COS221 - L27 - Indexing

Linda Marshall

10 May 2023

Previously...

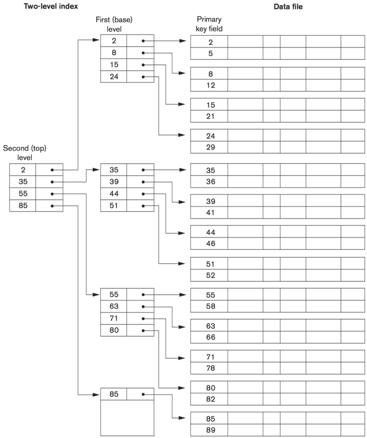
- Indexes provide secondary access paths to records on file, either on block level or record level
- Indexes are sparse or dense
- Types of indexes
  - Single-level ordered indexes - *primary* or *clustering* - on the file ordering field ,and *secondary* - not on the file ordering field
  - **Multilevel indexes**
  - **Indexes based multiple values that act as keys**

Multilevel Indexes

- Ordered indexes require a binary search on the index to be completed to find the block in which the field resides in the data file.
- To reduce this search time, multilevel indexes are used thereby reducing the search space much quicker. That is, instead of dividing the search space into two halves as with binary search, the search spaced is divided into *n* parts (fan-outs) for multilevel indexes.
- A multilevel index compromises of an index file (referred to as the first level). This is an ordered file with distinct values for each key and a pointer to the data file. A second level index is created with a key and a pointer to a block of the first level index. This can continue for multiple levels.
- Files using this ordering are called index sequential files. IBM's ISAM (Indexed Sequential Access Method) is a two-level index method related to disk cylinders and tracks.

Multilevel Indexes

Figure 18.6
A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.

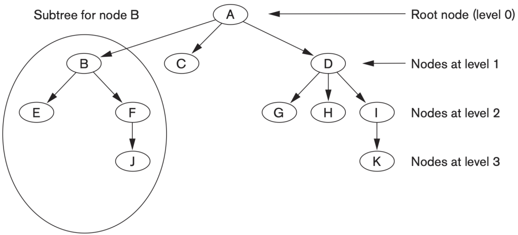


- This method reduces the number of blocks accessed to search for a record. It still however has insertion and deletion problems.

Dynamic Multilevel Indexes

- The dynamic multilevel index extends the advantages of the multilevel indexes, but addresses the insertion and deletion problems they have.
- Insertion and deletion results in an increase and decrease of blocks and therefore a data structure which manages this growing and shrinking is required. Enter search trees and specifically B-trees and B$^+$-trees.
- A quick overview of terminology:

Figure 17.7
A tree data structure that shows an unbalanced tree.

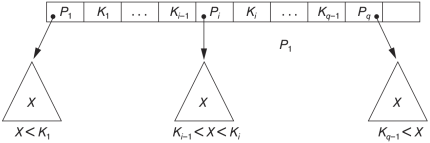


(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Dynamic Multilevel Indexes - Search Tree

- A search tree is used to guide the search for a record.
- A search tree of order p is a tree in which each node contains at most $p - 1$ search values and p pointers in the order $< P_1, K_1, P_2, K_2, ..., P_{(q-1)}, K_{(q-1)}, P_q >$ where $q <= p$. Each P_i is a pointer to a child node (or NULL pointer) and K_i the search key for some ordered set of values. The basic search tree structure is therefore:

Figure 17.8
A node in a search tree with pointers to subtrees below it.



- An example of a search tree with $p = 3$ is given by:

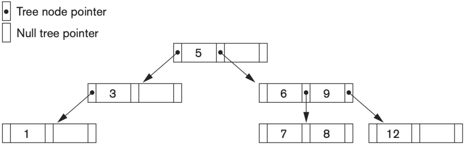


Figure 17.9
A search tree of order $p = 3$.

Dynamic Multilevel Indexes - Search Tree

- In general, these trees are not balanced. A balanced tree is a tree with all leaf nodes at the same level. Why balance a tree?
  - to guarantee an even distribution of nodes and minimise the depth of the tree
  - to make search speed uniform.
- Another, space related goal is to optimally use all nodes - that is keep them as full as possible. B-tree address these problems by placing additional constraints on the tree structure.

Dynamic Multilevel Indexes - B-Tree

- B-trees have constraints ensuring the tree is always balanced and that space wasted by deletion never becomes excessive.
- B-tree node structures comprise of tree pointers (P_i) and data pointers (when the search key field X equals K_i).
- Each node has at most p tree pointers and each node, except the leaf nodes, have at least $p/2$ tree pointers.
- Leaf nodes are at the same level and their tree pointers are NULL.

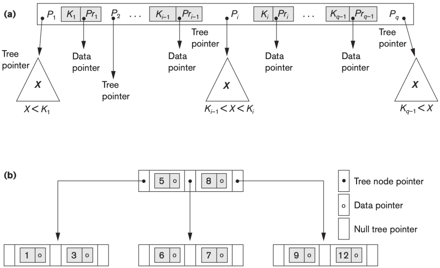


Figure 17.10
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

Dynamic Multilevel Indexes - B-Tree

- When a node becomes full, it is split, keeping the middle value in the current node and the rest of the values split between the two new nodes. When is it too empty (that is less than half full), nodes are merged.
- B-trees are sometimes used as primary file organisations. In this case, whole records are kept in the B-tree nodes if the number of records are small and the records are small in size.
- Most implementations of dynamic multilevel indexes use a variation of B-trees called the B^+-tree.

Dynamic Multilevel Indexes - B^+-Tree

- With B-trees, every search value appears once in the tree at the same level along with the data pointer.
- With B^+-trees, data pointers are stored only at the leaf nodes.
- The leaf nodes therefore have an entry for every value of the search field along with a pointer to the record or block containing the record. Thereby giving an extra level of indirection.
- Each leaf node points to the next leaf node of the B^+-tree.
- The structure of the internal and leaf nodes of a B^+-tree of order p is given. In both figures, $q <= p$.

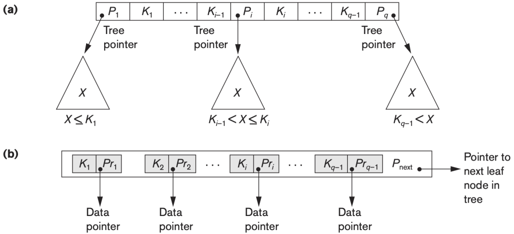


Figure 17.11
The nodes of a B^+-tree. (a) Internal node of a B^+-tree with $q - 1$ search values. (b) Leaf node of a B^+-tree with $q - 1$ search values and $q - 1$ data pointers.

Dynamic Multilevel Indexes - B$^+$-Tree

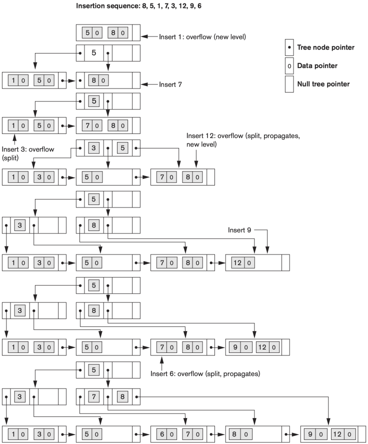
- With more blocks being packed into the internal nodes than in B-trees, searching is more efficient in the B$^+$-tree. A high level description of the search algorithm is given by:

```
N = block continuing the root node
Read block N
While (N is not a leaf node) do
  Q = number of tree pointers in node N
  If (the key being looked for is less than the first key in N) then
    N = block pointed to by P1
  Else if (the key being looked for is greater than the last key in N) then
    N = block pointed to by the last pointer Pq in the block
  Else
    Search block N for an entry i such that K(i-1) < K < Ki
    N = block pointed to by Pi
  End if
  Read block N
End while
Search block N where Ki equals the search key
If (found) then
  Read the file block pointed to by the data pointer Pr(i)
Else
  Record with search field K is not in the data file
End if
```

Dynamic Multilevel Indexes - B$^+$-Tree

- Inserting a record requires the node to be found, where the insertion is to take place and either inserting it into the node or splitting the node and then inserting. The algorithm is given in the book and illustrated as follows

Figure 17.12
An example of insertion in a B$^+$-tree with $p = 3$ and $p_{leaf} = 2$.



Insertion sequence: 8, 5, 1, 7, 3, 12, 9, 6
Insert 1: overflow (new level)
Tree node pointer
Data pointer
Null tree pointer
Insert 7
Insert 3: overflow (split)
Insert 12: overflow (split, propagates, new level)
Insert 9
Insert 6: overflow (split, propagates)

Dynamic Multilevel Indexes - B$^+$-Tree

- Deletion from a B$^+$-tree is given by the following example:

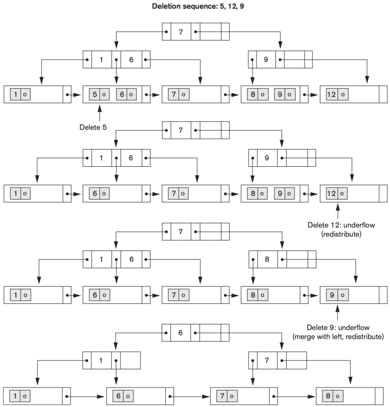

Deletion sequence: 5, 12, 9
Delete 5
Delete 12: underflow (redistribute)
Delete 9: underflow (merge with left, redistribute)

Figure 17.13
An example of deletion from a B$^+$-tree.

Indexes on Multiple Keys

- So far we have only considered indexes on a single field. In updates and queries, multiple attributes are usually involved. If a certain combination of attributes is frequently used, it is best to set up an index with this combination of attributes as key - called a composite key.
- The following are techniques that can be used to address this requirement.
  - Ordered index on Multiple Attributes
  - Partitioned Hashing
  - Grid Files

Indexes on Multiple Keys

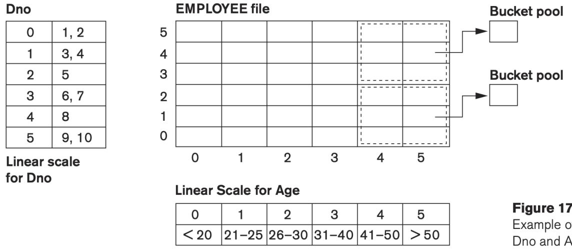
- All the techniques discussed thus far still apply.
- Instead of using a single value key, the key comprises of multiple values $< A_{k_1}, A_{k_2} ... A_{k_n} >$.
- By keeping lexicographical ordering within the key, for example $< 3, m >$ precedes $< 4, n >$ for any value of m or n, the techniques previously discussed can be applied on this more detailed tuple.

Indexes on Multiple Keys - Partitioned Hashing

- This is an extension of static external hashing discussed previously.
- In this case, access to multiple keys is allowed.
- Note, it is only suitable for equality comparisons $(=)$ and not range comparisons $(<, >, <=, >=)$.
- A key comprising of *n* components will result in *n* different hash addresses and the bucket address is a concatenation these hash addresses.
- Partitioned hashing can be extended to any number of attributes.

Indexes on Multiple Keys - Grid File

- Construct a grid with one linear scale for each search attribute.
- Choose the scales to achieve a uniform distribution of the attribute.
- Each cell in the grid points to some bucket address where the records corresponding to that cell are stored.
- Grid files perform well for multiple key access. They however are not efficient in terms of space.
- A frequent reorganisation of the files results in a maintenance cost to the grid file.



Dno

0	1, 2
1	3, 4
2	5
3	6, 7
4	8
5	9, 10

Linear scale for Dno

Linear Scale for Age

0	1	2	3	4	5
< 20	21–25	26–30	31–40	41–50	> 50

Figure 17.14
Example of a grid array on Dno and Age attributes.

Other Types of Indexes

- Hash indexes
- Bitmap indexes
- Function-based indexing

Other Types of Indexes

- Secondary structure using hashing on a search key.
- Stored in the hash is the key and either a pointer to the record in the file or the block in which the record is.

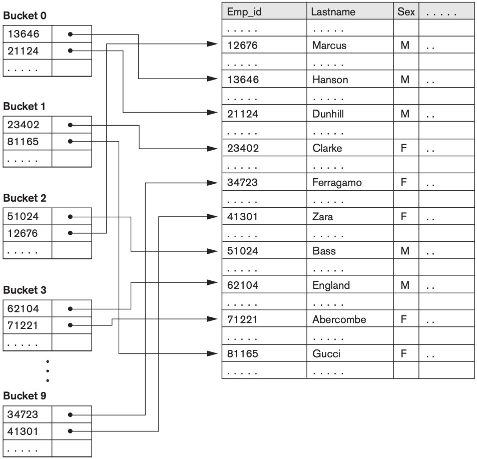


Figure 17.15
Hash-based indexing.

Bucket 0

13646	•
21124	•
. . . . .	

Bucket 1

23402	•
81165	•
. . . . .	

Bucket 2

51024	•
12676	•
. . . . .	

Bucket 3

62104	•
71221	•
. . . . .	

Bucket 9

34723	•
41301	•
. . . . .	

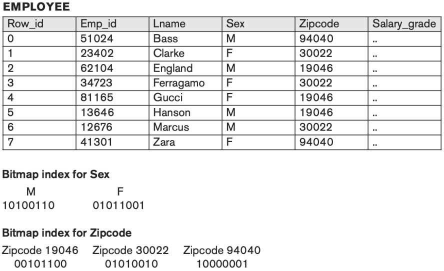
Emp_id	Lastname	Sex	. . . . .
. . . . .	. . . . .		
12676	Marcus	M	. .
. . . . .	. . . . .		
13646	Hanson	M	. .
. . . . .	. . . . .		
21124	Dunhill	M	. .
. . . . .	. . . . .		
23402	Clarke	F	. .
. . . . .	. . . . .		
34723	Ferragamo	F	. .
. . . . .	. . . . .		
41301	Zara	F	. .
. . . . .	. . . . .		
51024	Bass	M	. .
. . . . .	. . . . .		
62104	England	M	. .
. . . . .	. . . . .		
71221	Abercombe	F	. .
. . . . .	. . . . .		
81165	Gucci	F	. .
. . . . .	. . . . .		

Other Types of Indexes - Bitmap indexes

- Popular for querying multiple indexes.
- Used for relations with a large number of rows.
- An index is created for one or more columns and each value in the column is indexed.
- Building a bitmap requires the records to be numbered from 0 to n with a record/row id which is mapped to a row number or record offset within a block

Other Types of Indexes - Bitmap indexes

- A bitmap is therefore just a row of bits.
- The result indicates the rows in which the equality holds.



EMPLOYEE

Row_id	Emp_id	Lname	Sex	Zipcode	Salary_grade
0	51024	Bass	M	94040	..
1	23402	Clarke	F	30022	..
2	62104	England	M	19046	..
3	34723	Ferragamo	F	30022	..
4	81165	Gucci	F	19046	..
5	13646	Hanson	M	19046	..
6	12676	Marcus	M	30022	..
7	41301	Zara	F	94040	..

Bitmap index for Sex

M	F
10100110	01011001

Bitmap index for Zipcode

Zipcode 19046	Zipcode 30022	Zipcode 94040
00101100	01010010	10000001

Figure 17.16
Bitmap indexes for Sex and Zipcode.

- For queries using multiple attributes, the intersect of the bitmaps is calculated.

Other Types of Indexes - Bitmap indexes

- Deleting is interesting. An existence bitmap is kept which indicates with a 1 if the row exists and a 0 if it has been deleted.
- Rows are typically inserted at the end of the relation or placed where a row was deleted to limit bitmap updating.
- Bitmaps are sometimes used for B^+-tree leaf nodes.

Other Types of Indexes - Function-based indexing

- The result of applying a function to a field or collection of fields becomes the key to the index.
- It is often used on selective columns in the where clause. These may contain calculations as well.

  ```
  CREATE INDEX emp_index ON EMPLOYEE (UPPER(Lname));
  ```
- The following query is then function-based:

  ```
  SELECT Fname, Address, DeptNo
  FROM EMPLOYEE
  WHERE UPPER(Lname) = 'Tompson';
  ```
- Developed and used extensively in Oracle database systems.
- Function-index use either B-tree or Bitmap indexes.

Issues with Indexes

- Till now it has been assumed that index entries comprise of a key and a pointer to a block or record. This is referred to as a physical index.
- The disadvantage with this structure is that the pointer must be changed if the record is moved to another disk location.
- Enter a logical index where the index takes the form of the key and the field used for primary file organisation.
- When a secondary index is searched, the second index parameter takes one to the primary file organisation and the record is accessed from the primary index rather than the secondary.