Name:_____

CSE 444, Fall 2010, Final Examination
15 December 2010

Rules:

- Open books and open notes.
- No laptops or other mobile devices.
- Please write clearly.
- Relax! You are here to learn.

Question	Max	Grade
1	12	
2	16	
3	20	
4	12	
5	10	
6	10	
Total	80	

Initials:_____

1. (**12** points) **Data Storage and Indexing**

   Suppose we have a relation `R(a,b,c,d,e)` and there are at least 1000 distinct values for each of the attributes.

   Consider each of the following query workloads, independently of each other. If it is possible to speed it up significantly by adding up to **two** additional indexes to relation R, specify for each index (1) which attribute or set of attributes form the search key of the index, (2) if the index should be clustered or unclustered, (3) if the index should be a hash-based index or a B+-tree. You may add at most two new indexes. If adding a new index would not make a significant difference, you should say so. Give a brief justification for your answers. **Your grade will depend on the quality of the indexes that you recommend**.

   (a) (**4** points)

   100,000 queries have the form: select * from R where $b <$?

   10,000 queries have the form: select * from R where $c =$?

   Solution:

   Since we need efficient range-queries on R(b), we definitely want a clustered, B+-tree index on R(b).

   For the second query, an index on R(c) will help. It can be either a B+-tree or a hash-based index since queries look-up specific key values. The index must be unclustered since the index on R(b) is clustered. That is fine, however, since queries will look-up specific key values and we know that there are many distinct values in the relation.

   (b) (**4** points)

   100,000 queries have the form: select * from R where $b <$? and $c =$?

   10,000 queries have the form: select * from R where $d =$?

   1,000 queries have the form: select * from R where $a =$?

   Solution:

   For the first query, a clustered B+-tree index on R(c,b) would be most helpful since we could use it to look-up all data items that match both the given value on c and the range on b.

   Since we can only add a second index, we will favor the most frequent query and add an index on R(d). As in the question above, this index must be unclustered and can be either a B+-tree or a hash-based index.

Initials:_____

(c) (**4** points)

100,000 queries have the form: select a, c from R where $b <$?

10,000 queries have the form: select * from R where $d <$?

Solution:

Since both queries are range-selection queries, we need clustered indexes for both of them, but we cannot have more than one such index. However, we can have a *covering* index.

We thus recommend:

- A clustered, B+-tree index on R(d).
- An unclustered, B+-tree index on R(b,a,c). This is also a covering index in that we only need to use the index to answer the query. We don't need to touch the data.

Initials:_____

2. (**16** points) **Relational Algebra and Query Processing**

Consider three tables R(a,b,c), S(d,e,f), and T(g,h,i).

(a) (**6** points) Consider the following SQL query:

```
SELECT   R.b
FROM     R, S, T
WHERE    R.a = S.d
         AND S.e = T.g
         AND T.h > 21
         AND S.f < 50
GROUP BY R.b
HAVING count(*) > 2
```

For each of the following relational algebra expressions, indicate if it is a correct translation of the above query or not. For your convenience, we provided a tree representation of each relational algebra expression in the following page.

i. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(\sigma_{T.h>21 \text{ AND } S.f<50}(R \bowtie_{R.a=S.d} (S \bowtie_{S.e=T.g} T)))))$

CORRECT INCORRECT

Solution:
CORRECT.

ii. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(R \bowtie_{R.a=S.d} ((\sigma_{S.f<50}(S)) \bowtie_{S.e=T.g} (\sigma_{T.h>21}(T))))))$

CORRECT INCORRECT

Solution:
CORRECT.

iii. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\sigma_{T.h>21 \text{ AND } S.f<50}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(R \bowtie_{R.a=S.d} (S \bowtie_{S.e=T.g} T)))))$

CORRECT INCORRECT

Solution:
INCORRECT.

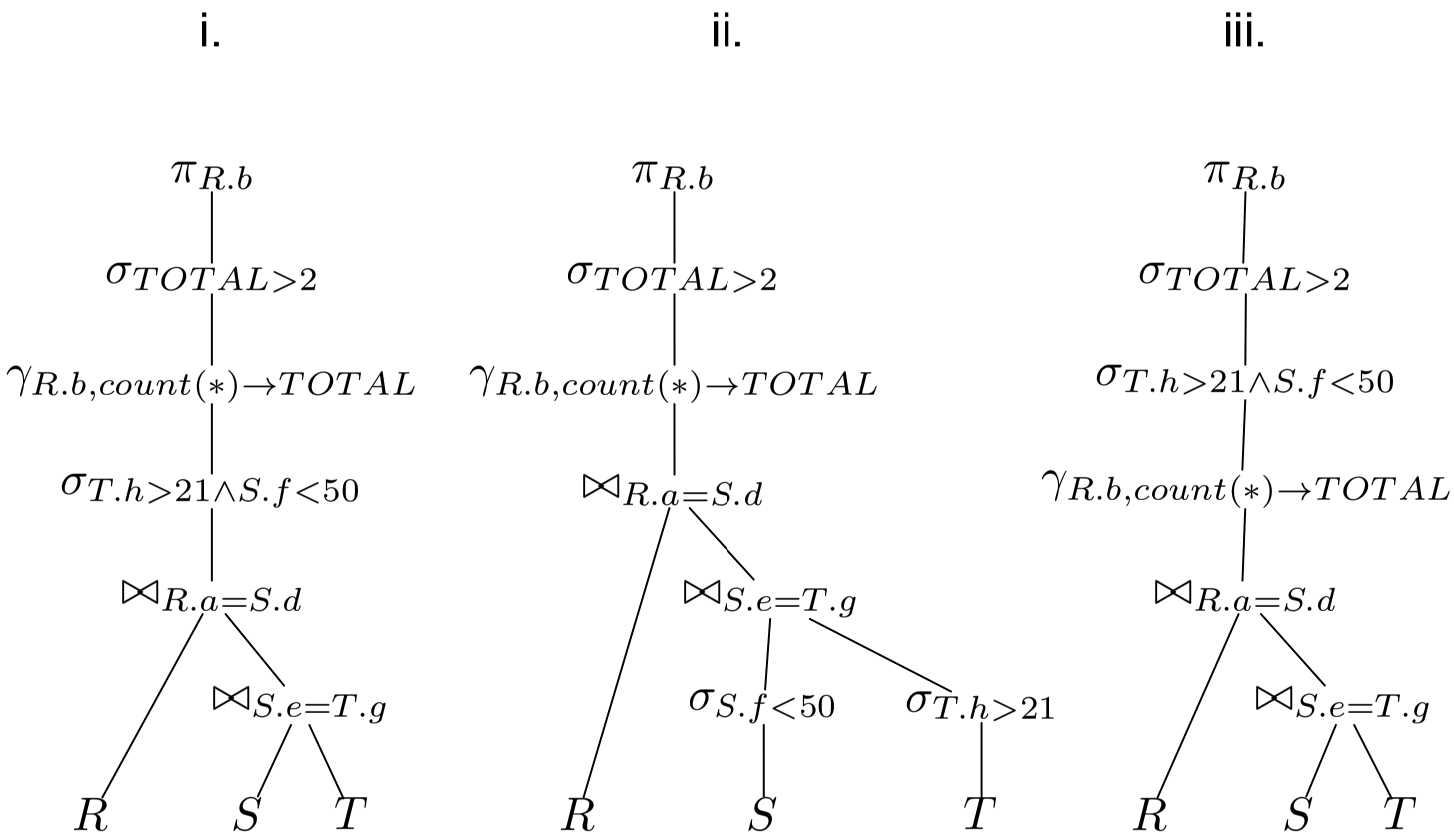

Figure 1: Tree representation of each expression in Problem 2-(a)

Initials:_____

(b) (**5** points) For the following SQL query, show an equivalent relational algebra expression. You can give the expression in the same format as we used above or you can draw it in the form of an expression tree or logical query plan.

```
SELECT R.b
FROM R, S
WHERE R.a = S.d
AND R.b NOT IN (SELECT R2.b FROM R as R2, T WHERE R2.b = T.g)
```

Solution:
$\pi_{R.b}(R \bowtie_{R.a=S.d} S) - \pi_{R.b}(R \bowtie_{R.b=T.g} T)$

Initials:_____

(c) (**5** points) What is the difference between a logical and a physical query plan?

Solution:
A logical query plan is an extended relational algebra tree. A physical query plan is a logical query plan with extra annotations that specify the (1) access path to use for each relation (whether to use an index or a file scan and which index to use), the (2) implementation to use for each relational operator, and (3) how the operators should be executed (pipelined execution, intermediate result materialization, etc.).

Initials:_____

3. (**20** points) **Query Optimization**

Consider the schema R(a,b), S(b,c), T(b,d), U(b,e).

(a) (**5** points) For the following SQL query, give two equivalent logical plans in relational algebra such that one is likely to be more efficient than the other. Indicate which one is likely to be more efficient. Explain.

```
SELECT   R.a
FROM     R, S
WHERE    R.b = S.b
         AND S.c = 3
```

Solution:

i. $\pi_a(\sigma_{c=3}(R \bowtie_{b=b} (S)))$

ii. $\pi_a(R \bowtie_{b=b} \sigma_{c=3}(S)))$

ii. is likely to be more efficient

With the select operator applied first, fewer tuples need to be joined.

One point for each plan, one point for indicating which is more efficient, and two points for the explanation.

Other solutions possible, but unlikely with this simple example.

(b) (**5** points) Recall that a *left-deep* plan is typically favored by optimizers. Write a left-deep plan for the following SQL query. You may either draw the plan as a tree or give the relational algebra expression. If you use relational algebra, be sure to use parentheses to indicate the order that the joins should be performed.

```
SELECT   *
FROM     R, S, T, U
WHERE    R.b = S.b
         AND S.b = T.b
         AND T.b = U.b
```

Solution:

$((R \bowtie_{b=b} S) \bowtie_{b=b} T) \bowtie_{b=b} U$

Initials:_____

(c) (**3** points) Physical plans. Assume that all tables are clustered on the attribute b, and there are no secondary indexes. All tables are large. Do not assume that any of the relations fit in memory.

For the left-deep plan you gave in (b), suggest an efficient physical plan.

Specify the physical join operators used (hash, nested loop, sortmerge, etc.) and the access methods used to read the tables (sequential scan, index, etc.). Explain why your plan is efficient.

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 topmost join reads the result of the lower one.

Solution:

join order doesn't matter, sortmerge for every join, seqscan for R,S,T,U. Fully pipelined.

"clustered index scan" instead of seqscan is also correct.

(d) (**2** points) For the physical plan you wrote for (c), give the estimated cost in terms of $B(...)$, $V(...)$, and $T(...)$. Explain each term in your expression.

Solution:

$B(R) + B(S) + B(T) + B(U)$. Just need to read each table once.

Initials:_____

(e) (**3** points) For the same logical plan you derived in (b), suggest yet another physical plan under the following assumptions.

Recall that a *star schema* consists of one large *fact table* and many (relatively) small *dimension tables*. Assume that R is a large fact table clustered on a and S, T, and U are small dimension tables that are unclustered and fit entirely in memory (all at once). Further, assume a secondary index exists on $U.e$.

Specify the physical join operators used (hash, nested loop, sortmerge, 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 topmost join reads the result of the lower one.

Solution:
left-deep plan, hash joins, seqscan for R,S,T, index scan on U. Hash S,T,U, stream R. Fully pipelined.

(f) (**2** points) For the physical plan you wrote for (e), give the estimated cost of your physical plan in terms of $B(...)$, $V(...)$, and $T(...)$. Explain each term in your expression.

Solution:
$B(R) + B(S) + B(T) + B(U)/V(U.e)$

Initials:_____

4. (**12** points) **XML/XPath/XQuery** Consider the following XML document, named txns.xml.

```
<transactions>
   <purchase>
      <company> Vivace </company>
      <item> latte </item>
      <price> 3.50 </price>
   </purchase>
   <purchase>
      <company> Aveda Institute </company>
      <service> hair cut </service>
   </purchase>
   <purchase>
      <company>  Schultzy's </company>
      <item> Baja Chicken Sandwich </item>
      <price> 8.95 </price>
   </purchase>
   <purchase>
      <company> Vivace </company>
      <item> americano </item>
      <price> 2.50 </price>
   </purchase>
</transactions>
```

(a) (**5** points) Design a simple DTD for this XML document. The XML document must be valid given your DTD (ignoring headers).

Solution:
A possible solution is the DTD below:

```
<!DOCTYPE transactions [
<!ELEMENT transactions (purchase)* >
<!ELEMENT purchase (company, item | service, price?)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT item (#PDCATA) >
<!ELEMENT service (#PDCATA) >
<!ELEMENT price (#PDCATA) >
]>
```

Initials:_____

(b) (**3** points) Write an XPath expression that computes the name of every company where a purchase of more than 5 dollars was made. Don't worry about duplicates.

Solution:

```
/transactions/purchase[price/text() > 5]/company/text()
```

(c) (**4** points) Write an XQuery expression that outputs all items purchased at companies where at least two purchases were made. The output should be a well-formed XML document. **Solution:**

```
<items>
{
FOR $c IN document("txns.xml")/transactions/purchase/company
WHERE count(document("txns.xml")/transaction/purchase[company=$c]) >= 2
RETURN
      {FOR $i in document("txns.xml")/transaction/purchase[company=$c$]/item
      RETURN <item> {$i} </item> }
}
</items>
```

Initials:_____

5. (**10** points) **Parallel Query Processing, MapReduce**

Recall that a distributed program can be expressed in MapReduce using two functions: a Map and a Reduce.

Consider two relations $R(a, b)$, $S(b, c)$.

In this formulation, we will assume that each Map function processes a block of a relation, where the input is a (key,value) pair of the form `(block_id, block_of_records)` For example, using this convention, we can implement a select operation using the following Map function:

```
map(block_id, block_of_records) {
  for each record in block_of_records:
    if condition(record):
      emit(record.id, record)
}
```

For a simple selection, the reduce function does no useful work, and simply emits the records that satisfy the condition:

```
reduce(record_id, list_of_records) {
  for each record in list_of_records:
      emit(record_id, record)}
```

(a) (**5** points) Using the example above as a guideline, write a MapReduce program that implements $\gamma_{a,sum(b)}(R)$.

Solution:

```
map(block_id, block_of_records) {
  for each record in block_of_records:
     emit(record.a, record.b)
}

reduce(groupkey, list_of_b_values) {
  sumb = 0
  for each b in list_of_b_values:
    sumb += b
  emit(groupkey, sumb)
}
```

Initials:_____

(b) (**5** points) MapReduce can only implement unary operators. We can simulate a binary operation in MapReduce by tagging each tuple with its source. For example, each tuple in R might be encoded as

$$(key = a, value = (b, \texttt{'R'}))$$

where `'R'` is a string literal indicating the tuple came from table R.

Using this convention, a MapReduce algorithm to count the tuples in each relation might be expressed in pseudocode as

```
map(block_id, block_of_records) {
  Rcount = 0, Scount = 0
  for each record in block_of_records:
    if record.table = 'R':
      Rcount++
    if record.table = 'S':
      Scount++
  emit('R', Rcount)
  emit('S', Scount)
}

reduce(table, list_of_counts) {
  total = 0
  for c in list_of_counts:
    total = total + c
  emit(table, total)
}
```

Write a **Map** function that, when used with the following reduce function, implements a join. (*Hint:* the reduce function handles the join itself, you just need to make sure all the joinable tuples are gathered in one place.)

```
reduce(key, list_of_records) {
  R = list
  S = list

  for each record in list_of_records:
    if record.table = 'R':
      insert record in R
    if record.table = 'S':
      insert record in S

  for each Srecord in S:
    for each Rrecord in R:
      emit( dummykey, (Rrecord, Srecord))
}
```

Initials:_____

Please write your Map function in this page.

Solution:
Idea is just to hash on the join attribute. Easy if they understand MapReduce, but hard otherwise.

```
map(block_id, block_of_records) {
  for each record in block_of_records:
    emit(record.b, record)
}
```

Initials:_____

6. (**10** points) **Databases as a Service**

Alice is a director of IT department of a global company. The company is running a complex Customer Relationship Management (CRM) system as well as Enterprise Resource Management (ERP) system. Both systems are packaged software (you do not have the source code and can only modify configurations) and use full features of RDBMS (joins, stored procedures, and transactions). The two systems are currently running in a company's private data center. The company recently has decided to close the private data center because of its high operational cost. She considers both Amazon Web Services and Microsoft Azure platform for this big migration.

She asked you, a database expert (or DBA), about this big migration.

(a) (**2** points) List two potential benefits of this move.

Solution:
Many answers were possible for this question including:

i. No management of physical infrastructure.
ii. Elastic scalability.
iii. "pay-as-you-go" price model.
iv. No worry about operations such as high-availability and backup.

(b) (**2** points) List two potential challenges that they will face.

Solution:
Again, many answers were possible including:

i. Data migration, especially when the data is huge
ii. Application migration
iii. Security
iv. Privacy
v. Service level agreement of cloud provider
vi. Limited set of features

Initials:_____

(c) (**2** points) For ERP and CRM systems, which one do you think more suitable in terms of migration cost, Amazon SimpleDB or Microsoft SQL Azure? **Explain** your answer briefly.

Solution:
SQL Azure is more promising because both systems are heavily using RDBMS and SQL Azure supports most of features of RDBMS.

(d) (**2** points) Currently, SQL Azure only supports 50 GB per database with business subscription. However, both ERP and CRM systems have more than 5 TB of data. Does this limitation prohibit the migration? If not, how would you overcome this limit without modifying the systems? **Explain** your answer briefly. (*Hint*: a SQL Azure Server can host up to 149 databases)

Solution:
No. You can partition your data across multiple databases and multiple servers. In this way, you also obtain more CPU, memory, and disks for your query processing. The applications can run without modification if you define views over the partitioned tables and provide the original schema. In reality, SQL Azure does not support distributed query and transaction yet. Thus, you will have to write a middleware layer for distributed queries and transactions but both software remains intact.

With your advice, the company has successfully migrated the IT system to the cloud! You are promoted to a manager and lead a team to develop a new web service. You will roll out a new feature every other week. You are the only database expert in the team and too busy to manage the database.

(e) (**2** points) Which one do you think more suitable for this new service, Amazon SimpleDB or Microsoft SQL Azure? **Explain** your answer briefly.

Solution:
SimpleDB.

- Flexible data model makes rapid evolution of service easier.
- Flexible schema as well as automatic indexing reduces burdens of database management.
- When service picks up the traffic, SimpleDB will automatically scale.