Chapter 10: Storage and File Structure

- Overview of Physical Storage Media
- Magnetic Disks
  - RAID
  - Tertiary Storage
- Storage Access
- File Organization
- Organization of Records in Files

DataBytes

Magnitude	Name	Abbreviation
$10^{24} \sim 2^{80}$	yotta	y,Y
$10^{21} \sim 2^{70}$	zetta	z,Z
$10^{18} \sim 2^{60}$	exa	e, E
$10^{15} \sim 2^{50}$	peta	p, P
$10^{12} \sim 2^{40}$	tera	t, T
$10^{9} \sim 2^{30}$	giga	g, G
$10^{6} \sim 2^{20}$	mega	m, M
$10^{3} \sim 2^{10}$	kilo	k, K
$10^{0} \sim 2^{0}$		
10^{-3}	mili	m
10^{-6}	micro	µ
10^{-9}	nano	n
10^{-12}	pico	p
10^{-15}	fempto	f

Big Data


THE WALL STREET JOURNAL. Wednesday, October 9, 2013 | 7

U.S. NEWS

Meltdowns Hobble NSA Data Center

Bigger Data

Biggest Data

190

© Nick Roussopoulos

Classification of Physical Storage Media

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- We differentiate storage into:
  - volatile storage:
    loses contents when power is switched off
  - non-volatile storage:
    Contents persist even when power is switched off.

191

© Nick Roussopoulos

Physical Storage Media

- **Cache**
  - fastest and most costly form of storage; volatile; managed by the computer system hardware.
  - Volatile — contents of main memory are usually lost

- **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 Gigabytes
    - Capacities have gone up and per-byte costs have decreased steadily roughly factor of 2 every 2 to 3 years
  - Volatile — content of main memory is lost on power failure

192

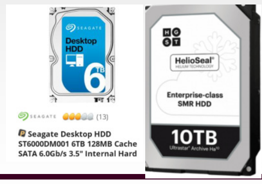
Physical Storage Media (Cont.)

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and re-written in large blocks
    - Can support only a limited number 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 (1-2 μS)
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices such as digital cameras
  - Copiers, printers, etc. use flash memory EEPROM (Electrically Erasable Programmable Read-Only Memory)
  - Used in Solid State drives
    - No moving parts

193

Physical Storage Media (Cont.)

- **Magnetic-disk**
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for processing and written back for storage
    - Much slower access than main memory
  - Direct-access – possible to read data on disk in any order
  - Capacities range up to 10 TB
    - Much larger capacity and cost/byte a lot less 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



SSD

- **Solid State Drives**
  - Details on SSD can be found in
    http://en.wikipedia.org/wiki/Solid-state_drive#Comparison_with_hard_disk_drives
  - NAND-based flash memory- non-volatile memory
  - Solid-state hybrid drives (SSHDs) combine the features of SSDs and HDDs (SSD as a cache to a hard drive)
  - War over SSD market
  - 2.5-inch solid state drive (SSD)
    - $10K

Samsung 16TB SSD is the World's Largest Hard Drive

Seagate's new 60TB SSD is world's largest

Seagate's 60TB SSD comes a year after Samsung's 15TB SSD.

- $30-40K?

Physical Storage Media (Cont.)

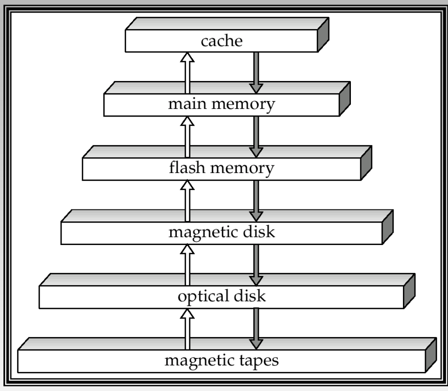
- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (700 MB) and DVD (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R and DVD-R)
  - Multiple write versions also available (CD-RW, DVD-RW, and DVD-RAM)
  - Reads and writes are much slower than those in magnetic disk
  - Juke-box systems- large numbers of removable disks

Physical Storage Media (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
    - Lots of TB or PB

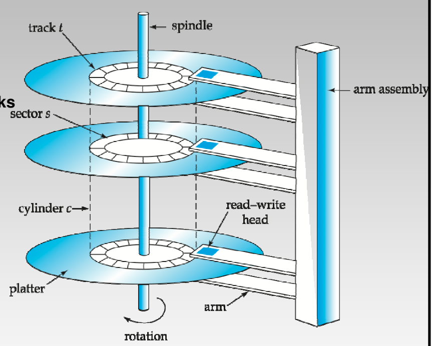
Storage Hierarchy

- primary storage: Fastest media but volatile (cache, main memory).
- secondary storage: next level in hierarchy, non-volatile, moderately fast access time
  - also called on-line storage
  - E.g. flash memory, magnetic disks
- tertiary storage: lowest level in hierarchy, non-volatile, slow access time
  - also called off-line storage
  - E.g. magnetic tape, optical storage



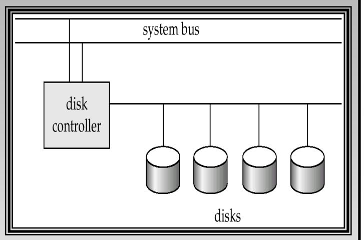
Magnetic Disks

- 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)
  - one head per platter, mounted on a common arm.
- Cylinder *i* consists of i^{th} track of all the platters



Magnetic Disks (Cont.)

- 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
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors



- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SATA (Serial ATA)
  - SCSI (Small Computer System Interconnect) range of standards
  - SAS (Serial Attached SCSI)
  - Several variants of each standard (different speeds and capabilities)

Disk Subsystem

- Disks usually connected directly to computer system
- 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 or 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.
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to appear under the head.
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
- **Data-transfer rate** – how fast data can be retrieved from or stored to the disk
  - 25-100 MB/s (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)
    - Ultra320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Performance Measures (Cont.)

- **Mean time to failure** (MTTF) – the average time the disk is expected to run continuously without any failure.
  - Typically 3 to 5 years
    - if you have 1500 disks then 1500/5=300 per year will fail

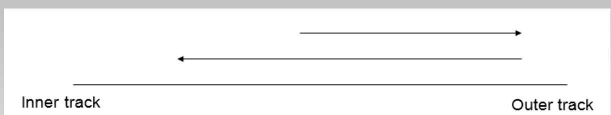
      (almost like a vitamin - one a day!)
  - Probability of failure of new disks is quite low
  - "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours means that given 1000 relatively new disks, on an average one will fail every 1200 hours or every 50 days
  - MTTF decreases as disk ages

Optimization of Disk-Block Access

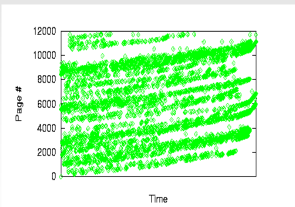
- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - block sizes range from 512 bytes to 16KB (typical 4-16KB)
  - Smaller blocks: more block transfers from disk
  - Larger blocks: more space wasted due to partially filled blocks
  - Read-ahead brings adjacent blocks
- **Access time of multiple blocks**
  - Random access= (seek time + rotational latency) * Number of blocks + TOTAL BYTES / transfer rate
  - Access contiguous blocks = seek time + rotational latency + TOTAL BYTES / transfer rate
  - Sequential 1-2 orders of magnitude faster than random

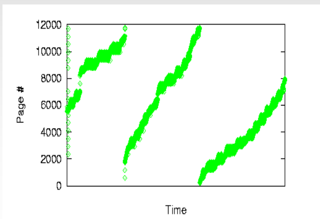
Optimization of Disk-Block Access

- **Disk-arm-scheduling algorithms** order pending accesses to tracks so that disk arm movement is minimized



elevator algorithm: move disk arm in one direction





Optimization of Disk-Block Access

- Log-based file system (LFS): does not update in-place but logs the writes to a sequential disk (achieving the sequential speeds)
- Clustering of data: organize it to match to the access
  - if hierarchical access, then put the daughters next to the mothers
  - for joining tables, put the joining tuples from the two tables next to each other
- Non-Volatile write buffering: speeds disk writes
  - Battery backed RAM or flash memory
  - Writes can be reordered to minimize disk arm movement (seek time)
- Log disk – a disk devoted to writing a sequential log of block updates
  - Used exactly like non-volatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)

206

Flash Storage

- NOR flash vs NAND flash
  - NOR is more flexible- can do random access of words (flexible) and
  - very fast (almost like RAM)
  - expensive
- NAND flash
  - used widely for storage, much cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
  - transfer rate around 20 MB/sec
  - High-end can get to 100 MB/sec
- Solid State Disks: use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec
  - erase is very slow (1 to 2 millisecs)
  - erase block contains multiple pages
  - remapping of logical page addresses to physical page addresses avoids waiting for erase
  - after 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used

RAID

- **RAID:** Redundant Arrays of Independent Disks
  - Originally the letter "I" stood for *inexpensive*
  - Manage a numbers of disks as a single disk
- Benefits:
  - high capacity and high speed by exploiting parallelism
  - high reliability by storing data redundantly
    - data can be recovered when a disk fails
- The chance that some disk (out of *N)* will fail is much higher than the chance of a single disk will
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (~ 11 years), will have a system MTTF of 1000 hours (approx. 42 days)
- The chance of two disks failing at the same time is near zero
- Redundancy to avoid data loss is critical with large numbers of disks

Improvement of Reliability using Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or shadowing)
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take place from either disk
      e.g. round robin, probabilistically
  - If one disk fails, data still available in the other
    - Data loss would occur only if a disk fails, and its mirror disk also fails before the first is repaired
- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
  - E.g. 2 disks with MTTF 100,000 hours
  - mean time to repair of 10 hours gives
  - mean time to data loss of $100{,}000^2/(2*10)=500*10^6$ hours (57,000 years)
    (calculation ignores dependent failures)

Improving Performance via Parallelism

- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Spread high volume of accesses across multiple disks to reduce response time

- Improve transfer rate by striping data across multiple disks.

- Bit-level striping – split the bits of each byte across multiple disks
  - In an array of eight disks, write bit *i* of each byte to disk *i.*
  - Each access can read data at eight times the rate of a single disk.
  - But seek/access time worse than for a single disk
  - Bit level striping is not used any more

- Block-level striping – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1
  - Blocks transfers can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel

RAID Levels

- RAID Level 0: Block striping; non-redundant

(a) RAID 0: nonredundant striping

  - Used in high-performance applications where data loss is not critical.

- RAID Level 1: Mirrored disks with block striping

(b) RAID 1: mirrored disks

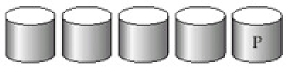
  - Popular for applications such as storing log records in a database system.

RAID Levels (Cont.)

- **RAID Level 2: Error-Correcting-Codes (ECC) with bit striping**

(c) RAID 2: memory-style error-correcting codes

- **RAID Level 3: Bit-Interleaved Parity**
  - **A single parity bit is enough for error correction**
    - **Every read has to access all disks and every write has to write the parity disk**
    - **To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)**

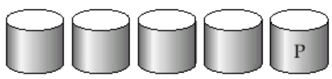


(d) RAID 3: bit-interleaved parity

Bit striping- too low level and expensive

RAID Levels (Cont.)

- **RAID Level 4: Block-Interleaved Parity**
  - **uses block-level striping, and**
  - **parity block on a separate disk**
- **Higher I/O rates for independent block reads than Level 3**
  - **a block read goes to a single disk**
  - **blocks stored on different disks can be read in parallel**
- **High transfer rates for reads of multiple blocks than no-striping**



(e) RAID 4: block-interleaved parity

RAID Level 4 (Cont.)

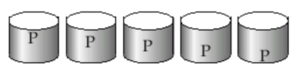
- Before updating a block B to B’, its parity block P’ must be computed
  - read block B
  - read old parity block P
  - compute P’= B $\oplus$ B’ $\oplus$ P
  - write new value B’
  - write new parity P’
  - 2 block reads + 2 block writes
- When a block is damaged, we computed it from the other blocks and the parity one (see below in level 5)
- Parity block becomes a bottleneck since every block write also writes to parity disk

214

RAID Levels (Cont.)

- **RAID Level 5: Block-Interleaved Distributed Parity**; partitions data and parity among all *N* + 1 disks, rather than storing actual data in *N* disks and parity data in 1 disk.
  - E.g., with 5 disks, parity block for *n*th blocks is stored on disk (*n mod* 5), other data blocks stored on the other 4 disks.
  - Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4



(f) RAID 5: block-interleaved distributed parity

215

RAID Levels 5 (Cont.)

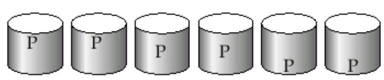
Data block reads and writes done in parallel

- Read block 5 (1 read)
- Read, Modify, and Write a block 5
  - Read 5 & P1 (2 reads)
  - Compute P1'= 5 ⊕ 5' ⊕ P1
  - Write 5' & P1' (2 writes)
- Suppose disk 3 that stores 5 fails
  - To read 5
    - Read 4, P1, 6, 7 (4 reads)
    - Compute 5 = 4 ⊕ P1 ⊕ 6 ⊕ 7
  - To write 5'
    - Read 4, P1, 6, 7 (4 reads)
    - Compute 5 = 4 ⊕ P1 ⊕ 6 ⊕ 7
    - Compute P1'= 5' ⊕ 5 ⊕ P1
    - Write P1' (1 write - less writes!)

D1	D2	D3	D4	D5
P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Levels (Cont.)

- RAID Level 6: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.



(g) RAID 6: P + Q redundancy

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g., data can be recovered quickly from other sources
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for almost all applications
- So competition is between 1 and 5 only

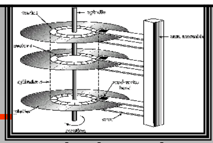
218

Choice of RAID Level (Cont.)

- Level 1 provides better write performance than level 5
  - Level 5 requires at least 2 block reads (1 extra read) and 2 block writes for each block write, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks (mostly blind writes)
- Level 1 has higher storage cost than level 5
  - mirroring is more expensive
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required I/O demand, they often have spare storage capacity
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

219

Storage Access



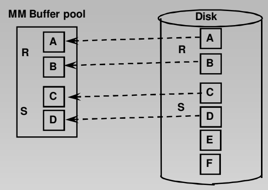
- block-based: a block is a contiguous sequence of sectors from a single track. Blocks are units of both storage allocation and data transfer.
- a file is a sequence of records stored in fixed size blocks (pages) on the disk
- each block (page) has a unique address called BID
- optimization is done by reducing I/O, seek time, etc.
- The DBMS minimizes the number of block transfers between disks and memory
  - reduction is achieved by keeping as many blocks as possible in main memory and
  - smarter query execution algorithms
- Buffer – portion of main memory available to temporarily store copies of disk blocks.
- Buffer manager – subsystem responsible for allocating buffer space in main memory and efficiently manage it

Buffer Management

- the buffer pool is the part of the main memory allocated for
  - temporarily storing disk blocks read from disk and
  - making these blocks available to the query processor
  - its purpose is identical to caching for reducing I/O
- the buffer manager is transparent to the users
- when a process requests a block (page) the buffer mgr takes the following steps:
  - checks if the page is in the buffer pool
  - if it is, it passes its address to the process
  - if it is not, it brings it from the disk and then passes its address to the process
- very similar to the *virtual memory managers,* although it does a lot better

Buffer Replacement Strategies

```
for each  block br  in  r  do begin
   for each block  bs  in s  do begin
       join(br ,bs)
   end
end
```



R: outer
S: inner

- LRU, FIFO, etc. used in OS do not perform well in DBMSs- MRU is better for some operations

LRU: A C D E F B C D E F Total: 10

MRU: A C D E F B E F Total: 8

MRU: A C D E F B E Total: 7

- Some times we toss immediately a block if we know we do not need it again
- Or we fasten (or pin) some blocks to keep them during the operation
- Prevents the replacement strategy to touch them until released
- These are called ***fastened or pinned*** blocks

Buffer-Replacement Policies (Cont.)

- Pinned block – memory buffer block that is not allowed to be replaced
- Toss-immediate strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- In MRU strategy the 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. Heuristic: keep data-dictionary blocks in main memory buffer

Buffer Management (cont)

- OS affects DBMSs operations by:
  - read ahead, write behind
  - wrong replacement strategies
  - running a DBMS on top of Unix is bad
  - most commercial systems implement their own I/O on a raw disk partition
- Variations of buffer allocation
  - common buffer pool for all relations
  - separate -"- -"- each relation
  - as above but with relations borrowing from each other
  - adaptive allocation based on their needs
  - prioritized buffers for very frequently accessed blocks, e.g. data dictionary
- for each buffer the manager keeps the following
  - which disk and which block it is
  - whether it has modified or not (*dirty bit*)
  - information for the replacement strategy (e.g. the time it was last accessed)

224

Access Methods

- take care of the following:
  1. allocate records (tuples) within blocks
  2. support record addressing by address and by value
  3. support auxiliary (secondary indexing) file structures for more efficient accessing

225

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.

- how do we organize a file into blocks and records
  - formatting fields within a record
  - -"   records within a block
  - assigning records to blocks

F1	F2	F3	F4
←L1→	←L2→	←L3→	←L4→

b0: base address

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
    - Modification: do not allow records to cross block boundaries

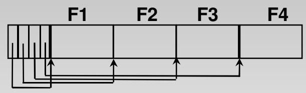
- Deletion of record *I:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

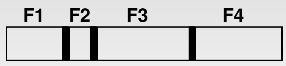
Fixed & Variable Length Records

Formatting fields within a record

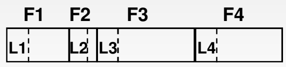
- fixed length fields
  - address of Fi = b0 + L1+..+Li-1
- fixed or variable length fields stored in an slotted page
  - need not be stored in order
  - header position is always present even if the field is not



- variable length
  - with delimiters between the fields

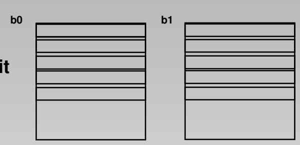


  - with the length in front of each field

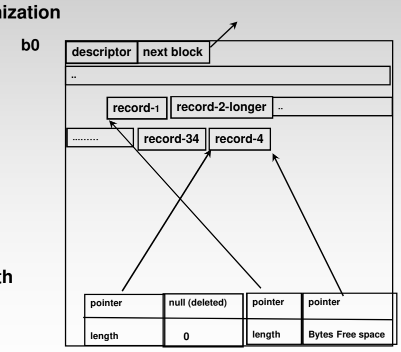


Organize Records in a Block

- Packed: records are stored contiguously
  - blocks are referenced by their block id, b0, b1, …
  - records are referenced by their block id and position in it
  - highly inflexible
  - records may span over the block's boundary
  - fragmentation with deletions and insertions
  - external pointer prevent internal block reorganization
    records are ***pinned*** to their address



- Slotted page structure (indexed heap):
  - block address and offset is used to reference the record
    TID = b0 + blocksize - offset
  - records are indexed within a block
  - insertions and deletions are easy
  - records can be rearranged within the block with no problems with external pointers
  - records are ***unpinned*** with the block



© Nick Roussopoulos

Column-oriented Storage

- Transpose rows and columns

```
1,Smith,Joe,40000;
2,Jones,Mary,50000;
3,Johnson,Cathy,44000;
```

➔

```
1,2,3;
Smith,Jones,Johnson;
Joe,Mary,Cathy;
40000,50000,44000;
```

- Benefit: High level of compression
  - Column values are similar (compression uses run lengths)
  - This can further increase if relation is sorted
  - Layout the data so that the columns in increasing cardinality of distinct values
- Row-oriented architectures are well-suited for OLTP
  - Lots of interactive transactions
- Column stores are well-suited for OLAP and Data warehouses
  - Computing lots of aggregates, statistics, and data mining

230

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 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

231

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

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Sequential File Organization (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

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

A-888	North Town	800	

Clustering File Organization

- Simple file structure stores each relation in a separate file
- Store several relations in one file using a clustering **file organization**
- E.g., clustering organization of ***customer*** and ***depositor:***

  - good for queries involving: customer $\bowtie$ depositor
  - Also for queries involving one single customer and his accounts
  - Bad for queries involving only customer
  - Results in variable size records

Data Dictionary Storage

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

- **Information about relations**
  - names of relations
  - names and types 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
    - operating system file name or
    - disk addresses of blocks containing records of the relation
- **Information about indices**

Data Dictionary Storage (Cont.)

- Catalog structure: can use either
  - specialized data structures designed for efficient access
  - a meta-database schema (a set of relations), together with system features to ensure efficient access (preferred)

- A possible catalog representation: