1. Consider a procedure that assigns seats for airline travel. Assume that multiple versions of this procedure can execute concurrently.

(a) A schedule for \( n \) concurrently executing procedures is an \( n \)-column list in which only one column has an entry per line. An entry represents the execution of a statement. The sequence of statements, line-by-line, from top to bottom represents the interleaving of statement execution among the \( n \) concurrently executing procedures. Give a schedule that would (incorrectly) allow two passengers to be assigned to the same seat on the same flight. (Assume that the database stores seat assignments as records with NULLs for the passenger ID when no one is assigned to a seat and that the database replaces the NULLs with passenger IDs when it assigns seats.)

(b) Locking can prevent double seat assignments. Locking granularity is the level at which a database applies locks—lock the entire database, lock a relation in the database, lock a designated set of tuples of a relation, lock a single tuple, lock a designated set of columns of a relation, lock a single column, lock a combination of tuples and columns. Locking at the lowest granularity allows the greatest amount of concurrency. For a procedure that assigns seats for airline travel, what should the locking granularity be to allow the greatest amount of concurrency?

(c) Allowing "dirty reads" can increase concurrency. Dirty data is data written by a transaction that has not yet been committed. A dirty read is a read of dirty data. The problem with dirty reads is that a transaction that has written data may abort (e.g. it may later violate an integrity constraint and thus abort) and any data it has written may be rolled back. Thus, any action that reflects knowledge of dirty data may be incorrect. Suppose we allow the transaction that assigns seats for airline travel to do dirty reads of data written by other invocations of the same transaction, but we only allow writes when the transaction obtains a lock on the data. Describe the anomalies that might occur. Are these anomalies bad enough that the seat-assignment transaction should not be allowed to do dirty reads? Justify your answer.

(d) The execution of procedures operating on the same database is serial if one procedure executes completely before any other procedure begins. The execution is serializable if procedures operating on the same database behave as if they were run serially, even though their executions may overlap in time. Suppose two seat-assignment transactions are running concurrently under the two-phase locking protocol, which guarantees serializability, and suppose that there is only one seat left. The serializability property does not decide which transaction "wins" and claims the seat for its customer. Explain how it is decided.

(e) A transaction is atomic in the sense that either all of the actions of the transaction complete or none of them complete. Consider a transaction that records airline ticket purchase information in one table and seat assignments in another table. What anomalous situations could arise if the transactions were not treated as being atomic.
2. Using logging with immediate updates, assume that the following log records are on disk at the time the system crashes. We are using our class BandB database where Room is a relation or file, Cost and Name are attributes or fields, and RoomNr with rooms 1, ..., 5 is the primary key.

\[
<T1 \text{ starts}> \\
<T1, \text{Room, Cost}, 1, 90, 95> \\
<T2, \text{starts}> \\
<T1, \text{Room, Cost}, 2, 80, 85> \\
<T2, \text{Room, Cost}, 4, 60, 70> \\
<T2, \text{commits}> \\
<T1, \text{Room, Cost}, 3, 80, 85> \\
<T1, \text{Room, Cost}, 4, 70, 75> \\
<\text{checkpoint}> \\
<T1, \text{Room, Cost}, 5, 60, 65> \\
<T1 \text{ commits}> \\
<T3 \text{ starts}> \\
<T3, \text{Room, Name}, 4, \text{Blue, Gold}> \\
<T4 \text{ starts}> \\
<T4, \text{Room, Name}, 1, \text{Kennedy, Clinton}> \\
<T4, \text{Room, Name}, 2, \text{Nixon, Bush}> \\
<T3, \text{Room, Name}, 5, \text{Green, Red}> \\
<T4, \text{Room, Name}, 3, \text{Carter, Reagan}> \\
<T4 \text{ commits}> \\
\]

When the system recovers:

(a) Which transactions need to be redone (i.e. have values replaced by new values using the log)?

(b) Which transactions need to be undone (i.e. have values replaced by old values using the log)?

(c) What are the five cost values and the five names for the five rooms?

3. As an alternative to the crash-recovery scheme discussed in class consider the following scheme, which we call log with deferred modifications. In this log-with-deferred-modifications crash-recovery scheme, we record \text{starts} and \text{commits} entries in the log as before, but we do not record old values in log entries for updates. Thus the format for an update log entry is

\[
<T, f, F, K, \text{new value}> \\
\]

where \(T\) is the transaction identifier, \(f\) is the file identifier, \(F\) is the field identifier, and \(K\) is the primary-key value that identifies the updated record. The \text{new value} is the new value recorded when \(T\) executes. In the log-with-deferred-modifications scheme, we create log entries as we execute write statements internally but defer all actual writes to the database until the transaction commits and the \text{commits} log entry, as well as any previous entries not already written to the disk, have been stored on disk.

(a) Explain how we recover for a transaction \(T\) if there is a crash after \(<T, \text{starts}>\) has been logged on disk along with several update entries but before \(<T, \text{commits}>\) has been logged on disk.

(b) Explain how we recover for a transaction \(T\) if there is a crash after \(<T, \text{commits}>\) has been logged on disk. (Assume that there are no \(<\text{checkpoint}>\) entries.)
(c) Give an example to show how an inconsistent database state could be the result of this recovery scheme if writes are not deferred, that is, if we write an updated block to disk before committing the transaction.

4. Consider the following transactions applied to the Room relation in our class database. (Assume Room is also the file name, and let the Read and Write pseudo code denote simple gets and puts of the values indicated by the variable names.)

\[ T_0: \, \text{Read}(\text{Cost}_{\text{GreenRoom}}) \]
\[ \text{Cost}_{\text{GreenRoom}} := \text{Cost}_{\text{GreenRoom}} + 10 \]
\[ \text{Write}(\text{Cost}_{\text{GreenRoom}}) \]
\[ T_1: \, \text{Read}(\text{Cost}_{\text{GreenRoom}}) \]
\[ \text{Cost}_{\text{GreenRoom}} := \text{Cost}_{\text{GreenRoom}} + 5 \]
\[ \text{Write}(\text{Cost}_{\text{GreenRoom}}) \]
\[ \text{Read}(\text{NrBeds}_{\text{GreenRoom}}) \]
\[ \text{NrBeds}_{\text{GreenRoom}} := \text{NrBeds}_{\text{GreenRoom}} + 1 \]
\[ \text{Write}(\text{NrBeds}_{\text{GreenRoom}}) \]

Let \( \text{Cost}_{\text{GreenRoom}} \) initially be 50, and let \( \text{NrBeds}_{\text{GreenRoom}} \) initially be 1. Assume that \( T_0 \) executes, then \( T_1 \) executes.

(a) If the recovery scheme is using a log with deferred modifications, show the state of the log if the system crashes just after “\text{Write}(\text{NrBeds}_{\text{GreenRoom}})” but before the transaction commits. Assume all possible log entries have been written to disk. (Be sure to express log entries for values as proper 5-tuples and don’t forget the \text{starts} and \text{commits} entries.)

(b) Explain how the system recovers in terms of any and all undo’s, redo’s and restart’s properly ordered.

(c) What is the cost value for the Green room just after the system recovers?

5. Consider the following transactions

\[ T_0: \, \text{Read}(A) \]
\[ A := A + 200 \]
\[ \text{Write}(A) \]
\[ T_1: \, \text{Read}(A) \]
\[ A := A - 100 \]
\[ \text{Write}(A) \]
\[ \text{Read}(B) \]
\[ B := B + 100 \]
\[ \text{Write}(B) \]

(a) Add LX and UN instructions to transactions \( T_0 \) and \( T_1 \) so that they observe 2PLP.

(b) Add LX and UN instructions to transaction \( T_1 \) so that they are valid, but violate 2PLP.

(c) For this example, does it matter whether the locking protocol is 2-Phase? Explain.