Relational Algebra

- (π) Projection: returns only attributes in projection list
  ex: S2: name, phone num
  π name (S2)
- Selection (σ) - select rows satisfying condition
  σ age > 8 (S2)

sid	name	age
1	z	10
3	K	12

- cross product (×): each row of S1 paired w/ each row of R1 (S1 × R1)
- union (∪) union-compatible.... same # of fields, fields of same type.
- set difference (−)

sid	sname	rating
22	A	1
31	B	2
58	C	3

S1

S1 − S2
58, C, 3

S2 − S1
28, D, 5
8, Y, D

sid	sname	rating
28	D	5
22	A	1
31	B	2
8	Y	3

S2

- compound operators: (intersect) ∩ and ⋈ join

natural join example:

R1

sid	bid
1	3
2	4

S1

sid	s	age
1		20
2		21
3		22
4		23

R1 ⋈ S1

sid	sname	bid
1	a	3
2	b	4

- conditional join $\bowtie_\theta$ theta join)
  $R \bowtie_\theta S = \sigma_\theta (R \times S)$

---

examples:

- find names of sailors who've reserved red boat
  $\pi_{sname}((\sigma_{color='red'} \text{Boats}) \bowtie \text{reserves} \bowtie \text{Sailors})$
- find names of artists who have albums of genre of either "pop" or "rock"
  π artist.artist_name (Artists ⋈ (σ albums.genre = "pop" ∨ albums.genre = "rock" Albums))
- find id of artists who have albums of genre "pop" or have spent over 10 weeks in top 40.
  π artist.id (Artist ⋈ (σ Album.genre = "pop" Album)) ∪ π album.artist_id (Album ⋈ (σ Songs.wk_top-40 > 10 songs))
- find names of artists who do not have albums
  π artists.artist_name (Artists ⋈ (π Artist.artist-id Artists) − (π Album.artist-id Albums))

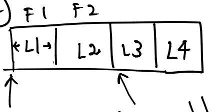
Info: Songs (Song-id, song_name, album-id, top40)
Artists (artist_id, artist_name, first yr)
Albums (album-id, album-name, artistid, yr released, genre)

Storing Data: Disks & Files

DBMS stores info. on disks
lowest level manages space.

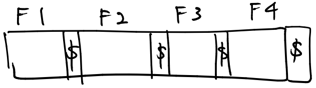
unordered heap files: linked list or pg. directory
↳ allows us to retreive records: Scan sequentially / specify rid <pg id, slot)

record formats: fixed length



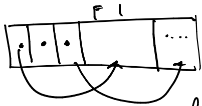
base address (b)   B + L1 + L2

variable length (2 options)



fields delimited by special symbols

OR



array of field offsets

Joins

[R] : # pages to store R
P_R : # records / pg R
|R| : # records in R

$([R] \times P_R) = |R|$

Simple Nested loops Join

$(P_r \times [R]) \times [S] + [P]$

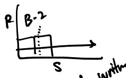
for each tuple, scan S.

Page Oriented Nest Loop Join

$([P] \times [S]) + R$ (doesn't exploit buffers)

for each page R, scan S.

Chunk Nest Loop Join



Say .... B = 100 + 2 memory buffers

Join cost: [outer + [outer_chunks] * [inner]]

#outer chunks = $\frac{\text{outer}}{100}$

Or ...... $[R] + \frac{[R]}{B-2} * [S]$

cost of writing final join

Hash Join

2 pass hash join: $3([R]+[S])$ + output

b/c

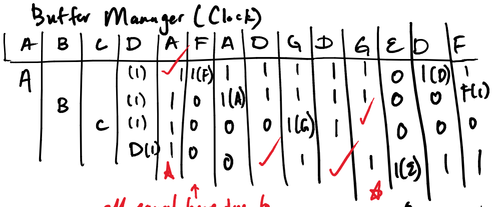
- partioning phase:
  $2([R]+[S])$ I/Os to read/write both relations
- matching phase
  read both rel., write output:
  $([R]+[S]+[\text{output}]$

Compare

nested loops: non equijoins (hash doesn't work)

hash better if relation fits into mem, equivalent

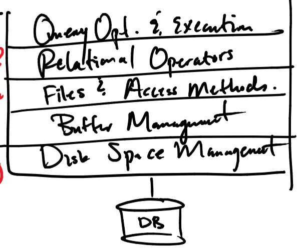
Buffer Manager (Clock)



all equal here due to ref bit. A isn't 'ran' b/c it already had a ref bit of 1!

B/c they don't "add" if ref 1

- When to change col's ref bit to 0? See * .... when you have a col of 1's and LRU (last changed replaced)


Query Opt. & Execution
Relational Operators
Files & Access Methods.
Buffer Management
Disk Space Management

DB

Files & Access Management

Indexes: Disk based data structure for fast lookup by value.

- contains a collection of data entries <k, {items}>

Alternatives for Data Entry k* in index

① Actual data record (w/ key value k)
② < k, rid of matching data record >
③ < k, list of rids of matching records>

- Alt 1 ⇒ clustered

Cost of Operations

- assume single record insert / delete
- exactly 1 match for equality sel.
- heap files: insert always appends to end
- sorted files: Also compacted after deletions & selection on search key.

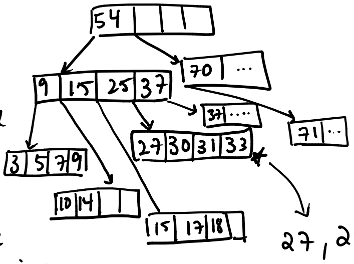
B: no. data pages
R: no. records / page
- D: (avg) time to read/write disk page.

	Heap file	Sorted file	Clustered File
Scan all records	BD	BD	1.5 BD
Equality Search	.5 BD	$(\log_2 B) D$	$(\log_F 1.5B + 1) D$
range search	BD	$[(\log_2 B) + \#\text{match pg}] \times D$	$[(\log\ \ B) + \#\ \text{ma}] \times D$
Insert	2D	$((\log_2 B) + B) D$	$[(\log_F 1.5B) + 2] D$
delete	.5BD + D	" "	" "

B+ trees

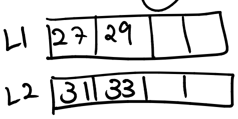
- Create index on files to speed up selection on search key fields. Can have diff indexes on diff keys.
- insert / delete at $\log_f N$ cost
  f = # entries / pg (fanout)
  N = # leaf pages
- height balanced!, length of path from root to leaf node.
- min 50% node occupancy except root
- each node: m entries where $d \le m \le 2d$ entries, where "d" is order of tree.

Insert into B+ tree! (29)



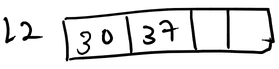
27, 29, 30, 31, 33

①

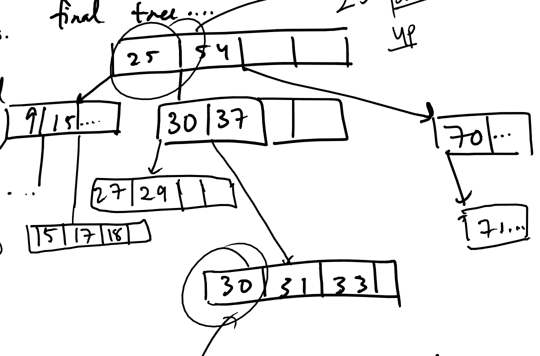


② recursively split

9, 15, 25, 30, 37



- final tree ....



25 pushed up

30 still appears in leaf!! copied up b/c data node

B+ Discussion

Info: 2 million users in database each user entry 2kb, mainly performing range queries on user's age.
page size: 16kb.

① Clustered B+ on age field.
fanout = 200, height = 3
avg 50,000 users / query

How many I/Os per query?: $3 + \left(\frac{50000 \times 2}{16}\right)$

② unclustered. worst case?

Index entry 3x smaller than entry.

- 3 I/Os to descend to leaf pgs.
- read data entries:
  $\left(\frac{50,000 \times \frac{2}{3}}{16}\right) = 2084$ I/Os.
- 50,000 to read unordered data pages.

Data vs Index Pg. Split

data page split: entry inserted in parent node appears in leaf. (copy up)

Index pg. split: push up

Page formats: fixed length

Rid = <page id, Slot #>

- problematic: moving records for free space changes rid!

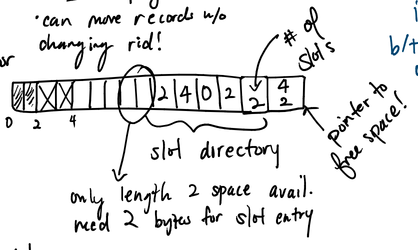
Q. 80 bytes free space (slotted pg). Costs 4 bytes for directory entry.

How many 1 byte records?

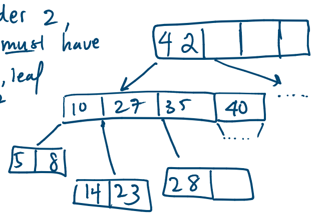
$\frac{80}{(1+4)} = 16$ records!

Slotted page

- can move records w/o changing rid!



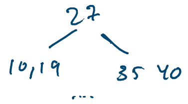
order 2, index must have b/t 2-4, leaf only 2

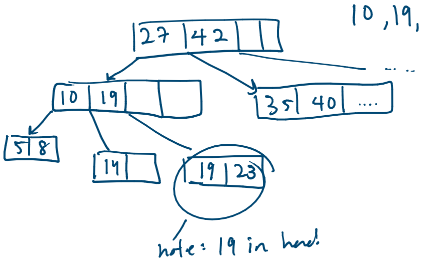


add 19....

14, 19, 23

10, 19, 27, 35, 40





Example: file org/indexes

Consider 500 pgs (B), 6k tuples, query sid > 4500 (sid unique, range 0-6k)

① i/os on heap file?

B = 500

② i/o if stored in sorted file by sid?

$\log_2(500) + \frac{1}{4} \cdot 500$

$\uparrow$

$\frac{4500}{6000} = .25$

Hashing

150 buffer pages, 4 tuples/page

40,000 animals

hash aggregation w/ hybrid hashing

reserve k pages for in-mem table.

↓

max k to hash all animals in 2 passes? (using B-k buffers?)

1 pass .... B-k pages for output partitions, each... no bigger than B. $N \le (B-k) * B$

B = 150 and n = 10,000 $\left(\frac{40000}{4}\right)$

$B - k = 66.67$

$k = 150 - 66.67 = \boxed{83}$

Buffer Manager.

record evictions, 3 page buffer

1	2	3	4	5	6	7	8	9	10
D	B	S	A	B	C	D	S	B	S

Mru: D B ~~S~~ A ~~B~~ C ~~D~~ ~~S~~ ~~B~~ S

evicted ~~C~~

clock: D(1) A(1) 1 1 D(1) 0 0 0

B(1) 0 +1 1 1 S(1) 1 ✓

S(1) 0 0 C(1) 1 0 B(1) 1

1	2	3	4	5	6	7	8	9	10
D	B	S	A	B	C	D	S	B	S

Sequential flooding: LRU + repeated seq. scans

buffer frames < pgs in file

each pg request ⇒ i/o!