CSE 444 Final Exam

December 17, 2009

Name _____Sample Solution_____

Question 1	/ 24
Question 2	/ 20
Question 3	/ 16
Question 4	/ 16
Question 5	/ 16
Question 6	/ 8
Total	/ 100

Question 1. SQL (24 points, 6 each part) You've been hired to work on a web site that maintains customer reviews of products. The main data is stored in the following tables:

Product(pid, pname, description)
Reviewer(rid, rname, city)
Review(rid, pid, rating, comment)

The tables contain the following information:

- Product: unique product id (pid), product name (pname), and product description. All strings.
- Reviewer: unique reviewer id (rid), and reviewer name (rname) and city, also all strings.
- Review: One entry for each individual review giving the reviewer and product ids, an integer rating in the range 0-5, and the reviewer comment, which is a string.

(a) Write a SQL query that returns the ratings and comments from all reviews written by reviewers in Seattle for products named 'Zhu Zhu'. The results should be sorted in descending order by rating.

```
SELECT r.rating, r.comment
FROM Product p, Reviewer rr, Review r
WHERE p.pname = 'Zhu Zhu' AND rr.city = 'Seattle' AND p.pid = r.pid AND rr.rid = r.rid
ORDER BY r.rating DESC
```

(b) Write a SQL query that returns the number of reviewers in each distinct city. The results should list the city name and the number of reviewers in that city, and should be sorted alphabetically by city name.

```
SELECT rr.city, COUNT(rr.rid)
FROM Reviewer rr
GROUP BY rr.city
ORDER BY rr.city
```

Question 1. SQL (cont) Schemas repeated below for convenience.

Product(pid, pname, description)
Reviewer(rid, rname, city)
Review(rid, pid, rating, comment)

(c) Write a SQL query that returns the names of all grumpy reviewers. A grumpy reviewer is one whose average review rating is less than or equal to 2. If someone is in the Reviewer table but has no reviews in the Review table, they should not be included in the result. The reviewer names in the result can be in any order.

```
SELECT rr.rname
FROM Reviewer rr, Review r
WHERE rr.rid = r.rid              -- Reviewers with no reviews are excluded from join here
GROUP BY r.rid, rr.rid, rr.rname
HAVING AVG(r.rating) <= 2
```

(d) Write a SQL query that returns the names and descriptions of all cool products. A "cool product" is one that has no reviews in the database with any rating less than 4. A product must have at least one review in the database to be considered as a possible "cool product". The results can be in any order.

```
SELECT p.pname, p.description
FROM Product p, Review r
WHERE p.pid = r.pid
GROUP BY p.pid, p.pname, p.description
HAVING MIN(r.rating) >= 4
```

This can also be done with "WHERE 4 < ALL(…)" or "WHERE NOT EXISTS …", although the above solution is a bit simpler.

Question 2. Logical query plans (20 points) We would like to explore logical query plans involving data about downhill skiing races. Three tables are involved:

Athlete(aid, name, country)
Event(eid, year, location)
Result(eid, aid, time)

The Athlete table gives a unique id (aid) for each athlete and his/her name and country. The Event table gives a unique id (eid) for each event and the event year and location (for example, 2010, Vancouver). The Result table gives the athlete's time for each race that he/she participated in. In the Result table, eid and aid are foreign keys in the Athlete and Event tables.

We have the following statistics for these tables:

B(Athlete) = 100	B(Event) = 40	B(Result) = 250
T(Athlete) = 2,000	T(Event) = 400	T(Result) = 50,000
V(Athlete, country) = 100	V(Event, year) = 50	
	V(Event, location) = 25	

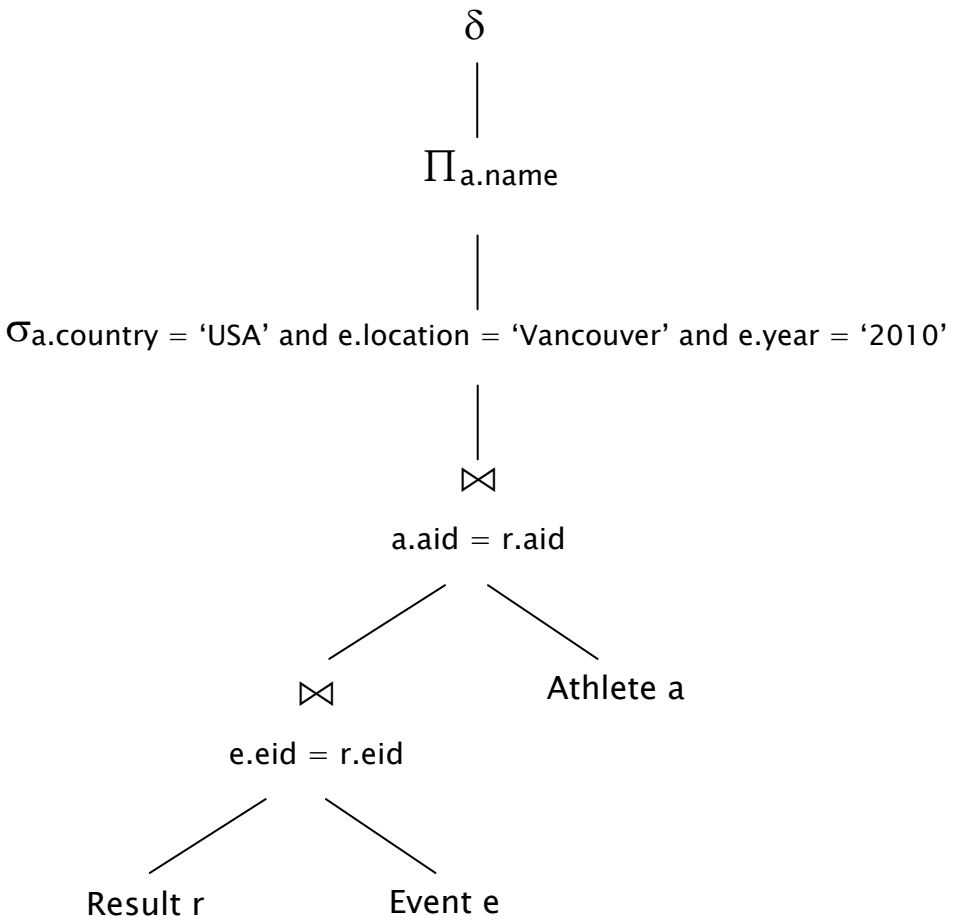
The Athlete and Event tables are clustered on their primary keys, aid and eid respectively. The Result table is not clustered.

Table Athlete has separate B+ tree indices on all three attributes: aid, name, and country. Table Event also has separate B+ tree indices on all three of its attributes: eid, year, and location. Table Result has separate B+ tree indices on eid and aid, but is not indexed on time.

You should assume that there is more than enough main memory (M) to hold any or all parts of these tables so we can use one-pass algorithms to implement queries.

(Continued next page. You can remove this page and the next for reference while you are working on the problem if that is convenient.)

Question 2. (cont.) Consider the following logical query plan involving the relations on the previous page.



Answer questions about this query on the next two pages.

Question 2 (cont.) (a) Translate the logical query plan in the diagram on the previous page to SQL.

```
SELECT DISTINCT a.name
FROM Athlete a, Event e, Result r
WHERE e.eid = r.eid AND a.aid = r.aid AND a.country = 'USA' AND
      e.location = 'Vancouver' AND e.year = 2010
```

(b) 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 give a numerical answer as your final estimate. Be sure to show your work so we can fairly award partial credit if there is a problem.

The goal here is to estimate the size of the intermediate results, so the interesting statistics are number of tuples, not number of blocks or physical access plans.

Since eid and aid are foreign keys in Result, each tuple in Result will join with exactly one tuple in Athlete and exactly one in Event. The cost of each join, then, is the number of tuples in T(Result), so each join will produce 50,000 tuples.

Each clause in the select operation picks 1/V(R,a) for each attribute a. We assume that the distribution of values in the join results matches the original distribution of the attributes in the various relations. The number of tuples that is input into the select operation is T(Result) and the expected number of tuples produced by the select is

T(Result) / (V(Athlete,country) * V(Event,location) * V(Event, year))

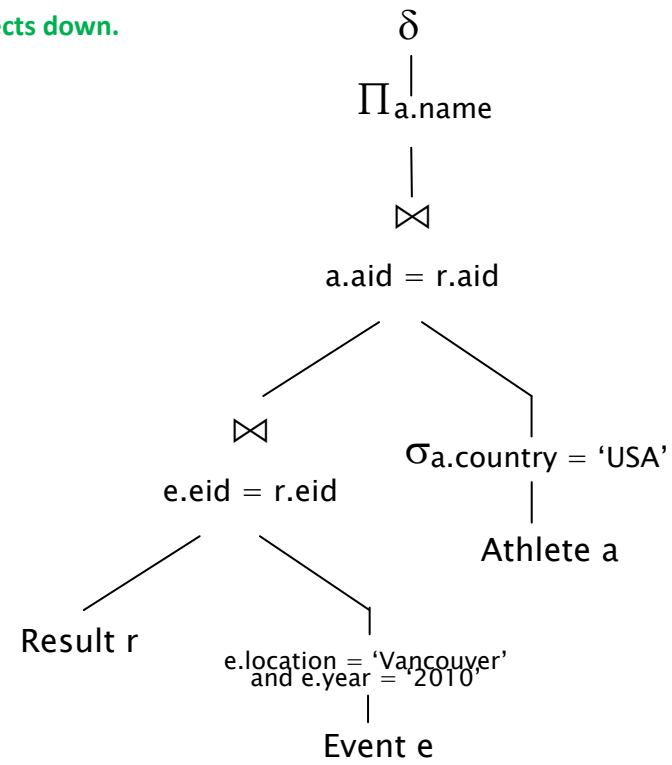
= 50,000 / (100*25*50) < 1

That is clearly a lowball estimate, but it is correct using the traditional estimators.

(continued next page)

Question 2 (cont.) (c) Change or rearrange the original logical query plan to produce one that has the same final results but which is estimated to be significantly faster if possible. Draw your new plan below and give a brief explanation of why you expect it to be cheaper.

Push the selects down.



(d) Give an estimate of the cost of your new plan like the one you did in part (b) for the original plan.

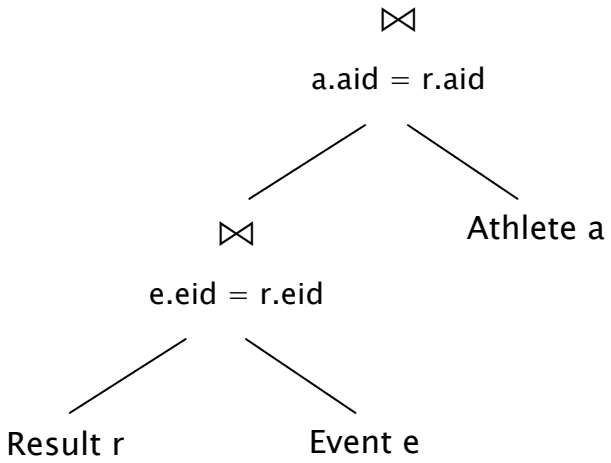
The selections from Event have an estimated size of T(Event) / (V(Event,location)*V(Event,year)) which is 400/50*25 < 1. So we'll estimate that the result of this select has size ≈ 1.

The selections from Athlete have an estimated size of T(Athlete) / V(Athlete, country) = 2,000 / 100 = 20.

The cost of a join T(R⋈S) on an attribute a is estimated as T(R)T(S) / max(V(R,a), V(S,a)). For the first join on Result and the output from the select on event, we have 50,000*1 / max(1, 400) = 125.

The cost of the second join is T(first join)T(select from A) / max (V(join,aid), V(select, aid)) = 125*20/max(2000,2000) = 2500/2000 = 1.25 ≈ 1. So our estimate of the final result size is basically the same as in the original plan.

Question 3. (16 points) For this question we go back to the original set of joins at the base of the tree from the previous problem, using the same relations with the same statistics as before.



If we decide to implement this part of the logical plan exactly as given, what physical plan should we use to perform it? Your answer should specify the physical join operators used (hash, nested loop, sort-merge, etc.) and the access methods used to read the tables (sequential scan, index, etc.) For operations where it matters, be sure to include the details – for instance, for a hash join, which relation would be stored in the hash tables; for a loop join, which relation would be the inner or outer loop. You should specify how the top-most join reads the result of the lower one. You may assume there is enough main memory available for whichever algorithm(s) you choose.

Give the estimated cost of your physical plan in terms of the number of disk operations needed. Also give a brief explanation of why your plan is the best one in terms of the overall cost.

It would be hard to do better than simple hash joins since all tables fit in main memory.

Hash Event into one table at a cost of B(Event). Hash Athlete into another table at a cost of B(Athlete). Then run the joins simultaneously connecting the output of the first into the second with an iterator interface. We read the contents of Result in the process at a cost of B(Result).

Total cost is B(E) + B(A) + B(R) = 40 + 100 + 250 = 390.

Question 4. XML (16 points) As another quarter draws to a close, everyone is thinking not only about the upcoming break, but also their gpa's. Here is an XML DTD for a document describing student transcripts.

```
<!DOCTYPE students [
<!ELEMENT students (transcript)* >
<!ELEMENT transcript (name, id, course*) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT id (#PCDATA) >
<!ELEMENT course (dept, nbr, year, qtr, grade) >
<!ELEMENT dept (#PCDATA) >
<!ELEMENT nbr (#PCDATA) >
<!ELEMENT year (#PCDATA) >
<!ELEMENT qtr (#PCDATA) >
<!ELEMENT grade (#PCDATA) >
]>
```

Here is a short sample of some data that we might find in this database for one student:

```
<students>
    <transcript>
        <name>A. Hacker</name>
        <id>0642379</id>
        <course><dept>CSE</dept><nbr>143</nbr><year>2007</year><qtr>wi</qtr>
                <grade>3.9</grade></course>
        <course><dept>CSE</dept><nbr>444</nbr><year>2009</year><qtr>su</qtr>
                <grade>3.7</grade></course>
    </transcript>
<students>
```

On the next page, write XPath or XQuery expressions as requested to perform the desired tasks. If it matters, you can assume the data is stored in a file named "transcripts.xml".

(You may detach this page for reference if you wish.)

Question 4. (cont). (a) Write an XPath or XQuery expression that returns the names and ID numbers of all students in CSE 378 in Winter quarter ('wi') of 2008.

```
for $t in doc("gpa.xml"/students/transcript[course[dept='CSE'][nbr=378][year=2008][qtr='wi']]/
                                                    (names/text()|id/text())
```

This can also be done with for/where:

```
for $t in doc("gpa.xml")/students/transcript
where $t/course/dept/text() = 'cse' and $t/course/nbr/text() = 378
        and $t/course/year/text=2008 and $t/course/qtr/text = 'wi'
return { $t/name/text(), $t/id/text() }
```

[In grading the question we were looking for correct access to the right elements and weren't too particular about whether the elements or the enclosed text were returned. It probably would have been better to specify more specifically what was required.]

(b) Write an XQuery expression that reformats a valid XML document specified by the original DTD to give a list of students, their id numbers, and their GPAs. The GPA should be computed using a simple average of the each student's course grades – we are ignoring credit hours in this problem. The result should be a complete, well-formed and valid XML document that conforms to this DTD:

```
<!DOCTYPE root-gpa [
<!ELEMENT root-gpa (student)* >
<!ELEMENT student (name, id, gpa) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT id (#PCDATA) >
<!ELEMENT gpa (#PCDATA) >
]>
```

Hint: XQuery includes an avg(…) function.

[There was a bug in this question which, fortunately, didn't seem to cause problems. The root element in the document can't have the same name as a nested element. That has been fixed above and in the solution below.]

```
<root-gpa> {
    for $t in doc("gpa.xml")/students/transcript
    return <student>
            <name> { $t/name/text() } </name>
            <id> { $t/id/text() } </id>
            <gpa { avg($t/course/grade/text()) } </gpa>
          </student>
} </root-gpa>
```

Question 5. Map-Reduce (16 points) For each of the following problems, describe how you would solve it using map-reduce. You should explain how the input is mapped into (key, value) pairs by the map stage, including, if needed, how the key(s) and value(s) are computed. Then you should explain how the (key, value) pairs produced by the map stage are processed by the reduce stage to get the final answer(s). If the job cannot be done in a single map-reduce pass, describe how it would be structured into two or more map-reduce jobs with the output of the each job as input to the next one(s). You should just describe your solution algorithm. You should not translate the solution into Pig Latin or any other specific programming language.

The input data for this problem is a large collection of weather observations. Each line of the input data gives information about a single observation from a particular weather station, and contains the following:

station#, city, state, elevation, year, month, day, time, temperature, precipitation

(a) Print the maximum recorded temperature for each year in the database. The output should have a single line for each year giving the year and the highest temperature found in any observation for that year. The output need not be sorted in any particular order.

Map: **Input: whole file**
Output: (year, temperature) where year is the key and temperature is the value

Reduce: **Input: year and bag of temperature values**
Output: year, max(bag of temperature values)

(continued next page)

Question 5. (cont) (b) Produce a table of cities and their average annual rainfall (precipitation). Note that city names are not necessarily unique (there are over 20 cities named "Lincoln" in the United States), but you may assume that no single state has two cities with the same name. The output need not be sorted.

This can be done in one or two map/reduce jobs, depending on how much work is done in the first reduce job.

One pass version:

Map: Input: whole file
Output: key is (city, state) pair, value is (year, precipitation)

Reduce: Input: key is (city, state), value is bag of {(y1,p1), y2,p2), …} pairs
Processing: sum up the pi's in an array with one entry for each year
Output: city, average computed by averaging the sums in the array elements

Two pass version:

Map1: Input: whole file
Output: key is (city, state, year), value is (precipitation)

Reduce1: Input: (city, state, year) keys and bag of precipitation values
Output: (city, state, year, sum of precipitation)

Map2: Input: output values from Reduce1
Output: key is (city, state), value is (year, sum of precipitation)

Reduce2: Input: (city, state) keys, bag of (year, sum of precipitation) values
Output: city, average of sum of precipitation values / count(years)

Question 6. Databases as a service. (8 points) In the large-scale cloud computing services like AWS and Google, the primary storage that is offered is designed to store large amounts of unstructured or semi-structured data. These systems like Amazon's SimpleDB and Google's Datastore provide the ability to store, index, retrieve, and update data, but do not typically have fixed schemas like the ones that are found in standard relational databases.

(a) Why is this semi-structured organization useful or preferred for the kinds of applications found in large-scaled clustered computing, compared to a traditional relational database? Be brief but give specific reasons or examples that support your point.

Semi-structured or flat files are best for massive data that is read, possibly frequently, but with minimal updates.

There is much less overhead to process data in this format since it doesn't need to be loaded into relational tables first, or cleaned up to fit a fixed schema. Plus we have the flexibility to process data that doesn't have a completely fixed structure.

(b) What sorts of applications are not as well-suited for this kind of data storage and are better served by more highly structured data? When would you want to use a standard relational database instead of less structured storage like SimpleDB or similar services?

Anything that requires frequent updates as well as reads, or that requires high integrity and atomicity (ACID properties). Examples are things like transaction databases for inventory and financial records.

Note that this is not just a question of massive data or distributed processing. There are large, distributed relational databases like Visa or Amazon that need more structured data with transaction semantics. These applications are better suited to relational databases even at large scale.