Undo Logging Rules

Undo 1:

If transaction T modifies the database element X that held value OLD

- Write $\langle T, X, \text{OLD} \rangle$ to the log
- *Only* when the log record appears on disk can we write the new value for X to disk.

Undo 2:

If transaction T commits, then

- Write *all* pages with modified database elements to disk
- *Then*, write $\langle \text{COMMIT}\, T \rangle$ to the log and disk, as soon as possible.

Redo Logging Rules

Redo 1:

If transaction T modifies the database element X setting its value to NEW

- Write $\langle T, X, \text{NEW} \rangle$ to the log

Redo 2:

If transaction T commits, then

- Write $\langle \text{COMMIT}\, T \rangle$ to the log, and flush the log to *the disk*.
- Only *then*, write the new value for X to disk.

Hence, all log entries must be written to disk, before modifying any database element on disk.

Undo/Redo Logging Rules

Undo/Redo 1:

If transaction T modifies database element X that held the value OLD to the value NEW

- Write $\langle T, X, \text{OLD}, \text{NEW} \rangle$ to the log
- Log records must be flushed to disk before corresponding modified pages are written to disk.
- When the transaction commits, write $\langle \text{COMMIT}\, T \rangle$ to the log and flush the log.
- Modified database pages can be flushed before or after commit.

Database System Recovery

Task:

Consider the following log:

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle T, B, 20, 21 \rangle \langle \text{COMMIT } T \rangle$$

Tell all sequences of events that are legal for an Undo-, Redo-, and Undo/Redo-based recovery system, where the events are:

$$\text{Output}(A), \text{Output}(B), \text{Flush-Log}(A), \text{Flush-Log}(B), \text{Commit}$$

Note that for convenience of presentation, we only use Undo/Redo log events in these slides!

Database System Recovery

Solution (Undo)

The constraints are:

- Flush-Log(A) $<$ Output(A)
- Flush-Log(B) $<$ Output(B)
- Flush-Log(A) $<$ Flush-Log(B) $<$ Commit
- Output(A) $<$ Commit
- Output(B) $<$ Commit

Hence, the valid sequences are:

- Flush-Log(A), Output(A), Flush-Log(B), Output(B), Commit
- Flush-Log(A), Flush-Log(B), Output(A), Output(B), Commit
- Flush-Log(A), Flush-Log(B), Output(B), Output(A), Commit

Database System Recovery

Solution (Redo)

The constraints are:

- Flush-Log(A) $<$ Output(A)
- Flush-Log(B) $<$ Output(B)
- Flush-Log(A) $<$ Flush-Log(B) $<$ Commit
- Commit $<$ Output(A)
- Commit $<$ Output(B)

Hence, the valid sequences are:

- Flush-Log(A), Flush-Log(B), Commit, Output(A), Output(B)
- Flush-Log(A), Flush-Log(B), Commit, Output(B), Output(A)

Database System Recovery

Solution (Undo/Redo)

The constraints are:

- Flush-Log(A) $<$ Output(A)
- Flush-Log(B) $<$ Output(B)
- Flush-Log(A) $<$ Flush-Log(B) $<$ Commit

Hence, the valid sequences are:

- Flush-Log(A), Output(A), Flush-Log(B), Output(B), Commit
- Flush-Log(A), Flush-Log(B), Output(A), Output(B), Commit
- Flush-Log(A), Flush-Log(B), Output(B), Output(A), Commit
- Flush-Log(A), Flush-Log(B), Commit, Output(A), Output(B)
- Flush-Log(A), Flush-Log(B), Commit, Output(B), Output(A)
- Flush-Log(A), Output(A), Flush-Log(B), Commit, Output(B)
- Flush-Log(A), Flush-Log(B), Output(A), Commit, Output(B)
- Flush-Log(A), Flush-Log(B), Output(B), Commit, Output(A)

Database System Recovery

Task:

Consider the following log, after a crash:

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

- What values might/must have been changed?
- How does the recovery manager get the database back to a consistent state?

Discuss for Undo-, Redo-, and Undo/Redo-logging.

Database System Recovery

Solution (Undo)

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

We first identify the transactions that we need to undo. They are T, and U.

By reading the log we can conclude that:

- A *might* have had its value changed on disk.

Database System Recovery

Solution (Undo)

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

Starting from the *end* of the log, we undo as follows:

- Append $\langle \text{ABRT } U \rangle$ to the log.
- Write value 10 for A.
- Append $\langle \text{ABRT } T \rangle$ to the log.

Database System Recovery

Solution (Redo)

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

We first identify the transactions that we need to redo. No transaction has committed, so we do not need to redo any transaction.

By reading the log we can conclude that:

- A *cannot* have had its value changed on disk.

Database System Recovery

Solution (Redo)

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

We do not need to redo any transaction.

- Append $\langle \text{ABRT } U \rangle$ to the log.
- Append $\langle \text{ABRT } T \rangle$ to the log.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START } T \rangle \langle T, A, 10, 11 \rangle \langle \text{START } U \rangle$$

We first identify the transactions that we need to undo or redo. No transaction has committed, so we do not need to redo any transaction. We need to undo transactions T and U.

By reading the log we can conclude that:

- A *might* have had its value changed on disk.

We recover from the crash as with Undo.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START}\, T \rangle \langle T, A, 10, 11 \rangle \langle \text{START}\, U \rangle$$

We do not need to redo any transaction.

Starting from the *end* of the log, we undo as follows:

- Append $\langle \text{ABRT}\, U \rangle$ to the log.
- Write value 10 for A.
- Append $\langle \text{ABRT}\, T \rangle$ to the log.

Database System Recovery

Task:

Consider the following log, after a crash:

$$\langle \text{START } T\rangle \langle T, A, 10, 11\rangle \langle \text{START } U\rangle \langle U, B, 20, 21\rangle \langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT } U\rangle$$

- What values might/must have been changed?
- How does the recovery manager get the database back to a consistent state?

Discuss for Undo-, Redo-, and Undo/Redo-logging.

Database System Recovery

Solution (Undo)

$$\langle \text{START } T\rangle \langle T, A, 10, 11\rangle \langle \text{START } U\rangle \langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT } U\rangle$$

We first identify the transactions that we need to undo. Only transaction T must be undone.

By reading the log we can conclude that:

- A *might* have had its value changed on disk.
- B *must* have had its value changed on disk.
- C *might* have had its value changed on disk.
- D *must* have had its value changed on disk.

Database System Recovery

Solution (Undo)

$$\langle \text{START}\, T\rangle \langle T, A, 10, 11\rangle \langle \text{START}\, U\rangle \langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT}\, U\rangle$$

Starting from the *end* of the log:

- Ignore changes of transaction U altogether.
- Write value 30 for C.
- Write value 10 for A.
- Append $\langle \text{ABRT}\, T\rangle$ to the log.

Database System Recovery

Solution (Redo)

$$\langle \text{START } T\rangle \langle T, A, 10, 11\rangle \langle \text{START } U\rangle \langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT } U\rangle$$

We first identify the transactions that we need to redo. Only transaction U must be redone.

By reading the log we can conclude that:

- A *cannot* have had its value changed on disk.
- B *might* have had its value changed on disk.
- C *cannot* have had its value changed on disk.
- D *might* have had its value changed on disk.

Database System Recovery

Solution (Redo)

$$\langle \text{START}\, T\rangle \langle T, A, 10, 11\rangle \langle \text{START}\, U\rangle \langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT}\, U\rangle$$

Starting from the *beginning* of the log:

- Ignore changes of transaction T altogether.
- Write value 21 for B.
- Write value 41 for D.
- Append $\langle \text{ABRT}\, T\rangle$ to the log.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START } T\rangle \langle T, A, 10, 11\rangle \langle \text{START } U\rangle \langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle \langle U, D, 40, 41\rangle \langle \text{COMMIT } U\rangle$$

We first identify the transactions that we need to redo and those that we need to undo. U must be redone, while T must be undone.

By reading the log we can conclude that:

- A *might* have had its value changed on disk.
- B *might* have had its value changed on disk.
- C *might* have had its value changed on disk.
- D *might* have had its value changed on disk.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START } T\rangle\langle T, A, 10, 11\rangle\langle \text{START } U\rangle\langle U, B, 20, 21\rangle$$
$$\langle T, C, 30, 31\rangle\langle U, D, 40, 41\rangle\langle \text{COMMIT } U\rangle$$

Starting from the *end* of the log (undo):

- Ignore changes of transaction U altogether.
- Write value 30 for C.
- Write value 10 for A.
- Append $\langle \text{ABRT } T\rangle$ to the log.

Then, starting from the *beginning* of the log (redo):

- Ignore changes of transaction T altogether.
- Write value 21 for B.
- Write value 41 for D.

Database System Recovery

Task:

Consider the following log, where a checkpoint start has been added:

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

- When is $\langle$CKPT END$\rangle$ written?
- What happens if a crash occurs? (for each possible point at which a crash can occur)

Discuss for Undo-, Redo-, and Undo/Redo-logging.

Database System Recovery

Solution (Undo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

- The checkpoint entry identifies the transactions that are currently *active*. Hence, the checkpoint entry is $\langle \text{CKPT START } (T)\rangle$.
- Every one of these active transactions must commit before writing $\langle \text{CKPT END}\rangle$. We can write $\langle \text{CKPT END}\rangle$ right after $\langle \text{COMMIT } T\rangle$.

Database System Recovery

Solution (Undo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

The recovery depends on whether we first meet $\langle$CKPT END$\rangle$ or $\langle$CKPT START$\rangle$:

- if $\langle$CKPT END$\rangle$ was written last, we only need to consider the log up to $\langle$CKPT START $(T)\rangle$.
- if $\langle$CKPT START $(T)\rangle$ was written last, we need to consider the log up to $\langle$START $T\rangle$, as it was the only *active* transaction.

Database System Recovery

Solution (Redo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

- The checkpoint entry identifies the transactions that are currently *active*. Hence, the checkpoint entry is $\langle \text{CKPT START } (T)\rangle$.
- We first write blocks from transaction that were commited at $\langle \text{CKPT START}\rangle$ to the disk.
- We only write $\langle \text{CKPT END}\rangle$ after these blocks.
- We cannot predict when the dirty blocks will be written on disk: $\langle \text{CKPT END}\rangle$ can occur anywhere after $\langle \text{CKPT START}\rangle$.

Database System Recovery

Solution (Redo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

The recovery depends on whether we the last checkpoint entry was $\langle$CKPT END$\rangle$ or $\langle$CKPT START $(T)\rangle$:

- if $\langle$CKPT END$\rangle$ was written last, we know that transaction S was fully written. The transactions that were active at $\langle$CKPT START $(T)\rangle$ or started later and that have committed must be redone (that is, T, U, and V, depending on the place of the crash).
- if $\langle$CKPT START $(T)\rangle$ was written last, the checkpoint does not help. We need to go back to the previous $\langle$CKPT END$\rangle$, or to the beginning of the log.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

- The checkpoint entry identifies the transactions that are currently *active*. Hence, the checkpoint entry is $\langle \text{CKPT START } (T)\rangle$.
- All dirty blocks are written to disk first.
- We only write $\langle \text{CKPT END}\rangle$ after these blocks.
- We cannot predict when the dirty blocks will be written on disk: $\langle \text{CKPT END}\rangle$ can occur anywhere after $\langle \text{CKPT START}\rangle$.

Database System Recovery

Solution (Undo/Redo)

$$\langle \text{START } S\rangle\langle S, A, 60\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10\rangle$$
$$\langle \text{CKPT START}\rangle\langle \text{START } U\rangle\langle U, B, 20\rangle\langle T, C, 30\rangle\langle \text{START } V\rangle$$
$$\langle U, D, 40\rangle\langle V, F, 70\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50\rangle$$
$$\langle \text{COMMIT } T\rangle\langle V, B, 80\rangle\langle \text{COMMIT } V\rangle$$

The recovery depends on whether we the last checkpoint entry was $\langle$CKPT END$\rangle$ or $\langle$CKPT START $(T)\rangle$:

- if $\langle$CKPT END$\rangle$ was written last, we know that all the dirty buffers were written to disk. We only need to redo transactions from $\langle$CKPT START $(T)\rangle$, but we also need to undo transactions that were active at $\langle$CKPT START $(T)\rangle$. Hence, we may need to go back to $\langle$START $T\rangle$ when undoing.
- if $\langle$CKPT START $(T)\rangle$ was written last, the checkpoint does not help. We need to go back to the previous $\langle$CKPT END$\rangle$, or to the beginning of the log.