Weekly exercises IN3020&4020 (2026)
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?

   Solution:

   - Primary index: data is physically sorted on disk; indexing field is a candidate key and sort value is unique
   - Clustering index: data is physically sorted on disk; any number of tuples for a search-key value
   - Secondary index: No actual sorting, only index is sorted. Can be used in addition to other indexes.

   Dense indexes have a key-pointer pair for each tuple in the table, sparse indexes have a pair for each block.

2. Why can we have at most one primary or clustering index on a file, but several secondary indexes?

   Solution: We can only sort the data on one attribute

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?

   Solution: The first level of a multilevel index is essentially a usual index, which is presumably does not fit into one block, while the second and higher levels are essentially indexes on the previous levels. This allows to access data with even less block reads (assuming large-enough number of data blocks), because we do not need to do binary search over the first level. A dense second level is useless, because it will look exactly as the first level and hence does not speed up, but slows down the search by one extra block-read.

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?

   Solution: Essentially, B(+)-trees are dynamic multi-level indexes (i.e., the number of levels may change when new data is added/removed). The main difference is that B+ -trees have pointers to data only from the last level, while B-trees have pointers to data at all levels. The order is the maximal number of search keys in a tree node. The main advantages are that the number of layers is not pre-defined and that there are efficient algorithms for adding-removing data.

5. What is, briefly, a hash-based index? What is its main advantages and disadvantages in comparison to usual indexes?

   Solution: Essentially, it is two-level index with one second-level block where hash-values of a dedicated hash function are used instead of search keys. The advantage is that fewer block accesses than with B(+)-trees for exact search. The disadvantage is that poor for interval search and difficulties with referencing relations beyond a pre-defined size.

6. Introduce alternative choices for accessing a file based on multiple attributes.

   Solution: Naive: Intersecting two indexes. More advanced multidimensional indexes:

   - Tree-like indexes: multiple-key index; k-d trees, quad-trees, R-trees
   - Hash-like indexes: grid files
   - Bitmap indexes

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?

    Solution: We don't have any sorting of the data itself that need to be updated. Indexes are easy to update when the nodes are not full.
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?

    Solution: Yes, when you are fetching many rows. If you want all person with personid < 10000, you can probably see a difference between clustered and non clustered index.

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

Solution:

1. Data file: $\lceil n/5 \rceil$
2. A dense index: $\lceil n/20 \rceil$
3. A sparse index: $\lceil n/100 \rceil$

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

   Solution:
   We have 100 possible records in a block with average 70% full 69 keys and 70 pointers. So, we need $100000/70 \approx 1429$ leaf nodes, the level above this needs $1429/70 \approx 21$ nodes, and the level above has only the root node; thus in total: $100000/20 + 1429 + 21 + 1 = 6451$ blocks. To retrieve one block, we access the block with the root node, one block in the second and in the third level before we access the block containing the record. In total 4 disk I/Os.

2. The same as 1, except that the data file is not ordered

   Solution: Same answers as 1

3. The same as 1, but the B+-tree is a sparse index

   Solution: $100000/20 = 5000$ blocks that need an index/pointer pair. $5000/70 \approx 72$. All 72 will fit in the root node. Total $5000+72+1 = 5073$ blocks and 3 disk I/Os.

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

   Solution: Primary block and overflow block have on average 30 records in total. $100000/30 \approx 3334$ double blocks. Which gives $3334/70 \approx 48$ leaf nodes. Total $3334 * 2 + 48 + 1 = 6717$ blocks and 31/3 disk I/Os. (On average we need to check the overflow block 1/3 of the times).

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.

Solution: We can take $K = 7$ and any $M = 9, \ldots, 12$. Then a B+-tree node can accommodate 4 pointers and 3 keys, and so for 1 root node on the second level we may have 4 nodes on the first level with 3 keys on each—that is, 12 first-level keys with pointers overall. However, a B-tree can accommodate 3 sub-tree pointers and 2 key-pointer pairs, and so for 1 root node on the second level we may have 3 nodes on the first level with 2 key-pointer pairs on each—that is, at most 8 keys with pointers overall.

Note that the overall larger number of layers for B-trees than for B+-trees does not mean that they are less efficient on average; this is because for exact search using a B+-tree we always need to retrieve a from-the-root-to-a-leaf branch in the tree, while using a B-tree does not require to reach a leaf in (nearly) half of the cases. Overall, these two approaches are approximately equal in this-sense efficiency, and other considerations should be taken into account when making a choice (e.g., the insertion-deletion algorithms).

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

   Solution: Record length $R = (30+9+9+40+9+8+1+4+4)+1 = 115$ 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).

   Solution: Index record size $R = \text{SSN} + \text{P9} + 6 = 15$ bytes

   (a) We can have $\lfloor 512/115 \rfloor = 4$ records per block. Number of first level entries = number of file blocks = $\lceil 30000/4 \rceil = 7500$ entries.

   (b) Number of first level index entries: $\lceil 7500/34 \rceil = 221$ blocks.

   (c) The number of levels:first-level index entries is 221, above this level we need $\lceil 221/34 \rceil = 7$ nodes. In addition, we need a root node. Hence the index has 3 levels.

   (d) Total number of blocks for the index: $221 + 7 + 1 = 229$ blocks

   (e) Number of block accesses: 4

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.

   Solution:

   (a) Number of first-level index entries = number of file records = 30000.

   (b) Number of first level index entires is $\lceil 30000/34 \rceil = 883$ blocks

(c) Number of levels: second level needs $\lceil 883/34 \rceil = 26$, and the third level needs one root node. Hence the index has 3 levels.

(d) Total number of blocks for the index: $883 + 26 + 1 = 918$ blocks.

(e) Number of block accesses: 4.

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).

Solution:

(a) Number of first-level index entries is equal to the number of distinct `Department_code` values = 1000 entries.

(b) One index entry has size `Department_code` + P = $9 + 6 = 15$ bytes; so the number of entries per block is $512/15 \approx 34$; so the number of first-level index blocks is $\lceil 1000/34 \rceil = 30$.

(c) The number of secondary-level entries is 30, so the second level has only one block it is the root and we need 2 levels.

(d) The number of blocks is $30 + 1 = 31$.

(e) The number of block accesses to search for the first block in the cluster of blocks is $1 + 2 = 3$; so we have $\approx 30000/1000 = 30$ records; 115 bytes/record gives 4 records per block and $\lceil 30/4 \rceil = 8$ blocks; thus we need 2 + 8 blocks to access all (note that we do not need to access the first block twice).