Indexes in SQL Server

Back to: SQL Server Tutorial For Beginners and Professionals

Indexes in SQL Server with Examples

In this article, I am going to discuss **Indexes in SQL Server with Examples** and we will also discuss how indexes make your search operations faster. Please read our previous article, where we discussed **Joins in SQL Server** with Examples. As part of this article, we are going to discuss the following pointers in detail.

1. **How does data search when there is no index?**
2. **Understanding the Balanced Tree (B-Tree) in SQL Server.**
3. **How will the database engine retrieve the data from a table?**
4. **Multiple Examples to understand Indexes in SQL Server.**
5. **What is an index?**
6. **When SQL Server uses Indexes?**
7. **Types of indexes in SQL Server.**

The goal of the index is to make the search operation faster. Then the next question that should come to your mind is how does index make your search operation faster? Indexes make the search operation faster by creating something called a **B-Tree (Balanced Tree)** structure internally. So, in this article, first, we will understand the theory of **Balanced Tree (B-Tree) Structure,** and then we will see the practical implementation of how indexes make the search operator faster. And finally, we will discuss the different types of indexes available in the SQL Server database.

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

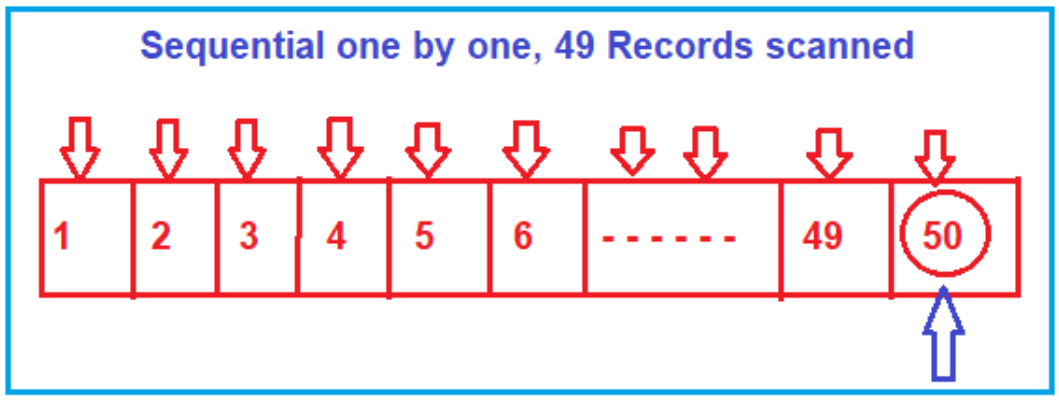
Whenever the database engine wants to retrieve the data from a database table it will adopt two different mechanisms for searching the data

1. **Table scan**
2. **Index Scan/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. Please have a look at the following image for a better understanding.



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.

Understanding the Balanced Tree (B-Tree) in SQL Server:

Whenever you create an index (or indexes) on some column(s) of a table in SQL Server then what happens internally is, it creates a B-Tree structure. In the B-Tree structure, the data is divided into three sections i.e. Root Node, Non-Leaf Nodes, and Leaf Nodes. In order to understand this better please have a look at the following image which shows how the data is divided and stored. As you can see, in the Root Node it has 30 and 50. In the Non-Leaf node, it has 10, 30, 40, and 50. And in the leaf node, we have the actual data. So, the leaf node is actually pointing to data.

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.

How Index improves search performance in SQL Server?

Let us understand how SQL Server Index improves search performance with an example. We are going to use the following Employee table to understand how Indexes improve search performance.

Id	Name	Salary	Gender	City	Dept
1	Anurag	2500	Male	London	IT
2	Hina	500	Female	Sydney	HR
3	Pranaya	4500	Male	New York	IT
4	Priyanka	5500	Female	Tokiyo	HR
5	Sambit	3000	Male	Toronto	IT
6	Tarun	4000	Male	Delhi	IT
7	Preety	6500	Female	Mumbai	HR
8	John	6500	Male	Mumbai	HR
9	Sara	500	Female	London	IT
10	Pam	4000	Female	Delhi	IT

Please use the following SQL Script to create and populate the Employee table with the required sample data.

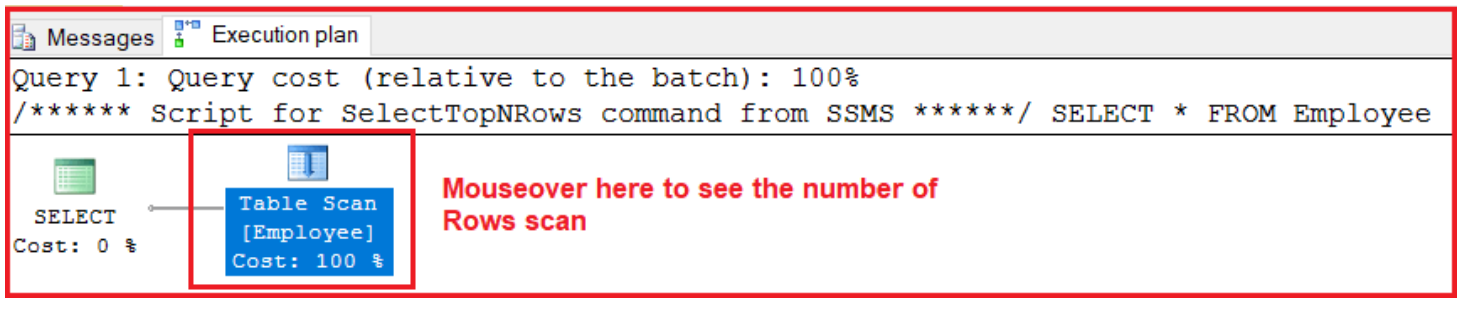
```
CREATE TABLE Employee
(
 Id INT,
 Name VARCHAR(50),
 Salary INT,
 Gender VARCHAR(10),
 City VARCHAR(50),
 Dept VARCHAR(50)
)
GO

INSERT INTO Employee VALUES (3,'Pranaya', 4500, 'Male', 'New York',
'IT')
INSERT INTO Employee VALUES (1,'Anurag', 2500, 'Male', 'London',
'IT')
INSERT INTO Employee VALUES (4,'Priyanka', 5500, 'Female', 'Tokiyo',
'HR')
INSERT INTO Employee VALUES (5,'Sambit', 3000, 'Male', 'Toronto',
'IT')
INSERT INTO Employee VALUES (7,'Preety', 6500, 'Female', 'Mumbai',
'HR')
INSERT INTO Employee VALUES (6,'Tarun', 4000, 'Male', 'Delhi', 'IT')
INSERT INTO Employee VALUES (2,'Hina', 500, 'Female', 'Sydney', 'HR')
INSERT INTO Employee VALUES (8,'John', 6500, 'Male', 'Mumbai', 'HR')
INSERT INTO Employee VALUES (10,'Pam', 4000, 'Female', 'Delhi', 'IT')
INSERT INTO Employee VALUES (9,'Sara', 500, 'Female', 'London', 'IT')
```

Currently, we don't have an index in any of the columns of the Employee table. Let us write a query to get the employee info whose id is 8.

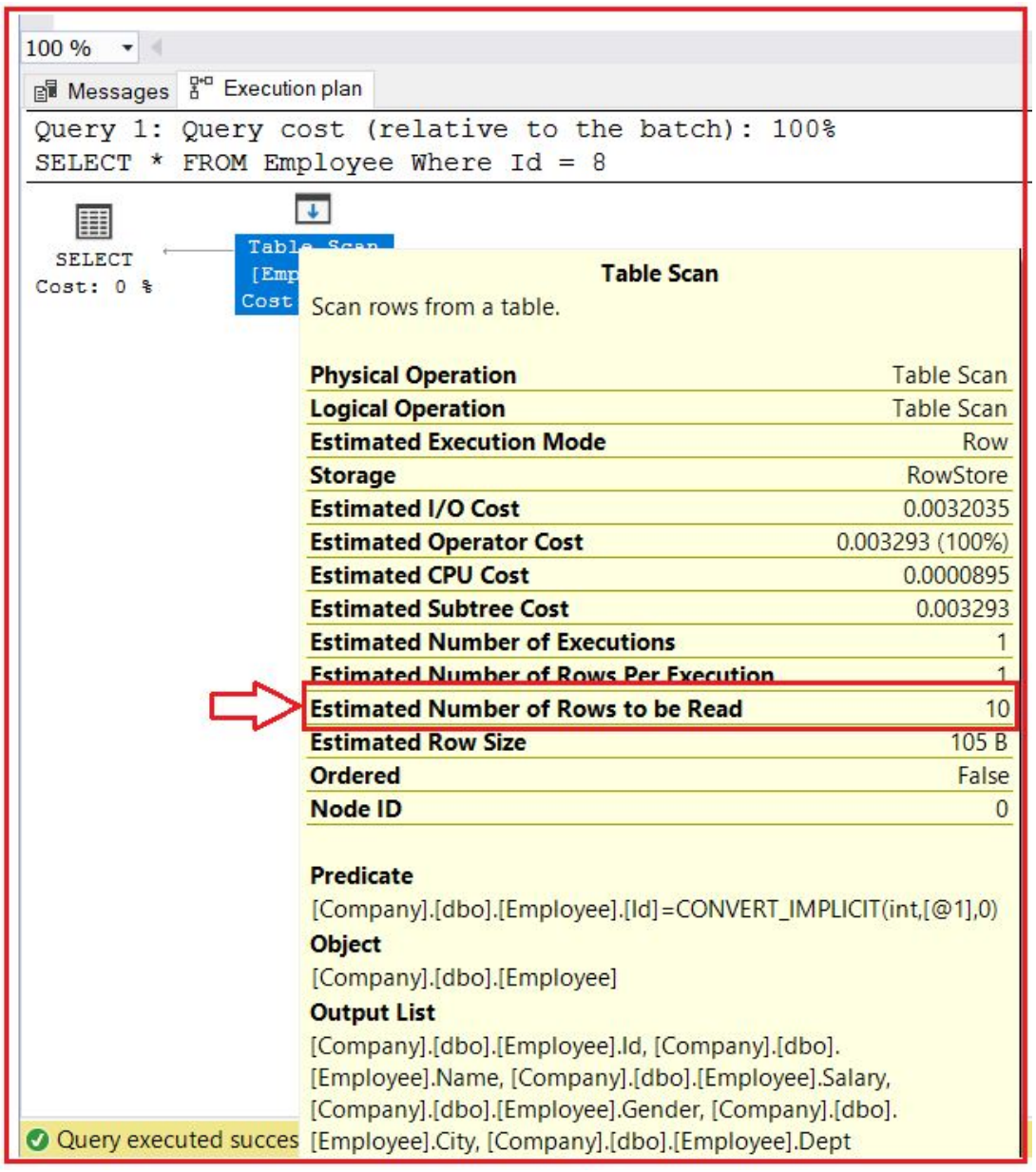
```
SELECT * FROM Employee Where Id = 8;
```

When you execute the above query it will use a table scan to get the data. In order to make sure it uses a table scan, please click on the **Display Estimated Execution Plan** button as shown in the below image.

Once you click on the above **Display Estimated Execution Plan** icon it will show the following image which clearly says that it performs a Table scan.



Further, if you mouse over of Table Scan option, then you will see that the value of the **Estimated Number of Rows** is 10 as our table currently holds 10 records as shown in the below image.



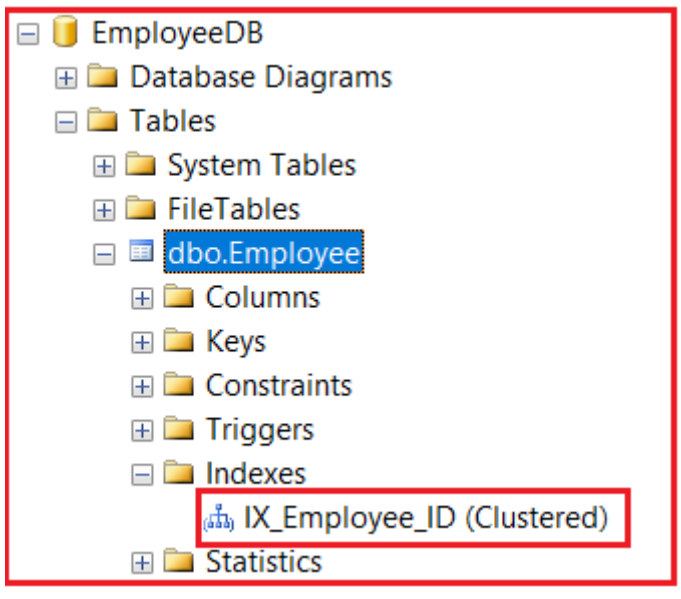
You will not find any performance issues currently as the number of records is less. But if your table contains a huge amount of data let's say 1000000 records, then it will definitely take much more time to get the data.

Creating Index on Id Column:

Let us create an index on the **Id** column of the Employee table by executing the following query. Later we will discuss the syntax and the different types of indexes and their need and use. But for now, we are just focusing on the need for Indexes and how we can improve the search operation performance using indexes in SQL Server.

```
CREATE CLUSTERED INDEX IX_Employee_ID ON Employee(Id ASC);
```

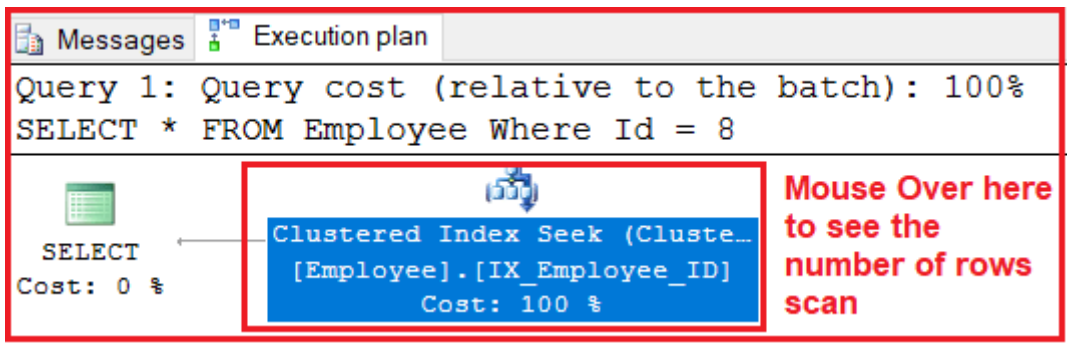
Once you execute the above statement, the index gets created and you can see the index in the indexes folder which is present inside the Employee table as shown in the below image.



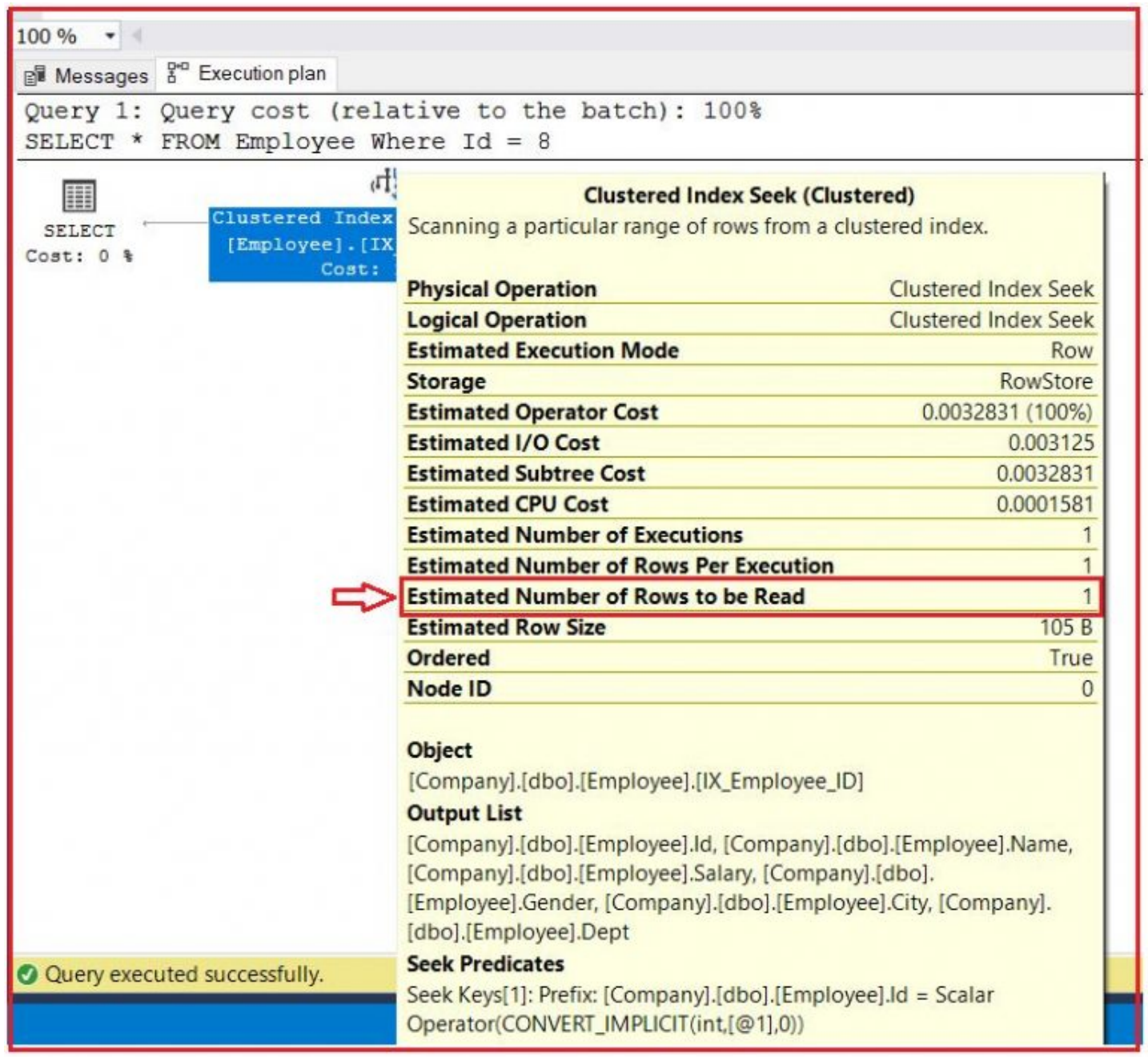
The above SQL Server Index stores the Id of each employee in ascending order. Once you create the index, now let us execute the same SQL query to get the employee info whose id is 8 as shown below.

```
SELECT * FROM Employee Where Id = 8;
```

Once you execute the above query, now again click on the **Display Estimated Execution Plan** option which will show the following image which clearly shows that it performs an Index Scan or Index Seek.



Further, if you mouse over of Index Seek option, then you will see that the value of the **Estimated Number of Rows** is 1 as shown in the below image which means it only scan 1 row which improves the search operation.



Now, I hope you understood the basic need for Indexes and how indexes improve the performance of the Search Operation in the SQL Server. With this keep in mind let us proceed and first discuss some of the theoretical concepts which are good if you are preparing for an interview and then we will discuss the different types of Indexes in SQL Server and their needs with Examples.

What is an Index in SQL Server?

1. It is a database object in SQL Server which is used to improve the performance of search operations.
2. When we create an index on any column of a table, then SQL Server internally maintains a separate table called the index table. And when we are trying to retrieve the data from the existing table, depending on the index table, SQL Server directly goes to the table and retrieves the data very quickly.
3. In a table, we can use a maximum of 1000 indexes (1 Clustered Index plus 999 Non-Clustered Index).

When SQL Server uses Indexes?

The SQL Server uses indexes of a table provided that the select or update or delete statement contained the “**WHERE**” clause and moreover the where condition column must be an indexed column. If the select statement contains an “**ORDER BY**” clause then also the indexes can be used.

Note: When SQL Server is searching for information under the database, first it will verify the best execution plan for retrieving the data and uses that plan which can be either a full-page scan or an index scan.

The syntax for creating an Index in SQL Server:

CREATE [UNIQUE] [CLUSTERED/ NON-CLUSTERED] INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN LIST>)
To see the index: sp_helpindex Employee
To drop an index: Drop index Employee.IX_Employee_Id

Types of indexes in SQL Server

SQL Server Indexes are divided into two types. They are as follows:

1. **Clustered index**
2. **Non-Clustered index**

What is SQL Server Clustered index?

The **Clustered Index in SQL Server** 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.

When a table has a clustered index then that table is called a clustered table. If a table has no clustered index its data rows are stored in an unordered structure.

What is SQL Server Non-Clustered Index?

In **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 the next article, I am going to discuss the **Clustered Index in SQL Server with B-Tree Structure**. Here, in this article, I try to explain the need for Indexes in SQL Server with Examples. I hope you enjoy this Indexes in SQL Server with Examples article and understand how indexes improve search operations.

---

6 thoughts on “Indexes in SQL Server”

TRAINEE
OCTOBER 16, 2019 AT 2:09 AM

great job man !

Reply

ARIF
FEBRUARY 25, 2020 AT 2:33 PM

great tutorial

Reply

LUCKY
JUNE 8, 2021 AT 10:11 AM

This tutorial explains every concept in depth and best tutorial for interviews .

Reply

MANOJ KUMAR
FEBRUARY 21, 2022 AT 2:20 PM

very nice explanation and in a very simple manner . Thank you team dotnettutorials

Reply

GOVIND GHODKE
APRIL 12, 2022 AT 6:43 PM

great explanation to point of interview purpose. Thanks....

Reply

AJIT PANDEY
NOVEMBER 23, 2022 AT 11:22 AM

Great man, for the interview.
Thank You So much

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name*

Email*

Website

Post Comment