Weekly exercises IN3020&4020 (2024)
Week 5: Indexes

1 Review

1. What are the differences between primary, secondary, and clustering indexes? What are the differences between dense and sparse indexes?
2. Why can we have at most one primary or clustering index on a file, but several secondary indexes?
3. What is a (static) multilevel index? How does multilevel indexing improve the efficiency of searching an index file? Why the second (and higher) level of the multilevel index should not be dense?
4. Briefly describe B+ and B-trees. What are the main difference between them? What is the order of a B(+)-tree? What is the advantage of using them over static multi-level indexes?
5. What is, briefly, a hash-based index? What is its main advantages and disadvantages in comparison to usual indexes?
6. Introduce alternative choices for accessing a file based on multiple attributes.

2 PostgreSQL

1. Create a table `person` in your Postgres database with the following attributes
   - `personid` INTEGER
   - `lastname` VARCHAR
   - `firstname` VARCHAR
   - `gender` CHAR
2. Download this dataset `https://www.uio.no/studier/emner/matnat/ifi/IN3020/v22/weekly-exercises/person.csv` (it is a dump of the

person table from the movie database) about persons and copy the data into your table:
`=>\copy person from person.csv`

3. Turn on timing (`=>\timing on`)
4. Insert a person `(007, Bond, James, M)` into the table. How long did it take?
5. Update the lastname of person 007. How long does it take?
6. Delete the person `(007, Bond, James, M)` from the table. How long did it take?
7. How long does it take to find the person with `personid` 12345?
8. How long does it take to find the person with `firstname = 'Will'` and lastname = 'Smith'?
9. Create an index on `personid` and do exercise 7 again. What is the time now? Use: `create index pid_index on person(personid);` to create the index.
10. Create another index on firstname and last name and do exercise 8 again. Use: `create index name_index on person(firstname, lastname);` to create this index What is the time now?
11. Is there a time difference if we have one index for both first and lastname or if we have two indexes, one for firstname and one for lastname? Use `drop index name_index;` to remove the index on both first and lastname before you create the two separate indexes.
12. Insert, update and delete a person from the table. How long does this take with the two indexes?
13. Do you see some obvious benefits and/or drawbacks by using indexes?
14. You can cluster (sort) the table by: `cluster person using pid_index`. Does this improve runtime? Why do you think so? In what cases do you think clustering improves performance?

3 Calculations

3.1 Basic Indexes

Suppose a block holds either up to five records (i.e., tuples), or up to 20 key-pointer pairs. As a function of n, the number of records, how many blocks do we need to hold:

1. A data file

2. A dense index
3. A sparse index

3.2 B+ trees

Suppose that a block can hold either up to 20 records or up to 99 keys and 100 pointers. Also assume that an average B+-tree node is 70% full—that is, it has 69 keys and 70 pointers. We can use B+-trees as part of several different structures. For each structure described below, approximately determine the following:

- The total number of blocks needed for a 100,000-record file
- The average number of disk I/Os (i.e., block reads) needed to retrieve a record given its search key

1. The data file is sorted on the search key, with 20 records per block; the B+-tree is a dense index
2. The same as 1, except that the data file is not ordered
3. The same as 1, but the B+-tree is a sparse index
4. The same as 1, but each primary block of the data file has one overflow block such that, on average, the primary block is full and the overflow block is half full, while records in a primary block and its overflow block are in no particular order

3.3 (*) B+ vs B trees

Let both a pointer and a search key takes 1 byte. Let K be the maximal number of bytes for a node in a B(+) tree. Let M be the number of records in a table. Give an example of concrete values of K and M so that a B(+)-tree based dense index may use only 2 layers, while a B-tree needs at least 3.

3.4 More calculation (if there is time)

Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $PR = 7$ bytes long. A file has $r = 30000$ EMPLOYEE records of fixed length. Each record has the following fields:

- `Name` (30 bytes)
- `Ssn` (9 bytes)
- `Department_code` (9 bytes)
- `Address` (40 bytes)
- `Phone` (10 bytes)

- `Birth_date` (8 bytes)
- `Sex` (1 byte)
- `Job_code` (4 bytes)
- `Salary` (4 bytes, real number).

An additional byte is used as a deletion marker.

1. Calculate the record size R in bytes
2. Suppose the file is ordered by the (primary) key field `Ssn` and we want to construct a sparse primary index on `Ssn`. Calculate (a) the number of first-level index entries we need; (b) the number of first-level index blocks; (c) the number of levels needed if we make it into a multi-level index; (d) the total number of blocks required by the multi-level index; (e) the number of block accesses needed to search for and retrieve a record from the file given its SSN value (using the primary index).
3. Suppose the file is not ordered by `Ssn` and we want to construct a secondary index on `Ssn`. Repeat the previous exercises for the secondary index and compare it with the primary index.
4. Suppose that the file is ordered by `Department_code` and we want to construct a sparse clustering index of type 1 on `Department_code` that uses *block anchors* (every new value of `Department_code` starts at the beginning of a new block in EMPLOYEE).Assume there are 1000 distinct values of `Department_code` and that EMPLOYEE records are (nearly) evenly distributed among these values. Calculate (a) the number of first-level index entries; (b) the number of first-level index blocks; (c) the number of levels needed to make a multi-level index (with the maximal number of levels such that only the highest layer has 1 block); (d) the total number of blocks required by the multi-level index; (e) the number of block accesses needed to search for and retrieve all records in the file having a specific `Department_code` value using the clustering index (assume that multiple blocks in a cluster are either contiguous or linked by pointers).