Indexing

Last Time : Binary Searching, Keysorting, Introduction to Indexing.

Today : Indexing (Part II)

Reference : Folk, Zoellick and Riccardi - Chapters 7.4 - 7.6

Indexing

Operations to Maintain an Indexed File

1. Create the original empty index and data files.
2. Load the index file into memory before using it.
3. Rewrite the index file from memory after using it.
4. Add data records to the data file.
5. Delete records from the data file.
6. Update records in the data file.
7. Update the index to reflect changes in the data file.

We will take a closer look at operations 3-7.

Rewrite the Index File from Memory

When the data file is closed, the index in memory needs to be written to the index file.

An important issue to consider is what happens if the rewriting does not take place (power failures, turning the machine off, etc.)

Two important safeguards:

- Keep an status flag stored in the header of the index file. The status flag is “on” whenever the index file is not up-to-date. When changes are performed in the index in main memory the status flag in the file is turned on. Whenever the file is rewritten from main memory the status flag is turned off.
- If the program detects the index is out-of-date it calls a procedure that reconstruct the index from the data file.

Record Addition

This consists of appending the data file and inserting a new record in the index. The rearrangement of the index consists of “sliding down” the records with keys larger than the inserted key and then placing the new record in the opened space.

Note: This rearrangement is done in main memory.

Record Deletion

This should use the techniques for reclaiming space in files (Chapter 6.2) when deleting from the data file. We must delete the corresponding entry from the index:

- Shift all records with keys larger than the key of the deleted record to the previous position (in main memory); or
- Mark the index entry as deleted.

Record Updating

There are two cases to consider:

- The update changes the value of the key field:
  Treat this as a deletion followed by an insertion
- The update does not affect the key field:
  If record size is unchanged, just modify the data record. If record size changes treat this as a delete/insert sequence.

Indexes too Large to Fit into Main Memory

The indexes that we have considered before could fit into main memory. If this is not the case, we have the following problems:

- Binary searching of the index file is done on disk, involving several "seeks".
- Index rearrangement (record addition or deletion) requires shifting on disk.

Two main alternatives:

- Hashed organization (Chapter 11) (When speed is a top priority)
- Tree-structured (multilevel) index such as B-trees and B+ trees (Chapter 9,10) (It allows keyed and ordered sequential access).

But a simple index is still useful, even in secondary storage:

- It allows binary search to obtain a keyed access to a record in a variable-length record file.
- Sorting and maintaining an index is less costly than sorting and maintaining the data file, since the index is smaller.
- We can rearrange keys, without moving the data records when there are pinned records.

Indexing to Provide Access by Multiple Keys

In our recording file example, we built an index for LABEL ID key. This is the **primary key**.

There may be **secondary keys**: title, composer and artist.

We can build **secondary key indexes**.

Composer index:

Secondary key	Primary key
Beethoven	ANG3795
Beethoven	DG139201
Beethoven	DG18807
Beethoven	RCA2626
Corea	WAR23699
Dvorak	COL31809
Prokofiev	LON2312

Note that in the above index the secondary key reference is to the primary key rather than to the byte offset.

This means that the primary key index must be searched to find the byte offset, but there are many advantages in postponing the binding of a secondary key to an specific address.

Record Addition

When adding a record, an entry must also be added to the secondary key index.
Store the field in **Canonical Form** (say capital letters, with pre-specified maximum length).
There may be duplicates in secondary keys. Keep duplicates in sorted order of primary key.

Record Deletion

Deleting a record implies removing all the references to the record in the primary index and in all the secondary indexes. This is too much rearrangement, specially if indexes cannot fit into main memory.
Alternative:

- Delete the record from the data file and the primary index file reference to it. Do not modify the secondary index files.
- When accessing the file through a secondary key, the primary index file will be checked and a deleted record can be identified.

This results in a lot of saving when there are many secondary keys.

The deleted record still occupy space in the secondary key indexes. If a lot of deletions occur, we can periodically cleanup these deleted records from the secondary key indexes.

Record Updating

There are three types of updates :

- Update changes the secondary key :
  We have to rearrange the secondary key index to stay in sorted order.

- Update changes the primary key :
  Update and reorder the primary key index; update the references to primary key index in the secondary key indexes (it may involve some re-ordering of secondary indexes if secondary key occurs repeated in the file).

- Update confined to other fields :
  This won't affect secondary key indexes. The primary key index may be affected if location of record changes in data file.