Computer Science 425
Fall 2004
Second Take-home Exam
Out: noon Thursday January 13, 2005
Due: 5:00PM SHARP Monday, January 17, 2005

Instructions: This exam must be entirely your own work. Do not consult with anyone else regarding this exam. If you have questions about what is being asked, contact Professor LaPaugh or the graduate teaching assistant, Sumeet Sobti (sobti@cs.princeton.edu, Room 103C of the Computer Science Building). (Please note that Professor LaPaugh will be unavailable Monday morning; she will be available during the rest of the exam period.)

You are allowed to use the following reference materials: You may use your personal notes and problem set submissions, *Database Management Systems* by Ramakrishnan and Gehrke, the solutions to odd-numbered exercises provided by the authors at `http://www.cs.wisc.edu/~dbbook`, and copies of any of the material on the Web site for the course (staying within the site http://www.cs.princeton.edu/courses/archive/fall04/cos425/...). *No other materials are allowed.* Also you are *NOT allowed to use online database interfaces.*

There are 6 problems, with point values indicated, totaling 100 points. *Be sure to give explanations for your answers.*

On the cover page of your exam submission, write and sign the acknowledgment of original work:

"This exam represents my own work in accordance with University regulations."

Turn in your exam by 5:00pm Monday January 17 to Professor LaPaugh.

Problem 1 (20 points)

Part A In *Database Management Systems* by Ramakrishnan and Gehrke, the following algorithm is given for implementing R-S for relations R and S, which are union-compatible:

1. Sort R using the combination of all fields;
2. Sort S using the combination of all fields;
3. Scan the sorted R and S in parallel; write to the result those tuple of R that do not appear in S.

For each stage of the algorithm write its disk I/O cost and the number of buffers required. Your costs should be in terms of parameters:

- M, the number of disk pages of R
- N, the number of disk pages of S
- p, the number of records per page of R and S
- B, the number of buffer pages available in memory

Part B Now suppose that S has an Alternative 2 hash index on attribute *a*, which is a candidate key for S. Propose an algorithm for R-S that uses the hash index and analyze its disk I/O cost and buffer use. Use the parameters given in Part A. Does it matter whether attribute *a* is a candidate key for R?

Part C Suppose p=10, M=1000 N=20,000 and B=150. Which algorithm is more efficient? What change in parameter values would change which algorithm wins?

Problem 2 (20 points)
In this problem you will consider the evaluation of
((R JOIN S on R.a=S.b) JOIN T on S.c=T.d).

Suppose that

- *R* contains 40,000 tuples with 20 tuples per disk page and has an Alternative 1, clustered B+ tree on attribute *a*, which is the primary key,
- *S* contains 5,000 tuples with 15 tuples per disk page and has no indexes,
- *T* contains 40,000 tuples with 20 tuples per disk page and has an Alternative 1, clustered B+ tree on attribute *d*, which is the primary key.

Execute the dynamic programming algorithm to find an optimal left-deep plan for the evaluation of ((R JOIN S on R.a=S.b) JOIN T on S.c=T.d). **However**, only consider algorithms block-nested-loop, index-nested-loop (where applicable), and sort-merge join. Assume that both B+ trees are of height 2 and that there are 100 buffer pages. Give the disk I/O cost of each alternative and state which alternative (or alternatives) is most efficient. Also state whether each instance of a join algorithm is using pipelining and how the buffer pages are distributed during any pipelining.

Note that R and T have the same size, but you cannot assume they share any attributes or domains. All you know is that R contains primary key a and S contains an attribute b that can be compared to $R.a$, and T contains a primary key d and S contains an attribute c that can be compared to $T.d$. Assume attributes b and c of S are distinct.

Problem 3 (20 points)

Part A When using strict, conservative two-phase locking, a transaction gets all locks before accessing any database object and releases locks only when it commits. Give a sequence of reads and writes for two or more transactions that adheres to strict, conservative two-phase locking and still achieves concurrent execution of the transactions. Indicate where the lock requests, lock releases and commit actions occur in your sequence.

Part B Let T1, T2, T3 and T4 be four transactions and A, B, and C be database objects. Consider the following sequence of actions, listed in the order they are executed:

```
T1 reads A
T2 reads C
T3 reads A
T2 writes C
T4 reads C
T1 reads B
T4 writes C
T3 writes A
T4 reads B
T2 reads A
T4 writes B
T4 reads A
```

Use a precedence graph to show whether or not the execution schedule is a conflict serializable schedule. Explicitly say whether or not the schedule is conflict serializable.

Part C Let T1, T2, T3 and T4 be four transactions and A, B, and C be database objects. Consider the following sequence of actions, listed in the order they are *submitted* for execution:

```
T1 reads A
T2 reads A
T3 reads B
T3 writes B
T4 reads B
T1 reads C
T4 writes B
T3 reads C
T2 writes A
T2 writes C
T2 commits
T4 reads C
T1 commits
T3 writes C
T4 writes C
T4 commits
T3 commits
```

Add two-phase locking (not strict, not conservative) with the wait-die policy for deadlock prevention to this sequence of actions and show the actual order of execution. Show when locks are requested, when they are granted, when they are released, when a transaction blocks waiting for a lock, and when a transaction aborts due to the wait-die policy. Distinguish between shared locks and exclusive locks. When a transaction blocks waiting for a lock, assume execution proceeds with the the next action in the list above that belongs to an unblocked transaction. When a transaction aborts, assume its actions go to the end of the list of actions to be executed. The two-phase locking protocol gives some flexibility as to when locks are released. Try to release locks so as to avoid blocking or aborting transactions when possible.

Problem 4 (20 points)
Consider the following sequence of actions occurring as five transactions T1, T2, T3, T4 and T5 execute on a set of shared pages under the ARIES recovery manager protocol:

```
T1 writes page 1
T2 writes page 2
T2 writes page 1
T3 writes page 2
begin checkpoint
end checkpoint
T3 writes page 3
T4 writes page 4
T1 writes page 4
T1 begins commit
T1 ends commit
T2 begins commit
T5 writes page 1
T4 writes page 3
T5 writes page 2
```

Part A Write the transaction log for the sequence of actions above. Be sure to include prevLSN values and the contents of the transaction table and the dirty page table saved at the checkpoint.

Part B Suppose the system crashes after your log of Part A is flushed to stable storage. Execute the ARIES recovery phases. Describe all actions and show all additions to the log, including undoNextLSN values.

Part C Suppose the system crashes again during recovery from the first crash after exactly two transactions have been aborted. (The log has been flushed to stable storage and the ENDs for these aborted transactions have been written.) Execute the ARIES recovery phases after this second crash, again describing all actions and showing all additions to the log.

Problem 5 (12 points)
Consider a relational schema R with 6 attributes A, B, C, D, E and F. Answer the following questions given the functional dependences

```
A->BCDEF
BC->ADEF
E->F
EF->C
```

Part A What are the candidate keys for R?

Part B For each of the 4 functional dependences above, say whether the dependency satisfies the conditions of BCNF and of 3NF and why.

Part C What is the minimal cover for the given set of functional dependencies?

Problem 6 (8 points)
Consider a relational schema R with 4 attributes A, B, C, and D. Answer the following questions given the functional dependences

```
A->C
B->D
```

Part A Give a lossless-join dependency preserving decomposition of R into 3NF relations.

Part B Suppose the attributes A, B, C, and D all take on values that require the same amount of space to store (e.g. they are all character strings of length 100). Under what circumstances if any would the decomposition save storage space over the original relation R? Justify your answer.