

How to Implement Atomicity and Durability?

1. Recovery Mechanism in DBMS

This part of DBMS ensures two things:

Property	Meaning
Atomicity	A transaction must happen completely or not at all
Durability	Once changes are saved, they should never be lost—even after system crash

2. Shadow Copy Scheme

This method keeps **two versions of the database**:

Copy	Meaning
Original copy → Shadow copy	Safe backup copy
New copy	The copy where changes are made

Steps:

1. Before updating, a **full new copy** of database is made.
2. All updates happen on this **new copy only**.
3. If transaction fails → **delete new copy**, old copy remains safe.
4. If transaction succeeds → change pointer to point to new copy.

So success = pointer updates

Failure = pointer never changed → old DB remains safe

Real-World Example

Think of editing a Word file named **Project.docx**

Before editing, you make a copy:

Project_original.docx → shadow copy

Project_edit.docx → new copy

You do all the changes in Project_edit.docx.

Two situations:

If editing fails / file corrupts	If editing complete
Delete Project_edit.docx → original safe	Replace old file with new one

So you either:

apply all changes
or apply none

This is **Atomicity**.

If your laptop restarts before saving, you still have original file.

If you save and reboot, new changes remain.

This is **Durability**.

Why Shadow Copy Scheme is not used much?

Because **full database copy is made for every transaction** → very slow & wasteful.

3. Log-Based Recovery Method

Here instead of copying full DB, system **records all actions in a log file**.

Every operation is written in log first, then applied to DB.

If system crashes → log helps undo/redo changes

Much faster than shadow copy

Deferred Modification

Write actions are **recorded first**, but **actual database changes are done only after transaction finishes**.

Situation	Result
If crash before completion	Logs ignored → nothing applied
If transaction completes	Logs are used to update DB

If crash happens during update → use log to **redo** operations.

Real Life Example – Deferred Method

Imagine writing notes but keeping them in **rough notebook** first (log)

You write everything → but don't write in fair book (DB) yet.

If midway book tears or you decide not to finish → rough notes ignored.

If you complete fully → now copy final notes neatly into fair book.

Immediate Modification

Here changes are written directly to DB *even while transaction is running*.

Log contains:

- **old value** (before update)
- **new value** (after update)

So we know what to undo or redo.

Situation	Action
Crash before commit	Use old values → Undo changes
Crash after commit	Use new values → Redo changes

Real-Life Example – Immediate Method

Suppose you are editing a Google Sheet directly (live database)

Auto-save stores every step (log):

Old value = before edit

New value = after edit

If laptop dies midway → you undo changes using old values.

If changes were completely saved → you redo using new values.

Remember

Method	How it Works	When Crash Happens
Shadow Copy	Full DB copy made	Keep old copy if fail
Deferred Log	First log everything, update later	Ignore logs if fail
Immediate Log	Update DB directly + log values	Undo/Redo through log