[Winter 2005 final, problem 2]

Consider an XML document containing information about job postings and job applications. The postings are grouped by company, and applications are listed under postings. An application contains only the applicant's name. For example:

```
<jobs>
  <company>
    <name> MicroScience Corp. </name>
    <posting>
      <jobtitle> sales rep </jobtitle>
      <salary> 30000 </salary>
      <application> Mark </application>
      <application> Lucy </application>
      <application> Frank </application>
    </posting>
    <posting>
      <jobtitle> technology consultant </jobtitle>
      <salary> 80000 </salary>
      <application> Lucy </application>
      <application> Fred </application>
      <application> Mark </application>
      <application> Betty </application>
    </posting>
  </company>
  <company>
    <name> MacroConsulting Inc. </name>
    <posting>
      <jobtitle> technology consultant </jobtitle>
      <salary> 40000 </salary>
      <application> Frank </application>
    </posting>
    <posting>
      <jobtitle> debugger </jobtitle>
      <salary> 20000 </salary>
    </posting>
    <posting>
      <jobtitle> programmer analyst </jobtitle>
      <salary> 35000 </salary>
      <application> Lucy </application>
      <application> Mark </application>
    </posting>
  </company>
</jobs>
```

a. For each of the XPath expressions, indicate how many answers it will return on the example XML document on previous page. For example, if the XPath expression is

/jobs/company[name/text()='MacroConsulting Inc.']/posting

then you will answer 3. For each question, you have to turn in a number.

i. //application
10

ii. /jobs/company/posting[salary/text()>60000]//application
4

iii. /jobs/company[posting/salary/text()>60000]//application
7

b. Write an XQuery expression that returns the names of all applicants who submitted applications to at least two companies. Your query should a list of **<name>** elements, and each element should be included only once. For example, if your query were to run on the XML document shown above, then it will return:

```
<name> Mark </name>
<name> Lucy </name>
<name> Frank </name>
```

```
for $applicant_name in distinct-values (//application/text())
where count(//company[//application/text() = $applicant_name]) >= 2
return
        <name> { $applicant_name } </name>
```

Recall the following pseudo code for counting words in a document.

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1"):

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Working with R(A, B) S(C,D), implement the following in Map Reduce pseudo code:

a) Select * from R where B<9   [selection]

```
map (String key, String value):
    if(value.B <9)
        EmitIntermediate(key, value);

reduce (String key, Iterator values):
    for each v in values:
        Emit(key + values);
```

b) Select distinct B from R [duplicate elimination]

```
map (String key, String value):
    EmitIntermediate(value.B, key + value);

reduce (String key, Iterator values):
    Emit(key);
```

c) Select * from R, S where R.B = S.D [join]

```
map (String key, String value):
    If(value.type==R)
        EmitIntermediate(value.B, key + value);
    Else
        EmitIntermediate(value.D, key + value);
```

```
reduce (String key, Iterator values):
    for each s in values:
        if(s.type == S){
            for each r in values:
            if(r.type == R){
                Emit(key + s + r);
            }

        }
```

(a) **[8 points]** Consider two tables $R(A, B)$ and $S(C, D)$ with the following statistics:

$$\begin{aligned} B(R) &= 5 \\ T(R) &= 200 \\ V(R, A) &= 10 \\ B(S) &= 100 \\ T(S) &= 400 \\ V(S, C) &= 50 \\ M &= 1000 \end{aligned}$$

There is a clustered index on $S.C$ and an unclustered index on $R.A$. Consider the logical plan:

$$P = \sigma_{A=v}(R) \bowtie_{B=C} S$$

There are two logical operators, $S = \sigma_{A=v}$ and $J = \bowtie_{B=C}$, and for each we consider two physical operators:

$$\begin{aligned} s1 &= \text{one pass table scan} \\ s2 &= \text{index-based selection} \\ j1 &= \text{main memory hash join} \\ j2 &= \text{index-based join} \end{aligned}$$

Both $s1$ and $s2$ are pipelined, i.e. the result of the select operator is not materialized. For each of the resulting four physical plans compute its cost in terms number of disc I/0s, expressed as a function of the statistics above. Your answer should consists of four expressions, e.g. $\text{COST}(s1j1) = B(R)B(S)/M + V(R, A)$ (not the real answer).

i. $\text{COST}(s1j1) =$ $\boxed{B(R) + B(S)}$

ii. $\text{COST}(s2j1) =$ $\boxed{T(R)/V(R,A) + B(S)}$

iii. $\text{COST}(s1j2) =$ $\boxed{B(R) + (T(R)/V(R,A)) * (B(S)/V(S,A))}$

iv. $\text{COST}(s2j2) =$ $\boxed{T(R)/V(R,A)) + (T(R)/V(R,A)) * (B(S)/V(S,A))}$

(b) **[2 points]** Indicate the cheapest plan of the four, together with its cost expressed as a number. You will get credit for this point only if you compute correctly all four expressions above.

$\boxed{45}$

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	

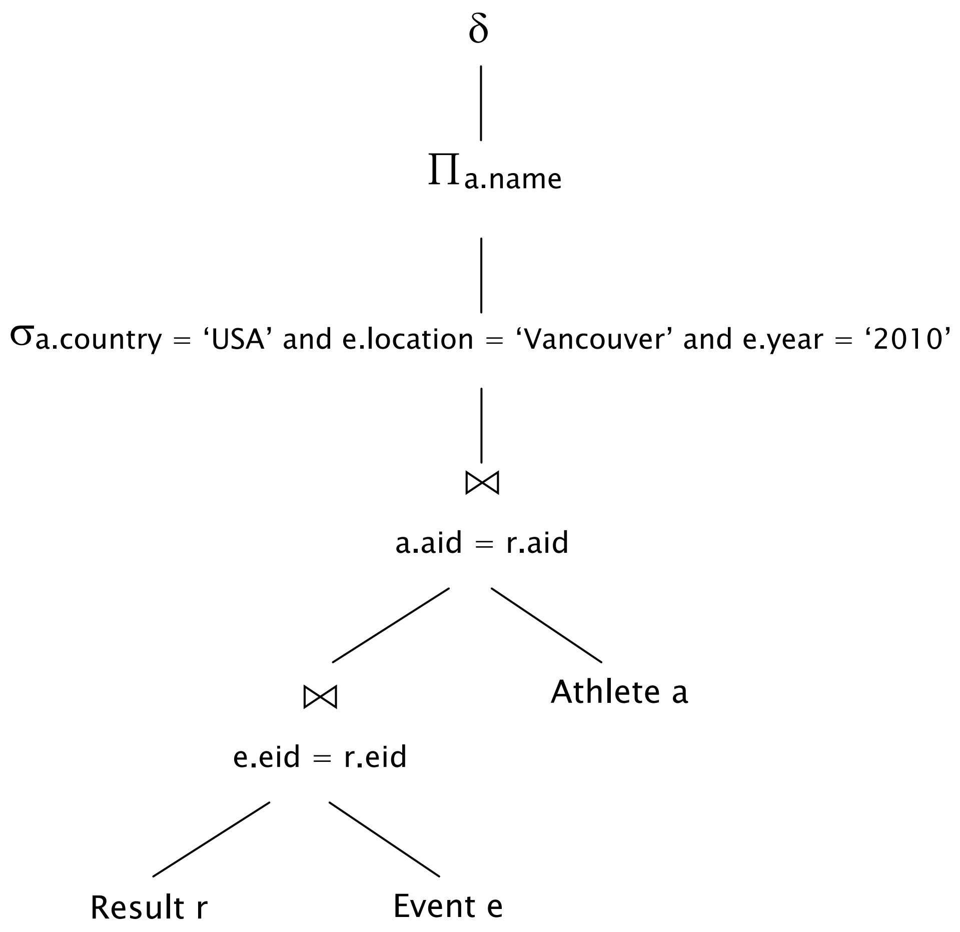
The Athlete and Event tables are clustered on their primary keys, aid and eid respectively. The Result table is not clustered.

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.


δ
Π a.name
σ a.country = 'USA' and e.location = 'Vancouver' and e.year = '2010'
⋈ a.aid = r.aid
⋈ e.eid = r.eid
Athlete a
Result r
Event e

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

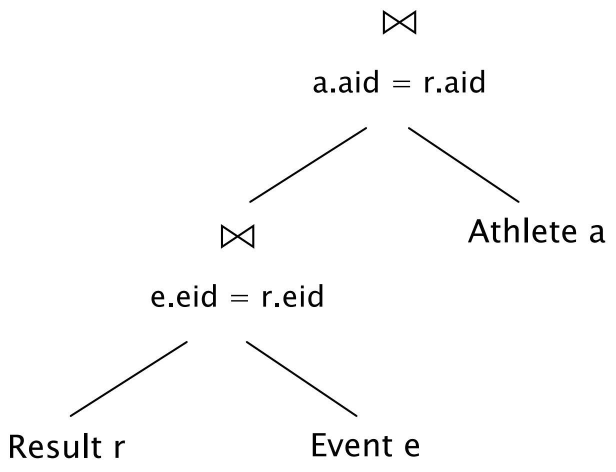
T(Result) / (V(Athlete,country) * V(Event,location) * V(Event, year))

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

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

(continued next page)

Question 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.