CSE 444 ~~Final Exam~~ 2nd Midterm

August 20, 2010

Name _____Sample Solution_____

Question 1	/ 16
Question 2	/ 16
Question 3	/ 30
Question 4	/ 20
Question 5	/ 18
Total	/ 100

The exam is open textbook and lecture slides only. No other written materials, including homework solutions, and no other devices, including laptops, phones, signal flares, or other means of communication.

Question 1. Performance Tuning (16 points) Suppose we have a relation R(a,b,c,d), where attribute a is the key. The relation is clustered on a, and there is a single B+-tree index on a.

Consider each of the following query workloads, independently of each other. If it is possible to speed it up significantly by adding a *single* additional B+-tree index to relation R, specify which attribute or set of attributes that index should cover. You may only add at most one new index. If two or more possible indices would do an equally good job, pick one of them. If adding a new index would not make a significant difference, you should say so. Give a brief justification for your answers.

(a) All queries have the form: select * from R where b > ? and b < ? and d = ?

An index on (d,b) would be best since it allows us to select exactly the tuples that satisfy these queries.

(Indices on d only would require reading through unneeded values of b. Indices on b alone or (b,d) also are less selective since although they allow us to access the specified values of b, some of the d values in the selected b range might match, while others might not.)

(b) 100,000 queries have the form: select * from R where b = ? and c = ?
and 100,000 queries have the form: select * from R where b = ? and d = ?

An index on either (b,c) or (b,d) would be most selective for half of the queries and would be helpful for the others.

An index on b alone would not be quite as good. It would help all queries equally well, but miss the opportunity to speed up half of them further.

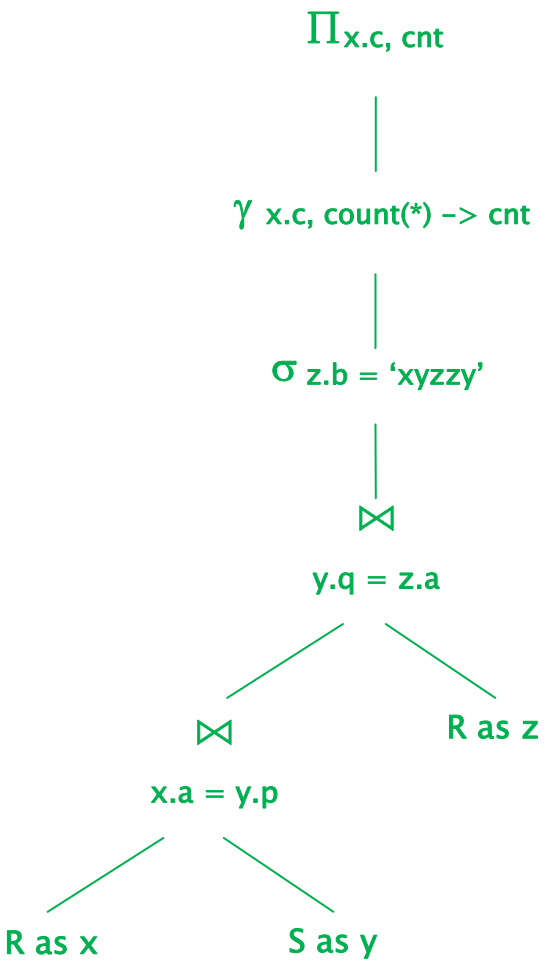
Question 2. Query translation (16 points) Suppose we have two relations

R(a,b,c)
S(p,q)

Draw a logical query plan (relational algebra tree) for the following query. You may choose any query plan as long as it is correct – don't worry about efficiency.

```
SELECT x.c, count(*)
FROM R as x, S as y, R as z
WHERE x.a = y.p and y.q = z.a and z.b = 'xyzzy'
GROUP BY x.c
```

Here is a straightforward translation. One useful optimization would be to push the select down below the uppermost join, but that would not be expected from a simple query parser. It's fine if you did that however. It's also fine if the joins were in the opposite order.



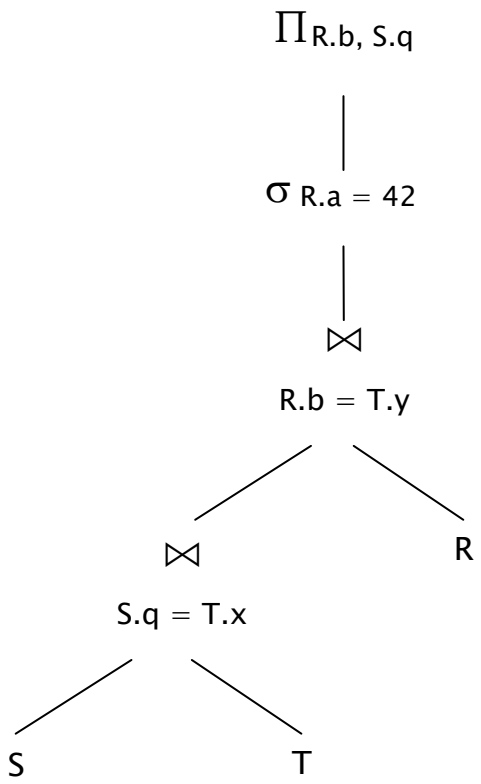
Question 3. Query plans (30 points) We would like to explore query plans involving three new relations:

R(a,b)
S(p,q)
T(x,y)

There are no foreign key relationships between these tables. Each table is clustered on its primary key and has a B+ tree index on that key. We have the following statistics about the tables:

B(R) = 50	T(R) = 2000	V(R,a) = 50	V(R,b) = 20
B(S) = 200	T(S) = 4000	V(S,p) = 400	V(S,q) = 100
B(T) = 100	T(T) = 1000	V(T,x) = 200	V(T,y) = 100

Now consider the following logical query plan:



Answer questions about this query on the next pages. You may remove this page for reference if it is convenient.

Question 3 (cont.) (a) (15 points) What is the estimated cost of the logical query plan given in the diagram on the previous page? For full credit you should both give formulas involving things like T(…), B(…), and V(…, …), and then substitute actual numbers and simplify to get your final estimate. Be sure to show your work so we can fairly award partial credit if there is a problem.

Hint: Cost estimates for logical plans do not include physical operations, but do consider things like the sizes of relations and estimated sizes of intermediate results.

The estimated size of a join T(X ⋈ Y) on a common attribute a is T(X)T(Y) / max (V(X,a), V(Y,a)).

First join: T(S)T(T) / max (V(S,q), V(T,x)) = 4,000 * 1,000 / max (100, 200) = 4,000,000 / 200 = 20,000

Second join: T(first join)T(R) / max (V(R,b), V(T,y)) = 20,000 * 2,000 / max (20, 100) = 400,000

The select on attribute a is estimated to return 1/V(R,a) of the input, so we estimate the number of tuples in the select result to be T(second join) / V(R,a) = 400,000 / 50 = 8,000.

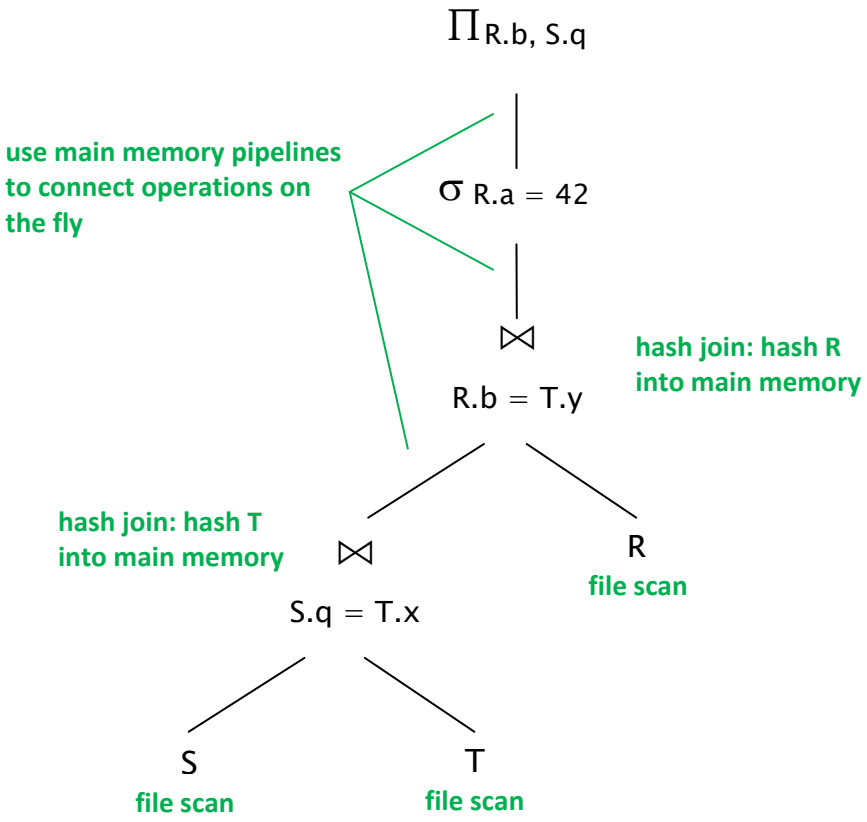
A project does not change the size of the result, so the estimated size of the final result is 8,000 tuples.

[During the exam one student noticed a bug in this question. Attribute a is a key for R, but V(R,a) is only 50. We took that into account when grading in the one case where it mattered.]

(continued next page)

Question 3 (cont.) (15 points) (b) Assume that we have M = 180 blocks of memory available. *Without* rearranging this logical plan, pick the best (cheapest) physical plan to execute it in the available memory. The query is repeated below and you should annotate the diagram showing which physical operations you would use. You should indicate how you plan to access the files as well as how to implement the operations. Be sure it is clear whether intermediate results are materialized on disk or not, which relations are stored in hash tables or other data structures in memory, etc. You do not need to justify your answers, but some brief explanations would help, particularly if it is necessary to assign partial credit.

[There is enough room in main memory to store both R and T in hash tables, then scan S sequentially and pull tuples through the pipelines on the fly without materializing any intermediate results. Total cost would be B(R) + B(S) + B(T).]



Question 4. Oink! (20 points) Consider the following Pig Latin program, which uses the same input data as in the Pig Tutorial:

```
raw = LOAD 's3n://uw-cse444-proj4/excite.log.bz2' USING PigStorage('\t')  AS (user, time, query);
a = GROUP raw BY user;
b = FOREACH a GENERATE group AS user, COUNT(raw) AS n_searches;
c = GROUP b ALL;
d = FOREACH c GENERATE AVG(b.n_searches), MIN(b.n_searches), MAX(b.n_searches);
STORE d INTO '/user/hadoop/answer.txt' USING PigStorage();
```

(a) (6 points) What does this program compute and store into the final output file? Describe what the result is, not the details of how it is computed. (Hint: "GROUP b ALL" sends all tuples of bag/relation b to a single group.)

The output contains a single row:

avg min max

where

avg = mean number of searches by one user

min = minimum number of searches by one user

max = maximum number of searches by one user

[For those who are curious, the actual output when this was run against the excite.log.bz2 data was:

4.705126673040153 1 452

Thanks to Michael, our intrepid TA, for running the program to verify that it worked, as well as coming up with it in the first place.]

(b) (14 points) In order to run this program, the Pig system translates it into a sequence of one or more Hadoop Map-Reduce jobs. On the next page describe the Map-Reduce job(s) needed to execute this program. For each job you should describe the input and output of each map and reduce phase, including the keys and values at each step. You do *not* need to guess exactly how Pig would translate the program as long as your answer gives a reasonable implementation as a sequence of map-reduce jobs.

Please write your answer on the next page.

Question 4. (cont.) (b) Give a series of one or more map-reduce jobs needed to execute this program. Be sure to describe the key/value pairs at each stage.

```
raw = LOAD 's3n://uw-cse444-proj4/excite.log.bz2' USING PigStorage('\t')  AS (user, time, query);
a = GROUP raw BY user;
b = FOREACH a GENERATE group AS user, COUNT(raw) AS n_searches;
c = GROUP b ALL;
d = FOREACH c GENERATE AVG(b.n_searches), MIN(b.n_searches), MAX(b.n_searches);
STORE d INTO '/user/hadoop/answer.txt' USING PigStorage();
```

This program generates two map-reduce jobs.

Job 1:

map 1 input: keys = tuple IDs, values = (user, time, query) tuples.

map 1 output: key = user, value = ● . The actual value doesn't matter, it could be the integer 1, a single character, or any other marker to record one search by that user.

reduce 1 input: key = user, value = [●], i.e., array of search markers.

reduce 1 output: key = user, value = length of array, i.e., search count for that user.

Job 2:

map 2 input: key = user, value = search count for that user

map 2 output: key = "x", value = search count v. Here the key value doesn't matter except that it needs to be the same for all map outputs.

reduce 2 input: key = "x" (same as map 2 output key), value = [v], i.e., array of all individual user search counts.

reduce 2 output: average, min, and max of values in input array [v].

Question 5. XML (18 points) Suppose we have an XML file named "food.xml" that describes a series of products carried in a grocery store according to the following DTD:

```
<!DOCTYPE groceries [
<!ELEMENT groceries (product)* >
<!ELEMENT product (stock_nbr, name, description, price) >
<!ELEMENT description (category, color?) >
…
]>
```

The rest of the DTD is omitted to save space. It describes all of the remaining elements as #PCDATA, i.e., character strings. Each product has a unique stock_nbr element (stock number). The price is given in pennies, i.e., 317 means $3.17.

(a) Write an XPath or XQuery expression that returns the names of all products whose price is less than or equal to 200. The output should be a sequence of elements:

```
<name>gum</name>
<name>banana</name>
```

There are many possible answers using either XPath or XQuery. Here is a simple XPath version:

```
/groceries/product[price/text() <= 200]/name
```

Here "text()" is optional and would be inferred if it were omitted.

(continued next page)

Question 5. (cont) DTD repeated for reference:

```
<!DOCTYPE groceries [
<!ELEMENT groceries (product)* >
<!ELEMENT product (stock_nbr, name, description, price) >
<!ELEMENT description (category, color?) >
…
]>
```

(b) Write an XQuery that returns a well-formed xml document that contains a list of product names and prices of products whose category is "veggie". The output should be formatted as follows, but don't worry about where the line breaks are as long as the proper elements are included in the right order:

```
<result>
    <item><name>broccoli</name>
            <price>149</price>
    </item>
    <item>
            …
    </item>
</result>
```

There are many possible ways to do this. Here's a fairly straightforward one:

```
<result> {
    for $x in document("food.xml")/groceries
    for $y in $x/product[description/category = "veggie"]
    return
        <item>
        { $y/name, $y/price }
        </item>
} </result>
```

Many solutions used a where clause to specify the "veggie" condition instead of including it in the path.

That's it! Have a great summer vacation!!