CMSC424: Storage and Indexes

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Class

- Storage and Query Processing
  - Using iPython Notebook
  - Indexes; Query Processing Basics

- Other things
  - Project 4: released
  - Poll on Piazza
Query Processing/Storage

1. **Query Processing Engine**
   - Given a input user query, decide how to “execute” it
   - Specify sequence of pages to be brought in memory
   - Operate upon the tuples to produce results

2. **Buffer Management**
   - Bringing pages from disk to memory
   - Managing the limited memory

3. **Space Management on Persistent Storage (e.g., Disks)**
   - Storage hierarchy
   - How are relations mapped to files?
   - How are tuples mapped to disk blocks?

- How are relations mapped to files?
- How are tuples mapped to disk blocks?
Updates on B⁺-Trees: Deletion

- Find the record, delete it.
- Remove the corresponding (search-key, pointer) pair from a leaf node
  - Note that there might be another tuple with the same search-key
  - In that case, this is not needed
- Issue:
  - The leaf node now may contain too few entries
    - Why do we care?
  - Solution:
    1. See if you can borrow some entries from a sibling
    2. If all the siblings are also just barely full, then *merge* (opposite of split)
- May end up merging all the way to the root
- In fact, may reduce the height of the tree by one
Examples of B⁺-Tree Deletion

Before and after deleting “Srinivasan”
B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 3: $133^3 = 2,352,637$ entries
  - Height 4: $133^4 = 312,900,700$ entries
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes
B+ Trees: Summary

- Searching:
  - $\log_d(n)$ – Where $d$ is the order, and $n$ is the number of entries

- Insertion:
  - Find the leaf to insert into
  - If full, split the node, and adjust index accordingly
  - Similar cost as searching

- Deletion
  - Find the leaf node
  - Delete
  - May not remain half-full; must adjust the index accordingly
More…

- Primary vs Secondary Indexes
- More B+-Trees
- Hash-based Indexes
  - Static Hashing
  - Extendible Hashing
  - Linear Hashing
- Grid-files
- R-Trees
- etc…
Secondary Index

- If relation not sorted by search key, called a *secondary index*
  - Not all tuples with the same search key will be together
  - Searching is more expensive
B+-Tree File Organization

- Store the records at the leaves
- Sorted order etc..
B-Tree

- Predates
- Different treatment of search keys
- Less storage
- Significantly harder to implement
- Not used.
Hash-based File Organization

Store record with search key $k$ in block number $h(k)$

e.g. for a person file, $h(SSN) = SSN \% 4$

Blocks called “buckets”

What if the block becomes full?
Overflow pages

Uniformity property:
Don’t want all tuples to map to the same bucket
$h(SSN) = SSN \% 2$ would be bad
Hashed on “branch-name”

Hash function:
- \( a = 1, b = 2, \ldots, z = 26 \)
- \( h(abz) = (1 + 2 + 26) \% 10 \)
- \( = 9 \)
Hash Indexes

Extends the basic idea

Search:
Find the block with search key
Follow the pointer

Range search ?
a < X < b ?
Hash Indexes

- Very fast search on equality
- Can’t search for “ranges” at all
  - Must scan the file
- Inserts/Deletes
  - Overflow pages can degrade the performance
- Two approaches
  - Dynamic hashing
  - Extendible hashing
Grid Files

Multidimensional index structure
Can handle: \( X = x_1 \) and \( Y = y_1 \)
\( a < X < b \) and \( c < Y < d \)

Stores pointers to tuples with:
branch-name between Mianus and Perryridge
and balance < 1k
R-Trees

For spatial data (e.g. maps, rectangles, GPS data etc)
Conclusions

- Indexing Goal: “Quickly find the tuples that match certain conditions”
- Equality and range queries most common
  - Hence B+-Trees the predominant structure for on-disk representation
  - Hashing is used more commonly for in-memory operations
- Many many more types of indexing structures exist
  - For different types of data
  - For different types of queries
    - E.g. “nearest-neighbor” queries