6. Storage and File Structures

Goals

Understand the basic concepts underlying different storage media, buffer management, files structures, and organization of records in files.

Contents

- Overview of Physical Storage Media
- Magnetic Disks, Tertiary Storage
- Buffer Management
- Storage Access
- File Organization

Classification of Physical Storage Media

- Main criteria: Speed with which data can be accessed and cost per unit of data to buy medium

- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device

- Classification:
  - *volatile storage:* loses contents when power is turned off
  - *non-volatile storage:* contents persist when power is switched off

Physical Storage Media

- *Cache:* fastest and most costly form of storage; volatile; managed by the hardware and/or operating system

- *Main Memory:*
  - general purpose instructions operate on data in main memory
  - fast access but in general too small to store the entire database (or even an entire relation)
  - volatile – content of main memory is usually lost if a power failure or system crash occurs

- *Magnetic-Disk Storage:* primary medium for the long term storage of data; typically stores the entire database (i.e., all relations and associated access structures)
  - data must be moved from disk to main memory for access and written back for storage (**insert, update, delete, select**)
  - direct-access, i.e., it is possible to read data on disk in any order
  - usually survives power failures and system crashes; disk failure, however, can destroy data, but is much less frequent than system crashes

- *Optical Storage:* non volatile; CD-ROM/DVD most popular form; Write-Once-Read-Many (WORM) optical disks are typically used for archival storage.

- *Tape Storage:* non-volatile, used primarily for backup and export (to recover from disk failures and to restore previous data), and for archival data
  - sequential access, much slower than disk
  - very high capacity (8GB tapes are common)
  - tape can be removed from drive $\leadsto$ storage cost much cheaper than disk.

Storage Hierarchy

- *Primary Storage:* Fastest media but volatile (cache, main memory)
- *Secondary Storage:* next lower level in hierarchy, non-volatile, moderately fast access time, sometimes also called on-line storage (magnetic disks, flash memory)
- *Tertiary Storage:* lowest level in hierarchy, non-volatile, slow access time, also called off-line storage (magnetic tape, optical storage)

Magnetic Disk Mechanisms

Important here:

- *Access time* (time it takes from when a read or write request is issued to when the data transfer begins) is determined by *seek time* and *Rotational Latency*.
- *Data-Transfer Rate*: the rate at which data can be retrieved from or stored to the disk.
- *Mean time to failure (MTTF)*: the average time the disk is expected to run continuously without any failure.

Optimization of Disk-Block Access

- *Block:* A contiguous sequence of sectors from a single track
  - Data is transfered between main memory and disk at the granularity of blocks
  - Block size ranges from 512 bytes to several kilobytes
- File organization – optimize block access time by organizing the blocks to correspond to how data will be accessed (e.g., store related information on the same or nearby cylinder).
- Non-volatile buffers speed up disk writes by immediately writing blocks to a non-volatile RAM buffer; controller then writes to disk whenever the disk has no other requests.

RAID

- *Redundant Arrays of Independent Disks:* disk organization that takes advantage of utilizing large numbers of inexpensive, mass-market disks
- Main Idea: Improvement of reliability via redundancy, i.e., store extra information that can be used to rebuild information lost in case of a disk failure. Use Mirroring (or shadowing): duplicate every disk (logical disk consists of two physical disks)
- Different RAID levels (0-6) have different cost, performance, and reliability characteristics.

Storage Access

- A database file is partitioned into fixed-length storage units called *blocks* (or *pages*). Blocks/pages are units of both storage allocation and data transfer.
- Database system seeks to minimize the number of block transfers between disk and main memory. Transfer can be reduced by keeping as many blocks as possible in main memory.

- *Buffer Pool:* Portion of main memory available to store copies of disk blocks.
- *Buffer Manager:* System component responsible for allocating and managing buffer space in main memory.

Buffer Manager

Program calls on buffer manager when it needs block from disk

- The requesting program is given the address of the block in main memory, if it is already present in the buffer.
- If the block is not in the buffer, the buffer manager allocates space in the buffer for the block, replacing (throwing out) some other blocks, if necessary to make space for new blocks.
- The block that is thrown out is written back to the disk only if it was modified since the most recent time that it was written to/fetched from the disk.
- Once space is allocated in the buffer, the buffer manager reads in the block from the disk to the buffer, and passes the address of the block in the main memory to the requesting program.

Buffer Replacement Policies

- Most operating systems replace the block least recently used (*LRU strategy*)
- LRU – Use past reference of block as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  LRU can be a bad strategy for certain access patterns involving repeated sequential scans of data files
- Mixed strategy with hints on replacement strategies provided by the query optimizer is preferable (based on the used query processing algorithm(s)).
- *Pinned block:* memory block that is not allowed to be written back to disk
- *Toss immediate strategy:* frees the space occupied by a block as soon as the final record (tuple) of that block has been processed.
- *Most recently used strategy (MRU):* system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation, e.g., the data dictionary is frequently accessed $\leadsto$ keep data dictionary blocks in main memory buffer

File Organization

- A database is stored as a collection of files. Each file is a sequence of records, and a record is a sequence of fields.

- Approaches to organizing records in files:
  - Assume the record size is fixed
  - Each file has records of one particular type only
  - Different files are (typically) used for different relations

Fixed-Length Records

- Simple approach:
  - Store record i starting at byte $n * (i - 1)$, where n is the size of each record.
  - Record access is simple but records may span blocks.

- Deletion of record i (alternatives to avoid *fragmentation*)
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - Link all free records in a *free list*

Free Lists

- Maintained in the block(s) that contains deleted records.

- Store the address of the first record whose content is deleted in the file header.

- Use this first record to store the address of the second available record, and so on.

  One can think of these stored addresses as *pointers*, because they "point" to the location of a record. (linked list)

- More space efficient representation: reuse space for normal attributes of free records to store pointers (i.e., no pointers are stored in in-use records).

- Requires careful programming: *Dangling pointers* occur if a record is moved or deleted and another record contains a pointer to this record; that pointer then doesn't point any longer to the desired record.

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable length for one or more fields (e.g., **varchar**)

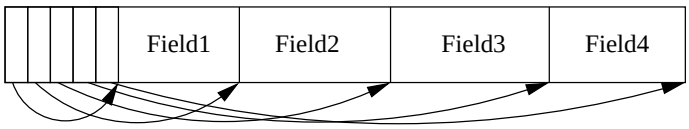
- Approaches to store variable length records:

  1. *End-of-Record* marker
     – difficult to reuse space of deleted records (fragmentation)
     – no space for a record to grow (e.g., due to an update)
       $\rightsquigarrow$ requires moving

  2. *Field delimiters* (e.g., a \$)

     – requires scan of record to get to n-th field value
     – requires a field for a *null* value

  3. Each record has an *array of field offsets*



     + For the overhead of the offset, we get direct access to any field
     + Null values: begin/end pointer of a field point to the same address

Variable length records typically cause problems in case of (record) attribute modifications:

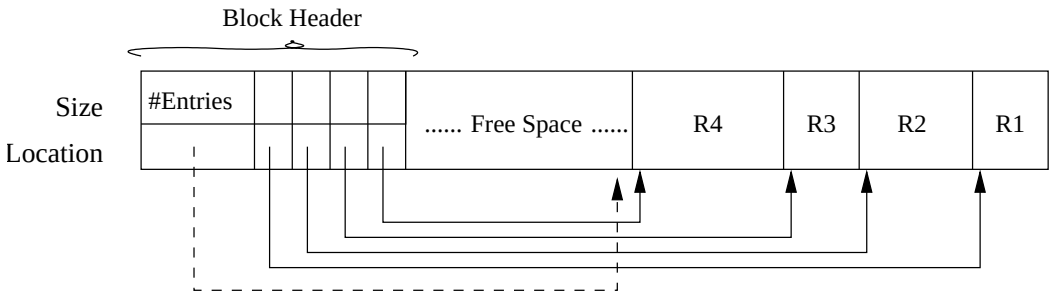
- Growth of a field requires shifting all other fields
- A modified record may no longer fit into the block (leave forwarding address at the old location)
- A record can span multiple blocks

Block formats for variable length records

Each record is identified by a *record identifier (rid)* (or *tuple identifier (tid)*). The rid/tid contains number of block and position in block

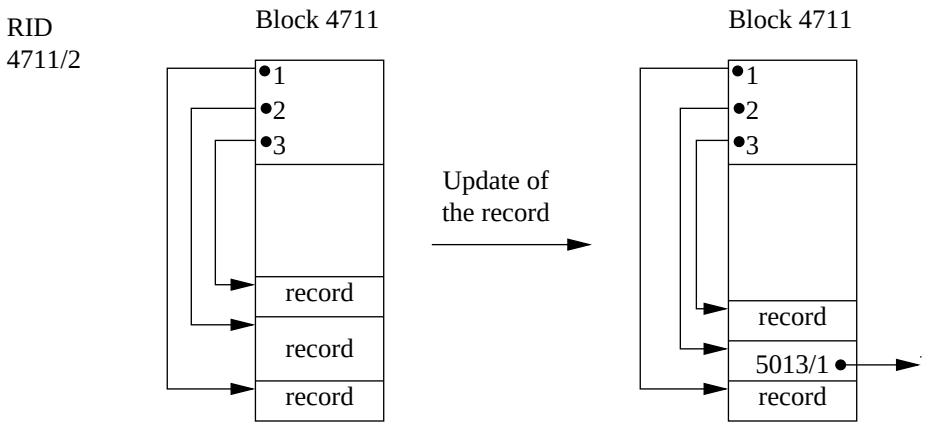
⇝ simplifies shifting a record from one position/block to another position/block

Allocation of records in a block is based on *slotted block structure*:



- block header contains number of record entries, end of the free space in the block, and location and size of each record
- records are inserted from the end of the block
- records can be moved around in block to keep them contiguous

RID-Concept:



Organization of Records in Files

Requirements: A file must (efficiently) support

- insert/delete/update of a record
- access to a record (typically using rid)
- scan of all records

Ways to organize blocks (pages) in a file:

Heap File (unsorted file) simplest file structure; contains records in no particular order; record can be placed anywhere in the file where there is space

Sequential File records are stored in sequential order, based on the value of the search key of each record

Clustered Index related to sequential files; we'll talk about this later in Section 7 (Indexes)

Heap File Organization

- At DB run-time, pages/blocks are allocated and deallocated
- Information to maintain for a heap file includes
  pages, free space on pages, records on a page
- A typical implementation is based on a two doubly linked list of pages; starting with header block.
- Two lists can be associated with header block: (1) full page list, and (2) list of pages having free space

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a *search-key*.
- Deletions of records are managed using pointer chains.
- Insertions – must locate the position in the file where the record is to be inserted
  - if there is free space, insert the record there
  - if no free space, insert the record in an *overflow block*
  - in either case, pointer chain must be updated
- If many record modifications (in particular insertions and deletions), correspondence between search key order and physical order can be totally lost $\Longrightarrow$ file reorganization