## **Database Index**

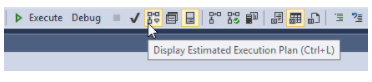
**SQL Server Indexes**

### Introduction

Indexes are special data structures associated with tables or views that help speed up the query. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

SQL Server has two types of indexes: clustered index and non-clustered index.

Note that to display the estimated execution plan in SQL Server Management Studio, you click the Display Estimated Execution Plan button or select the query and press the keyboard shortcut Ctrl+L:



### SQL Server Clustered Indexes

A clustered index stores data rows in a sorted structure based on its key values. Each table has only one clustered index because data rows can be only sorted in one order. The table that has a clustered index is called a clustered table.

The syntax for creating a clustered index is as follows:

CREATE CLUSTERED INDEX index\_name ON schema\_name.table\_name (column\_list);

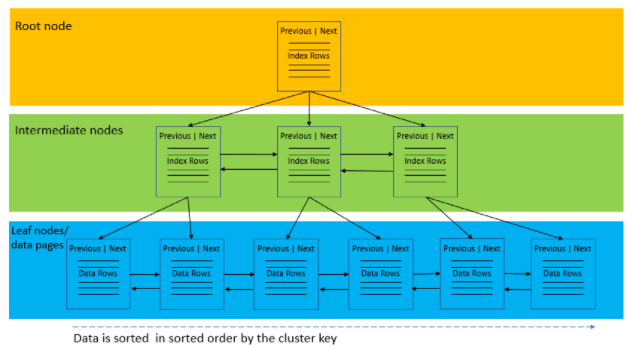
A clustered index organizes data using a special structured so-called B-tree (or balanced tree) which enables searches, inserts, updates, and deletes in logarithmic amortized time.

In the B-Tree, the root node and intermediate level nodes contain index pages that hold index rows. The leaf nodes contain the data pages of the underlying table. The pages in each level of the index are linked using another structure called a doubly-linked list.

When you create a table with a primary key, SQL Server automatically creates a corresponding clustered index based on columns included in the primary key.

If you add a primary key constraint to an existing table that already has a clustered index, SQL Server will enforce the primary key using a non-clustered index.

The following picture illustrates the structure clustered index:



### SQL Server Non-Clustered Indexes

A nonclustered index is a data structure that improves the speed of data retrieval from tables. Unlike a clustered index, a nonclustered index sorts and stores data separately from the data rows in the table. It is a copy of selected columns of data from a table with the links to the associated table.

To create a non-clustered index, you use the CREATE INDEX statement:

CREATE [NONCLUSTERED] INDEX index\_name ON table\_name(column\_list);

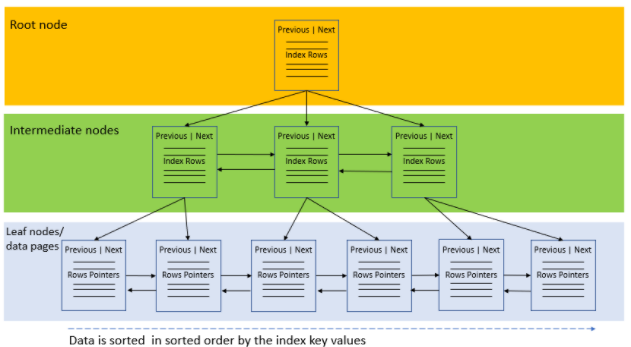
Similar to a clustered index, a nonclustered index uses the B-tree structure to organize its data.

A table may have one or more nonclustered indexes and each non-clustered index may include one or more columns of the table.

Besides storing the index key values, the leaf nodes also store row pointers to the data rows that contain the key values. These row pointers are also known as row locators.

When you create a nonclustered index that consists of multiple columns, the order of the columns in the index is very important. You should place the columns that you often use to query data at the beginning of the column list.

The following picture illustrates the structure non-clustered index:

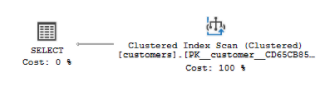


### SQL Server CREATE INDEX Statement Example

This statement finds customers who locate in Atwater:

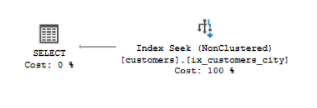
SELECT customer\_id, city FROM sales.customers WHERE city = 'Atwater';

If you display the estimated execution plan, you will see that the query optimizer scans the clustered index to find the row. This is because the sales.customers table does not have an index for the city column.



To improve the speed of this query, you can create a new index named ix\_customers\_city for the city column:

CREATE INDEX ix\_customers\_city ON sales.customers(city);

Now, if you display the estimated execution plan of the above query again, you will find that the query optimizer uses the nonclustered index ix\_customers\_city:  


### SQL Server Unique Index

A unique index ensures the index key columns do not contain any duplicate values.

A unique index can be clustered or non-clustered.

To create a unique index, you use the CREATE UNIQUE INDEX statement as follows:

CREATE UNIQUE INDEX index\_name ON table\_name(column\_list);

**SQL Server Unique Index and NULL**

NULL is special. It is a marker that indicates the missing information or not applicable.

SQL Server treats NULL values that if you create a unique index on a nullable column, you can only have only one NULL value in this column.

**Unique Index vs. UNIQUE Constraint**

Both unique index and UNIQUE constraint enforce the uniqueness of values in one or many columns.

When you create a unique constraint, behind the scene, SQL Server creates a unique index associated with this constraint.

### SQL Server Disable Index Statements

To disable an index, you use the ALTER INDEX statement as follows:

ALTER INDEX index\_name ON table\_name DISABLE;

To disable all indexes of a table, you use the following form of the ALTER INDEX statement:

ALTER INDEX ALL ON table\_name DISABLE;

If you disable an index, the query optimizer will not consider that disabled index for creating query execution plans.

If you disable a clustered index of a table, you cannot access the table data using data manipulation language such as SELECT, INSERT, UPDATE, and DELETE until you rebuild or drop the index.

### SQL Server Enable Indexes

Sometimes, you need to disable an index before doing a large update on a table. By disabling the index, you can speed up the update process by avoiding the index writing overhead.

After completing the update to the table, you need to enable the index. Since the index was disabled, you can rebuild the index but cannot just simply enable it. Because after the update operation, the index needs to be rebuilt to reflect the new data in the table.

In SQL Server, you can rebuild an index by using the ALTER INDEX statement or DBCC DBREINDEX command.

This statement uses the ALTER INDEX statement to “enable” or rebuild an index on a table:

ALTER INDEX index\_name ON table\_name REBUILD;

The following statement uses the ALTER INDEX statement to enable all disabled indexes on a table:

ALTER INDEX ALL ON table\_name REBUILD;

**Enable indexes using DBCC DBREINDEX statement**

This statement uses the DBCC DBREINDEX to enable an index on a table:

DBCC DBREINDEX (table\_name, index\_name);

This statement uses the DBCC DBREINDEX to enable all indexes on a table:

DBCC DBREINDEX (table\_name, " ");

### SQL Server DROP INDEX

The DROP INDEX statement removes one or more indexes from the current database. Here is the syntax of the DROP INDEX statement:

DROP INDEX [IF EXISTS] index\_name ON table\_name;

The DROP INDEX statement does not remove indexes created by PRIMARY KEY or UNIQUE constraints. To drop indexes associated with these constraints, you use the ALTER TABLE DROP CONSTRAINT statement.