CS322: Database Systems

Storage and File Structure

Dr. Manas Khatua
Assistant Professor
Dept. of CSE
IIT Jodhpur
E-mail: manaskhatua@iitj.ac.in

Why Storage in DBMS?

- Database system provides a high-level view of data, but, ultimately data have to be stored as bits on one or more storage devices
- The physical characteristics of storage devices play a major role
- Access to a random piece of data on disk is much slower than memory access
- Disk access takes tens of milliseconds, whereas memory access takes a tenth of a microsecond.
- We would learn
  - An overview of physical storage media, including mechanisms to minimize the chance of data loss due to device failures
  - How records are mapped to files, which in turn are mapped to bits on the disk.
  - Querying a record in file

Physical Storage Media

- Several types of data storage exist

- The storage media are classified
  - by Speed with which data can be accessed
  - by Cost per unit of data
  - by Reliability
    - data loss on power failure or system crash
    - physical failure of the storage device

- Can differentiate storage into:
  - Volatile storage:
    - Loses contents when power is switched off
  - Non-volatile storage:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as battery-backed up main-memory.

Cont…

- **Cache** – fastest and most costly form of storage; volatile; managed by the computer system hardware.
- **Main memory**:
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - capacities of up to a few GB widely used currently
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly
  - Volatile in nature

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles.
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices (e.g., digital cameras, phones, and USB keys)
  - Two types – NAND and NOR flash
  - used in server systems to improve performance by caching frequently used data

Cont…

- **Magnetic-disk**
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data is stored on spinning disk, and read/written magnetically
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory
  - direct-access – possible to read data on disk in any order, unlike magnetic tape
  - Capacities range
    - Much larger capacity and cost/byte than main memory/flash memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

Cont…

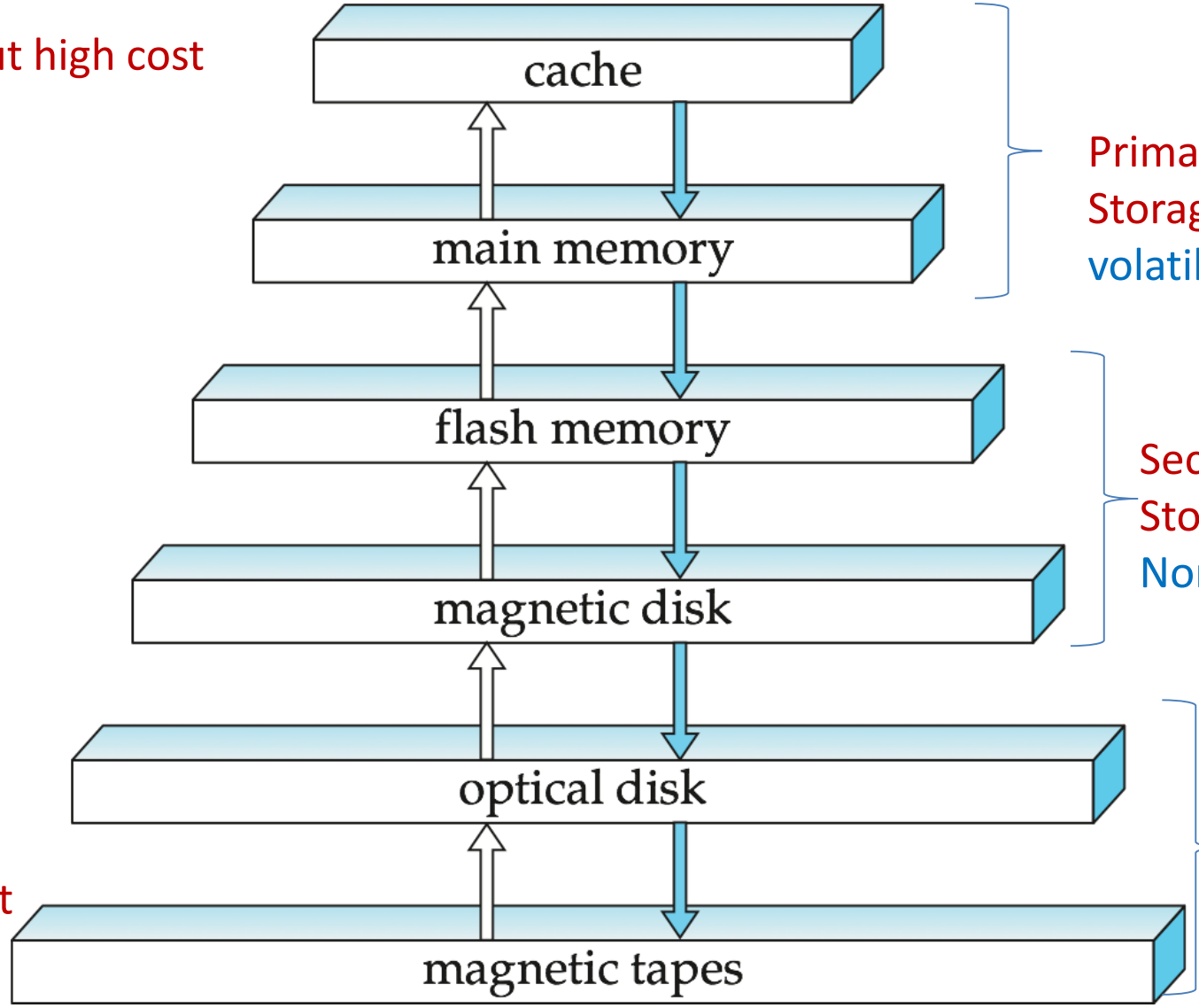
- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - Compact Disk (CD) (640 MB) and Digital Video Disk (DVD) (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - Juke-box systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

Cont…

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - sequential access
  - much slower than disk
  - very high capacity (40 to 300 GB tapes available)
  - tape can be removed from drive ⇒ storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - hundreds of terabytes (1 terabyte = 10^9 bytes) to even multiple **petabytes** (1 petabyte = 10^{12} bytes)

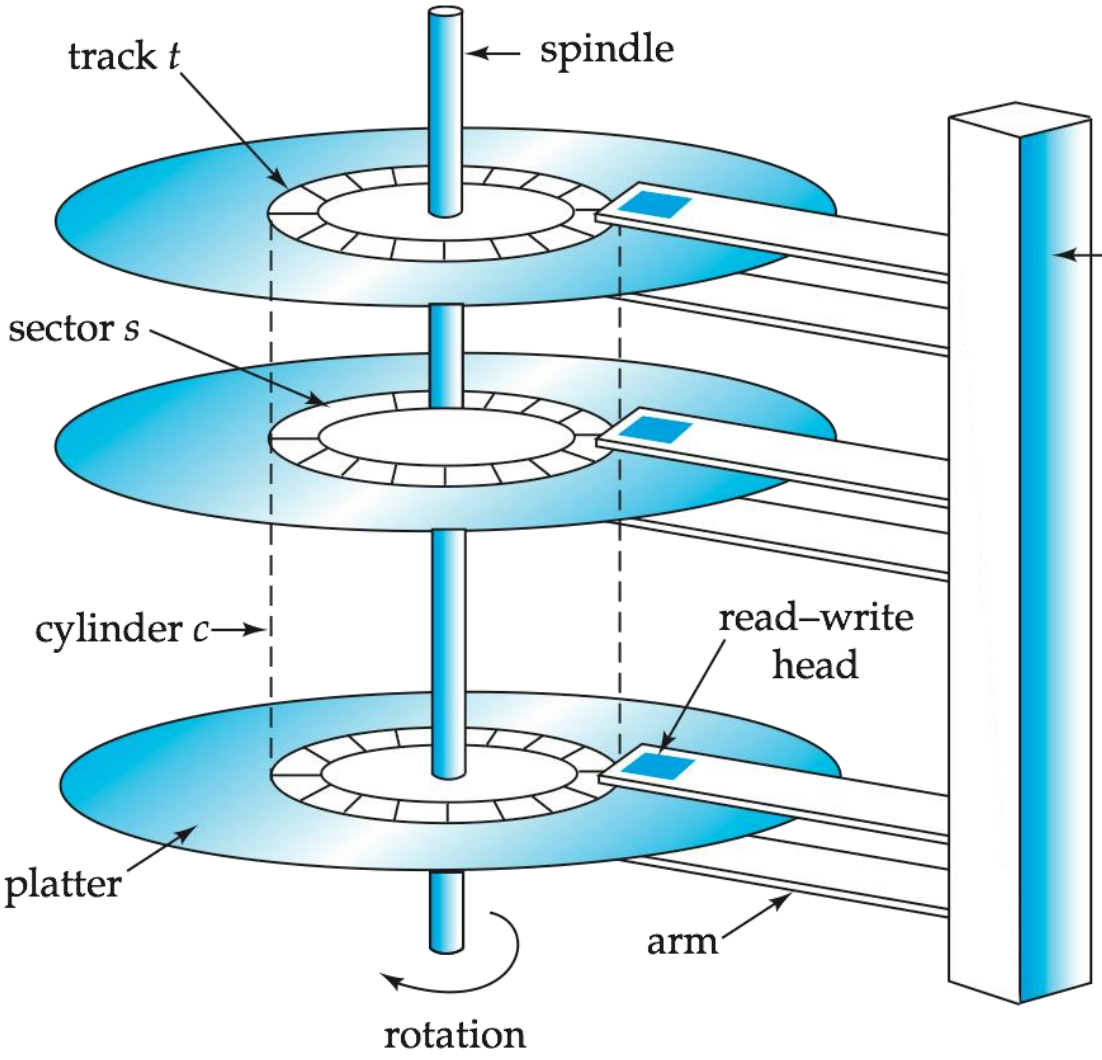
Storage Hierarchy

Fastest, but high cost



Cheapest, but slow speed

Magnetic Hard Disk Mechanism



- Platters are made from rigid metal or glass.
- Its two surfaces are covered with a magnetic material, and information is recorded on the surfaces
- a drive motor spins the platter at a constant high speed (60-120 rpm)
- a read-write head positioned just above the surface of the platter

Cont…

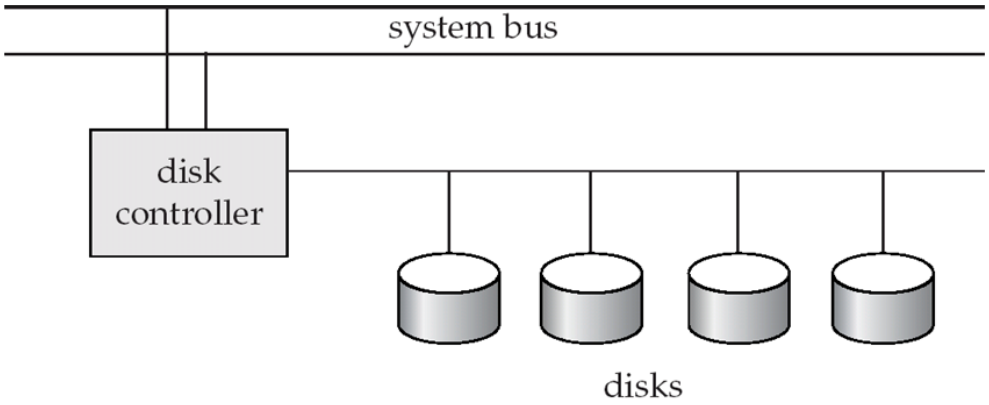
- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors.**
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)

- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - multiple disk platters on a single spindle (1 to 5 usually)
  - one head per platter, mounted on a common arm.
- **Cylinder** *i* consists of i^{th} track of all the platters

Cont…

- Head Crash:
  - Earlier generation disks were susceptible to head-crashes
  - head typically floats or flies only microns from the disk surface
  - If the head contacts the disk surface, the head can scrape the recording medium off the disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted
- Today, disks with a platter diameter of 3.5 inches dominate the market
- Disk controller – interfaces between the computer system and the disk drive
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors. it can logically map the bad sector to a different physical location

Disk Subsystem



- Multiple disks connected to a computer system through a controller
- Common interfaces for connecting disks to computers
  - ATA (AT Adaptor)
  - SATA (Serial ATA)
  - SCSI (Small Computer System Interconnect) range of standards
  - SAS (Serial Attached SCSI)
- In **Storage Area Networks (SAN)**, a large number of disks are connected by a high-speed network to a number of servers
- In **Network Attached Storage (NAS)** networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Performance Measures of Disks

- **Access time** – the time it takes from when a read / write request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - Typical seek time 2 to 30 milliseconds on typical disks
    - Average seek time 4 to 10 milliseconds
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milliseconds on typical disks (speed: 5400 to 15000 r.p.m.)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 100 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA (150 MB/sec), SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI (320 MB/sec), SAS (3 - 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/sec

Cont…

- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Probability of failure of new disks is quite low; almost 5,00,000 to 12,00,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - Most disks have an expected life span of about 5 years
  - MTTF decreases as disk ages

Optimization of Disk-Block Access

- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes

- Sequence of requests for blocks from disk may be classified as
  - a sequential access pattern
    - Successive requests are for successive block numbers, which are on the same track, or on adjacent tracks.
    - a disk seek may be required for the first block; subsequent block require a seek to an adjacent track or in same track
  - a random access pattern
    - successive requests are for blocks that are randomly located on disk
    - Each such request would require a seek.

Optimization of Disk-Block Access

- Techniques for improving the speed of access to blocks
  - Read-ahead in-memory buffering
  - Scheduling: If several blocks from a cylinder need to be transferred from disk to main memory, it may save access time by requesting the blocks in the order in which they will pass under the heads. Scheduling allows to do this.
  - File organization: To reduce block-access time, we can organize blocks on disk in a way that corresponds closely to the way we expect data to be accessed.
  - Nonvolatile write buffers: The idea is that, when the DBMS (or the OS) requests that a block be written to disk, the disk controller writes the block to an NV-RAM buffer, and immediately notifies the OS that the write completed successfully. The controller writes the data to their destination on disk whenever the disk does not have any other requests, or when the NV-RAM buffer becomes full.
  - Log disk: a disk devoted to writing a sequential log—in much the same way as a NV-RAM buffer. The log disk can do the write later, without the DBMS having to wait for the write to complete.

Disk-arm Scheduling

- Disk-arm-scheduling algorithms attempt to order pending accesses to tracks so that disk arm movement is minimized
  - elevator algorithm:
    - For each track for which there is an access request, the arm stops at that track, services requests for the track, and then continues moving outward until there are no waiting requests for tracks farther out.
    - At this point, the arm changes movement direction, again stopping at each track for which there is a request, until it reaches a track where there is no request for tracks farther toward the center.

File Organization, Record Organization and Storage Access

File Organization

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

- Each file is also logically partitioned into fixed-length storage units called blocks,
  - blocks are the units of both storage allocation and data transfer
  - databases use block sizes of 4 to 8 KB by default
  - A block may contain several records

- Record Types based on size
  - Fixed-Length Record
  - Variable-Length Record

- One simple approach for Fixed Size:
  - assume record size is fixed
  - assume that *no record is larger than a block*.
  - each file has records of one particular type only
  - different files are used for different relations

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record.
  - Record access is simple; but deletion is difficult

- Deletion of record *i:*

  Few alternatives*:*

  1) move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$

  2) move record *n* to *i*

  3) do not move records, but link all free records on a *free list*

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting record 3 and compacting

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

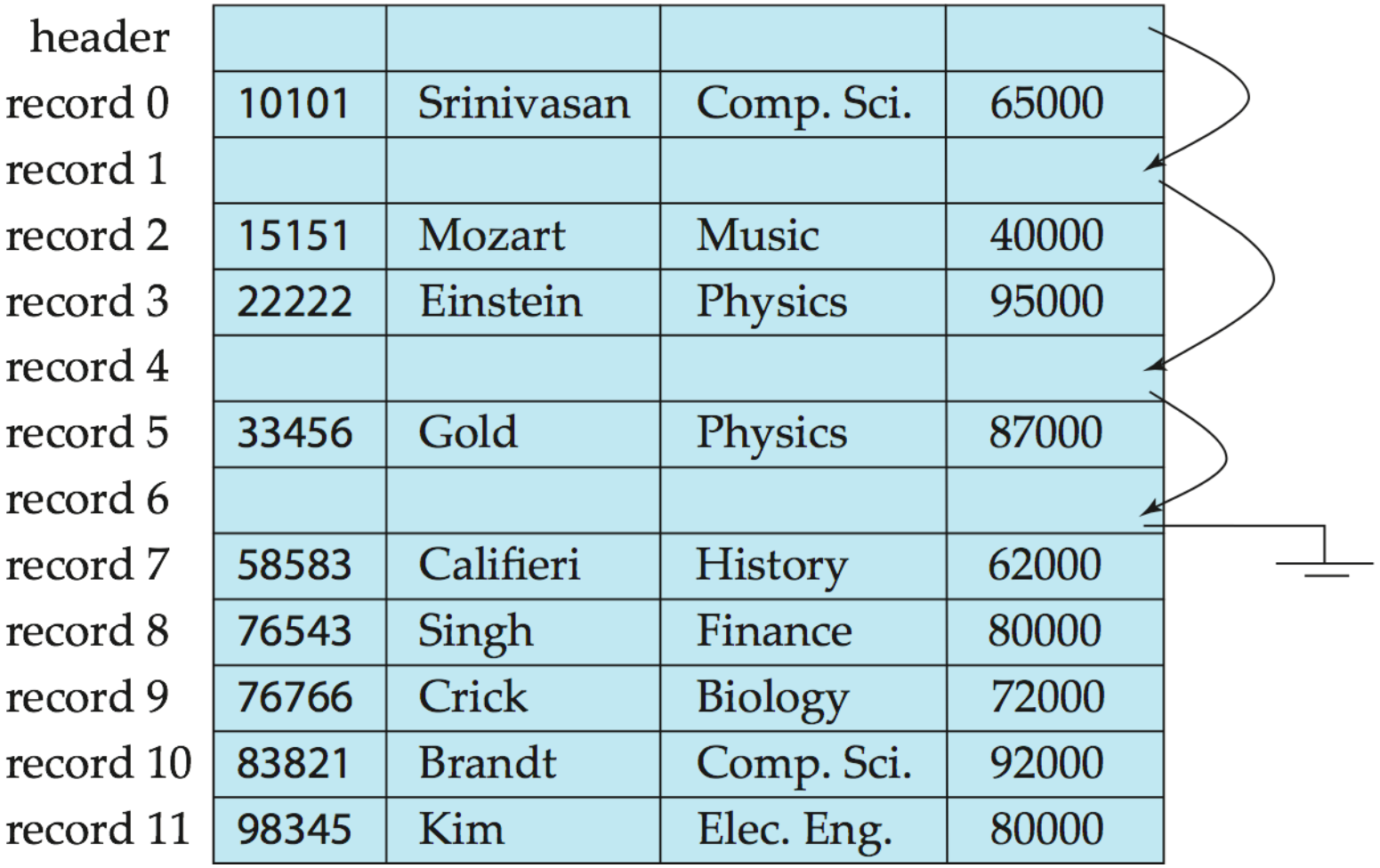
Approach 1

Deleting record 3 and moving last record

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Approach 2

Deleting records and Maintain Free Lists



header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

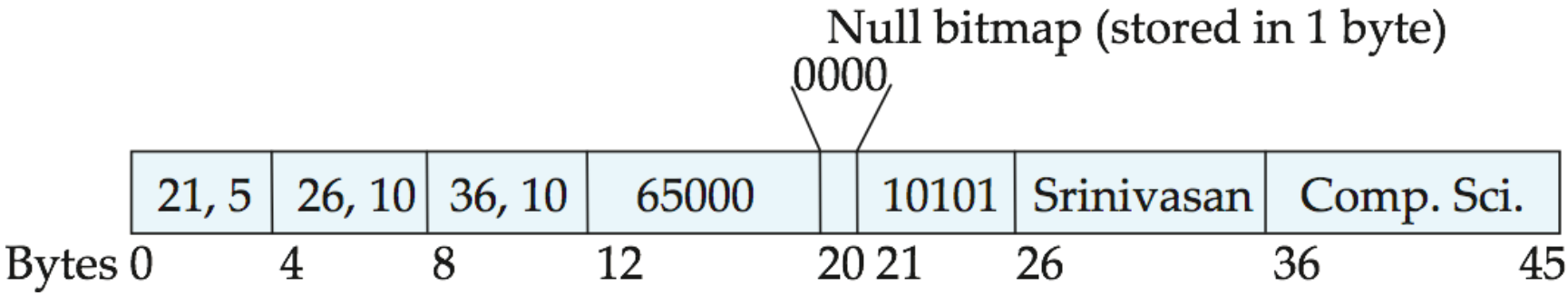
Approach 3

Variable-Length Records

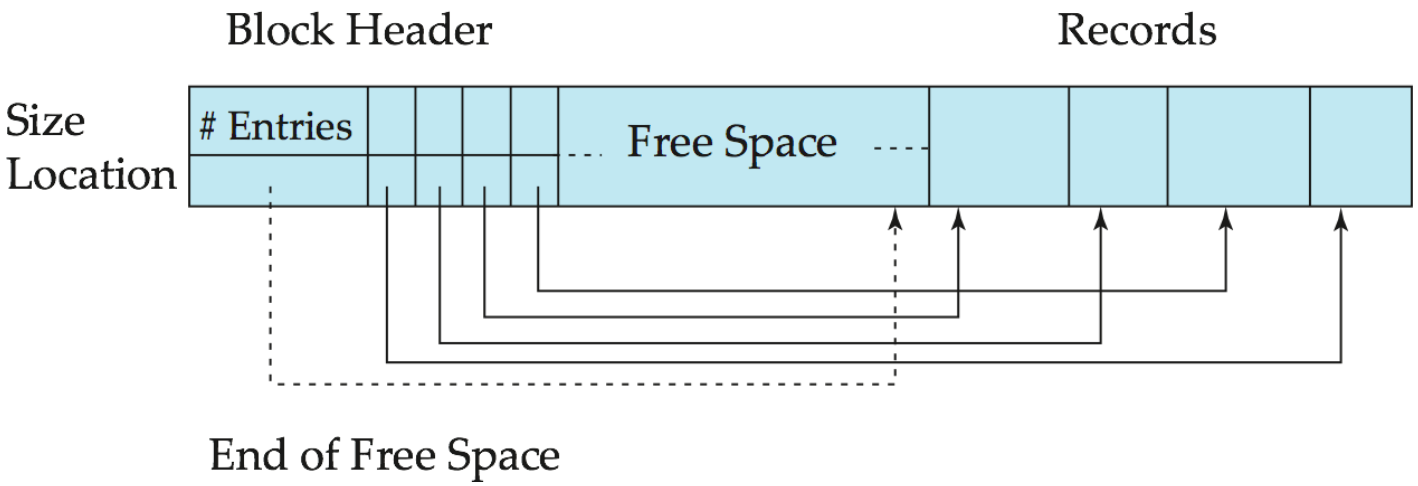
- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (varchar)
  - Record types that allow repeating fields.

- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes

- Null values represented by **null-value bitmap**
- Variable length records are stored in **slotted page structure**



Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries (#entries)
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

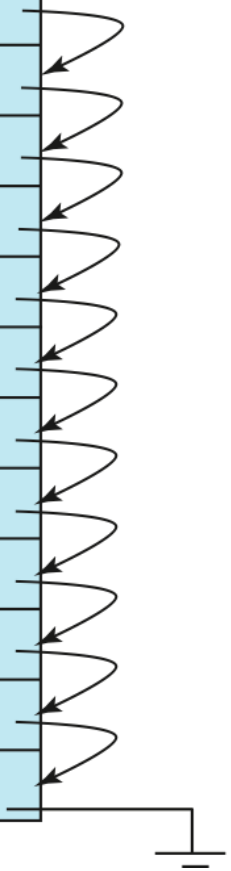
Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Generally, a separate file is used to store the records of each relation.
- **Multitable clustering:** In a multitable clustering file organization, records of several different relations can be stored in the same file
  - Motivation: store related records in the same block to minimize I/O

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

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

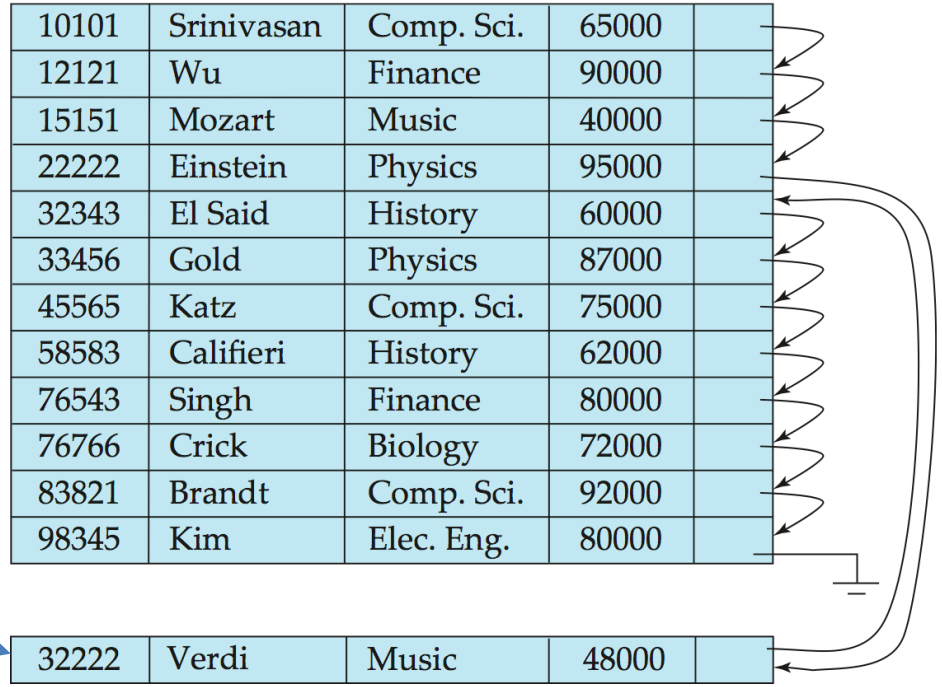


Cont…

- Deletion – use pointer chains

- Insertion – locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an **overflow block**
  - In either case, pointer chain must be updated

- Need to reorganize the file from time to time to restore sequential order



Multitable Clustering File Organization

- Many relational database systems store each relation in a separate file
- Usually, tuples of a relation can be represented as fixed-length records. Thus, relations can be mapped to a simple file structure.
- In this case, DBMS completely depends on the OS file management.
- This simple approach to relational database implementation becomes less satisfactory when the size of the database increases.
- **Solution**: multiple relations are stored in a single file
  - one large OS file is allocated to the database system. The DBMS stores all relations in this one file, and manages the file itself
- Let's examine one query

  select *dept_name*, *building*, *budget*, *ID*, *name*, *salary*

  from *department* **natural join** *instructor*;

Cont…

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

department

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

instructor

- In the worst case, each record will reside on a different block, forcing us to do one block read for each record required by the query.

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

multitable clustering of *department* and *instructor*

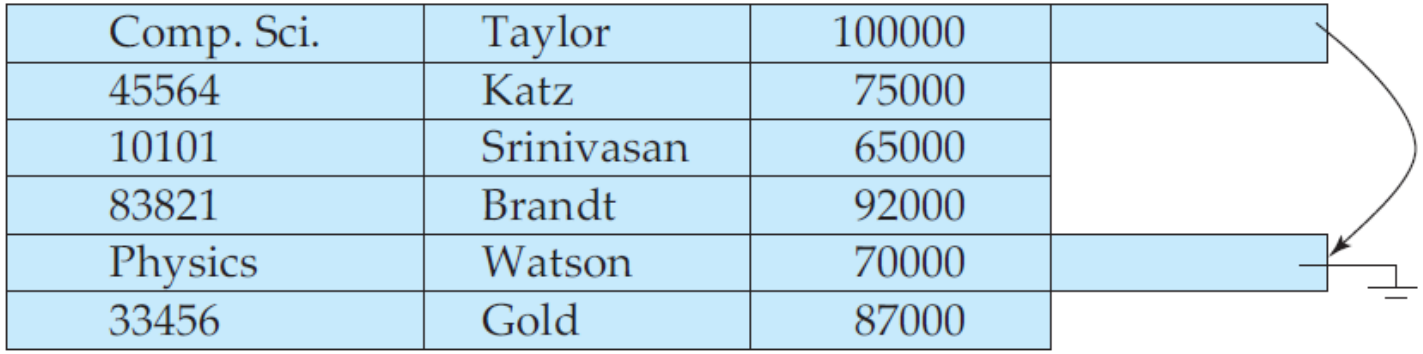
- This structure mixes together tuples of two relations, but allows for efficient processing of the join;
- Because, the corresponding *instructor* tuples are stored on the disk near the *department* tuple.

Cont…

- good for queries involving *department* <natural join> *instructor*, and for queries involving one single department and its instructors

- bad for queries involving only *department*
- results in variable size records

- Solution:
  - Can add pointer chains to link records of a particular relation



Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Figure 10.15 Multitable clustering file structure with pointer chains.

Data Dictionary Storage

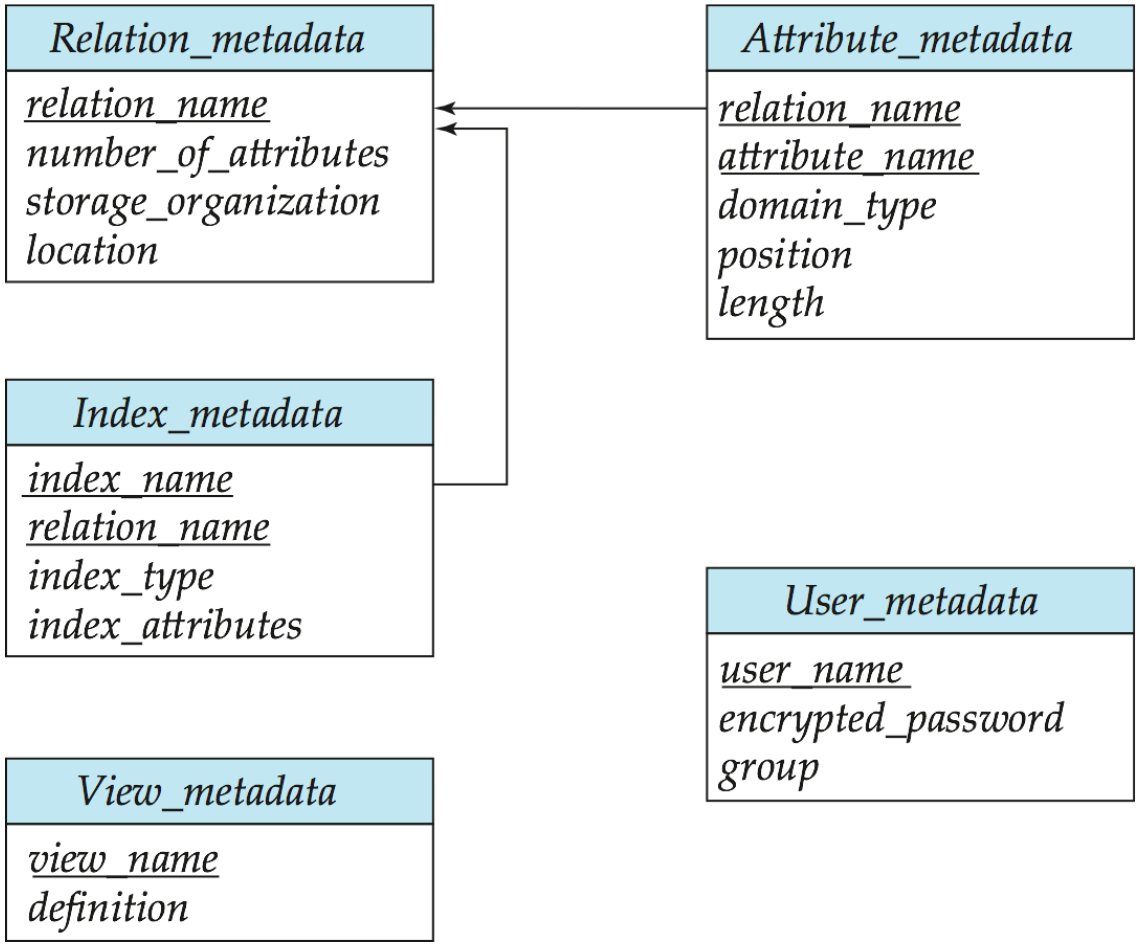
- A RDBMS needs to maintain data *about* the relations, such as the schema of the relations.
- In general, such "data about data" is referred to as metadata.

- Different Metadata:
  - Information about relations
    - names of relations
    - names, types and lengths of attributes of each relation
    - names and definitions of views
    - integrity constraints
  - User and accounting information, including passwords
  - Statistical and descriptive data
    - number of tuples in each relation
  - Physical file organization information
    - How relation is stored (sequential/hash/…)
    - Physical location of relation
  - Information about indices

Cont…

- The exact choice of how to represent system metadata by relations must be made by the system designers.

- One possible representation, with primary keys underlined
- Relational representation on disk
- Specialized data structures are used for efficient access in memory
- The data dictionary is often stored in a non-normalized form to achieve fast access.



Thanks!