Reminders

• Weight: 5% of the homework grade.
• Out of 100 points.
• Please type your answers. Illegible handwriting may get no points, at the discretion of the grader. Only drawings may be hand-drawn, as long as they are neat and legible.
• Estimated time to solve the assignment: 5-8 hours.

Remember that:
• There could be more than one correct answer. We shall accept them all.
• Whenever you are making an assumption, please state it clearly.

Problem 1: Serializability [10pts]

Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.

1. **S1:** T1:R(X), T2:R(X), T1:W(Y), T2:W(Y), T1:R(Y), T2:R(Y)
2. **S2:** T3:W(X), T1:R(X), T1:W(Y), T2:R(Z), T2:W(Z), T3:R(Z)

For each of the schedules, answer the following questions:

• What is the precedence graph for the schedule?

• Is the schedule conflict-serializable? If so, what are all the conflict equivalent serial schedules?

• Is the schedule view-serializable? If so, what are all the view equivalent serial schedules?
Problem 2: Two-phase locking [16pts]

Consider the following schedules.

1. **S1:** T1:R(X), T2:W(Y), T2:R(X), T1:W(Y), T1:Commit, T2:Commit

2. **S2:** T1:R(X), T2:R(Y), T1:W(Z), T1:Commit, T3:R(Y), T3:R(Z), T2:W(Y), T3:W(X), T2:Commit, T3:Commit

For each schedule state which of the following concurrency control protocols allows it, that is, allows the actions to occur in exactly the order shown:

- 2PL
- Strict 2PL

Please provide a brief explanation for your answer.

Problem 3: [24pts]

In this question you will be given a brief description of different relaxed locking modes. Please circle all TRUE statements in each part, and give a brief reasoning for your answers. There may be zero, one, two or more true statements in each part.

1. Transaction running in “Read Uncommitted” mode always read data without setting any locks in the lock table. Read Uncommitted transactions always set exclusive locks before writing as usual, and hold them until end-of-transaction. Said differently, Read Uncommitted transactions use **No S-Locks, and Strict 2-Phase X-Locks**. Which of the following conflicts can occur between a Read Uncommitted transaction, and a transaction following the Strict 2PL locking protocol. We define two transactions as conflicting if they access the same object and at least one of them wants to write, and none of the transactions have committed yet (the last condition is not given in the book definition). Read page 526-528 for details.

   (a) Write-Write
   (b) Read-Write
   (c) Write-Read

2. Transaction running in “Read Committed” mode obtains locks in the same way as Strict 2PL, and release exclusive locks at the end-of-transaction just like strict 2PL. However, Read Committed transaction release **shared** locks immediately after finishing a read. Said differently, Read Committed transactions use **Short-term S-Locks, and Strict 2-Phase X-Locks**. Which of the following conflicts can occur between a Read Committed transaction, and a transaction following the Strict 2PL locking protocol.

   (a) Write-Write
   (b) Read-Write
(c) Write-Read

3. Consider what would happen if a transaction released a shared lock immediately after a read, and released an exclusive lock immediately after a write (Short-term S-Locks, and Short-term X-Locks). Which of the following conflicts could happen in those cases?

(a) Write-Write conflicts
(b) Inconsistent data in the database
(c) Deadlocks

Problem 4: Hierarchical locking [16pts]

Consider the following database schema:
STUDENT(name, id, gpa, dept)
ENROLL (course_id, student_id)

Suppose that the database is organized in terms of the following hierarchy of objects: The database itself is an object (D), and it contains two files (STUDENT and ENROLL). The STUDENT file contains 200 pages (S1....S200) while the ENROLL file contains 2000 pages (E1....E2000). Each page contains 100 records, and records are identified as p:i, where p is the page identifier and i is the slot of the record on that page.

Multiple granularity locking is used, with S, X, IS, IX and SIX locks, and database-level, file-level, page-level and record-level locking. For efficiency reasons assume that if 60% of the children require a certain lock, then the DBMS obtains that lock on the parent node itself (thus, it does not need repeating the same operation on the children).

For each of the following operations, indicate the sequence of lock and unlock requests that must be generated by a transaction that wants to carry out (just) these operations:

- **Transaction 1:** Read record E600:4.
- **Transaction 2:** Read records S20:20 through S26:20.
- **Transaction 3:** Read all records of file STUDENT.
- **Transaction 4:** Read pages E10 through E80 (included).
- **Transaction 5:** Read pages S10 through S80 (included).
- **Transaction 6:** Read all pages in STUDENT and (based on the values read) modify 10 pages.
- **Transaction 7:** Update record S10:40.
- **Transaction 8:** Update all pages in ENROLL.
Problem 5: Deadlocks [24pts]

Consider the following sequence of actions, listed in the order it is submitted to the DBMS:

- **Original sequence:** T1:R(A), T2:W(A), T2:W(B), T3:W(B), T1:W(B), T1:Commit, T2:Commit, T3:Commit

Problem 5.1: Deadlock Detection [7pts]

For the sequence given above, describe how “Strict 2PL with deadlock detection” will handle it. (Show the waits-for graph in case of deadlock.) Add the lock requests as per the locking protocol. For example, your solution may look like T1:S(A), T1:R(A).....deadlock, where T1:S(A) means T1 obtains shared lock for object A.

Assumptions/reminders: The DBMS processes actions in the order shown in the sequence. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the original sequence) of an unblocked transaction.

Problem 5.2: Deadlock Prevention [17pts]

For the sequence given above and each concurrency control mechanism below, match the schedules which can result from use of the concurrency control mechanism on the sequence (more than one final schedule is possible).

- **Concurrency scheme:**
  
  A: Strict 2PL with timestamps used for deadlock prevention (wound-wait).
  
  B: Strict 2PL with timestamps used for deadlock prevention (wait-die).

- **Final schedules:**

  1. T1:S(A), T1:R(A), T3:X(B), T3:W(B), T3:Abort, T1:X(B), T1:W(B), T1:Commit, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit, T3:Restart, T3:X(B), T3:W(B), T3:Commit
  2. T1:S(A), T1:R(A), T1:Abort, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit, T3:X(B), T3:W(B), T3:Commit, T1:Restart, T1:S(A), T1:R(A), T1:X(B), T1:W(B), T1:Commit
  3. T1:S(A), T1:R(A), T1:X(B), T1:W(B), T1:Commit, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit, T3:X(B), T3:W(B), T3:Commit
  4. T1:S(A), T1:R(A), T2:Abort, T3:X(B), T3:W(B), T3:Commit, T1:X(B), T1:W(B), T1:Commit, T2:Restart, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit
  5. T1:S(A), T1:R(A), T3:X(B), T3:W(B), T3:Abort, T1:X(B), T1:W(B), T1:Commit, T3:Restart, T3:X(B), T3:W(B), T3:Commit, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit
6. T1:S(A), T1:R(A), T3:X(B), T3:W(B), T3:Abort, T1:X(B), T1:W(B), T1:Commit, T2:X(A), T2:W(A), T3:Restart, T3:X(B), T3:W(B), T3:Abort, T2:X(B), T2:W(B), T2:Commit, T3:Restart, T3:X(B), T3:W(B), T3:Commit

7. T1:S(A), T1:R(A), T2:Abort, T3:X(B), T3:W(B), T3:Commit, T2:Restart, T2:Abort, T1:X(B), T1:W(B), T1:Commit, T2:Restart, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit

8. T1:S(A), T1:R(A), T2:Abort, T3:X(B), T3:W(B), T3:Abort, T1:X(B), T1:W(B), T1:Commit, T2:Restart, T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit, T3:X(B), T3:W(B), T3:Commit

Hints: Your solution should look something like this:

- Concurrency scheme A matches to the final schedules 1, 2 and 4.

- Concurrency scheme B matches to none of the given schedules. A final schedule that is possible under scheme B is T1:S(A), T1:R(A)...

Assumptions/reminders: the timestamp of transaction Ti is i. (The lower the timestamp, the higher is the transaction’s priority.) For each concurrency control mechanism, the DBMS starts with the original sequence of actions, and adds lock and unlock requests as per the locking protocol. The DBMS processes actions in the sequential order. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the original sequence) of an unblocked transaction.

An aborted/blocked transaction can be restarted/resumed later at any time. When restarted its actions arrive in the same order as in the original sequence (e.g., T1:R(A) comes before T1:W(B), T2:W(A) comes before T2:W(B)), but they can be arbitrarily interleaved with the other transactions (e.g., if both T1 and T2 are aborted, then when they are restarted, T1:R(A) may arrive before or after T2:W(A)).

Problem 6: Z-ordering [10pts]

We want to plot the z-curve for an 8x8 grid. Write the code (in C/C++ or Java) and show the resulting curve, in an 8x8 grid. Submit the code (hard-copy) as well as the curve.

Clarification: the first cell is the (0,0) cell and the second is the (0,1) cell (the coordinates in each dimension go from 0 to 7, included).