Storing Data: Disks and Files

Roberto Marabini
EDAT

2017

Syllabus

- Implementation of Databases
  - Physical structure: records and registers
  - Indexes
    - Simple
    - B trees
    - Hashing

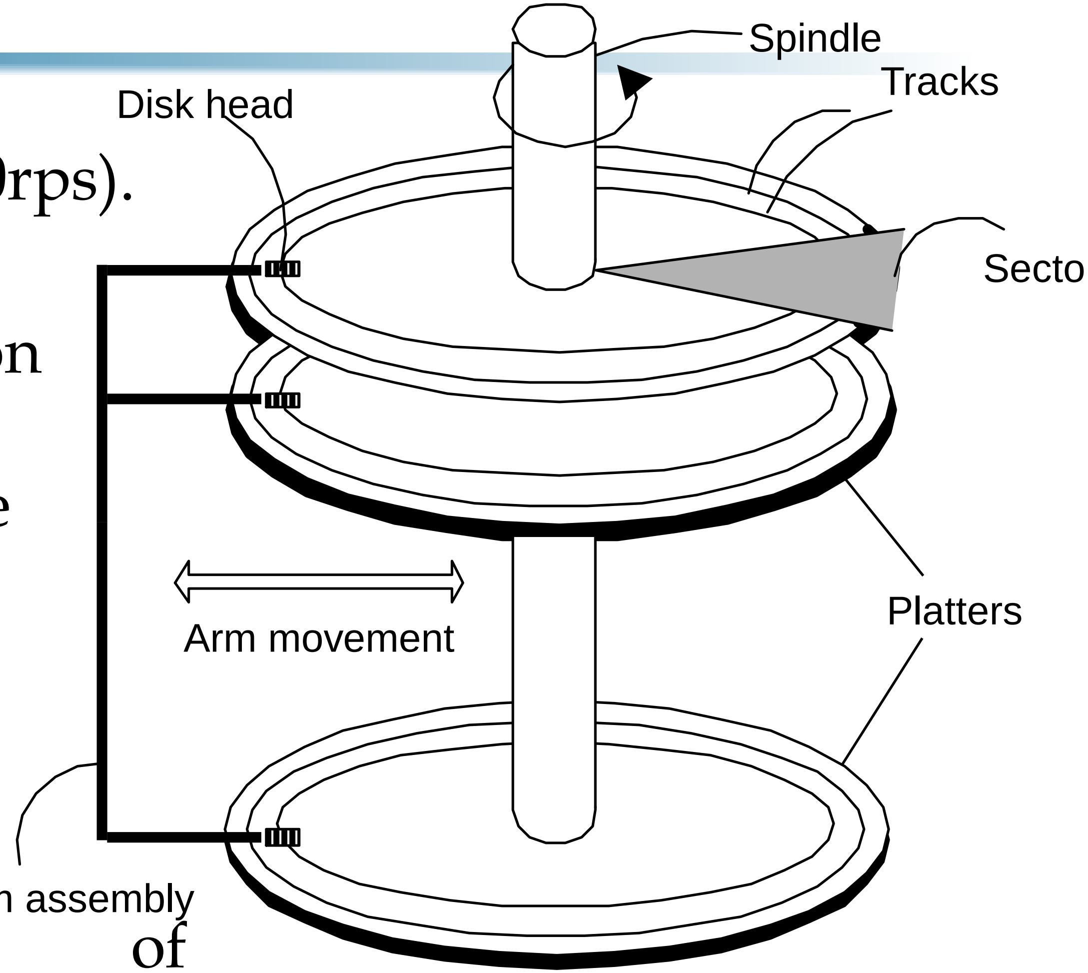
Disks and Files

- DBMS stores information on (“hard”) disks.
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are **high-cost operations**, relative to in-memory operations, **so must be planned carefully**!

Disks

- Secondary storage device of choice.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages.*

Components of a (magnetic) Disk

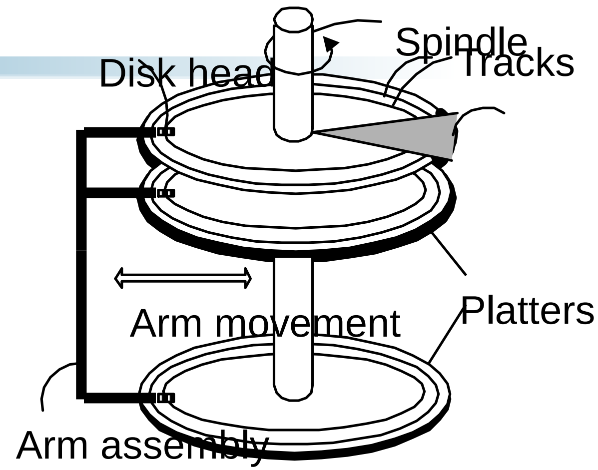


- The platters spin (say, 90rps).
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).
- Only one head reads/writes at any one time.
- *Block size* is a multiple of *sector size* (which is fixed).

Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track 1/3R)
  - *rotational delay aka rotational latency* (waiting for block to rotate under head 1/2)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: reduce seek/rotation delays!

Arranging Pages on Disk

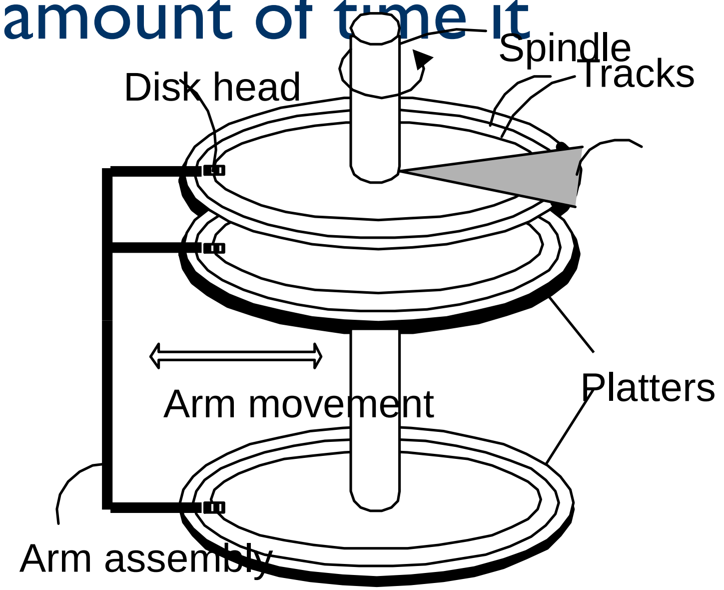


- `*Next*' block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Blocks in a file should be arranged sequentially on disk (by \`next'), to minimize seek and rotational delay.
- For a sequential scan, *pre-fetching* several pages at a time is a big win!

Example i

As an example let us compute the amount of time it will take to:

- read 1 MByte (10^6Bytes)
- from a 7200 RPM drive
- with a 8ms average seek time
- that has 500 sectors per track
- Each sector has 512 Bytes
- (assume all sectors are consecutive)



Example

- Number of sectors needed to store the file:
  - 1954
- Time needed to read one sector
  - 16.7 micros
- Reading 1M should take
  - 32.6 ms
- Total time including average seek and rotational latency:
  - 8ms + 0.5 *8.3 ms + (1964 * 16.7 micros) = 44.8 ms

Example II

As an example let us compute the amount of time it will take to:

- read 1 Mbyte (10^6bytes)
- from a 7200 RPM drive
- with a 8ms average seek time
- that has 500 sectors per track
- Each sector has 512 bytes
- (assume all sectors are at random places)

Example

- Number of sectors needed to store the file:
  - 1954
- Time needed to read one sector
  - 16.7 micros
- Reading 1M should take
  - 32.6 ms
- Total time including average seek and rotational latency:
  - 1954 * (8ms + 0.5 *8.3 ms + 16.7 micros) = 44.8 ms

Another Exercise

Imagine that we have a 2MB file with 8000 registers of 25&B each (fixed length). This file is stored in a magnetic hard disk with the following characteristics:

- 40 sectors per track
- 8 sectors per cluster
- 512 per sector
- Average seek time= 9.5 ms
- Spin speed 5400 rpm

How long will it take to read the whole file in the worst and best cases?

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**.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations
  - This case is easiest to implement; will consider variable length records later.

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 records (may cross blocks)
- Deletion of record i: alternatives:
  - move records i + 1, . . ., n to i, . . . , n – 1
  - move record n to i
  - 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

Deleting record 3 and moving last record (no easy access)

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

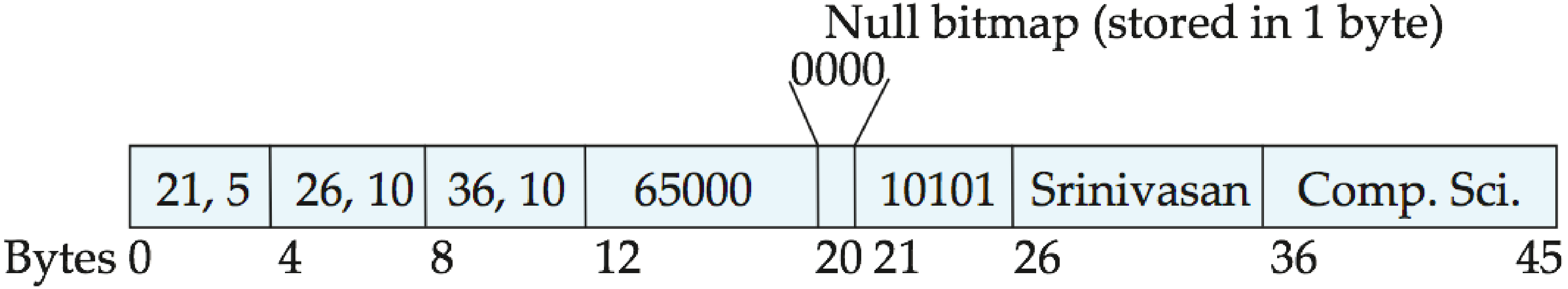
Free List

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they "point" to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)

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

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**)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes (many possible alternatives here, (1) only offset, (2) only length, (3) all info in the file header or a index file, delimiter character, …)



File Header

- Number of records
- Size of each record
- Fields per record
- Field names and size (if fixed size)
- Pointer to first record
- Pointer to list of "deleted" records
- How records are delimited
- Etc, time creation, modification,…

Insert Registers

- Usually make sense to keep the records ordered by a search-key (PK)
- From time to time no may reorganize the file but usually we relay in pointer to keep order

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	

Insert Registers II

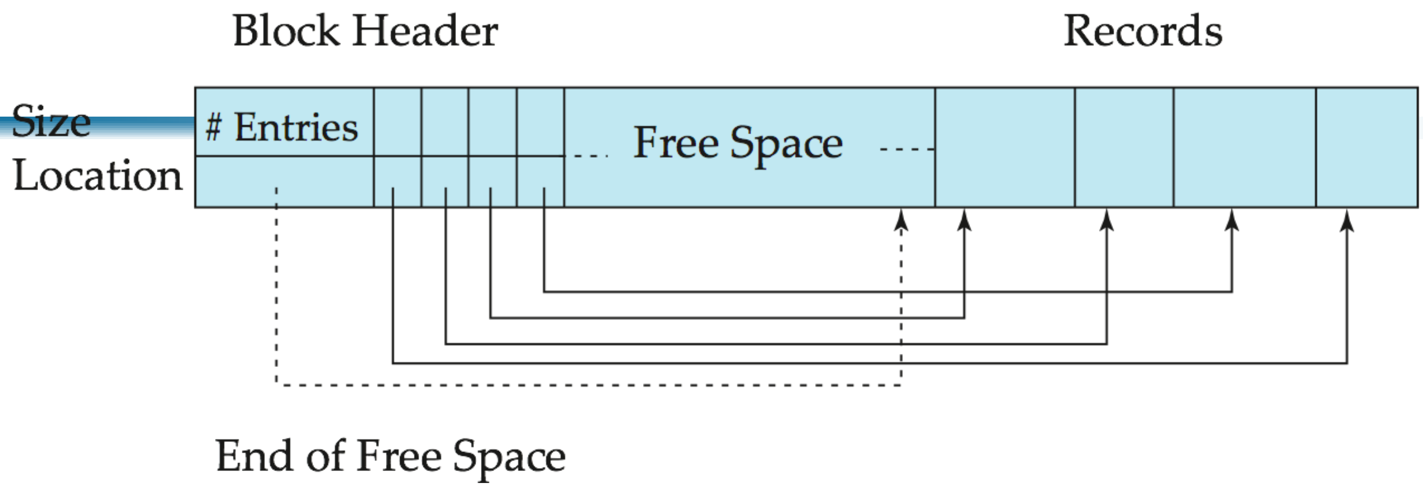
- Insertion –locate the position where the record is to be inserted
  - Locate free space (header has point to free space)
  - if no free space add overflow block
  - In either case, pointer chain must be updated
  - Fixed records are easier to handle than variable ones
- Deletion – use pointer chains
- Worst, best, first fit strategies

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	

32222	Verdi	Music	48000	

Indexes

Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record 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
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization

department

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

instructor

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

multitable clustering of *department* and *instructor*

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

Multitable Clustering File Organization (cont.)

- good for queries involving *department* $\bowtie$ *instructor*, and for queries involving one single department and its instructors
- bad for queries involving only *department*
- results in variable size records
- Can add pointer chains to link records of a

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

Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

- 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 (Chapter 11)