Lecture 19: DMBS Overview and Data Storage

Monday, November 6, 2000

Outline

- DBMS Overview (1.1, 1.2)
- Data Storage (2)

A Naïve Database System

- Store data in text files
  - Schema:
    Students(sid, name, dept), Courses(cid, name), Takes(sid,cid)
  - Schema file:
    Students #sid#INT #name#STR#dept#STR
    Courses#cid#INT#name#STR
    Takes#sid#INT#cid#INT
  - Data file:
    Smith#123#CSE
    John#456#EE
    …

A Naïve DBMS

- Query processing
  SELECT Students.name
  FROM Students, Takes, Courses
  WHERE Students.sid=Takes.sid AND Takes.cid=Courses.cid
  AND Courses.name='Databases'
- Execution:
  - Read/parse query
  - Read schema file to determine attributes
  - Execute as nested loops
  - Print results

What is Wrong with the Naïve DBMS

- Tuple layout is rigid: what do we do on updates ?
- Search is expensive: always read the entire relation
- Query processing is by "brute force": more clever ways exists

What is Wrong with the Naïve DBMS

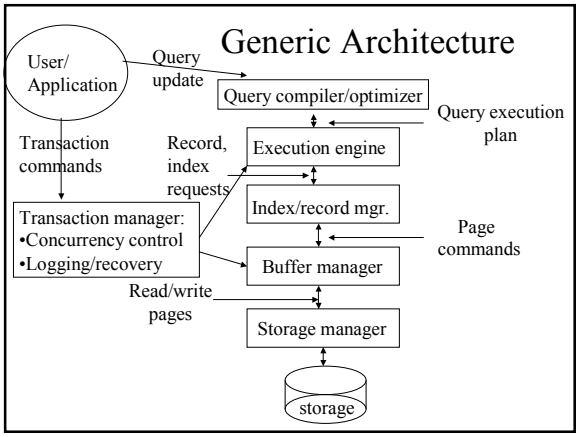
- No way to buffer data in memory
- No concurrency control
- No reliability: we can lose data in a crash
- No security

What Should a DBMS Do?

- Store large amounts of data
- Process queries efficiently
- Allow multiple users to access the database concurrently and safely.
- Provide durability of the data.

- How will we do all this??

Generic Architecture

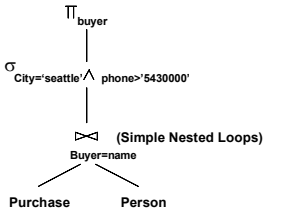


Query Optimization

Goal:

Declarative SQL query → ***Imperative query execution plan:***

```
SELECT  Q.sname
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
      Q.city='seattle' AND
    Q.phone > '5430000'
```

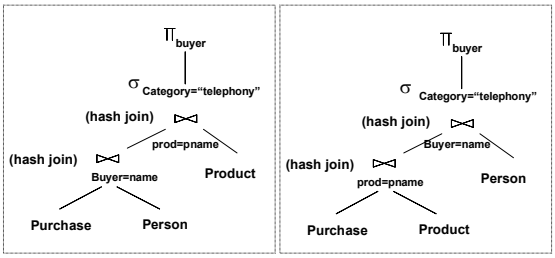


Plan: *Tree of R.A. ops, with choice of alg for each op.*

Ideally: Want to find best plan. Practically: Avoid worst plans!

Alternate Plans

Find names of people who bought telephony products



But what if we're only looking for Bob's purchases?

ACID Properties

*A*tomicity: **all** actions of a transaction happen, or **none** happen.

*C*onsistency: if a transaction is consistent, and the database starts from a consistent state, then it will end in a consistent state.

*I*solation: the execution of one transaction is isolated from other transactions.

*D*urability: if a transaction commits, its effects persist in the database.

Problems with Transaction Processing

Airline reservation system:
Step 1: check if a seat is empty.
Step 2: reserve the seat.

Bad scenario: (but very common)
Customer 1 - finds a seat empty
Customer 2 - finds the same seat empty
Customer 1 - reserves the seat.
Customer 2 - reserves the seat.

Customer 1 will not be happy; spends night in airport hotel.

The Memory Hierarchy

Main Memory

- Volatile
- limited address spaces
- expensive
- **average access time: 10-100 nanoseconds**

Cache: access time 10 nano's

Disk

- 5-10 MB/S transmission rates
- 2-10 GB storage
- **average time to access a block: 10-15 msecs.**
- Need to consider seek, rotation, transfer times.
- Keep records "close" to each other.

Tape

- 1.5 MB/S transfer rate
- 280 GB typical capacity
- **Only sequential access**
- Not for operational data

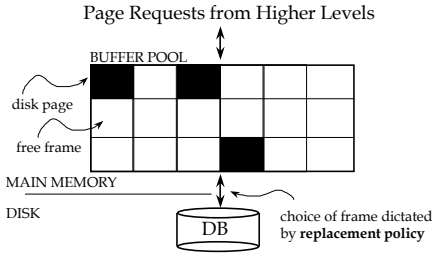
Main Memory

- Fastest, most expensive
- Today: 256MB are common even on PCs
- Many databases could fit in memory
  - New industry trend: Main Memory Database
  - E.g TimesTen
- Main issue is volatility

Secondary Storage

- Disks
- Slower, cheaper than main memory
- Persistent !!!
- Used with a main memory buffer

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *Table of <frame#, pageid> pairs is maintained.*
- *LRU is not always good.*

Buffer Manager

Manages buffer pool: the pool provides space for a limited number of pages from disk.

Needs to decide on page replacement policy.

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Why not use the Operating System for the task??

- DBMS may be able to anticipate access patterns
- Hence, may also be able to perform prefetching
- DBMS needs the ability to force pages to disk.

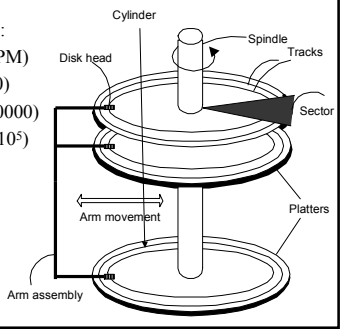
Tertiary Storage

- Tapes or optical disks
- Extremely slow: used for long term archiving only

The Mechanics of Disk

Mechanical characteristics:

- Rotation speed (5400RPM)
- Number of platers (1-30)
- Number of tracks (<=10000)
- Number of bytes/track(10^5)



Disk Access Characteristics

- Disk latency = time between when command is issued and when data is in memory
- Disk latency = seek time + rotational latency
  - Seek time = time for the head to reach cylinder
    - 10ms – 40ms
  - Rotational latency = time for the sector to rotate
    - Rotation time = 10ms
    - Average latency = 10ms/2
- Transfer time = typically 10MB/s
- Disks read/write one block at a time (typically 4kB)

The I/O Model of Computation

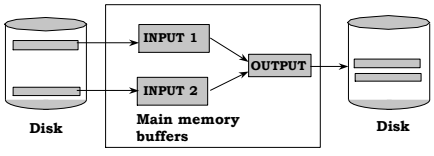
- In main memory algorithms we care about CPU time
- In databases time is dominated by I/O cost
- Assumption: cost is given only by I/O
- Consequence: need to redesign certain algorithms
- Will illustrate here with sorting

Sorting

- Illustrates the difference in algorithm design when your data is not in main memory:
  - Problem: sort 1Gb of data with 1Mb of RAM.
- Arises in many places in database systems:
  - Data requested in sorted order (ORDER BY)
  - Needed for grouping operations
  - First step in sort-merge join algorithm
  - Duplicate removal
  - Bulk loading of B+-tree indexes.

2-Way Merge-sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 2, 3, …, etc.:
  - three buffer pages used.



Two-Way External Merge Sort

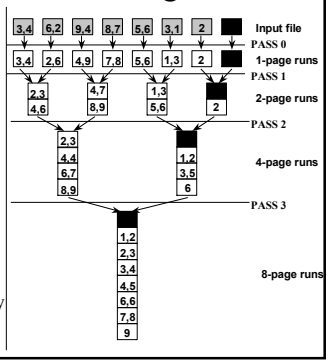
- Each pass we read + write each page in file.
- N pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

- So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

- Improvement: start with larger runs
- Sort 1GB with 1MB memory in 10 passes



Can We Do Better ?

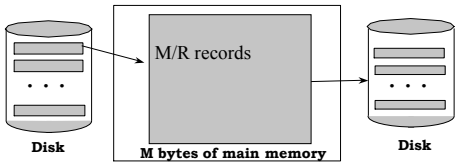
- We have more main memory
- Should use it to improve performance

Cost Model for Our Analysis

- **B:** Block size
- **M:** Size of main memory
- **N:** Number of records in the file
- **R:** Size of one record

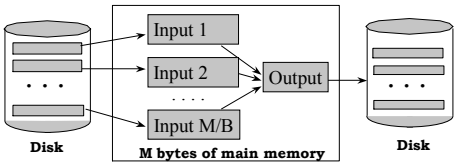
External Merge-Sort

- Phase one: load M bytes in memory, sort
  - Result: runs of length M/R records



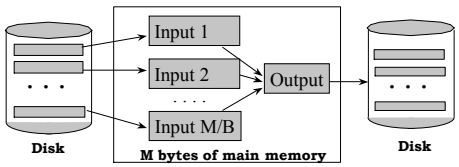
Phase Two

- Merge M/B – 1 runs into a new run
- Result: runs have now M/R (M/B – 1) records



Phase Three

- Merge M/B – 1 runs into a new run
- Result: runs have now M/R $(M/B – 1)^2$ records



Cost of External Merge Sort

- Number of passes: $1+\lceil \log_{M/B-1} \lceil NR/M \rceil \rceil$
- Think differently
  - Given B = 4KB, M = 64MB, R = 0.1KB
  - Pass 1: runs of length M/R = 640000
    - Have now sorted runs of 640000 records
  - Pass 2: runs increase by a factor of M/B – 1 = 16000
    - Have now sorted runs of 10,240,000,000 = 10^{10} records
  - Pass 3: runs increase by a factor of M/B – 1 = 16000
    - Have now sorted runs of 10^{14} records
    - Nobody has so much data !
- Can sort everything in 2 or 3 passes !