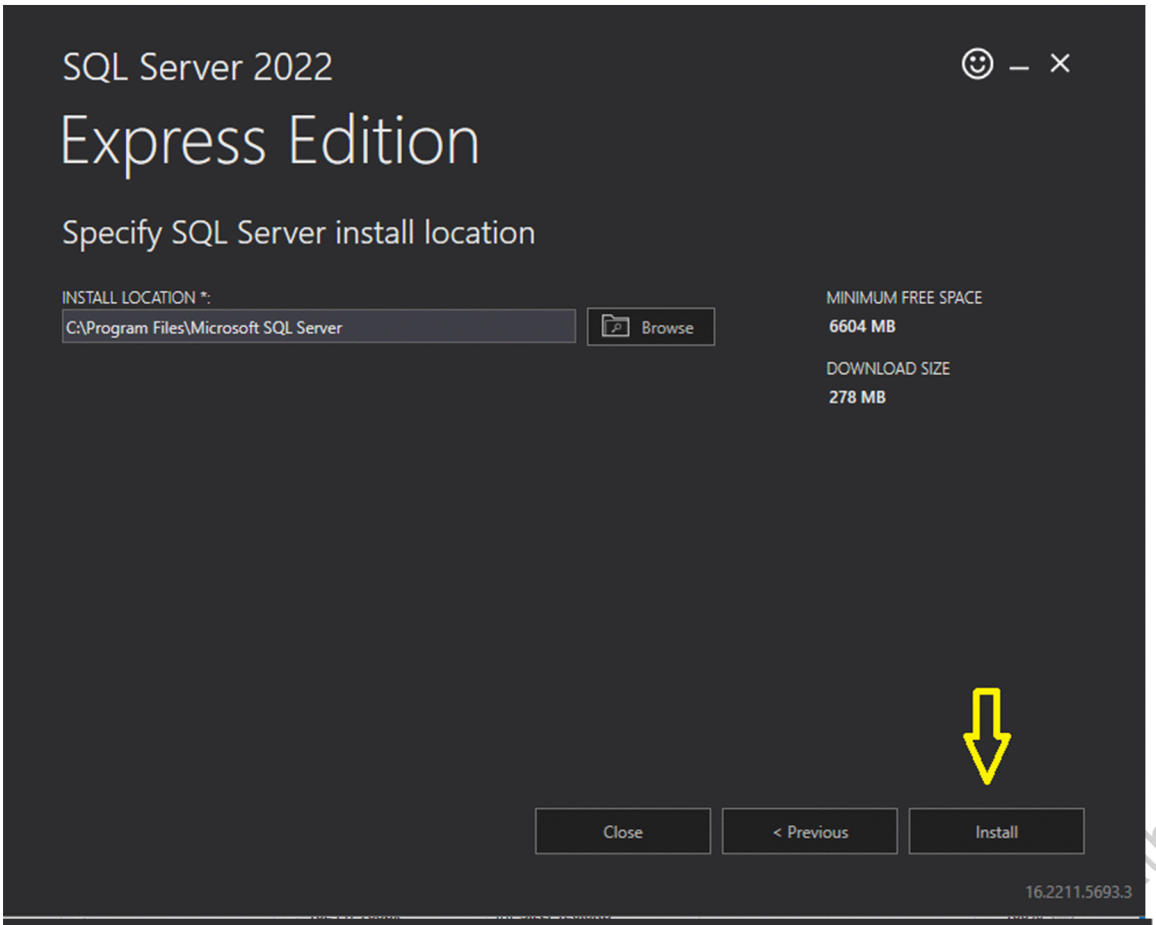
select * from bonus

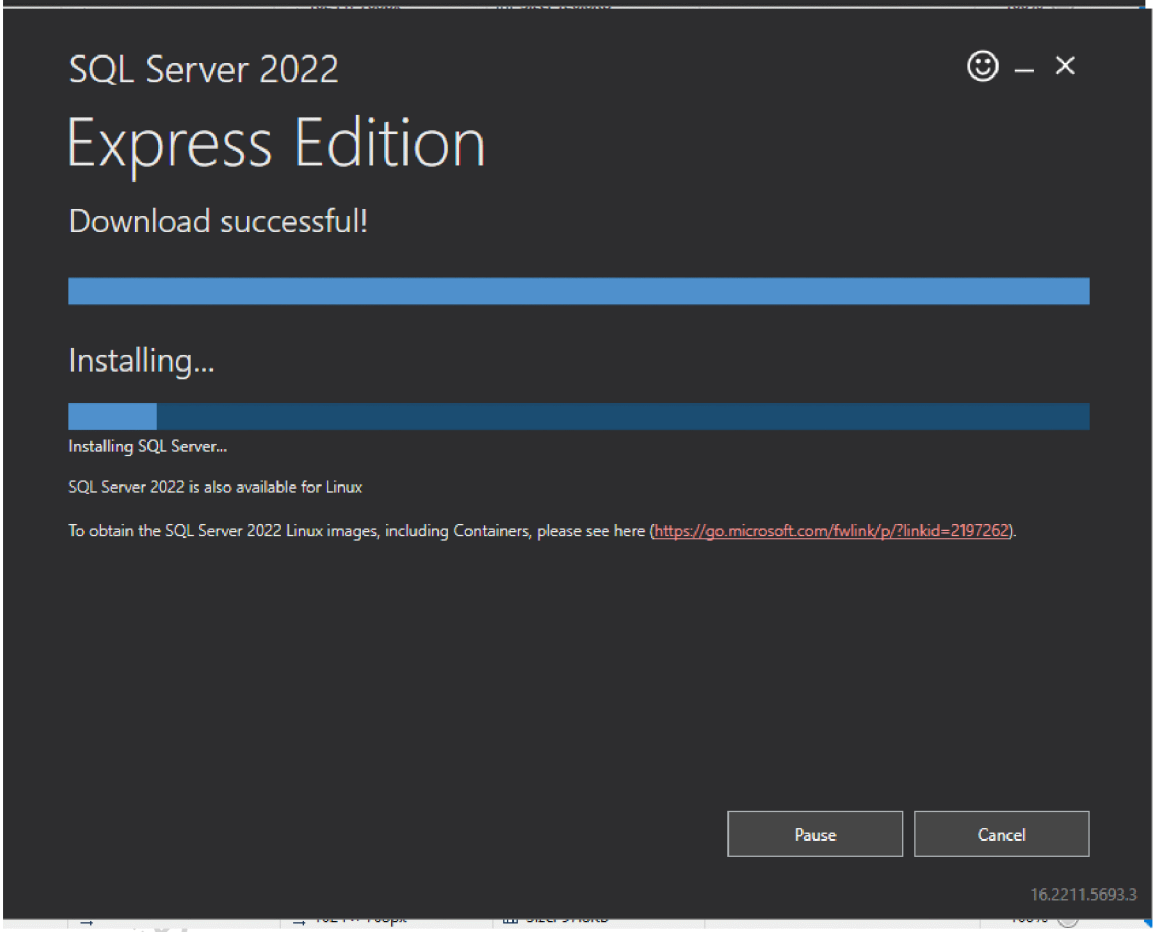
select * from dummy

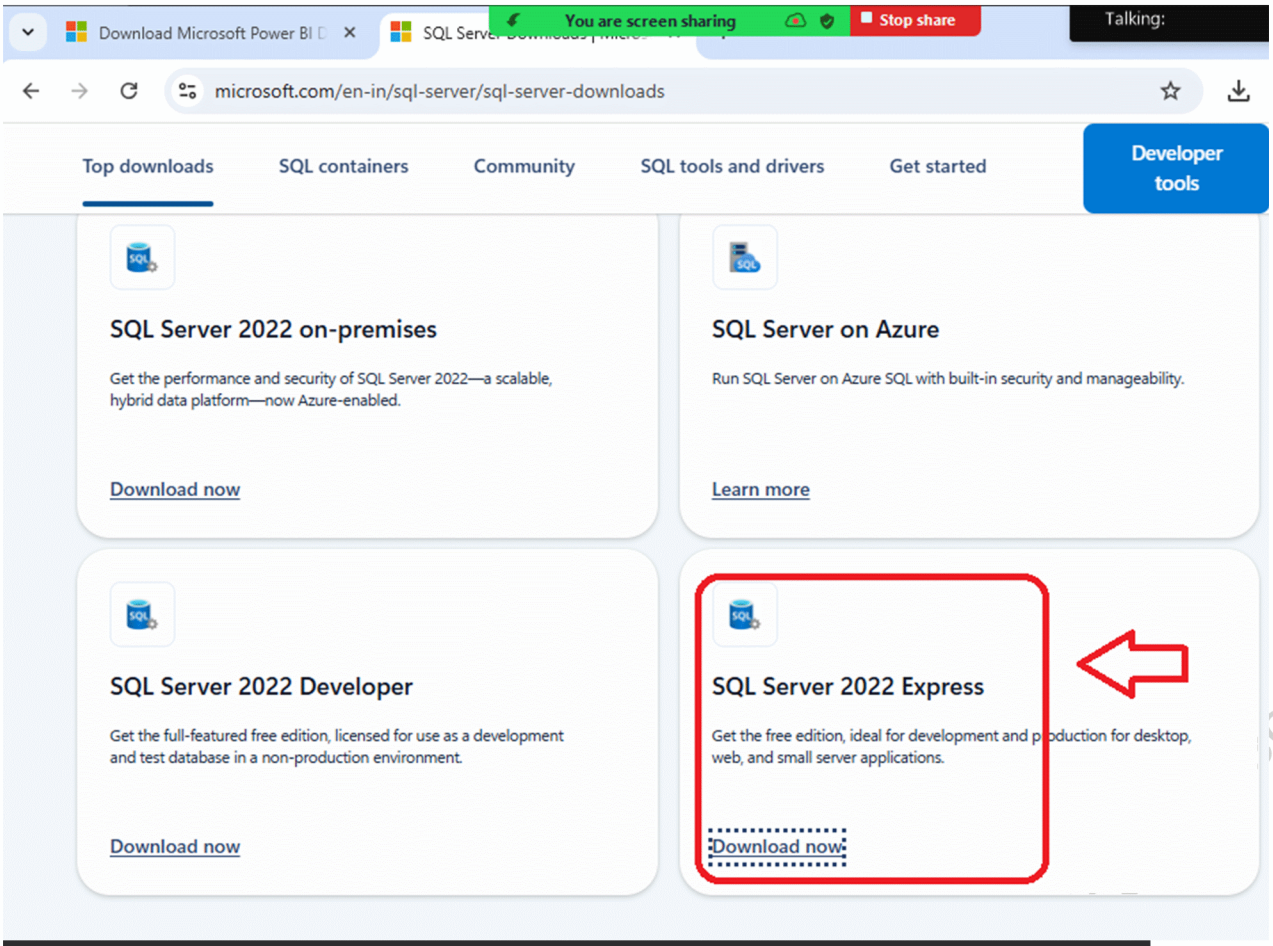
Sql Server – Power BI Integration

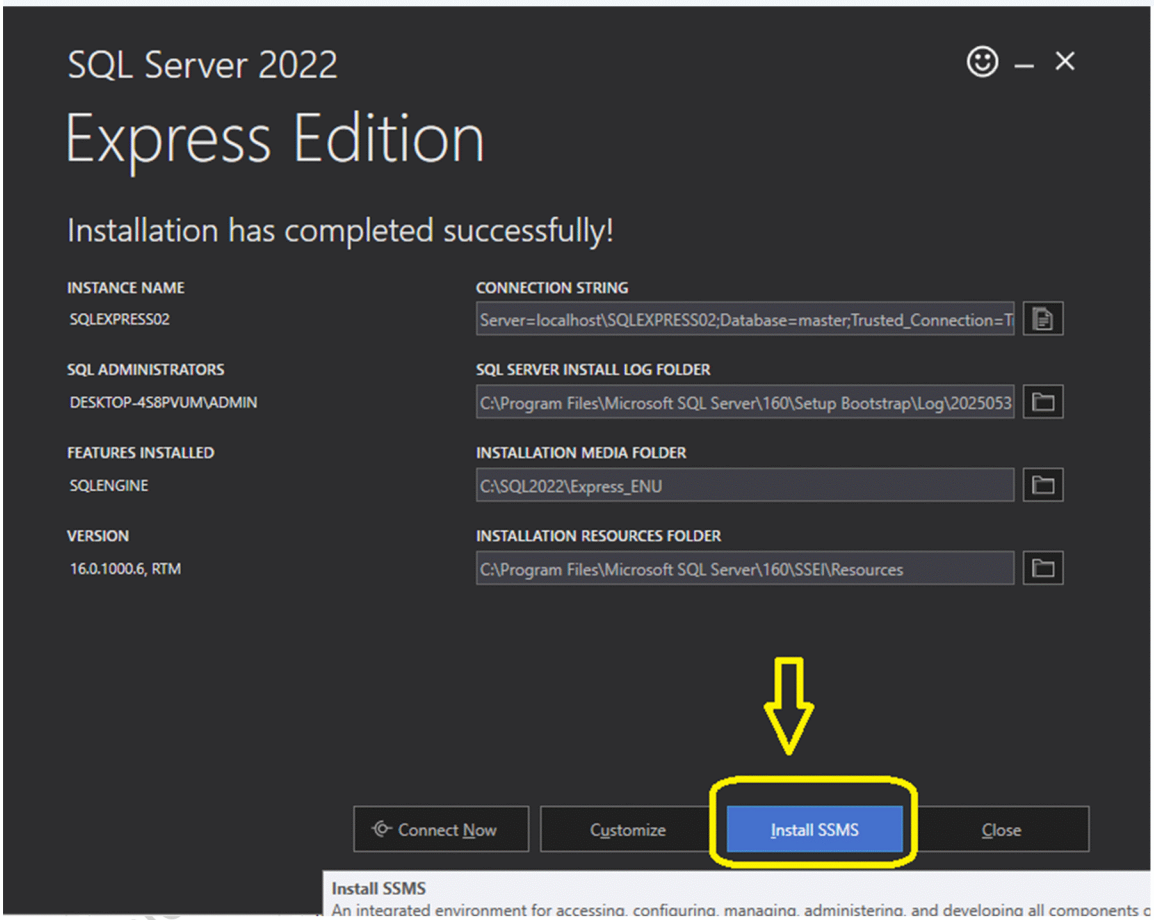









You are screen sharing
Stop share
Talking:
microsoft.com/en-in/sql-server/sql-server-downloads
Top downloads
SQL containers
Community
SQL tools and drivers
Get started
Developer tools
SQL Server 2022 on-premises
Get the performance and security of SQL Server 2022—a scalable, hybrid data platform—now Azure-enabled.
Download now
SQL Server on Azure
Run SQL Server on Azure SQL with built-in security and manageability.
Learn more
SQL Server 2022 Developer
Get the full-featured free edition, licensed for use as a development and test database in a non-production environment.
Download now
SQL Server 2022 Express
Get the free edition, ideal for development and production for desktop, web, and small server applications.
Download now


SQL Server 2022
Express Edition
Installation has completed successfully!
INSTANCE NAME
SQLEXPRESS02
CONNECTION STRING
Server=localhost\SQLEXPRESS02;Database=master;Trusted_Connection=T
SQL ADMINISTRATORS
DESKTOP-4S8PVUM\ADMIN
SQL SERVER INSTALL LOG FOLDER
C:\Program Files\Microsoft SQL Server\160\Setup Bootstrap\Log\2025053
FEATURES INSTALLED
SQLENGINE
INSTALLATION MEDIA FOLDER
C:\SQL2022\Express_ENU
VERSION
16.0.1000.6, RTM
INSTALLATION RESOURCES FOLDER
C:\Program Files\Microsoft SQL Server\160\SSEI\Resources
Connect Now
Customize
Install SSMS
Close
Install SSMS

learn.microsoft.com/en-us/ssms/install/install?redirectedfrom=MSDN

- Free up space. Remove unneeded files and applications from your system drive by, for example, running the Disk Cleanup application.

You can install SQL Server Management Studio 21 side-by-side with other versions. For more information, see Install SQL Server Management Studio versions side-by-side.

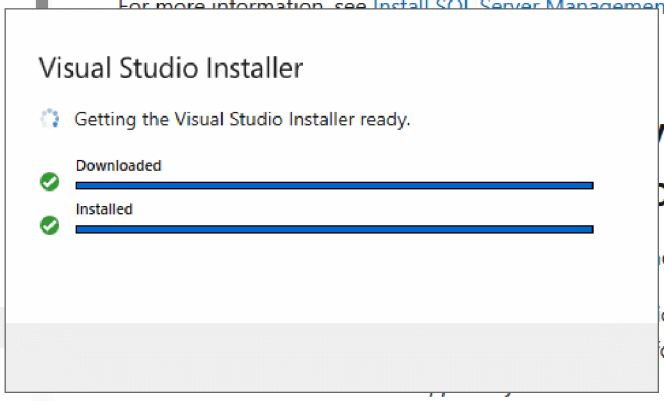
Step 2 - Determine which version of SQL Server Management Studio to install

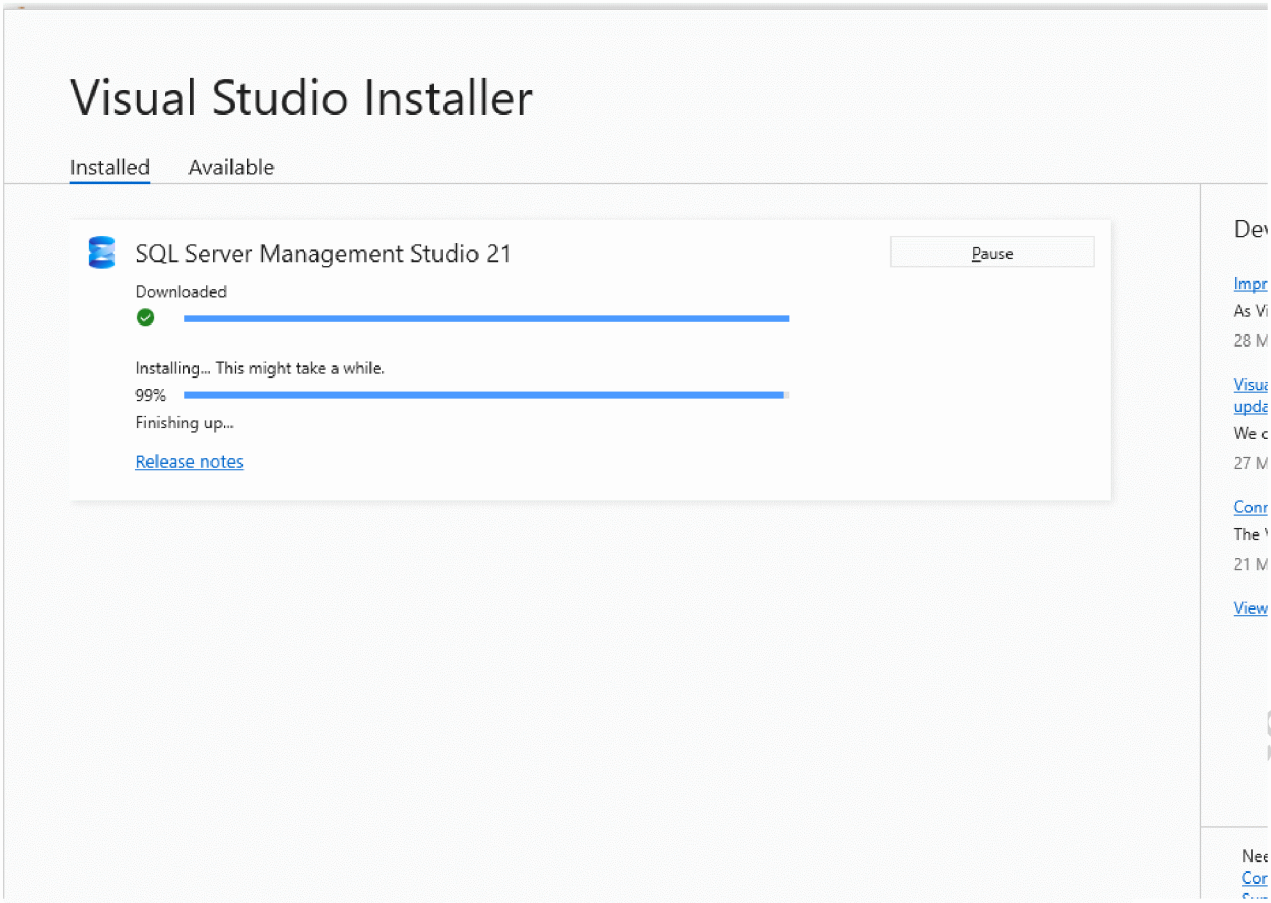
Decide which version of SSMS to install. The most common options are:

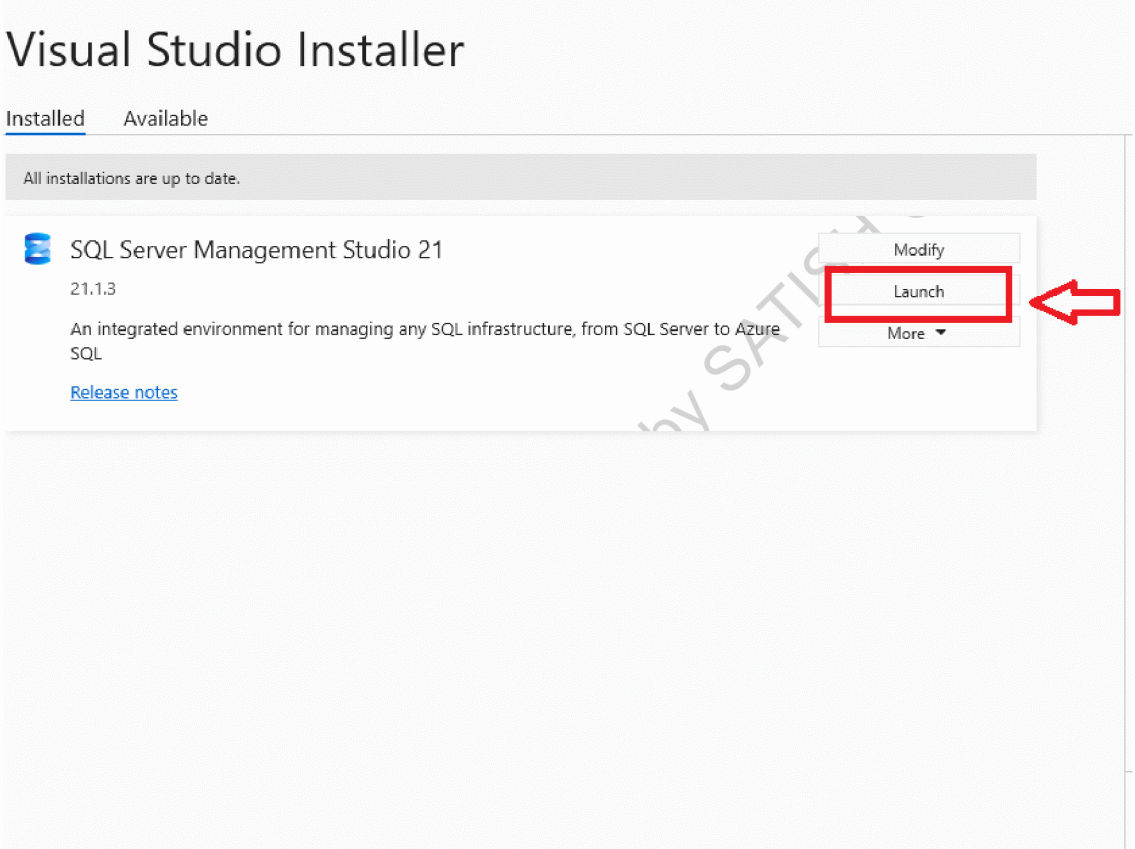
- The latest release of SQL Server Management Studio 21 that is hosted on Microsoft servers. To install this version, select the following link. The installer downloads a small *bootstrapper* to your *Downloads* folder.

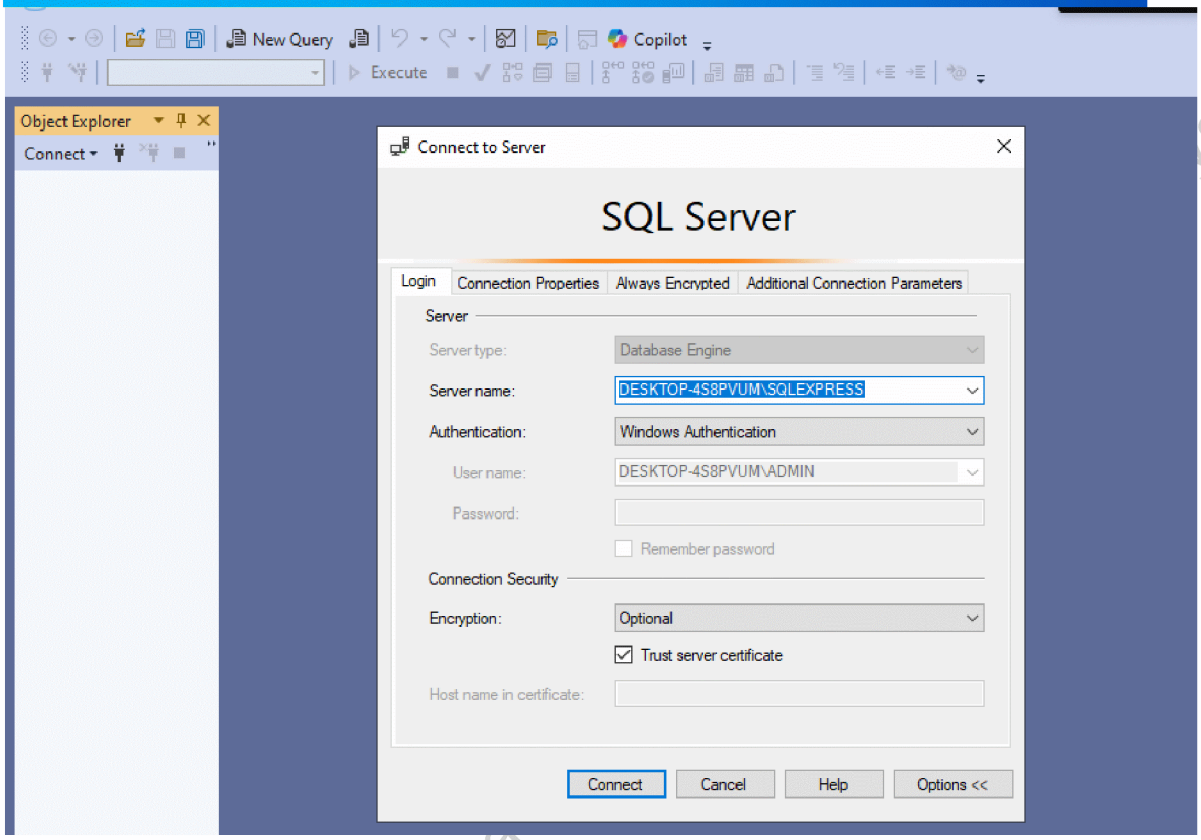
Download SSMS 21

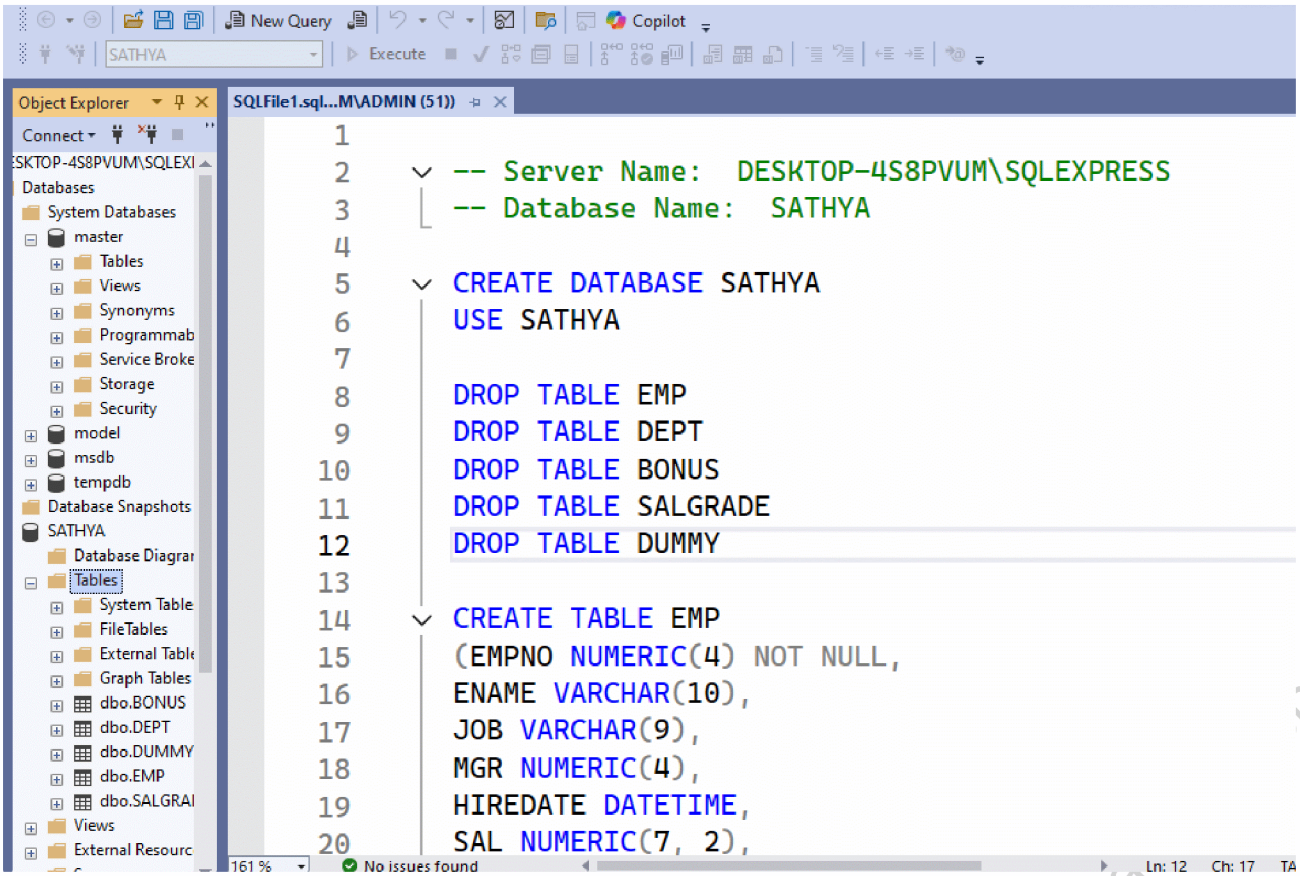
- If you already have SQL Server Management Studio 21 installed, you can install another version alongside it.
- You can download a bootstrapper or installer for a specific version from the Release history page and use it to install another version of SSMS.

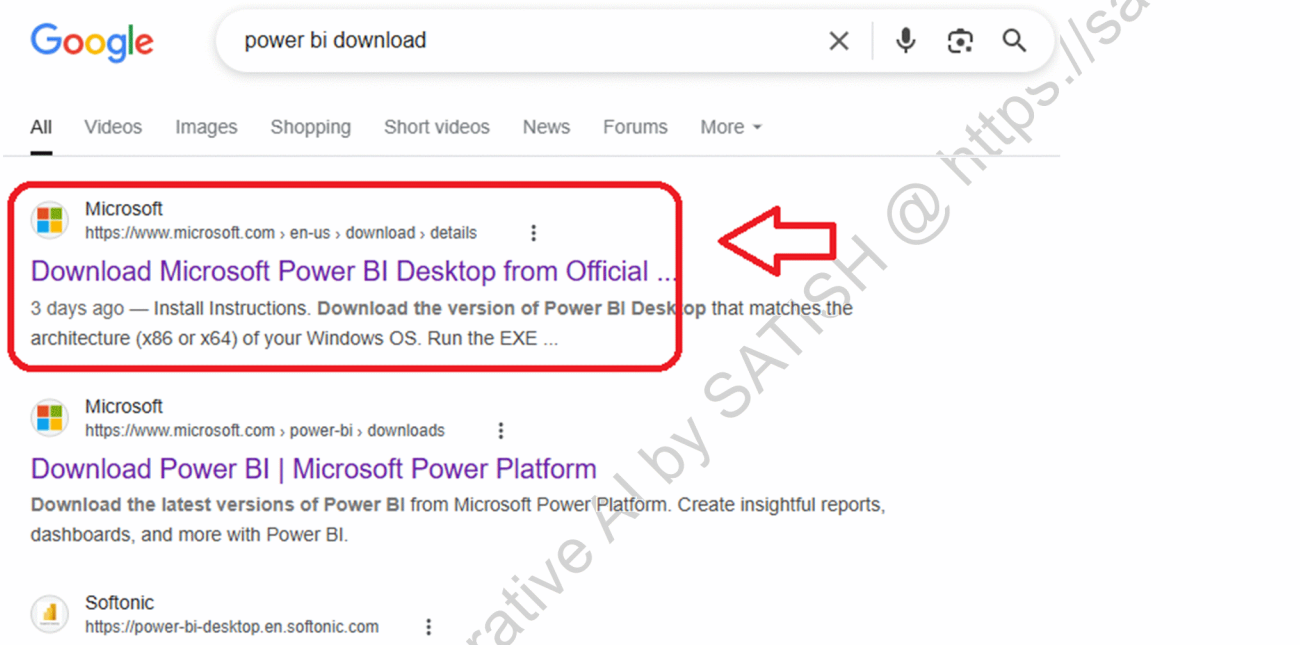


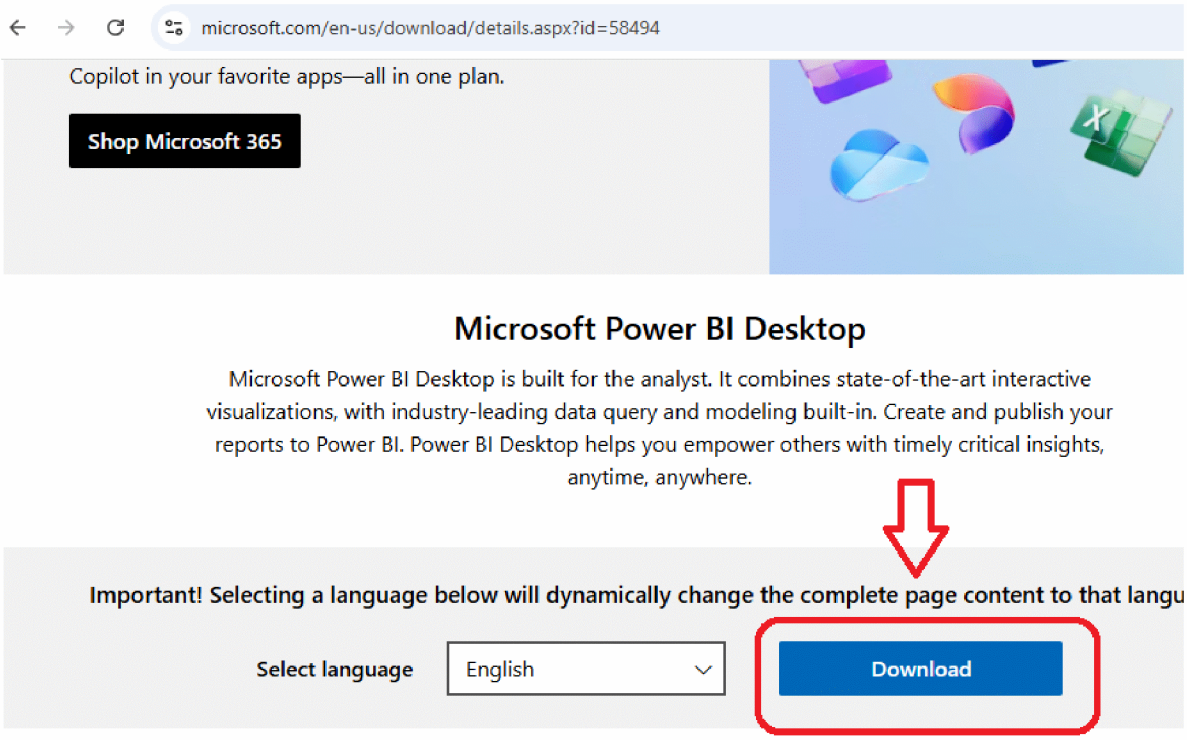


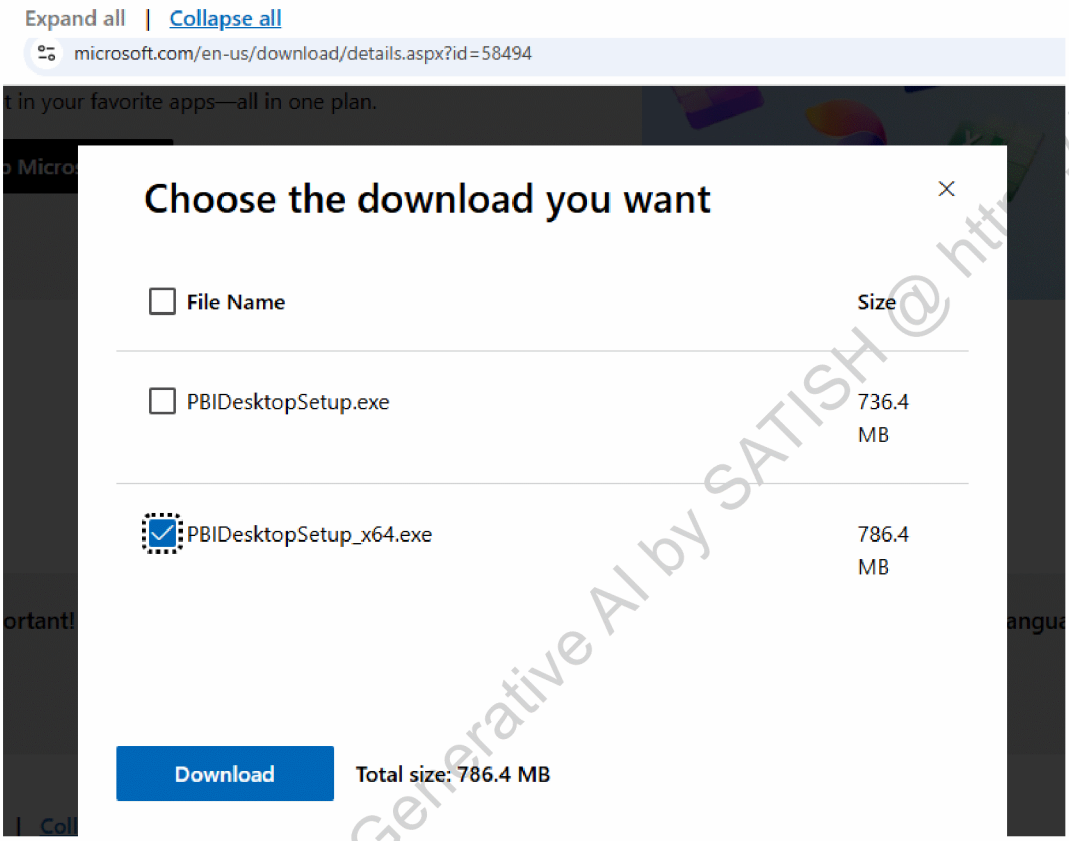


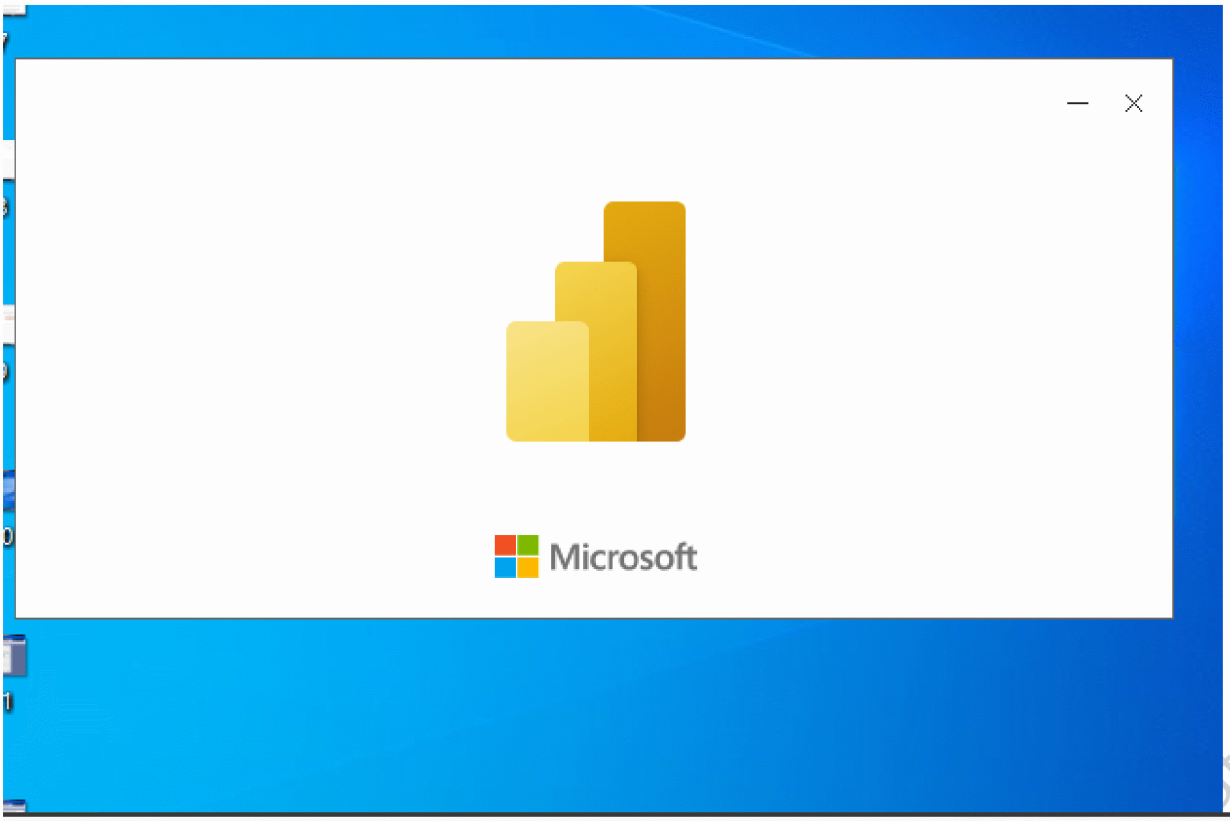


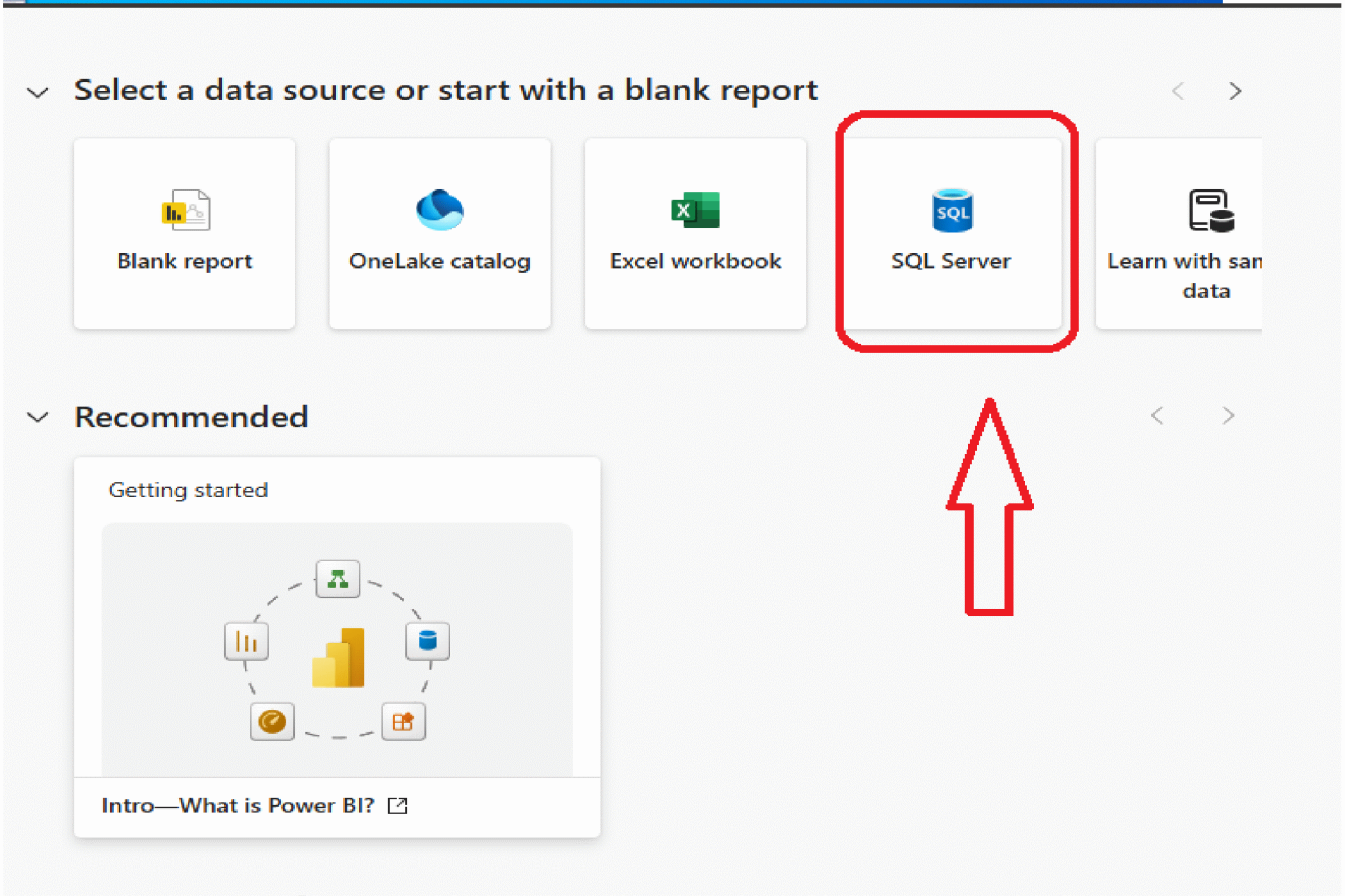


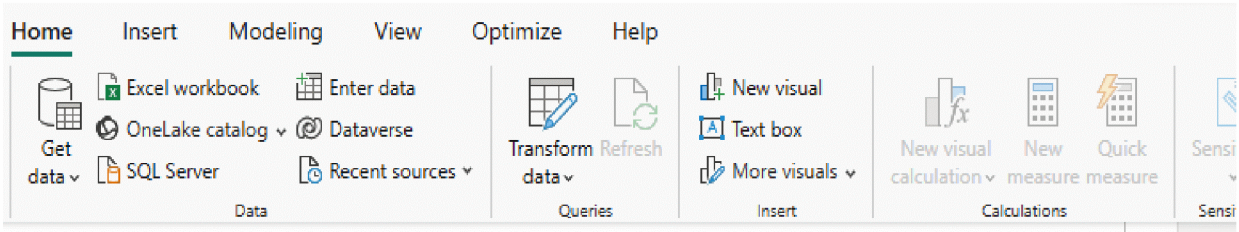


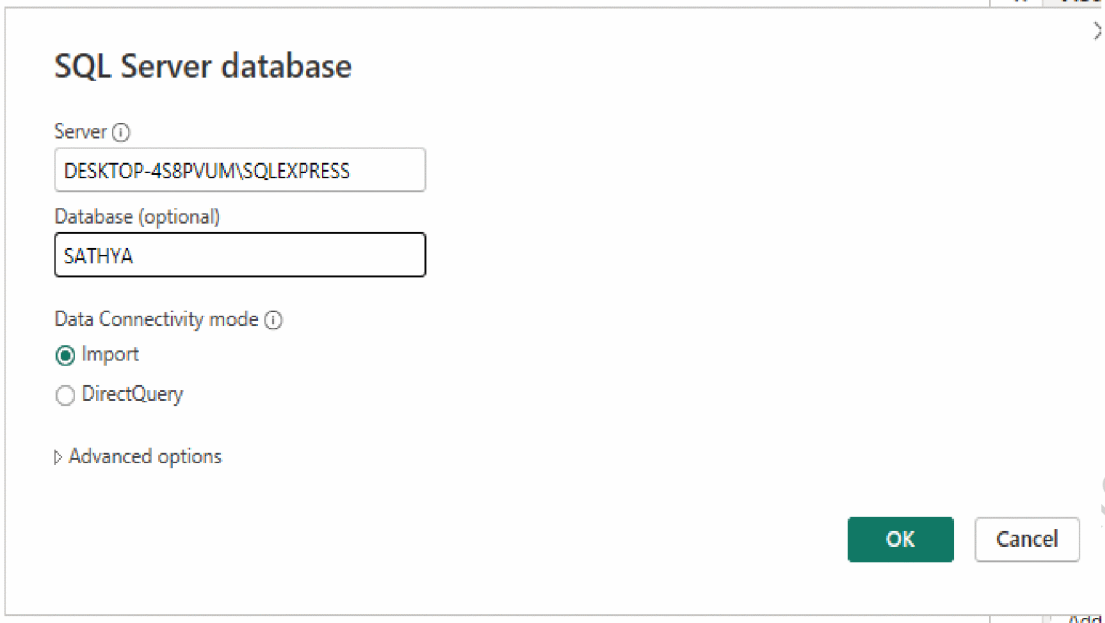


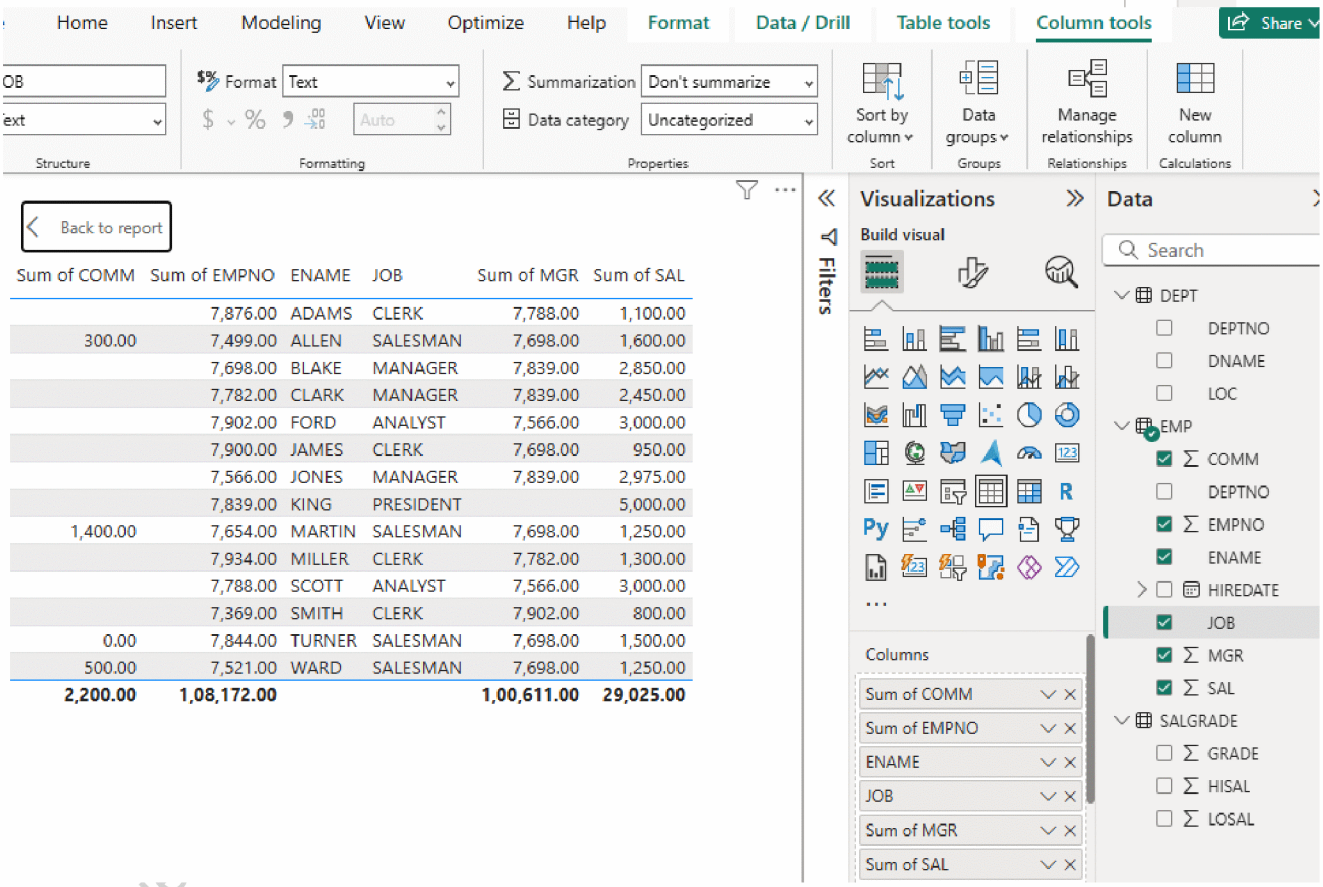












Sum of COMM	Sum of EMPNO	ENAME	JOB	Sum of MGR	Sum of SAL
	7,876.00	ADAMS	CLERK	7,788.00	1,100.00
300.00	7,499.00	ALLEN	SALESMAN	7,698.00	1,600.00
	7,698.00	BLAKE	MANAGER	7,839.00	2,850.00
	7,782.00	CLARK	MANAGER	7,839.00	2,450.00
	7,902.00	FORD	ANALYST	7,566.00	3,000.00
	7,900.00	JAMES	CLERK	7,698.00	950.00
	7,566.00	JONES	MANAGER	7,839.00	2,975.00
	7,839.00	KING	PRESIDENT		5,000.00
1,400.00	7,654.00	MARTIN	SALESMAN	7,698.00	1,250.00
	7,934.00	MILLER	CLERK	7,782.00	1,300.00
	7,788.00	SCOTT	ANALYST	7,566.00	3,000.00
	7,369.00	SMITH	CLERK	7,902.00	800.00
0.00	7,844.00	TURNER	SALESMAN	7,698.00	1,500.00
500.00	7,521.00	WARD	SALESMAN	7,698.00	1,250.00
2,200.00	**1,08,172.00**			**1,00,611.00**	**29,025.00**