Name:_____ **Student ID:**_____

FINAL EXAM

CS 564 Introduction to Database Management Systems
Department of Computer Sciences
University of Wisconsin, Madison

Exam Rules:

1) Open book and notes, 60 minutes

2) Please write down your name and student ID number NOW.

3) Please wait until being told to start reading and working on the exam.

4) Calculators are allowed.

Fall 2006 final exam with some answers.

1. **(5 points)** What is the difference between logical and physical operators? Give an example of each.

A logical operator specifies the type, such as join, selection, projection, etc. But it does not specify the exact implementation to be used. A physical operator specifies the implementation, such as a block nested loop join, a sort-merge join, etc. A logical operator can be associated with multiple possible physical operators.

2. (15 points) Given two relations R and S with size B(R) = 10000 and B(S) = 8000:
(a) **(5 points)** Suppose the amount of memory available is M = 2001. Compute the disk IO cost of a nested-loop join for R and S.

The smaller table must be the outer table, which is table S in this case.
We will need ceiling[|S|/(M-2)] iterations, in each iteration we need to read through the entire table R. So the total cost of reading R is ceiling[|S|/(M-2)] * |R|. We also read through S once, and the cost of that is |S|. We are ignoring the cost of writing the output.

So the total cost is ceiling[|S|/(M-2)] * |R| + |S|. You can write this, or actually compute and give a number. Either way is okay.

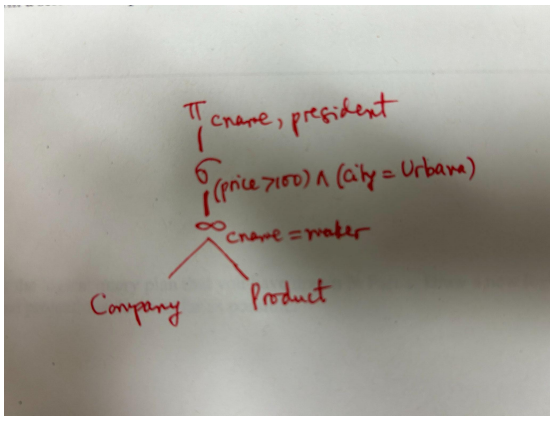
(b) **(10 points)** Suppose the amount of memory available is M=6000. Is it possible to perform a nested-loop join for R and S with cost no more than 25000 disk IO? Explain your answer.

This is a trick question to see if you remember the "ceiling". If you use the above formula without the ceiling, then the amount [|S|/(M-2)] * |R| + |S| will be 21337, and you will answer that it is possible to join with cost less than 25000. However, if you remember that the correct formula is [|S|/(M-2)] * |R| + |S| = 28000. It is not possible to join for less than 25000.

3. (30 points; 6 points each) Consider the two relations: **Company(cname, city, president)**
Product(pname, maker, price).
Assume B(Company) = B(Product) = 10000 blocks. Assume further that each block can accommodate 5 records, and that the memory has 101 buffers.

Consider the SQL query: **SELECT Company.cname, Company.president**
FROM Company, Product
WHERE (cname=maker) AND [(price > 100) AND (city = "Urbana")]

(a) Draw a logical query plan tree for the above SQL query. This plan tree should join Company and Product, then perform a selection on price and city, then project out cname and president.



(b) For the logical query plan that you draw in Part a, consider the following physical query plan. First, perform a nested loop join of Company and Product, with Company being the outer relation. Next, pipeline the result of the join into the selection operation. Finally, pipeline the result of the selection operation into the projection operation. Compute the total cost of this physical query plan.

Since we pipeline the selection and the projection, the cost of this plan is just the cost of the join, and this is nested loop join, so we can use the formula for this (see examples in Question 2).

(c) Assume that there is an index on attribute cname of relation Company, and that cname is a key for that relation. Can we replace the nested loop join operation in the physical query plan mentioned in Part b with an index based join operation? If so, which relation should be the outer relation in this index based join operation, and what should be the cost of this new physical query plan (which uses the index based join operation)?

Yes, we can replace. It will be an index nested loop join. Product will be the outer relation. We must read in the entire Product relation, so the cost of that is 10000 pages (aka blocks). For each page, we have 5 records.

For each record, we can probe the index on cname of Company, to find matching tuples in Company, and we will find just ONE matching tuple, because cname is a KEY for Company.

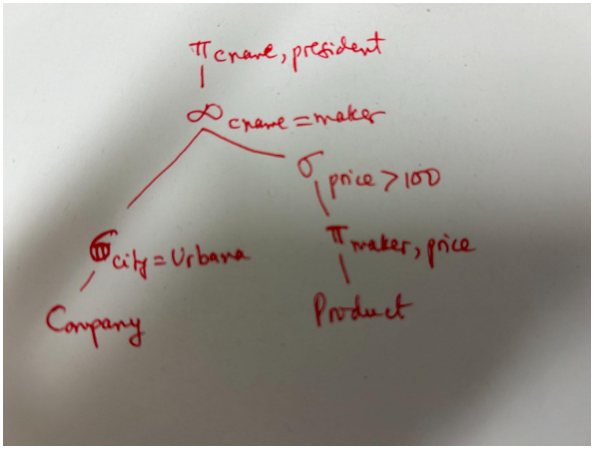
Assuming that each tuple we find in Company resides on a separate page (a worst-case scenario), we need to read in that page. Since we have 10000*5 = 50K records in Product, and for each record we may need to read

in a page in Company, the total number of pages we read in is 50K in worst case scenario. So the total cost is 10K (to read in Products) plus 50K (of reading in Company pages).

Assumption: here we also assume the index fits in memory, so there is no cost of looking up entries in the index.

Note: this problem is underspecified. So as long as you make clear which assumptions you are making, and they are reasonable, then you should be fine.

(d) Consider the logical query plan that you have drawn in Part a. Draw a new logical query plan that pushes selections and projections down as far as possible.



It is also okay if you put projection(maker) above selection(price > 100) on the Product side.

(e) Outline at least one scenario where pushing selections down is not desirable. Explain why.

If you are doing an index nested loop join between A and B, where A is the outer table, and you have an index over the inner table B, then pushing a selection down below the join, on B is not desirable, because the join then can't use the index on B.

4. (20 points) Compute the optimal plan for R ⋈ S ⋈ T ⋈ U using the dynamic programming algorithm, assuming the following:
B(R) = 500, B(S) = 200, B(T) = 600, B(U) = 400
The size of a join is estimated as: B(A ⋈ B) = 0.01 * B(A) * B(B)
The cost of a join is estimated to be the cost of the subplans plus the size of the intermediate results (the same as the cost model we have covered in the lecture).
Please draw the table for dynamic programming, to show how you compute the optimal plan.

We did not cover this topic in class, so no question like this on exam.

5. (10 points) Briefly describe the ACID properties.

I covered this in the lecture in the class.

6. (20 points) A database has four elements A, B, C, and D. It has just crashed. Assume we maintain an undo log whose content after the crash is

```
<start T1>
<T1,A,3>
<start T2>
<T2,B,1>
<start ckpt(T1,T2)>
<T1,A,4>
<start T3>
<commit T1>
<T2,B,2>
<T3,C,5>
<commit T2>
<end ckpt>
<start T4>
<T4,D,8>
<start ckpt(T3,T4)>
<start T5>
<T4,D,9>
<T5,A,10>
<commit T5>
<start T6>
<T6,B,12>
```

Assume further that after the crash the four elements have value: A=1, B=2, C=3, and D=4.
Recover the database. Clearly indicate which portion of the log you would need to inspect, and which transactions have to be executed again. Show the values of A, B, C, and D after the recovery.

We start from end of the log and scan up. We see an <end ckpt> and then a bit later a <start ckpt(T1,T2)> record. So we have to process from end of log to this <start ckpt(T1,T2)> record. Specifically, within this range, if a transaction has not committed, then we have to undo its actions.

These transactions are T6, T4, and T3. By undoing their actions, we got A=1, B=12, C=5, and D=8.