Database Systems

Luiz Jonatã Pires de Araújo
l.araujo@innopolis.university

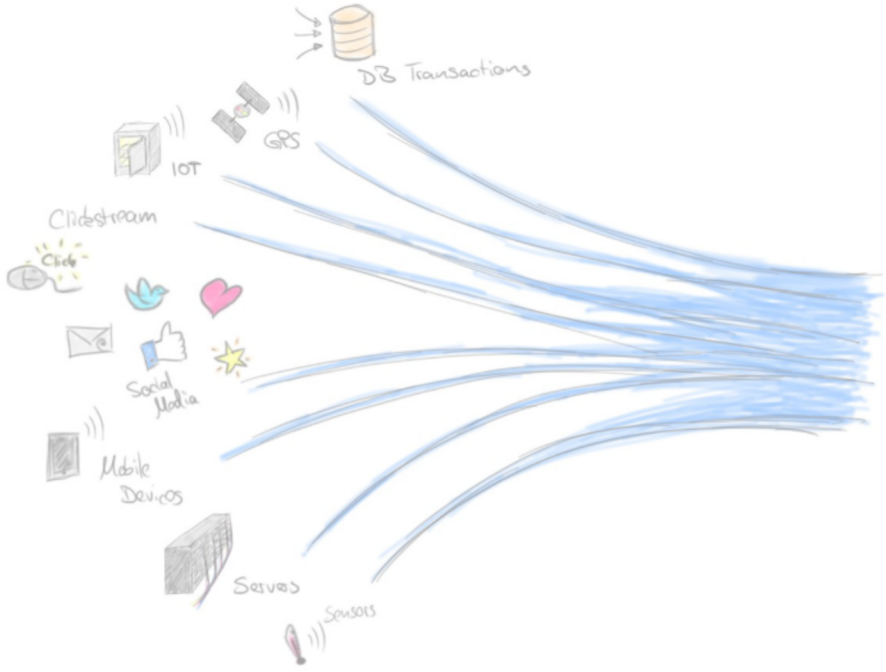
3. Storage Architectures

Giving credit where it's due:

- This material is based on the 7th edition of 'Fundamentals of Database Systems" by Elmasri and Navathe
- https://www.pearson.com/us/higher-education/program/Elmasri-Fundamentals-of-Database-Systems-7th-Edition/PGM189052.html

Outline

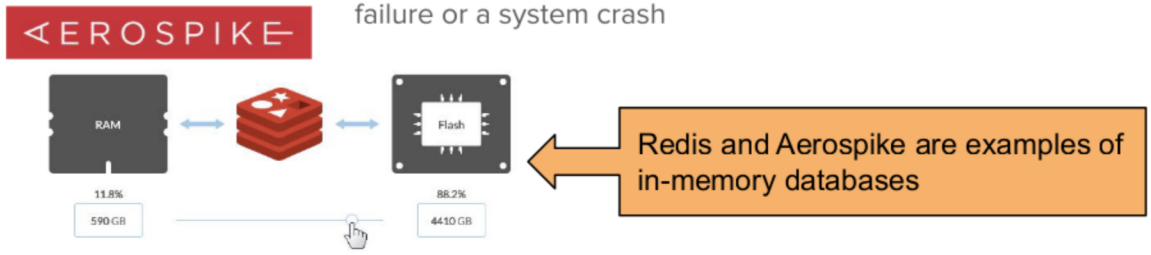
- Introduction
- Secondary Storage Devices
- Buffering of Blocks
- Buffering of Blocks
- Files of Unordered Records (Heap Files)
- Files of Ordered Records (Sorted Files)
- Hashing Techniques
- Other Primary File Organizations



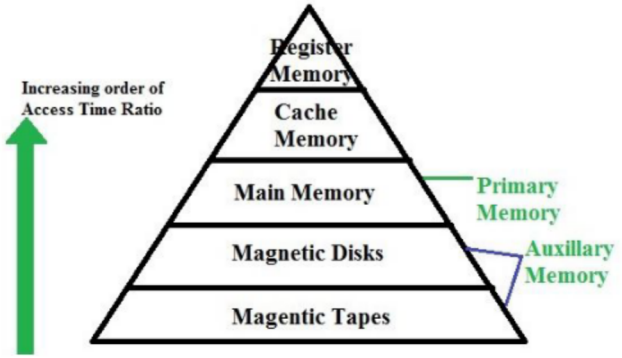
Introduction

Introduction

- The collection of data that makes up a computerized database must be stored physically on some computer storage medium
- The DBMS software can then retrieve, update, and process this data as needed
- Computer storage media form a **storage hierarchy** that includes some main categories:
    - **Primary storage**: usually provides fast access to data but is of limited storage capacity.
        - Examples: main memory and smaller but faster cache memories.
        - They are still more expensive and have less storage capacity
        - Also, the contents of main memory are lost in case of power failure or a system crash



Redis and Aerospike are examples of in-memory databases



Storage hierarchy

- Still on the main categories…
  - **Secondary storage**. The primary choice of storage medium for online and **nonvolatile** storage of enterprise databases has been magnetic disks
    - However, flash memories are becoming a common medium of choice for storing moderate amounts of permanent data
    - When used as a substitute for a disk drive, such memory is called a solid-state drive (SSD)
  - **Tertiary storage**. Optical disks (CD-ROMs, DVDs, and other similar storage media) and tapes
    - Removable media used in today's systems as offline storage
    - These devices usually have a larger capacity, cost less, and provide slower access to data than do primary storage devices
    - It cannot be processed directly by the CPU; first it must be copied into primary storage and then processed by the CPU

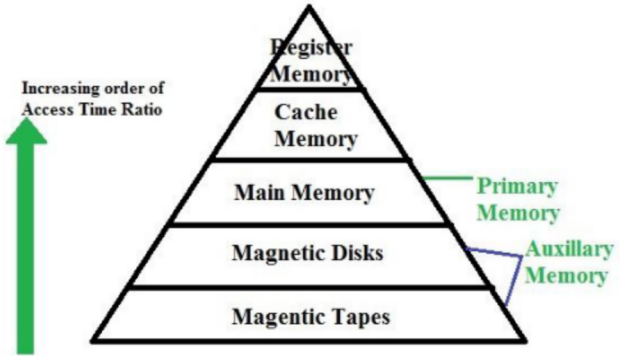


Table 16.1 Types of Storage with Capacity, Access Time, Max Bandwidth (Transfer Speed), and Commodity Cost

Type	Capacity*	Access Time	Max Bandwidth	Commodity Prices (2014)**
Main Memory- RAM	4GB–1TB	30ns	35GB/sec	$100–$20K
Flash Memory- SSD	64 GB–1TB	50µs	750MB/sec	$50–$600
Flash Memory- USB stick	4GB–512GB	100µs	50MB/sec	$2–$200
Magnetic Disk	400 GB–8TB	10ms	200MB/sec	$70–$500
Optical Storage	50GB–100GB	180ms	72MB/sec	$100
Magnetic Tape	2.5TB–8.5TB	10s–80s	40–250MB/sec	$2.5K–$30K
Tape jukebox	25TB–2,100,000TB	10s–80s	250MB/sec–1.2PB/sec	$3K–$1M+

*Capacities are based on commercially available popular units in 2014.

**Costs are based on commodity online marketplaces.

Storage hierarchy

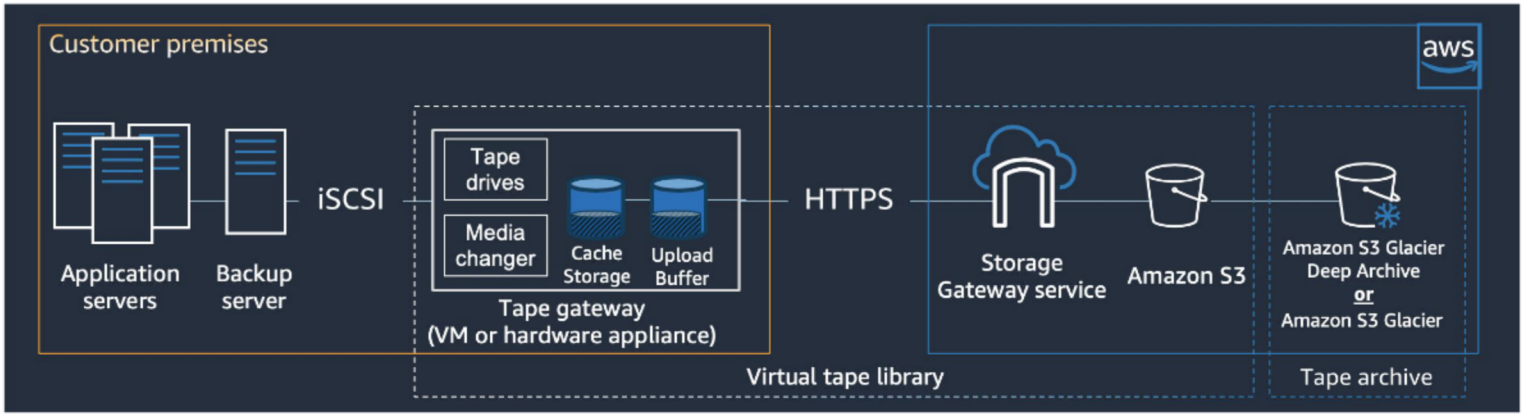
Recommended reading on the use of tape databases:

https://www.securedatarecovery.com/services/tape-data-recovery/why-do-companies-still-use-tape-drives

https://www.networkworld.com/article/3575810/should-you-upgrade-tape-drives-to-the-latest-standard.html

https://www.securedatarecovery.com/services/tape-data-recovery/tape-backup-vs-disk-backup

https://aws.amazon.com/getting-started/hands-on/replace-tape-with-cloud/



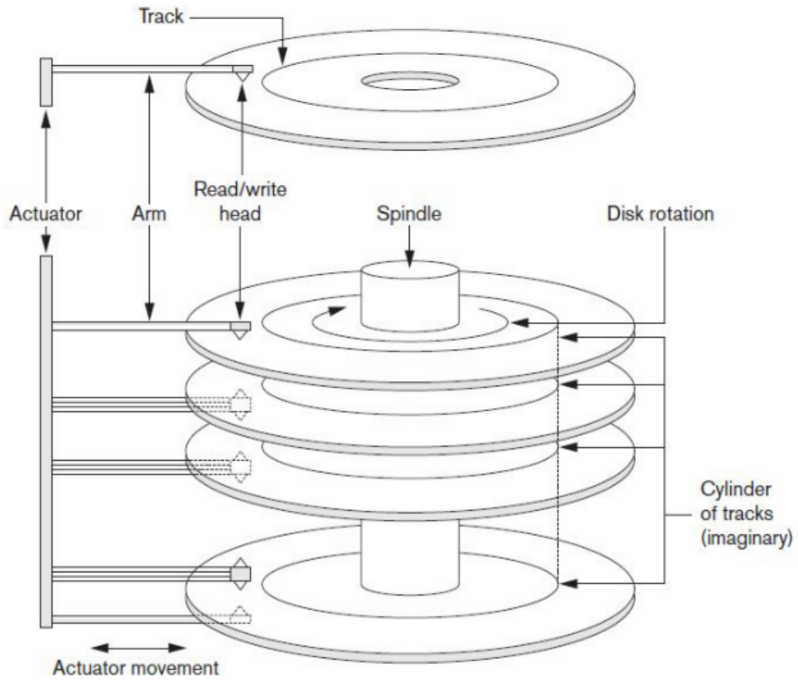
File Organizations

- **Primary file organizations**, which determine how the file records are physically placed on the disk, and hence how the records can be accessed
  - A **heap file** (or unordered file) places the records on disk in no particular order by appending new records at the end of the file,
  - **Sorted file** (or sequential file) keeps the records ordered by the value of a particular field (called the sort key).
  - A **hashed file** uses a hash function applied to a particular field (called the hash key) to determine a record's placement on disk.
  - **B-trees**
- A secondary organization or **auxiliary access structure** allows efficient access to file records based on alternate fields than those that have been used for the primary file organization
  - Most of these exist as indexes

Secondary Storage Devices

Hardware Description of Disk Devices

- Disks are made of magnetic material shaped as a thin circular disk protected by a plastic or acrylic cover
- A disk can be either single-sided or double-sided if both surfaces are used
- To increase storage capacity, disks are assembled into a disk pack, which may include many disks
- Information is stored on a disk surface in concentric circles of small width, each having a distinct diameter. Each circle is called a track
- Disk tracks with the same diameter on the various surfaces are called a cylinder because of the shape they would form if connected in space
- The concept of a cylinder is important because data stored on one cylinder can be retrieved much faster than if it were distributed among different cylinders



Hardware Description of Disk Devices

- Because a track usually contains a large amount of information, it is divided into smaller blocks
- One type of sector organization calls a portion of a track that subtends a fixed angle at the center a sector
  - As an alternative, the sectors subtend smaller angles at the center as one moves
- The division of a track into equal-sized **disk blocks (or pages)** is set by the operating system during disk formatting
- Block size is fixed and cannot be changed dynamically. Typical disk block sizes range from 512 to 8192 bytes

Hardware address of a DISK BLOCK = cylinder + track + block

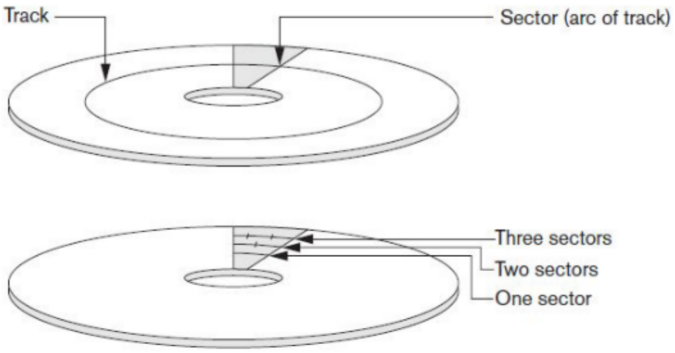


Figure 16.2
Different sector organizations on disk.
(a) Sectors subtending a fixed angle.
(b) Sectors maintaining a uniform recording density.

Locating the data

To transfer a disk block, given its address, the disk controller must:

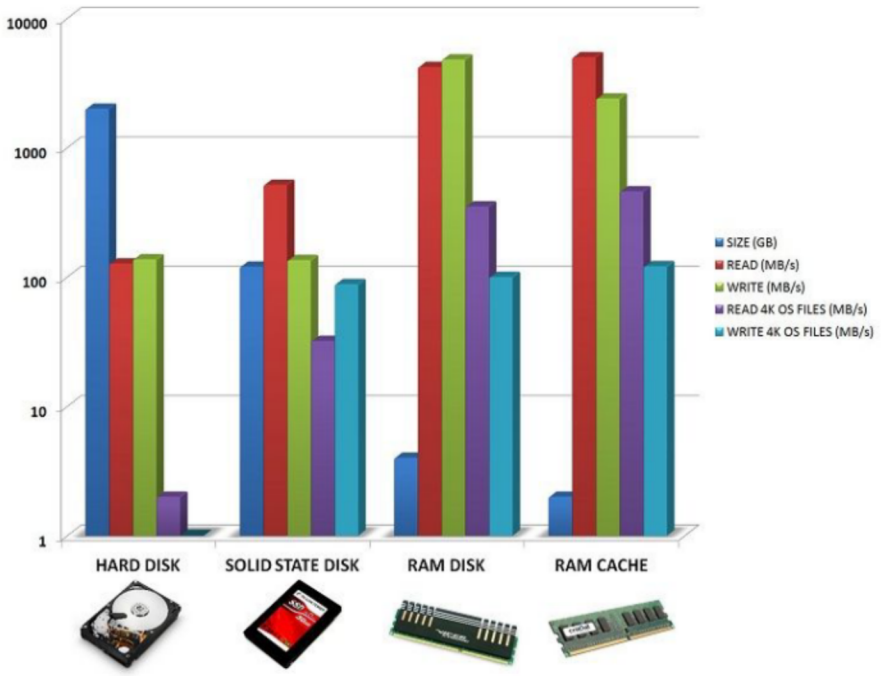
1. Mechanically position the read/write head on the correct track. The time required to do this is called the seek time. Typical seek times are 5 to 10 msec on desktops and 3 to 8 msec on servers
2. There is another delay - called the rotational delay or latency - while the beginning of the desired block rotates into position under the read/write head. It depends on the rpm of the disk, between 2 and 3 msec
3. Finally, some additional time is needed to transfer the data; this is called the block transfer time.
4. (seek time + latency) >> block transfer time

Conclusions

- Locating data on disk is a major bottleneck in database applications
- The file structures attempt to minimize the number of block transfers needed to locate and transfer the required data from disk to main memory
- Placing “related information” on contiguous blocks is the basic goal of any storage organization on disk

Making Data Access More Efficient on Disk

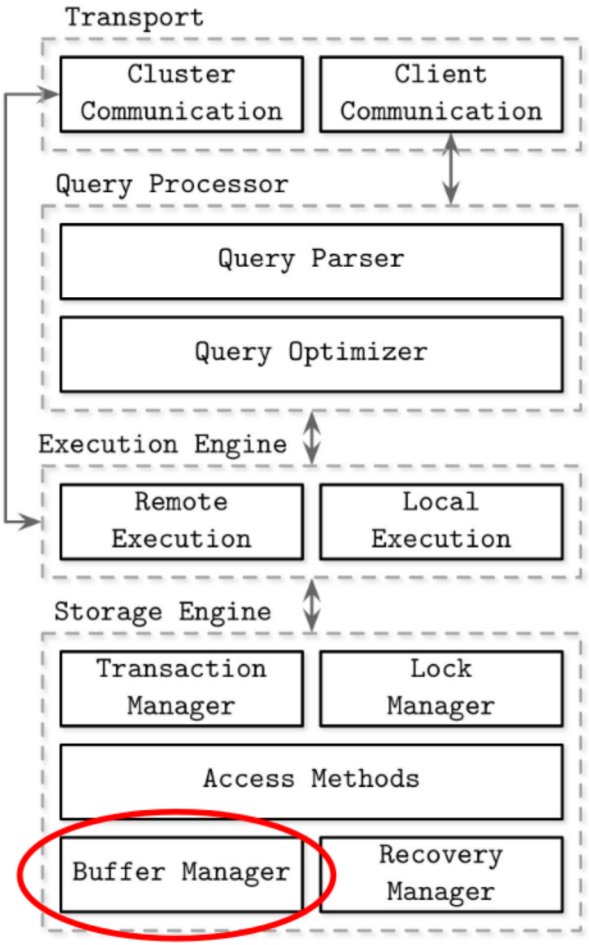
1. **Buffering** of data in memory
2. Proper **organization of data** on disk: keep related data on contiguous blocks
3. **Reading data ahead** of request to minimize seek times
4. Proper **scheduling** of I/O requests, for example using the elevator algorithm
5. Use of log disks to **temporarily hold writes** to minimize arm movement when writing
6. Use of **SSDs** or flash memory for recovery purposes



Buffering of Blocks

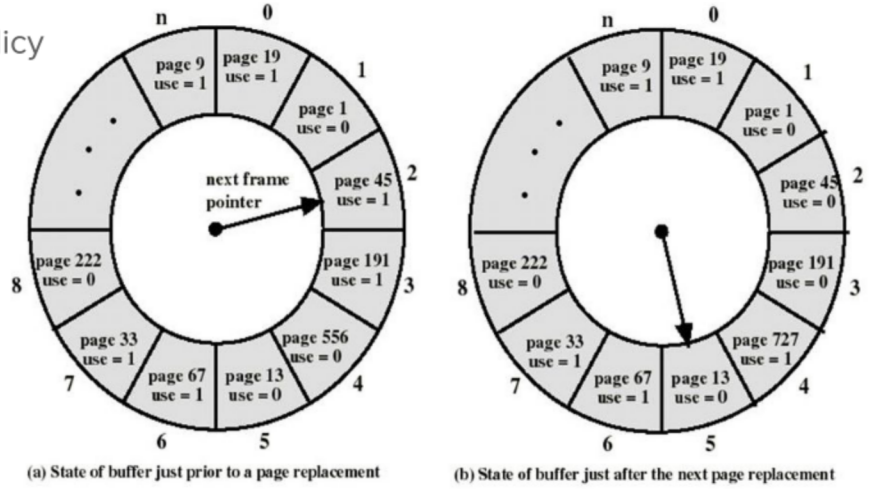
Buffering of Blocks

- Buffer pool is a part of main memory that is available to receive blocks or pages of data from disk
  - The size of the shared buffer pool is typically a parameter for the DBMS controlled by DBAs
- Buffer manager is a software component of a DBMS that responds to requests for data and decides what buffer to use and what pages to replace in the buffer to accommodate the newly requested blocks
  - When a certain page is requested, the buffer manager takes following actions:
    - it checks if the requested page is already in a buffer in the buffer pool; if so, it increments its pin-count and releases the page
    - If the page is not in the buffer pool, the buffer manager does the page for replacement
  - Goals:
    - to maximize the probability that the requested page is found in main memory
    - in case of reading a new disk block from disk, to find a page to replace that will cause the least harm in the sense that it will not be required shortly again using a **pin-count** and a **dirty bit**


Transport
Cluster Communication
Client Communication
Query Processor
Query Parser
Query Optimizer
Execution Engine
Remote Execution
Local Execution
Storage Engine
Transaction Manager
Lock Manager
Access Methods
Buffer Manager
Recovery Manager

Buffer Replacement Strategies

- **Least recently used (LRU)** throws out that page that has not been used (read or written) for the longest time
- **First-in-first-out (FIFO)** needs less maintenance than LRU
- **Clock policy**: This is a round-robin variant of the LRU policy
  - Imagine the buffers are arranged like a clock
  - Each buffer has a flag with a 0 or 1 value
  - When the buffer is accessed, the flag is set to 1
  - Buffers with a 0 are vulnerable and may be used for replacement and their contents read back to disk
  - If the clock hand passes buffers with 1s, it sets them to a zero



(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

See: Internals of buffer manager in PostgreSQL ➜ http://www.interdb.jp/pg/pgsql08.html
Example of how to customize size of the buffer and visualize stats on buffer pages: https://habr.com/en/company/postgrespro/blog/491730/

Placing File Records on Disk

Placing File Records on Disk

- Data in a database is regarded as a set of records organized into a set of files
- Data is usually stored in the form of records
- A collection of field names and their corresponding data types constitutes a record type (numeric, characters, Boolean, date and time).
    - In some database applications, the need may arise for storing data items that consist of large unstructured objects, which represent images, digitized video or audio streams, or free text. These are referred to as BLOBs (binary large objects). A BLOB data item is typically stored separately from its record in a pool of disk blocks
- A file is a sequence of records. In many cases, all records in a file are of the same record type
    - If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records
        - One or more of the fields are of varying size (variable-length fields). For example, the Name field can be a variable-length field
    - If different records in the file have different sizes, the file is said to be made up of variable-length records

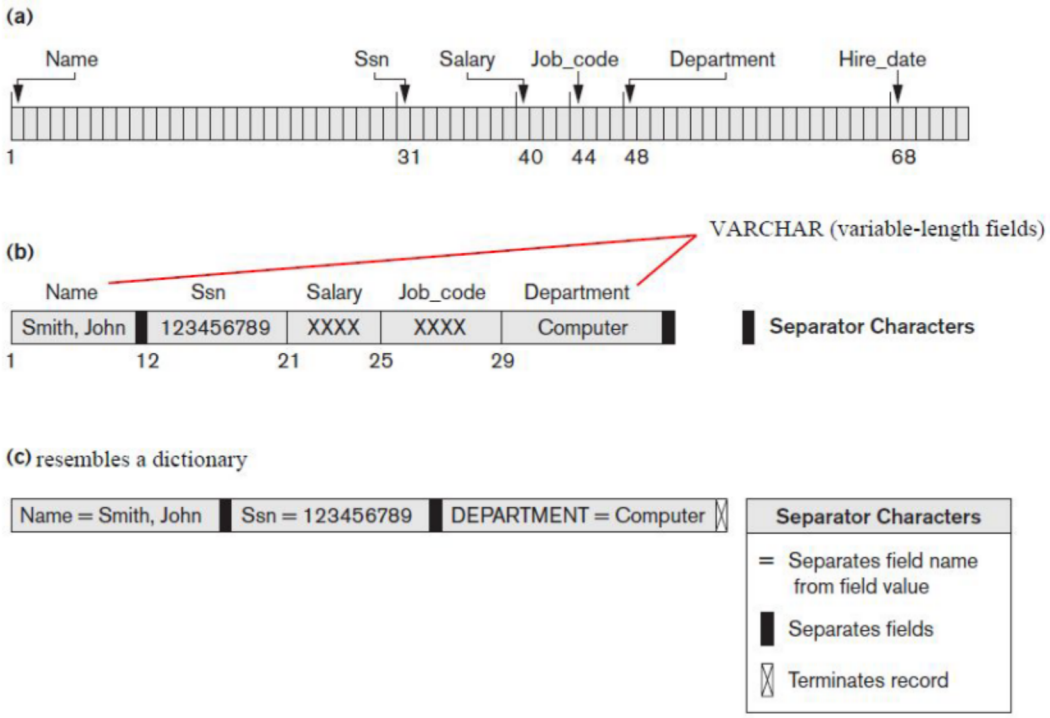


Figure 16.5
Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

Interesting links about the internals of files and pages in PostgreSQL: https://postgrespro.com/docs/postgresql/11/storage
https://habr.com/en/company/postgrespro/blog/469087/
MySQL: https://dev.mysql.com/doc/internals/en/innodb-record-structure.html

Record Blocking; and Spanned vs. Unspanned Records

- Suppose that the block size is B bytes
- For a file of fixed-length records of size R bytes, with $B \geq R$, we can fit $bfr = \lfloor B/R \rfloor$ records per block
- The value bfr is called the blocking factor for the file
- In general, R may not divide B exactly, so we have some unused space in each block $= B - (bfr * R)$ bytes
- To utilize this unused space, we can store part of a record on one block and the rest on another
  - A pointer at the end of the first block points to the block containing the remainder of the record in case it is not the next consecutive block on disk
  - This organization is called spanned because records can span more than one block

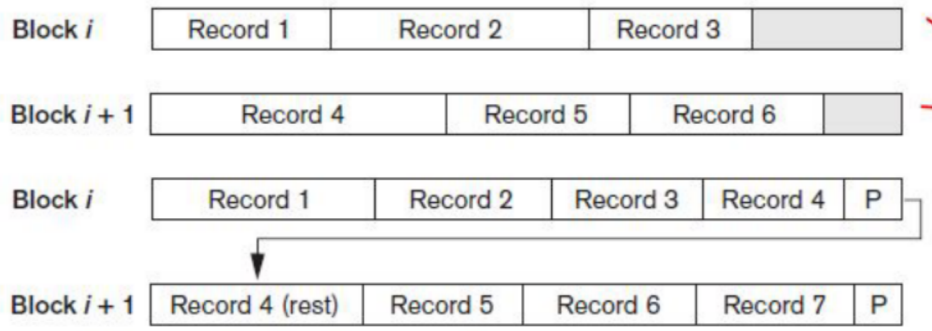


Figure 16.6
Types of record organization.
(a) Unspanned.
(b) Spanned.

File organization vs. and Access Method

- A file organization refers to the organization of the data of a file into records, blocks, and access structures; this includes the way records and blocks are placed on the storage medium and interlinked
  - Some files may be static, meaning that update operations are rarely performed; other, more dynamic files may change frequently, so update operations are constantly applied to them
  - There are different methods to organize records of a file on disk
- An access method, on the other hand, provides a group of operations that can be applied to a file. In general, it is possible to apply several access methods to a file organized using a certain organization
  - Some access methods, though, can be applied only to files organized in certain ways. For example, we cannot apply an indexed access method to a file without an index (next lecture)

About File Organizations (methods for organizing records of a file on disk)

The goal of a good file organization is to avoid linear search or full scan of the file and to locate the block that contains a desired record with a minimal number of block transfers.

Some of the alternatives include:

- Heap files
- Sorted files
- Hash files
- others

Files of Unordered Records (Heap Files)

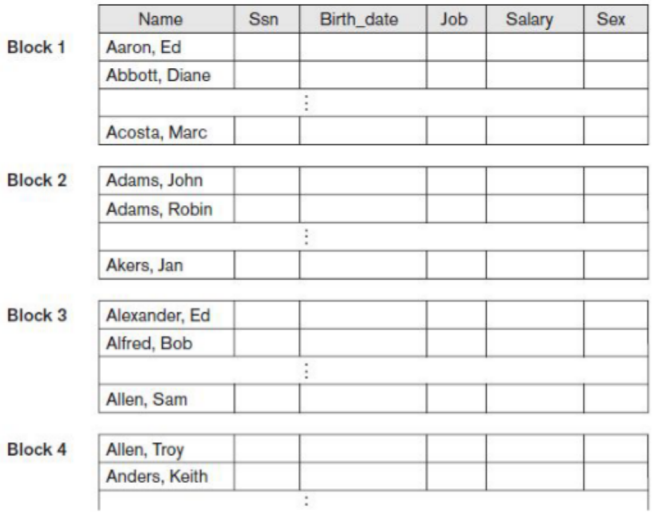
Files of Unordered Records (Heap Files)

- The simplest and most basic type of organization
- Records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file
- This organization is often used with additional access paths, such as the secondary indexes
- Inserting a new record is very efficient
  - The last disk block of the file is copied into a buffer
  - The new record is added, and
  - The block is then rewritten back to disk
- To delete a record, a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally rewrite the block back to the disk.
  - This leaves unused space in the disk block → Internal fragmentation
  - Requires periodic reorganization
- Searching for a record using any search condition involves a linear search

Files of Ordered Records (Sorted Files)

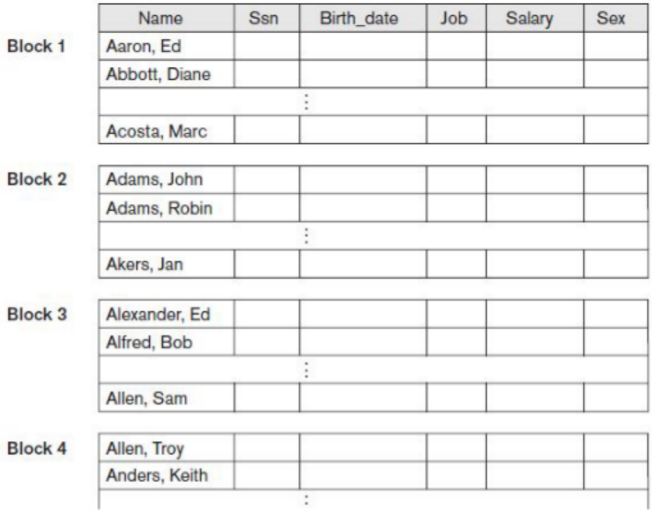
Files of Ordered/Sequential Records (Sorted Files)

- Records of a file on disk are ordered based on the values of one of their fields (the ordering field)
- If the ordering field is also a key field of the file, then the field is called the ordering key for the file.
- Advantages:
  - Reading the records in order of the ordering key values becomes extremely efficient
  - Finding the next record from the current one in order of the ordering key usually requires no additional block accesses
  - Using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used
    - A binary search for disk files can be done on the blocks rather than on the records
    - Requires log(b) blocks, whether the record is found or not



Files of Ordered/Sequential Records (Sorted Files)

- Disadvantages:
  - Inserting and deleting records are expensive operations for an ordered file because the records must remain physically ordered
    - To insert a record, we must find its correct position in the file, based on its ordering field value, and then make space in the file to insert the record in that position
    - For a large file this can be very time consuming because, on the average, half the records of the file must be moved to make space for the new record
    - One option for making insertion more efficient is to keep some unused space in each block for new records
    - For record deletion, the problem is less severe if periodic reorganization is used



	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
	⋮					

Table 16.3 Average Access Times for a File of b Blocks under Basic File Organizations

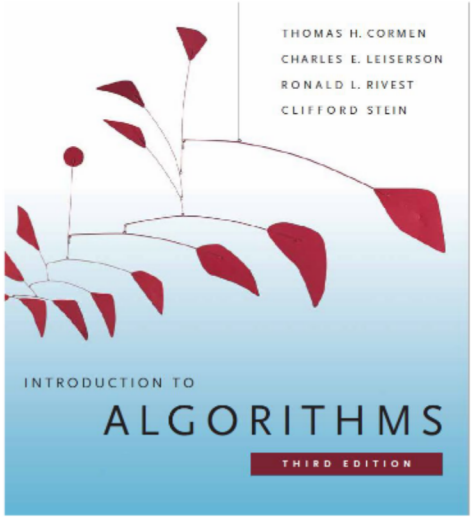
Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

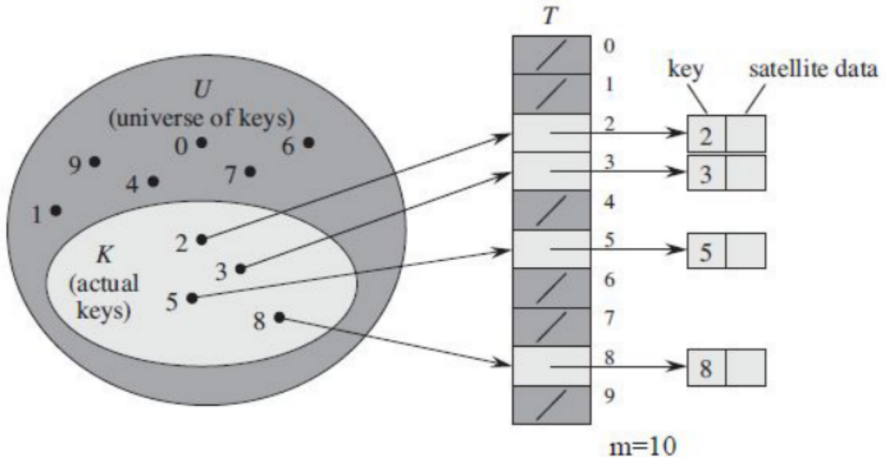
Hash Files

Hash Files

Characteristics:

- Fast access to records - average search time = O(1)
- The search condition must be an equality condition on a single field, called the hash field
- In most cases, the hash field is also a key field of the file, in which case it is called the hash key
- The idea behind hashing is to provide a function h, called a hash function or randomizing function, which is applied to the hash field value of a record and yields the address of the disk block in which the record is stored
- A search for the record within the block can be carried out in a main memory buffer
- For most records, we need only a single-block access to retrieve that record





Theorem 11.1

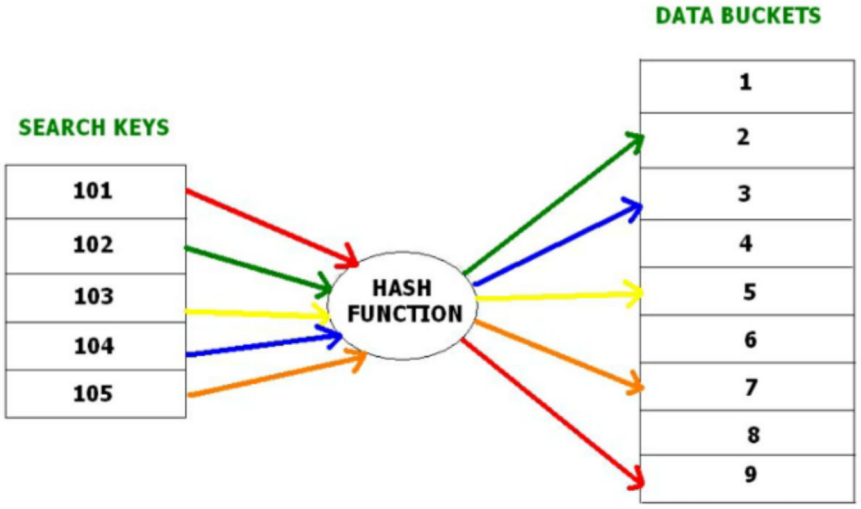
In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Theorem 11.6

Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

Internal Hashing (field→offset)

- We choose a hash function that transforms the hash field value into an integer between 0 and M − 1, being M the range of slots for records within the file
- One example of hash function is the **h(K) = K mode M** function, which returns the remainder of an integer hash field value K after division by M; this value is then used for the record address



Buckets → external hashing

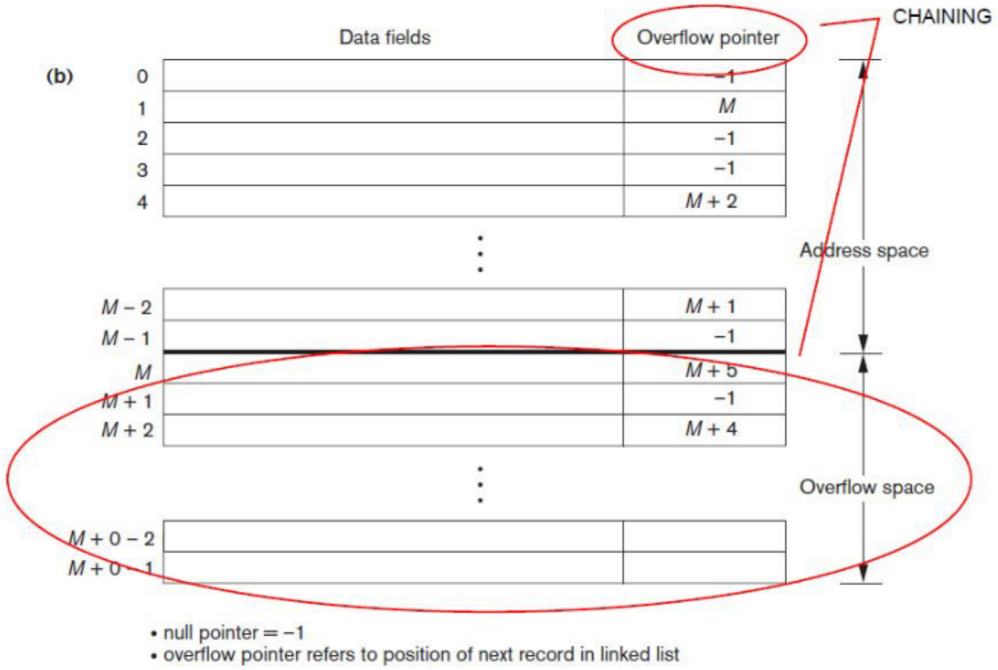
Hash Files - Hashing Collision

- A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In this situation, we must insert the new record in some other position.
- There are numerous methods for collision resolution, including the following:
  - Open addressing: proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found
  - Chaining: for this method, various overflow locations are kept, usually by extending the array with a number of overflow positions
  - Multiple hashing: The program applies a second hash function if the first results in a collision
- The algorithms for chaining are the simplest. Deletion algorithms for open addressing are rather tricky

(a)

	Name	Ssn	Job	Salary
0				
1				
2				
3				
	⋮			
$M-2$				
$M-1$				

Figure 16.8
Internal hashing data structures. (a) Array of M positions for use in internal hashing. (b) Collision resolution by chaining records.



Hash Files - Goals

1. To distribute the records uniformly over the address space so as to minimize collisions, thus making it possible to locate a record with a given key in a single access
2. To achieve (1) yet occupy the buckets fully, thus not leaving many unused locations

Simulation and analysis studies have shown that it is usually best to keep a hash file between 70 and 90% full so that the number of collisions remains low and we do not waste too much space.

External Hashing

- The target address space is made of buckets, each of which holds multiple records
- A **bucket** is either one disk block or a cluster of contiguous disk blocks
- The hashing function maps a key into a relative bucket number rather than assigning an absolute block address to the bucket
- The collision problem is less severe with buckets, because as many records as will fit in a bucket can hash to the same bucket without causing problems

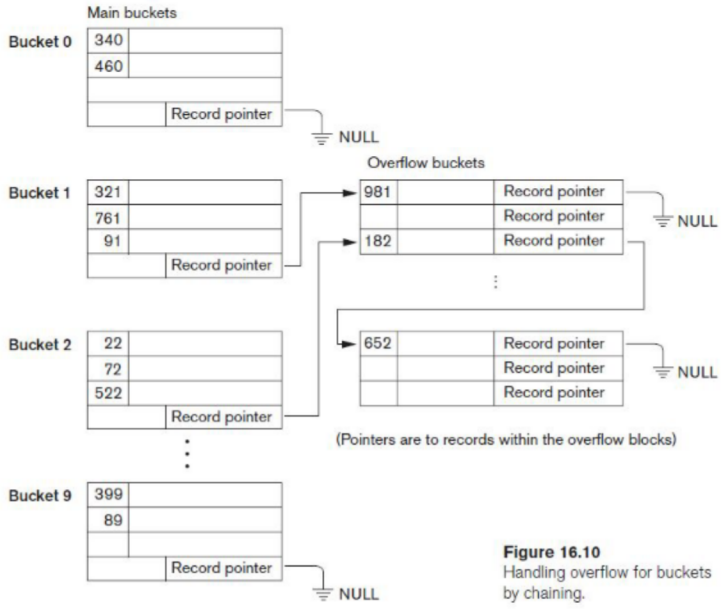


Figure 16.10
Handling overflow for buckets by chaining.

Extendible Hashing

- A type of directory - an array of 2^d bucket addresses - is maintained, where d is called the global depth of the directory
- The integer value corresponding to the first (highorder) d bits of a hash value is used as an index to the array to determine a directory entry, and the address in that entry determines the bucket in which the corresponding records are stored
- However, there does not have to be a distinct bucket for each of the 2^d directory locations

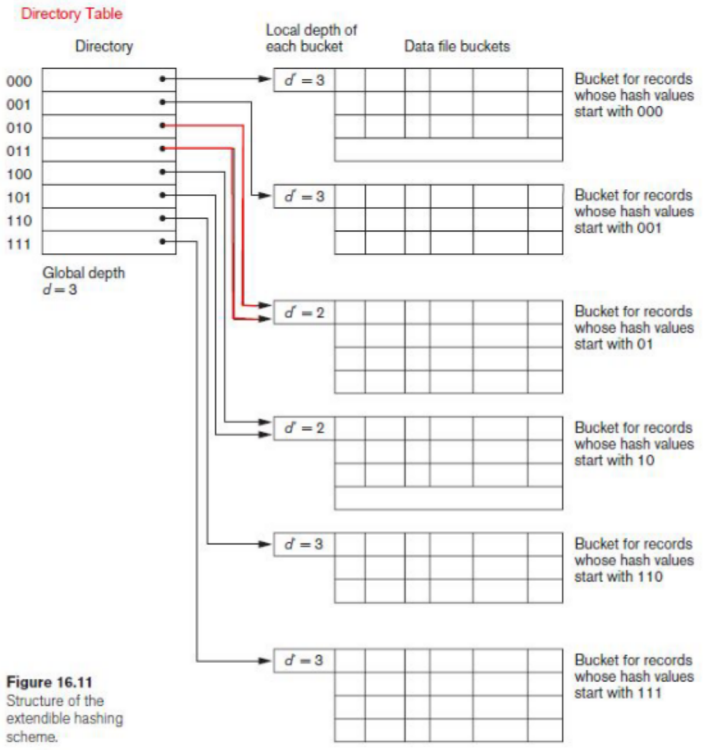


Figure 16.11
Structure of the extendible hashing scheme.

Extendible Hashing

- Several directory locations with the same first d′ bits for their hash values may contain the same bucket address if all the records that hash to these locations fit in a single bucket
- A local depth d′—stored with each bucket—specifies the number of bits on which the bucket contents are based
- Figure 16.11 shows a directory with global depth d = 3
- The value of d can be increased or decreased by one at a time, thus doubling or halving the number of entries in the directory array

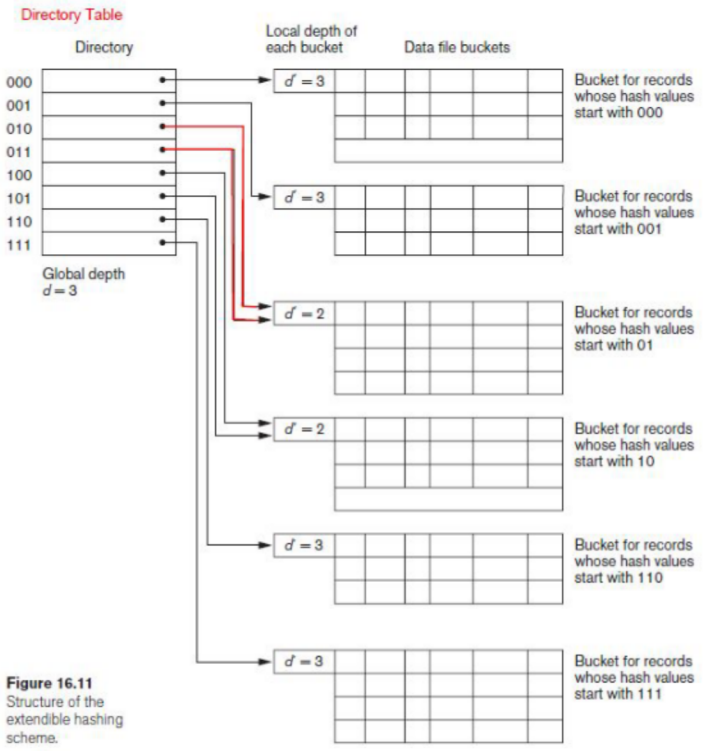


Figure 16.11
Structure of the extendible hashing scheme.

Other Primary File Organizations

Other Primary File Organizations

- B+ tree
- Column-based storage of data
  - Column-oriented databases improve the performance by reducing disk IO (mainly by compressing similar columnar data)
  - Work well for OLAP-like workloads (e.g. data warehouses)
  - ClickHouse (Yandex), Apache Kudu

- Object-Based Storage
  - The data is managed in the form of objects rather than files made of blocks
  - Objects carry metadata that contains properties that can be used for managing those objects
  - Each object carries a unique global identifier that is used to locate it

Ceph Object storage: