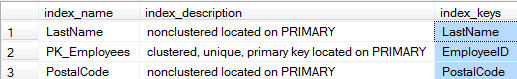
Task 1:

D)



F)

Select Count(i.name) as [Indices Count]

From sysobjects AS t JOIN sysindexes AS i ON t.id = i.id

where t.id > 100 and t.name = 'Customers'

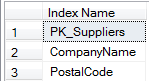


G)

Select i.name as [Index Name]

From sysobjects as t join sysindexes as i on t.id = i.id

where t.id > 100 and t.name = 'Suppliers'



Task 2:

C)

Create nonclustered index Orders\_Customers\_link on Orders (CustomerID)

with (Fillfactor = 75);

go

F)

Create nonclustered index Products\_SupplierID\_link on Products (SupplierID)

with (fillfactor = 100);

go

Task 3

C)

i) Select Products.ProductName, Products.UnitPrice

from Products

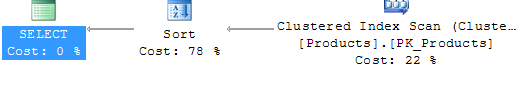
where UnitPrice > 10

order by UnitPrice desc

Update Products

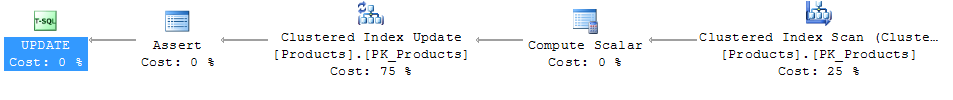
set Products.UnitPrice = Products.UnitPrice \* 1.02

ii)



iii) Sorting

iv) There is no index on unitprice so it must iterate over the elements and impose an ordering on them.

v) 

vi) The clustered index update (Products.Pk\_Products).

vii) Whatever index that the program can use to find all the entries such that we can update them for unit price, it must go back through and update all of those entries to contain the new unit price that we have prescribed.

E)

i) No there was no change.

ii) There is no change because the query must still go through the clustered index that is already existing. The SQL database prefers to use the clustered list because it has to fetch a attribute that is not kept in the non-clustered index, and therefore will have to go back through the clustered list anyways.

iii) Yes the execution plan changed. It splits the computing scalar phase into 3, or at least does 3 computer scalar phases.

iv) Since we now can be ordered on the non-clustered index and on the clustered index we must keep both of them updated as we update the rows of the table, which means we need to update each entry in both indices, this causes a larger amount of overhead in the update stage to ensure database health.

v) Clustered index Updating (86%)

Task 4

1. Indices are imposing orders on data in tables. They allow for tree like searching, although typically not log(2)n, they are more likely log(8)n, as that is the number of pointers that can be stored per page.
2. If you will use a query that must search along this attribute that is easily divided often.
3. If you are going to divide into groups that would require you to traverse most of the data anyways. See: in-class example of creating an index by gender.
4. A clustered index is ordered by index value and data value, a non-clustered index is only ordered by index value.
5. How much empty space remains in each block before resizing, this is used to minimize the overhead of inserting / creating blocks.