Solution to Homework #6

1. (24 points) Investigate (a) whether the default lock granularity in Oracle is at the table level or tuple level (if it is at the table level, then when one tuple of a table is locked, the entire table is locked; if it is at the tuple level, then when one tuple of a table is locked, other tuples of the tables are not locked); (b) whether deferred database modification or immediate database modification is used in Oracle (need to determine whether changes made by an uncommitted transaction have been made to the actual database); (c) whether you can still access (select) a tuple from a different session when the tuple is being modified (it is modified but the change has not been committed). Report the steps and SQL queries you used in your investigation as well as the conclusion of your investigation for each question. (Hint: Open two sessions with your Oracle for these experiments.)

Answer (examples and steps are not provided). (a) Tuple level because different tuples of the same table can be modified at the same time by operations from different sessions. (b) Deferred database modification because changes will not be visible to others (including different sessions) before a commit is issued. (c) You can still select the tuple but you cannot see the change being made before the commitment.

2. Suppose when a crash occurred, the log in the stable storage has the following records in the given order (where T1, T2 and T3 represent different transactions):

   <T1 start>, <T1,A,30,40>, <T2 start>, <T2,B,40,20>, <T1,C,25,35>, <T3 start>, <T1 commit>, <T2,A,40,60>, <T3,C,35,45>

   Answer the following questions :

   (a) (6 points) If the **deferred database modification** recovery technique is used, what should be done to T1, T2 and T3 (the choices are "no action", "redo" and "undo") during recovery? Just before the crash, what are the values of A, B, and C as can be seen by other database users with access privilege (note that only changes made by committed transactions can be seen by other users with deferred database modification)? Immediately after the recovery is completed, what are the values of A, B and C in the database?

   Answer:
   no action to T2 and T3; redo T1.
   before crash: A = 40, B = 40, C = 35
   after recovery: A = 40, B = 40, C = 35

   (b) (6 points) If the **immediate database modification** recovery technique is used and the real database is updated as soon as possible (assume that for all log records in the stable storage, the corresponding changes have been made to the real database), what should be done to T1, T2 and T3 during recovery? Just before the crash, what are the values of A, B, and C as can be seen by other database users with access privilege? Immediately after the recovery is completed, what are the values of A, B and C in the database?

   Answer:
   undo T2 and T3; redo T1.
   before crash: A = 60, B = 20, C = 45
   after recovery: A = 40, B = 40, C = 35

   (c) (4 points) If record <T1,C,25,35> is changed to <T1,B,20,50> while everything else in the log stays the same, discuss what problem this situation could cause to the recovery when **immediate database modification** is used? (You just need to identify the problem; solution is not required.)

Answer: This would mean that T1 used the result of transaction T2 on item B. Here T1 committed before T2 committed. If it turns out that T2 needs to be aborted after T1 committed, then we will not be able to erase the impact of T2 on T1, creating a serious problem.

3. (20 points) Consider the following three transactions (time goes from left to right):

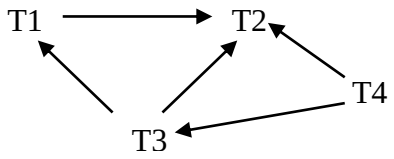
T1: R1(X) R1(Y)W1(Y)
T2: W2(X) R2(Z)W2(Z)
T3: R3(Y)W3(Y) R3(X)W3(Z)
T4: R4(Z) W4(Z)

Give a schedule that satisfies the **strict** two-phase locking (S2PL) protocol. In addition, enforce the rule that each transaction releases its locks as soon as possible without violating S2PL. Use the TSS algorithm to find out the serial schedules that are equivalent to your non-serial schedule. Show the precedence graph of your schedule.

Answer: The schedule is:

rl1(X) R1(X) wl4(Z) R4(Z) wl3(Y) R3(Y) W3(Y) W4(Z) ul4(Z) rl3(X) R3(X) wl3(Z) ul3(X) W3(Z) ul3(Y) ul3(Z) wl1(Y) ul1(X) wl2(X) W2(X) wl2(Z) R2(Z) W2(Z) ul2(X) ul2(Z) R1(Y) W1(Y) ul1(Y)

The following is the precedence graph of the above schedule:



It is easy to see that the above schedule is equivalent to the serial schedule (T4 T3 T1 T2) based on the topological sort of the above precedence graph.

4. (40 points) Assume that there are 3 data items X, Y and Z in the database. Consider the following three transactions T1, T2 and T3 with their operations coming in the given order (from left to right):

T1: R1(X) W1(X) R1(Y) W1(Y)
T2: R2(X) W2(X) R2(Z) W2(Y)
T3: R3(Y) W3(Z)

Provide the schedules that can be generated by
(a) 2PL (You may stop the scheduling when a deadlock occurs);
(b) Resource ordering (with order: Y X Z)
(c) Wound-Wait Rule
(d) Wait-Die Rule

Note that (b), (c) and (d) need to be used in combination with 2PL. It is assumed that no data items are locked initially. For (c) and (d), you are not required to show the restart of aborted transactions (however, we assume that we do not kill a transaction that has not actually started; just let it wait until the next earliest possible time to start but it should keep its original starting/arriving timestamp). Whenever possible (i.e., without violating the corresponding protocol/rule), requests should be accommodated based on first-come-first-service and locks should be released as soon as possible.

Answer:

(a) wl1(X) R1(X) rl3(Y) R3(Y) W1(X) wl3(Z) ul3(Y) wl1(Y) ul1(X) wl2(X) R2(X) W2(X) R1(Y) W3(Z) ul3(Z) rl2(Z) R2(Z) W1(Y) ul1(Y) wl2(Y) ul2(X) ul2(Z) W2(Y) ul2(Y)

(b) wl1(Y) wl1(X) R1(X) W1(X) ul1(X) R1(Y) W1(Y) ul1(Y) rl3(Y) R3(Y) wl3(Z) ul3(Y) wl2(Y) wl2(X) R2(X) W2(X) W3(Z) ul3(Z) rl2(Z) ul2(X) R2(Z) ul2(Z) W2(Y) ul2(Y)

(c) wl1(X) R1(X) rl3(Y) R3(Y) (T2-waits-for-T1-on-X) W1(X) (T1-waits-for-T3-on-Y; T1-kills-T3) ul3(Y) wl1(Y) ul1(X) wl2(X) R2(X) W2(X) rl2(Z) R2(Z) R1(Y) (T2-waits-for-T1-on-Y) W1(Y) ul1(Y) wl2(Y) ul2(X) ul2(Z) W2(Y) ul2(Y)

(d) wl1(X) R1(X) rl3(Y) R3(Y) (T2-waits-for-T1-on-X; T2 should die, however, since it has not started, we will let it wait) W1(X) (T1-waits-for-T3-on-Y) wl3(Z) ul3(Y) wl1(Y) ul1(X) (restart-T2) wl2(X) R2(X) W2(X) (T2-waits-for-T3-on-Z; T2 dies) ul2(X) R1(Y) W3(Z) ul3(Z) W1(Y) ul1(Y)