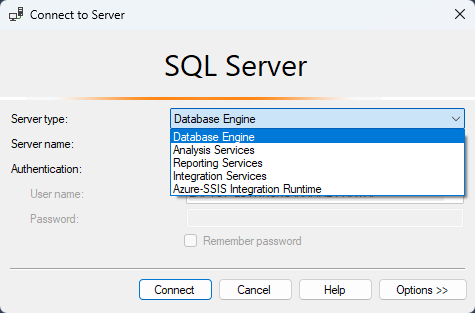
**What is SQL Server**

The SQL Server is a relational database management system (RDBMS) which is developed by Microsoft. It is also called MS SQL Server. This product is built on the basic function of storing and retrieving data as required by other applications. It can be run either on the same computer or on another across a network.

## **Connecting to SQL Server using SQL Server Management Studio**

In this article, I am going to discuss how to connect to **SQL Server** **Database using SQL Server Management Studio** (**SSMS**). The most important thing that you need to keep in mind is that both SQL Server and SSMS are two different things. Here SQL Server is the database and SQL Server Management Studio (SSMS) is the tool that can be used by a user to write and execute SQL queries against the SQL Server Database.

##### ****1.Server Type****



###### **Database Engine:**

1. The database engine is the core service of SQL Server which will use for storing a large amount of data, accessing the data, manipulating the data and providing security to the information.
2. In the database engine, the data will be stored in the form of a 2-D format (Tables).

###### **Analysis Services (SSAS):**

1. The SSAS (SQL Server Analysis Service) is a tool that is used under the data warehousing/ data mining environment for storing the information in the form of a 3-D format.

###### **Reporting Services (SSRS):**

1. SQL Server Reporting Service (SSRS) is a tool that is used to generate various reports such as MS-Word file format, MS-Excel format, .pdf format, XML format, .tiff file format et.
2. The report is a document that is used to store some business-related information.

###### **Integration Services (SSIS):**

1. The SQL Server Integration Service (SSIS) is a tool that is used to convert one database tables into another database understandable format. For example, SQL Server database Tables are converting into Oracle understandable table format.

**2.Server Name**

The server name is nothing but the name of the SQL Server or IP address of SQL Server. One more thing that you need to remember is if the SQL Server is installed on your machine, then you can specify the server name as a dot (.) or 127.0.0.1 or local.

##### 3. ****Authentication****

In SQL Server there are two types of authentication i.e. Windows Authentication and SQL Server Authentication. But it will depend on how you installed the SQL Server. That means, at the time of installation, if you select the mixed-mode authentication then you will get both Windows and SQL Server authentication to connect with the SQL Server database or else you will only have windows authentication to connect with the database.

###### **Windows Authentication:**

1. It is the default authentication mode of SQL Server.
2. In Windows authentication, we will work on user admin.
3. With Windows authentication mode there is no need to enter the user credentials i.e. user Id and password because User Id and Password are generated by the Operating System by default,

###### **SQL Server Authentication:**

1. In SQL Server authentication we will work on the current user.
2. When we will work with SQL Server authentication we should enter user Id and Password (This user Id and password are created by the user at the time of SQL Server software installation).

**Different Types of Database in SQL Server.**

In SQL Server we are going to interact with 2 types of databases such as

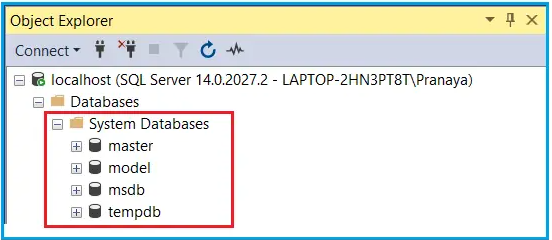
1. **System databases**
2. **User databases**

**System Databases in SQL Server:**

The databases which are created and managed by the SQL Server itself called System databases. SQL Server has four system databases as shown in the below image.

s of each of the above databases.

**Master database:** This database is used to store all system-level information such as system id, culture, server id no, server version, server culture, etc  
**Model database:** The model database will act as a template for creating new databases under a server environment.  
**Msdb (Microsoft database):** Microsoft database will store jobs and alerts information i.e. backup file information.  
**Tempdb database:** It is a temporary database location that is allocated by the server when the user connected to the SQL Server for storing temporary table information.



**Master database:** This database is used to store all system-level information such as system id, culture, server id no, server version, server culture, etc

**Model database:** The model database will act as a template for creating new databases under a server environment.

**Msdb (Microsoft database):** Microsoft database will store jobs and alerts information i.e. backup file information.

**Tempdb database:** It is a temporary database location that is allocated by the server when the user connected to the SQL Server for storing temporary table information.

**SQL Sub Languages:**

SQL contains the following sublanguages

1. **DDL** (5 commands- create, alter, sp\_rename, drop, truncate)
2. **DML** (3 commands- Insert, Update, Delete).
3. **DQL/ DRL** (1 command- select).
4. **TCL** (3 commands- commit, rollback, savepoint)
5. **DCL** (2 commands- Grant, Revoke).

**What are the differences between Delete and Truncate Command in SQL Server?**

|  |  |
| --- | --- |
| **Delete** | **Truncate** |
| It is a DML command. | It is a DDL command |
| By using the delete command we can delete a specific record from the table. | But it is not possible with truncate command. |
| Delete supports WHERE clause. | Truncate does not support the WHERE clause |
| It is a temporary deletion | It is a permanent deletion |
| Delete supports rollback transactions for restoring the deleted data. | Truncate doesn’t support rollback transaction so that we cannot restore the deleted information |
| Delete command will not reset identity property. | But it will reset the identity property |

##### ****How to create SQL Server Database using Query?****

Create database DatabaseName

Whether we create a database graphically using the designer window or using a query, the following 2 files get generated.

**.MDF file:** Master Data File (Contains actual data). This file will store all Tables data and will be saved with an extension of .mdf (master data file)

**.LDF file:**Transaction Log file (Used to recover the database). This file will store transaction Query information (insert, update, delete. Create, etc) and saved with an extension of .ldf (log data file

##### ****Rename a database in SQL Server?****

Alter database DatabaseName Modify Name = NewDatabaseName

Execute SP\_RENAMEDB 'OldDatabaseName','NewDatabaseName'

##### ****Drop a database in SQL Server?****

Drop Database DatabaseName

##### ****Create Table in SQL Server?****

CREATE TABLE student

(

studid INT,

sname VARCHAR (max),

salary DECIMAL (6, 2)

)

##### ****Change the width of a column****

ALTER TABLE Student ALTER COLUMN Name VARCHAR(100)

##### ****Changing the data type column.****

ALTER TABLE Student ALTER COLUMN Name NVARCHAR(100)

##### ****Adding new column in table****

ALTER TABLE Student ADD Branch VARCHAR(20)

##### ****Delete column in table****

ALTER TABLE Student DROP COLUMN Branch

##### ****Rename Table Name****

SP\_RENAME 'OLD TABLE NAME','NEW TABLE NAME'

##### ****Drop a table in SQL Server?****

Drop Table DatabaseName

##### ****Constraint in SQL Server****

1. **Default Constraint**
2. **UNIQUE KEY constraint**
3. **NOT NULL constraint**
4. **CHECK KEY constraint**
5. **PRIMARY KEY constraint**
6. **FOREIGN KEY constraint.**

##### ****Understanding NULL in SQL Server****

NULL represents the absence of data or value. It’s neither ZERO nor EMPTY.

##### ****Default Constraint in SQL Server****

Default constraints enable the SQL Server to insert a default value to a column when the user doesn’t specify a value.

##### ****Check Constraint in SQL Server****

The Check Constraint is used to enforce domain integrity. Domain integrity ensures that the values going to store in a column must follow some defined rules such as range, type, and format.

CREATE TABLE Employee (

Id int NOT NULL,

Name varchar(50) NOT NULL,

City varchar(255) DEFAULT 'Mumbai',

DateOfBirth date DEFAULT GETDATE(),

Salary DECIMAL (18, 2) DEFAULT 5000.00

Emp\_id INT NOT NULL CHECK(Emp\_id BETWEEN 0 AND 1000),

Entered\_date DATETIME NOT NULL CHECK(Entered\_date <= CURRENT\_TIMESTAMP),

Dept\_no INT CHECK(Dept\_no > 0 AND Dept\_no < 100)

)

##### ****Unique Constraint in SQL Server****

The UNIQUE Constraint prevents two records from having identical values in a column. UNIQUE constraint will accept only 1 NULL value.

CREATE TABLE Customer

(

Id INT UNIQUE,

Name VARCHAR(30) UNIQUE,

Emailid VARCHAR(100) UNIQUE

)

CREATE TABLE Customer

(

Id INT,

Name VARCHAR(30),

Emailid VARCHAR(100),

CONSTRAINT id\_unique UNIQUE(Id),

CONSTRAINT name\_unique UNIQUE(Name),

CONSTRAINT email\_unique UNIQUE(Emailid)

)

##### ****Primary Key in SQL Server****

The **Primary Key in SQL Server** is the combination of **Unique** and **Not Null**Constraint**.**That means it will not allow either **NULL** or **Duplicate** values into a column or columns on which the primary key constraint is applied. Using the primary key we can enforce entity integrity i.e. using the primary key value we should uniquely identify a record.

CREATE TABLE Branches

(

Bcode INT PRIMARY KEY,

Bname VARCHAR(40),

Bloc CHAR(40) PRIMARY KEY

)

When we try to execute the above query, it will give us the error as: “**Cannot add multiple PRIMARY KEY constraints to table ‘Branches’**.”

To overcome the above drawback, rewrite the query like below

CREATE TABLE Branches

(

Bcode INT PRIMARY KEY,

Bname VARCHAR(40),

Bloc CHAR(40)

)

##### ****Composite Primary key in SQL Server****

It is also possible in SQL Server to create the Primary Key constraint on more than one columns and when we do so, it is called a Composite Primary Key. The maximum number of columns is including in the composite primary key is 16 columns.

CREATE TABLE BranchDetails

(

City VARCHAR(40),

Bcode INT,

Bloc VARCHAR(30),

PRIMARY KEY(City, Bcode)

)

##### ****Foreign Key Constraint in SQL Server****

A foreign key in one TABLE points to a primary key or unique key in another table. The foreign key constraints are used to enforce referential integrity.

CREATE TABLE Dept

(

Dno INT PRIMARY KEY,

Dname VARCHAR(30),

Dlocation CHAR(40)

)

INSERT Dept VALUES (10, '.NET', 'HYD')

INSERT Dept VALUES (20, 'JAVA', 'PUNE')

INSERT Dept VALUES (30, 'PHP', 'DELHI')

Select \* from Dept

CREATE TABLE Employee

(

Eid INT PRIMARY KEY,

Ename VARCHAR(30),

Salary MONEY,

Dno INT FOREIGN KEY REFERENCES Dept(Dno)

)

INSERT into Employee VALUES (101,'AA', 25000, 10) -- Allowed

INSERT into Employee VALUES (102,'BB', 32000, 20) -- Allowed

INSERT into Employee VALUES (103,'CC', 52000, 40) -- Not Allowed

##### Foreign key Constraint in SQL Server at table level

CREATE TABLE Employee

(

Empid INT,

Ename VARCHAR(40),

Job VARCHAR(30),

Salary MONEY,

Deptno INT,

CONSTRAINT deptno\_fk FOREIGN KEY (Deptno) REFERENCES Dept(Dno)

)

##### ****Cascading Referential Integrity Constraints in SQL Server****

In the Microsoft SQL server if we want to delete any record or column from one table but that record or column is a foreign key for another table then we will get the error to solve this problem we use **Cascading referential integrity constraint.**

###### **SET NULL:**

If a user tries to delete or update statement(s) that will affect rows in the foreign key table, then those values will be set to NULL when the primary key record is deleted or updated in the Primary key table. The important thing that we need to keep in mind that the foreign key columns affected must allow NULL values.

###### **CASCADE:**

If a user tries to delete the statement(s) which will affect the rows in the foreign key table, then those rows will be deleted when the primary key record is deleted. Similarly, if an update statement affects rows in the foreign key table, then those rows will be updated with the value from the primary key record after it has been updated.

###### **SET DEFAULT:**

If a delete or update statement affects rows in a foreign key table, then all rows containing those foreign keys are set to the default value. All foreign key columns in the related table must have default constraints defined on them.

###### **NO ACTION:**

This is the default action that SQL Server performs. This specifies that if an update or deletes statement affects rows in foreign key tables, then the action will be denied and rolled back. An error message will be raised.

CREATE TABLE Gender

(

Id INT PRIMARY KEY,

Gender NVARCHAR(50)

)

-- Insert some test data in Gender Table

Insert into Gender values (1, 'Male')

Insert into Gender values (2, 'Female')

Insert into Gender values (3, 'Unknown')

CREATE TABLE [Person](

[Id] [int] PRIMARY KEY,

[Name] [varchar](100) NOT NULL,

[Email] [varchar](100) NOT NULL,

[GenderID] [int] NULL

)

--Insert some test data in Person Table

Insert into Person values (1, 'abc','abc@gmail.com',1)

Insert into Person values (2, 'pqr','pqr@gmail.com',2)

Insert into Person values (3, 'xyz','xyz@gmail.com',3)

-- Add a foreign key reference using query

Alter table Person

add constraint Person\_GenderId\_FK FOREIGN KEY (GenderId) references Gender(ID)

DELETE FROM Gender WHERE Id = 2 --Not Allowed

##### ****Use Of Delete Cascase/Update Cascade****

CREATE TABLE [Person](

[Id] [int] PRIMARY KEY,

[Name] [varchar](100) NOT NULL,

[Email] [varchar](100) NOT NULL,

[GenderID] [int] CONSTRAINT FK\_Person\_GenderID FOREIGN KEY REFERENCES dbo.Gender(Id)

ON DELETE CASCADE

ON UPDATE CASCADE

)

DELETE FROM Gender WHERE Id = 2 --Allowed

**What is Identity in SQL Server?**

The Identity in SQL Server is a property that can be applied to a column of a table whose value is automatically created by the server. So, whenever you marked a column as identity, then that column will be filled in an auto-increment way by SQL Server. That means as a user we cannot insert a value manually into an identity column.

1. **Seed:** Starting value of a column. The default value is 1.
2. **Increment:**It specifies the incremental value that is added to the identity column value of the previous row. The default value is 1.

Create Table Person

(

PersonId int identity(1, 1),

Name nvarchar(20)

)

**How to explicitly supply Values for Identity Column in SQL Server?**

To explicitly supply a value for the identity column

1. First, turn on identity insert – SET Identity\_Insert Person ON
2. Secondly, you need to specify the identity column name in the insert query as shown below

Insert into Person(PersonId, Name) values(3, 'Sara')

So once we filled the gaps in the identity column, and if we wish the SQL server to calculate the value, turn off **Identity\_Insert** as shown below.

SET Identity\_Insert Person OFF

##### ****How to Reset the Identity Column Value in SQL Server?****

If you have deleted all the rows in a table, and you want to reset the identity column value, then you need to use the **DBCC CHECKIDENT** command. This command will reset the identity column value.

**Syntax:**DBCC CHECKIDENT(TableName, RESEED, 0)

##### ****Get the last generated identity column value in SQL Server****

Select SCOPE\_IDENTITY()

Select @@IDENTITY

Select IDENT\_CURRENT('Person')

1. **SCOPE\_IDENTITY():**The SCOPE\_IDENTITY() built-in function returns the last identity column value that is created within the same session and the same scope.
2. **@@IDENTITY:**The @@IDENTITY() built-in function returns the last identity column value that is created in the same session but with any scope.
3. **IDENT\_CURRENT(‘TableName’):**The IDENT\_CURRENT() built-in function returns the last identity column value that is created for a specific table across any session and any scope.

##### ****What is a Sequence Object in SQL Server?****

A sequence is an object in SQL Server that is used to generate a number sequence. This can be useful when we need to create a unique number to act as a primary key.

A sequence is an object in SQL Server that is used to generate a number sequence. This can be useful when we need to create a unique number to act as a primary key.

The Sequence Object is one of the new features introduced in SQL Server 2012. A sequence is a user-defined object and as its name suggests it generates a sequence of numeric values according to the properties with which it is created.

CREATE SEQUENCE [dbo].[SequenceObjectIncrement]

AS INT

START WITH 1

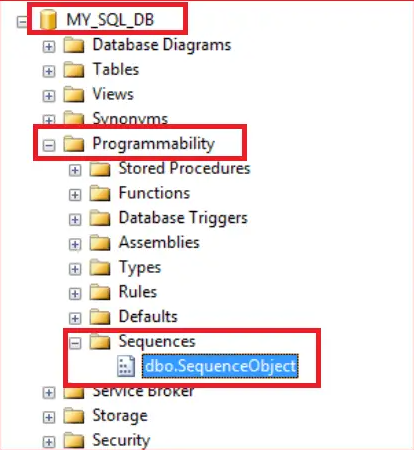
INCREMENT BY 1

CREATE SEQUENCE [dbo].[SequenceObjectDecrement]

AS INT

START WITH 100

INCREMENT BY -1



**Properties of Sequence Object:**

1. **DataType:**Built-in integer type (tinyint, smallint, int, bigint, decimal, etc…) or user-defined integer type. The default is bigint.
2. **START WITH:** The Start With Value is nothing but the first value that is going to be returned by the sequence object
3. **INCREMENT BY:** The Increment by value is nothing but the value to increment or decrement by the sequence object for each row. If you specify a negative value then the value is going to be decrement.
4. **MINVALUE:** It specifies the value for the sequence object
5. **NO MINVALUE:**It specifies that there is no minimum value specified for the given sequence object.
6. **MAXVALUE:** Maximum value for the sequence object
7. **NO MAXVALUE:**It means that there is no maximum value specified for the sequence.
8. **CYCLE:** It specifies that reset the sequence object when the Sequence Object reached the maximum or minimum value.
9. **NO CYCLE:**When you specify the No Cycle option, then it will throw an error when the Sequence Object reached its maximum or minimum value.
10. **CACHE:** Cache sequence values for performance. The default value is CACHE.
11. **NO CACHE:**As the name says, if you specify the NO CACHE option then it will not cache the sequence numbers.

##### ****How to Generate the Next Sequence Value in SQL Server?****

SELECT NEXT VALUE FOR [dbo].[SequenceObject]

**Output:** 1

CREATE TABLE Employees

(

Id INT PRIMARY KEY,

Name NVARCHAR(50),

Gender NVARCHAR(10)

)

-- Generate and insert Sequence values

INSERT INTO Employees VALUES

(NEXT VALUE for [dbo].[SequenceObject], 'Ben', 'Male')

##### ****Recycling the Sequence values in SQL Server?****

When the sequence has reached its maximum value, and if we want to restart from the minimum value, set CYCLE option

ALTER SEQUENCE [dbo].[SequenceObject]

INCREMENT BY 10

MINVALUE 100

MAXVALUE 150

CYCLE

##### ****Drop Sequence object****

Once we have created our sequence in SQL Server, we might find that we need to remove it from the database.

**Syntax:**DROP SEQUENCE sequence\_name

**Associate Sequence object to a table**

CREATE TABLE dbo.Customer (

ID INT DEFAULT(NEXT VALUE FOR DBO.SequenceObject),

Name VARCHAR(100)

)

INSERT INTO dbo.Customer(Name)

VALUES('Pranaya Kumar')

##### ****Group By Clause in SQL Server****

The Group by Clause in SQL Server is used to divide similar types of records or data as a group and then return. If we use group by clause in the query then we should use grouping/aggregate function such as count(), sum(), max(), min(), and avg() functions.

SELECT Department, COUNT(\*) AS TotalEmployee

FROM Employee

GROUP BY Department

##### ****What is Having Clause and its need in SQL Server?****

The **Having Clause in SQL Server** is used for restricting or you can say filtering the data just like the where clause in SQL Server. So, the Having Clause in SQL Server is an additional filter that is applied to the result set. Logically, the having clause filters the rows from the intermediate result set that is built by using the FROM, WHERE, or GROUP BY clauses in the SELECT statement.

##### ****What are Top n Clause and its use in SQL Server?****

The **Top n Clause in SQL Server** is used to specify the number of data rows to return. In large tables with thousands or millions of data rows, it takes more time to return all the records, which cause database performance issue.

SELECT TOP(3)

ID, Name, EmailID, Gender, CITY, Department

FROM Employee

WHERE Gender = 'Male'

ORDER BY ID;

SELECT TOP (70) PERCENT

ID, Name, EmailID, Gender, CITY, Department

FROM Employee

WHERE Gender = 'Male'

ORDER BY ID;

## **Difference Between Where and Having Clause in SQL Server**

The WHERE clause cannot be used with aggregate function whereas the HAVING clause can. The Where clause filters rows before aggregate calculations are performed whereas the HAVING clause filters rows after aggregate calculations are performed. Let us understand this with an example.

## **UNION, UNION ALL, INTERSECT and EXCEPT Operators in SQL Server**

1. **UNION**: Combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: Combine two or more result sets into a single set, including all duplicates.
3. **INTERSECT**: Takes the data from both result sets which are in common.
4. **EXCEPT**: Takes the data from the first result set, but not in the second result set (i.e. no matching to each other)

UNION ALL gives better performance in query execution as it does not waste resources on removing duplicate rows.

CREATE TABLE EmployeeIndia

(

ID INT PRIMARY KEY,

Name VARCHAR(50),

Gender VARCHAR(10),

Department VARCHAR(50)

)

GO

INSERT INTO EmployeeIndia VALUES(1, 'Pranaya', 'Male','IT')

INSERT INTO EmployeeIndia VALUES(2, 'Priyanka', 'Female','IT')

INSERT INTO EmployeeIndia VALUES(3, 'Preety', 'Female','HR')

INSERT INTO EmployeeIndia VALUES(4, 'Subrat', 'Male','HR')

INSERT INTO EmployeeIndia VALUES(5, 'Anurag', 'Male','IT')

GO

CREATE TABLE EmployeeUK

(

ID INT PRIMARY KEY,

Name VARCHAR(50),

Gender VARCHAR(10),

Department VARCHAR(50)

)

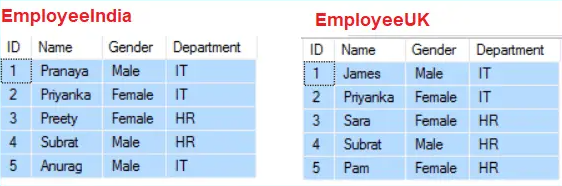
INSERT INTO EmployeeUK VALUES(1, 'James', 'Male','IT')

INSERT INTO EmployeeUK VALUES(2, 'Priyanka', 'Female','IT')

INSERT INTO EmployeeUK VALUES(3, 'Sara', 'Female','HR')

INSERT INTO EmployeeUK VALUES(4, 'Subrat', 'Male','HR')

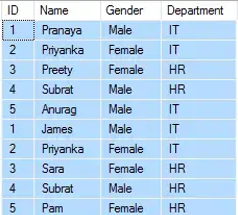
INSERT INTO EmployeeUK VALUES(5, 'Pam', 'Female','HR')



SELECT ID, Name, Gender, Department FROM EmployeeIndia

UNION ALL

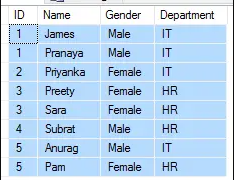
Select ID, Name, Gender, Department FROM EmployeeUK



SELECT ID, Name, Gender, Department FROM EmployeeIndia

UNION

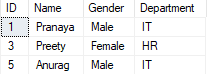
SELECT ID, Name, Gender, Department FROM EmployeeUK



SELECT ID, Name, Gender, Department FROM EmployeeIndia

EXCEPT

SELECT ID, Name, Gender, Department FROM EmployeeUK



SELECT ID, Name, Gender, Department FROM EmployeeIndia

INTERSECT

SELECT ID, Name, Gender, Department FROM EmployeeUK



**How will the database engine retrieve the data from a table in SQL Server?**

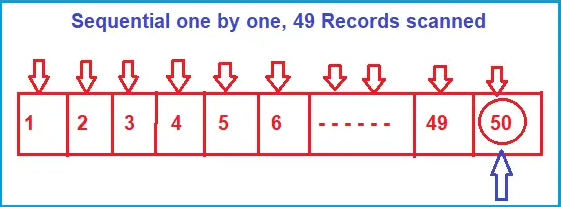
Whenever the database engine wants to retrieve the data from a database table it will adopt two different mechanisms for searching the data.

1. **Table Scan**
2. **Index Seek**

**What is Table Scan in SQL Server?**

In Table Scan, the SQL Server Search Engine will search for the required information sequentially one by one from the start to the last record of the table. If the table has more rows, then it will take more time for searching the required data, so it is a time-consuming process.

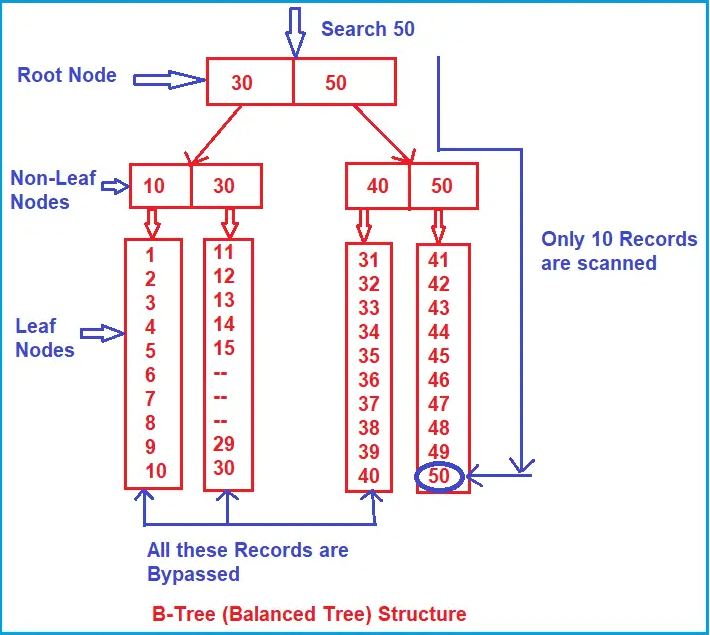
Let us understand how the SQL Server Database Engine searches the data when there is no index available on the table i.e. Table Scan. When there is no index in the table, SQL Server searches the data sequentially.



Suppose, you want to search the value 50, then the search engine (i.e. SQL Server Search Engine) will scan the record sequentially one by one from the beginning i.e. from 1, and until it reaches the value 50. If you want to increase the search performance, then somehow you have to minimize the number of scans. That is what exactly the B-Tree (Balanced Tree) does.

**What is Index Scan/Seek in SQL Server?**

In Index Scan, the SQL Server Search Engine uses a **B-Tree structure** to search the required data which drastically improves the performance of your search query by reducing the number of scans. So, let us first understand what B-Tree structure is and how it reduces the number scan which ultimately improves the performance of your search query.



Suppose, you want to search 50 here, then what will happen internally is, the search engine will start the search from the root node. It will check whether 50 is less than or equal to 30. As 50 is not less than or equal to 30, so the non-leaf nodes and leaf nodes that come under the root node 30 are completely bypassed.

Then it will go to the next node i.e. 50 and check whether 50 is less than or equal to 50. And the condition satisfies here. Then it goes to the non-leaf nodes (40, 50) which are under the root node 50. It will check whether 50 is less than or equal to 40 and the condition fail, so, it will bypass all the leaf nodes which come under the non-leaf node 40. Then it will check the other non-leaf node i.e. 50 and here the condition satisfies as 50 equals 50 and it goes to scan the leaf node sequentially. That is, it approximately scans 10 records.

So, as you can see, due to the Root Node, Non-Leaf Nodes, and Leaf Nodes arrangement, the complete records from 1 to 40 are bypassed.

##### ****What is SQL Server Clustered index?****

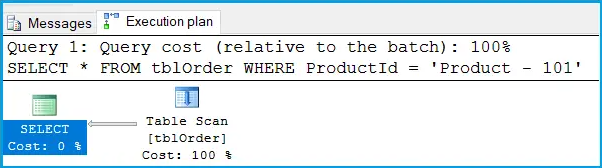
The [**Clustered Index in SQL Server**](https://dotnettutorials.net/lesson/sql-server-clustered-index/) defines the order in which the data is physically stored in a table. In the case of a clustered index,  the leaf node store the actual data. As the leaf nodes store the actual data a table can have only one clustered index. The Clustered Index by default was created when we created the primary key constraint for that table. That means the primary key column creates a clustered index by default.

CREATE CLUSTERED INDEX IX\_Employee\_ID ON Employee(Id ASC);

##### ****What is SQL Server Non-Clustered Index?****

In [**SQL Server Non-Clustered Index**](https://dotnettutorials.net/lesson/sql-server-non-clustered-index/), the arrangement of data in the index table will be different from the arrangement of data in the actual table. The data is stored in one place and the index is stored in another place. Moreover, the index will have pointers to the storage location of the actual data. In SQL Server, a maximum of 999 non-clustered indexes are created per table.

SELECT \* FROM tblOrder WHERE ProductId = 'Product – 101';

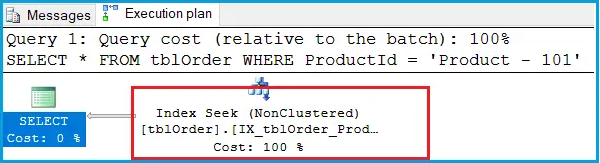


CREATE NONCLUSTERED INDEX IX\_tblOrder\_ProductId

ON dbo.tblOrder (ProductId)

INCLUDE ([Id],[CustomerId],[ProductName])

GO



**Disadvantages of Indexes**

##### **Additional Disk space**

1. **Insert Update and delete statement become Slow**

**Built-in Functions**

CREATE TABLE Employees

(

ID INT,

Name VARCHAR(50),

Department VARCHAR(50),

Salary int

)

INSERT INTO Employees Values (1, 'James', 'IT', 15000)

INSERT INTO Employees Values (2, 'Smith', 'IT', 35000)

INSERT INTO Employees Values (3, 'Rasol', 'HR', 15000)

INSERT INTO Employees Values (4, 'Rakesh', 'Payroll', 35000)

INSERT INTO Employees Values (5, 'Pam', 'IT', 42000)

INSERT INTO Employees Values (6, 'Stokes', 'HR', 15000)

INSERT INTO Employees Values (7, 'Taylor', 'HR', 67000)

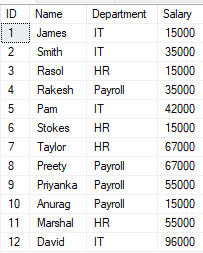
INSERT INTO Employees Values (8, 'Preety', 'Payroll', 67000)

INSERT INTO Employees Values (9, 'Priyanka', 'Payroll', 55000)

INSERT INTO Employees Values (10, 'Anurag', 'Payroll', 15000)

INSERT INTO Employees Values (11, 'Marshal', 'HR', 55000)

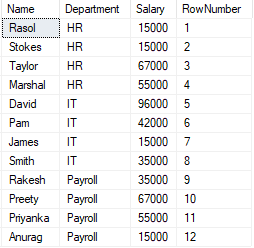
INSERT INTO Employees Values (12, 'David', 'IT', 96000)



SELECT Name, Department, Salary,

ROW\_NUMBER() OVER (ORDER BY Department) AS RowNumber

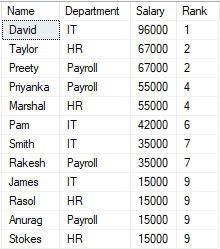
FROM Employees



SELECT Name, Department, Salary,

RANK() OVER (ORDER BY Salary DESC) AS [Rank]

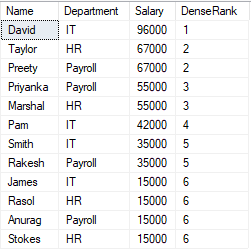
FROM Employees



SELECT Name, Department, Salary,

DENSE\_RANK() OVER (ORDER BY Salary DESC) AS [DenseRank]

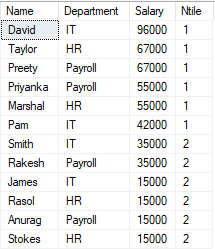
FROM Employees



SELECT Name, Department, Salary,

NTILE(2) OVER (ORDER BY Salary DESC) AS [Ntile]

FROM Employees



##### ****Row\_Number Function with PARTITION BY Clause****

SELECT Name, Department, Salary,

ROW\_NUMBER() OVER

(

PARTITION BY Department

ORDER BY Name

) AS RowNumber

FROM Employees

##### ****What is a Stored Procedure in SQL Server?****

##### **A SQL Server Stored Procedure is a database object which contains pre-compiled queries (a group of T-SQL Statements). In other words, we can say that the Stored Procedures are a block of code designed to perform a task whenever we called.**

CREATE PROCEDURE spDisplayWelcome

AS

BEGIN

PRINT 'WELCOME TO PROCEDURE in SQL Server'

END

##### ****Calling a stored procedure****

EXECUTE spDisplayWelcome

Or

EXEC spDisplayWelcome

Or

spDisplayWelcome

##### ****Input Stored Procedure****

CREATE PROCEDURE spGetEmployeesByGenderAndDepartment

@Gender VARCHAR(20),

@DeptID INT

AS

BEGIN

SELECT Name, Gender, DOB, DeptID

FROM Employee

WHERE Gender = @Gender AND DeptID = @DeptID

END

-- Calling the procedure:

EXECUTE spAddTwoNumbers 10, 20

##### ****Output Stored Procedure****

CREATE PROCEDURE spGetResult

@No1 INT,

@No2 INT,

@Result INT OUTPUT

AS

BEGIN

SET @Result = @No1 + @No2

END

-- To Execute Procedure

DECLARE @Result INT

EXECUTE spGetResult 10, 20, @Result OUT

PRINT @Result

##### ****SQL Server Temporary Stored Procedure****

CREATE PROCEDURE #LocalProcedure

AS

BEGIN

PRINT 'This is Local Temporary Procedure'

END

-- Calling the Local Temporary Procedure

EXEC #LocalProcedue

##### ****Global Temporary Stored Procedures****

CREATE PROCEDURE ##LocalProcedure

AS

BEGIN

PRINT 'This is Local Temporary Procedure'

END

-- Calling the Local Temporary Procedure

EXEC ##LocalProcedue

##### ****What is a function in SQL Server?****

##### **A function in SQL Server is a subprogram that is used to perform an action such as complex calculation and returns the result of the action as a value. There are two types of functions in SQL Server, such as**

##### **System Defined Function**

##### **User-Defined Function**

##### **The functions which are already defined by the system and ready to be used by the developer are called system-defined functions whereas if the function is defined by the user or developer then such types of functions are called the user-defined function.**

##### ****Types of User-Defined Function:****

##### **In SQL Server, we can create three types of User-Defined Functions, such as**

##### **Scalar Valued Functions**

##### [**Inline Table-Valued Functions**](https://dotnettutorials.net/lesson/inline-table-valued-function-in-sql-server/)

##### [**Multi-Statement Table-Valued Functions**](https://dotnettutorials.net/lesson/multi-statement-table-valued-function-in-sql-server/)

##### ****Example1: Create a Scalar Function in SQL Server which will return the cube of a given value.****

##### **The function will an integer input parameter and then calculate the cube of that integer value and then returns the result. Here, the result produced will be in Integer. So, while creating the function, we need to specify the Returns data type as INT. The following function exactly does the same.**

CREATE FUNCTION SVF1(@X INT)

RETURNS INT

AS

BEGIN

RETURN @X \* @X \*@X

##### END

##### 

##### ****Table-Valued Function in SQL Server****

##### **In the case of a Table-Valued Function, we can return a table as an output from the function.**

-- Create Student Table

CREATE TABLE Student

(

ID INT PRIMARY KEY,

Name VARCHAR(50),

Gender VARCHAR(50),

DOB DATETIME,

Branch VARCHAR(50)

)

-- Populate the Student Table with test data

INSERT INTO Student VALUES(1, 'Pranaya', 'Male','1996-02-29 10:53:27.060', 'CSE')

INSERT INTO Student VALUES(2, 'Priyanka', 'Female','1995-05-25 10:53:27.060', 'CSE')

INSERT INTO Student VALUES(3, 'Anurag', 'Male','1995-04-19 10:53:27.060', 'ETC')

INSERT INTO Student VALUES(4, 'Preety', 'Female','1996-03-17 10:53:27.060', 'ETC')

INSERT INTO Student VALUES(5, 'Sambit', 'Male','1997-01-15 10:53:27.060', 'CSE')

--Create a function

CREATE FUNCTION FN\_GetStudentDetailsByID

(

@ID INT

)

RETURNS TABLE

AS

RETURN (SELECT \* FROM Student WHERE ID = @ID)

--Calling a function

##### SELECT \* FROM FN\_GetStudentDetailsByID(2)

##### 

##### ****Multi-Statement Table-Valued Function in SQL Server****

The Multi-Statement Table Valued Function in SQL Server is the same as the Inline Table-Valued Function means it is also going to returns a table as an output but with the following differences.

1. The Multi-Statement Table-Valued Function body can contain more than one statement. In Inline Table-Valued Function, it contains only a single Select statement prepared by the return statement.
2. In Multi-Statement Table-Valued Function, the structure of the table returned from the function is defined by us. But, in Inline Table-Valued Function, the structure of the table is defined by the Select statement that is going to return from the function body.

-- Multi-statement Table Valued function:

CREATE FUNCTION MSTVF\_GetEmployees()

RETURNS @Table Table (ID int, Name nvarchar(20), DOB Date)

AS

BEGIN

INSERT INTO @Table

SELECT ID, Name, Cast(DOB AS Date)

FROM Employee

Return

##### End

##### ****What is a Transaction in SQL Server?****

A transaction is a set of SQL statements that should be executed as one unit. That means a transaction ensures that either all of the command succeeds or none of them. If one of the commands in the transaction fails, all of the commands fail and any data that is modified in the database is rolled back.

##### ****How to implement Transaction Management in SQL Server?****

1. **Begin Transaction**: It indicates that the transaction is started.
2. **Commit Transaction**: It indicates that the transaction was completed successfully and all the data manipulation operations performed since the start of the transaction are committed to the database and frees the resources held by the transaction.
3. **Rollback Transaction**: It indicates that the transaction was Failed and will roll back the data to its previous state.
4. **Save Transaction**: This is used for dividing or breaking a transaction into multiple units so that the user has a chance of roll backing a transaction up to a point or location.

--Create Product table

CREATE TABLE Product

(

ProductID INT PRIMARY KEY,

Name VARCHAR(40),

Price INT,

Quantity INT

)

GO

-- Populate Product Table with test data

INSERT INTO Product VALUES(101, 'Product-1', 100, 10)

INSERT INTO Product VALUES(102, 'Product-2', 200, 15)

INSERT INTO Product VALUES(103, 'Product-3', 300, 20)

INSERT INTO Product VALUES(104, 'Product-4', 400, 25)

**BEGIN TRANSACTION**

INSERT INTO Product VALUES(105,'Product-5',500, 30)

UPDATE Product SET Price =350 WHERE ProductID = 103

DELETE FROM Product WHERE ProductID = 103

##### COMMIT TRANSACTION

##### 

##### ****ROLLBACK command in SQL Server?****

##### The Rollback command in SQL Server is used to undo the transactions that have not already been saved to the database and get back to the initial state from where the transaction was started. So, if you want to restore the data into its previous state then you need to use the ROLLBACK command at any time after the DML queries have been written but remember once the COMMIT has been written then we cannot ROLLBACK the data.

BEGIN TRANSACTION

INSERT INTO Product VALUES(110,'Product-10',600, 30)

INSERT INTO Product VALUES(110,'Product-10',600, 30)

IF(@@ERROR > 0)

BEGIN

Rollback Transaction

END

ELSE

BEGIN

Commit Transaction

END

##### What are ACID Properties of a Transaction?

##### In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction, such as

##### Atomicity

##### Consistency

##### Isolation

##### Durability.

##### The **Atomicity Property of a Transaction in SQL Server** ensures that either all the DML Statements (i.e. insert, update, delete) inside a transaction are completed successfully or all of them are rolled back

##### The **Consistency Property of a Transaction in SQL Server** ensures that the database data is in a consistent state before the transaction started and also left the data in a consistent state after the transaction is completed. If the transaction violates the rules then it should be rolled back.

##### The **Isolation Property of a Transaction in SQL Server** ensures that the intermediate state of a transaction is invisible to other transactions. The Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. Most databases use locking to maintain transaction isolation.

##### The **Durability Property of a Transaction in SQL Server** ensures that once the transaction is successfully completed, then the changes it made to the database will be permanent. Even if there is a system failure or power failure or any abnormal changes, it should safeguard the committed data.

##### ****Exception Handling in SQL Server?****

##### **In SQL Server, whenever an exception occurred, it displays the exception message and then continues the program execution. But in programming languages such as C#, Java, C++, etc. whenever an exception occurred then the program execution terminates abnormally on the line where the exception occurred.**

##### **We handle errors both in programming languages as well as in databases. Handling errors in a programming language means stopping the abnormal termination of the program and allowing the statements which are not related to the error to execute but handling an error in SQL Server means stopping the execution of the statements which are related to the error.**

Create PROCEDURE spDivideTwoNumber

@Number1 INT,

@Number2 INT

AS

BEGIN

DECLARE @Result INT

SET @Result = 0

IF(@Number2 = 0)

BEGIN

RAISERROR('Second Number Cannot be zero',16,1)

END

ELSE

BEGIN

SET @Result = @Number1 / @Number2

END

IF(@@ERROR <> 0)

BEGIN

PRINT 'Error Occurred'

END

ELSE

BEGIN

PRINT 'RESULT IS :'+CAST(@Result AS VARCHAR)

END

END

##### Let’s execute the procedure by giving the second value as zero as shown below.

##### EXEC spDivideTwoNumber 100, 0

##### When we execute the above procedure with 100 and 0, it gives us the below error message.

##### 

##### The RaiseError System defined Function in SQL Server takes 3 parameters as shown below.

##### RAISERROR(‘Error Message’, ErrorSeverity, ErrorState)

##### ****Try Catch in SQL Server to Handle Error?****

**From SQL Server 2005 we are provided with a structure error handling mechanism with the help of TRY and CATCH blocks. The syntax of using TRY- CATCH is shown in the below image.**

ALTER PROCEDURE spDivideTwoNumber

@Number1 INT,

@Number2 INT

AS

BEGIN TRY

-- We need to start the transaction

BEGIN TRANSACTION

-- First we need to reduce the quantity available

DECLARE @Result INT

SET @Result = 0

IF(@Number2 = 0)

BEGIN

RAISERROR('Second Number Cannot be zero',16,1)

END

ELSE

BEGIN

SET @Result = @Number1 / @Number2

--Insert/Update/Delete command here

END

COMMIT TRANSACTION

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

SELECT ERROR\_NUMBER() as ErrorNumber,

ERROR\_MESSAGE() as ErrorMessage,

ERROR\_PROCEDURE() as ErrorProcedure,

ERROR\_STATE() as ErrorState,

ERROR\_SEVERITY() as ErrorSeverity,

ERROR\_LINE() as ErrorLine

##### END CATCH

## **Triggers in SQL Server with Examples**

##### Triggers are nothing but they are logic’s like stored procedures that can be executed automatically before the Insert, Update or Delete happens in a table or after the Insert, Update, or Delete happens in a table. In simple words, we can say that, if you want to execute some pre-processing or post-processing logic before or after the Insert, Update, or Delete in a table then you need to use Triggers in SQL Server.

There are two types of triggers. They are as follows:

* **Instead of Triggers:** The Instead Of triggers are going to be executed instead of the corresponding DML operations. That means instead of the DML operations such as Insert, Update, and Delete, the Instead Of triggers are going to be executed.
* **After Triggers:** The After Triggers fires in SQL Server execute after the triggering action. That means once the DML statement (such as Insert, Update, and Delete) completes its execution, this trigger is going to be fired.

-- Create Employee table

CREATE TABLE Employee

(

Id int Primary Key,

Name nvarchar(30),

Salary int,

Gender nvarchar(10),

DepartmentId int

)

GO

-- Insert data into Employee table

INSERT INTO Employee VALUES (1,'Pranaya', 5000, 'Male', 3)

INSERT INTO Employee VALUES (2,'Priyanka', 5400, 'Female', 2)

INSERT INTO Employee VALUES (3,'Anurag', 6500, 'male', 1)

INSERT INTO Employee VALUES (4,'sambit', 4700, 'Male', 2)

INSERT INTO Employee VALUES (5,'Hina', 6600, 'Female', 3)

**For/After Insert DML Trigger in SQL Server**

CREATE TRIGGER trInsertEmployee

ON Employee

FOR INSERT --UPDATE / DELETE

AS

BEGIN

PRINT 'YOU CANNOT PERFORM INSERT OPERATION'

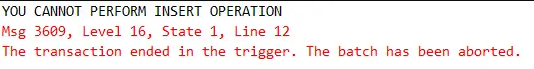
ROLLBACK TRANSACTION

END

Let’s try to insert the following record into the employee table.

**INSERT INTO Employee VALUES (6, ‘Saroj’, 7600, ‘Male’, 1)**

When you try to execute the above Insert statement it gives you the below error. First, the INSERT statement is executed, and then immediately this trigger fired and roll back the INSERT operation as well as print the message.



**Instead Of Insert DML Trigger in SQL Server**

CREATE TRIGGER tr\_vwEmployeeDetails\_InsteadOfUpdate

ON Employee

INSTEAD OF INSERT --UPDATE / DELETE

AS

BEGIN

INSERT INTO Employee VALUES (6, 'Saroj', 7600, 'Male', 1)

##### END

## **Joins in SQL Server**

##### The SQL Server Joins are used to retrieve the data from two or more related tables. In general, tables are related to each other using the primary key and foreign key relationship but it is not mandatory. The tables involved in the joins must have a common field. And based on that common field the SQL Server JOINS retrieves the records.

##### The SQL Server Joins are classified into two types such as

##### ANSI format JOINS

##### NON-ANSI format JOINS

##### Again the ANSI format joins classified into three types such as

##### Inner join

##### Outer join

##### Cross join

##### Further, the outer join is divided into three types are as follows

##### Left outer join

##### Right outer join

##### Full outer join

##### NON-ANSI join in SQL Server are classified into four types such as

##### EQUI join

##### NON-EQUI join

##### SELF-join

##### Natural Join

-- Create Company Database

CREATE DATABASE Company;

-- Create Employee Table

CREATE TABLE Employee (

Id INT PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Department VARCHAR(100) NOT NULL,

Salary FLOAT NOT NULL,

Gender VARCHAR(45) NOT NULL,

Age INT NOT NULL,

City VARCHAR(45) NOT NULL

);

-- Populate Employee Table

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1001, 'John Doe', 'IT', 35000, 'Male', 25, 'London');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1002, 'Mary Smith', 'HR', 45000, 'Female', 27, 'London');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1003, 'James Brown', 'Finance', 50000, 'Male', 28, 'London');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1004, 'Mike Walker', 'Finance', 50000, 'Male', 28, 'London');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1005, 'Linda Jones', 'HR', 75000, 'Female', 26, 'London');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1006, 'Anurag Mohanty', 'IT', 35000, 'Male', 25, 'Mumbai');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1007, 'Priyanla Dewangan', 'HR', 45000, 'Female', 27, 'Mumbai');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1008, 'Sambit Mohanty', 'IT', 50000, 'Male', 28, 'Mumbai');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1009, 'Pranaya Kumar', 'IT', 50000, 'Male', 28, 'Mumbai');

INSERT INTO Employee (Id, Name, Department, Salary, Gender, Age, City) VALUES (1010, 'Hina Sharma', 'HR', 75000, 'Female', 26, 'Mumbai');

-- Create Projects Table

CREATE TABLE Projects (

ProjectId INT PRIMARY KEY IDENTITY(1, 1),

Title VARCHAR(200) NOT NULL,

ClientId INT,

EmployeeId INT,

StartDate DATETIME,

EndDate DATETIME,

FOREIGN KEY (EmployeeId) REFERENCES Employee(Id)

);

-- Populate Projects Table

INSERT INTO Projects (Title, ClientId, EmployeeId, StartDate, EndDate) VALUES

('Develop ecommerse website from scratch', 1, 1003, GETDATE(), (Getdate() + 35)),

('WordPress website for our company', 1, 1002, GETDATE(), (Getdate() + 45)),

('Manage our company servers', 2, 1007, GETDATE(), (Getdate() + 55)),

('Hosting account is not working', 3, 1009, GETDATE(), (Getdate() + 65)),

('MySQL database from my desktop application', 4, 1010, GETDATE(), (Getdate() + 75)),

('Develop new WordPress plugin for my business website', 2, NULL, GETDATE(), (Getdate() + 85)),

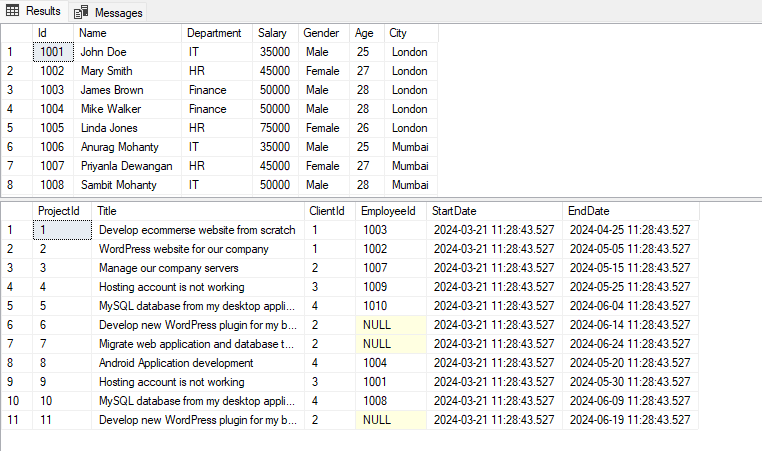
('Migrate web application and database to new server', 2, NULL, GETDATE(), (Getdate() + 95)),

('Android Application development', 4, 1004, GETDATE(), (Getdate() + 60)),

('Hosting account is not working', 3, 1001, GETDATE(), (Getdate() + 70)),

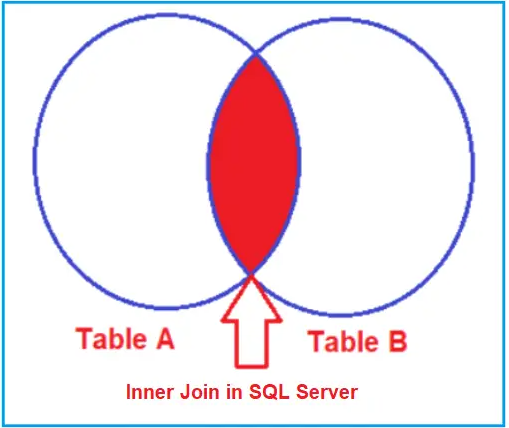
('MySQL database from my desktop application', 4, 1008, GETDATE(), (Getdate() + 80)),

('Develop new WordPress plugin for my business website', 2, NULL, GETDATE(), (Getdate() + 90));

****

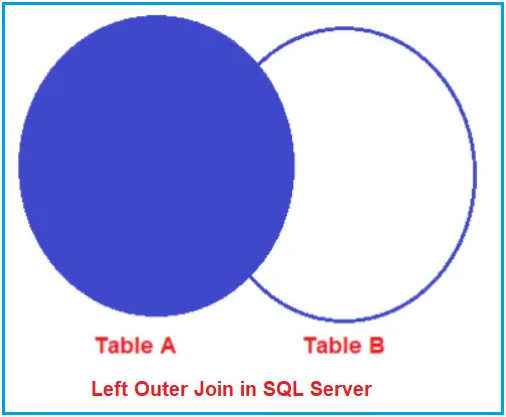
##### ****Inner Join in SQL Server****

The Inner Join in SQL Server is used to return only the matching rows from both the tables involved in the join by removing the non-matching records. The following diagram shows the pictorial representation of SQL Server Inner Join.

****

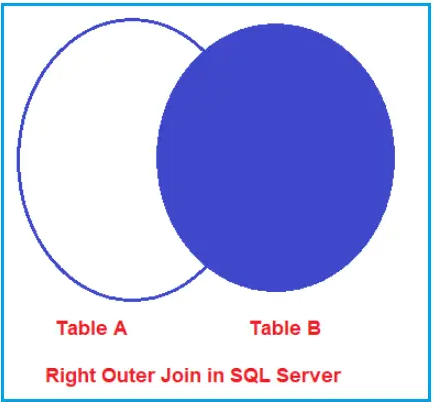
##### ****Left Outer Join in SQL Server****

The LEFT OUTER JOIN in SQL Server is used to retrieve all the matching rows from both the tables involved in the join as well as non-matching rows from the left side table. In this case, the un-matching data will take a null value.

****

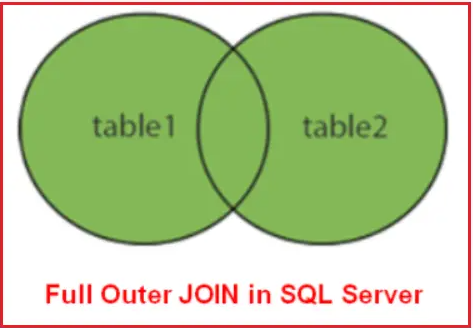
##### ****Right Outer Join in SQL Server****

The RIGHT OUTER JOIN in SQL Server is used to retrieve all the matching rows from both the tables involved in the join as well as non-matching rows from the right-side table. In this case, the un-matching data will take NULL values.

****

##### ****Full Outer Join in SQL Server****

The Full Outer Join in SQL Server is used to retrieve all the matching records as well as all the non-matching records from both the tables involved in the JOIN. The Un-matching data in such cases will take the NULL values. The following diagram shows the pictorial representation of Full Outer Join in SQL Server.

****

##### ****Cross Join in SQL Server****

The CROSS JOIN is created by using the CROSS JOIN keyword. The CROSS JOIN does not contain an ON clause. In Cross Join, each record of a table is joined with each record of the other table. In SQL Server, the Cross Join should not have either an ON or where clause.

SELECT Employee.Id as EmployeeId, Name, Department, City, Title as Project

FROM Employee

CROSS JOIN Projects;

**What is Self-Join in SQL Server?**

Joining a table by itself is called self-join in SQL Server. When we have some relation between the columns within the same table then we need to use the self-join mechanism.

SELECT E.Name as Employee, M.Name as Manager

FROM Employee E

INNER JOIN Employee M

ON E.ManagerId = M.EmployeeId

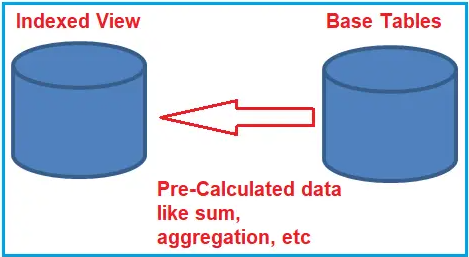
**We Need Views in SQL Server**

1. To protect the data. If we have a table containing sensitive data in certain columns, we might wish to hide those columns from certain groups of users. For instance, customer names, addresses, and social security numbers might all be stored in the same table; however, for lower-level employees like shipping clerks, you can create a view that only displays customer names and addresses. You can grant permissions to a view without allowing users to query the original tables.
2. A view is a logical table but what it stores internally is a select statement that is used for creating the view. So that whenever a user performs an operation on the view like select, insert, update or delete internally the view performs those operations on a table.
3. Simply we can say that view will act as an interface between the data provider (Table) and the User.
4. The view is created based on a table; any changes that are performed on the table reflect into the view any changes performed on the view reflect on the table also.

**What is an Indexed View in SQL Server?**

Indexed View is a virtual table whose result set (output of a select statement) is persistent on the hard disk. In general, when we create a view, the view does not store any data by default. So, when we query a view, it actually queries the underlying base table and gets the data. But we can change this default behaviour in SQL Server i.e. the SQL Server Views can store the data physically. In order to do this, first, you need to create an index on the view.

When we create an index on a view, then the view is called Index View. In the case of Indexed views, the result set (output of select statement which includes pre-calculated data like aggregation, summation, count, etc.) is persisted on the hard disk. As the data is persisted on the hard disk, now the SQL Server engine does not have to go again and again and get the data from the underlying base table. So, what it does is it gets the data that is already persisted in the index view virtual table. And because of this, it saves lots of overheads.



**Note:** Indexed View is not always suggested for all situations. For example, if your table is highly transactional (i.e. the base table having lots of Insert, Update and Delete operations), then the indexed view is not suggested. This is because the synchronization between the base table and the indexed view is a time-consuming task.

##### ****Creating an Index view in SQL Server****

-- Create table Product

CREATE TABLE Product

(

ProductId INT PRIMARY KEY,

Name VARCHAR(20),

UnitPrice INT

)

GO

-- Populate Product table with some test data

INSERT INTO Product VALUES(1, 'Books', 40)

INSERT INTO Product VALUES(2, 'Pens', 30)

INSERT INTO Product VALUES(3, 'Pencils', 10)

GO

-- Create table ProductSales

CREATE TABLE ProductSales

(

ProductId INT,

QuantitySold INT

)

GO

-- Populate ProductSales table with some test data

INSERT INTO ProductSales VALUES(1, 10)

INSERT INTO ProductSales VALUES(3, 23)

INSERT INTO ProductSales VALUES(3, 21)

INSERT INTO ProductSales VALUES(2, 12)

INSERT INTO ProductSales VALUES(1, 13)

INSERT INTO ProductSales VALUES(3, 12)

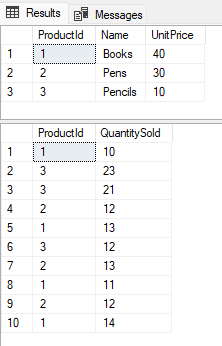
INSERT INTO ProductSales VALUES(2, 13)

INSERT INTO ProductSales VALUES(1, 11)

INSERT INTO ProductSales VALUES(2, 12)

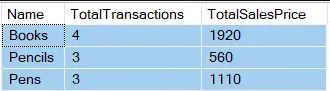
INSERT INTO ProductSales VALUES(1, 14)

GO



###### **Example:**

Create a view that will return the Total Sales and Total Transactions by Product. The output should be as shown below.



CREATE VIEW vwTotalSalesPriceByProduct

WITH SCHEMABINDING

AS

SELECT Name,

COUNT\_BIG(\*) AS TotalTransactions,

SUM(ISNULL((QuantitySold \* UnitPrice), 0)) AS TotalSalesPrice

FROM dbo.ProductSales prdSales

INNER JOIN dbo.Product prd

ON prd.ProductId = prdSales.ProductId

GROUP BY Name

**Rules for creating an Index view in SQL Server:**

1. The view should be created with the **SCHEMABINDING** option
2. If an Aggregate function is used in the **SELECT LIST** which references an expression, and if there is a possibility for that expression to become **NULL**, then, a replacement value should be specified. In this example, we are using **ISNULL()** function, to replace **NULL** values with **ZERO** with the expression **SUM(ISNULL((QuantitySold \* UnitPrice), 0)) AS TotalSalesPrice**
3. If the **GROUP BY** clause is specified, then the view select list must contain a **COUNT\_BIG(\*)** expression
4. The base tables in the view should be referenced with 2 part name. In this example, Product and ProductSales are referenced using **dbo.Product** and **dbo.ProductSales** respectively.

**Creating an index on a view**

##### 

The first index that we create on a view must be a unique clustered index. After the unique clustered index has been created, then we can create additional non-clustered indexes.

CREATE UNIQUE CLUSTERED

INDEX UIX\_vwTotalSalesPriceByProduct\_Name

ON vwTotalSalesPriceByProduct(Name)

Now we have an index on the vwTotalSalesPriceByProduct view, so the view gets materialized. The data is stored in the view. So when we execute

Select \* from vwTotalSalesPriceByProduct

The data is returned from the view itself, rather than retrieving data from the underlying base tables. The Indexed view in SQL Server can significantly improve the performance of the queries that involve Joins and Aggregations. The cost of maintaining an indexed view is much higher than the cost of maintaining a table index.

Once the Indexed view is created, its data will be stored in your database the same as any other clustered index, so the storage space for the view’s clustered index should be taken into consideration. Having the indexed view’s clustered index stored in the database, with its own statistics created to optimize the cardinality estimation, different from the underlying tables’ statistics, the SQL engine will not waste the time substituting the source tables’ definition in the main query, and it will read directly from the view’s clustered index.

##### ****When to use Indexed View in SQL Server?****

Indexed views are ideal for scenarios, where the underlying data is not frequently changed. Indexed views are more often used in OLAP systems because the data is mainly used for reporting and analysis purposes. The Indexed views may not be suitable for OLTP systems, as the data is frequently added and changed.

##### ****REVERSE String Function in SQL Server****

SELECT REVERSE (Column\_Name) AS Alias\_Name FROM Table\_Name;

##### ****Count characters in string****

Declare @myvar varchar(20),@substr Varchar(1),@strcount int

Set @myvar = 'POOJA'

Declare @i int=0

While(@i<LEN(@myvar))

Begin

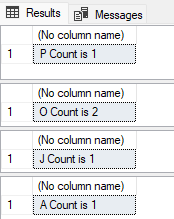
Select @substr=SUBSTRING (@myvar, 1, 1)

Select @strcount=LEN(@myvar) - LEN(REPLACE(@myvar,@substr,''))

Select @substr+ ' Count is '+Cast(@strcount As varchar)

Select @myvar=REPLACE(@myvar,@substr,'')

##### End



**Common Table Expression**

A Common Table Expression, also called as CTE in short form, is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. The CTE can also be used in a View.

create table Employees

(

ID int,

Name nvarchar(50),

Gender char(10),

Salary int

)

insert into Employees values(1,'Farhan Ahmed','Male',60000)

insert into Employees values(5,'Monika','Female',25000)

insert into Employees values(2,'Abdul Raheem','Male',30000)

insert into Employees values(4,'Rahul Sharma','Male',60000)

insert into Employees values(1,'Farhan Ahmed','Male',60000)

insert into Employees values(2,'Abdul Raheem','Male',30000)

insert into Employees values(5,'Monika','Female',25000)

insert into Employees values(4,'Rahul Sharma','Male',60000)

insert into Employees values(1,'Farhan Ahmed','Male',60000)

insert into Employees values(3,'Priya','Female',20000)

insert into Employees values(5,'Monika','Female',25000)

insert into Employees values(4,'Rahul Sharma','Male',60000)

insert into Employees values(5,'Monika','Female',25000)

insert into Employees values(2,'Abdul Raheem','Male',30000)

insert into Employees values(1,'Farhan Ahmed','Male',60000)

insert into Employees values(4,'Rahul Sharma','Male',60000)

WITH EmployeesCTE as

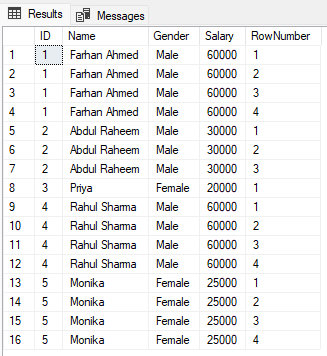
(

SELECT\*, ROW\_NUMBER() over (PARTITION BY ID ORDER BY ID) as RowNumber

FROM Employees

)

SELECT \* FROM EmployeesCTE



WITH EmployeesCTE as

(

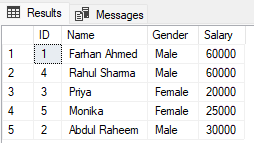
SELECT\*, ROW\_NUMBER() over (PARTITION BY ID ORDER BY ID) as RowNumber

FROM Employees

)

DELETE FROM EmployeesCTE WHERE RowNumber>1

SELECT \* FROM Employees



### **Differences between Temporary Table and Table variable in SQL Server**

1. The table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if the size increases it will be moved to the tempdb.
2. Table variables cannot be involved in transactions, logging, or locking. This makes @table faster than #temp. So, table variable is faster than the temporary table.
3. Temporary tables are allowed CREATE INDEXes whereas, Table variables aren’t allowed CREATE INDEX instead they can have an index by using Primary Key or Unique Constraint.
4. A table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables.
5. A temporary table allows Schema modifications, unlike Table variables.

# **User Defined Table Types And Table Valued Parameters**

## Create User-Defined Table Type

CREATE TYPE [dbo].[CustomerType] AS TABLE(

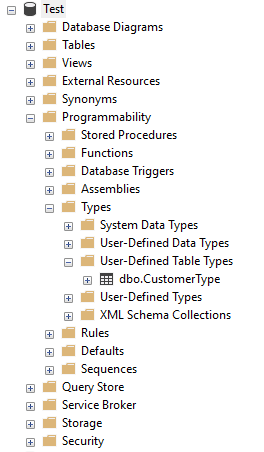
[Id] [int] NULL,

[Name] [varchar](100) NULL,

[Country] [varchar](50) NULL

)

GO



CREATE PROCEDURE [dbo].[Insert\_Customers]

@tblCustomers CustomerType

AS

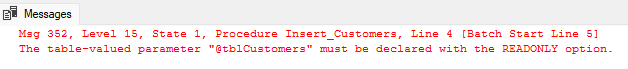
BEGIN

SET NOCOUNT ON;

INSERT INTO Customers(CustomerId, Name, Country)

SELECT Id, Name, Country FROM @tblCustomers

END



When we try to execute the above query, we get the error of Read Only option, which means Table-valued parameters must be passed as READONLY parameters, and we can’t also perform DML operations like UPDATE, DELETE, or INSERT on a table-valued parameter in the body of a routine.

CREATE PROCEDURE [dbo].[Insert\_Customers]

@tblCustomers CustomerType READONLY

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Customers(CustomerId, Name, Country)

SELECT Id, Name, Country FROM @tblCustomers

END

public void Insert(DataTable dt)

{

if (dt.Rows.Count > 0)

{

string constr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;

using (SqlConnection conn = new SqlConnection(constr))

{

string sql = "Insert\_Customers";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.CommandType = CommandType.StoredProcedure;

                    cmd.Parameters.AddWithValue("@tblCustomers", dt);

                    conn.Open();

                    cmd.ExecuteNonQuery();

                    conn.Close();

}

}

}

}