Assignment 6

Problem 1.

Suppose a transaction sets an intention-write lock on a file and later sets a write lock on a record of the file. Is it safe for the transaction to release the intention-write lock before it commits? Why?

Problem 2.

The multi-granularity locking protocol requires that if a transaction has a *w* or *iw* lock on a data item *x*, then it must have an *iw* lock on *x*'s parent.

a. Is it correct for a transaction to hold an *r* lock on *x*'s parent instead? Either explain why it's correct or give an example where it fails.
b. Redo question (a), replacing "*r* lock" by "*w* lock".
c. Assuming the lock graph is a tree, suggest a case where it would be useful to set such a *w* lock as in (b) (whether or not it's correct).

Problem 3.

Consider the following database table, which supports a multiversion concurrency control.

TID	Previous TID	Account#	Balance
1	Null	10	100
3	1	10	200
1	Null	11	300
4	1	11	400
5	4	11	350
6	Null	12	500

Suppose the commit list contains {1,2,3,4,6} and there are no active transactions.

a. What is the state of the table after running the following transaction?:
TID=8: Increment the balance of account 10 by 100;
Delete account 12;
Insert account 13 with balance 700.

b. Suppose a read-only query with TID=7 reads all the accounts. It starts executing before executing transaction 8 starts executing and finishes after transaction 8 commits (same transaction 8 as part (a)). Which versions of which rows does it read?

c. After transactions 7 and 8 have finished and no other transactions are active, suppose we garbage collect all of the versions that aren't needed. Assuming transaction ids increase monotonically with respect time, what does the table look like after the garbage collection step?

Problem 4.

Suppose file *F* contains a sequence of fixed-length records, and *F*'s descriptor includes a *count* of the number of records in *F*, which is used to find the end of *F*. Consider the following two transactions:

- T_1:
  - Scan *F*, returning all the records in *F*
  - Read(*x*)
- T_2:
  - Insert a record into *F*
  - Write(*x*)

Data item *x* is not in F. Both transactions are two-phase locked (locking records in *F* and *x*), but neither transaction locks *count*.

a. Given an example of a non-serializable execution of T_1 and T_2. Explain why it's non-serializable.
b. Explain why this is an example of the phantom problem.