## Chapter 3.4 Multi - Dimension Index

*Definition:*

The Index which we discussed so far has only single dimension, which is to say, it just uses one Single Query Key to query the record according to the Key. Such as B - Tree, it needs Property Value on Query Key, then Query Process can be processed very smoothly; Otherwise, this Index can not play any role. The Single Key Space has several methods:

* *The Index on Sequence File and B - Tree all use Single, Sorted Sequence Key Value.*
* *Hash Table needs to know the Query Key before hand. If the Query Key consists of several fields, then even with one single unknown field, then we can not use Hash Table, but need to Query all Buckets.*

### Chapter 3.4.1 Application of Multi - Dimension Index

*Definition:*

Many applications needs us to treat Data as Binary or much more Dimension Space. Such application can be supported by Traditional DBMS System, but also some kinds are designed by Multi - Dimension Applications. The special aspect of such Application is that some kinds of Query Data Structure are used in the normal SQL application.\

*Application:*

One important application of Multi - Index is that it supports the process procedure of Geography Data. *GSI (Geography Information System) is one Two - Dimensional Space Storage Object, while object may be Dot or Shape. Normally, these Databases are Map, among which, the stored object may be house, road, bridge, pipe or other Physical Object.* The picture below is just like this.

*GSI (Geography Information System) is not ordinary SQL Query, although through some efforts, many queries can be present as the format of SQL.* Such Queries are as below:

1. *Partial Matching Query:* *We need to assign values on Single - Dimension or Multi - Dimension, and match all points which equal to points on these Dimensions.*
2. *Range Query:* *We need to assign range on Single - Dimension or Multi - Dimension, and find the exact range which matches the assigned range.* Or it has such possibility that when it presents the Shape, we need to find partial or the whole shape collection which satisfies range condition.
3. *The Nearest Neighbor Query:* *We need to query the nearest point from the given point.* For example, if the point presents City, then we may need to find one city which has more than 100 000 population compared with the given smallest city.
4. *Where - am - I Query:* *Given one point, and we need to know the shape of the given point.* Especially, when you click the mouse, the system would help decide which element would be shown when you click.

### Chapter 3.4.2 Range Query by Traditional Index

### *Definition:*

For simple Realization, assume that there had two Dimensions, which are x and y. We can set up Auxiliary Index for each Dimension. Setting up B - Tree for each Dimension would make the range Query very easy.

* *Given two - Binary range, at first, we can get all pointers in range of x through B - Tree Index. Then, we can get all pointers in the range of y through B - Tree Index.*
* *However, if Main Memory can hold these pointers, then the total Disk I/O would be the total number of Leaf Node that waited to be checked and little I/O Query Operations which query according to the B - Tree.*
* *We need to add the required Disk I/O when Querying, no matter how much are they.*

*Example:*

Let’s consider one assumption Dot Collection which consists of 1 000 000 points, and they are distributed randomly in the range of 0 - 1000 x / y spaces. Assume that each Storage Block can stores 100 points, and each Leaf Node contains 200 Key - Pointer Pairs. We can assume that there has B - Tree Index on x and y.

Image we have one range Query, which requires to find all points included in the edge 100 square, which is 450 <= x <= 550, 450 <= y <=550. By using the index on B - Tree Index, we can find all records pointers in this range, which about to have 100 000 pointers, so this number of pointers should stay in the Main Memory. Similar, we can use B - Tree index to get all record pointers in this range, which is nearly 100 000. The intersection of these two pointer collections would have 10 000 pointers, however, these 10 000 record pointers are our answers.

Now let’s answer the required I/O amount of this Query. First of all, it is useful to save the root of B - Tree into Main Memory. What we need to do is to check one Intermediate Level of Pointers and all included Leaf Node. Since we assume that the Leaf Node includes 200 Key - Pointers Pair, so for each B - Tree we need to check for about 500 *(100 000 / 200 = 500)* Leaf Node Storage Block. When we add one more Intermediate Node for each B - Tree, then the total Disk I/O we required would equal to 1002.

At last, we need to query the required Storage Block includes 10 000 records. If they are stored randomly, then we need to assume that the would be stored in 10 000 different Storage Blocks. *(1 000 000 / 100 = 10 000 Storage Blocks.)*

### Chapter 3.4.3 The Nearest Neighbor Query by Traditional Index

### Nearly all Data Structures which we use permit us to answer the nearest Neighbor Query:

1. *Choose one range from each Dimension.*
2. *Execute Range Query.*
3. *Select the nearest point which is close to target in this range.*

*Attention:*

* *There may has no point in the selected range.*
* *The nearest point in the range may be not always the nearest point.*

However, when we deal with this kind of problem, we need to estimate the nearest point range, execute the range query. If we can not find any point, then we need enlarge the query range, until we find at least one point. Then we need to consider that whether there has the much more nearer point out of the range. If yes, we need to enlarge the range query one more time, get all points in this range, and check it again.

### Chapter 3.4.4 Overview of Multi - Index Index