Indexing in DBMS

Storage and file structure: Memory Hierarchies and Storage Devices, Placing File Records on Disk, Hashing Techniques, Indexing Techniques (Primary Indexes, Secondary Indexes, Clustering Indexes, Multilevel Indexes, Dynamic Multilevel Indexes Using B-Trees and B+-Trees).

Indexing

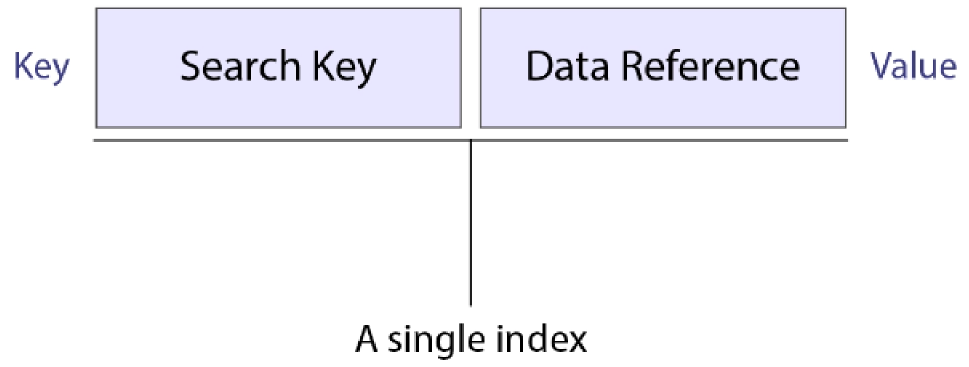
- If the records in the file are in sorted order, then searching will become very fast.
- But, in most the cases, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file.
- It indicates, the records are not in sorted order. To make searching faster in the files with unsorted records, indexing is used.

Indexing

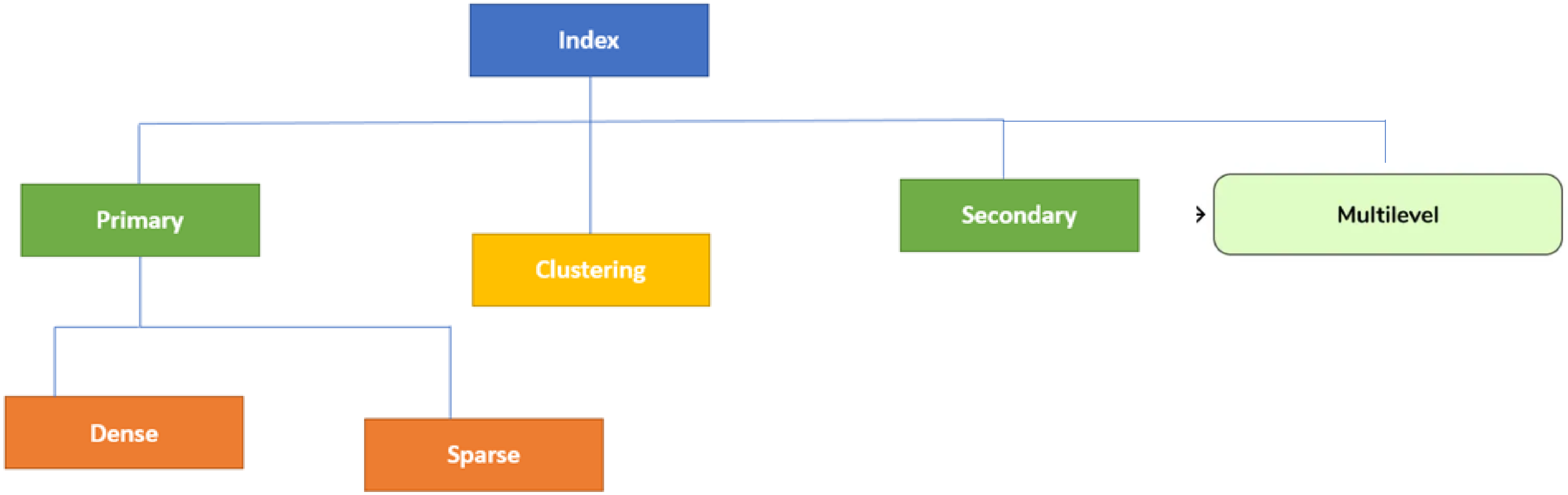
- It is a data structure technique that allows you to quickly retrieve records from a database file.
- An Index is a small table having only two columns. The first column contains a copy of the primary or candidate key of a table. The second column contains a set of disk block addresses where the record with that specific key value is stored.

Indexing

- Index usually consists of two columns which are a key-value pair. The two columns of the index table(i.e., the key-value pair) contain copies of selected columns of the tabular data of the database.



Indexing Methods



Primary Indexing

- Primary indexing helps in efficiently retrieving records based on their primary key values.
- It creates an index structure that maps primary key values to disk block addresses.
- The primary index consists of a sorted list of primary key values and their corresponding disk block pointers.
- It speeds up data retrieval by minimizing search efforts and is particularly useful for primary key-based queries.

Properties:

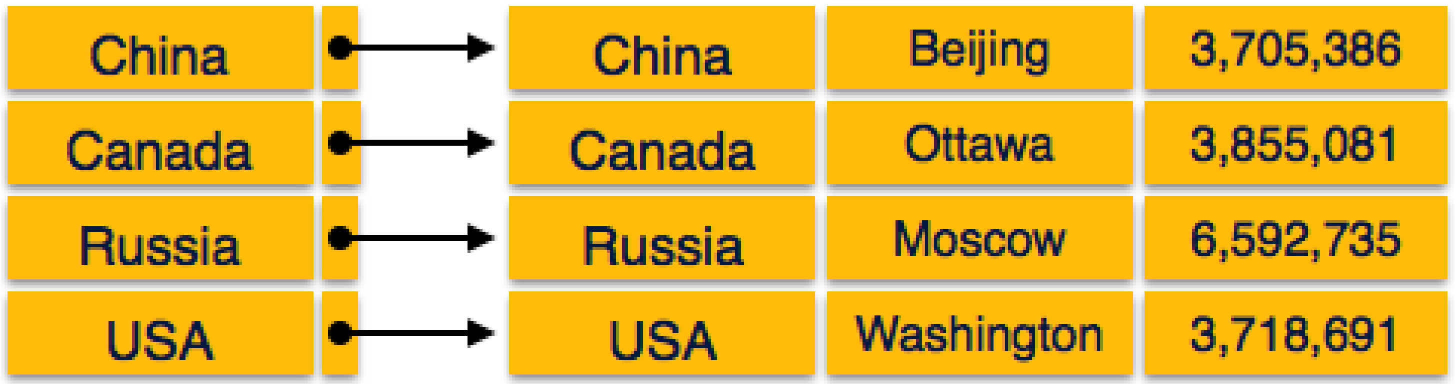
  - Primary index ensures the uniqueness of primary key values.
  - It provides efficient access to individual rows based on their primary key values.

- Example: Indexing a table on its primary key.

Primary Indexing

Dense index

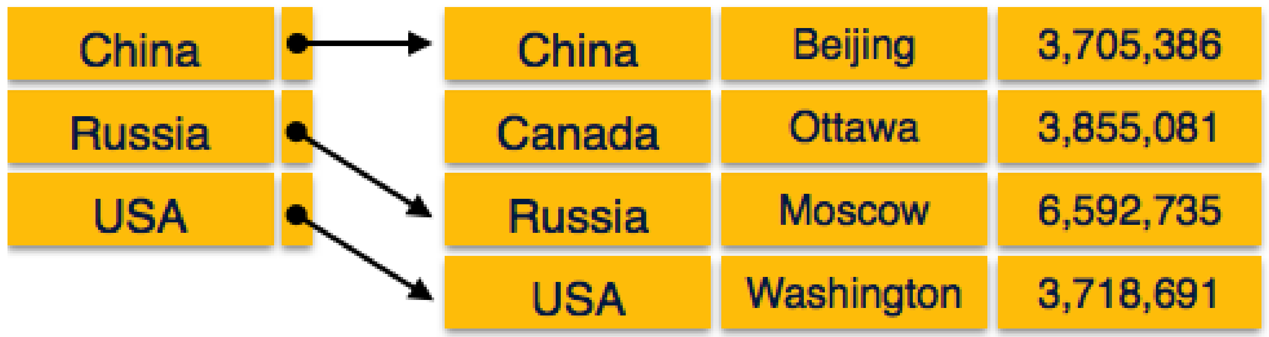
- An index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain a search key value and a pointer to the actual record on the disk.



Primary Indexing

Sparse index

- In the data file, the index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.
- If the data we are looking for is not where we directly reach by following the index, then the system starts a sequential search until the desired data is found.



Primary Indexing

Search operation is faster with Primary Indexes

No of access required in primary index of b blocks: $\text{Log}_2(\text{Blocks}) + 1$

Major problem: insertion and deletion of records

Move records around and change index values

Solutions:

- Use unordered overflow file
- Use linked list of overflow records

Clustering Index

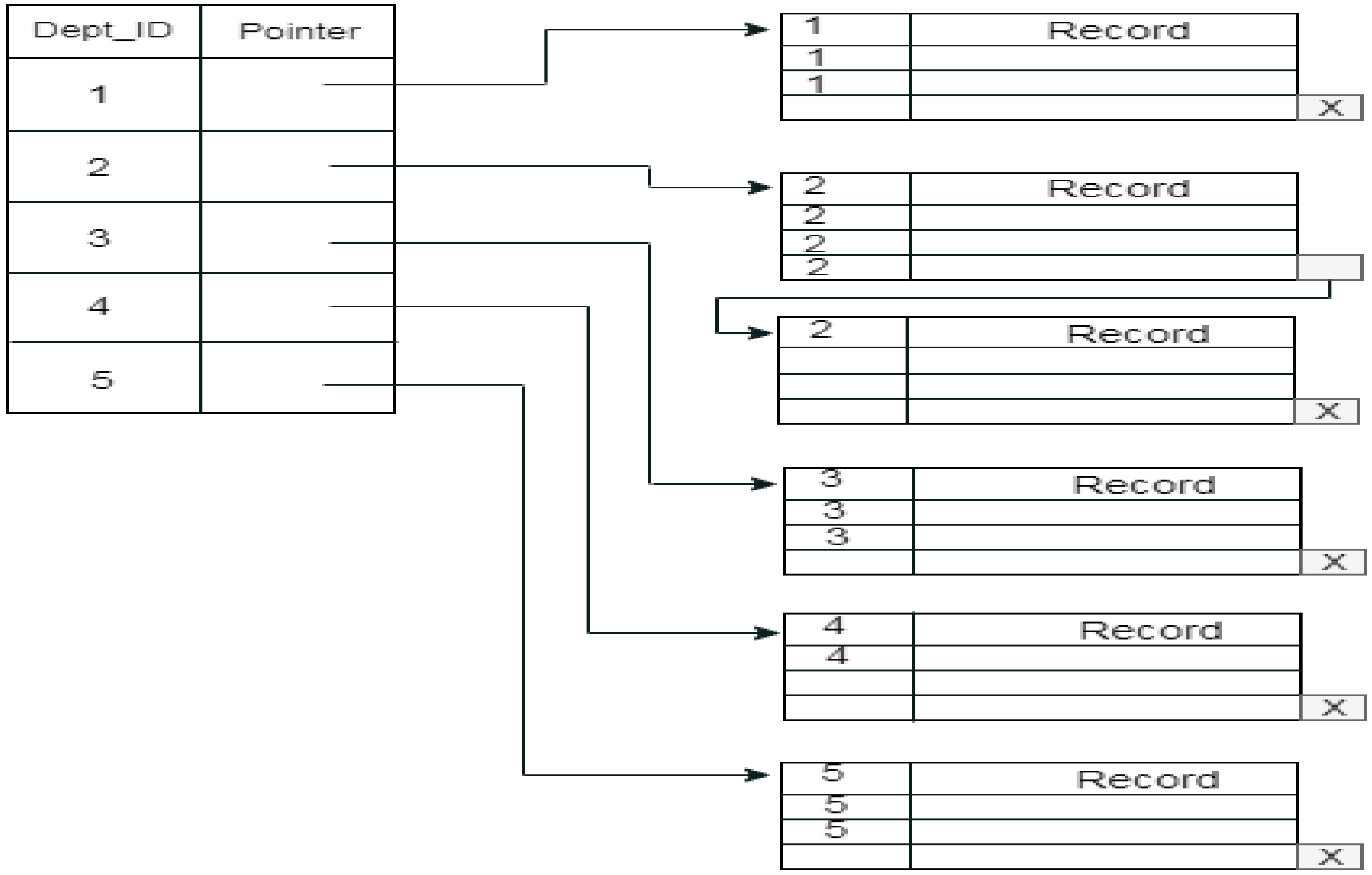
- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create an index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these groups.

Example

- suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees who belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.

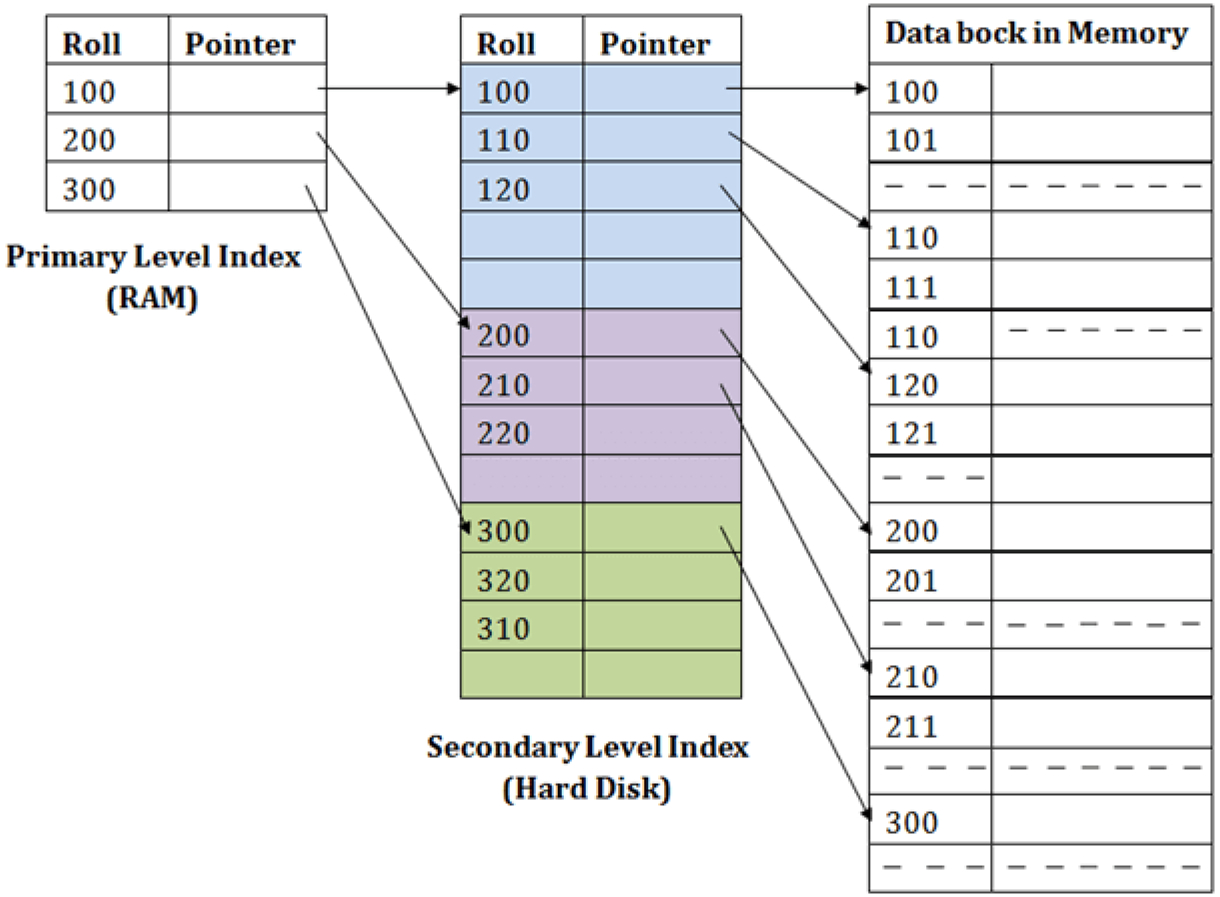
Clustering Index

Example



Secondary Index

- An index whose search key specifies an order different from the sequential order of the file. Also called the non-clustering index



Roll	Pointer
100	
200	
300	

Primary Level Index (RAM)

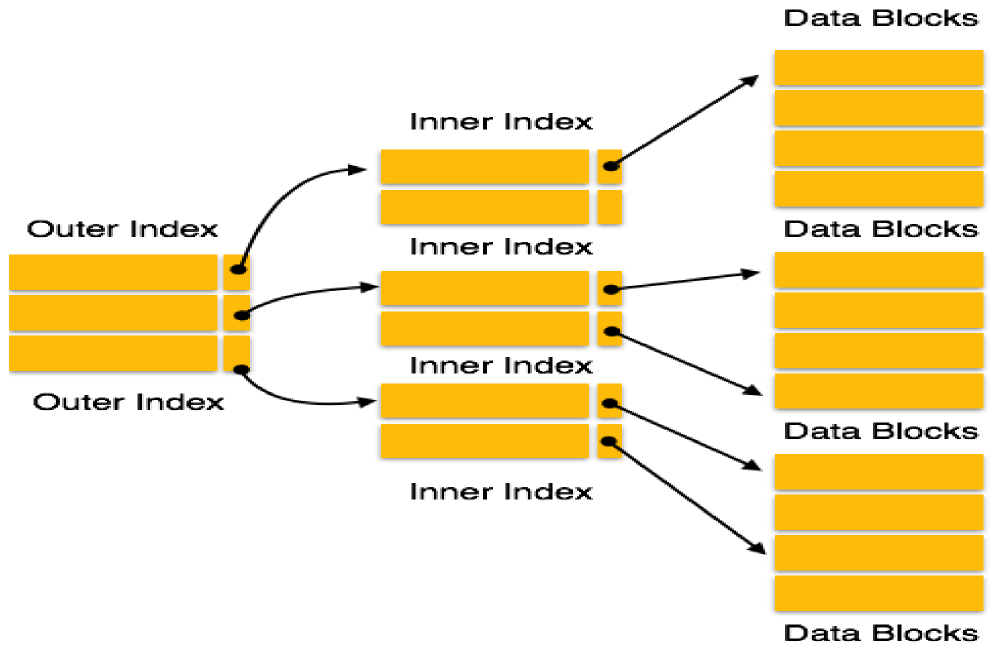
Roll	Pointer
100	
110	
120	
200	
210	
220	
300	
320	
310	

Secondary Level Index (Hard Disk)

Data bock in Memory	
100	
101	
- - -	- - - - - -
110	
111	
110	- - - - - -
120	
121	
- - -	
200	
201	
- - -	- - - - - -
210	
211	
- - -	- - - - - -
300	
- - -	- - - - - -

Multilevel Indexing

- Multilevel Indexing in a Database is created when a primary index does not fit in memory. In this type of indexing method, you can reduce the number of disk accesses to short any record kept on a disk as a sequential file, and create a sparse base on that file.



- As the size of the database increases, the size of the index also increases. But, storing a single- level index in main memory may not be practical due to its large size, which requires multiple disk accesses.

Index Evaluation Metrics

- Access types supported efficiently
    - Records with a specified value in the attribute.
    - or records with an attribute value falling in a specified range of values.

- Access time
- Insertion time
- Deletion time
- Space overhead

Basic Terminology

Records

- Represents a tuple in a relation.
- A file is a sequence of records.
- Records could be either fixed-length or variable-length.
- Records comprise of a sequence of fields (column, attribute)

Blocks

- Refer to physical units of storage in storage devices (Example: Sectors in hard disks, page in virtual memory)
- Usually (but not necessarily) stores records of a single file of fixed length, based on physical characteristics of the storage/computing device and operating system.
- Storage device is either defragmented or fragmented depending on whether contiguous sets of records lie in contiguous blocks

Basic Terminology

Blocking Factor

- The number of records that are stored in a block is called the "blocking factor".
- Blocking factor is constant across blocks if record length is fixed, or variable otherwise.
- If B is block size and R is record size, then the blocking factor is

$$bfr = (B/R)$$

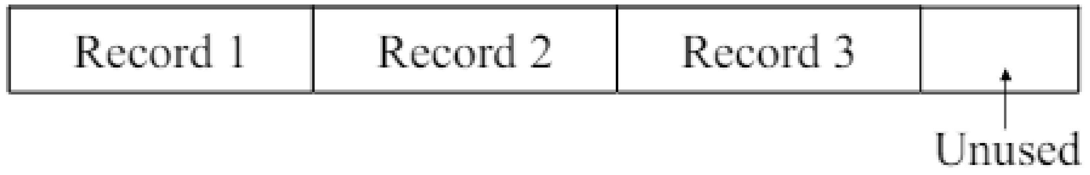
Since R may not exactly divide B, there could be some left-over space in each block equal to:

$B - (bfr * R)$ bytes.

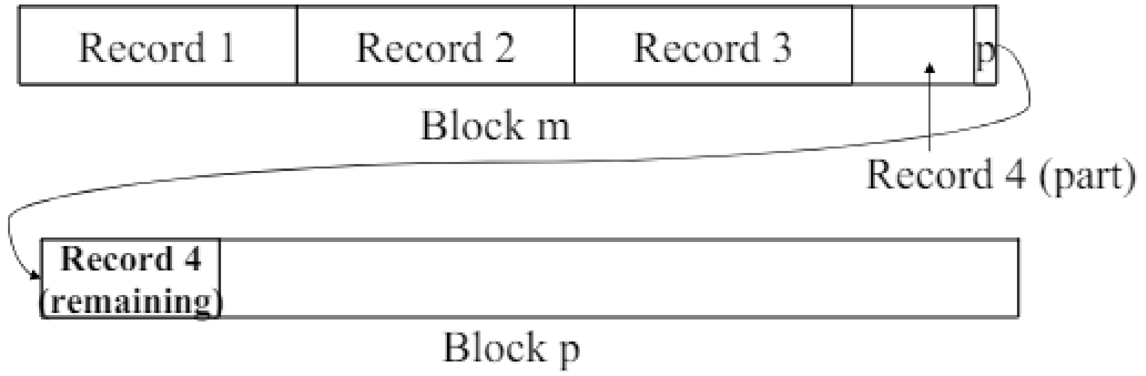
Basic Terminology

Spanned and Unspanned Records

- When extra space in blocks are left unused, the record organization is said to be "unspanned".



- In "spanned" record storage, records can be split so that the "span" across blocks.



When the record size is greater than the block size (i.e. R > B), the use of "spanned" record storage is compulsory.

Basic Terminology

Average Blocking Factor

- For variable-length records, with the use of record spanning, the "average" record length can be used in bfr computation to compute the number of blocks required for a file of records.

$$b = (r/bfr)$$

- where b is the number of blocks required to store a file having r records of variable length with the overall blocking factor bfr.

Primary Indexing- Example -1

Ordered file of r=50000 records

Block size B =1024 bytes

Records Fixed sized and unspanned with record length R=100 bytes

(a) Blocking Factor Bfr = B/R = 1024/100 =10 records per block

(b) The number of blocks needed for file is

$b = r/bfr = 50000/10 =$ 5000 blocks

(c) A binary search on the data file would need approx

$\log_2 b = \log_2 10000 =$ 13 block accesses

Primary Indexing- Example -2

Assume a file (b) consists of:

Total no of records (r): 30,000

Each record size (R): 100 bytes

Block size (B): 1024 bytes

How many records a block can hold (bfr)?

-> 1024/100 = 10 records/block

How many blocks will be required?

-> 30,000/10 = 3000 blocks

How many blocks should be accessed to perform binary search?

$\text{Log}_2(\text{Blocks}) = \text{Log}_2(3000) = 12$ blocks

Primary Indexing- Example -3

Assume: (With Primary Index)

Total no of records: 30,000 Each record size: 100 bytes

Block size: 1024 bytes Index size: 15 bytes (9 for field, 6 for pointer)

How many records a block can hold? -> 1024/100 = 10 records/block

How many blocks will be required? -> 30,000/10 = 3000 blocks

How many index can hold in a block? -> 1024/15 = 68 index/block

How many blocks required to store 3000 indexed? -> 3000/68 = 45 blocks

How many blocks should be accessed to perform binary search?

$\text{Log}_2(\text{Blocks}) = \text{Log}_2(45) = 6$ blocks

Additional disk access required to access from index to file : 1 block

There fore the final disk access using Index is : 6+1 = 7 blocks

Primary Indexing- Example -4

Assume: (With Primary Index)

Total no of records: 30,000 Each record size: 100 bytes

Block size: 1024 bytes Index size: 15 bytes (9 for field, 6 for pointer)

How many records a block can hold? -> 1024/100 = 10 records/block

How many blocks will be required? -> 30,000/10 = 3000 blocks

How many index can hold in a block? -> 1024/15 = 68 index/block

How many blocks required to store 3000 indexed? -> 3000/68 = 45 blocks

How many blocks should be accessed to perform binary search?

$\text{Log}_2(\text{Blocks}) = \text{Log}_2(45) = 6$ blocks

Additional disk access required to access from index to file : 1 block

There fore the final disk access using Index is : 6+1 = 7 blocks

Secondary Indexes Example -1

Assume: (With Secondary Index)

Total no of records: 30,000 Each record size: 100 bytes

Block size: 1024 bytes Index size: 15 bytes (9 for field, 6 for pointer)

How many records a block can hold? How many blocks will be required?

-> 1024/100 = 10 records/block

-> 30,000/10 = 3000 blocks

How many index can hold in a block? -> 1024/15 = 68 index/block

How many blocks required to store 3000 indexed? -> 30000/68 = 442 blocks How many blocks should be accessed to perform binary search?

$\text{Log}_2(\text{Blocks}) = \text{Log}_2(442) = 9$ blocks

Additional disk access required to access from index to file : 1 block

There fore the final disk access using Index is : 9+1 = 10 blocks

Example

- Suppose that we consider the same ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes.
- Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average of 300 records per zip code, assuming even distribution across zip codes.)
- The index in this case has 1,000 index entries of 11 bytes each (5-byte Zipcode and 6-byte block pointer) with a blocking factor

  $bfr_i = (B/Ri) = (4{,}096/11) = 372$ index entries per block.

- The number of index blocks is hence

  $b_i = (ri/bfri) = (1{,}000/372) = 3$ blocks.

To perform a binary search on the index file would need

$(\log_2 b_i)\rceil = \lceil(\log_2 3)\rceil = 2$ block accesses

Example: Given the following data file EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)

- Suppose that: record size R=150 bytes block size B=512 bytes r=30000 records
- Then, we get:
- blocking factor Bfr= B div R= 512 div 150= 3 records/block
- number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks

For an index on the SSN field, assume the field size V_{SSN}=9 bytes, assume the record pointer size P_R=7 bytes. Then:

index entry size R_I=(V_{SSN}+ P_R)=(9+7)=16 bytes

index blocking factor Bfr_I= B div R_I= 512 div 16= 32 entries/block

number of index blocks b= (r/ Bfr_I)= (30000/32)= 938 blocks

binary search needs $\log_2 bI$= $\log_2 938$= 10 block accesses

This is compared to an average linear search cost of:

(b/2)= 30000/2= 15000 block accesses

If the file records are ordered, the binary search cost would be:

$\log_2 b$= $\log_2 10000$= 14 block accesses

Advantages of Indexing

- Improved Query Performance
- Efficient Data Access
- Optimized Data Sorting
- Consistent Data Performance

Disadvantages of Indexing

- Increased database maintenance overhead.
- Choosing an index can be difficult.
- Indexing can reduce insert and update performance since the index data structure must be updated each time data is modified.