

Database Tuning, ITU, Spring 2010

Rasmus Pagh

March 4, 2010

1 Sharing indexes

Several queries may share the same index, which is not necessarily the best index for any of them! Essentially, before adding an index one should ask if an existing index would suffice. Possibly, extra columns could be added to allow index-only query plans (but this has a cost). Consider the following relation:

```
Articles(ID,title,journal,issue,year,startpage,endpage,fulltext)
```

Our application uses 6 types of queries, exemplified below:

1. `SELECT year FROM Articles WHERE title='Cuckoo Hashing';`
2. `SELECT title FROM Articles WHERE startpage=100 AND year=1990 AND issue=1;`
3. `SELECT title FROM Articles WHERE year>1995 AND year<2000 AND journal='JACM';`
4. `SELECT title FROM Articles WHERE journal='JACM' AND issue=55;`
5. `SELECT title,fulltext FROM Articles WHERE issue=55 AND journal='JACM';`
6. `SELECT title FROM Articles WHERE endpage-startpage>50;`

a) Suggest a collection of indexes that would support these queries as well as possible. You should aim to minimize the number of indexes. For each query, indicate likely access path(s).

2 Using multiple indexes

In this problem we will investigate whether or not it can be beneficial to use two separate indexes on the same attribute for a relation, in a specific setting. The indexes are defined on a key for the relation, which has N records. We will consider two scenarios:

Scenario 1 Primary hash index on the key.

Scenario 2 Primary hash index **and** secondary B-tree index on the key.

Note that in both Scenario 1 and 2 the relation is only stored once, clustered according to the hash index. In the following we will assume for simplicity that the hash function used distributes the keys evenly, and that no overflow blocks are necessary, even after doing new insertions. Thus,

hash table access uses 1 I/O. We assume that a search in the B-tree requires 2 I/Os. In both cases we assume that an insertion requires 1 additional I/O (we don't consider reorganization cost). Also, assume that the free space in B-tree nodes is negligible, and can be ignored.

a) Suppose a key is 8 bytes and that a pointer is 4 bytes. What are the storage requirements of Scenario 1 and 2, besides the space used for storing the relation itself? (In giving the answer, you may ignore negligible terms.)

The following questions consider insertions of new records, point queries (to look up the record with a specific value for the key), and range queries (returning all records with key in a range of the form $[j, \dots, j + r - 1]$). We denote the size of the range by r , and the number of records returned by a range query by l .

b) Argue that it is possible to obtain the following I/O complexities for the operations insert, point query, and range query, in the two scenarios:

	Insert	Point query	Range query
Scenario 1	2	1	r
Scenario 2	5	1	$2 + l$

c) Consider a sequence of X insertions, Y point queries, and Z range queries, where range queries have range of average size r , and l records in the answer on average. When is scenario 1 and 2, respectively, the most efficient? Express your answer using an inequality involving (a subset of) the variables X , Y , and Z .