B+Trees

Root node

Leaf nodes; RIDs in sorted order, w/ link pointers
B+Trees

Leaf nodes; RIDs in sorted order, w/ link pointers

Root node

Inner nodes

<val11

>val21, <val22

<valn1
$B+$Trees

Leaf nodes; RIDs in sorted order, w/ link pointers
Properties of B+Trees

- Branching factor = B
- \( \log_B(\text{tuples}) \) levels
- Logarithmic insert/delete/lookup performance
- Support for range scans

- Link pointers
- No data in internal pages
- Balanced (see text “rotation”) algorithms to rebalance on insert/delete
- Fill factor: All nodes except root kept at least 50% full (merge when falls below)
- Clustered / unclustered
## Indexes

<table>
<thead>
<tr>
<th></th>
<th>Heap File</th>
<th>B+Tree</th>
<th>Hash File</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert</strong></td>
<td>O(1)</td>
<td></td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>O(P)</td>
<td></td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Scan</strong></td>
<td>O(P)</td>
<td></td>
<td>-- / O(P)</td>
</tr>
<tr>
<td><strong>Lookup</strong></td>
<td>O(P)</td>
<td></td>
<td>O(1)</td>
</tr>
</tbody>
</table>

- \(n\) : number of tuples
- \(P\) : number of pages in file
- \(B\) : branching factor of B-Tree
- \(R\) : number of pages in range
# Indexes

<table>
<thead>
<tr>
<th></th>
<th>Heap File</th>
<th>B+Tree</th>
<th>Hash File</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert</strong></td>
<td>$O(1)$</td>
<td>$O(\log_B n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>$O(P)$</td>
<td>$O(\log_B n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Scan</strong></td>
<td>$O(P)$</td>
<td>$O(\log_B n + R)$</td>
<td>-- / $O(P)$</td>
</tr>
<tr>
<td><strong>Lookup</strong></td>
<td>$O(P)$</td>
<td>$O(\log_B n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$n$: number of tuples  
$P$: number of pages in file  
$B$: branching factor of B-Tree  
$R$: number of pages in range
Can you think of potential optimizations?
B+Trees are Inappropriate For Multi-dimensional Data

• Consider points of the form \((x,y)\) that I want to index

• Suppose I store tuples with key \((x,y)\) in a B+Tree

• Problem: can’t look up \(y\)’s in a particular range without also reading \(x\)’s
Example of the Problem

Have to scan every $X$ value to look for matching $Y$s.

QUERY
For $Y$
In Some Range

$X = 1$  $X = 2$  $X = 3$  $X = 4$  $X = 5$

$X$

Have to scan every $X$ value to look for matching $Y$s.
R-Trees / Spatial Indexes
R-Trees / Spatial Indexes

Diagram showing a spatial index with rectangles representing regions in an x-y plane.
R-Trees / Spatial Indexes
Study Break

- What indexes would you create for the following queries (assuming each query is the only query the database runs)

SELECT MAX(amount) FROM order
   B+Tree on order.amount

SELECT c_id FROM order WHERE id = 1
   Hash index on order.id

SELECT name FROM order WHERE amount > 100k
   B+Tree on order.amount (maybe)

SELECT * FROM order o WHERE amount > 100k AND c_id = 2
   B+tree on o.amount (maybe), Hash on o.c_id (maybe)
Clicker Question

Assuming disk can do 100 MB/sec I/O, and 10ms / seek, and the following schema:

grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))

Estimate time to sequentially scan grades, assuming it contains 1M records (Consider: field sizes, headers)

(a) .22 sec   (b) 1 sec   (c) .23 sec   (d) 0.21 sec

Int = 8 Bytes, 1MB = 1000KB, 1000000 B-
Seq Scan Grades

grades (cid int, g_sid int, grade char(2))

• 8 bytes (cid) + 8 bytes (g_sid) + 2 bytes (grade) + 4 bytes (header) = 22 bytes

• 22 x 1M = 22 MB / 100 MB/sec = .22 sec + 10ms seek

⇒ .23 sec
Join Processing

- $S = \{S\}$ tuples, $|S|$ pages
- $R = \{R\}$ tuples, $|R|$ pages
- $|R| < |S|$
Nested Loop Join

for s in S
    for r in R
        if pred(s, r) output s join r

Does it matter which is inner, outer? (yes)

suppose |SI| = 4, |RI| = 2, M=3, LRU
S inner = 2 + 4 + 4 = 10 i/o pages read
R inner = 4 + 2 = 6 i/o pages read
Nested Loop Join

B = block size (< M)
while (not at end of S)
    S' = read B records from S
    for r in R
        for s in S'
            if pred(s,r) output (s join r)
Index Nested Loops (Index on R)

for s in S
    find matches in R

$|S| + \{R\} \text{ i/o}$
Sort-Merge Join

\((|R| + |S| < M -- in memory)\)

Sort R, Sort S

Merge

Hash Join

(simple, |R| < M -- in memory)

Build hash on R, probe with S

\(|R| + |S| \ i/\os\)
Blocked hash ($|R| > M$)

Repeat until read all of $R$:
- Read in $M$ pages of $R$
- Build hash on read pages
- Probe with $S$

$|R| + |S| \times \text{ceil} \left( \frac{|R|}{M} \right)$ i/ os
Grace hash

choose P partitions, with one page per partition
hash r into partitions, flushing pages as they fill
hash s into partitions, flushing pages as they fill
for each partition p
  build a hash table T on r tuples in p
  lookup each s in T outputting matches
Grace hash - Example

\[ R = 1, 4, 3, 6, 9, 14, 1, 7, 11 \]
\[ S = 2, 3, 7, 12, 9, 8, 4, 15, 6 \]
\[ h(x) = x \mod 3 \]

Partitioning:
\[ R_0 = 3 6 9 \]
\[ R_1 = 1 4 1 7 \]
\[ R_2 = 14 11 \]

\[ S_0 = 3 12 9 15 6 \]
\[ S_1 = 7 4 \]
\[ S_2 = 2 8 \]

Now, join \( R_0 \) with \( S_0 \), \( R_1 \) with \( S_1 \), \( R_2 \) with \( S_2 \)

Because we are using the same hash function for \( R \) and \( S \) we can guarantee that the only tuples that will join with partition \( R_i \) are those in \( S_i \)
Partition Size / IO-Complexity

How do I pick the partition size?

(Assume uniform distribution of tuples to partitions, make each partition equal to M pages (minus a couple for active pages of S being read.)

I/O:
read R+S (seq)
write R+S (semi-random)
read R+S (seq)

3x (|R|+|S|) I/Os
What's hard about this?

• Possible that some partitions will overflow -- e.g., if many duplicate values
• What do they say we should do?
  • (Leave some more slop (assign fewer values to each partition) by assuming that each record takes a few more bytes to store in hash table.)
• Split partitions that overflow
Clicker Question

Assuming disk can do 100 MB/sec I/O, and 10ms / seek, and the following schema:

grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))

Estimate time to join these two tables, using nested loops, assuming students fits in memory but grades does not, and students contains 10K records.

(a) .241 sec  (b) 0.251 sec  (c) .211 sec  (d) 0.502 sec

Int = 8 Bytes, 1MB = 1000KB, 1000000 B-
NL Join Grades and Students

grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))

10 K students x (100 + 8 + 4 bytes) = 1.1 MB

Students Inner (Preferred)
• Cache students in buffer pool in memory: 1.1/100 s + 0.01 = .021 s
• One pass over students (cached) for each grade (no additional cost beside caching)
• Time to scan students (previous slide) = .23 s
⇒ .251 s

Grades Inner
• One pass over grades for each student, at .22 sec / pass, plus one seek at 10 ms (.01 sec) ⇒ .23 sec / pass
⇒ ~2300 seconds overall

• (Time to scan students is .011 s, so negligible)