9 INDEXING

Indexing

- Operations in order to maintain an Indexed File
  1. Create the original empty 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 the index to reflect changes in the data file

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 residing on main memory the status flag in the file is turned on.
  - Whenever the file is written from main memory the status flag is turned off.
- If the program detects the is index is out-of-date it calls a procedure that reconstruct the index from the data file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 that this rearrangement is done in main memory

Record Deletion

- This should use the techniques for reclaiming space in files 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 “fseek()” calls
  - Index rearrangement (record addition or deletion) requires shifting on disk

Two Main Alternatives

- Tree-structured index such as B-trees and B+ trees (Chapters 11-12)
- Hashed Organization (Chapters 13,14)

A Simple Index is still Useful, even in SSD

- 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

- Suppose that you are looking at a collection of recordings with the following information about each of them:
  - Identification Number
  - Title
  - Composer or Composers
  - Artist or Artists
  - Label (publisher)

Data File

Address of Record	Actual data record
32	LON|2312|Romeo and Juliet|Prokofiev . . .
77	RCA|2626|Quarter in C Sharp Minor . . .
132	WAR|23699|Touchstone|Corea . . .
167	ANG|3795|Sympony No. 9|Beethoven . . .
211	COL|38358|Nebeaska|Springsteen . . .
256	DG|18807|Symphony No. 9|Beethoven . . .
300	MER|75016|Coq d'or Suite|Rimsky . . .
353	COL|31809|Symphony No. 9|Dvorak . . .
396	DG|139201|Violin Concerto|Beethoven . . .
442	FF|245|Good News|Sweet Honey In The . . .

Indexing to Provide Access by Multiple Keys

- So far, our index only allows key access. i.e., you can retrieve record DG188807, but you cannot retrieve a recording of Beethoven's Symphony no. 9.
- We need to use secondary key fields consisting of album titles, composers, and artists.
- Although it would be possible to relate a secondary key to an actual byte offset, this is usually not done.
- Instead, we relate the secondary key to a primary key which then will point to the actual byte offset.

Example: Composer Index

- Composer Index

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

Record Addition

- When adding a record, an entry must also be added to the secondary key index.
- Store the field in Canonical Form
- 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

An Alternative to Record Deletion

- 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 also from the secondary key indexes

Record Updating

- There are three types of updates

1. The update changes the secondary key
   - We have to rearrange the secondary key index to stay in sorted order.
2. The 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)

Record Updating (Con't)

3. Update confined to other fields
   - This will not affect secondary key indexes.
   - The primary key index may be affected if the location of record changes in the data file.

Retrieving Records using Combinations of Secondary Keys

- Secondary key indexes are useful in allowing the following kinds of queries:
  - Find all records with composer "BEETHOVEN"
  - Find all records with the title "Violin Concerto"
  - Find all records with composer "BEETHOVEN" and title "Symphony No.9"

Solution

- Use the matched list and primary key index to retrieve the two records from the file.

Matches from composer index	Matches from title index	Matched list (logical "and")
ANG3795 DG139201 DG18807 RCA2626	ANG3795 COL31809 DG18807	ANG3795 DG18807

Improving the Secondary Index Structure: **Inverted Lists**

- Two difficulties found in the proposed secondary index structures:
  - We have to rearrange the secondary index file even if the new record to be added in for an existing secondary key
  - If there are duplicates of secondary keys then the key field is repeated for each entry, wasting space

Array of References

- No need to rearrange
- Limited reference array
- Internal fragmentation

Revised composer index

Secondary key	Set of primary key references			
BEETHOVEN	ANG3795	DG139201	DG18807	RCA2626
COREA	WAR23699			
DVORAK	COL31809			
PROKOFIEV	LON2312			
RIMSKY-KORSAKOV	MER75016			
SPRINGSTEEN	COL38358			
SWEET HONEY IN THE R	FF245			

Inverted Lists

- Organize the secondary key index as an index containing one entry for each key and a pointer to a linked list of references.

Secondary Key Index File

0	Beethoven	3
1	Corea	2
2	Dvorak	5
3	Prokofiev	7

LABEL ID List File

0	LON2312	-1
1	RCA2626	-1
2	WAR23699	-1
3	ANG3795	6
4	DG18807	1
5	COL31809	-1
6	DG139201	4
7	ANG36193	0

- Beethoven is a secondary key that appears in records identified by the LABEL IDs: ANG3795, DG139201, DG18807 and RCA2626

Advantages

- Rearrangement of the secondary key index file is only done when a new composer's name is added or an existing composer's name is changed. Deleting or adding records for a composer only affects the LABEL ID List File. Deleting all records by a composer can be done by placing a "-1" in the reference field in the secondary index file.
- Rearrangement of the secondary index file is quicker since it is smaller
- Smaller need for rearrangement causes a smaller penalty associated with keeping the secondary index file in disk

Advantages (Con't)

- The LABEL ID List File never needs to be sorted since it is entry sequenced.
- We can easily reuse space from deleted records from the LABEL ID List File since its records have fixed-length.

Disadvantages

- Lost of "locality": labels of recordings with same secondary key are not contiguous in the LABEL ID List File (seeking).
- To improve this, keep the LABEL ID List File in main memory

Selective Indexes

- Selective Index: Index on a subset of records
- Selective index contains only some part of entire index
  - provide a selective view
  - useful when contents of a file fall into several categories
    - e.g. *20 < Age < 30 and $1000 < Salary*

Binding

- In our example of indexes, when does the binding of the index to the physical location of the record happens?
  - For the primary index, binding is at the time the file is constructed.
  - For the secondary index, it is at the time the secondary index is used.

Advantages of Postponing Binding

- We need small amount of reorganization when records are added or deleted.
- It is safer approach: important changes are done in one place rather than in many places.

Disadvantages

- It results in slower access times (Binary search in secondary index + Binary search in primary index)

When to use Tight Binding/Bind-at-retrieval

- Use Tight Binding
  - When data file is nearly static (little or no adding, deleting or updating of records)
  - When rapid retrieval performance is essential.
    - Example: Data stored in CD-ROM should use tight binding
- Use Bind-at-retrieval
  - When record additions, deletions and updates occur more often