Introduction to SQL Server clustered indexes

The following statement [creates a new table](https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/) named production.parts that consists of two columns part\_id and part\_name:

CREATE TABLE production.parts(

part\_id INT NOT NULL,

part\_name VARCHAR(100)

);

Code language: CSS (css)

And this statement [inserts some rows](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/) into the production.parts table:

INSERT INTO

production.parts(part\_id, part\_name)

VALUES

(1,'Frame'),

(2,'Head Tube'),

(3,'Handlebar Grip'),

(4,'Shock Absorber'),

(5,'Fork');

Code language: JavaScript (javascript)

The production.parts table does not have a [primary key](https://www.sqlservertutorial.net/sql-server-basics/sql-server-primary-key/), therefore, SQL Server stores its rows in an unordered structure called a **heap**.

When you query data from the production.parts table, the query optimizer needs to scan the whole table to locate the correct one.

For example, this statement finds the part whose part id is 5.

SELECT

part\_id,

part\_name

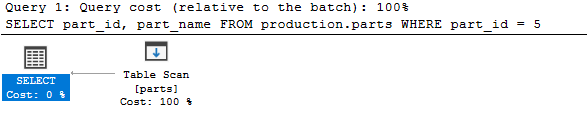
FROM

production.parts

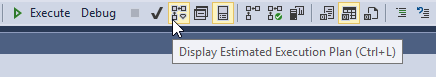
WHERE

part\_id = 5;

If you display the estimated execution plan in SQL Server Management Studio, you can see how SQL Server came up with the following query plan:



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:



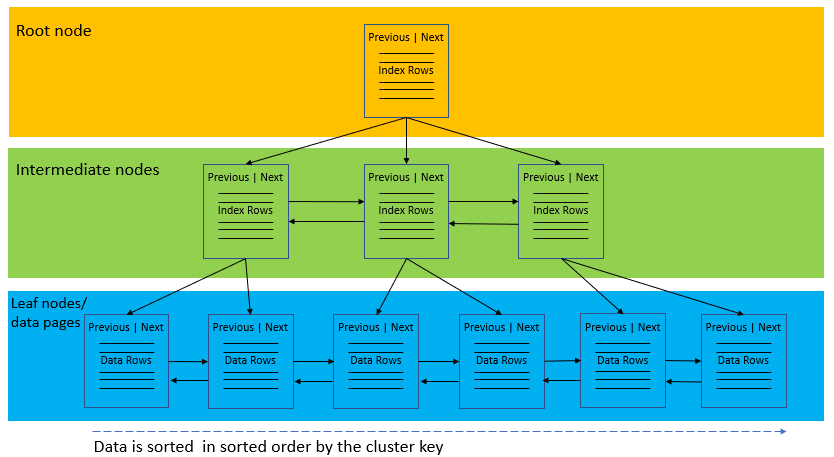
Because the production.parts table has only five rows, the query will execute very fast. However, if the table contains a large number of rows, It will take a lot of time and resources to search for data.

To resolve this issue, SQL Server provides a dedicated structure to speed up retrieval of rows from a table called index.

SQL Server has two types of indexes: clustered index and non-clustered index. We will focus on the clustered index in this tutorial.

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 following picture illustrates the structure of a clustered index:



A clustered index organizes data using a special structured so-called [B-tree](https://en.wikipedia.org/wiki/B-tree) (or balanced tree) which enables searches, inserts, updates, and deletes in logarithmic amortized time.

In this structure, the top node of the B-tree is called the **root node**. The nodes at the bottom level are called the **leaf nodes**. Any index levels between the root and the leaf nodes are known as intermediate levels.

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.

SQL Server Clustered Index and Primary key constraint

When you create a table with a [primary key](https://www.sqlservertutorial.net/sql-server-basics/sql-server-primary-key/), SQL Server automatically creates a corresponding clustered index based on columns included in the primary key.

This statement [creates a new table](https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/) named production.part\_prices with a primary key that includes two columns: part\_id and valid\_from.

CREATE TABLE production.part\_prices(

part\_id int,

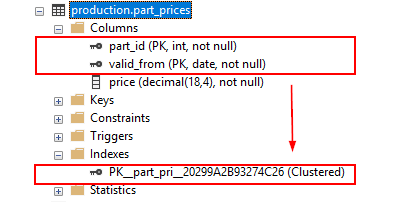
valid\_from date,

price decimal(18,4) not null,

PRIMARY KEY(part\_id, valid\_from)

);

Code language: JavaScript (javascript)



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:

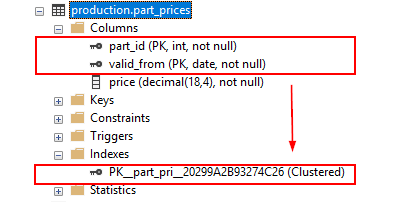
This statement defines a primary key for the production.parts table:

ALTER TABLE production.parts

ADD PRIMARY KEY(part\_id);

Code language: CSS (css)

SQL Server created a non-clustered index for the primary key.



Using SQL Server CREATE CLUSTERED INDEX statement to create a clustered index.

In case a table does not have a primary key, which is very rare, you can use the CREATE CLUSTERED INDEX statement to define a clustered index for the table.

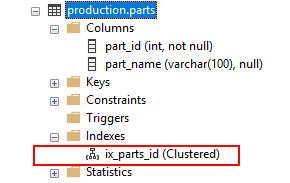
The following statement creates a clustered index for the production.parts table:

CREATE CLUSTERED INDEX ix\_parts\_id

ON production.parts (part\_id);

Code language: CSS (css)

If you open the **Indexes** node under the table name, you will see the new index name ix\_parts\_id with type Clustered.



When executing the following statement, SQL Server traverses the index (Clustered index seek) to locate the row, which is faster than scanning the whole table.

SELECT

part\_id,

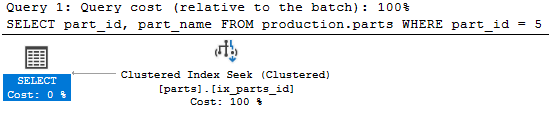
part\_name

FROM

production.parts

WHERE

part\_id = 5;



SQL Server CREATE CLUSTERED INDEX syntax

The syntax for creating a clustered index is as follows:

CREATE CLUSTERED INDEX index\_name

ON schema\_name.table\_name (column\_list);

Code language: CSS (css)

In this syntax:

* First, specify the name of the clustered index after the CREATE CLUSTERED INDEX clause.
* Second, specify the schema and table name on which you want to create the index.
* Third, list one or more columns included in the index.

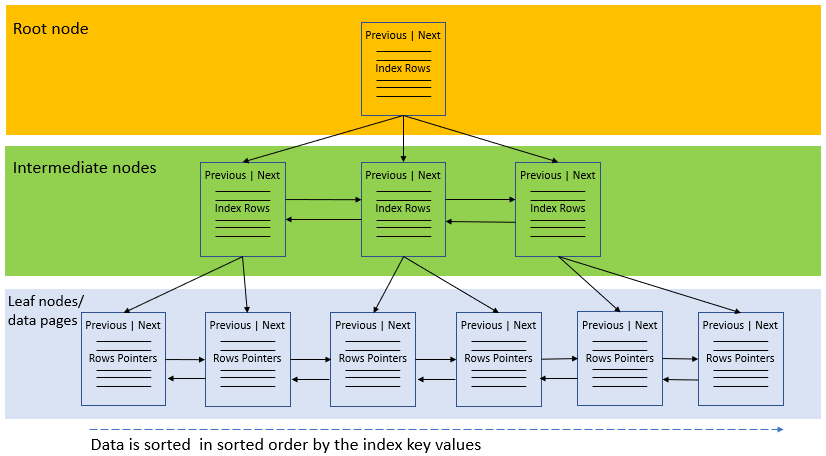
## Introduction to SQL Server non-clustered indexes

A nonclustered index is a data structure that improves the speed of data retrieval from tables. Unlike a [clustered index](https://www.sqlservertutorial.net/sql-server-indexes/sql-server-clustered-indexes/), 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.

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.

The following picture illustrates the structure non-clustered index:



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.

If the underlying table is a clustered table, the row pointer is the clustered index key. In case the underlying table is a heap, the row pointer points to the row of the table.

## SQL Server CREATE INDEX statement

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

CREATE [NONCLUSTERED] INDEX index\_name

ON table\_name(column\_list);

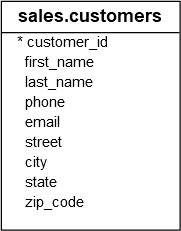
Code language: SQL (Structured Query Language) (sql)

In this syntax:

* First, specify the name of the index after the CREATE NONCLUSTERED INDEX clause. Note that the NONCLUSTERED keyword is optional.
* Second, specify the table name on which you want to create the index and a list of columns of that table as the index key columns.

## SQL Server CREATE INDEX statement examples

We will use the sales.customers from the [sample database](https://www.sqlservertutorial.net/sql-server-sample-database/) for the demonstration.



The sales.customers table is a clustered table because it has a [primary key](https://www.sqlservertutorial.net/sql-server-basics/sql-server-primary-key/) customer\_id.

### **A) Using the SQL Server CREATE INDEX statement to create a nonclustered index for one column example**

This statement finds customers who locate in Atwater:

SELECT

customer\_id,

city

FROM

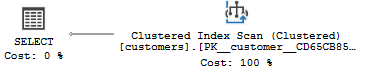
sales.customers

WHERE

city = 'Atwater';

Code language: SQL (Structured Query Language) (sql)

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.



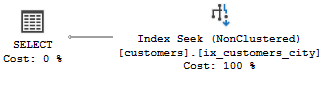
To improve the speed of this query, you can create a new index named ix\_customers\_city for the city column:

CREATE INDEX ix\_customers\_city

ON sales.customers(city);

Code language: SQL (Structured Query Language) (sql)

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:



### **B) Using SQL Server CREATE INDEX statement to create a nonclustered index for multiple columns example**

The following statement finds the customer whose last name is Berg and first name is Monika:

SELECT

customer\_id,

first\_name,

last\_name

FROM

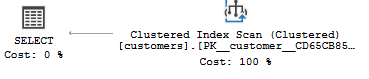
sales.customers

WHERE

last\_name = 'Berg' AND

first\_name = 'Monika';

Code language: SQL (Structured Query Language) (sql)



The query optimizer scans the clustered index to locate the customer.

To speed up the retrieval of data, you can create a nonclustered index that includes both last\_name and first\_name columns:

CREATE INDEX ix\_customers\_name

ON sales.customers(last\_name, first\_name);

Code language: SQL (Structured Query Language) (sql)

Now, the query optimizer uses the index ix\_customers\_name to find the customer.

SELECT

customer\_id,

first\_name,

last\_name

FROM

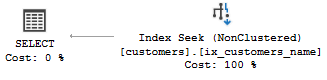
sales.customers

WHERE

last\_name = 'Berg' AND

first\_name = 'Monika';

Code language: SQL (Structured Query Language) (sql)



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.

For example, the following statement finds customers whose last name is Albert. Because the last\_name is the leftmost column in the index, the query optimizer can leverage the index and uses the index seek method for searching:

SELECT

customer\_id,

first\_name,

last\_name

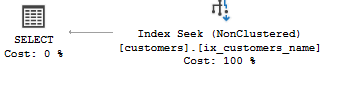
FROM

sales.customers

WHERE

last\_name = 'Albert';

Code language: SQL (Structured Query Language) (sql)



This statement finds customers whose first name is Adam. It also leverages the ix\_customer\_name index. But it needs to scan the whole index for searching, which is slower than index seek.

SELECT

customer\_id,

first\_name,

last\_name

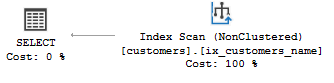
FROM

sales.customers

WHERE

first\_name = 'Adam';

Code language: SQL (Structured Query Language) (sql)



Therefore, it is a good practice to place the columns that you often use to query data at the beginning of the column list of the index.