

Physical Data Organization

CPS 116

Introduction to Database Systems


Announcements (November 2)

2

- ❖ Deadline for Problem X1 of Homework #3 is today
- ❖ Project milestone #2 due in one week
 - You should be working with “production” dataset now

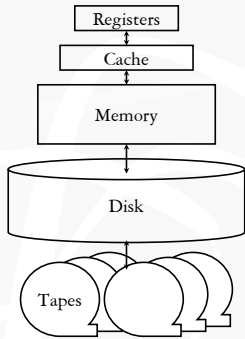
Outline

3

- ❖ 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
- ❖ Storing data on a disk
 - Record layout
 - Block layout

Storage hierarchy

4



Why a hierarchy?

How far away is data?

5

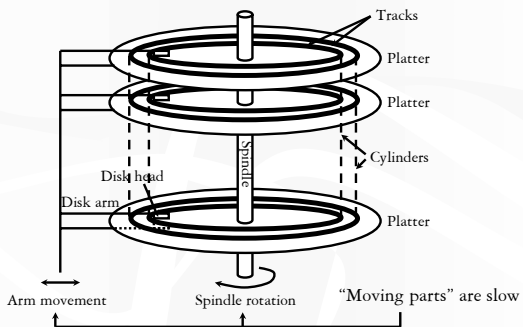
| Location | Cycles | Location | Time |
|----------------|--------|----------|------|
| Registers | 1 | | |
| On-chip cache | 2 | | |
| On-board cache | 10 | | |
| Memory | 100 | | |
| Disk | 10^6 | | |
| Tape | 10^9 | | |

(Source: AlphaSort paper, 1995)

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

A typical disk

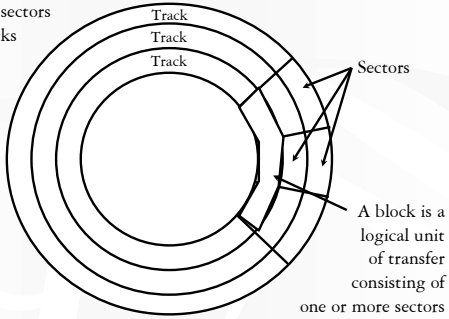
6



Top view

7

Higher-density sectors on inner tracks
and/or more sectors
on outer tracks



Disk access time

8

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

9

Seek time + rotational delay + transfer time

- ❖ Average seek time
 - Time to skip one half of the cylinders?
 -
 - "Typical" value: 5 ms
- ❖ Average rotational delay
 - Time for a half rotation (a function of RPM)
 - "Typical" value: 4.2 ms (7200 RPM)

Sequential disk access

10

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

11

- ❖ 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

12

Record = row in a table

- ❖ Variable-format records
 - 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
 - Computed from schema, stored in the system catalog
- ❖ Example: CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT);

| | | | | |
|-----|--------------------------|----|-----|----|
| 0 | 4 | 24 | 28 | 36 |
| 142 | Bart (padded with space) | 10 | 2.3 | |
- ❖ Watch out for alignment
 - May need to pad; reorder columns if that helps
- ❖ What about NULL?

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?)

| | | | | | |
|-----|----|-----|--------|--------------|--|
| 0 | 4 | 8 | 16 | | |
| 142 | 10 | 2.3 | Bart\0 | Weird kid!\0 | |
- ❖ Approach 2: use an offset array

| | | | | | | |
|-----|----|-----|------|------------|----|----|
| 0 | 4 | 8 | 16 | 18 | 22 | 32 |
| 142 | 10 | 2.3 | Bart | Weird kid! | | |
| | | | 22 | 32 | | |
- ❖ Put all variable-length fields at the end (why?)
- ❖ Update is messy if it changes the length of a field

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
- ❖ Decomposition (automatically done by DBMS and transparent to the user)
 - Student(SID, name, age, GPA)
 - StudentPicture(SID, picture)

Block layout

16

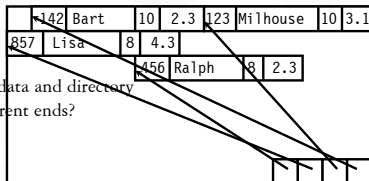
How do you organize records in a block?

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

NSM

17

- ❖ Store records from the beginning of each block
- ❖ Use a 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?

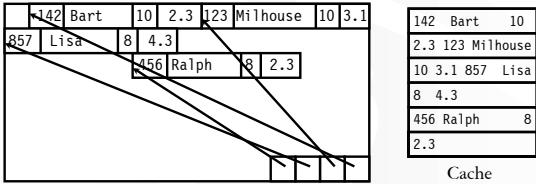
Options

18

- ❖ 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?

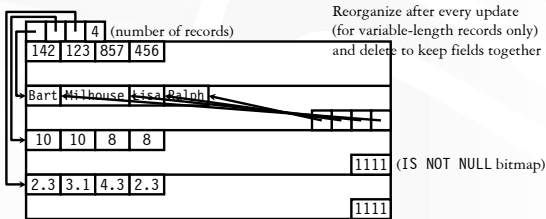
Cache behavior of NSM

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



PAX

- ❖ Most queries only access a few columns
- ❖ Cluster values of the same columns 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



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: the traditional layout
 - PAX: a layout that tries to improve cache performance
