Indexes

Exercises –Indexes

A. Oracle

1. Create a table **person** in your Oracle database with the following attributes
   personid INTEGER
   lastname VARCHAR
   firstname VARCHAR
   gender CHAR
2. Download the dataset "person.csv" (moodle) and copy the data into to table;
3. Insert a person (007, Bond, James, M) into the table. How long did it take?
4. Update the lastname of person 007. How long does it take?
5. Delete the person (007, Bond, James, M) from the table. How long did it take?
6. How long does it take to find the person with personid 12345?
7. How long does it take to find the person with firstname = 'Will' and lastname = 'Smith'?
8. 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.
9. 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?
10. 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.
11. Insert, update and delete a person from the table. How long does this take with the two indexes.
12. Do you see some obvious benefits and/or drawbacks by using indexes?
13. 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?

B. 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 lock is half full, while records in a primary block and its overflow block are in no order.

3.4 More calculation

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.