CUBRID Transaction & Recovery

MVCC + Locks, WAL, ARIES, Vacuum, 2PC

2026-05 · Code Analysis Seminar

© 2026 CUBRID Corporation. All rights reserved.

Agenda

1. **Why we need a recovery story** — what `COMMIT` has to mean after a crash
2. **Three timelines that must reconcile** — transactional, physical-log, page
3. **The ACID lens** — eleven CUBRID modules placed by which letter they own
4. **The WAL pipeline** — prior list, log manager, checkpoint, recovery, vacuum, DWB
5. **Extensions** — group commit, 2PC, flashback, backup / restore

> Closing: a symbol-level walkthrough of where to read next.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Why we need a recovery story — ACID-D

A relational engine sells the client four guarantees. Three of them survive a clean shutdown by accident; **durability** has to be engineered.

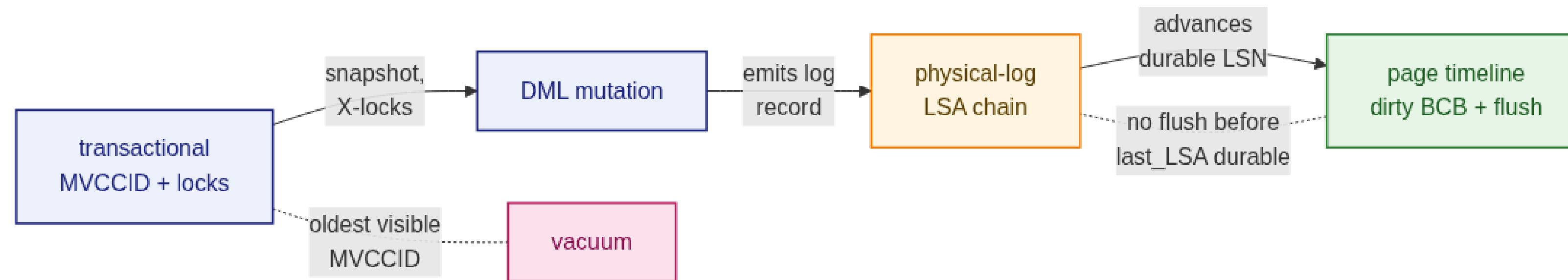
- **Anomalies that survive crashes.** A half-applied transaction, a torn data page, a committed log record never fsync'd, a dead version no one cleaned up — every one of these is recoverable only if the engine has bookkeeping for it.
- **Durability is the deepest pipeline.** It threads through five modules: TDES (per-tx state), log manager (WAL), prior list (commit pipe), checkpoint (recovery anchor), recovery manager (ARIES restart). Plus the DWB across the storage-engine boundary.
- **Atomicity is durability's twin.** The same log records that prove durability **for committed work** prove atomicity **for aborted work** — undo walks the same chain backward.
- **Isolation rides on top.** MVCC and the lock manager produce the right concurrent order; if the order isn't preserved by recovery, isolation is a lie.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID™

Three timelines that must reconcile

Every commit has to be valid against three independent clocks.



- **Transactional** — MVCCIDs name versions; locks order writers.
- **Physical-log** — LSAs name records; WAL ordering is the structural invariant.
- **Page** — BCBs name dirty frames; the page buffer enforces "log first, then page".

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Six common DBMS patterns

Every WAL-based engine composes the same handful of patterns. CUBRID is one point in the dial-space — not an invention.

Pattern	What it solves	CUBRID's version
WAL ordering	dirty page never reaches disk before its log	`page_buffer` checks `oldest_unflush_lsa` against durable LSN
MVCC snapshot isolation	readers never block writers	monotonic MVCCID + active-MVCCID table
ARIES three-pass restart	bounded recovery time	analysis -> redo -> undo, anchored at `chkpt_lsa`
Fuzzy checkpoint	no engine freeze	`LOG_START_CHKPT` / `LOG_END_CHKPT` bracket
Torn-write protection	half-written page survives crash	double-write buffer (in Storage Engine)
Vacuum + watermark	reclaim dead MVCC versions	WAL-driven forward walk below oldest-visible MVCCID

> The next eleven slides walk these patterns through CUBRID's eleven modules.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Part II

How CUBRID realises these

© 2026 CUBRID Corporation. All rights reserved.

The ACID lens — what each letter maps to

Each letter has one primary owner and a small set of supporting modules.

Letter	Primary owner	Supporting modules
A Atomicity	`recovery-manager` (undo pass + CLRs)	`log-manager`, `transaction` (TDES LSA chains)
C Consistency	enforced **above** this section (DDL & Schema)	bookkeeping flows through `log-manager` + `prior-list`
I Isolation	`mvcc` + `lock-manager` jointly	`vacuum` (reclaims dead versions)
D Durability	`log-manager` + `prior-list`	`checkpoint`, `recovery-manager`, DWB (in Storage Engine)

- **Consistency lives elsewhere**, but every constraint check it fires still routes log records through this section.
- **Atomicity and durability share the same WAL.** Different passes over the same log records.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID™

`transaction` — TDES, trantable, isolation dispatch

The per-transaction state hub.

- `LOG_TDES` — `trid`, lifecycle state, isolation, MVCCID, head/tail/undo-next/postpone-next LSAs, savepoint chain.
- **Trantable** — `log_Gl.trantable` is a fixed-size array indexed by transaction index; recycled via `hint_free_index`.
- **Isolation dispatch** — TDES `isolation` gates **snapshot timing** (per-stmt vs at-start) and **lock duration** (per-stmt vs to-commit).
- **Nested partial rollback** — `topops` system-op stack; savepoints are named LSAs with a `prv_savept` chain.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

MVCC pointer — snapshot isolation in two paragraphs

CUBRID's default isolation is **snapshot isolation** built on monotonic MVCCIDs.

- **What it owns.** `MVCCID` (64-bit lazy counter), the **active-MVCCID table** (per-server, bit array + overflow list), per-TDES `mvccinfo.snapshot`, and the **oldest-visible-MVCCID watermark** that vacuum consults.
- **Visibility rule.** A row version is visible iff its inserter committed before snapshot time and its deleter (if any) committed after. Implemented inline on the heap record header — no log lookup needed on the read path.
- **Snapshot timing is the isolation knob.** Read Committed acquires fresh per statement; RR / SI acquires once at transaction start; Serializable falls back to the lock manager for write skew.
- **Cost of the model.** One long-running writer pins the watermark and stalls vacuum — the same structural limit PostgreSQL and InnoDB carry.

> See `2026-05-cubrid-mvcc-analysis` deck for the active table, snapshot construction, and vacuum coordination at depth.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Lock manager pointer

CUBRID's lock manager handles **write-write conflicts** and **schema stability** — the half of isolation MVCC doesn't cover.

- **Multi-granularity.** A coarse intent lock at the class first (`IS`, `IX`, `SIX`), then a fine real lock on the row (`S`, `X`, `U`). DDL takes `SCH-S` / `SCH-M`.
- **12-mode compatibility matrix.** Beyond the textbook 5×5 (S, X + intentions), CUBRID adds `BU` (bulk update), `SCH-S/M` (schema), `NON_2PL` (read-committed read), and the conversion path goes through the `lub` (least-upper-bound) table.
- **Strict 2PL for writes.** X-locks held to commit / abort. Read-lock duration follows isolation: per-statement under Read Committed, to-commit under RR / Serializable.
- **Deadlock detection.** Background daemon scans the waits-for graph for cycles; the most-recently-blocked transaction is the victim.

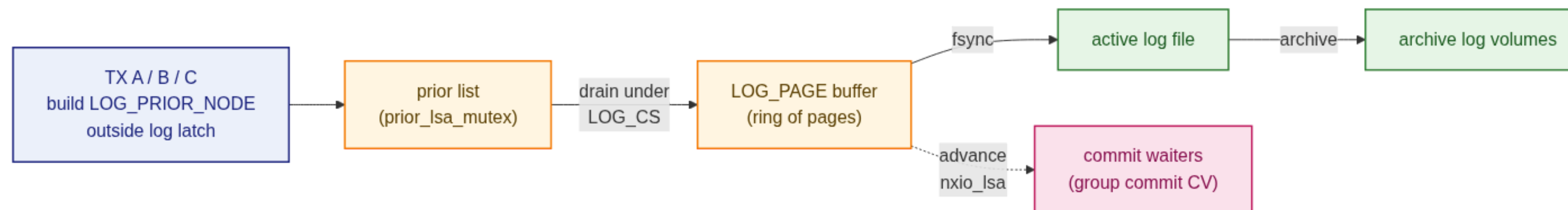
> See `2026-05-cubrid-lock-manager-analysis` deck for the matrix, `LK_ENTRY` shape, and the deadlock detector at depth.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

`log_manager` + `prior_list` — the WAL pipeline

The single substrate every other module rides on.



- **Producer side stays off the latch.** Record bytes, including zlib compression, are formatted into a fresh `LOG_PRIOR_NODE` **before** `prior_lsa_mutex` is taken; the mutex covers only LSN arithmetic and tail-link manipulation.
- **Single-writer drain.** `logpb_prior_lsa_append_all_list` is the only function that copies bytes into the authoritative `LOG_PAGE` ring buffer.
- **Durable LSN watermark** = `nxio_lsa`. Commit waiters park on the group-commit CV until it advances past their tail LSA.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

WAL record taxonomy — `RVxx` families

`LOG_RCVINDEX` is the universal dispatch key. Each record carries an index into `RV_fun[]` — a table of `(undofun, redofun)` per record kind.

Family	What it covers	Example records
`RVDK_*`	disk manager — volumes, sectors	sector reserve, volume extend
`RVFL_*`	file manager — file allocation	file create, file destroy
`RVHF_*`	heap — slotted-page record mutations	`heap_insert`, `heap_update`, MVCC delete
`RVBT_*`	B+Tree — node mutations	leaf insert, key delete, split, merge
`RVEH_*`	extendible hash — directory + bucket	bucket split, directory double
`RV_MVCC_*`	MVCC operations	version chain, vacuum-clean
`LOG_SYSOP_END_LOGICAL_*`	nested top-op boundary	B+Tree split logical-undo

- **Physiological for the data side**, logical for index operations where physical undo would require re-emitting whole pages.
- **CLRs (`LOG_COMPENSATE`) are themselves redo-only** — a second crash during undo replays the partial undo as forward redo.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Checkpoint — fuzzy ARIES

The periodic record that **bounds** recovery work.

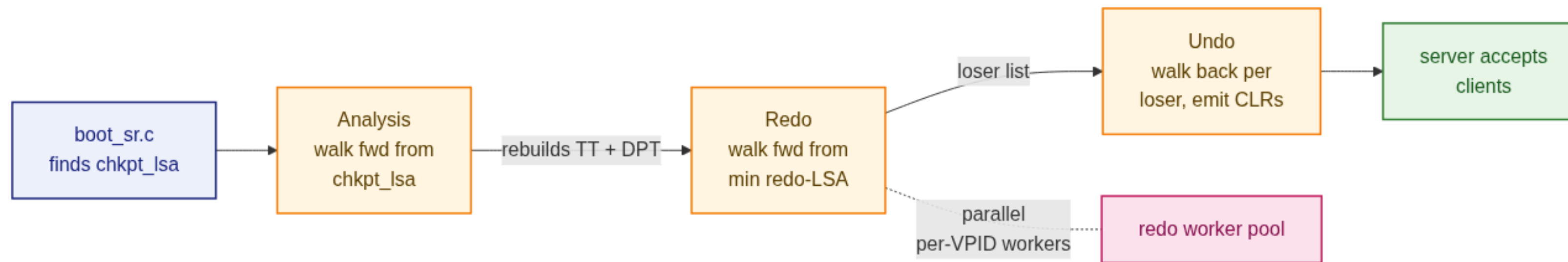
- **Trigger.** `log_Checkpoint_daemon` wakes on `log_checkpoint_interval` (timer-only in CUBRID — Postgres adds a space trigger; CUBRID does not).
- **Two bracket records.** `LOG_START_CHKPT` marks the LSA the next analysis will start from. `LOG_END_CHKPT` carries the snapshot in a `LOG_REC_CHKPT { redo_lsa, ntrans, ntops }` payload plus trailing per-tx and per-sysop arrays.
- **Redo-LSA hint, not a full DPT.** Most modern engines compress the dirty-page table into a single scalar — the smallest `oldest_unflush_lsa` across all dirty pages. Recovery rebuilds the DPT inline by walking forward.
- **Does NOT force every dirty page.** `pgbuf_flush_checkpoint` flushes enough pages that the **next** redo-LSA can advance; the checkpoint itself remains fuzzy — the engine keeps serving traffic between the two brackets.
- **Publishes via** **`log_Gl.hdr.chkpt_lsa`** after both brackets are durable. The header is then forced.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID™

Recovery — three-pass restart

ARIES, anchored at `log_Gl.hdr.chkpt_lsa`.



- **Analysis** rebuilds the transaction table and the dirty-page table. End state: who was active at crash, where redo must start.
- **Redo** is sequential per page (LSN order), parallel across pages — the modern code path bins records by target VPID and dispatches each bin to a worker.
- **Undo** walks each loser backward, applies `RV_fun[idx].undofun`, and emits a CLR so the undo itself is restartable.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

Vacuum — master + workers + watermark

The GC for MVCC. Driven by the **WAL itself**, not by a heap scan.

- **Watermark.** `log_Gl.hdr.oldest_visible_mvccid` — the smallest snapshot lower bound across all live transactions. Long writers pin it down.
- **Block of work.** The log is chunked into fixed-size **vacuum blocks** (default 31 log pages). One block is one dispatch unit.
- **Master / worker.** `vacuum_master_task` walks `vacuum_Data` looking for blocks below the watermark; up to 50 `VACUUM_WORKER` threads consume blocks in parallel.
- **Per-page sequential, across-page parallel.** Page-fix is the natural synchronisation — only one worker can hold the WRITE latch on a page at a time.
- **Dropped-files tracker.** When a relation is dropped while old versions still reference it, vacuum consults `vacuum_dropped_files_page` (a `vfid -> mvccid` map) before chasing a record into a freed extent.

> Workloads with many never-updated tuples save scan cost vs. PostgreSQL's heap-scan autovacuum; long scans without modifications gain nothing.

© 2026 CUBRID Corporation. All rights reserved.

CUBRID™

DWB cross-section

The double-write buffer sits in **Storage Engine** but is load-bearing for **durability**, so it earns a slide here.

- **The problem.** A page is 4-16 KiB; Linux only guarantees atomic writes at sector granularity (512 B or 4 KiB). A crash mid-write leaves a torn page — half old, half new.
- **The defence.** Every dirty page lands in a sequential staging volume (`<db>_dwb`) and is fsync'd there before its home write. On restart, any home page that fails its checksum is replaced from its DWB copy.
- **Recovery handshake runs BEFORE analysis.** `dwb_load_and_recover_pages` runs in the boot path, so by the time ARIES analysis starts, every home page is coherent.
- **The cost.** Two writes per dirty page — sequential on the DWB side, random on the home side. InnoDB has paid the same bill in production for two decades.

> Depth in the storage-engine deck and `cubrid-double-write-buffer.md`.

© 2026 CUBRID Corporation. All rights reserved.

Group commit emerges from queue batching

CUBRID has no `group_commit_interval` parameter. The behaviour falls out of the prior list.

- **The setup.** Each committer puts its commit record on the prior list, calls `logpb_flush_pages(my_commit_lsa)`, and parks on the group-commit condition variable.
- **The emergent batch.** While one fsync is in flight, every new committer queues behind it. When the fsync returns and `nxio_lsa` advances, **one broadcast** wakes every waiter whose target LSN is now durable.
- **Cost-amortisation per fsync.** N committers pay one fsync's latency, the fsync's `O(1)` syscall cost is split N ways.
- **No tuning knob needed.** The batch size auto-scales with load: light traffic = small batches, heavy traffic = large batches. The cost is paid by the daemon, not by waiters.

> Postgres ships `commit_delay`; InnoDB ships `innodb_flush_log_at_trx_commit` levels. CUBRID's queue makes both redundant — group commit is a property of the data structure, not a policy.

© 2026 CUBRID Corporation. All rights reserved.

Distribution extensions — 2PC, flashback, backup

Three extensions reuse the same TDES + WAL machinery without adding new substrates.

- **2PC.** Adds atomicity across servers. `LOG_2PC_EXECUTE` dispatches one TDES through coordinator vs. participant code paths. Prepared-state records (`LOG_2PC_PREPARE`) survive crash; in-doubt recovery is folded into the ARIES analysis pass — it rebuilds the `gtrid -> tid` map from the log. XA clients drive the FSM externally via `tran_2pc_*` .
- **Flashback.** Turns the WAL into queryable history. Two-phase forward walk: phase 1 builds a per-transaction summary (`trid` , user, time, INSERT / UPDATE / DELETE counts, classes touched); phase 2 materialises a chosen transaction's row images. Shares the CDC entry format but reads archived volumes.
- **Backup / restore.** Bundles data pages with the log records bracketed by `start_lsa` (checkpoint at copy start) and the next `LOG_END_CHKPT` . Restore mounts the page images and runs the recovery manager's redo pass forward to a user timestamp = PITR. Three incremental levels (`L0 / L1 / L2`) chain by skipping pages whose `prv.lsa <= parent.start_lsa` .

© 2026 CUBRID Corporation. All rights reserved.

CUBRID

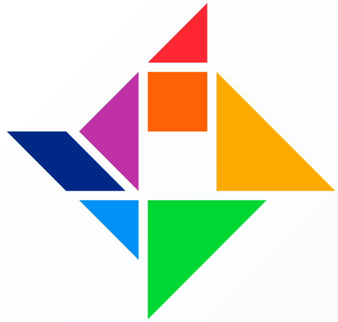
Source walkthrough — where to read next

Symbol names are the stable handle; line numbers drift. Detail docs live in `knowledge/code-analysis/cubrid/`.

Module	Key symbols	Files	Detail doc
Transaction	`LOG_TDES`, `logtb_assign_tran_index`, `log_sysop_start`	`log_impl.h` · `log_tran_table.c`	`cubrid-transaction.md`
MVCC	`mvcctable::build_mvcc_info`, `mvcc_satisfies_snapshot`	`mvcc.c` · `mvcc_table.cpp`	`cubrid-mvcc.md`
Lock manager	`lock_object`, `lock_detect_local_deadlock`	`lock_manager.{h,c}`	`cubrid-lock-manager.md`
Log + prior	`prior_lsa_alloc_and_copy_data`, `logpb_prior_lsa_append_all_list`	`log_manager.c` · `log_append.{cpp,hpp}`	`cubrid-log-manager.md` · `cubrid-prior-list.md`
Checkpoint	`logpb_checkpoint`, `pgbuf_flush_checkpoint`	`log_page_buffer.c`	`cubrid-checkpoint.md`
Recovery	`log_recovery`, `log_rv_redo_record_sync<T>`, `RV_fun[]`	`log_recovery.c` · `log_recovery_redo.cpp`	`cubrid-recovery-manager.md`
Vacuum + 2PC	`vacuum_master_task`, `log_2pc_commit`, `log_2pc_recovery`	`vacuum.c` · `log_2pc.{h,c}`	`cubrid-vacuum.md` · `cubrid-2pc.md`
Flashback + backup	`flashback_make_summary_list`, `logpb_backup`	`flashback.c` · `log_page_buffer.c` · `file_io.c`	`cubrid-flashback.md` · `cubrid-backup-restore.md`

© 2026 CUBRID Corporation. All rights reserved.

CUBRID



Thank you

Q & A

- Section overview: `knowledge/code-analysis/cubrid/cubrid-overview-txn-recovery.md`
- Detail docs (11): `cubrid-{transaction,mvcc,lock-manager,vacuum,log-manager,prior-list,checkpoint,recovery-manager,2pc,flashback,backup-restore}.md`
- Code: `src/transaction/*` · `src/query/vacuum.{h,c}` · `src/storage/double_write_buffer.{hpp,cpp}`

© 2026 CUBRID Corporation. All rights reserved.