EE382V
10/28/18
V1.0

Homework 8 (Lab 2)

Assigned: 3/28/18
Due: 11/5/18

Required Reading:

1. The original paper defining what has subsequently name, "The Wisconsin Benchmark": "Benchmarking Database Systems , A Systematic Approach" http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.7764

2. A relatively recent retrospective paper: "The Wisconsin Benchmark: Past, Present, and Future" http://firebird.sourceforge.net/download/test/wisconsin_benchmark_chapter4.pdf

Introduction:

A first goal of this lab is to observe that query optimizers do consider the statistical properties of the data and respond with different query plans. An operational intent is that between this lab and the previous lab, you will have, not just visibility, but you will finish the semester with experience with tools that can be used to assess a database workload and optimize the execution of individual queries and the physical organization (partitioning and indexing data) of a database.

The earlier lab targeted the relationship between select predicates, secondary indexes and disk locality. This homework targets the join operator and the optimizer's cost function wrt the size of intermediate join results. An ancillary goal of the lab is the introduction of the classic methodology used in the benchmarking of database systems.

Deliverable:

The deliverable for this homework are the physical plans determined by the query optimizer for each of the queries. This can be obtained using the SQL standard *EXPLAIN* command. The, standard explain command is a command line directive that takes a query as an argument and instructs the query system to process the query all the way to the point of creating a physical plan, but does not then execute the plan. Explain, at the command line, will return text-based details of the physical plan; the expression tree, the physical operators used and execution cost related details. Though the command is part of the SQL standard, the details of the return result are vendor specific.

It is also the case that some database vendors, and many third-party vendors, provide a graphical client that will execute and present explain results in a much more easily digestible visual form. Clearly, you will prefer to employ a graphical interface.

You should come to the next class with hardcopy of these plans.

Next class meeting, the class will breakdown into groups, by database vendor, elect a leader who will facilitate a class discussion comparing the behavior and performance of different database vendors.

BTW:
Finally, after all these many years, (like 40), advances in machine learning are, recently, fulfilling a vision of products that can monitor database workloads, accumulate for themselves the kind of data revealed by explain commands and suggest database optimizations. The nature of these products, necessarily, is that they make suggestions. While there are certainly many organizations that blindly hit a button and the suggestions, and do so with success, it will be quite a while before an organization with a significant database implementation will do so without someone with your (newly aquired) background reviewing the suggested changes. (These intelligent systems are not placing product recommendations or ads on a web page hoping to improve the probability of making a sale. They are assessing *trade-offs* in performance and do not necessarily know all the expectations of the database, the dbms and its operating environment.)

Important: You do not have to execute the queries, just get the optimizer to explain its plan. You will have to run an explicit database command for the database to examine itself and gather statistics for the optimizer. (e.g. Oracle has a suite of such commands, they are prefixed by the word "analyze". DB2's command is runstats.) **Be sure you do this after loading the database and building the indexes, BUT before generating the query plans.**

Per the last assignment, budget time for the initial database load. Note,(hint), if you are clever, you will only have to experience the overhead of your database generator feeding the database server for only one of the tables required exist in the database.

Test data: The test data will comprise the same data table replicated 6 times. Let that table be defined as *TestData*(pk, ht, tt, ot, hund, ten, filler) Let there be at least 5,000,000 rows. Recall you don't have to execute the queries. The columns are defined:
a unique primary key, pk.
Column ht should contain uniformly distributed values from 0 to 99,999 (i.e. **h**undred **t**housand),
Column tt, should contain uniformly distributed values from 0 to 9,999 (i.e. **t**en **t**housand),
Column ot, should contain uniformly distributed values from 0 to 999 (i.e. **o**ne **t**housand),
Column hund, should contain uniformly distributed values from 0 to 99 (i.e. etc.),
Column ten, should contain uniformly distributed values from 0 to 9
Column filler – same as before to make a row at least, but not much larger than 256 bytes.

The values should be generated at random, with replacement. Each columns contents should be generated independent of the other columns (i.e. → the easiest possible way of going about this).

Note this schema is modeled after the schema in a famous database benchmark, *The Wisconsin Benchmark*, Per above, a pair of papers are required reading for this assignment. The original paper: "Benchmarking Database Systems , A Systematic Approach", and a retrospective paper are assigned reading: "The Wisconsin Benchmark: Past, Present, and Future"
http://firebird.sourceforge.net/download/test/wisconsin_benchmark_chapter4.pdf

Physical Schema: Create three copies of the test data table. Call the tables A, B and C. The contents should be identical. Only the names of the tables are different.

Create three more copies of the test data table. Call them A', B', C', (or Aprime, Bprime, Cprime). Build secondary indices on all eligible columns of A',B', and C'

BTW – How big a database (total rows and bytes) are you now being asked to manage?

Queries: Determine physical query plans for SFW queries with the following where clauses You need only turn in certain plans and question answers.

At minimum you should consider the structure of the resulting expression tree, the physical operators chosen, the estimated runtime and size of the final result. Depending on the vendor, there will be additional information concerning the cost, (in CPU and I/O time and space), of each individual operator and subtree. The depth with which you wish engage with investigating that is your choice. But this is very much a situation where what you get out of this is proportional to what you put in.

Section 1:
Determine if and when data range size, T(R), V(R, column-name) and index impact choice of join order and algorithm. The spacing of the questions is suggestive of particular comparisons you should examine, within and across groupings.

1. A.pk = B.pk -- special case of merge-join reigning supreme. Does your system choose merge-join?
2. A.ht = B.ht -- a highly selective join. What is the impact of a secondary index?
3. A'.ht = B'.ht

4. A.ten = B.ten -- same, but, a not very selective join; low selectivity.
5. A'.ten = B'.ten -- note, relatively speaking, a more selective join will have a lower numeric value for
-- join selectivity, than the value of join selectivity for a less selective join. Confused?
-- then, please, go turn up the air conditioner.

6. A.ht = B.ten -- does the optimizer choose its own order for the join, independent of
7. B.ten = A.ht -- what order you wrote the query. Be sure to reverse the mention of the tables
-- in both the FROM and WHERE clauses

8. A.ht = B'.ten -- how about now?
9. B'.ten = A.ht

-- these three-way joins will have to be expressed as two, two-way joins ANDed together.
10. A.ht = B.ht = C.ht
11. A.ten = B.ten = C.ten
12. If 10 and 11 produce different plans, then try these queries using the other columns. See if you can identify a threshold where the plan changes, else simply report " 10 and 11 produce the same plan"

13. A.pk = B.ht = C.ht -- something else to try
14. A.pk = B.hund = C.hund
15. Same as 12, as applied to 13 and 14

16. A'.ht = B'.ht = C'.ht -- rinse and repeat
17. repeat 16 for the hund column
18. Same as 6, as applied to 16 and 17.

Section 2:

19. -- Let's try something big, just on principle

A'.ht = C'.ot AND
A.ht = A'.ten AND
B.pk = C'.hund AND
A'.ht = C'.ot AND
B'.ten = C.ot AND
B'.hund = A.ht

In addition to developing a plan, draw the query graph.
How big is the estimated result, in rows?

20. -- Let's try something, not as big but with richer predicates

A'.ht = C'.ot AND A'.ten = 5 AND
A.ht < A'.ten AND
B.pk = C'.hund AND B.ot < 500 AND
A'.ht = C'.ot

Notice, the A.ht < A'.ten join predicate is equivalent to A.ht < 5 AND A.ht < A'.ten.
If your RDBMS "noticed" this too. Please send me a note with name of the vendor.
How big is the estimated result, in rows?

16. A.ht = B.th = C.ot
17. C.ot = B.th = A.ht
18. A.pk = B'.th = C'.ot
19. C'.ot = B'.th = A.pk

Turn in screenshots that show the same query plan was produced, (or not), even though the order in the where clause is different.

Test to see if the order of the tables in the from clause impacts the final plan. If not, say so. If it does turn in the example.

Read Ch. 15 & 20.