**Indexing**

Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Col1(PK) | Col2 | Col3 | Col4 | Col5 |
|  |  |  |  |  |
|  |  |  |  |  |

1. **Clustered index**
   1. Sorts data physically based on columns on which index is created.
   2. Suppose index is created on Col1 then it physically stores table in database with col1 sorted. If you want to find col5 value respective to col1, in query keep condition on col1.

i.e. SELECT col5 from table where col1 = 254

* 1. **Always use less columns in clustered index**

1. **Non clustered index**
   1. Doesn’t sorts data physically but uses a separate table structure to maintain pointers. This table is sorted on key values

|  |  |
| --- | --- |
| Keys (on which index is based) value | Pointer for lookup |
|  |  |
|  |  |

* 1. Disadvantage is lookup takes extra cost here.
  2. **In clustered index, it directly finds the required key (in filter condition) from b-tree and uses that record.  
     But in non-clustered index, it first finds the required key in the additional table structure and then uses pointer to lookup to actual record.**

1. **Covering index**
   1. This is extension of non-clustered index.
   2. Here if you want that lookup cost is saved, you can add required column into the query using *‘INCLUDE’*
   3. Example if you create non-clustered index on col1 then query: SELECT col5 from table where col1 = 254, will create an extra lookup to the actual row for fetching col5 value.  
      IF covering index is created on col5 then col5 will be saved to the above additional table structure, which then doesn’t require lookup.
2. **Column Store**
   1. Every index had purpose to search a set of rows among all records, but if you are sure to read all rows but specific column then this is used
   2. it stores the table as column wise in memory
   3. Example : you want the sum on col5 then use this type.