Assignment 3 – INLS 623

Ramakrishnan Chapter 20, Exercises 20.2 [1,3] (Page 686)

Consider the following BCNF relational schema for a portion of a university database

Prof(ssno, *pname*, *office*, *age*, *sex*, *specialty*, *dept_id*)
Dept(did, *dname*, *budget*, *num_majors*, *chairs_ssno*)

Suppose you know that the following queries are the five most common queries in the workload for the university and that all five are roughly equivalent in frequency and importance:

a) List the names, ages and offices of professors of a user-specified sex (male or female) who have a user-specified research specialty (eg.: *recursive query processing*). Assume that the university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.
b) List all the department information for departments with professors in a user specified age range.
c) List the department id, department name and chairperson name for departments with a user specified number of majors.
d) List the lowest budget for a department in the university.
e) List all the information about professors who are department chairpersons

These queries occur much more frequently than updates, so you should build whatever indexes you need to speed up these queries.

1. Specify first the physical design you would choose for each individual query above. So write optimized index choices for 1(a) through 1(e). Then determine an overall choice of indexes for your physical design taking all the queries into account and provide that 1(overall). In all cases, decide which attributes on each table should be indexed, whether the index is dense or non-dense, note if you are setting it up for an index only solution, and be sure to identify what attribute you are ordering the file by (your "*one free index*", if you choose to do so). Assume that both B+ trees and hashed indexes are supported by the DBMS and that both single- and multiple-attribute index search keys are permitted. When you choose your overall choice, design a physical schema for the university database that will give good performance for the expected workload, but you should not build any unnecessary indexes, as updates will occur (and would be slowed down by unnecessary indexes). In all cases explain you rationale.

2. Redesign the physical schema assuming that the set of important queries is changed to be the following and write out your optimized queries for 2(a) through 2(c) and your overall plan for this schema (2).

a) List the number of different specialties covered by professors in each department, by department.
b) List the department name(s) for the department(s) with the fewest majors.
c) Find the youngest professor who is a department chairperson.

For **both questions** follow the same process we used for Exercise 4, i.e. prepare your answer for each query (what's the best index solution for that query) and I want to prepare an overall answer that gives the best overall plan for index choice(s) for the tables. You can also use either a Hash or B+Tree file ordering for each table (but if you choose a Hash ordering you get only that single Hash ordering, i.e. no other Hash or B+Tree orderings on that table; you can have other B+Tree dense indexes though). For your output you should list the

- SQL equivalent to the query
- Description of your how you think the query will be processed and how you choose your index(es).
- Ordering (or not) of each table file by what attribute(s).
- Specify each index for each table, include (B+Tree or Hash, non-dense vs dense, whether it's index only)

Here is an example result, showing what information you are expected to produce for an individual query optimization:

Dept table: order table file by dept(dname)
Non-dense B+ tree on dept(dname)
Dense B+ tree index on dept(salary)
Dense B+ tree index on dept(ename, deptid)

Emp table: order table file by dept(eid)
Non-dense B+ tree on dept(eid)
Dense B+ tree index on dept(salary)
Dense B+ tree index on dept(ename, deptid)

WorksOn table: order file by hash WorksOn(eid)

Projects table: Not ordered

Explanation: XXXX

3. Consider the following BCNF relations, which describe employees and departments that they work in:

Emp(eid, sal, did)
Dept(did, location, budget)

- Find the location where a user specified employee works
- Check whether the budget of a department is greater than the salary of each employee in that department.

You designed a good optimized design (indexes) for these two relations (Emp and Dept). You analyzed the performance and modified the choice of indexes, and squeezed the best performance you could given the database setup (schema design).

a) Suppose that your customers subsequently complain that performance is still not satisfactory (given the indexes and file organization that you chose for the relations in part (a)). Since you cannot afford to buy new hardware or software you have to consider a schema redesign. Explain how you would try to obtain better performance by describing your new proposed schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations in your new schema.

b) Suppose that your database system has very inefficient implementations of index structures. What kind of a design would you try in this case?

Question 3: index work with our MySQL database:

a. Load into your database (db1_*) the following database dump. It was created using mysqldump (see manual). You can load it with the following command. This example is for user db1_10; change as appropriate for your database id. Note the command is all one line.

```
mysql -u db1_10 -p -h pearl.ils.unc.edu db1_10 <
/htdocs/courses/2006_fall/inls623_001/Exercises/test_search
dbnew
```

This feeds the database dump file (stored in our exercise directory) into mysql which loads it into the db1_10 directory. Note, that this is a large file and will take about ten minutes to load in. This is good practice for real life usage of mysql.

b. Create four different indexes on the “subject” field of the “metadata” table. Use each of the indexes to retrieve the records where subject field contains a user specified string "X". Use X="gene" for your testing. So, your results should include things like “… gene …” but also “genetic…” or “..general.”, i.e. anything containing “gene”. It is helpful to view the records to see what is being returned and I encourage you to do this, however, for what you turn into me, please do not show all the resulting records as there are too many. Instead just do “select COUNT(*)” to report the records returned. Also, show the time the query takes (i.e. what MySQL reports after each command, on the last line). Be sure to drop any existing indexes created by MySQL before each test, and only have the index created that you are testing. To see if your index is being used, use “EXPLAIN” prefix to SQL commands.

   Show the output for all four different solutions: using no index, standard index on text field, shortened character index (char(10)) on subject, and fulltext index (use MATCH AGAINST). Demonstrate all four versions in MySQL, including showing run times. Contrast and compare the results (speed, results returned), and say what you think is the best overall choice, and why.

c. Determine and show the minimal index(es) necessary to allow the following query to return in real time (0.01 second or less).

   Report the subject and title from the metadata table for a range of metadata IDs, that contain the **word** "gene" (i.e. NOT including “genetic”, “general”). The acceptable range for the ID attribute value is between 3000001 and 3300001.