Assignment 7 (Revision A)

Reading –Read Section 6.6 and 6.7 of the revised Chapter 6 that was handed out in class, plus Sections 8.1-8.4 of Chapter 8 of the textbook.

Problem 1 (Revision A)

In the following multiversion database table, the columns Account# and Balance are accessible to user transactions. The TID and Previous columns are to support the multiversion algorithm described in the lecture notes and in Section 6.6 of the Chapter 6 handout.

TID	Previous	Account#	Balance
1	Null	298	1000
5	1	298	1100
1	Null	114	100
5	1	114	1000
7	5	114	900
8	Null	13	500

Suppose the commit list contains {1,2,5,6,7} and there are no active transactions. Now we run the following two transactions, serially:
TID=9: Insert account 200 with balance 400, and decrement the balance of account 298 by 200.
TID=10: Run a read-only query that reads all the accounts.
Then we garbage collect all the versions that aren't needed.

a. What is the state of the table after transaction 9 runs?
b. Which versions of which accounts are read by transaction 10?
c. Assuming transaction id increase monotonically with respect time, what does the table look like after the garbage collection step?

Problem 2

Consider a database consisting of one file, F. Each transaction begins by issuing a command "getlock where Q," where Q is a qualification (i.e. a Boolean formula) that's true for some records and false for others. The data manager processes the Getlock command by setting write locks on every record in F that satisfies Q (it sets no other locks). The data manager will only allow a transaction to read and modify records that were locked by its Getlock command (otherwise the read or modify operation returns an error). The transaction can insert a new record; the data manager write locks the new record just before inserting it. The data manager holds a transaction's locks until it commits. Does this locking algorithm prevent phantoms? If so, give a careful argument that every execution must be serializable. If not, show a non-serializable execution.

Problem 3

Latches are used to ensure the order of conflicting updates is reflected by the order of their corresponding log records (see pp. 247-248 of textbook, and slide 10 of DB System Recovery). The standard protocol is:

1. Fetch(P) read P into cache
2. Pin(P) ensure P isn't flushed
3. write lock (P) for two-phase locking
4. latch P get exclusive access to P
5. update P update P in cache
6. log the update to P append it to the log
7. unlatch P release exclusive access
8. Unpin(P) allow P to be flushed

Problem 3 (continued)

Briefly explain undesirable behavior, if any, that results from each of the following alterations to the protocol. Treat each alteration as a change of the standard protocol (i.e. the changes are not cumulative):

a. Switch the order of steps 2 and 3
b. Switch the order of steps 3 and 4
c. Switch the order of steps 5 and 6
d. Switch the order of steps 6 and 7
e. Switch the order of steps 7 and 8