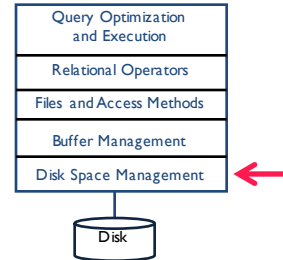


L8 Disk, Storage, and Indexing

Eugene Wu
Fall 2015

Work from the bottom up



\$ Matters

Why not store all in RAM?

Costs too much

High-end Databases today ~Petabyte (1000TB) range.
~60% cost of a production system is in the disks.

Main memory not persistent

Obviously important if DB stops/crashes

Some systems are *main-memory* DBMSes, topic for advanced DB course

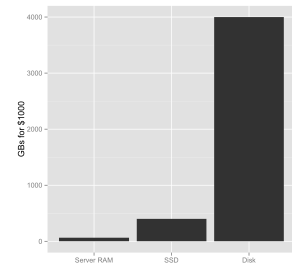
\$ Matters

Newegg enterprise \$1000

RAM: 64

SSD: 400

Disk: 4000

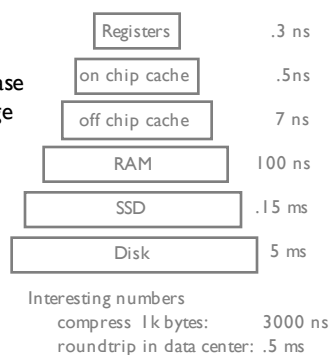


RAM for active data

Disk for main database
secondary storage

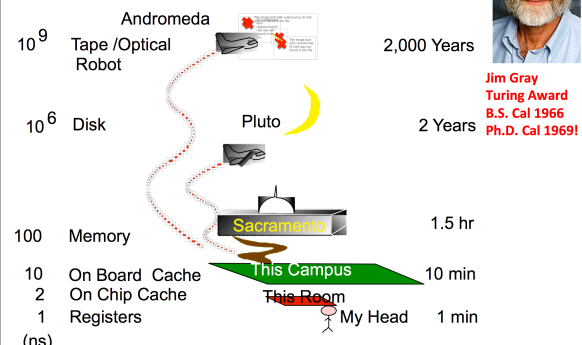
Tapes for archive

SSD is still in flux,
some in use



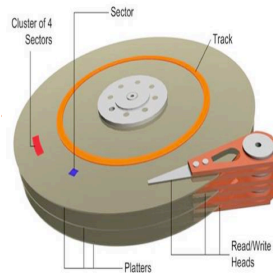
<https://gist.github.com/jboner/2841832>

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Spin speed: ~7200 RPM

Arm moved in/out to position head over a track



Time to access (read or write) a disk block
 seek time 2-4 msec avg
 rotational delay 2-4 msec
 transfer time 0.3 msec/64kb page

Throughput
 read ~150 MB/sec
 write ~50 MB/sec

Key: reduce seek and rotational delays
 HW & SW approaches

Next block concept (in order of speed)
 blocks on same track
 blocks on same cylinder
 blocks on adjacent cylinder

Sequentially arrange files
 minimize seek and rotation latency

When sequentially scanning: Pre-fetch
 > 1 page/block at once

SSD maybe

Fast changing, not yet stabilized

Read small & fast

single read: 0.03ms
 4kb random reads: 500MB/sec
 seq reads: 525MB/sec

Write is slower for random

single write: 0.03ms
 4kb random writes: 120MB/sec
 seq writes: 120MB/sec

Write endurance limited

2-3k cycle lifetimes
 6-10 months

4 byte values read per second



Pragmatics of Databases

Most databases are pretty small

All global daily weather since 1929: 20GB

2000 US Census: 200GB

2009 english wikipedia: 14GB

Data sizes grow faster than moore's law

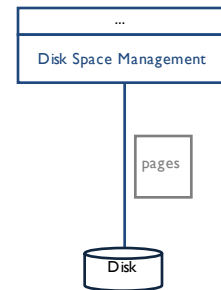
Disk Space Management

VLDBs SSDs: reduce variance
Small DBs interesting data is small

Huge data exists
Many interesting data is small

People will still worry about magnetic disk.
May not care about it

Work from the bottom up



Disk Space Management

Lowest layer of DBMS, manages space on disk

Low level IO interface:
allocate/deallocate a page
read/write page

Sequential performance desirable
try to ensure sequential pages are sequential on disk
hidden from rest of DBMS
but algorithms may assume sequential performance

Files

Pages are IO interface
Higher levels work on records and files (of records)

File: collection of pages
insert/delete/modify record
get(record_id) a record
scan all records

Page: collection of records
typically *fixed* size (8kb in PostgreSQL)

May be stored in multiple OS files spanning multiple disks

Units that we'll care about

Ignore CPU cost
Ignore RAM cost

B # data pages on disk for relation
R # records per data page
D avg time to read/write data page to/from disk

Simplifies life when computing costs
Very rough approximation, but OK for now
ignores prefetching, bulk writes/reads, CPU/RAM

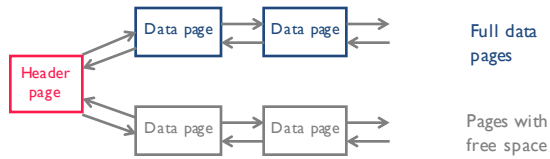
Unordered Heap Files

Collection of records (no order)

As add/rm records, pages de/allocated

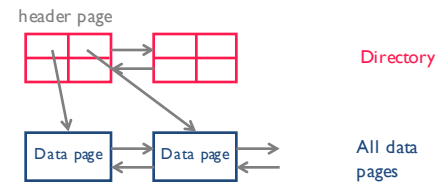
To support record level ops, need to track:
pages in file
free space on pages
records on page

Heap File



Header page and heap file pointers stored in catalog
Data page = 2 pointers + data

Use a directory



Directory entries track #free bytes on data pages
Directory is collection of pages

Indexes

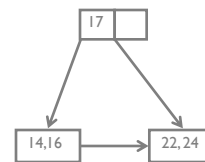
Heap files can get data
by rid
by sequential scan

Queries use *qualifications* (predicates)
find students in "CS"
find students from CA

Indexes
file structures for value-based queries
B+-tree index (~1970s)
Hash index

Overview! Details in 4112

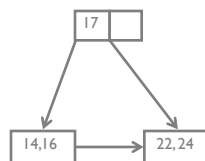
B+ Tree



Query: `SELECT * WHERE val = 14`



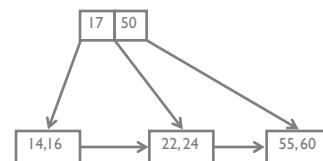
B+ Tree



Query: `SELECT val WHERE val = 14`
(index only)



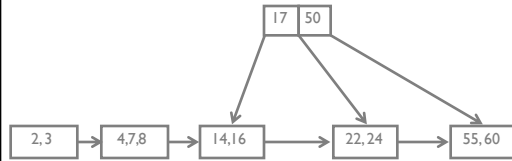
B+ Tree



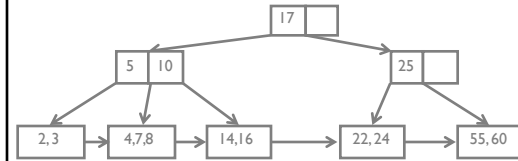
Query: `SELECT * WHERE val = 55`



B+ Tree



B+ Tree



Query: `SELECT * WHERE val > 20`

Some numbers (8kb pages)

How many levels?

fill-factor: ~66%

~300 entries per directory page

if fill completely

height 2: $300^3 \sim 27$ Million

height 3: $300^4 \sim 8.1$ Billion

Top levels often in memory

Level 3 only 90k pages ~750MB

Cool B+ Tree viz: <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

Hash Index

1

$$h(v) = v \% 3$$

0, 6

4, 13

5, 14

Hash Index

1

$$h(v) = v \% 3$$

0, 6

4, 13

5, 14

1

Hash Index

11

$$h(v) = v \% 3$$

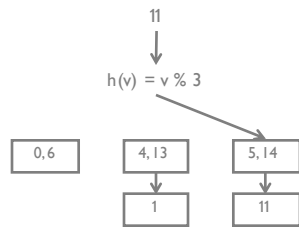
0, 6

4, 13

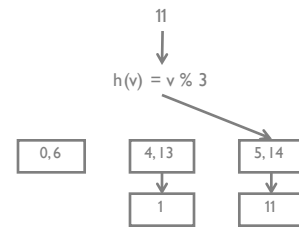
5, 14

1

Hash Index



Hash Index



Good for equality selections
 Index = data pages + overflow data pages
 Hash function $h(v)$ takes as input the search key

Costs

Three file types

Heap, B+ Tree, Hash

Operations we care about

Scan all data `SELECT * FROM R`

Equality `SELECT * FROM R WHERE x = 1`

Range `SELECT * FROM R WHERE x > 10 and x < 50`

Insert record

Delete record

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything				
Equality				
Range				
Insert				
Delete				

B # data pages
D time to read/write page
M # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD			
Equality	0.5BD			
Range	BD			
Insert	2D			
Delete	Search + D			

Heap File

equality on a key. How many results?

B # data pages
D time to read/write page
M # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD		
Equality	0.5BD	D(log₂B)		
Range	BD	D(log₂B + M)		
Insert	2D	Search + BD		
Delete	Search + D	Search + BD		

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B # data pages
D time to read/write page
M # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD	1.2BD	
Equality	0.5BD	$D(\log_2 B)$	$D(\log_{80} B + I)$	
Range	BD	$D(\log_2 B + M)$	$D(\log_{80} B + M)$	
Insert	2D	Search + BD	$D(\log_{80} B)$	
Delete	Search + D	Search + BD	$D(\log_{80} B)$	

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B+ Tree

100 entries/directory page

80% fill factor

B # data pages**D** time to read/write page**M** # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD	1.2BD	1.2BD
Equality	0.5BD	$D(\log_2 B)$	$D(\log_{80} B + I)$	D
Range	BD	$D(\log_2 B + M)$	$D(\log_{80} B + M)$	1.2BD
Insert	2D	Search + BD	$D(\log_{80} B)$	2D
Delete	Search + D	Search + BD	$D(\log_{80} B)$	2D

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B+ Tree

100 entries/directory page

80% fill factor

Hash index

no overflow

80% fill factor

B # data pages**D** time to read/write page**M** # pages in range query

How to pick?

Depends on your queries (workload)

Which relations?

Which attributes?

Which types of predicates ($=, <, >$)

Selectivity

Insert/delete/update queries? how many?

How to choose indexes?

Considerations

which relations should have indexes?

on what attributes?

how many indexes?

what type of index (hash/tree)?

Naïve Algorithm

for each query in order of importance

calculate best cost using current indexes

if there's best plan with a new index

create it

Why not create every index?

update queries slowed down (upkeep costs)

takes up space

High level guidelines

Check the WHERE clauses

attributes in WHERE are search/index keys

equality predicate \rightarrow hash indexrange predicate \rightarrow tree index

Multi-attribute search keys supported

order of attributes matters for range queries

may enable queries that don't look at data pages (index-only)

Summary

Design depends on economics, access cost ratios

Disk still dominant wrt cost/capacity ratio

Many physical layouts for files

- same APIs, difference performance

- remember physical independence

Indexes

- Structures to speed up read queries

- Multiple indexes possible

- Decision depends on workload