Ledger Roadmap

This document outlines potential improvements, new features, code cleanup opportunities, and technical debt identified in the Ledger codebase.

Feature Proposals

[Priority: High] Schema System with Attribute Definitions

Description: Implement a schema system allowing attributes to be declared with types, cardinality, uniqueness constraints, and indexing preferences.

Rationale: Currently Ledger is schema-free which provides flexibility but lacks type safety, cardinality enforcement (one vs. many), uniqueness constraints, and the ability to optimize indexes for specific attributes. A schema system would enable: - Compile-time type checking for attribute values - Cardinality validation (`:db.cardinality/one` vs `:db.cardinality/many`) - Unique value constraints (`:db.unique/identity`, `:db.unique/value`) - Component relationships for cascading operations - Better query optimization through attribute metadata

Affected Files: - New file: `Ledger/Schema/Types.lean` - New file: `Ledger/Schema/Validation.lean` - Modify: `Ledger/Db/Database.lean` (add schema to Db structure) - Modify: `Ledger/Tx/Types.lean` (add schema validation errors) - Modify: `Ledger/DSL/TxBuilder.lean` (add schema-aware builders)

Estimated Effort: Large

Dependencies: None

[Priority: High] Persistence Layer

Description: Implement durable storage with disk-based persistence, append-only transaction log, and crash recovery.

Rationale: Currently, all data is in-memory and lost on process termination. For production use, the database needs: - Append-only log file for durability - Periodic snapshots for faster recovery - Memory-mapped indexes for large datasets - Crash recovery from log replay

Affected Files: - New file: `Ledger/Storage/Log.lean` - New file: `Ledger/Storage/Snapshot.lean` - New file: `Ledger/Storage/Recovery.lean` - Modify: `Ledger/Db/Connection.lean` (add persistence to Connection)

Estimated Effort: Large

Dependencies: None

[Priority: High] Aggregation Functions in Queries

Description: Add support for aggregate functions like `count`, `sum`, `avg`, `min`, `max` in queries.

Rationale: The query engine currently only supports pattern matching and returns raw bindings. Aggregation is essential for analytics:

```
db.query do
  find [?count]
  where_ [[?e, ":person/age", ?age]]
  aggregate [(count ?e)]
```

Affected Files: - Modify: `Ledger/Query/AST.lean` (add aggregate clause) - New file: `Ledger/Query/Aggregates.lean` - Modify: `Ledger/Query/Executor.lean` (process aggregates) - Modify: `Ledger/DSL/QueryBuilder.lean` (add aggregate builders)

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Predicate Expressions in Queries

Description: Add support for predicate expressions in where clauses like `[(> ?age 21)]`, `[(= ?status "active")]`.

Rationale: The README shows predicate syntax but the current implementation in `Query/AST.lean` only supports pattern matching. Predicates would enable: - Comparisons: `>`, `<`, `>=`, `<=`, `=`, `!=` - String operations: `contains`, `starts-with`, `ends-with` - Arithmetic: `+`, `-`, `*`, `/` - Boolean logic: `and`, `or`, `not`

Affected Files: - Modify: `Ledger/Query/AST.lean` (add predicate clause type) - New file: `Ledger/Query/Predicate.lean` - Modify: `Ledger/Query/Executor.lean` (evaluate predicates) - Modify: `Ledger/DSL/QueryBuilder.lean` (add predicate builders)

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Entity Retraction (Cascading Delete)

Description: Implement whole-entity retraction that removes all facts about an entity, optionally cascading to component entities.

Rationale: Currently, retractions must specify exact attribute-value pairs. There is no way to: - Retract all facts about an entity at once - Handle component relationships (where child entities should be retracted with parent) - Clean up dangling references

Affected Files: - Modify: `Ledger/Tx/Types.lean` (add `TxOp.retractEntity`) - Modify: `Ledger/Db/Database.lean` (implement entity retraction) - Modify: `Ledger/DSL/TxBuilder.lean` (add `retractEntity` builder)

Estimated Effort: Medium

Dependencies: Schema System (for component relationships)

[Priority: Medium] Transaction Functions

Description: Allow custom functions to run within transactions for complex multi-step operations with validation.

Rationale: Datomic supports transaction functions that can read current database state and produce operations atomically. This enables: - Increment/decrement operations - Compare-and-swap patterns - Custom validation logic - Conditional transactions

Affected Files: - New file: `Ledger/Tx/Functions.lean` - Modify: `Ledger/Tx/Types.lean` (add function invocation TxOp) - Modify: `Ledger/Db/Database.lean` (execute transaction functions)

Estimated Effort: Large

Dependencies: None

[Priority: Medium] Rules and Recursive Queries

Description: Implement Datalog rules for defining derived relationships and recursive queries.

Rationale: Rules would enable powerful graph queries like "find all ancestors" or "compute transitive closure". Example:

```
rule "ancestor" [?a, ?d] :=
  [?a, ":person/parent", ?d]
  or
  [?a, ":person/parent", ?p], ancestor(?p, ?d)
```

Affected Files: - New file: `Ledger/Query/Rules.lean` - Modify: `Ledger/Query/AST.lean` (add rule definitions) - Modify: `Ledger/Query/Executor.lean` (implement rule evaluation)

Estimated Effort: Large

Dependencies: None

[Priority: Medium] Query Macro DSL

Description: Implement a Lean 4 macro-based DSL for writing queries with syntax closer to Datomic's.

Rationale: The current builder pattern is verbose. A macro DSL would provide cleaner syntax:

```
ledger_query! {
  :find [?name ?age]
  :where [[?e :person/name ?name]
          [?e :person/age ?age]
          [(> ?age 21)]]
}
```

Affected Files: - New file: `Ledger/DSL/Macros.lean` - Modify: `Ledger.lean` (export macro)

Estimated Effort: Medium

Dependencies: None

[Priority: Low] Full-Text Search Index

Description: Add a full-text search index for string attributes.

Rationale: The AVET index supports exact value matching but not text search. Full-text search would enable: - Tokenized text indexing - Fuzzy matching - Relevance scoring

Affected Files: - New file: `Ledger/Index/Fulltext.lean` - Modify: `Ledger/Index/Manager.lean` (manage fulltext index) - Modify: `Ledger/Query/AST.lean` (add fulltext clause)

Estimated Effort: Large

Dependencies: Schema System (to mark attributes as fulltext-indexed)

[Priority: Low] Range Queries on Indexes

Description: Implement true range queries on RBMap indexes instead of filtering.

Rationale: Currently, index queries like `datomsForEntity` iterate the entire index and filter (`filterMap`). The RBMap structure supports efficient range queries but they are not used.

Affected Files: - Modify: `Ledger/Index/EAVT.lean` - Modify: `Ledger/Index/AEVT.lean` - Modify: `Ledger/Index/AVET.lean` - Modify: `Ledger/Index/VAET.lean`

Estimated Effort: Medium

Dependencies: None

[Priority: Low] Database Listeners/Watchers

Description: Implement a mechanism to subscribe to database changes.

Rationale: Applications often need to react to data changes: - UI updates when data changes - Triggering side effects (notifications, cache invalidation) - Synchronization with external systems

Affected Files: - New file: `Ledger/Db/Listeners.lean` - Modify: `Ledger/Db/Connection.lean` (add listener management)

Estimated Effort: Medium

Dependencies: None

[Priority: Low] Database Statistics and Query Planning

Description: Maintain statistics about data distribution and use them for query optimization.

Rationale: The current query executor orders patterns by selectivity using a simple heuristic (bound term count). Statistics would enable: - Cardinality estimation - Cost-based query planning - Adaptive query optimization

Affected Files: - New file: `Ledger/Stats/Statistics.lean` - Modify: `Ledger/Query/IndexSelect.lean` (use statistics) - Modify: `Ledger/Db/Database.lean` (maintain statistics)

Estimated Effort: Large

Dependencies: None

Code Improvements

[Priority: High] Implement Efficient Range Queries in Indexes

Current State: All index query functions (`datomsForEntity`, `datomsForAttr`, etc.) in EAVT.lean, AEVT.lean, AVET.lean, and VAET.lean use `Batteries.RBMap.toList` followed by `filterMap`. This is O(n) for every query regardless of result size.

Proposed Change: Use RBMap's range query capabilities or implement a B-tree/skip list for efficient range scans: - Use `RBMap.foldlM` with early termination - Or implement lower/upper bound navigation - Or switch to a data structure with native range query support

Benefits: - O(log n + k) query time where k is result size - Significant performance improvement for selective queries - Required for production-scale workloads

Affected Files: - `Ledger/Index/EAVT.lean` (lines 39-57) - `Ledger/Index/AEVT.lean` (lines 35-50) - `Ledger/Index/AVET.lean` (lines 36-60) - `Ledger/Index/VAET.lean` (lines 41-65)

Estimated Effort: Medium

[Priority: High] Implement Proper Negation in Query Executor

Current State: The query executor in `Ledger/Query/Executor.lean` returns an empty relation for negation (`.not` clause) with a comment "For now, simplified: returns empty (proper impl needs stratification)".

Proposed Change: Implement proper negation-as-failure semantics: - Detect stratification violations (negation in recursive rules) - Evaluate negated clauses after positive clauses - Subtract matching bindings from the result

Benefits: - Enable queries like "find people who are NOT managers" - Complete Datalog semantics - Required for many real-world queries

Affected Files: - `Ledger/Query/Executor.lean` (line 92-95) - New file: `Ledger/Query/Stratification.lean`

Estimated Effort: Medium

[Priority: High] Add DecidableEq Instance for Datom

Current State: `Datom` only has a `BEq` instance but not `DecidableEq`. This limits its use in proofs and some library functions.

Proposed Change: Add `deriving DecidableEq` to the Datom structure or implement the instance manually.

Benefits: - Enable use with more Batteries/mathlib functions - Support for proofs about datom equality - Better Lean 4 idiom compliance

Affected Files: - `Ledger/Core/Datom.lean` (line 28)

Estimated Effort: Small

[Priority: Medium] Remove Partial Annotations in Pull Executor

Current State: The `pullNestedEntity` and `pullPatternRec` functions in `Ledger/Pull/Executor.lean` are marked as `partial`. While cycle detection and depth limits prevent infinite recursion at runtime, the Lean type system cannot verify termination.

Proposed Change: - Use fuel-based recursion with a Nat counter - Or restructure to use `decreasing_by` with well-founded recursion on visited set size

Benefits: - Remove `partial` annotation - Total functions are easier to reason about - Better for potential formal verification

Affected Files: - `Ledger/Pull/Executor.lean` (lines 84-146)

Estimated Effort: Medium

[Priority: Medium] Improve Error Handling in DSL.withNewEntity

Current State: In `Ledger/DSL/TxBuilder.lean`, the `withNewEntity` function silently returns a dummy TxReport on error instead of propagating the error.

Proposed Change: Return `Except TxError (Db × EntityId × TxReport)` or use the Except monad properly.

Benefits: - Proper error handling - No silent failures - Consistent API design

Affected Files: - `Ledger/DSL/TxBuilder.lean` (lines 134-140)

Estimated Effort: Small

[Priority: Medium] Add Hashable Instance for Datom

Current State: `Datom` lacks a `Hashable` instance, limiting its use in hash-based collections.

Proposed Change: Implement `Hashable` for Datom by combining hashes of its components.

Benefits: - Enable use in HashMap/HashSet - Performance improvement for certain operations - More flexible data structure options

Affected Files: - `Ledger/Core/Datom.lean`

Estimated Effort: Small

[Priority: Medium] Consolidate Key Types with Datom Comparison Functions

Current State: There are four separate key types (EAVTKey, AEVTKey, AVETKey, VAETKey) in `Ledger/Index/Types.lean` that duplicate ordering logic that also exists in `Ledger/Core/Datom.lean` (compareEAVT, compareAEVT, etc.).

Proposed Change: Consider: - Use newtype wrappers over Datom with different Ord instances - Or generate key types and their instances with metaprogramming - Or consolidate comparison logic to avoid duplication

Benefits: - Reduced code duplication - Single source of truth for ordering semantics - Easier maintenance

Affected Files: - `Ledger/Index/Types.lean` - `Ledger/Core/Datom.lean`

Estimated Effort: Medium

[Priority: Medium] Add Ord Instance for Datom

Current State: `Datom` has comparison functions (`compareEAVT`, etc.) but no `Ord` instance. The indexes use separate key types instead.

Proposed Change: Add an `Ord` instance for Datom (perhaps defaulting to EAVT order) to enable direct use in sorted collections.

Benefits: - Simplify index implementation - Enable Datom use in sorted containers directly - Better API ergonomics

Affected Files: - `Ledger/Core/Datom.lean`

Estimated Effort: Small

[Priority: Low] Use Array Instead of List in Relation

Current State: `Relation` in `Ledger/Query/Binding.lean` uses `List Binding` internally.

Proposed Change: Use `Array Binding` for better performance with random access and modifications.

Benefits: - O(1) random access - Better cache locality - More efficient joins

Affected Files: - `Ledger/Query/Binding.lean` (lines 96-142) - `Ledger/Query/Executor.lean`

Estimated Effort: Small

[Priority: Low] Cache getAllAttrs Result in Pull.pullWildcard

Current State: `getAllAttrs` in `Ledger/Pull/Executor.lean` iterates through entity datoms and deduplicates attributes on every wildcard pull.

Proposed Change: Consider caching attribute lists per entity, or using a set for deduplication.

Benefits: - Faster wildcard pulls - Reduced memory allocations - Better performance for repeated pulls

Affected Files: - `Ledger/Pull/Executor.lean` (lines 66-77)

Estimated Effort: Small

Code Cleanup

[Priority: High] Add Documentation Comments to All Public APIs

Issue: Many public functions lack documentation comments (docstrings). While some have brief comments, a consistent documentation standard would improve usability.

Location: Throughout all modules, particularly: - `Ledger/Query/Executor.lean` - `Ledger/Pull/Executor.lean` - `Ledger/DSL/*.lean`

Action Required: 1. Add docstrings to all public `def` and `structure` declarations 2. Document parameters, return values, and usage examples 3. Use consistent formatting

Estimated Effort: Medium

[Priority: Medium] Standardize Naming Convention for Query Functions

Issue: Inconsistent naming between modules. For example: - `findByAttrValue` vs `datomsForAttrValue` - `entitiesWithAttr` vs `findEntitiesWith`

Location: - `Ledger/Db/Database.lean` - `Ledger/DSL/Combinators.lean` - `Ledger/DSL/QueryBuilder.lean`

Action Required: 1. Establish naming convention (verb-first vs noun-first) 2. Rename functions for consistency 3. Add deprecation aliases for backward compatibility

Estimated Effort: Small

[Priority: Medium] Extract Test Helpers into Separate Module

Issue: The test file `Tests/Main.lean` is 945 lines and contains all test cases. No test helper utilities are extracted.

Location: `Tests/Main.lean`

Action Required: 1. Create `Tests/Helpers.lean` with common test setup (create test db, populate test data) 2. Split tests into topic-specific files: `Tests/Core.lean`, `Tests/Query.lean`, `Tests/Pull.lean`, etc. 3. Use shared fixtures for common scenarios

Estimated Effort: Medium

[Priority: Medium] Add Docstrings to Inductive Constructors

Issue: Inductive types like `Value`, `TxOp`, `PullPattern`, `Clause` have constructor documentation in some cases but not all.

Location: - `Ledger/Core/Value.lean` - `Ledger/Tx/Types.lean` - `Ledger/Pull/Pattern.lean` - `Ledger/Query/AST.lean`

Action Required: 1. Add docstrings to all constructors explaining their purpose 2. Include usage examples where appropriate

Estimated Effort: Small

[Priority: Low] Remove Redundant BEq Instances

Issue: Several types derive `DecidableEq` but also define explicit `BEq` instances that are equivalent. For example, `EntityId` derives both `DecidableEq` and defines a `BEq` instance.

Location: - `Ledger/Core/EntityId.lean` (lines 19-20) - `Ledger/Core/Attribute.lean` (lines 19-20) - `Ledger/Index/Types.lean` (multiple key types)

Action Required: 1. Remove redundant `BEq` instances where `DecidableEq` is derived 2. Or remove `DecidableEq` derivation if only `BEq` is needed

Estimated Effort: Small

[Priority: Low] Add Module-Level Documentation

Issue: While files have header comments, they lack structured module documentation that would appear in generated docs.

Location: All `.lean` files

Action Required: 1. Add `/-! ... -/` module documentation at the top of each file 2. Include module purpose, key types, and usage examples 3. Add `@[inherit_doc]` where appropriate

Estimated Effort: Small

[Priority: Low] Consolidate Helper Functions

Issue: Some helper functions are duplicated or very similar across modules: - `filterVisible` in `Ledger/Db/Database.lean` vs `filterVisibleAt` in `Ledger/Db/TimeTravel.lean` - `sameFact` and `groupByFact` in `TimeTravel.lean` could be in Core

Location: - `Ledger/Db/Database.lean` - `Ledger/Db/TimeTravel.lean`

Action Required: 1. Move shared utilities to a common module (e.g., `Ledger/Core/Util.lean`) 2. Reuse rather than reimplement similar logic 3. Document the shared utilities

Estimated Effort: Small

[Priority: Low] Use Type Aliases for Clarity

Issue: Some types could benefit from clearer aliases: - `List TxOp` is abbreviated to `Transaction` - good - `List Datom` is not aliased - could be `DatomSeq` or similar - `List (Attribute x PullValue)` repeated in Pull code

Location: - `Ledger/Core/Datom.lean` - `Ledger/Pull/Result.lean`

Action Required: 1. Introduce type aliases for commonly used compound types 2. Update usage sites

Estimated Effort: Small

Technical Debt

[Priority: High] Index Performance (Full Scans)

Issue: Every index query does a full scan of the RBMap, converting to list and filtering. This is O(n) regardless of query selectivity.

Location: - `Ledger/Index/EAVT.lean` - `Ledger/Index/AEVT.lean` - `Ledger/Index/AVET.lean` - `Ledger/Index/VAET.lean`

Impact: Poor query performance at scale; unsuitable for production workloads.

Remediation: Implement proper range queries (see Code Improvements section).

[Priority: High] No Validation of Retractions

Issue: The transaction processor in `Db.transact` does not verify that retracted facts actually exist. It simply adds retraction datoms without checking.

Location: `Ledger/Db/Database.lean` (lines 61-71)

Impact: - Silent failures when retracting non-existent facts - Potentially inconsistent historical data - `TxError.factNotFound` is defined but never raised

Remediation: Add validation that checks if the fact exists before creating retraction datom, or document that retractions are unconditional.

[Priority: Medium] Query Executor OR Clause Implementation

Issue: The `.or` clause implementation in `Ledger/Query/Executor.lean` (lines 89-91) simply concatenates results without proper handling of variable scoping or deduplication across branches.

Location: `Ledger/Query/Executor.lean`

Impact: - Potential duplicate results - Unexpected variable binding behavior

Remediation: Implement proper union semantics with deduplication based on find variables.

[Priority: Medium] TimeTravel.asOf Rebuilds Indexes

Issue: `Connection.asOf` in `Ledger/Db/Connection.lean` rebuilds all indexes from scratch by iterating through the transaction log and reinserting datoms.

Location: `Ledger/Db/Connection.lean` (lines 77-91)

Impact: - O(n) time for every time-travel query - Memory allocation for each query - Not practical for frequent historical queries

Remediation: Consider persistent data structures or snapshot caching for common time-travel points.

[Priority: Medium] Binding.merge is O(n^2)

Issue: The `merge` function in `Ledger/Query/Binding.lean` does linear lookup for each entry being merged, resulting in O(n*m) complexity.

Location: `Ledger/Query/Binding.lean` (lines 79-83)

Impact: Slow query execution for queries with many variables.

Remediation: Use a hash-based map for bindings instead of association list.

[Priority: Low] Missing Test Coverage

Issue: Some features lack test coverage: - OR clauses in queries - NOT clauses (currently broken) - Limited pull patterns - TxError handling - Edge cases in time-travel (empty history, single transaction)

Location: `Tests/Main.lean`

Impact: Potential bugs in untested code paths.

Remediation: Add comprehensive tests for all query clause types and edge cases.

[Priority: Low] Unused TxError Variants

Issue: `TxError.invalidEntity` is defined but never used in the codebase.

Location: `Ledger/Tx/Types.lean` (line 41)

Impact: Dead code that may confuse developers.

Remediation: Either use the variant for actual validation, or remove it.

Summary

This roadmap identifies improvements across several categories:

Category	High	Medium	Low	Total
Features	3	6	4	13
Code Improvements	4	4	3	11
Code Cleanup	1	3	5	9
Technical Debt	2	3	2	7

Priority Focus: 1. **Performance:** Index range queries are critical for any production use 2. **Correctness:** Proper negation, OR clause semantics, and retraction validation 3. **Features:** Schema system, persistence, and aggregation would significantly expand use cases 4. **Developer Experience:** Documentation, naming consistency, and macro DSL