Lecture #08: Tree Indexes II

15-445/645 Database Systems (Fall 2018)

https://15445.courses.cs.cmu.edu/fall2018/

Carnegie Mellon University
Prof. Andy Pavlo

1 Additional Index Usage

Implicit Indexes: Most DBMSs will automatically create an index to enforce integrity constraints (e.g., primary keys, unique constraints).

Partial Indexes: Create an index on a subset of the entire table. This potentially reduces size and the amount of overhead to maintain it.

Covering Indexes: All attributes needed to process the query are available in an index, then the DBMS does not need to retrieve the tuple. The DBMS can complete the entire query just based on the data available in the index.

Index Include Columns: Embed additional columns in index to support index-only queries.

Function/Expression Indexes: Store the output of a function or expression as the key instead of the original value. It is the DBMS's job to recognize which queries can use that index.

2 Skip List

A *skip list* is a sorted linked list with multiple levels of extra points that skip over intermediate nodes. In general, a level has half the keys of the level below it. It provides approximately $O(logn)$ searches.

To insert a new key, flip a coin to decide how many levels to add the new key into.

To Delete, first **logically** remove a key from the index by setting a flag to tell threads to ignore it, and then **physically** remove the key once we know that no other thread is holding the reference.

Advantages over B+Tree:

1. Uses less memory than B+Tree.
2. Insertions and deletions do not require re-balancing. Each change is localized to that part of the skip list.

Disadvantages to B+Tree:

1. Not disk/cache friendly because they do not optimize locality.
2. Reverse search is non-trivial unless you have a doubly linked list.

3 Radix Tree

A *radix tree* is a variant of a trie data structure. It uses digital representation of keys to examine prefixes one-by-one instead of comparing entire key. It is different than a trie in that there is not a node for each element in key, nodes are consolidated to represent the largest prefix before keys differ.

The height of tree depends on the length of keys and not the number of keys like in a B+Tree. The path to a leaf nodes represents the key of the leaf. Not all attribute types can be decomposed into binary comparable digits for a radix tree.

4 Inverted Indexes

An *inverted index* stores a mapping of words to records that contain those words in the target attribute. These are sometimes called a *full-text search indexes* in DBMSs.

Most of the major DBMSs support inverted indexes natively, but there are specialized DBMSs where this is the only table index data structure available.

Query Types:

- Phrase Searches: Find records that contain a list of words in the given order.
- Proximity Searches: Find records where two words occur within n words of each other.
- Wildcard Searches: Find records that contain words that match some pattern (e.g., regular expression).

Design Decisions:

- What To Store: The index needs to store at least the words contained in each record (separated by punctuation characters). It can also include additional information such as the word frequency, position, and other meta-data.
- When To Update Updating an inverted index every time the table is modified is expensive and slow. Thus, most DBMSs will maintain auxiliary data structures to "stage" updates and then update the index in batches.