Physical Data Organization

CPS 216

Advanced Database Systems

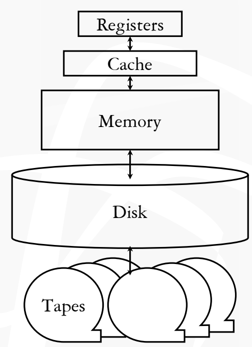
Announcements (January 27)

- Reading assignment for next week
  - System R paper and Lomet's B^+-tree tricks
  - Due next Thursday night
- Homework #1 due in 12 days

Outline

- It's all about disks!
  - That's why we always draw databases as 
  - And why the single most important metric in database processing is the number of disk I/O's performed
- Record layout
- Block layout

Storage hierarchy



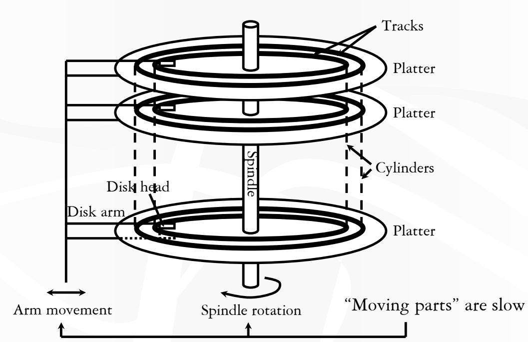
How far away is data?

Location	Cycles	Location	Time
Registers	1	My head	1 min.
On-chip cache	2	This room	2 min.
On-board cache	10	Duke campus	10 min.
Memory	100	Washington D.C.	1.5 hr.
Disk	10^6	Pluto	2 yr.
Tape	10^9	Andromeda	2000 yr.

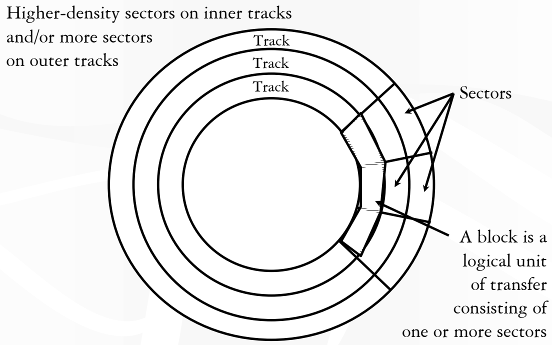
(Source: AlphaSort paper, 1995)

☞ I/O dominates—design your algorithms to reduce I/O!

A typical disk



Top view



Disk access time

Sum of:

- ❖ Seek time: time for disk heads to move to the correct cylinder
- ❖ Rotational delay: time for the desired block to rotate under the disk head
- ❖ Transfer time: time to read/write data in the block (= time for disk to rotate over the block)

Random disk access

Seek time + rotational delay + transfer time

- ❖ Average seek time
  - Time to skip one half of the cylinders?
  - Not quite; should be time to skip a third of them (why?)
  - "Typical" value: 5 ms
- ❖ Average rotational delay
  - Time for a half rotation (a function of RPM)
  - "Typical" value: 4.2 ms (7200 RPM)
- ❖ How do you calculate transfer time (function of transfer size)?

Sequential disk access

Seek time + rotational delay + transfer time

- ❖ Seek time
  - 0 (assuming data is on the same track)
- ❖ Rotational delay
  - 0 (assuming data is in the next block on the track)
- ❖ Easily an order of magnitude faster than random disk access!

Performance tricks

- ❖ Disk layout strategy
  - Keep related things (what are they?) close together: same sector/block → same track → same cylinder → adjacent cylinder
- ❖ Double buffering
  - While processing the current block in memory, prefetch the next block from disk (overlap I/O with processing)
- ❖ Disk scheduling algorithm
  - Example: "elevator" algorithm
- ❖ Track buffer
  - Read/write one entire track at a time
- ❖ Parallel I/O
  - More disk heads working at the same time

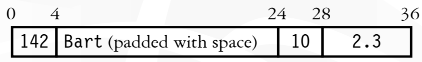
Record layout

Record = row in a table

- ❖ Variable-format records
  - Number and types of fields not known in advance
  - Rare in DBMS—table schema dictates the format
  - Relevant for semi-structured data such as XML
- ❖ Focus on fixed-format records
  - With fixed-length fields only, or
  - With possible variable-length fields

Fixed-length fields

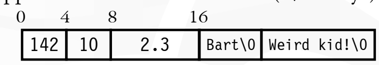
- All field lengths and offsets are constant
  - Can be pre-computed from schema
- Example: `CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT);`



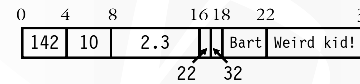
- Watch out for alignment
  - May need to pad; reorder columns if that helps
- What about `NULL`?
  - Add a bitmap at the beginning of the record

Variable-length records

- Example: `CREATE TABLE Student (SID INT, name VARCHAR(20), age INT, GPA FLOAT, comment VARCHAR(100));`
- Approach 1: use field delimiters ("\0" okay?)



- Approach 2: use an offset array



- Put all variable-length fields at the end (why?)
- Update is messy if it changes the length of a field

Record layout in commercial systems

- DB2, SQL Server, Informix, Sybase: all variants of the offset array approach
  - DB2: in the fixed-length part of the record, store (offset, length) for a variable-length field, where offset points to the start of the field in the variable-length part of the record; no need to reorder fields
- Oracle: records are structured as if all fields are potentially of variable length
  - A record is a sequence of (length, data) pairs, with a special length value denoting `NULL`

LOB fields

- Example: `CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT, picture BLOB(32000));`
- Student records get "de-clustered"
  - Bad because most queries do not involve `picture`
- Store LOB's in a difference place (automatically done by DBMS and transparent to the user)
  - Conceptually, the table is decomposed into
    - *Student(<u>SID</u>, name, age, GPA, picture_id)*
    - *Picture(<u>picture_id</u>, picture)*
  - ☞Like System R Phase 0's XRM storage manager

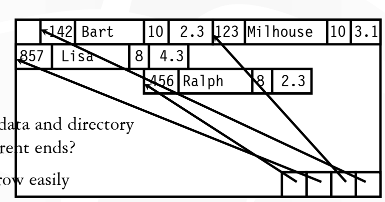
Block layout

How do you organize records in a block?

- NSM (N-ary Storage Model)
  - Most commercial DBMS
- PAX (Partition Attributes Across)
  - Research work (Ailamaki et al., *VLDB* 2001)

NSM

- Store records from the beginning of each block
- Use a slot directory at the end of each block
  - To locate records and manage free space
  - Necessary for variable-length records



Why store data and directory at two different ends?

Both can grow easily

19

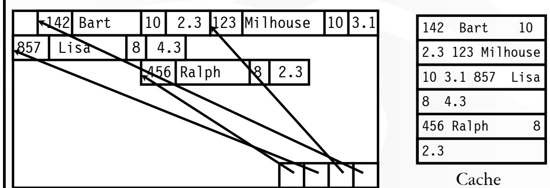
Options

- Reorganize after every update/delete to avoid fragmentation (gaps between records)
  - Need to rewrite half of the block on average
- What if records are fixed-length?
  - Reorganize after delete
    - Only need to move one record
    - In slot directory, keep a pointer to the beginning of free space
  - Do not reorganize after update
    - In slot directory, keep a bitmap showing which slots are in use

20

Cache behavior of NSM

- Query: `SELECT SID FROM Student WHERE GPA > 2.0;`
- Say cache block size < record size
- Lots of cache misses
  - `ID` and `GPA` are not close enough by memory standard



21

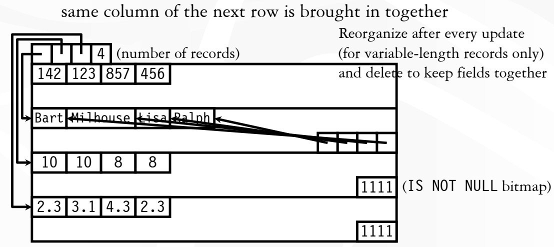
Do caches misses matter in DBMS?

- No? Compared to disk I/O's, memory-related stall time is nothing
- Yes?
  - You may mask some I/O cost
  - You may avoid some I/O's by memory buffering
  - Percentage of memory-related stall time due to data cache misses is high
    - 90% for OLAP workloads (lots of large, complex, range-based queries, few updates)
    - 50-70% for OLTP workloads (lots of small, simple, point-based queries and updates)

22

PAX

- Most queries only access a few columns
- Cluster same columns in "minipages" in each block
  - When a particular column of a row is brought into the cache, the same column of the next row is brought in together



23

PAX versus NSM

- Space requirement: roughly the same
- Cache performance: PAX incurs 75% less data cache misses than NSM
- Overall performance
  - For OLAP queries (TPC-H), PAX is 11-48% faster
  - For updates, PAX is 10-16% faster (assuming NSM also reorganizes)
  - Unanswered question: How about OLTP queries/updates (typically very selective)?
- ☞ Adaptive hybrid of PAX and NSM
  - Hankins and Patel. "Data Morphing…" *VLDB* 2003
- ☞ Dynamic adjustment of layout when fetching fro
  - Shao et al. "Clotho: Decoupling…" *VLDB* 2004

24

"Pointers" to records

- Logical record id: value of the primary key
  - Used in references (e.g., *Enroll*(*SID*, *CID*))
- Physical record id: (disk block id, slot number)
  - Used in index entries: (key, physical record id)
- Pros and cons
  - Physical id is faster
    - Disk block id directly gives exact location of record on disk; while given the primary key value, we need to go through the primary index
    - Primary key value might be huge (in terms of size in bytes)
  - Some tables do not declare primary key
    - Can invent a surrogate key
  - Logical record id is more informative
    - May save an access to the actual record
  - Physical id must be maintained when record moves around on disk

Record pointers in commercial systems

- At user/SQL level, logical record id is the only option (why?)
- Internally, virtually all commercial systems use physical record id
  - Except Oracle and SQL Server, who use primary key as record id if one exists

Summary

- Storage hierarchy
  - Why I/O's dominate the cost of database operations
- Disk
  - Steps in completing a disk access
  - Sequential versus random accesses
- Record layout
  - Handling variable-length fields
  - Handling `NULL`
  - Handling modifications
- Block layout
  - NSM versus PAX
- Logical versus physical record ids

Next: more SQL; then indexing