File Storage and Indexing

Chapters 4 and 5

DBs Reside on Disks

- Main reasons: (main) memory is too small and too volatile
- Understanding disk operation and performance is important
- Data structures and algorithms for disk are very different from those appropriate for memory
- The single most important fact about disks, relative to memory, is… (we shall see)

Review: Disk Organization

- Spinning magnetic surfaces, stacked
- R/W head per surface
- Track: circular path
  - Cylinder: vertical stack of tracks
- Sectors: fixed length physical blocks along track
  - separated by gaps
  - overhead info (ID, error-correction, etc.)
  - each sector has a hardware address

Disks and OS

- OS manages disks
- Files are allocated in blocks (groups of sectors)
  - units may be scattered on disk
  - OS makes file look contiguous
  - no simple map between logical records (program/file view) and physical placement

Program Viewpoint

- Files are collections of logical "records"
  - records are commonly (but not always) fixed in size and format
  - fields in records are commonly (but not always) fixed in size and format
- Sequential access: program gets data one record at a time
- Random access: program can ask for a record by its relative block # (not disk address!)

Common Modes of Record Processing

- Single record (random)
  …WHERE SSN=345667899
- Multiple records (no special order)
  *SELECT * FROM DEPT...*
- Multiple records (in order)
  …ORDER BY SSN

An efficient storage scheme should support all of these modes

Naïve Relational View

- Each table is a separate file
  - All rows in a table are of fixed size and format
  - Rows are stored in order by primary key

What's the Big Deal?

(actual numbers will vary)

- CPU speed: 100-500MHz
- Memory access speed: (100ns)
  how many CPU cycles is this?
- Disk speed has three components
  - **Seek tim**e: to position the W/R head (10ms)
    how many memory cycles is this?
  - **Latency**: rotational speed (3600rpm)
  - **Transfer time**: movement of data to/from memory (5Mb/sec)

Fact of Life

Seek time outweighs all other time factors

Implication: Martin's three rules for efficient file access:

- 1. Avoid seeks
- 2. Avoid seeks
- 3. Avoid seeks

- Corollary: *Any technique that takes more than one seek to locate a record is unsatisfactory*.

Memory vs Disk Data Structures

- Example: Binary Search of a sorted array
  - Needs $O(Log_2N)$ operations
- How many passes if 1,000,000 integers in memory?
  - The answer (20) is acceptable
- ***How many seeks if 1,000,000 records on disk?***
  - The answer (20) in unacceptable
  - And how did they get sorted (NlogN at best)?

Avoiding Seeks

- Overlapping disk operations
  - double buffering
- Disk cache
  - locality principle: recently used pages are likely to be needed again
  - maintained by OS or DBMS
- Sequential access
  - Seek needed only for first record
- Smart file organizations

A Smart File Organization...

- One which needs very few seeks to locate any record
- **Hashing**: go directly to the desired record, based on a simple calculation
- **Indexing**: go directly to the desired record, based on a simple look-up

Review: Hashing

- Main idea: address of record is *calculated* from some value in the record
  - usually based on primary key
- Zillions of possible hash functions
  - Hard to find a good hash function

Pitfalls of Hashing

- Conflicts: more than one record hashing to the same address.
  - schemes to overcome conflicts: overflow areas; chained records, etc.
  - all involve more disk I/O
- Wasted space
- No efficient access for other than primary key
- No efficient way for sequential, especially sorted traversal

Index

- Main idea: ***A separate data structure used to locate records***
- Frequently, index is a list of key/address pairs
- If index is small, a copy can be maintained in memory!
  - Permanent disk copy is still needed
- Many, many flavors of index organization

Some Indexing Terminology

- Index field (key)
- Primary index
- Secondary index
- Dense index
- Non-dense (sparse) index
- Clustering index

Indexing Pitfalls

- Each index takes space
- Index file itself must permit efficient reorganization
- Large indices won't fit in memory
  - May require multiple seeks to locate record entry
- Solution to some problems: ***Multilevel indexing***
  - each level is an index to the next level down

Requirements on Multilevel Indexes

- Must have low height
- Must be efficiently updatable
- Must be storage-efficient
- Top level(s) should fit in memory
- Should support efficient sequential access, if possible

B-Tree

- B-Tree is a type of multilevel index
  - from another standpoint: it's a type of balanced tree
- Invented in 1972 by Boeing engineers R. Bayer and E. McCreight
- By 1979: "the standard organization for indexes in a database system" (Comer)

B-Tree Overview

- Assume for now that keys are fixed-length and unique

A B-tree can be thought of as a generalized binary search tree

  - multiple branches rather than just L or R
- Some wasted space in the nodes is tolerated
- Trees are always perfectly balanced

B-Tree Concepts

- Each node contains
  - tree (node) pointers, and
  - key values (and record pointers)
- Order p means (up to) p tree pointers, (up to) p-1 keys
- Given a key K and the two node pointers L and R around it
  - All key values pointed to by L are < K
  - All key values pointed to by R are > K

B-Tree Growth and Change

When a node is full, it splits.

  - middle value is propagated upward
  - two new nodes are at same level as original node
- Height of tree increases only when the root splits
- Recommended: split only on the way down
- On deletion: two adjacent nodes recombine if both are < half full

B+ Tree

- Like B-Tree, except
  - record pointers are only in the leaves
    - left-side pointer has keys <= rather than <
  - leaf nodes are linked together left to right
- Allows sequential access of entire file!
  - "indexed sequential"
- Interior nodes have higher order than leaf nodes
  - Fewer non-leaf levels than B-Tree

Variations

- Could store the whole record in the index block
  - especially if records are few and small
  - in a B+ tree, this would make sequential access especially efficient
- Could redistribute records between adjacent blocks
  - esp. on deletion (B* tree)
- Variable order: accommodate varying key lengths

Other Forms of Indexing

- *Bitmap indexes*
  - One index per value (property) of interest
  - One bit per record
  - TRUE if record has a particular property
- *Indexed hash*: hash function takes you to an entry in an index
  - allows physical record locations to change
- Clever indexing schemes are useful in optimizing complex queries