

7 SYSTEM RECOVERY

7.1 An Overview

- After some failure, we must be able to restore the database to a state that is known to be correct.
- A failure may be caused by
 - Bugs in application programs, operating system, database system ...
 - Hardware errors on devices, channels, CPU, or memory
 - Operator errors
 - External causes: power failure, fire, high temperature, lightning ...
- To ensure that the database can be recovered after a failure, the following actions are required during normal operation:
 - Database backup

Periodically the entire database is copied to archival storage. This copy should be stored in a safe place.
 - Database journalizing (logging)

Every time a change is made to the database, a record containing the new value (**after-image**, redo part) and, possibly, also the old value (**before-image**, undo part) is written to a special file called the **journal** (log file).

The journal is often kept in duplicate.
 - Checkpointing

Checkpointing is an operation that synchronizes the journal and the database, in the simplest case by suspending all processing, and performing all the pending writing.
- Types of failure and recovery action required:
 - Transaction-local failure

Only one transaction affected, database not damaged.

Perform a ROLLBACK, i.e. undo whatever changes the transaction has made to the database.
ROLLBACK issued either by the transaction or by the system.

- System-wide failure, database not damaged

All transactions in progress affected.

Undo the changes made by any transaction in progress at the time of failure.

Redo every committed transaction for which it is not known whether all its changes have physically been written to the database.

Possibly, restart the transactions that were rolled back.

- System-wide failure, database damaged

Restore the database from the latest back-up copy and redo all committed transactions. This may be a very slow process.

7.2 Incremental journal with immediate updates

- Log structure:

- Begin transaction record:

transaction number, (input message)

- Sequence of change records:

transaction number, page address, old value, new value

- End transaction record:

transaction number, commit/rollback

- Before a modified page is physically written to the database, the journal record (at least the undo part) must be **physically** written to the journal (not just placed in the journal buffer).

- Before the buffer manager writes a modified page to the database it must make sure that all journal records related to that page have been written to the journal. The simplest solution is to flush the journal buffer.

- Before the transaction commits, all its journal records and the end-transaction record must be physically written to the journal. Again the simplest solution is to flush the log buffer.
- Transaction UNDO
 1. Scan backwards through the journal
 2. For every record related to the transaction to be undone:
 - undo the change by rewriting the old value (before-image) from the change record until reaching the begin-transaction record.
- Cascading rollbacks
 - When rollback of a transaction forces rollback of another transaction.
 - If some other transaction has been allowed to read or write a page modified by the transaction being rolled back, then that transaction must also be rolled back.
 - May occur as soon as transactions are allowed to see uncommitted changes (dirty reads).
 - Note that this may force rollback of a transaction that has already committed. NOT ACCEPTABLE.
 - To avoid cascading rollback, before a transaction commits, any data written by it are not allowed to be read by other transactions.
- Checkpointing
 - A checkpoint synchronizes the log and the contents of the database by making sure that all writes have been physically performed.
 - Taking a checkpoint consist of the following steps:
 1. Suspend transaction processing
 2. Physically write out all log buffers
 3. Physically write out all modified pages in the page buffer
 4. Write a checkpoint record to the log
 - list of active transactions
 - pointers to their most recent log records (possibly)

5. Record the address of the checkpoint record in a “restart file”
 6. Resume transaction processing
- During recovery, only the following transactions need to be considered:
- * transactions that started after the most recent checkpoint, and
 - * transactions that were active at the time of the most recent checkpoint.

- **Restart procedure**

1. Scan the log forwards from the most recent checkpoint, identifying the type of each transaction recorded
 - T1: started before the checkpoint and had an end-transaction record
 - T2: started after the checkpoint and had an end-transaction record
 - T3: started before the checkpoint and had no end-transaction record
 - T4: started after the checkpoint and had no end-transaction record
2. Scan backwards undoing all transactions of type T3 and T4.
3. Scan forwards from the most recent checkpoint redoing (rewriting the new values) all committed transactions of type T1 and T2. Ignore all transactions that were rolled back.
4. Resume transaction processing, possibly restarting transactions of type T3 and T4.

7.3 Incremental log with deferred updates

- All writes to the database are deferred until the transaction commits.
- Transaction rollback is now trivial: omit the database writes and write an end-transaction record indicating rollback to the log.
- Begin-transaction records, end-transaction records, and change records, but omitting the old value, are written to the log as before.
- The end-transaction record can be written as soon as all change records have been forced out and before the database writes have been performed.

- During restart no transactions need to be undone. Committed transaction must still be redone because the changes may not have been physically written at the time of failure.

Reading: Chapter 17.