RAG Outbox SQLite — Single-Writer Topology, Checkpoint Discipline & Integrity Watchdog (MC #106047)

RAG Outbox SQLite — Single-Writer Topology, Checkpoint Discipline & Integrity Watchdog

Status: ACTIVE **Created:** 2026-07-20 **Owner:** FlowForge **Related:** MC #106047 (root cause + fix), MC #106066 (follow-up: fsevents debounce)

Purpose

`~/system/state/ingest-queue.sqlite` is the durable outbox queue that feeds documents into LightRAG (BookStack pages, MC task outcomes, evidence files, specs, rules). It corrupted repeatedly (Apr 23, May 9, Jul 15→20-silent, Jul 20 2026) with the same `SQLITE_CORRUPT` / "database disk image is malformed" signature. This runbook documents the root cause, the fix, the writer topology, and the guard now in place so future recurrences are caught in minutes instead of days.

Writer Topology (as of 2026-07-20)

All writers use the shared library `~/system/lib/rag-outbox.js` (`openOutbox()`), which owns WAL-mode pragmas and the SQLite connection lifecycle.

Daemon (LaunchAgent)	Script	Trigger	Lifetime	Checkpoint mode
`com.alai.rag-drain-worker`	`tools/rag-drain-worker.js`	KeepAlive	Long-lived, continuous	**TRUNCATE** (explicit) — the single owner of periodic WAL truncation
`com.alai.rag-bookstack-adapter`	`tools/rag-bookstack-adapter.js`	StartInterval=300s	Fresh process per run	PASSIVE (default)
`com.alai.rag-mc-adapter`	`tools/rag-mc-adapter.js`	StartInterval=300s	Fresh process per run	PASSIVE (default)
`com.alai.rag-fsevents-adapter`	`tools/rag-fsevents-adapter.js`	WatchPaths (`evidence/` , `specs/` , `rules/`)	Fresh process per fs-event	PASSIVE (default)
`com.alai.lightrag-outbox-ingest`	`tools/lightrag-outbox-ingest.js`	StartInterval=21600s (6h)	Fresh process per run	PASSIVE (default)

`com.john.outbox-processor` (`daemons/outbox-processor.js`) is **not** a writer to this DB — it polls unrelated outbox tables in `durable-runner.db` , `mission-control.db` , `drafts.db` .

Drain Worker Rate Configuration (MC #101501, 2026-07-29)

`com.alai.rag-drain-worker` is deployed with:

Setting	Value	Source
`MAX_UPLOADS_PER_MINUTE`	`20`	`~/Library/LaunchAgents/com.alai.rag-drain-worker.plist` and `~/system/config/launchagents/com.alai.rag-drain-worker.plist`
`DRAIN_INTERVAL_MS`	`10000`	LaunchAgent environment
`BATCH_SIZE`	`5`	`~/system/tools/rag-drain-worker.js` default
`MAX_CONCURRENT`	`1`	LaunchAgent environment; serial uploads remain the design

`rag-drain-worker.js` now treats `DRAIN_INTERVAL_MS` as the deployed alias for `DRAIN_SLEEP_MS` . Effective behavior: batch-5 every 10s can burst near 30 docs/min, while the token bucket caps steady-state uploads at 20 docs/min. This implements the MC #101501 increase from the previous deployed cap of 10 docs/min without adding concurrency.

Verification command after edits/restarts:

```
launchctl print "gui/$(id -u)/com.alai.rag-drain-worker" | grep -E 'state =|pid
=|MAX_UPLOADS_PER_MINUTE|DRAIN_INTERVAL_MS'
tail -12 ~/system/logs/rag-drain-worker.log
```

Expected evidence lines:

```
MAX_UPLOADS_PER_MINUTE => 20
DRAIN_INTERVAL_MS => 10000
[drain] Rate config: max_uploads_per_min=20 batch_size=5 drain_sleep_ms=10000
```

Root Cause (MC #106047)

`rag-outbox.js`'s own header comment says "*Do NOT require() this from two processes simultaneously*" — the design assumed a single writer. The deployed topology has five. `openOutbox()` unconditionally ran `PRAGMA wal_checkpoint(TRUNCATE)` on every open, with **no `busy_timeout`** set anywhere in the file.

`TRUNCATE` mode requires exclusive WAL access to zero out the WAL file. The four short-lived adapters each open a brand-new connection (and therefore force a new `TRUNCATE` checkpoint) on every invocation — for `rag-fsevents-adapter` this can mean several times per minute during write bursts (1,965+ file touches observed under `evidence/` alone since 2026-07-15). Meanwhile `rag-drain-worker` holds one long-lived WAL connection open continuously. Without `busy_timeout`, contending checkpoints don't wait each other out — and a `TRUNCATE` checkpoint that's interrupted or races another process's checkpoint on the same WAL/shm can leave the b-tree in an inconsistent state. This matches the observed corruption signature exactly (`tree2/page2` btree errors).

The Jul 20 event was **silent for 5 days** (corruption occurred ~Jul 15 12:57 per file mtime, discovered manually by John on Jul 20) — proof no integrity guard existed before this fix.

Confirmed *not* the cause: unclean shutdown/reboot (no reboot near either corruption window per `last reboot`), disk pressure (6% used, 198Gi free at time of investigation).

Fix (applied 2026-07-20)

File: `~/system/lib/rag-outbox.js`

- Added `db.pragma('busy_timeout = 5000')` right after the `journal_mode`/`synchronous` pragmas, so a connection waits up to 5s for a lock instead of failing/racing immediately.

- `openOutbox(dbPath, opts)` now accepts `opts.checkpointMode` (`'PASSIVE'` default, or `'TRUNCATE'`). `PASSIVE` never blocks or forces exclusivity — it checkpoints whatever it safely can.

File: `~/system/tools/rag-drain-worker.js`

- Its `openOutbox(DB_PATH)` call now passes `{ checkpointMode: 'TRUNCATE' }` explicitly — it remains the single designated owner of periodic WAL truncation, per the original single-writer spec.

All four other callers (`rag-bookstack-adapter.js`, `rag-mc-adapter.js`, `rag-fsevents-adapter.js`, `lightrag-outbox-ingest.js`) pass no second argument and therefore get the new `PASSIVE` default automatically — no code changes needed in those files.

Verification: all 5 writer daemons were restarted via `launchctl kickstart -k gui/<uid>/<label>` (NOT `nohup` — orphaned `nohup` processes hold the SQLite lock and are a known trap on this system). Post-restart: `rag-drain-worker` confirmed alive and actively draining (LightRAG health OK, `pipeline_busy=true`); no new `SQLITE_CORRUPT`/malformed entries in any `.err` log; `node ~/system/tools/rag-ingest-integrity-watchdog.js --no-alert` returned `integrity=ok quick=ok`.

Integrity Watchdog (guard)

`~/system/tools/rag-ingest-integrity-watchdog.js` runs `PRAGMA integrity_check` + `PRAGMA quick_check` against the live DB (read-only), writes Prometheus metrics to `~/system/metrics/rag-ingest-integrity.prom`, appends JSONL events to `~/system/logs/rag-ingest-integrity-events.jsonl`, and alerts Slack (`#alerts`) + HiveMind on failure with a 4h cooldown. It never mutates the queue DB.

LaunchAgent staged at `~/Library/LaunchAgents/com.alai.rag-ingest-integrity-watchdog.plist` (StartInterval=900s / 15 min, RunAtLoad, `plutil -lint` validated). **Not yet loaded** — persistent daemon registration is an outward-facing action; loading it requires CEO go-ahead, tracked separately by John.

Manual check any time:

```
node ~/system/tools/rag-ingest-integrity-watchdog.js           # alerts on failure
node ~/system/tools/rag-ingest-integrity-watchdog.js --no-alert # dry-run, no Slack/HiveMind
node ~/system/tools/rag-ingest-integrity-watchdog.js --json     # machine-readable
```

Housekeeping (deferred)

Stale corruption/backup snapshots from Apr 23 and May 9 (4 files, ~46M, in `~/system/state/`) are superseded by two later recovery cycles and are deletion candidates, but were **not deleted** — disk is at 6% (no pressure) and John did not create these files, so deletion is held pending explicit CEO/John confirmation. Today's pair (`.corrupt-20260720-0825`, `.pre-recover-20260720-0825`, ~91M) is kept as forensic evidence until MC #106047 closes.

Follow-up

MC #106066 — `rag-fsevents-adapter.js` has no debounce on `WatchPaths` bursts; every single fs event spawns a fresh process + fresh `openOutbox()` call. Even with `PASSIVE` checkpoint + `busy_timeout` landed, this remains unnecessary connection churn on the shared outbox DB during evidence-writing storms. Recommend a 2-5s debounce/coalesce window before triggering an enqueue pass.

Revision #3
Created 2026-07-20 07:13:24 UTC by John
Updated 2026-07-29 00:12:10 UTC by John