Database Tuning, ITU, Spring 2007

S. Srinivasa Rao

February 20, 2007

Project – deliverable 2

Deadline: March 8, 23.59 Danish time.

Time plan for deliverable 2

February 20. Description, question/answer session.

February 27. Office hours for project supervisor (Milan Ruzic).

March 8. Hand-in by e-mail (pdf file) to `milan@itu.dk`

March 13. Feedback (individually for each group).

Purpose

The main purpose of this part of the project is to understand the importance of index tuning. Proper selection of indexes can speed up the queries significantly with only a small overhead in terms of updates, whereas an improper selection can degrade the performance. Hence a clear understanding of when and how to use indexes is important.

Workload description

More than half the operations performed on the VXL database will be updates. Some of these, that are relevant for the queries described later, are mentioned below (this will be extended in later parts of the project):

- Road segments and end points are very rarely updated, hence you can think of them as essentially static.
- Customers, vehicles and drivers will be added and deleted occasionally. The mileage *attribute* of vehicles is updated more frequently.
- History entries for parcels will be added very freqently.

Queries

The queries will be generated using a Java program by substituting various values for the parameters. You will be provided with a skeleton program containing the names and parameters of methods which you need to implement. You need to execute this program to test your database.

You can download this skeleton program from the ITU file system: `H:/milan/DBT-files` or `/import/home/milan/DBT-files`. Practical questions regarding these can be directed to `milan@itu.dk`.

Create any indexes required, execute the following queries and report the results output by the test program. The methods (of class DatabaseInterface) which you need to implement are specified for each query.

1. Find all 5-way junctions in a given postal code.
   public ResultSet queryNodes(int post_code)
2. Find the addresses (street number, street name and postal code) of all customers whose name starts with the string s.
   public ResultSet queryCustomers_01(String namePrefix)
3. Find the postal code with most number of customers, and list the names of all the customers with that postal code.
   public ResultSet queryCustomers_02()
4. Find the number of customers in the post codes between x and y.
   public int queryCustomers_03(int x, int y)
5. Find the number of parcels that have arrived at storage s during a given date.
   public ResultSet queryParcelHisory_01(int s, Date d)
6. Find the first storage to which parcel p was sent.
   public int queryParcelLog_02(int p)
7. Find all the parcels that were first sent to storage s.
   public ResultSet queryParcelLog_03(int s)
8. Find all the storage locations in the rectangular region with diagonal coordinates $(x1, y1)$ and $(x2, y2)$.
   public ResultSet queryStorages_01(int x1, int y1, int x2, int y2)
9. Find the IDs of all drivers who drive a vehicle of a given type.
   public ResultSet queryDrivers_01(String type)
10. Find the names of all drivers in region *reg* who drive a vehicle with capacity more than *cap* on a given day (of the week).
    public ResultSet queryDrivers_02(String reg, int cap, byte day)

Theoretical investigation — buffered B-trees

We now consider the possible applicability of buffered B-trees in the database. As you may know, a buffered B-tree is not a standard Oracle data type. Imagine that the third party vendor BlastSoft AB is now offering this functionality as a plugin. However, the price is substantial, so it would be nice to know if it would actually be a big advantage to use their software. In particular, we consider the information in the database that deals with arrival and departure of parcels to/from safe storage containers. A common query is to ask for the latest arrival or departure to/from a particular container. A composite index on container ID and time allows this to be answered

efficiently. (This assumes that you store this information in a single relation — if this is not the case, modify the questions below accordingly.)

Looking through a BlastSoft whitepaper, you learn that they are using blocks of 8192 bytes for their buffered B-tree. According to BlastSoft, the extra time for reading blocks of this size compared to a normal B-tree node access is negligible. Block pointers are 4 bytes. The root block is stored in internal memory. The maximum degree n of a buffered B-tree can be specified by the user. Assume in the following for simplicity that the degree of every node, except possibly the root, is $\frac{3}{4}$ times the maximum degree. Also, when answering the following it is allowed to round, make estimates, ignore negligible terms, etc. The aim is not precise calculation, but estimation. However, you should *not* ignore constant factors!

a) Estimate the size of tuples in the relation, and in particular how many tuples can be buffered in a B-tree node. This number should depend on n, as space is required for search keys and pointers.

b) What is the depth of the buffered B-tree in terms of n and the number of keys N?

c) What is the (amortized) cost of updating the buffered B-tree when inserting a new tuple? Your answer should be stated in terms of n and N.

d) Compare the I/O cost of doing x insertions and y queries in the two scenarios: 1. Standard B^+-tree index with block size 4096, and 2. Buffered B-tree index with $n = 4$.

To be handed in

A pdf file with the following:

- An updated E/R diagram for the VXL database (if it is changed from your earlier submission).
- A description of all the indexes created: attribute(s) of the search key, index type (clustered or unclustered) etc. (Include all the 'CREATE INDEX' statements used.)
- Output from the test program (and a brief description of the machine on which the program was run).
- Justification: a breif description indicating why these indexes are created. You can discuss other alternatives that you have thought of tried. You can also include reports on any experiments done (for example, to find the statistics, selectivity etc.) that are relevant. In particular, you should argue that all the indexes you have created are useful.
- Answer to the theoretical investigation.