Chapter 5: Record Storage and Primary File Organizations

Magnetic disks -

- bit - by magnetizing an area on disk in certain ways, we can make it represent a bit value of either 0 or 1
- byte - bits are grouped into either 4 or 8 to make a byte - one character per byte
- single-sided - disks that store info on only one of its magnetic surfaces
- double-sided - both surfaces are used
- multiple disks stacked ontop of each other form a disk pack
- info stored on a disk in concentric circles of small width - each circle called a track
- cylinder - tracks of the same diameter on different disks in a disk pack
- sector - section of disk divided by an angle from the center - hard-coded on disk
- blocks - division of a track made by operating system during formatting
  - separated by fixed-size interblock gap - include control info identifying next block
- transfer of info from main mem to disk is done in blocks
- hardware address - combination of surface number, track number and block number
- buffer - contiguous reserved area in main mem that holds one block
- read- block from disk copied to buffer
- write - contents of buffer copied to disk block
- cluster - group of blocks - sometimes data moved in clusters - buffer size matches cluster size
- disk drive - system that includes
  - read / write head - one for each surface - actuator moves r / w head over a cylinder of tracks
  - motor to rotate disks
- transfer a disk block:
  - position read / write head on correct track - seek time
  - rotate disk to the desired block - rotational delay
  - transfer data to / from main memory - block transfer time
  - seek time and rotational delay usually much longer than transfer time
  - locating data on disk is a major bottleneck in DB applications
  - ch 5 discusses file structures to minimize number of block transfers needed to locate and transfer required data

Magnetic tape

- sequential access
- tape drive - reads / writes data from / to tape reel
- blocks are larger than on disk - so are interblock gaps
- used to back up DB

Buffering blocks

- can reserve several buffers in main mem to speed up transfer
- while one buffer is being written, CPU can process data in other buffer
- double buffering - can be used to write a continuous stream of blocks from mem to disk
  - permits continuous reading / writing of data on consecutive disk blocks - eliminating seek time and rotational delay fo all but the first block

Placing file records on disk

- data stored in *record*s- collection of related data values - each value formed by one or more bytes - corresponds to *field* of a record
  - EX: one TOY record consists of the fields TOY#, name, manufacturer, etc...
- *record type* - collection of field names and their data types

data type - (of a field) specifies type of values the field can take
number of bytes for each data type is fixed for a given computer system
BLOB - binary large object - large unstructured objects that represent images, digitized video or audio or free text
- usually record includes a pointer to BLOB - somewhere else on disk
file - sequence of records
usually - all records in a file are of the same record type
fixed-length records - all records same size
variable-length records - different sizes
- variable-length fields in record
EX: a text field for customer comments
- repeating field - more than one value for a field in an individual record
EX: ages of children in customer relation
- optional fields
- mixed file - different record types
separator character used to separate variable-length records in a file

records are mapped to disk blocks - the unit of transfer from disk to MM buffers

blocking factor for a file - the number of records per block on disk
let B=size of block (bytes); R=size of record
B >= R ==> bfr = floor(B/R) (rounds down to next integer)
unused space = B - (bfr * R) bytes
unused space can be used by storing part of a record on one block and the rest on another
- a pointer at the end of one block to the other block storing the rest of the record
- called *spanned records*
- must be used if R >= B
bfr for variable-length records represents the average number of records per block

file allocation:
contiguous allocation - allocated to consecutive blocks
- reading whole file is easy - but expanding file is difficult
linked allocation - file block contains pointer to next file block
- slow reading - but easily expanded
combination - allocates clusters of consecutive disk blocks and clusters are linked together
indexed allocation - one or more index blocks contain pointers to actual file blocks

file header (descriptor) - contains info about file for programs to access file records
disk addresses of file blocks
record format descriptions - field lengths, field types, field order

Operations on files:
retrieval or update
select one or more records based on a *selection condition*
simple selection condition - equality comparison on some field value
complex conditions - involve other comparison operators - >, < ,...
in general - arbitrary Boolean condition
DBMS decomposes complex conditions to extract simple conditions that can be used to locate records on disk

high level programs (DBMS software) access records using commands like the following (all record-at-a-time operations)

Find - locates first (physically in file) record satisfying a search condition
- transfers block to buffer
- becomes current record

Read - copies current record from buffer to program variable user work area
- may also advance current record pointer

Find / Next - locates next record in file that satisfies search condition
- transfers block to buffer
-becomes current record

Delete - deletes current record
- updates file on disk to reflect deletion

Modify - modifies some field values of current record
- updates file on disk

Insert- inserts new record into file
- locates block where record is to be inserted
- transfers block into main memory
- writes record into buffer
- writes buffer to disk

Open - prepare file for access - retrieve file header

Close - indicates that we are through using file

some set-at-a-time operations (allowed in some file systems):

FindAll - locates all records in file that satisfy search condition

FindOrdered - retrieves all records in file in specified order

Reorganize - starts reorganization process

file organization - refers to the way records and blocks are placed on storage medium

access method - consists of a group of programs that allow operationsto be applied to a file
- it is possible to apply several different access methods to a single file organization

file organization should be chosen to reflect the types of searches that are expected to be used most often
- ex: if search is usually on a particular field - file should be organized so that search on that field is efficient
- may involve physically ordering records based on that field value

File organizations:

unordered records (heap files) - placed in file in the order they were inserted
- inserting new record very efficient
- any search requires a linear search - block by block
- deleting a record leaves unused space where the block was
- can use a deletion marker to mark deleted records so they can be skipped in searches
- requires periodic reorganization to reclaim unused space - records are packed
- can also use unused space when inserting new records - this requires extra bookkeeping

ordered records (sorted files)
- physically order records based on values of a field (ordering field)
- key field - guaranteed to have a unique value in each record

- ordering key - key field that is also the ordering field
- reading file in order of ordering field very efficient
- finding next in order is also efficient
- search condition on ordering field is also efficient (can use binary search - see algorithm on page 86)
- no advantages for random search or search on other fields
- inserting and deleting are expensive because position in order must be found and physical space must be inserted or deleted (moving all subsequent records)
    - deletion markers and periodic reorganization can help this with deletion
- to make insertion more efficient:
    - original file called main or master file (ordered)
    - inserted records inserted at end of overflow file (unordered)
    - periodically two files are merged

hashing

- provides very fast access to records on certain search conditions
- choose a field in the record to be the hash field (hash key if key field is chosen)
- perform some function on the hash field and use the result as the address of the disk block
- internal hashing - used within programs - adapted to be used with db files
    - implemented through an array of records
    - M array slots - choose a hash function that transforms the hash field (K) into an integer between 0 and M-1
    - EX: h(K) = K mod M
    - hash field space - number of possible values a hash field can take
        usually bigger than address space
    - collision - hash value of new record maps to the same address of an existing hash value
    - collision resolution - find another address for insertion
        - open addressing - first available position after full one
        - chaining - place new record in unused overflow location - set pointer of occupied hash address location to the overflow location
        - multiple hashing - apply second hash function and then either use open addressing for subsequent collisions or continue using another hash function

ex: applying the MOD hash function to a char string

```
tmp <- 1;
for i <- to 20 do temp <- temp * code(K[i]);  /* returns int code */
hash_address <- temp mod M;
```

ex: collision resolution by open addressing

```
i<- hash_address;
if location i is occupied
        then begin
                i <- (i+1) mod M;
                while (i<> hash_address) and loc i is occupied
```

```
                    do i <- (i+1) mod M;
            if (i = hash_address)
                    then all positions are full
                    else new_hash_address <- i
    end
```

- external hashing - hashing for disk files
    - target address space made of buckets - either one block or a cluster of contiguous blocks
    - hashing function maps key to a relative bucket number rather than an absolute block address
    - table in file header converts bucket number into disk block address

(picture of external hashing - fig 4.10)

- drawbacks to hashing db files
    - most hash functions do not preserve order of hash field (some do called order preserving hash functions)
    - fixed amount of space allocated to the file (number of buckets)
- searching on non-hash field can be expensive (like unordered files)
- deletion - remove record from bucket
- modifying hash field means moving record to another bucket and removing from old bucket
- read for yourself (if you want) about hashing techniques to allow dynamic file expansion

Files of mixed records

- hierarchical and network db's implement relationships among records physically - physical contiguity of records or record pointers
    - useful when relationships are used often

ex: if a query for a manufacturer and all of the toys manufactured by it is performed often, it makes sense to place each manufacturer and its toys condituously on disk in a mixed file

- each record has a record type field - first field used to determine what data follows