1. Concurrency Control:

(a) What is the significance of imposing the following constraints on histories: i. serializability, ii. avoiding cascading abort, iii. strictness. You can use examples to illustrate your arguments.

Solution is ignored.

(b) Consider a database with objects X and Y and assume that there are two transactions T1 and T2. Transaction T1 reads objects X and Y and then writes object X. Transaction T2 reads objects X and Y and then writes objects X and Y. Give three example histories over transactions T1 and T2 that result in a write-read conflict, a read-write conflict and a write-write conflict respectively.

Solution:
1. The following schedule results in a write-read conflict:
   T2:R(X), T2:R(Y), T2:W(X), T1:R(X) ...
   T1:R(X) is a dirty read here.
2. The following schedule results in a read-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X) ...
   Now, T2 will get an unrepeatable read on X.
3. The following schedule results in a write-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:W(X) ...
   Now, T2 has overwritten uncommitted data.

Show that each serial history involving transactions T1 and T2 preserves the consistency requirement of the database.
Solution:

T1 T2: R1X, R1Y, W1X, c1, R2X, R2Y, W2X, W2Y, c2

T2 T1: R2X, R2Y, W2X, W2Y, c2, R1X, R1Y, W1X, c1

(c) i. Give an example of two conflict-equivalent, but different, non-serializable schedules.
T1: RX, WY
T2: RY, WX
History1: R1X, R2Y, W2X, W1Y, c1, c2
History2: R1X, R2Y, W1Y, W2X, c1, c2

and they are non-serializable.

ii. Give an example of a serializable schedule which is conflict-equivalent with two different serial schedules.
T1: RX WY
T2: RX WZ

H: R1X, W1Y, R2X, W2Z, c1, c2.

Any schedules that don’t have conflict pairs are conflict-equivalent with serial schedules.

(d) Consider the following transactions:

T1: read(A);
    read(B);
    if A = 0 then B := B+1;
    write(B).

T2: read(B);
    read(A);
    if B = 0 then A := A+1;
    write(A).

Let the consistency requirement be A = 0 ∨ B = 0, and let A = B = 0 be the initial values.
i. Construct a history on T1 and T2 that produces a non-serializable history.
   T1: R1A, R1B, W1B
   T2: R2B, R2A, W2A
   H: R1A, R2B, R1B, R2A, W1B, W2A, c1, c2.

ii. Is there a non-serial history on T1 and T2 that produces a serializable history. If so, give an example.
   no non-serial history that is serializable.

iii. A. Add lock and unlock instructions to T1 and T2, so that they observe the two-phase locking protocol, but in such a way that interleaving between operations in T1 and T2 is still possible.
   T1: r1A, R1A, r1B, R1B, w1B, W1B, ru1A, ru1B, wu1B.
   If the operations do not include lock/unlock, the operations cannot be interleaved if we don’t change the order of R2B and R2A. If we change the order of R2B and R2A, the history is:
   H: r1A, R1A, r1B, R1B, r2A, R2A, w1B, W1B, ru1A, ru1B, wu1B, r2B, R2B, w2A, W2A, ru2B, ru2A, wu2A, c1, c2
   If the operations include lock/unlock, the history is:
   H: r1A, R1A, r1B, R1B, w1B, W1B, ru1A, ru1B, wu1B, r2B, R2B, r2A, R2A, w2A, W2A, ru2B, ru2A, wu2A

B. Can the execution of these transactions result in a deadlock? If so, give an example.
   yes.
   H: r1A, r1B, R1A, R1B, r2B, r2A, w1B, R2B, R2A, w2A...
circle: $rl_1 A, wl_2 A$ and $rl_2 B, wl_1 B$

(e) Define a lock point of a transaction to be any moment at which it owns all of its locks; that is, it is a moment after it has performed its last lock operation and before it has released any lock. Using serializability theory, prove that for every history $H$ produced by a 2PL scheduler there is an equivalent serial history in which transactions are in the order of their locked points in $H$.

It equals to the statement that a $H$ produced by a 2PL scheduler is equivalent to a serial history $H'$ that orders transactions based on the order of locked points in $H$.

proof:
Any RR pairs don’t conflict with each other.

For any WW pairs in $H$, if $W_i X$ is before $W_j X$ where $i \neq j$, $lw_j X$ conflicts with $lw_i X$. In 2PL scheduler $lw_j X$ must be delayed after $wu_i X$ that is after the locked point of transaction $i$. So the locked point of transaction $i$ is before the locked point of transaction $j$. Thus the transaction $i$ is before transaction $j$ in the $H'$ because they are ordered based on the order of locked points. Obviously $W_i X'$ is before $W_j X'$ in $H'$.

For any WR/RW pair, the proof is similar.

For each conflicting pairs in $H$, they have the same order in $H'$. So $H$ is equivalent to $H'$.

(f) Exercise 17.2 in Cow Book.
<table>
<thead>
<tr>
<th></th>
<th>SR</th>
<th>RC</th>
<th>ACA</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>undecidable</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>undecidable</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>undecidable</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>7</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>8</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>9</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>10</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

2. **Recovery**

(a) Excercise 18.5 in Cow Book.

 ignored. Can be found from the online solution.

(b) In logical logging, suppose each update record describes an operation that is applied to at most one data item. Suppose we implement undo and redo procedures for all operations so that for each log record LR on data item X, undo(LR) has no effect if X does not include LR’s update, and redo(LR) has no effect if X already includes LR’s update. Does Restart in the partial data item logging algorithm work correctly on a log with this structure? That is, assuming fuzzy checkpointing, is it correct to undo all uncommitted updates during a backward scan of the log, and then redo all committed updates during a forward scan? If so, argue the correctness of the algorithm. If not, explain why.

Yes. The partial data item logging algorithm is known to be correct for a log in which correctly restoring a before image or reapplying an after image does not depend on the current state of the data item. In the given technique, the correctness of the undo and redo procedures also does not depend on the current state of the item being undone or redone. Therefore the given technique also works with partial data item logging. (from Lindsey’s solution.)
(c) Specify a recovery strategy which would never require undoing or redoing side-effects (both undoing and redoing are not required). How would such a recovery strategy be implemented?

can be found in chapter 6 of online book.

(d) Specify a recovery strategy which would never require re-doing side-effects of write operations caused by committed transactions (only redoing is not required while undoing is required). How would such a recovery strategy be implemented?

can be found in chapter 6 of online book.

(e) Specify a recovery strategy which would never require undoing (only undoing is not required while redoing is required). How would such a recovery strategy be implemented?

(can be found in chapter 6 of online book.

(f) Suppose that there is a database system that never fails. Is a concurrency control scheduler required for this system? Is a recovery manager required for this system? Don’t just answer “yes” or “no”. Justify you answers. (Hint: consider this question in the context of the ACID requirements associated with transaction management.)

The concurrency control scheduler is required because it keeps the database consistency.
The recovery manager is required because transactions can be aborted due to the deadlock detection, missing data and so on.