THE UNIVERSITY OF SYDNEY

School of Information Technologies
Dr Bryn Jeffries

COMP9120 Relational Database Systems **Semester 2, 2016**

Tutorial Week 10 Solution: Storage and Indexing

Exercise 1. Calculating space and time

Suppose we have a table Rel1(A, B, C). Each A field occupies 4 bytes, each B field occupies 12 bytes, each C field occupies 8 bytes. Rel1 contains 100,000 records. There are 100 different values for A represented in the database, 1000 different values for B, and 50,000 different values for C. Rel1 is stored with a primary, sparse, clustered index on the composite key consisting of the pair of columns (A,B); assume this index has 2 levels. Assume that in this database, each block is 4K bytes, of which 250 bytes are taken for header information. Record locations within the index use a 4-byte *rowid*. Assume that reading a disk block into memory takes 150 msec, and that the time needed for any query can be approximated by the time spent doing disk I/O.

1a) Calculate the space needed for the data of Rel1.
Hint: How much space is used by a single row? Multiply by the number of rows in the table to get a quick estimate of space. [A more accurate answer can be done using the fact that rows are not split across blocks, and each block has a header; so we could work out how many rows per block, then how many blocks for the relation, and then convert this back to space.]

Each block has 4096 – 250 = 3846 bytes available to store data. As each row needs 4+12+8 = 24 bytes, we can fit 3846/24 = 160 records in a block (note that we round down to an integer, as one doesn't store just part of a row in a block). Since Rel1 has 100,000 records, we need 100,000/160=625 blocks to store the relation; measured in bytes it is approx 2.4 MB. (Note that our simplifying assumption here is 100% occupancy and you should always state your occupancy assumption).

1b) Calculate the time taken to perform a table scan through the relation Rel1.
Hint: How many blocks are needed to make up the space occupied by the relation? How many blocks will be read from disk during a scan?

To do a table scan we must fetch each of the 625 blocks of the relation; measured in seconds, this takes 625*150 = 93750 ms = 93.75 s or approximately 1.5 minutes.

1c) Calculate the time taken, using the primary index, to execute the following query

```
SELECT C
FROM Rel1
WHERE A = 'AQG' and (B between 'WPQ' and 'XYZ');
```

We must descend the primary index, reading 1 block at each level (that is, we read 2 blocks of index). Once we have a pointer to the first data record that satisfies the condition, we need to fetch all the data records that match (because the relation is clustered on the composite search key (A,B), the matching rows will all be together, one after another, in the data blocks). The question doesn't give us enough information to know how many matching rows there are, so we need to make some guesses. We know that there are 100 different A values, so we can guess that 100,000/100 = 1000 data records have the A value 'AQG';

what we need to guess is what fraction of them have B values in the given range (this is called the selectivity of the range condition). It is usual in databases to guess that selectivity is 10%, if one doesn't have any better information. Thus we expect 1000*10%=100 records will satisfy the matching condition, and be fetched from the data blocks. Since each block stores 160 records, we probably only need to fetch 1 or perhaps 2 blocks of data to get all the matching records; let's guess that we only fetch 1 data block. Overall, we will need to fetch 2 index blocks, and 1 data block; that is we read 3 blocks from disk, taking 3*150=450ms.

1d) Suppose that we often need to process a query

```
SELECT A, B
FROM Rel1
WHERE (B between 'WPQ' and 'XYZ') and C = 'UBMJ';
```

Why can't the primary index be useful in processing this? What index should be created to speed this up? How long will the query take using the extra index, assuming that the extra index has 2 levels?

The primary index is ordered based on A first, and within that on B; the records we want might have any or every possible A value, so the index doesn't help us find them. Instead let us create an (unclustered, secondary) index on C. We descend the index (doing 2 block reads from disk, one at each level of the index). This gets us to index entries that point at every data record with C = 'UBMJ'. We need to fetch each of these (and then check the B value once the record is fetched). Since there are 50,000 different C values, each C value occurs in approximately 100,000/50,000 = 2 data records; thus we need to fetch 2 data records. because the relation is sorted by (A,B), it is unclustered on C; that is, the 2 data records we fetch are probably located on different blocks of the data records. Overall, we read 2 index blocks and 2 data blocks, or 4 blocks in total. This takes 4*150=600ms.
An alternative solution would be to create an unclustered index on (C,B). In this case, we have to descend the index, and then we can find in the index pointers to all the data records that match the full 'where clause' condition. If we assume 10% as the selectivity of the range condition on B, there are likely to be 0.2 such records (that is, we often might have no matching records, or sometimes we will have 1 matching record). With this index, we often read no data blocks at all (we can determine that the query returns no rows, just from the index); but in other instances we fetch 1 data block.

Exercise 2. Index creation with Oracle

Find which indices already exist for the tables you defined in early tutorials and assignments. Identify an extra index, which would be useful for one of the queries you wrote (supposing that the database became much bigger, with many more rows in each table!). Create this index.

Note: To check which indexes are present in your schema (plus some details), you can use the following SQL commands in Oracle:

```
SELECT * FROM USER_INDEXES;
SELECT * FROM USER_IND_COLUMNS;
```

There are many valid choices here! For a query like "give names of authors born after 1940" an index on birthyear would be useful.
CREATE INDEX author_birthyear_idx on Author(birthyear)