Record Modifications & Indexes

CS157B

Chris Pollett

Feb 14, 2005.

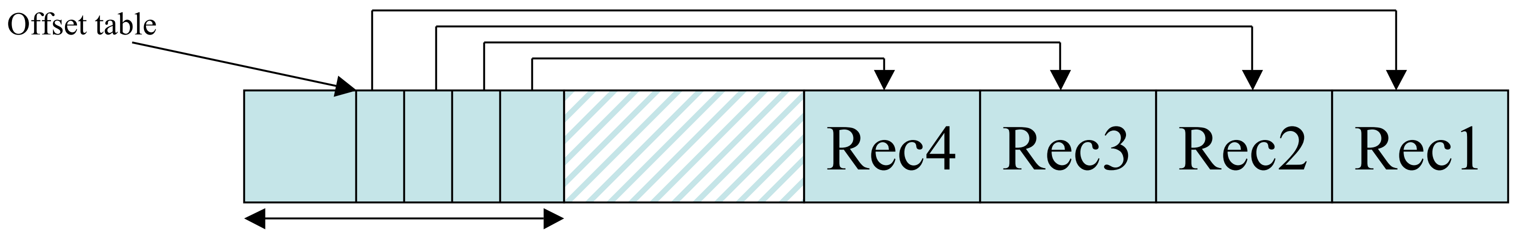
Outline

- Hand-out Oracle Accounts
- Record Insertion/Deletion/Update
- Sequential Files
- Dense Indexes
- Sparse Indexes
- Multi-level Indexes

Record Insertion

- If records not sorted can just find a block of table in question with space for a record and add our record to it. If no such block exists allocate a new block and associate with our table.
- The situation is harder if need to keep records sorted.

Record Insertion Sorted case



- Find block where would insert. If there is space add the record there.
- We might need to slide records around within the relevant block and update its offset table.
- If there is no space, then we could: (a) look at an adjacent block in the sorted order and try to find space there and redistribute record,s (b) create an overflow block.

Record Deletion

- When we delete a record we may be able to reclaim its unused space.
- We could try to slide records around so there is one big available space block.
- If this is not possible then need an available space list.
- We might also be able to get rid of any overflow blocks.
- We may need to place a tombstone on the record so that anything pointing to the record knows it was deleted.

Record Update

- For fixed length records can just overwrite the existing record.
- For variable length records one has all the problems associated with insert and delete except that one doesn’t have to have a tombstone.

Speeding Queries and Indexes

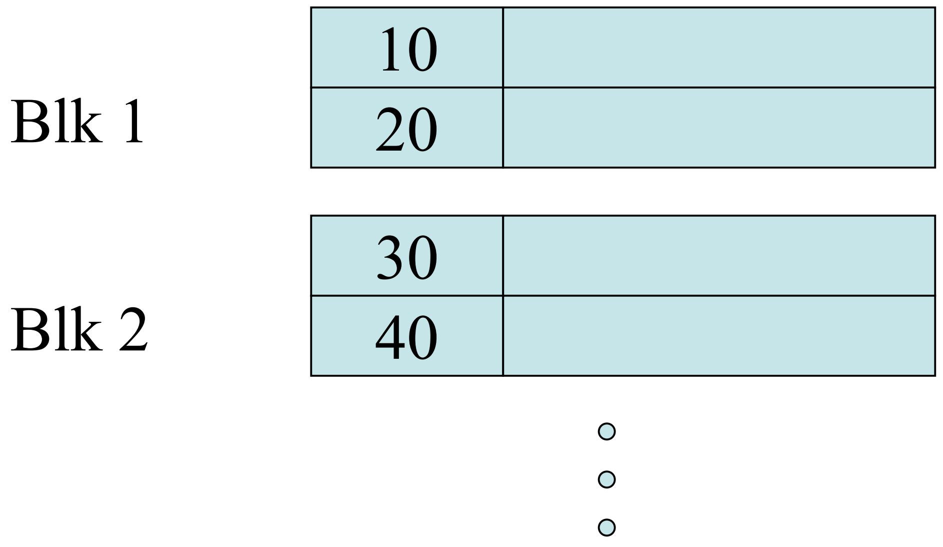
- When we store records we need to store enough information so we can process queries like SELECT * FROM R.
- Need to store in the block header where in a block the records begin, and in the record header of these records, which relation they belong to.
- Might want to reserve whole cylinders for particular relations to speed things up.
- To speed up queries like SELECT * FROM EMP WHERE name = ‘John Smith’; need to use indexes.

Types of Indexes

- Simple indexes on sorted file
- Secondary indexes on unsorted files
- B-trees
- Hash tables

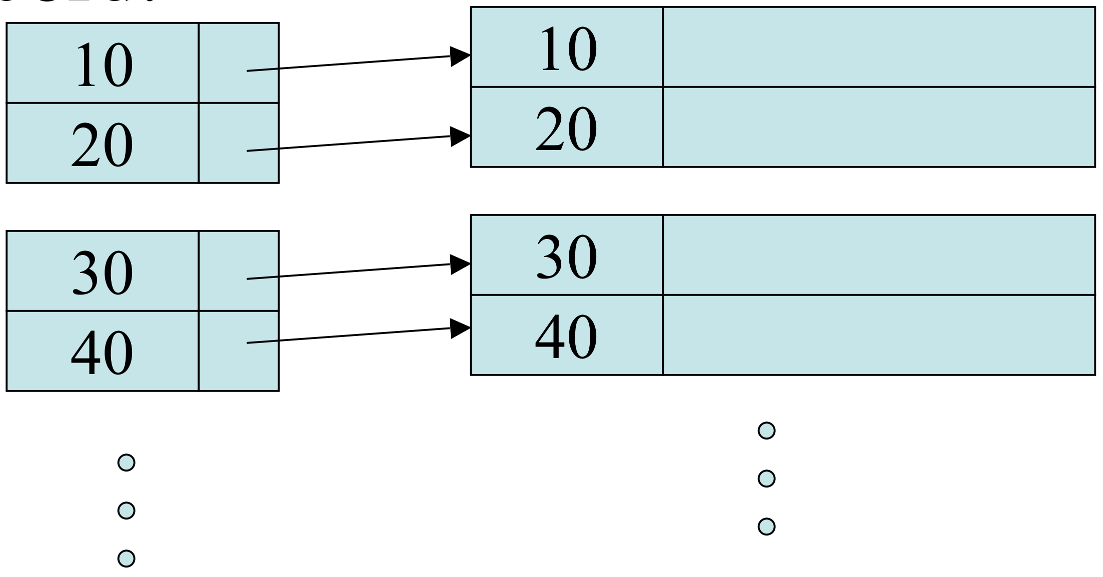
Sequential Files

- In this kind of file we sort the file on the attributes of the index.



Dense Indexes

- Now that the records are sorted we can build on top of them a dense index. That is, an index where we have one index entry per record.



More on Dense Indexes

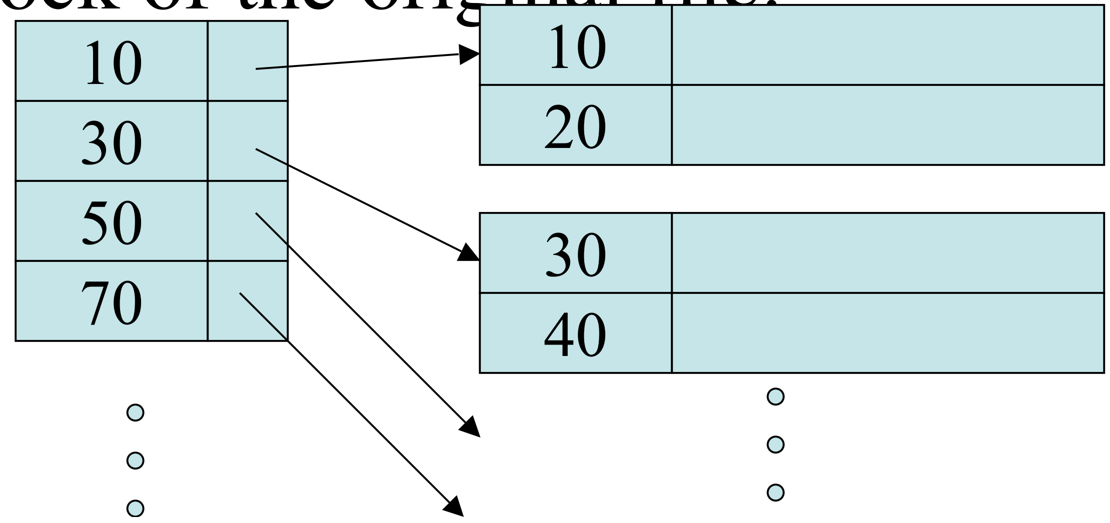
- Dense Indexes can be used for queries where we are searching by a particular value.
- Given K we look up K in the index using binary search (O(log n) time where n is number of index entries) then follow the pointer there to the record.
- Since the index only has (key, ptr) pairs rather than (key, lots of data), it will probably be much smaller than original file and so possibly fit in memory. This can be used to save further I/Os

Dense Index Example

- Suppose R had 1,000,000 tuples and 10 fit into a block of 4096 bytes. So R takes 100,000 blocks / 400 megabytes to store. Too much to be easily held in main memory on most systems.
- If the key is 30 bytes long and the record pointer 8 bytes then around 100 index entries can fit into a block. So the index takes roughly 40 megabytes (10,000 blocks) to store.
- This probably fits into main memory. If not, have only log (10000), roughly 13 or 14, accesses to look an entry.

Sparse Indexes

- These take less space but are a little slower than dense indexes.
- Idea is we no longer have an index entry for every record, instead we have one entry per block of the original file:



Sparse Index Example

- If R was 100, 000 blocks, each storing 10 records as before, and again could store 100 index entries in a block, then would only need 1000 blocks (4MB) to store this index.
- On the other hand, with a dense index could answer questions of the form: “ Is K in the file?” using only the index, without having to look up the block containing K in the file itself.

Multi-Level Indexes

- Since a sparse index itself can still take several blocks, one could imagine using a sparse index on a sparse index to look up the data.
- In our previous example, the 1000 block index could have a 10 block sparse on it to help look up the correct block.
- Multi-level indexes are fast, but can be a pain to update.