SQL

```
(5) SELECT [Distinct] col1, col2, agg(col3)
(1) FROM table1, table2,
(2) WHERE <predicate> AND <predicate> OR.
(3) GROUP BY <column list>
(4) HAVING <predicate>
(6) ORDER BY <columns> [DESC] [ASC]
(7) LIMIT <integer>;
```

No aggregates!

Everything in SELECT must be in GROUP-BY or is an aggregate

Must contain only GROUP-BY columns or aggregate functions

SQL String Comparison

- Old School SQL
  WHERE S.name LIKE 'B_%' ← returns Bob
  starts with B
  _ = any single char ; % = 0+ chars
- Standard Regular Expressions
  WHERE S.name ~ 'B.*' ← returns Bob, McBob
  . = any char , * = repeat (0+ instances of previous)

SQL Join Variants

```
From table1
  [INNER | NATURAL | {LEFT | RIGHT | FULL}
  OUTER] JOIN table2
  ON <qualification list>
```

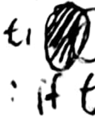
- INNER: join tables where "ON" qualification
- NATURAL: join tables for pairs of attributes w/ same name

```
FROM t1, t2              =   FROM t1 INNER JOIN t2
WHERE t1.id=t2.id            ON t1.id = t2.id
```

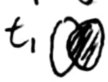
" FROM t1 NATURAL JOIN t2 "

(assume only matching column names = "id"

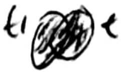
- LEFT/RIGHT/FULL OUTER JOIN
  FROM t1 [ ] OUTER JOIN
  ON t1.id = t2.id
  - LEFT: if t1.id has no match, t2.id is NULL



  - Right: if t2.id has no match, t1.id is NULL



  - FULL: LEFT and RIGHT both apply

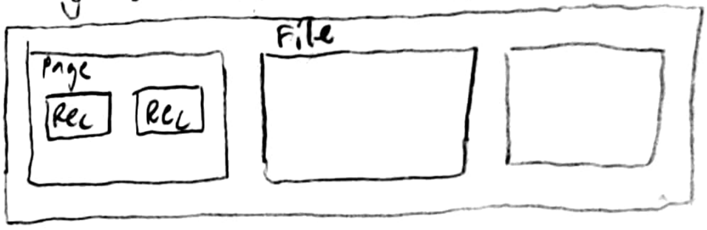


Relational Table Properties

- Schema is fixed
  - unique attribute names
  - atomic (primitive) types
- Tables are not ordered (sets, multisets)
- Tables are flat (no nested tables)
  - First Normal Form

Disk Representation

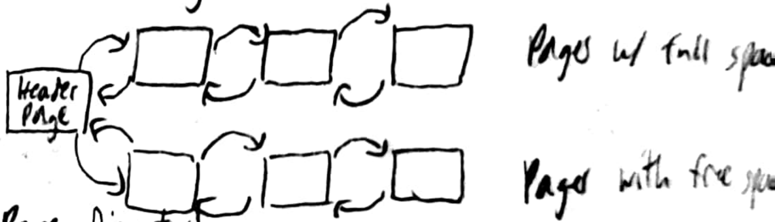
- DB File: collection of pages
- Pages: collection of records ← [pageID, location on page]



Unordered Heap Files

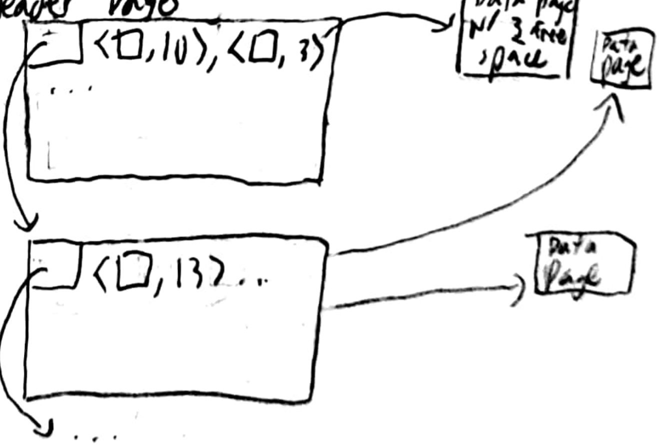
Records placed arbitrarily across pages

1. File as Doubly Linked List



2. Page Directory

Header Pages



Unordered Heap Files vs. Sorted File

| 2,5 | 1,6 | 4,7 | 3,10 | 8,9 | vs. | 1,2 | 3,4 | 5,6 | 7,8 | 9,10 |

	Heap	Sorted
Scan all records	$B \cdot D$	$B \cdot D$
Equality search	$0.5 \cdot B \cdot D$	$(\log B) \cdot D$
Range Search	$B \cdot D$	$(\log B + \text{pages}) \cdot D$
Insert	$2 \cdot D$	$(\log B + B) \cdot D$
Delete	$(0.5 \cdot B) \cdot D$	$(\log B + B) \cdot D$

Assume
- append page to end of heap files
- Sorted files are packed

Page Layout

- FIXED record length, UNPACKED records

Header	0 1 0 1	Rec	X	Rec	X	Rec

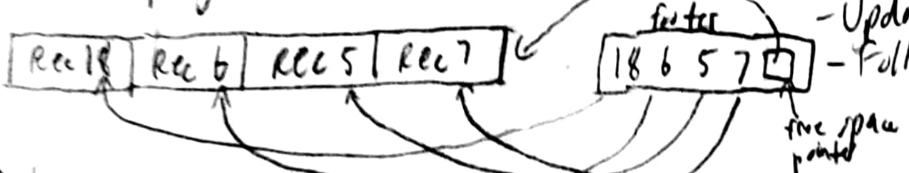
Insert: find first empty slot in bitmap ☹
Delete: find record and clear bit in Header ☺
- No organization needed

- FIXED record length, PACKED RECORDS

Header	Rec	Rec	Rec	Rec

☺
Insert: just append to end
Delete: Scan for record, delete, reorganize all records (b/c pack) ☹

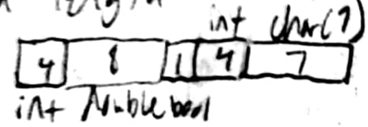
VARIABLE record lengths, UNPACKED records (slotted page)



Delete: remove footer pointer (set record slot to null)
Insert: First empty pointer slot of free space pointer
NOTE: footer stores one int and one pointer for EACH page

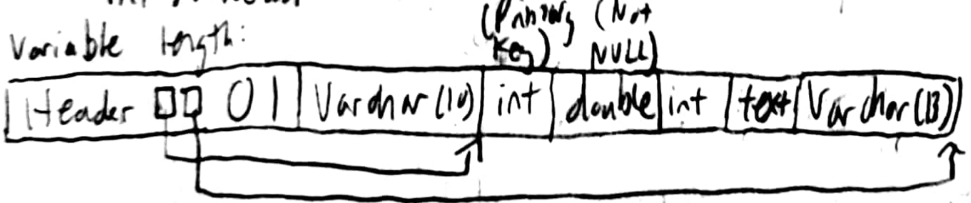
Record Length

- Fixed length:



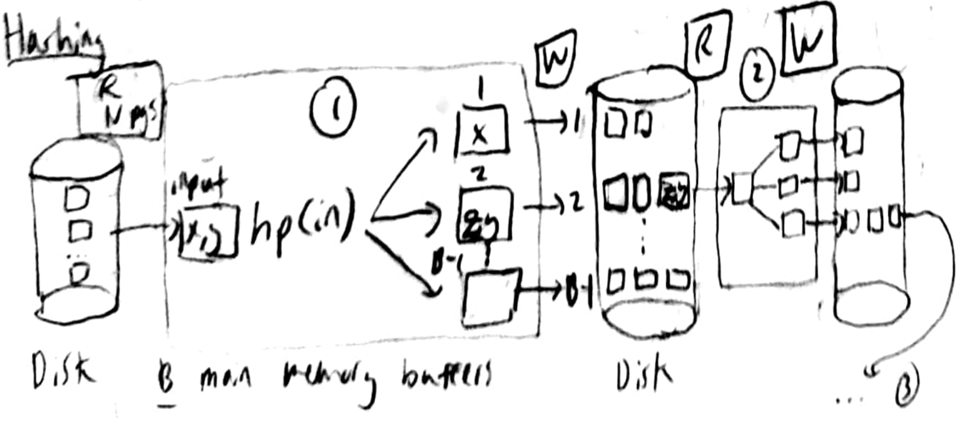
→ store 24 bytes

- Variable length:



- 2 pointers to variable length attributes
- bitmap (2 bits) where int and text are NULL

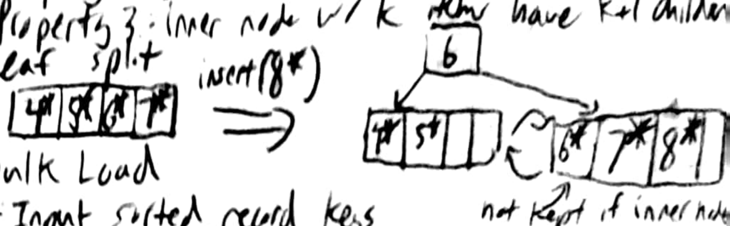
Hashing



Index Files (B+ trees) - acts similar to B-tr

- Property: # entries per node, $d = \text{order} \le \text{entries} \le 2d$
- Alternative 1: record contents stored in the leaf node itself
- Alternative 2: Leaf nodes: ⟨K, rid of matching record⟩
- Alternative 3: Leaf nodes: ⟨K, list of rids of records⟩

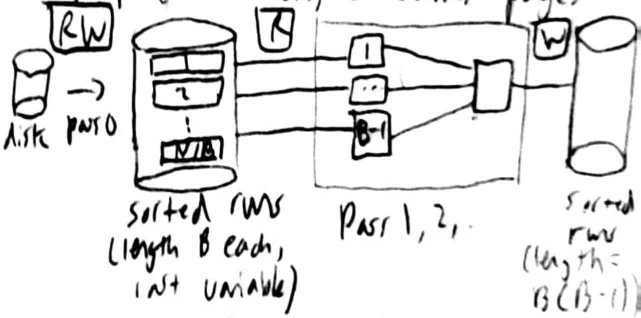
- Property 1: leaf node entries: $d = \text{order} \le \text{entries} \le 2d$
- Property 2: all leaves same dist. from root
- Property 3: inner node w/ k items have k+1 children
- Leaf split



- Bulk Load
  - Input sorted record keys (1*, 2*, 3*, ...)
  - Fill leaf page to fill factor > d
  - Update inner pages to full
  - Follow leaf and inner page split rules

Sorting

- General External Merge Sort
  - N pages to sort, B buffer pages


RW
R
W
disk
pass 0
1
2
⋮
N/B
1
...
B-1
sorted runs (length B each, last variable)
Pass 1, 2, ..
sorted runs (length = B(B-1))

- Number of passes: $1 + \lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Total I/Os $= (\text{I/Os per pass}) * (\#\text{ passes})$
  $= 2N * (1 + \lceil \log_{B-1} \lceil N/B \rceil \rceil)$
- Memory: B(B-1) after 2 passes

① Divide: split pages into B-1 partitions using hp hash function

② Conquer: Repeat for big partitions (>B pages), rehash w/ new hash function hr.

③ When partition fits in memory (<=B pages), read + write to check (Build Pass)