What is the difference between a Clustered and Non Clustered Index?

A clustered index determines the order in which the rows of a table are stored on disk. If a table has a clustered index, then the rows of that table will be stored on disk in the same exact order as the clustered index. An example will help clarify what we mean by that.

An example of a clustered index

Suppose we have a table named Employee which has a column named EmployeeID. Let's say we create a clustered index on the EmployeeID column. What happens when we create this clustered index? Well, all of the rows inside the Employee table will be ***physically*** – sorted (on the actual disk) – by the values inside the EmployeeID column. What does this accomplish? Well, it means that whenever a lookup/search for a sequence of EmployeeID's is done using that clustered index, then the lookup will be much faster because of the fact that the sequence of employee ID's are physically stored right next to each other on disk – that is the advantage with the clustered index. This is because the **rows in the table are sorted in the exact same order as the clustered index**, and the actual table data is stored in the leaf nodes of the clustered index.

Remember that an index is usually a tree data structure – and leaf nodes are the nodes that are at the very bottom of that tree. In other words, ***a clustered index basically contains the actual table level data in the index itself***. This is very different from most other types of indexes as you can read about below.

When would using a clustered index make sense?

Let's go through an example of when and why using a clustered index would actually make sense. Suppose we have a table named Owners and a table named Cars. This is what the simple schema would look like – with the column names in each table:

Owners
Owner_Name
Owner_Age

Cars
Car_Type
Owner_Name

Let's assume that a given owner can have multiple cars – so a single Owner_Name can appear multiple times in the Cars table. Now, let's say that we create a clustered index on the Owner_Name column in the Cars table. What does this accomplish for us? Well, because a clustered index is stored physically on the disk in the same order as the index, it would mean that a given Owner_Name would have all his/her car entries stored right next to each other on disk. In other words, if there is an owner named "Joe Smith" or "Raj Gupta", then each owner would have all of his/her entries in the Cars table stored right next to each other on the disk.

When is using a clustered index an advantage?

What is the advantage of this? Well, suppose that there is a frequently run query which tries to find all of the cars belonging to a specific owner. With the clustered index, since all of the car entries belonging to a single owner would be right next to each other on disk, ***the query will run much faster than if the rows were being stored in some random order on the disk.*** And that is the key point to remember!

Why is it called a clustered index?

In our example, all of the car entries belonging to a single owner would be right next to each other on disk. This is the "clustering", or grouping of similar values, which is referred to in the term "clustered" index.

Note that having an index on the Owner_Name would not necessarily be unique, because there are many people who share the same name. So, you might have to add another column to the clustered index to make sure that it's unique.

What is a disadvantage to using a clustered index?

A disadvantage to using a clustered index is the fact that if a given row has a value updated in one of it's (clustered) indexed columns what typically happens is that the database will have to move the entire row so that the table will continue to be sorted in the same order as the clustered index column. Consider our example above to clarify this. Suppose that someone named "Rafael Nadal" buys a car – let's say it's a Porsche – from "Roger Federer". Remember that our clustered index is created on the Owner_Name column. This means that when we do a update to change the name on that row in the Cars table, the Owner_Name will be changed from "Roger Federer" to "Rafael Nadal".

But, since a clustered index also tells the database in which order to physically store the rows on disk, when the Owner_Name is changed it will have to move an updated row so that it is still in the **correct sorted order**. So, now the row that used to belong to "Roger Federer" will have to be moved on disk so that it's grouped (or clustered) with all the car entries that belong to "Rafael Nadal". Clearly, this is a performance hit. This means that a simple UPDATE has turned into a DELETE and then an INSERT – just to maintain the order of the clustered index. For this exact reason, clustered indexes are usually created on primary keys or foreign keys, because of the fact that those values are less likely to change once they are already a part of a table.

A comparison of a non-clustered index with a clustered index with an example

As an example of a non-clustered index, let's say that we have a non-clustered index on the EmployeeID column. A non-clustered index will store both the value of the EmployeeID **AND**a *pointer* to the row in the Employee table where that value is actually stored. But a clustered index, on the other hand, will actually store the row data for a particular EmployeeID – so if you are running a query that looks for an EmployeeID of 15, the data from other columns in the table like EmployeeName, EmployeeAddress, etc. will all actually be stored in the leaf node of the clustered index itself.

This means that with a non-clustered index **extra work** is required to follow that pointer to the row in the table to retrieve any other desired values, as opposed to a clustered index which can just access the row directly since it is being stored in the same order as the clustered index itself. So, reading from a clustered index is generally **faster** than reading from a non-clustered index.

A table can have multiple non-clustered indexes

A table can have multiple non-clustered indexes because they don't affect the order in which the rows are stored on disk like clustered indexes.

Why can a table have only one clustered index?

Because a clustered index determines the order in which the rows will be stored on disk, having more than one clustered index on one table is impossible. Imagine if we have two clustered indexes on a single table – which index would determine the order in which the rows will be stored? Since the rows of a table can only be sorted to follow just one index, having more than one clustered index is not allowed.

Summary of the differences between clustered and non-clustered indexes

Here's a summary of the differences:

- A clustered index determines the order in which the rows of the table will be stored on disk – and it actually stores row level data in the leaf nodes of the index itself. A non-clustered index has no effect on which the order of the rows will be stored.
- Using a clustered index is an advantage when groups of data that can be clustered are frequently accessed by some queries. This speeds up retrieval because the data lives close to each other on disk. Also, if data is accessed in the same order as the clustered index, the retrieval will be much faster because the physical data stored on disk is sorted in the same order as the index.
- A clustered index can be a disadvantage because any time a change is made to a value of an indexed column, the subsequent possibility of re-sorting rows to maintain order is a definite performance hit.
- A table can have multiple non-clustered indexes. But, a table can have only one clustered index.
- Non clustered indexes store both a value and a pointer to the actual row that holds that value. Clustered indexes don't need to store a pointer to the actual row because of the fact that the rows in the table are stored on disk in the same exact order as the clustered index – and the clustered index actually stores the row-level data in it's leaf nodes.