## Chapter 4.6 Algorithm based on Index

***Instruction:***

Several kinds of Algorithm need Index on one or more fields in Relation. Selection Operation based on Index is especially useful, but for Join Algorithm and other Binary Operator Algorithm also have some benefits. In this chapter, we will introduce the algorithm.

### Chapter 4.6.1 Cluster and Non - Cluster Index

***Definition:***

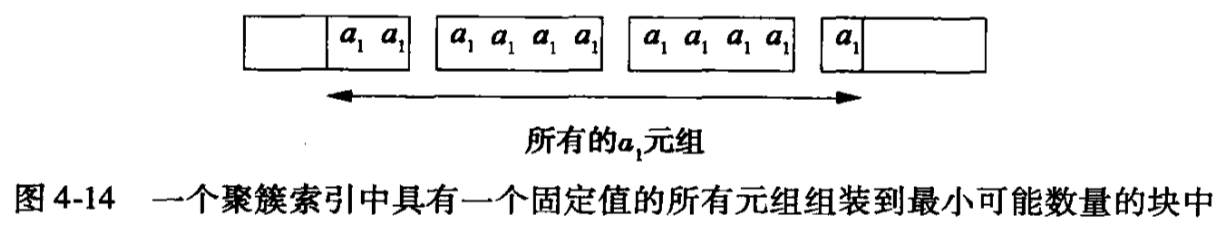
1. Cluster - If the number of tuple in one Relation can be contracted to store as little as possible, then this Relation is Cluster. All analysis so far are all based on cluster.
2. Cluster Index on one or more attributes, then all tuples with the fixed value can be put into little blocks.

***Attention:***

Non - cluster Relation can not have one cluster index, but one Cluster Relation can have non - cluster index.

***Example:***

* Relation R(a, b), sorted by property a and stored based on this sequence, then if Relation R(a, b) stores into the block, it must be cluster. For one given property *a* - with value *a1*, all tuples with value *a1* are stored sequentially, so the index on property *a* is cluster index.
* See the picture below, except the first and the last block which contain the value *a1*, they are all stored on the block. But actually, index on the property *b* may not be cluster, since there has one tuple with the fixed value *b* can be divided into the file, except that there has close relationship with property *a* and *b*.



### Chapter 4.6.2 Selection based on Index

***Principle:***

Think about when tuples that satisfy the condition C which equals to a = v, here a is one property with the value of v. Here using the value v to search index and get those tuples that satisfy the conditions, after that we just get them back.

***Situation 1 - Cluster:***

If the index on Relation R.a is Cluster, then the number of disk I/O that getting back collection which satisfies condition *a = v* equals nearly to *B(R) / V(R, a)*, but actually the number may be higher, because:

1. The index may not be all stored into main memory, it needs some disk I/O to support Query on index.
2. Even all tuples with the condition *a = v* can be stored into the b blocks, it may be divided into b + 1 blocks, since they may not start from one block.
3. Although tuples of Relation R is cluster, they may not be fully stored into blocks.
4. If *B(R) / V(R, a)* is not a integer, then we need to round up to an integer. If property a is one key of Relation R, then *V(R, a) = T(R)*, then we can suppose it must be bigger than *B(R)*. Then we need one disk I/O and another disk I/O to access the index.

***Situation 2 - Non - Cluster:***

Think about when the index on R.a is non - cluster, then what happens next. If each tuple we get back is on different blocks, and we must access T(R) / V(R, a) tuples. Therefore T(R) / V(R, a) is the estimated disk I/O times. Although the number may be higher, since we need to read some index blocks from disk; Also it may be lower, since these tuples may be appear in one block and this block exists in the main memory.

***Example:***

B(R) = 1000, T(R) = 20000, then there has 20000 tuples which has been put into 20 blocks, assume that a is one property of Relation R and there has an index on the property a. Then for Selection Operation when a = 0.

Below are possible situations and the numbers of disk I/O under the worst situation:

1. Relation R is cluster, but we do not use index, then the cost will be 1000 times disk I/O.
2. If Relation R is not cluster and we can not use index, then the cost will be 2000 times disk I/O. So we must retrieve each block of Relation R.
3. If V(R, a) = 100 and the index is cluster, then the algorithm based on index needs 1000 / 100 = 10 disk I/O, add the cost to access the index.
4. If V(R, a) = 100 and the index is non - cluster, then the algorithm based on index needs 20000 / 100 = 2000 times disk I/O.
5. If V(R, a) = 20000, which is to say that property a is a key, then considering the Algorithm based on the index, no matter the index is cluster or non - cluster, then we needs one disk I/O and the disk I/O to access the index.

***Addition:***

1. One index, such as B tree, can make us access the Query Key within the given range. If there has one index on property a on the Relation R, then we can use index to retrieve all needed tuples of Relation R within the condition of a >= 10, a <= 10 and a <= 20.
2. For one more complex Selection Operation with much more complex condition C, then using method below to realize: after scanning all index and getting tuples, then operate another Selection Operation on the result tuples.

### Chapter 4.6.3 Connection by Using Index

***Instruction:***

In this Chapter, we will consider the Natural Connection R(X, Y) Natural Join S(Y, Z).

***Principle:***

There exists an index on Property Y on Relation S. Check each block of the Relation R and consider each tuple t in each block.

Assume that tuple ty is value of the corresponding property Y. So we need to find the tuple ty in Relation S by using index. Luckily, these are all tuples that need to be connected with tuple t in Relation R, so we need to output each connection of ty in Relation R and Relation S.

***Cost:***

1. If Relation R is cluster, then we need to read B(R) blocks to get all tuples of Relation R. Otherwise, if Relation R is non - cluster, then we need to read T(R) disk I/O to get all tuples.
2. For each tuple in Relation R, then we need to read T(S) / V(S, Y) of Relation S. If Relation S is a non - cluster Relation, then the number of disk I/O equals to *T(R) \* T(S) / V(S, Y)*. But if the index is cluster, then the number of disk I/O equals to *T(R) \* B(S) / V(S, Y)*.*(Here, we need to add several disk I/O to read the index.)*

*In All: (Neglect the cost of Relation R)*

1. Relation S as non - cluster: *T(R) \* T(S) / V(S, Y)*.
2. Relation S as cluster: *T(R) \* ( max(1, B(S) / V(S, Y) ) )*.

***Example:***

R(X, Y) includes 1000 blocks and S(Y, Z) includes 500 blocks. Assume that one block contains 10 tuples of one Relation, so T(R) = 10 000, T(S) = 5000. V(S, Y) = 100, so the tuples in the Relation S has 100 different Y values.

1. Assume that Relation R is cluster, and Relation S on the property Y has one cluster index.
2. Therefore, the final disk I/O equals to 1000 times to access all blocks and *T(R) \* B(S) / V(S, Y) =* 10 000 \* 500 / 100 = 50 000 disk I/O. Totally, it equals to 1 000 + 50 000 = 51 000 times disk I/O.

### Chapter 4.6.4 Join by Using Ordered Index

***Principle:***

When calculating R(X, Y) Join S(X, Y), for Relation R and S, there has one Ordered index on property Y. There we can operate a normal Sort - Join but we do not need to operate the intermediate sort routing for either Relation.

The method is called *zig - zag connection*, because we need to query the shared Y - value.

*Attention:*

If Y - value does not appear in Relation R, then do not need to retrieve Y - value. Meanwhile, if Y - value in Relation S does not appear in Relation R, then does not need to retrieve Y - value.

***Example:***

1. Assume that we have Relation R(X, Y) and Relation S(Y, Z), these two Relation have index on property Y. Assume that Relation R has key search Y - value is the ordered 1, 3, 4, 4, 4, 5, 6, and the key search Y - value Of Relation S is 2, 2, 4, 4, 6, 7. Then we start from the first key value in Relation R and S, and they are 1 and 2. Because 1 < 2, then we skip the first key in Relation R, then look at the second key of Relation R, it is 3. Now, the current key 3 < 4, so we skip two 2 and reach at the key 4.
2. At that moment, the key 3 in Relation R is smaller, then we skip the key of Relation R. Now, the current key value of two Relation are both 4. So now we need to search along the key 4 and query the corresponding tuple, and connect them. *(Until we meet the common key 4, we do not need to search tuples in Relations.)*
3. After we finish the key 4, then go to key 5 of Relation R and key 6 of Relation S, since 5 < 6, then we skip to the next key of Relation R. Now the key is 6, and because we search the corresponding tuples and connect them. Since Relation R is empty, then we know there has no tuples that need to connect in both Relations.

***Example:***

See what happen when using Sort and Index.

1. First, assume that there has an index on property Y in Relation S which permits us to query the tuple sorting according to property Y. Also assume that the Relation and Index are all cluster. There has no index on property R.
2. Assume that there has 101 blocks in main memory, we can use them to build 10 sorted sub - tables for 1000 blocks in Relation R. The total disk I/O is 2000 to read and write all contents of Relation R.
3. We take 11 blocks of the main memory, 10 for the sorted sub - tables, one for saving the block of Relation S through Query. *(Here we neglect all disk I/O of operating index and main memory buffer.)*
4. Read all tuples of Relation R and Relation S which equals to 1500. So the total disk I/O equals to 1500 + 2000 = 3500.
5. Now assume that there has index on property Y on Relation R and Relation S. Then we do not need to sort any Relation. Then we use 1500 times disk I/O to read the blocks of Relation R and Relation S.