**Indexes in SQL server**

It is a database object in SQL Server which is used to improve the performance of search operations.

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.

In a table, we can use a maximum of 1000 indexes (1 Clustered Index plus 999 Non-Clustered Index).

The goal of the index is to make the search operation faster.

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, 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.

When we create a primary key it will create a clustered index by default.

**How the database engine retrieves the data from the table in SQL server?**

Whenever the database engine wants to retrieve data from the database, it will adopt two mechanisms they are

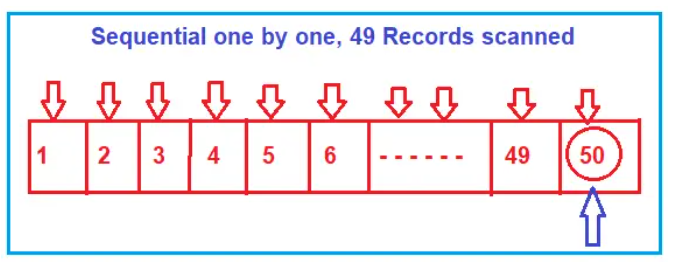
**Table scan**

**Index scan/seek.**

**What is table scan?**

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.



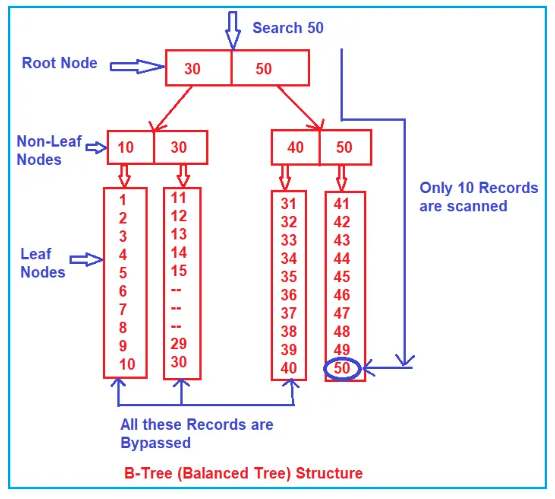
Suppose you want to search the value 50, then the database will scan the record sequentially one by one from the beginning i.e. from 1, and until it reaches the value 50. If you want to increase the search performance, then somehow you must minimize the number of scans. That is what exactly the B-Tree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

**What is index scan/seek?**

In SQL server the index scan uses the B-Tree structure to improve the query performance by reducing the no of scans

Now let us understand how B-Tree works and how it reduces the no of scans.

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. 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 pointing to data.



When you want to retrieve the value 50 from the above.

Then internally 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.

Now we will see the how the index will work by a practival approach

Create a table and inset some data into that table

CREATE TABLE Employee

(

Id INT,

Name VARCHAR(50),

Salary INT,

Gender VARCHAR(10),

City VARCHAR(50),

Dept VARCHAR(50)

)

INSERT INTO Employee VALUES

(3,'Pranaya', 4500, 'Male', 'New York', 'IT')

, (1,'Anurag', 2500, 'Male', 'London', 'IT')

, (4,'Priyanka', 5500, 'Female', 'Tokiyo', 'HR')

, (5,'Sambit', 3000, 'Male', 'Toronto', 'IT')

, (7,'Preety', 6500, 'Female', 'Mumbai', 'HR')

, (6,'Tarun', 4000, 'Male', 'Delhi', 'IT')

, (2,'Hina', 500, 'Female', 'Sydney', 'HR')

, (8,'John', 6500, 'Male', 'Mumbai', 'HR')

, (10,'Pam', 4000, 'Female', 'Delhi', 'IT')

, (9,'Sara', 500, 'Female', 'London', 'IT')

, (20,'Pranaya', 4500, 'Male', 'New York', 'IT')

, (11,'Anurag', 2500, 'Male', 'London', 'IT')

, (14,'Priyanka', 5500, 'Female', 'Tokiyo', 'HR')

, (15,'Sambit', 3000, 'Male', 'Toronto', 'IT')

, (17,'Preety', 6500, 'Female', 'Mumbai', 'HR')

, (16,'Tarun', 4000, 'Male', 'Delhi', 'IT')

, (12,'Hina', 500, 'Female', 'Sydney', 'HR')

, (18,'John', 6500, 'Male', 'Mumbai', 'HR')

, (10,'Pam', 4000, 'Female', 'Delhi', 'IT')

, (22,'Sara', 500, 'Female', 'London', 'IT')

A screenshot of a computer

Description automatically generated

Click the above button or enter (**ctrl+L**) before executing the below query which is used to show the estimated execution plan

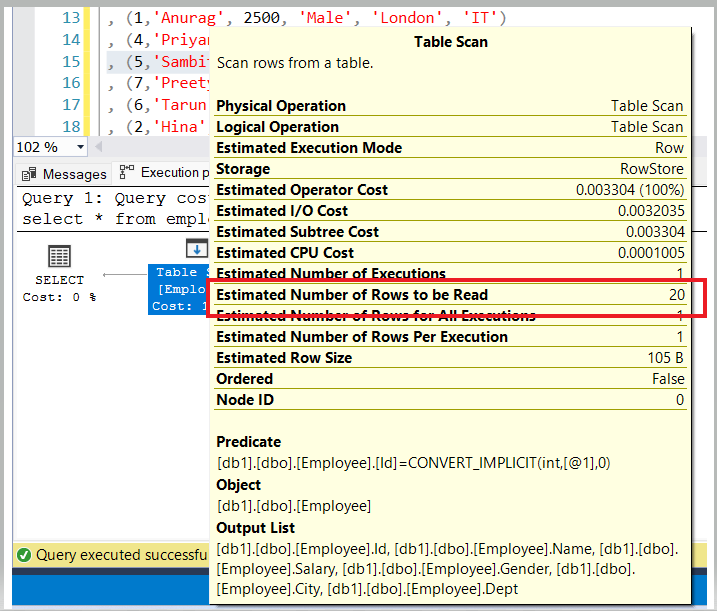
Now execute the below query

select \* from employee where id=22

A screenshot of a computer

Description automatically generated

Mouse hover to the table scan to see the no of rows to be scanned



Now create a clustered index on the id column on employee table

Create clustered index ix\_id on employee (id asc)

And execute the same above select statement and view the estimated execution plan

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

He we observer that the no of rows read and operation cost also decreased

To view the index on the columns in a table we can use the following syntax

sp\_helpindex Employee

or

Execute sp\_helpindex Employee

**Types of indexes in sql server**

1. **Clustered index**
2. **Non clustered index**

**Clustered index**

* A clustered index determines the physical order of data rows in a table.
* There can be only one clustered index per table, and it defines the logical order of the data.
* By default, when a primary key is defined on a table, it becomes the clustered index.
* If no clustered index is defined, the table is referred to as a "heap," and data rows are not stored in any specific order.
* The data pages in a clustered index are stored in a B-tree structure based on the key columns of the index.
* A clustered index is particularly efficient for range queries, as it minimizes I/O operations when retrieving contiguous rows.
* It may require periodic maintenance to reorganize or rebuild the index to maintain its efficiency.
* To create a clustered index Create clustered index ix\_id on employee (id asc)

**Non-Clustered Index:**

* A non-clustered index is a separate structure from the data rows and does not determine the physical order of data in a table.
* A table can have multiple non-clustered indexes.
* Non-clustered indexes store a copy of the indexed column(s) along with a reference to the corresponding data row's location in the table.
* These indexes are useful for optimizing query performance because they provide a quick way to find the rows that match specific criteria.
* Non-clustered indexes are suitable for various types of queries, including filtering, sorting, and joining operations.
* They do not require maintenance to the same extent as clustered indexes.
* CREATE NONCLUSTERED INDEX index\_name

ON table\_name (column1, column2, ...);

**Differences between Clustered and Non-Clustered Indexes:**

**Physical Order**: A clustered index determines the physical order of data in a table, while a non-clustered index does not.

**Unique Constraint**: A clustered index enforces a unique constraint on the key column(s), while non-clustered indexes can be created on non-unique columns.

Storage: Clustered indexes are stored with the data rows, making the table size larger, while non-clustered indexes are stored separately.

**Number of Indexes**: Each table can have only one clustered index, but multiple non-clustered indexes.

**Key Columns**: In a clustered index, the key columns are the actual data rows. In non-clustered indexes, the key columns are used to look up the data rows.

**Query Optimization**: Clustered indexes are efficient for range queries and can improve the performance of some SELECT, UPDATE, and DELETE operations. Non-clustered indexes are versatile and can enhance the performance of various types of queries.

clustered and non-clustered indexes serve different purposes in SQL Server. A clustered index determines the physical order of data and is most useful for range queries, while non-clustered indexes provide faster access to data rows based on specific column values and are versatile for optimizing various query types. The choice between them depends on your specific data retrieval and performance optimization requirements.