Problem 1

Consider the following log file (to be read from left to right):

$< \mathtt{START}\ \mathtt{T}_1 >$ $< \mathtt{T}_1, \mathtt{A}, 5 >$ $< \mathtt{T}_1, \mathtt{B}, 3 >$ $< \mathtt{START}\ \mathtt{T}_2 >$ $< \mathtt{T}_2, \mathtt{B}, 6 >$

$< \mathtt{START}\ \mathtt{T}_3 >$ $< \mathtt{T}_3, \mathtt{B}, 18 >$ $< \mathtt{T}_2, \mathtt{D}, 0 >$ $< \mathtt{T}_1, \mathtt{A}, 10 >$ $< \mathtt{COMMIT}\ \mathtt{T}_1 >$

$< \mathtt{T}_3, \mathtt{C}, 5 >$ $< \mathtt{COMMIT}\ \mathtt{T}_3 >$ $< \mathtt{START}\ \mathtt{T}_4 >$ $< \mathtt{T}_4, \mathtt{A}, 20 >$ $< \mathtt{T}_2, \mathtt{E}, 0 >$

$< \mathtt{COMMIT}\ \mathtt{T}_2 >$ $< \mathtt{T}_4, \mathtt{F}, 0 >$ $< \mathtt{COMMIT}\ \mathtt{T}_4 >$

Suppose that we start a Nonquiescent Checkpoint under an UNDO logging technique immediately after one of the following log records has been written:

1. $< \mathtt{T}_1, \mathtt{B}, 3 >$;
2. $< \mathtt{T}_2, \mathtt{B}, 6 >$.

For each of the above points, complete the log file with the insertion of START CKPT and END CKPT log records.

Nonquiescent Checkpointing

The steps in a *nonquiescent checkpointing* are:

1. Write $<$START CKPT($T_1; \ldots; T_k$)$>$ record to log, where T_i are all the active (uncommitted) transactions;
2. Wait until all transactions $T_1; \ldots; T_k$ commit or abort, *but do not prohibit new transactions*;
3. Write $<$END CKPT$>$ record to log, and flush the log.

If recovery is necessary, we know that all transactions prior to a recorded checkpoint have committed and need not be undone.

Problem 2

Consider now the log file obtained solving point 2 of Problem 1.

Suppose there is a crash. Using the primitive actions: `WRITE, OUTPUT, ABORT(T)`, give the actions that need to be done to recover from the crash in each of the following situations:

1. The last record in the log file is $< \mathtt{T_4, F, 0} >$.
2. The last record in the log file is $< \mathtt{COMMIT\ T_1} >$.

Furthermore, for each of the above points, give the new log file after deleting useless portions.

Recovery With Nonquiescent Checkpoints

To recover a crash using an Undo Nonquiescent Checkpoints: Scan the log from the end:

- If we first meet $<\mathtt{END\ CKPT}>$ then we can restrict to transactions that began after the $<\mathtt{START\ CKPT}(\mathtt{T}_1;\ldots;\mathtt{T}_k)>$. The log before $<\mathtt{START\ CKPT}(\mathtt{T}_1;\ldots;\mathtt{T}_k)>$ is useless and can be deleted.

- If we first meet $<\mathtt{START\ CKPT}(\mathtt{T}_1,\ldots,\mathtt{T}_k)>$, then the crash occurred during the checkpoint. We need to undo:

  1. All those transactions $\mathtt{T}$ with $<\mathtt{START\ T}>$ after the $<\mathtt{START\ CKPT}>$ but no $<\mathtt{COMMIT\ T}>$;

  2. All transactions $\mathtt{T}_i$ on the list associated with $<\mathtt{START\ CKPT}>$ with no $<\mathtt{COMMIT\ T}_i>$. The log before the start of the earliest of these incomplete transactions can be deleted.

Problem 3

The following is a sequence of redo-log records written by three transactions T, U, V:

$< \mathtt{START\ T} >< \mathtt{T, A, 10} >< \mathtt{START\ U} >< \mathtt{U, B, 20} >$

$< \mathtt{T, C, 30} >< \mathtt{START\ CKPT(T, U)} >< \mathtt{U, D, 40} >< \mathtt{COMMIT\ U} >$

$< \mathtt{T, E, 50} >< \mathtt{START\ V} >< \mathtt{V, C, 45} >< \mathtt{END\ CKPT} >$

$< \mathtt{COMMIT\ V} >< \mathtt{T, D, 45} >$

Suppose there is a crash and the log file is as above. Using the primitive actions: `WRITE, OUTPUT, ABORT(T)`, give the actions that need to be done to recover from the crash using a REDO-Logging technique.

Recovery for Redo Logging

- While incomplete transactions did not change the DB, committed transactions cause problems since we don't know which of their changes have been written to disk.

- **To recover a crash using a Redo-Logging we do:**

  1. Find the set of committed transactions from the log;
  2. Scan the log *forward* from the beginning and for each $<\mathtt{T}, \mathtt{X}, \mathtt{v}>$ do:
     - If T is committed, write value v for X to disk, i.e., do:
       WRITE(X, v); OUTPUT(X);
     - Otherwise, do nothing.
  3. For each incomplete transaction T add $<\mathtt{ABORT\ T}>$ to the log, and flush the log.