Paper Name: Database Management System

Topic : INDEXING

Paper Code: BCA CC410

Semester IV

By Ms. Manisha Prasad

Head, Assistant Professor,

Department of Computer Science

Patna Women's College

E – mail : manisha_prasad@yahoo.com

INDEXING

Let us assume we have a file having 10,000 records. We know that the records are organized in blocks. Let us assume each block contains 10 records. This means that the file will have 1000 data blocks. We would always want to access the records very fast. So to access a particular record, first we need to access the appropriate block and then we need to search the record within that block. To search the appropriate block takes time.

There can be two situations:- either Data File/Main File is sorted or it is not sorted. If the file is sorted we can implement Binary Search method to search the blocks. Suppose we want to search the record no 9454, in the worst case we will need at the most 10 searches to locate the record. If the file is unsorted, then we have to sequentially search all 1000 blocks in the worst case scenario. So if the file is very big, the time to search increases considerably, be the file sorted or unsorted.

The solution to this problem is Indexing. In this approach we create a separate small file called Index file in which we store the search key attribute and its block pointer. So we can define Indexing as:-

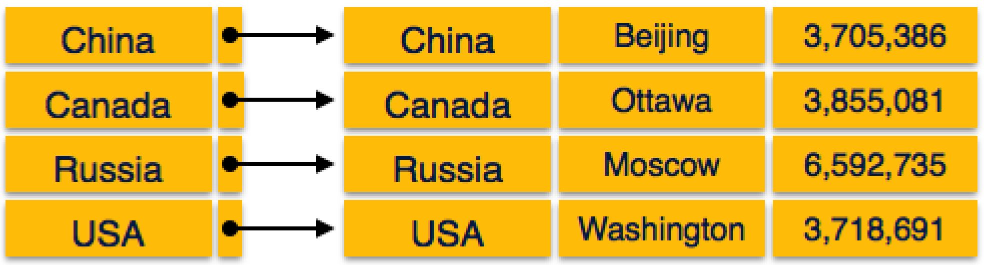
Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

Ordered Index is of two types −

- Dense Index
- Sparse Index

Dense Index

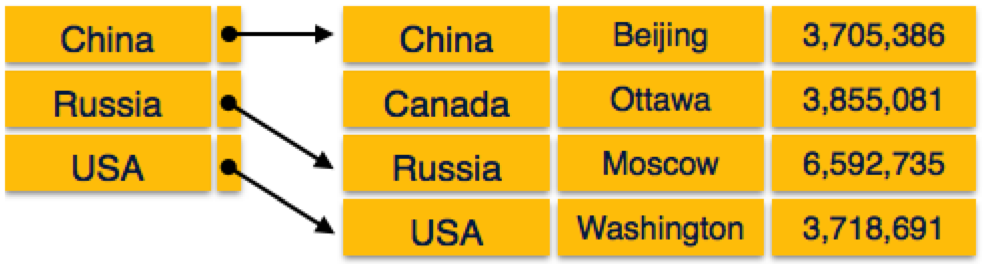
In the index file, if there is an index record for every search key value of the database, such an index is called dense index.This makes searching faster but requires more space to store index records itself. When the data file is unsorted we have to create Dense Index.



Index File Data File

Sparse Index

In sparse index, index records are not created for every search key value in the Index File. An index record here contains a search key and an actual pointer to the blocks on the disk. To search a record, we first proceed by index record to reach the appropriate block. If the data we are looking for is not where we directly reached by following the index, then the system starts sequential search within the block until the desired record is located. If we have sorted data file, then we can have Sparse Index.



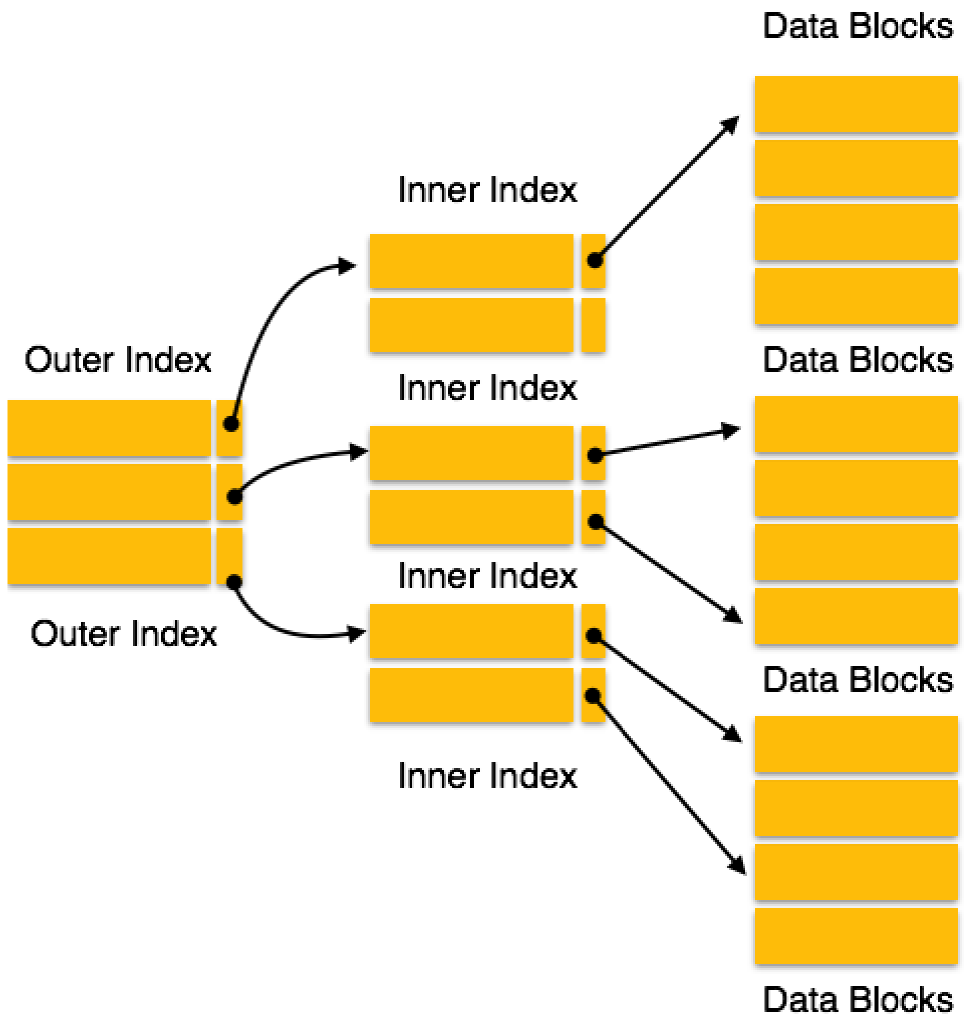
Index File Data File

Indexing can be of the following types −

- **Primary Index** – This method can be applied only on sorted Data file ordered on a **key field**. The key field is generally the Primary key of the relation. The indexing is done on this Primary Key attribute. Here we can have sparse index. The number of entries in the index file will be equal to the number of blocks in the data file
- **Clustering Index** − Clustering index can be applied on sorted data file, ordered on a non-key field having duplicate values. Here indexing is done on non key attribute. In this case the number of entries in the Index file will be equal to the number of unique value of the non-key attributes in the data file.
- **Secondary Index** – In both the above cases, data file was sorted but there are situations where the data file is unsorted. Here we can use Secondary indexing. Secondary index may be done on a key or a non-key attribute. The number of entries in the index file will be equal to the number of entries in the data file, so it is an example of dense index.

- Multilevel Index

  Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses. Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.


Data Blocks
Inner Index
Outer Index
Data Blocks
Inner Index
Inner Index
Outer Index
Data Blocks
Inner Index
Data Blocks