CIS 560, Fall 2011 **Name:______________________**
Exam 2 – 75 minutes

This test consists of questions in five categories. The number of points for each question is shown below.

- Read all questions carefully before starting to answer them.
- Write all your answers on the space provided in the exam paper.
- The order of the questions is arbitrary, so the difficulty may vary from question to question. Don't get stuck by insisting on doing them in order.
- Show your work. Correct answers without justification will not receive full credit. However, also be concise. Excessively verbose answers may be penalized.
- Clearly state any simplifying assumptions you may make when answering a questions.
- **Be sure to write your name on the test paper.**

Question	1	2	3	4	5	total
Points	20	20	20	20	20	100
Your points						

Exercise I. [True/False Questions -10 questions: you get 2 points for each correct answer; you lose 1 point for each incorrect answer; you get 0 points for questions that you don't answer]:

True/[✓ False]
The bad thing about a quiescent checkpoint is that recovery using it is much slower than it would be if we had used a non-quiescent checkpoint.

True/[✓ False]
In undo logging, it is not always possible to recover some consistent state of a database system if the system crashes during recovery.

True/[✓ False]
The weakness of redo logging is that you have to force all the dirty pages of a transaction to disk before you can write out its "COMMIT" record to the log.

[✓ True]/False
Regardless of UNDO or REDO, a logging system must write the corresponding log entry before any update of database values on disk.

[✓ True]/False
Every transaction that is strict two phase locked is also two phase locked.

[✓ True]/False
One characteristic of nested-loop joins is that you can always use them, even if you have very little memory and the data is not sorted.

[✓ True]/False
Generally speaking, a query plan that generates few intermediate result tuples is preferable to one that generates a lot of intermediate result tuples.

True/[✓ False]
A clustered index is one where each data record has an entry in the index, while in an unclustered index only some records have an entry.

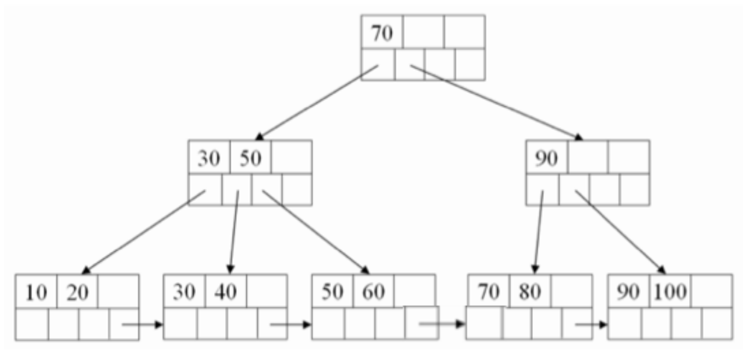
True/[✓ False]
To process a join operation, such as $R \bowtie S$, we can choose any join algorithm: nested-loop, index, sort-merge, hash – the only difference is their costs.

[✓ True]/False
Dynamic programming is a bottom-up method, where we consider only the best plan for each sub-expression of the logical-query plan.

Exercise II [B-tree indexes, 20 points]

Consider constructing a B+ tree of degree d=1.5 (that means that each index node should hold at least 2 keys and at most 3 keys).

(i) Show the resulting tree after inserting keys 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, in this order.



(ii) Using your tree in (i), show the steps for looking up all records in the range 30 to 70.

30 < 70 → follow 1st pointer

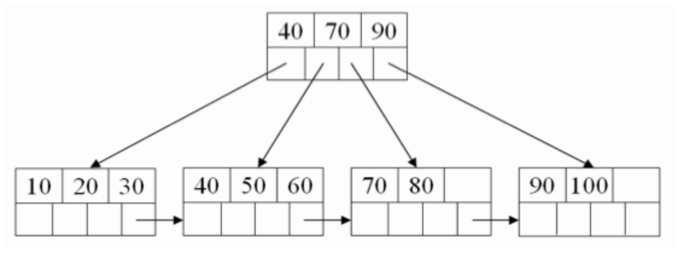
30 <= 30 < 50 → follow 2nd pointer

find 30 in the 1st key

get records with keys 30, 40, 50, 60, 70 by following links.

(iii) Would it be possible, with the same set of keys, to construct a "shorter" tree– i.e., one that has a smaller height? If no, explain why not. If yes, show an order of inserting the keys and the resulting tree.

Yes – one possible order is 10, 20, 40, 50, 70, 80, 90, 100, 30, 60



Exercise III [Selectivity and operator algorithms, 20 points]

Consider two relations R(a, b) and S(b, c) with the following statistics:

```
T(R)=10,000, B(R)=1,000 (each block contains 10 tuples)
V(R,b)=200 (number of distinct values of attribute b in R)

T(S)=6,000, B(S)=1,500 (each block contains 4 tuples)
V(S,b)=100 (number of distinct values of attribute b in S)
V(S,c)=20 (number of distinct values of attribute c in S)
```

Furthermore, the number of available memory blocks is M = 101.

Answer the following questions:

(i) Estimate the number of tuples in $\sigma_{c=150}(S)$
T(S)/V(S,c) = 6, 000/20 = 300

(ii) Assuming a join selectivity of 0.01, estimate the number of tuples in $R \bowtie S$
T(S) * T(R) * 0.01 = 10,000 * 6,000 * 0.01 = 600,000

(iii) We want to join R and S using a block-based nested loop join (without using any index). Assume R is used as the outer loop. Estimate the cost.
We shall use 100 blocks to buffer R. Thus, for each iteration, we do 100 disk I/O's to read the chunk of R and read S entirely in the second loop. So, the total number of disk I/O's is 1,000 + (1,000/100)*1,500=16,000

(iv) Now suppose we want to use sort-join. Assuming that the number of tuples with the same b value is not large, we aim to use the more efficient sort-based join approach:

a. Created sorted sublists of size M, using b as the sort key, for both R and S.
b. Bring the first block of each sublist into the memory buffer.
c. Repeatedly find the least b-value, and output the join of all tuples from R with all tuples from S that share this common b-value. If the buffer for one of the sublist is exhausted, then replenish it from disk.

Estimate the cost (total disk I/Os) of applying this sort-join on R and S. Also, state the requirement on the number of memory blocks M considering the maximum number of blocks K for holding tuples of R and S with the same b value.

Given the assumption that the number of tuples with the same b value is not large, the approach needs only 3B(R) + 3B(S) = 7,500 (the merge of sorted sublists and the join are done simultaneously).

We have 101 blocks of memory, 1000 blocks in R and 1500 blocks in S. That means we have 10 sorted sublists in R and 15 sorted sublists in S. In step b. we bring the first block of each sorted sublist in the memory, that is 25 sorted sublists. Furthermore, for each least b-value, we need to have in the memory all blocks of R and S with that b-value (i.e., K blocks). Therefore, M >= K + 25 must holds.

Exercise IV [Logical query plans, 20 points]

Consider the following query:

```
SELECT R.key, sum(T.val)
FROM R, S, T
WHERE R.key=S.fk and S.key=T.fk and R.A='abc' and T.B='cde'
GROUP BY R.key
```

The attributes R.key, S.key, T.key are keys, while S.fk and T.fk are foreign keys. Suppose that:

```
T(R) = 10,000 tuples
T(S) = 100,000 tuples
T(T) = 1,000,000 tuples
```

In each of the two cases below, show an optimal logical relational algebra plan.

Remember that a query plan that generates few intermediate result tuples is preferable to one that generates a lot of intermediate result tuples.

Thus, to optimize the plan, we need to perform the select as early as possible.

Furthermore, we need to optimize the order of the join $R \bowtie S \bowtie T$. Three possible orderings are: $(R \bowtie S) \bowtie T$, $R \bowtie (S \bowtie T)$, or $(R \bowtie T) \bowtie S$. The last option contains a cross-product $(R \bowtie T)$, and can't result in an optimal plan. Between $(R \bowtie S) \bowtie T$ and $R \bowtie (S \bowtie T)$, we need to chose the option that produces a smaller number of intermediate results.

The number of intermediate tuples for $(R \bowtie S) \bowtie T$ is equal to the size of $(R \bowtie S)$, which is $0.01 * T(R)/V(R,A) * T(S)$.

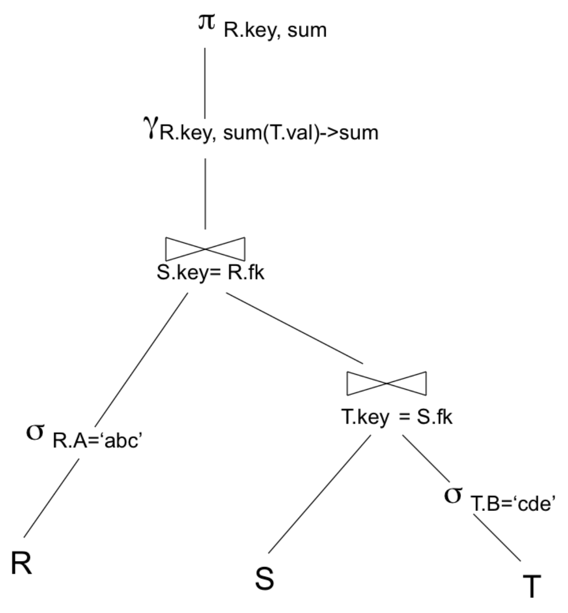
The number of intermediate tuples for $R \bowtie (S \bowtie T)$ is equal to the size of $(S \bowtie T)$, which is $0.01 * T(T)/V(T,B) * T(S)$.

(i) `V(R,A)=10, V(T,B)=10,000`

Given these statistics, the number of intermediate tuples for $(R \bowtie S) \bowtie T$ is $0.01 * 10{,}000/10 * 100{,}000 = 1{,}000{,}000$

The number of intermediate tuples for $R \bowtie (S \bowtie T)$ is $0.01 * 100{,}000 * 1{,}000{,}000/10{,}000 = 100{,}000$

Therefore, the optimal join order in this case is $R \bowtie (S \bowtie T)$. The tree is shown below.

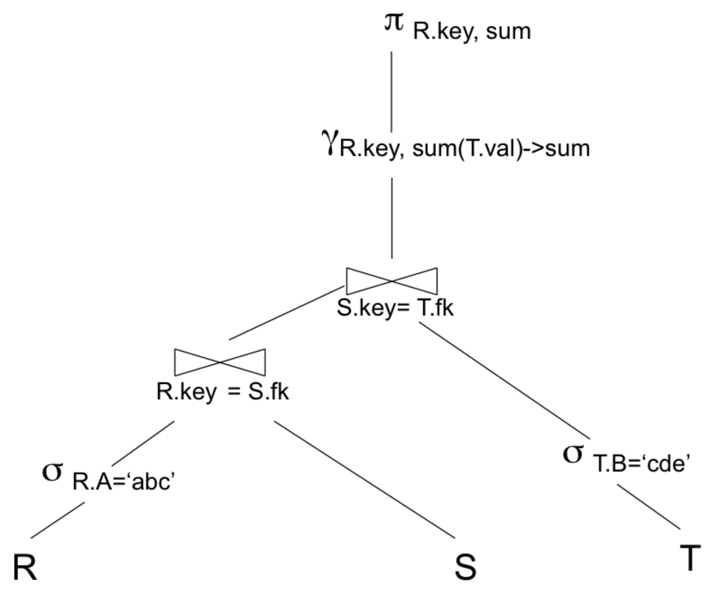


(ii) `V(R,A)=1,000, V(T,B)=100`

Given these statistics, the number of intermediate tuples for (R ⋈ S) ⋈ T is
0.01 * 10,000/1,000 * 100,000 = 10,000

The number of intermediate tuples for R ⋈ (S ⋈ T) is
0.01 * 100,000 * 1,000,000/100 = 10,000,000

Therefore, the optimal join order in this case is (R ⋈ S) ⋈ T. The tree is shown below.



Exercise V [Concurrency control, 20 points]

1. Consider a concurrency control manager by timestamps. Below are several sequences of events, including start events, where st*i* means that transaction Ti starts and co*i* means Ti commits. These sequences represent real time, and the timestamp-based scheduler will allocate timestamps to transactions in the order of their starts. In each case below, say what happens with the last request. You have to choose between one of the following four possible answers:
   i. the request is accepted,
   ii. the request is ignored,
   iii. the transaction is delayed,
   iv. the transaction is rolled back.

   a. `st1; st2; r2(A); co2; r1(A); w1(A);`
   The system will perform the following action for w1(A): rolled back.

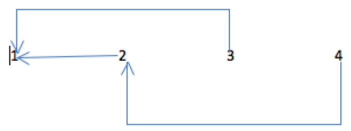
   b. `st1; st2; st3; r1(A); w1(A); r2(A);`
   The system will perform the following action for r2(A): delayed.

   c. `st1; st2; st3; r1(A); w2(A); w3(A); r2(A);`
   The system will perform the following action for r2(A): rolled back.

   d. `st1; st2; r1(A); r2(A); w1(B); w2(B);`
   The system will perform the following action for w2(B): accepted.

2. For the following schedule:
   `r1(A), r2(A), r1(B), r2(B), r3(A), r4(C), w1(A), r2(C), w2(C)`

   Answer the following questions:
   (i) What is the precedence graph for the schedule?



   (ii) Is the schedule conflict-serializable? If so, what are the equivalent serial schedules?

   There are no cycles, so the schedule is conflict serializable.

   Equivalent serial schedules: T3, T4, T2, T1 or T4, T2, T3, T1 or T4, T3, T2, T1

3. Consider the schedule S: `r1(X); r2(Y); w2(X); w1(Y).` Does a deadlock occur using the two-phase locking rule? Explain your answer.

   Yes, a deadlock can occur with such an interleaving of the actions of these transactions. T1 can gain read lock on X and T2 can gain read lock on Y, but T2 cannot gain write lock on X because it has conflict with read lock of transaction T1, so T2 has to wait. Similarly, T1 cannot gain write lock on Y due to the conflict of read lock of the transaction T1. So, T2 cannot proceed as well and they will wait forever.