Secondary Indexes, B-trees

CS157B

Chris Pollett

Feb 16, 2005.

Outline

- Using Oracle client
- Indexes with Duplicate Search Keys
- Indexes and Data Modification
- Secondary Indexes
- B-trees (a beginning)

Indexes with Duplicate Search Keys

- It is possible that more than one record has the same search key value in a sequential file. For example, if we had sorted on Salary, two people in an Employee table might have the same salary.
- One might modify a dense index to handle this by having one reference for each search key value and have it point to the first record with that value.
- To find a record we then look up the search key value in the index and do a linear search of the records with this value. (We still call this a dense index).
- If we have a sparse index, we don't have to change anything to handle duplicates.

Indexes and Data Modification

- When we change the data file, we also need to modify index.
- What will happen depends on whether the index is dense or space.

Action on file	Dense Index	Sparse Index
Create Blocks	none	insert if sequential
Delete Blocks	none	delete if sequential
Insert	insert also into index	update(?)
Delete	delete also from index	update(?)
Slide	update	update(?)

Avoiding Overflow Blocks, Preventing Row Migration

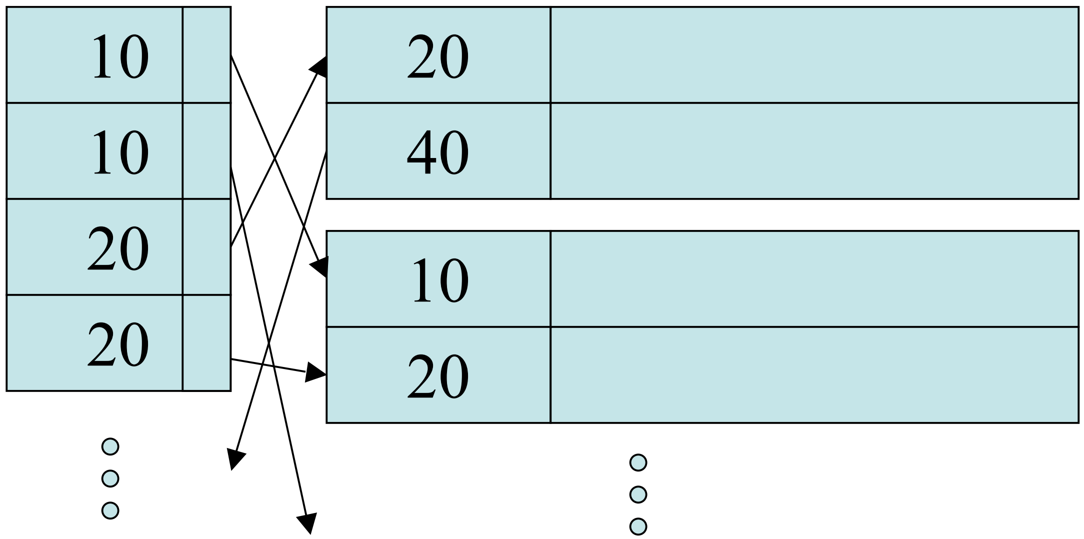
- Oracle has two parameters which are useful in avoiding overflow blocks and preventing row migration: PCTUSED and PCTFREE.
- PCTFREE - percentage of a block that is reserved for future updates to existing rows.
- PCTUSED - percentage full a block removed from the free blocks list must fall below to be added back to the free blocks list.

Secondary Indexes

- Up till now we have been considering primary indexes. These are indexes which are on the search key of a sequential file.
- Secondary indexes are indexes on fields which are not sorted in the file.
- Secondary indexes are always dense and will often have duplicates.
- One can create indexes with SQL like:

  CREATE INDEX MyIndex ON MyTable(MyField);

Design of Secondary Indexes



- Notice that although the indexed field is not sorted in the file. It is sorted in index.
- Indexes are still usually much smaller than the original file size.

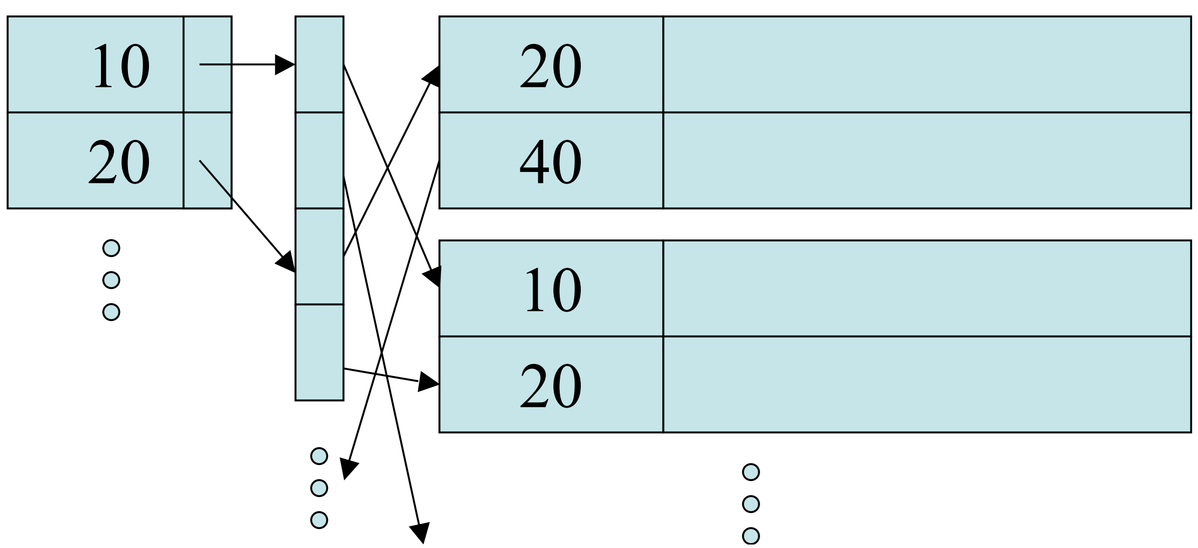
Applications of Secondary Indexes

In addition to allowing indexes on non-sorted fields of a table, secondary indexes have other uses:

- They might provide an index on a primary key if the file was not sequential but rather a *heap* (i.e., unsorted).
- They allow indexes to be had on a horizontally partitioned or *clustered* file.

Indirection in Secondary Indexes

- The scheme for secondary indexes previously presented wastes space because we often have to repeat the same key value multiple times.
- We can use indirection to minimize this waste:



Document Retrieval and Inverted Indexes

- An *inverted index* on a document is an index from the words in the document to the locations within the document where the word occurs.
- For web search probably store the inverted indexes of several documents into one big index.
- Such inverted indexes are usually secondary indexes and employ the kind of indirection mentioned on the last slide.
- Often the index also has *stem words* for word in the document. Ex: storing *dog* for *dogs*.
- Some *stop words* like ‘a’ or ‘the’ would not be included.

B-trees

- As we have already seen multi-level indexes can speed up the retrieval of data.
- B-Trees are a family of data structures that allow us to maintain such kind of indexes by supporting inserts, deletes, and updates.
  - They automatically maintain the correct number of levels for the file size in use.
  - B-tree index blocks are used so that they are always between half and completely filled.

The Structure of B-trees

- A B-tree is a balanced tree whose nodes are blocks. That is, every path from the root to a leaf is of the same length.
- A B-tree is characterized by a parameter n which depends on the block size.
- Within a node a B-tree stores n search key values and n+1 pointers. (pointers for internal nodes always point to one level lower in the tree)
- Key values are nondecreasing in going from left to right with tree at the same level
- We assume that the root will always use at least two of these pointers.
- At a leaf, the last pointer points to the block to right. At least half of the pointers and search key values in a leaf are used and the pointers point to actual records.
- At least half of the pointers in and key values are used in an internal node.