CS 348
Introduction to Database Management
Assignment 4

Due: 28 November 2012 9:00AM
Returned: 10 December 2012
Appeal deadline: By the end of final exam on the 19th
Lead TA: Taras Kinash

Plase submit the answers by placing hard copy solutions in the assignment boxes.

Question 1.
Given the following histories:

$$H_1 = \{W_2(x), W_1(x), R_3(x), R_1(x), W_2(y), R_3(y), R_3(z), R_2(x)\}$$
$$H_2 = \{R_3(z), R_3(y), W_2(y), R_2(z), W_1(x), R_3(x), W_2(x), R_1(x)\}$$
$$H_3 = \{R_3(z), W_2(x), W_2(y), R_1(x), R_3(x), R_2(z), R_3(y), W_1(x)\}$$
$$H_4 = \{R_2(z), W_2(x), W_2(y), W_1(x), R_1(x), R_3(x), R_3(z), R_3(y)\}$$

(a) Which of the above histories are conflict equivalent and why?

Solution: The easiest way to reason is to create the following table that shows the conflicting operations and their ordering in each of the histories:

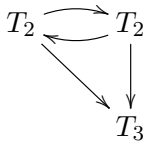
Conflicting ops	H_1	H_2	H_3	H_4
$W_2(x), W_1(x)$	$\prec$	$\succ$	$\prec$	$\prec$
$W_2(x), R_3(x)$	$\prec$	$\succ$	$\prec$	$\prec$
$W_2(x), R_1(x)$	$\prec$	$\prec$	$\prec$	$\prec$
$W_1(x), R_3(x)$	$\prec$	$\prec$	$\succ$	$\prec$
$W_2(y), R_3(y)$	$\prec$	$\succ$	$\prec$	$\prec$
$R_2(x), W_1(x)$	$\succ$	–	–	–

For any of the histories to be conflict-equivalent, they need to have identical relationships, i.e., we need to find the entries in the columns to be identical. From this table, one can see that there are no two columns that are identical. Therefore, none of these histories are conflct-equivalent.

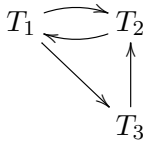
(b) Which of the above histories are serializable and why?

Solution: Again, it is best to refer to the table in Problem 1. From this table, you can reason about serializability by building serialization graphs for each history as follows.

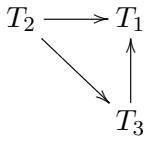
- H_1 is not serializable since it has the following serialization graph, which contains a cycle:



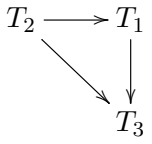
- H_2 is **not** serializable since it has the following serialization graph, which contains a cycle:



- H_3 is serializable since it it has the following serialization graph that is equivalent to the serial history $T_2 \rightarrow T_3 \rightarrow T_1$:



- H_4 is serializable since it it has the following serialization graph that is equivalent to the serial history $T_2 \rightarrow T_1 \rightarrow T_3$:



Question 2.
Question 16.7 in your book.

Solution: The answer to each question is given below.

1. Because we are inserting a new row in the table Enrolled, you may think that READ UNCOMMITTED would be sufficient. However, note that this isolation level is only allowable for read-only queries. Therefore, READ COMMITTED would need to be used.

2. Because we are updating one existing row in the table Enrolled, we need an exclusive access to the row which we are updating. So we would use READ COMMITTED.

3. To prevent other transactions from inserting or updating the table Enrolled while we are reading from it (known as the phantom problem), we would need to use SERIALIZABLE.

4. same as above.

Question 3.
Question 19.5 in your book. Please give 2-3 sentence (not longer) justification of your answer.

Solution: The answer to each case is given below:

1. 1NF. BCNF decomposition: AB, CD, ACE.
2. 1NF. BCNF decomposition: AB, BF
3. BCNF.
4. BCNF.
5. BCNF.

Question 4.
Question 20.1 in your book, but only the first part.

Solution:

- If we create a dense unclustered B+ tree index on $\langle age, sal \rangle$ of the Emp relation we will be able to do an index-only scan to answer the 5th query. A hash index would not serve our purpose here, since the data entries will not be ordered by *age*! If index only scans are not allowed create a clustered B+ tree index on just the *age* field of Emp.
- We should create an unclustered B+Tree index on *deptid* of the Emp relation and another unclustered index on $\langle dname, did \rangle$ in the Dept relation. Then, we can do an index only search on Dept and then get the Emp records with the proper *deptid*s for the second query.
- We should create an unclustered index on *ename* of the Emp relation for the third query.
- We want a clustered sparse B+ tree index on $floor$ of the Dept index so we can get the department on each floor in $floor$ order for the sixth query.
- Finally, a dense unclustered index on *sal* will allow us to average the salaries of all employees using an index only-scan. However, the dense unclustered B+ tree index on $\langle age, sal \rangle$ that we created to support Query (5) can also be used to compute the average salary of all employees, and is almost as good for this query as an index on just *sal*. So we should not create a separate index on just *sal*.