Versioned Semantic Workspaces: Branching, Diffing, and Merging Schemas with Data

Matthew Long
The YonedaAI Collaboration
YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com · https://yonedaai.com

23 July 2026

Abstract

Database branches are easy to describe until a branch changes what the data means. Two analysts can begin with the same customer records, then choose different schema revisions, mapping paths, identity decisions, provenance requirements, and materialized views. A line-oriented merge reports which files changed. It cannot decide whether an address split preserves a mapping, whether an identity decision invalidates an aggregate, or whether an old query can still be reproduced after source evidence expires.

This paper specifies versioned semantic workspaces for CatDB. A workspace is an isolated overlay over an immutable semantic commit. The commit jointly references compatible revisions of schemas, mappings, migrations, data observations, identity decisions, provenance profiles, policies, queries, and materialization definitions. Diff has five layers: structural, operational, impact, validation, and realization. Three-way merge is partial. It automatically combines only operation pairs admitted by a reviewed noninterference or commutativity rule, then validates the complete candidate state before atomically publishing a branch reference. Every other case becomes a typed conflict, a reconciliation request, a rejection, or an unsupported result.

The proposal is an ENGINEERING HYPOTHESIS, not an implemented result. Dataset branching, relational difference and merge, schema histories, mapping adaptation, coexisting schema versions, temporal query, persistent data structures, provenance, dynamic data citation, and materialized-view adaptation are established prior work. The current repository has no workspace crate, semantic commit profile, overlay evaluator, merge classifier, atomic reference store, workspace fixtures, Haskell oracle, Lean workspace theorem, or workspace benchmark. Claims about overlay equivalence, isolation, atomic publication, historical reproduction, and safe coexistence therefore remain OPEN or OBSTRUCTED. The paper's contribution is a falsifiable contract, a conflict taxonomy, a formalization boundary, and an evaluation plan for testing whether cross-object versioning adds practical value beyond relational dataset version control.

Contents

1 Introduction

1.1 One source snapshot, two interpretations

Consider a company that integrates customer records from an ERP system and a CRM database. At commit C_0, the canonical customer schema stores an address as a string, the revenue mapping reads gross revenue, and two source records remain separate because their identity evidence is inconclusive. A materialized report uses row-level provenance from both source snapshots.

Team A creates a workspace from C_0. It splits the address into an `Address` object, rewrites the ERP mapping to follow the new path, and approves the two source records as one canonical customer. Team B starts from the same commit. It keeps the old address shape, changes the revenue mapping from gross to net, rejects the identity match, and requires field-level provenance for the report.

Both workspaces can be internally useful. Team A wants to measure the effect of deduplication; Team B wants a conservative financial view. Trouble begins when someone

asks to merge them. The address edit and revenue edit touch different files, but that fact says little. The address split changes a typed mapping path. The identity decisions contradict each other. The stricter provenance policy may make the old materialization inadmissible. The two revenue definitions may produce different totals even when the source rows are unchanged.

A source-control tool can preserve both text histories. A database versioning system can preserve and compare record versions. Neither fact supplies the missing semantic decisions. The merge needs stable identities for schema and mapping elements, explicit dependencies, branch-local constraints, authority for reconciliation, provenance completeness, and a publication protocol that cannot expose half of a merged state.

1.2 Research question

The paper asks:

> What is a semantic branch, diff, and merge when schemas, mappings, data, provenance, materializations, and reconciliation decisions evolve together?

The proposed answer is deliberately conservative. A branch is a mutable reference to an immutable semantic commit plus, while work is in progress, an isolated typed overlay. A diff reports semantic operations and dependency effects rather than serialized changes alone. A merge is a partial procedure with closed outcomes. It may return a validated commit, typed conflicts, reconciliation requirements, or an unsupported result. Unknown does not mean safe.

1.3 Scope and claimed contribution

The database-versioning literature already provides nonlinear dataset branches, working copies, version-aware queries, relational difference and three-way merge, and multiple physical layouts [3, 4, 16, 19]. Schema-evolution systems record schema-modification operators, derive mappings, support legacy access, and rewrite queries or updates across schema versions [7–9, 20]. Mapping adaptation and model-management operators also precede CatDB [2, 24, 27–29].

The candidate CatDB contribution is one executable compatibility boundary across artifacts that these systems often treat separately:

1. distinguish logical identity, immutable revision identity, content digest, physical location, and mutable branch reference;
2. bind every query and merge to one semantic commit or explicit overlay;
3. co-version schema, mapping, migration, data observation, identity, provenance, authority, query, and materialization definitions;
4. compute typed structural, operational, impact, validation, and realization diffs;
5. restrict automatic merge to reviewed operation pairs and validate the whole candidate after local pair processing;

6. publish a merge atomically through a compare-and-swap reference; and
7. distinguish state reconstruction, query replay, and exact result reproduction.

Every item in this list is an ENGINEERING HYPOTHESIS. The paper gives no CatDB performance number and no successful workspace execution. Its strongest present result is a prior-art boundary: the constituent ideas are established, while the cross-object contract remains to be built and tested.

1.4 Contributions of this manuscript

This provisional manuscript contributes:

- a finite semantic model for commit bundles, workspace overlays, operations, dependencies, typed diffs, and merge outcomes;
- a taxonomy of direct and cross-object conflicts, with exact unsupported and reconciliation boundaries;
- contracts for branch-local constraints, provenance, materialization state, rollback, and historical query contexts;
- thirteen falsifiable claim anchors linked to fixtures, implementation stages, formal obligations, and experiments;
- a fair comparison plan with Decibel and ORPHEUSDB on mutually supported relational operations; and
- explicit limitations, nonclaims, open questions, and current repository status.

1.5 Paper organization

Section 2 records evidence labels and the repository audit. Section 3 establishes the prior-art boundary. Section 4 defines identities, artifacts, commits, workspaces, and operations. Section 5 defines semantic diff and impact. Section 6 gives the three-way merge procedure and conflict taxonomy. Sections 7 and 8 treat co-versioning, provenance, materializations, historical queries, and rollback. Section 9 states candidate laws without presenting them as proved. Sections 10 and 11 give the implementation and falsification programs. Section 12 returns to related work, followed by limitations, open questions, and a status-preserving conclusion.

2 Evidence vocabulary and present status

2.1 Evidence labels

Table 1 applies to every result-like sentence in this paper. A label belongs to one claim under stated assumptions. It does not transfer from one layer to another.

Label	Meaning in this paper
SUPPORTED BY PRIOR LITERATURE	A primary paper, standard, or official system artifact establishes the stated result within its assumptions.
ENGINEERING HYPOTHESIS	A proposed CatDB behavior requires implementation and experiment.
OPEN CONJECTURE	A mathematical statement has no accepted proof for the frozen formal fragment.
OPEN	Definitions, code, fixtures, experiments, or review are incomplete.
OBSTRUCTED	A prerequisite semantic boundary is missing, so the claim cannot yet be implemented or tested honestly.
SPECULATIVE	The claim is too underspecified for a stronger label.

Table 1: Evidence labels used throughout the manuscript. No P9 claim is currently labelled proved, machine-checked, computational, or experimentally supported.

An accepted path-composition theorem does not establish workspace isolation. A relational merge result does not establish a CatDB cross-object merge. A content digest does not prove semantic equivalence. A successful historical query would not, by itself, establish scientific reproducibility.

2.2 Current repository audit

The current Rust workspace contains `catdb-core`, `catdb-ir`, `catdb-schema`, `catdb-mapping`, `catdb-migration`, and `catdb-cli`. Fixture schema v1 covers eight categories: schema parsing, path typing, path composition, identity mapping, mapping composition, migration validation, information-loss detection, and a deterministic stable-identifier schema declaration diff. Haskell covers that admitted schema/path/mapping/migration slice. Lean coverage lists path identity and associativity plus selected lawful-mapping identity and composition declarations.

That evidence is useful, but it stops before P9. The repository contains no:

- `catdb-workspace` crate or semantic commit profile;
- immutable artifact/commit store or atomic mutable-reference store;
- workspace state machine, overlay evaluator, or branch-aware query context;
- dependency-aware diff, conflict classifier, or merge validator;
- identity, provenance, and materialization workspace integration;
- historical-query reproduction or rollback implementation;
- workspace fixture family, Haskell workspace oracle, Lean workspace representation, or D7 benchmark run.

The existing schema diff reports declaration changes between two finite schemas. It does not record operation history, infer rename continuity, compute mapping or data impacts, merge commits, or preserve branch context. Calling it a complete semantic diff would be an overclaim.

2.3 Program status

Item	Status	Reason
CR-069, overlay workspaces	PROPOSED	no workspace implementation or experiment
CR-045, reproducible histories	PROPOSED and disputed	no replay, merge, or convergence witness
`SYS-WS-01`	OPEN	no overlay/materialized equivalence test
`SYS-WS-02`	OPEN	no workspace isolation test
`SYS-WS-03`	OPEN	no commit/ref store or crash test
P9 formal fragment	OPEN	no Haskell or Lean workspace model
P9 systems evaluation	OPEN	no harness, baseline, or raw run
Slice 12	OBSTRUCTED	prerequisite contracts and merge rules are not frozen

Table 2: Present status of the principal P9 artifacts and claims.

2.4 Dependency boundary

P9 depends on five other paper contracts:

P3, migration.
Immutable schema revisions, typed migration packages, information loss, synthesis, ambiguity, and the boundary among eager, lazy, and virtual execution.

P4, mapping and integration.
Mapping direction, domain, coverage, path images, loss, synthesis, ambiguity, composition, and obligations.

P6, materialization.
The split between authoritative semantic definitions and replaceable realizations, together with invalidation, reuse, freshness, and recovery.

P7, identity and reconciliation.
Source-local identity, candidate evidence, authority-bearing decisions, contradiction, supersession, and reversal.

P13, provenance.
Immutable provenance references, completeness, retention, redaction, and survival across schema change.

These dependencies are not editorial conveniences. A workspace cannot merge a migration whose meaning is unspecified, silently choose among ambiguous mapping adaptations, or treat a stale cache as authoritative state. This boundary keeps the present P9 profile OBSTRUCTED.

3 Prior-art boundary

3.1 Database branching and version-aware query

DataHub proposed collaborative dataset branch, merge, difference, search, and access [3]. Decibel implemented nonlinear relational dataset branches, working copies, version-aware scan and difference, field-level three-way merge, and alternative physical layouts [19]. ORPHEUSDB added version control to a conventional RDBMS and evaluated version representation, retrieval, partitioning, and migration [16]. Principles of Dataset Versioning formalized the tradeoff between storing versions and recreating them from deltas [4].

P9 therefore cannot claim database branches, immutable dataset commits, checkout, relational difference, record merge, historical dataset query, or compact multi-version storage as new. Decibel and ORPHEUSDB are direct systems baselines on the relational subset.

3.2 Schema histories and coexisting schemas

Schema versioning for multitemporal databases predates the CatDB proposal [10]. PRISM records schema-modification operators, derives mappings, supports legacy access, and generates migration or views [8]. PRISM++ extends this line with update rewriting and integrity-constraint maintenance [9]. PRIMA represents evolving schemas with transaction-time data and rewrites historical queries across versions [20]. Later work systematizes automation of the schema-evolution process [7].

InVerDa and BiDEL let multiple schema versions access one dataset through generated bidirectional delta code [15]. P9 cannot claim that multiple schema interpretations over shared data are new. It must show a wider compatibility bundle and a cross-object merge contract.

3.3 Mapping adaptation and model management

Schema evolution can invalidate mappings. Velegrakis, Miller, and Popa developed methods to preserve mapping consistency and adapt mappings while retaining earlier user choices where possible [27, 28]. Yu and Popa used mapping composition and pruning to adapt mappings under evolution [29]. Composition itself requires a precise mapping language: ordinary source-to-target dependencies are not generally closed under composition [12].

Model Management 2.0 names match, compose, diff, merge, translation, and generation of executable transformations as first-class concerns [2]. Algebraic model management connects these operators to algebraic and categorical structure [24]. Functorial data migration provides foundational categorical semantics for schema mappings and instance migration [26]. A P9 paper cannot obtain novelty merely by using the words “semantic diff”, “mapping-aware merge”, or “categorical merge”.

3.4 Time, persistence, provenance, and reproduction

Transaction-time and application-time databases answer questions along declared temporal dimensions. SQL:2011 standardizes system-versioned and application-time periods [17]. Multiversion concurrency control retains versions to enforce transactional correctness [1]; persistent data structures retain access to old states [11]. Neither an MVCC row version nor a persistent node is automatically a semantic commit.

Scientific archives preserve historical data and element continuity so later readers can check earlier results [5]. The RDA dynamic-data-citation recommendations use versioned data, timestamps, normalized stored queries, and persistent identifiers to recover precise subsets [22, 23]. VQuel proposed unified queries over versions and provenance [6]. W3C PROV provides an interchange vocabulary for entities, activities, agents, revisions, derivations, and invalidations [18]; provenance semirings give an algebraic account for positive query provenance [13]. PROV-IDEA explicitly connects schema and data provenance during database evolution [21].

A commit identifier is therefore not a novelty claim and not a reproducibility certificate. Exact replay also needs the query, source observations, semantic profiles, external dependencies, backend semantics, and declared nondeterminism.

3.5 Change explanation and materialization adaptation

Explain-Da-V infers transformations that explain changes between dataset versions [25]. Such an explanation can be useful evidence, but it is not necessarily the operation that happened or the author's intent. P9 keeps recorded operation, reconstructed state diff, inferred explanation, and approved intent as separate evidence kinds.

Materialized-view adaptation reuses prior materializations after selected view-definition changes when retained information makes it safe [14]. P9 must classify reuse, staleness, invalidation, and rebuild. It cannot merge cached bytes as semantic truth.

4 Mathematical framework

4.1 Five identities

Definition 4.1 (Logical identity)**.** A logical identity names the continuing semantic role of an artifact or schema element. Examples include `schema/customer`, `attribute/customer/legal-name`, and `mapping/sap-customer`. Continuity is declared by an admitted operation; it is not inferred from similar spelling.

Definition 4.2 (Revision identity)**.** A revision reference is a triple

$$r = (k, \ell, v) \in \mathsf{Kind} \times \mathsf{LogicalId} \times \mathsf{RevisionId},$$

where k is an artifact kind, ℓ is a logical identity, and v names one immutable revision. Two revisions of one logical artifact remain distinct even when their display names match.

Definition 4.3 (Content digest)**.** For profile p and canonical body b, the content digest is

$$d = H(\mathsf{domain} \parallel p \parallel b).$$

The domain and profile separate otherwise identical byte strings used for different purposes.

Definition 4.4 (Physical locator)**.** A physical locator records where bytes or rows currently reside. It may change without changing logical or revision identity. It belongs to execution metadata and provenance, not semantic equivalence.

Definition 4.5 (Mutable reference)**.** A branch or tag reference is

$$\rho = (n, c, g),$$

where n is a name, c is a commit identifier, and g is a generation. Updating ρ requires the caller to present the expected generation or target.

These layers prevent three common errors. Equal content does not collapse independently declared logical artifacts. A display rename does not create a new logical artifact when an authorized rename operation preserves its identity. A branch name does not identify an immutable commit.

4.2 Artifact revisions and dependencies

Definition 4.6 (Artifact revision)**.** An artifact revision is the tuple

$$\begin{aligned} A = (k, \ell, v, p, d, P, D, \pi, \alpha, e),& \\ k : \mathsf{Kind}, \quad \ell : \mathsf{LogicalId}, \quad v : \mathsf{RevisionId},& \\ p : \mathsf{Profile}, \quad d : \mathsf{Digest}, \quad P : \mathsf{RevisionRef}^*,& \\ D : \mathsf{DependencyRef}^*, \quad \pi : \mathsf{Provenance}, \quad \alpha : \mathsf{AuthorityRef},& \\ e : \mathsf{EventRef}.& \end{aligned}$$

The digest covers the canonical semantic body. Display names, timestamps, and physical locations enter metadata or provenance unless the profile makes them semantically relevant.

The first profile admits the artifact classes in Table 3.

Definition 4.7 (Typed dependency)**.** A dependency edge is

$$(r_f, r_t, k, s, q, i, e),$$

where r_f and r_t are revision references, k is a dependency kind, s is a selector or scope, q is strength, i is an invalidation rule, and e is evidence. Kinds include `typed-against`, `maps-from`, `maps-to`, `queries`, `reconciles`, `derived-from`, `materializes`, `requires-policy`, and `observes-source`.

Compiler-emitted, user-declared, and inferred dependencies are separate evidence classes. An inferred edge may prompt review. It does not silently become authoritative.

Class	Authoritative versioned content	Replaceable or excluded content
schema	objects, arrows, paths, equations, domains, constraints, stable IDs	backend DDL cache
mapping	direction, path images, coverage, loss, synthesis, ambiguity, obligations	generated SQL text
migration	source/target revisions, semantic transformation, mode, validation	one run's lock schedule
data observation	source-local references, snapshot/cursor, normalized operations	unversioned live source
identity	profile, evidence, decisions, contradictions, supersession, reversal	similarity score alone
provenance	source tokens, activities, completeness, retention, redaction	opaque log strings
query	normalized logical query and semantics	cached backend plan
materialization	definition, coverage, frontier policy, provenance profile	stored rows, index files, cache entries
policy	principals, authority scope, obligations, approvals	ambient administrator assumptions

Table 3: Artifact classes in the proposed first workspace profile.

4.3 Semantic commits

Definition 4.8 (Semantic commit)**.** A semantic commit body is

$$\begin{aligned} C = (&p, \vec{c}, m, o, v, a, \pi, x, r, z),\\ &p : \mathsf{Profile}, \quad \vec{c} : \mathsf{CommitId}^*,\\ &m : \mathsf{ArtifactManifestRef}, \quad o : \mathsf{OperationBatchRef},\\ &v : \mathsf{ValidationReportRef}, \quad a : \mathsf{ApprovalSetRef},\\ &\pi : \mathsf{Provenance}, \quad x : \mathsf{SourceSnapshotSetRef},\\ &r : \mathsf{MaterializationStatusSetRef}, \quad z : \mathsf{ConflictResolutionSetRef}. \end{aligned}$$

Its identifier is the domain-separated digest of its canonical body.

A root commit has no parent. An ordinary commit has one parent. A merge commit has two or more parent references plus an explicit merge-base record. Parent order is semantic when the profile distinguishes target and source roles; otherwise canonicalization fixes an order.

Requirement 4.9 (Commit validity)**.** A commit is publishable only when all required artifacts resolve and verify, dependencies target compatible revisions, profile constraints agree, reconciliation decisions satisfy authority rules, provenance completeness is sufficient, invalid materializations are not advertised as current, approvals cover the exact candidate digest, and normalized operation application agrees with the manifest state.

Commit immutability does not certify data accuracy, mapping intent, source authority, scientific reproducibility, or long-term availability.

4.4 Workspaces and overlays

Definition 4.10 (Workspace)**.** A workspace is

$$W = (w, c, \vec{o}, u, p, s, h_v, h_a, \rho, \pi),$$

where w is the workspace ID, c the base commit, $\vec{o}$ an ordered overlay of typed operation references, u the owner, p a policy revision, s the workspace state, h_v the validation head, h_a the approval head, ρ a proposed target reference, and π provenance.

The workspace identifier is not a commit identifier. A workspace can change until it is committed or abandoned. An edit after validation changes the overlay digest and returns the workspace to a draft state.

Definition 4.11 (Workspace materialization)**.** Let $\mathsf{State}(C)$ denote the normalized state named by commit C. For profile p,

$$\mathsf{materialize}(W) = \mathsf{Apply}_p\big(\mathsf{State}(c), \mathsf{Normalize}_p(\vec{o})\big).$$

Application is partial and returns a typed outcome.

Requirement 4.12 (Workspace validity)**.** A workspace is valid only if

$$\mathsf{Validate}_p(\mathsf{materialize}(W)) = \mathsf{valid}.$$

An optimized overlay evaluator may avoid full materialization, but its normalized data and provenance must match this canonical semantics.

4.5 Workspace state machine

The proposed local state machine appears in Table 4.

4.6 Semantic operations

Definition 4.13 (Semantic operation)**.** An operation is

$$o = (i, k, r, e, b, R, W, P, Q, \pi, \alpha),$$

where i is an operation ID, k its kind, r the subject, e the expected subject revision, b a typed payload, R and W the semantic read and write sets, P preconditions, Q postconditions, π provenance, and α authority. The operation may also carry obligations.

The initial operation families cover:

- schema add, remove, rename, retype, equation, constraint, domain, split, and merge declarations;

State	Meaning	Permitted next states
`draft`	overlay accepts authorized edits	validating, abandoned
`validating`	exact overlay digest is under validation	draft, review-ready, blocked
`review-ready`	validation passed for the named profile	draft, approved, rejected
`approved`	approvals cover the exact candidate	draft, merge-staging
`merge-staging`	immutable candidate artifacts are staged	approved, merged, blocked
`merged`	target ref atomically names the merge commit	terminal
`blocked`	conflict, missing evidence, or failed gate	draft, abandoned
`rejected`	reviewer or policy rejected the candidate	draft, abandoned
`abandoned`	no publication from this workspace	terminal

Table 4: Candidate workspace states. Approval never survives a changed candidate unless policy proves that it remains applicable.

- mapping endpoint, object image, path image, direction, coverage, loss, synthesis, ambiguity, and obligation changes;
- keyed data insert, delete, field update, rekey, assertion, retraction, and source-frontier advance;
- identity evidence, confirmed same/different, contradiction, supersession, reversal, and owner change;
- provenance evidence, profile, incompleteness, redaction, expiry, and invalidation;
- materialization definition, freshness, invalidation, reuse, rebuild, checkpoint, and retirement; and
- policy proposal, approval, rejection, obligation, discharge, and delegated-authority revocation.

The operation's semantic footprint is

$$\mathsf{footprint}(o) = \big(\mathsf{reads}(o), \mathsf{writes}(o), \mathsf{invalidates}(o), \mathsf{requires}(o)\big).$$

A schema retype may write one attribute but invalidate mappings, queries, and materializations that depend on it.

Definition 4.14 (Local independence)**.** Operations a and b are locally independent in state s only when their direct and derived footprints do not conflict, each remains well typed after the other, preconditions survive in either order, authority decisions are compatible, and the combined local effects validate.

Definition 4.15 (Commutativity)**.** Admitted operations commute in state s when

$$\mathsf{Apply}(\mathsf{Apply}(s, a), b) \equiv \mathsf{Apply}(\mathsf{Apply}(s, b), a),$$

including the profile's normalized audit and provenance projection.

Local commutativity does not replace complete merged-state validation. Two operations can commute over their immediate subjects and still violate a global equation, key, cardinality, provenance, or policy constraint.

5 Semantic diff and dependency impact

5.1 Five diff layers

A semantic diff is not one list. The proposed result separates five layers:

1. *structural diff* reports added, removed, renamed, or changed artifacts and elements;
2. *operational diff* reports retained typed operations between an ancestor and descendant;
3. *impact diff* reports registered dependencies whose denotation or admissibility may change;
4. *validation diff* reports newly satisfied, violated, unknown, or stale obligations; and
5. *realization diff* classifies materializations, plans, indexes, caches, and checkpoints as reusable, stale, invalid, or rebuild-required.

Definition 5.1 (Semantic diff result)**.** For commits A and B under profile p,

$$\mathsf{Diff}_p(A, B) = (p, A, B, \Delta_s, \Delta_o, \Delta_i, \Delta_v, \Delta_r, \Delta_e, \Delta_a, \Delta_u, \kappa),$$

where Δ_s is structural change, Δ_o retained operations, Δ_i impact, Δ_v validation change, Δ_r realization change, Δ_e inferred explanations, Δ_a ambiguity, Δ_u unsupported cases, and κ completeness.

Recorded operations and inferred explanations never receive the same evidence label. When operations are retained along an ancestor path, the candidate reconstruction law is

$$\mathsf{Apply}\big(\mathsf{State}(A), \mathsf{operationalDiff}(A, B)\big) \equiv \mathsf{State}(B).$$

For a state-derived diff, the weaker target is extensional state reconstruction. It says nothing about the historical operation order or the author's intent.

5.2 Stable identities and rename

A schema diff must distinguish:

- explicit rename from remove-plus-add;
- object identity from display label;
- type widening from narrowing;
- equation addition from a changed equation subject;

- path spelling from path denotation under a stable profile;
- cardinality change with checked data witnesses from an unchecked declaration; and
- declared split or merge from a structural pattern inferred after the fact.

The default rules appear in Table 5.

Operation	Logical identity treatment	Required evidence
display rename	preserve logical ID; create new revision	explicit rename operation and authority
remove plus add	distinct IDs by default	no inferred continuity
copy	new logical ID; possible shared ancestry/content	copy provenance
split	new target IDs plus mapping from prior ID	migration, loss, synthesis, and identity policy
merge	new or designated surviving ID	collision policy, reconciliation, and provenance
rekey	explicit identity transition	source and canonical identity policy
content-equal import	profile-dependent deduplication only	namespace and ownership policy

Table 5: Identity treatment for changes that text diff often conflates.

Text similarity may propose a rename candidate. Only an accepted typed operation establishes logical continuity.

5.3 Mapping-aware diff

A mapping revision is not adequately described by changed field names. Its diff includes:

- source and target revision changes;
- object images and arrow or path substitutions;
- direction and total or partial domain;
- source and target coverage;
- information loss and synthesis/default rules;
- ambiguity alternatives and unresolved obligations;
- authority; and
- dependent migrations, queries, write policies, and materializations.

Mapping adaptation may return multiple valid alternatives [27, 29]. A workspace must preserve those alternatives or request a decision. It must not select one because its serialized representation sorts first.

5.4 Keyed data diff and identity ambiguity

The first data-diff profile is finite and keyed. It reports inserted and deleted keys, updated fields with before/after digests, rekeys, changed source-local identity, assertions, retractions, and source snapshot changes. If no stable key exists, the result is an ambiguity set or `unsupported`. Positional row comparison cannot establish semantic identity.

This restriction is intentional. P7 treats candidate generation, probable equivalence, confirmed sameness, confirmed difference, contradiction, and supersession as separate artifacts. A P9 diff may report changed identity evidence and decisions, but it cannot turn similarity into an identity merge.

5.5 Impact closure

Let $G = (V, E)$ be the versioned typed dependency graph and $X \subseteq V$ the changed artifact set. Define the conservative closure

$$\mathsf{Impact}_G(X) = \mu Y.\, X \cup \{v \mid \exists u \in Y.\, (u, v) \in E \wedge \mathsf{propagates}(u, v)\}.$$

Propagation is change-specific. An unused nullable attribute addition may leave a projection valid. A wildcard projection makes the same addition relevant. A stable-ID display rename may preserve a mapping, while a type narrowing may invalidate the mapping and existing data. A changed identity decision may alter aggregate groups. A stricter provenance profile can invalidate a data-equivalent materialization.

Every impact edge reports whether it is direct or transitive; compiler emitted, user declared, inferred, or wildcard; and valid, stale, or unknown. It also names the propagation rule and source revision.

Open conjecture 5.2 (Impact soundness for a frozen fragment). If an admitted change can alter the normalized denotation or admissibility of a registered dependent artifact, that artifact belongs to the computed impact set.

Evidence status. OPEN CONJECTURE. The dependency language and proof fragment are not frozen. One false negative would refute the statement for that profile.

Impact precision is empirical. False positives cost review and recomputation; they cannot be dismissed as harmless. The evaluation must report recall and precision against a preregistered seeded oracle.

6 Three-way semantic merge

6.1 Inputs and outcomes

Definition 6.1 (Merge request)**.** A merge request is

$$\begin{aligned} M_q = (&p, c_b, c_t, c_s, \rho, g, u, a, q, \sigma), \\ &p : \mathsf{Profile}, \quad c_b, c_t, c_s : \mathsf{CommitId}, \\ &\rho : \mathsf{TargetRef}, \quad g : \mathsf{ExpectedGeneration}, \\ &u : \mathsf{Principal}, \quad a : \mathsf{AuthorityRevision}, \\ &q : \mathsf{PolicyRevision}, \quad \sigma : \mathsf{RequestedStrategy}. \end{aligned}$$

The merge base is explicit. If ancestry yields several candidate bases, the profile must select or construct one and record that choice. The output is:

$$\begin{aligned}\mathsf{Merge}_p(B,T,S) \in\ & \mathsf{Merged}(C,\Pi,\Gamma)\ +\ \mathsf{Conflicts}(K)\\ & +\ \mathsf{Reconciliation}(R)\ +\ \mathsf{Unsupported}(U)\ +\ \mathsf{Rejected}(E).\end{aligned}$$

Here C is a candidate commit, Π its validation and approval record, and Γ decision provenance.

6.2 Merge stages

The procedure has twelve ordered stages:

1. resolve and verify base, target, and source commits;
2. compute or load typed diffs from the base;
3. expand semantic footprints and dependency impacts;
4. classify every relevant operation pair;
5. apply reviewed automatic rules;
6. emit typed conflicts and reconciliation requests;
7. apply authorized resolutions to a candidate state;
8. validate the entire candidate state;
9. recompute materialization validity and historical availability;
10. stage immutable artifacts, commit, and reports;
11. compare-and-swap the target reference; and
12. record publication provenance.

Requirement 6.2 (Closed automatic merge)**.** An operation pair may auto-merge only if a rule exists in the exact profile, the pair is well typed at the base, all rule preconditions hold, noninterference or commutativity is checked, authority and policy agree, dependency effects are included, and the complete resulting state validates. An unclassified pair returns `unsupported-pair`.

Different files, rows, or logical subjects can support an independence argument. None is sufficient for every operation kind.

6.3 Conflict records

Definition 6.3 (Conflict record)**.** A conflict record is

$$k = (i, c, B, T, S, A, W, R, V, s, z),$$

where i is a conflict ID, c the conflict class, B base references, T and S target/source operations, A affected artifacts, W witnesses, R allowed resolutions, V required validations, s status, and z an optional resolution reference. Required authority is part of the record.

Table 6 lists the core classes.

Table 6: Core semantic conflict taxonomy.

Class	Example	Default outcome
same-subject divergent edit	both branches replace one mapping path differently	conflict
delete/modify	target removes an attribute that source retypes	conflict
rename/rename	one stable ID receives two display names	policy or conflict
rename/remove-add ambiguity	one branch renames; the other creates a lookalike	reconciliation
type/domain incompatibility	one branch widens an integer while the other parses it as a date	conflict
equation violation	combined paths violate a declared schema equation	invalid candidate
constraint/data conflict	non-null strengthens while branch data still contain null	repair or block
schema/mapping conflict	a mapping image references a removed path	adaptation or conflict
migration/direction conflict	branches assume incompatible read directions	conflict
mapping coverage conflict	one branch excludes newly admitted source rows	policy review
data same-key conflict	both branches update one field differently	field conflict
data rekey conflict	two records claim one stable key	reconciliation
identity merge/split conflict	one confirms same; the other confirms different	reconciliation required
decision authority conflict	the approving actor lacks scope	reject decision
provenance completeness conflict	a stricter profile lacks retained evidence	block or degrade
retention or reproduction conflict	a branch expires a source needed by a cited result	policy conflict
materialization conflict	one branch reuses state invalidated by the other	rebuild or conflict
policy conflict	branches impose incompatible obligations	governance resolution
physical-only conflict	plans differ while semantic digest agrees	execution choice
unsupported operation pair	opaque transform combined with a schema split	unsupported or manual

The intended CatDB delta lies mainly in cross-object conflicts. Decibel already treats tuple and field conflicts [19]. P9 must show value on cases such as schema retype plus mapping edit, identity decision plus aggregate materialization, provenance-policy change plus retention event, and migration plus branch-local data operations.

6.4 Resolution and preserved disagreement

A conflict may describe two legitimate alternatives that cannot be combined automatically. The system preserves both branches and their evidence. Branch-local queries can continue while the target merge remains blocked.

Definition 6.4 (Conflict resolution)**.** A resolution contains the conflict reference, selected or synthesized effect, rejected alternatives, reason, evidence, authority, approvals, validations, reversibility declaration, and provenance.

Resolution never deletes the conflict history. If it is declared reversible, the inverse or compensating procedure and its affected-artifact policy must be explicit.

6.5 Why merge is not a pushout by default

Pushouts and other colimits can model selected integrations under a stated category and class of morphisms [24]. The operational merge above may also contain partial or lossy mappings, probabilistic identity evidence, authority, data constraints, retention policy, materialization state, and an atomic publication step. The paper does not call every merge a pushout.

A concrete counterexample is a base customer object changed on one branch into separate Person and Organization objects while another branch retains the customer object but records two plausible, authority-dependent identity assignments for one source record. The merge must preserve both candidates and synthesize a resolution artifact awaiting authority; there is no declared unique mediating morphism that chooses one candidate. That conflict-bearing result is not asserted to satisfy a pushout universal property.

If a future formal result uses a categorical construction, it must state the objects and morphisms, the diagram, existence conditions, quotient policy, and operational interpretation. Other merges remain partial procedures with validation and conflict outcomes.

6.6 Candidate merge laws

The following are targets, not results:

Open conjecture 6.5 (Identity merge law). For a valid base state B, merging B with two unchanged descendants returns a state extensionally equal to B.

Open conjecture 6.6 (One-sided merge law). If T is a valid admitted descendant of B and $S = B$, then $\mathsf{Merge}(B, T, S)$ returns a state extensionally equal to T, subject to publication authority and current policy.

Open conjecture 6.7 (Conservative automatic merge). Every pair accepted by the closed automatic rule table commutes or is noninterfering under its stated preconditions, and complete validation preserves the admitted global invariants.

Evidence status. All three statements are OPEN CONJECTURE. No P9 Lean declarations, Haskell properties, Rust implementation, or exhaustive pair cases exist.

7 Co-versioned semantic state

7.1 The compatibility bundle

A workspace state is not just a schema and rows. The proposed bundle is

$$\begin{aligned}\mathcal{B} &= (S, M, G, I, D, Q, P, R, A, X, F),\\ S &= \text{schema revisions},\\ M &= \text{mapping and migration revisions},\\ G &= \text{typed dependency graph},\\ I &= \text{data observations and overlay operations},\\ D &= \text{identity and reconciliation decisions},\\ Q &= \text{logical query definitions},\\ P &= \text{provenance profile and retained evidence},\\ R &= \text{materialization definitions and status},\\ A &= \text{authority and approval state},\\ X &= \text{source snapshots and connector observations},\\ F &= \text{freshness and completeness envelope}.\end{aligned}$$

Validity means that all cross-references and profile obligations hold.

7.2 Schema changes with data

A schema operation classifies its data response as:

- unchanged and still valid;
- transformed by an admitted migration;
- available through a compatibility view;
- partially valid with explicit rejected rows;
- requiring backfill, synthesis, or reconciliation;
- invalid pending repair; or
- unsupported.

Each branch-local data operation records the schema and identity context in which it was typed. A rebase or merge revalidates or migrates the operation. Matching field names do not justify blind replay against a different schema.

7.3 Mapping changes with data

A mapping revision must report:

- whether source and target schema revisions still match;
- whether the mapping is total over its declared domain;
- which source observations are reinterpreted;
- whether canonical identities or values change;
- which queries and materializations are affected;
- whether old provenance explanations remain meaningful; and
- whether physical rematerialization is required.

This is the practical meaning of mapping-aware merge in P9. It does not mean that all adapted mappings are unique or safe.

7.4 Branch-local constraints

Each workspace validates its own declared profile. Two branches can both be valid while imposing incompatible constraints. Team A may require unique canonical customer identity after its merge decision; Team B may require that the same source records remain separate. Independent validity is not merge compatibility.

A branch-local constraint record includes its logical identity, revision, scope, dependency footprint, authority, validation procedure, and evidence. Promoting the branch to a shared ref requires revalidation against the candidate commit, not reuse of a stale branch-local pass.

7.5 Alternative interpretations

An interpretation branch may change mapping choice, unit normalization, identity decision, trust policy, ontology, schema constraint, query definition, or materialization strategy while referencing the same immutable source observations.

Requirement 7.1 (Interpretation isolation)**.** Results from interpretation I_A may enter I_B only through an explicit import, mapping, or merge operation that records source and target contexts, transformation, provenance, authority, loss, ambiguity, validation, and conflict behavior.

A comparison across interpretations reports shared evidence, changed artifacts, output differences, provenance differences, the assumptions responsible for those differences, and unresolved ambiguity. Promoting one interpretation to a canonical branch is an authority-bearing reference update. It is not a claim that the interpretation is universally true.

Open conjecture 7.2 (Safe coexistence for the admitted local profile)*.* Alternative interpretations over retained source snapshots produce context-labelled results without leaking branch-local operations or decisions.

Evidence status. OBSTRUCTED. No workspace evaluator, isolation boundary, or provenance audit exists.

7.6 Identity decisions and downstream effects

P7 separates evidence, method output, and authority-bearing decision. P9 adds workspace context and effective decision projection. Confirming two source records as one canonical entity may merge aggregate groups, change uniqueness constraints, redirect relationships, alter coverage interpretation, invalidate caches, change explanations, and make write-back ambiguous.

If one branch confirms sameness and another confirms difference, the merge returns a typed contradiction. Last-writer-wins is not admitted. Reversing a decision creates a new event, restores or creates separate canonical projections under P7 rules, invalidates dependents, preserves prior result provenance, and reports any information that cannot be separated automatically.

8 Provenance, materialization, history, and rollback

8.1 Commit and result provenance

Commit provenance should answer:

- who or what proposed each operation;
- which base and source snapshots were used;
- which schema, mapping, migration, identity, query, and policy revisions applied;
- which validations ran over which candidate digest;
- which approvals and obligations were effective;
- how conflicts were resolved;
- which derived artifacts became invalid; and
- which target reference was published.

Every workspace query result names the workspace and overlay digest or the immutable commit, source snapshots, schema and mapping revisions, identity projection, provenance profile, query revision, materialization or plan if used, freshness, approximations, unsupported components, and retention or redaction boundary.

W3C PROV can carry an interoperability projection: commits and artifacts as `prov:Entity`, executions as `prov:Activity`, principals as `prov:Agent`, and revision, use, generation, invalidation, and delegation through the corresponding relations [18]. The projection is not the internal workspace semantics. A loss audit must identify CatDB distinctions that the projection cannot express exactly.

8.2 Materialization definition versus realization

The semantic commit versions the logical query or mapping definition, source coverage and frontier policy, provenance profile, consistency and freshness requirements, allowed execution

strategies, and dependency edges. Stored rows, indexes, cache entries, backend plans, and temporary overlay state are replaceable realizations.

A materialization status records:

$$(r_d, i_r, f_s, d_l, d_p, s, v, u),$$

where r_d is the definition revision, i_r a realization ID, f_s the source frontier, d_l the logical digest, d_p the provenance digest, s status, v validation, and u an optional reuse or rebuild rule.

After a merge, each materialization becomes one of:

- unaffected and reusable;
- reusable after validated adaptation;
- stale with an explicit frontier;
- invalid;
- rebuild required;
- provenance re-derivation required;
- retired; or
- unsupported.

Physical row differences need no semantic merge when both realizations can be discarded and rebuilt. The semantic conflict lies in definitions, dependencies, frontiers, provenance requirements, or promises made to readers.

8.3 Historical query context

Definition 8.1 (Historical query context)**.** A historical query context is

$$\begin{aligned} K = (&c, w, q, x, s, m, d, p, a, b, r, n),\\ &c : \mathsf{CommitId}, \quad w : \mathsf{OptionalOverlayDigest},\\ &q : \mathsf{LogicalQueryRevision}, \quad x : \mathsf{SourceSnapshotSet},\\ &s : \mathsf{SchemaRevisionSet}, \quad m : \mathsf{MappingMigrationRevisionSet},\\ &d : \mathsf{IdentityProjectionRevision}, \quad p : \mathsf{ProvenanceProfile},\\ &a : \mathsf{PolicyRevision}, \quad b : \mathsf{BackendSemanticsProfile},\\ &r : \mathsf{MaterializationPolicy}, \quad n : \mathsf{NondeterminismProfile}. \end{aligned}$$

Commit ancestry, system time, valid time, source cursor, and materialization frontier are distinct. A query may request a commit state C, facts valid at domain time t_v, decisions known at system time t_s, source observations through frontier f, and a materialization at checkpoint k. The response names every cut.

Re-executing a current query against an old commit is not the same as executing the historical query revision. Both operations can be useful, but the caller must choose explicitly.

8.4 Reconstruction, replay, and reproduction

Goal	Meaning	Minimum evidence
state reconstruction	rebuild the normalized semantic bundle	commits, artifacts, and operations or snapshots
query replay	execute the same logical query under the same declared context	reconstructed state, query revision, and execution semantics
result reproduction	obtain the same normalized data and provenance and explain nondeterminism	replay evidence plus retained external observations and dependencies

Table 7: Three historical goals that must not share one success label.

A reproduction envelope contains the result and commit IDs, canonical query, source snapshots, artifact and semantic-profile revisions, connector and backend versions, external-function digests, materialization and plan references, normalized result and provenance digests, environment manifest, retention status, and known nondeterminism.

For retained context K, exact reproduction requires:

$$\mathsf{Normalize}(\mathsf{execute}(q, K)) = (d, p),$$

where both the normalized data digest d and provenance digest p match the stored result. Equal data with different provenance is not exact P13-enabled reproduction.

The machine-readable status set includes:

- `exact-reproduction`;
- `exact-reconstruction-new-plan`;
- `semantically-equivalent-under-profile`;
- `partial-source`;
- `stale-materialization`;
- `provenance-incomplete`;
- `redacted`;
- `nondeterministic`;
- `unavailable-retention`;
- `unsupported-backend-semantics`; and
- `blocked`.

Immutable history does not override lawful deletion, confidentiality, access, or retention policy. The system may retain a tombstone, deletion event, non-sensitive digest, redacted provenance skeleton, or only an authorized audit record. The reproduction status must change accordingly.

8.5 Rollback and compensation

P9 distinguishes four operations:

Reference rollback.
An authorized mutable ref moves from C_n to an ancestor C_k. The event is recorded and immutable commits remain.

Revert commit.
A new commit applies admitted inverse or compensating operations on top of the current head:

$$C_{n+1} = \mathsf{commit}(C_n, \mathsf{compensate}(C_k \to C_n)).$$

Workspace discard.
A draft workspace becomes abandoned without changing its base commit.

Physical compensation.
Backend DDL, external writes, messages, and source-system effects use their own transaction or compensation protocol.

Every attempted external compensation creates an immutable `CompensationReceipt` artifact linked to the triggering revert commit. The receipt records the external-effect reference, connector and capability revision, idempotency key, declared footprint, pre- and post-observations, outcome $\{\texttt{succeeded}, \texttt{failed}, \texttt{unknown}\}$, authority, timestamps, and provenance. A `succeeded` receipt supports restoration only for the declared footprint and post-condition actually checked; it does not prove that the whole external system equals an earlier state. A missing or `unknown` receipt blocks any exact-restoration status and remains a visible unresolved artifact in the commit graph.

A semantic ref movement cannot claim to reverse external side effects. Rollback reclassifies materializations, checks source-frontier compatibility, preserves identity and provenance history, reevaluates policy, and keeps the original context of historical results.

9 Candidate results and formal obligations

9.1 Status of this section

This section contains no established P9 theorem. It defines the smallest fragment in which selected claims could become machine checked. Haskell is the proposed executable semantic oracle; Lean is the kernel-checked authority for selected pure laws; Rust is the proposed production implementation. Agreement among them would provide valuable evidence, but the evidence types remain different.

The first formal fragment is finite. It includes artifact kinds and revision references, canonical commit bodies, schemas and lawful mappings from the accepted CatDB fragment, keyed finite instances, typed dependency graphs, operations with decidable preconditions, base-plus-overlay workspaces, a closed conflict enumeration, and a finite automatic-merge table. It excludes arbitrary SQL, opaque functions, probabilistic matching algorithms, cryptographic

hash security, access-control correctness, persistent storage, wall-clock and distributed time, external connectors, cost optimization, and complete scientific reproducibility.

9.2 Haskell reference semantics

The planned Haskell surface is:

```
canonicalArtifact :: Profile -> ArtifactRevision
                  -> Either Error Bytes
canonicalCommit :: Profile -> SemanticCommit
                  -> Either Error Bytes
applyOperation :: Profile -> State -> Operation
                  -> Either Outcome State
normalizeOverlay :: Profile -> [Operation]
                  -> Either Error [Operation]
materialize :: Profile -> CommitState -> [Operation]
                  -> Either Outcome State
semanticDiff :: Profile -> State -> State
                  -> Either Outcome SemanticDiff
impactClosure :: Profile -> DependencyGraph -> Set Ref
                  -> ImpactReport
classifyPair :: Profile -> State -> Operation -> Operation
                  -> PairResult
merge3 :: Profile -> State -> State -> State
                  -> MergeResult
```

The property suite should test:

1. idempotence of canonicalization;
2. invariance under permutation of set-like members;
3. declared duplicate-operation replay behavior;
4. denotational preservation by overlay normalization;
5. operational-diff reconstruction from an ancestor;
6. state-diff reconstruction up to extensional equality;
7. commutativity for every pair labelled commuting;
8. precondition preservation for every pair labelled independent;
9. identity and one-sided merge laws;
10. refusal of unclassified operation pairs;
11. rejection by full validation when local pair checks are insufficient;
12. explicit commit or workspace context in result provenance; and
13. exact normalized equality with Rust fixtures.

Passing these properties would be *computational evidence* for the tested inputs and generators, not proof of the production runtime or real sources. No P9 property currently exists.

9.3 Lean targets

Table 8 lists the proposed proof obligations.

Table 8: Proposed Lean targets. Every row remains OPEN CONJECTURE.

ID	Target	Exact boundary
P9-F01	canonicalization idempotence	pure finite representation, not byte codec or hash security
P9-F02	normalization preservation	selected overlay rewrites preserve pure state and retained audit projection
P9-F03	materialization-fold determinism	fixed ordered operations over deterministic partial application
P9-F04	ancestor diff reconstruction	retained admitted operation sequence reconstructs the descendant
P9-F05	state-diff reconstruction	finite keyed states under one frozen diff algorithm
P9-F06	independent-operation commutativity	exact named operation pairs under explicit footprint assumptions
P9-F07	merge identity and one-sided laws	valid base and one-sided admitted changes
P9-F08	local well-formedness preservation	selected schema and mapping laws after admitted auto-merge
P9-F09	conflict-table completeness	total classification over the closed pair enumeration
P9-F10	pure context noninterference	separate state maps and explicit context, not storage isolation

Even successful proofs would not establish SHA-256 collision resistance, JSON codec correctness, SQLite durability, atomic publication under hardware failure, authorization soundness, human reconciliation correctness, provenance completeness from external sources, source retention, performance, or distributed convergence.

9.4 Cross-language acceptance

Each pure fixture follows:

$$\begin{aligned} \text{JSON fixture} &\longrightarrow \text{Haskell normalized result} \\ &\longleftrightarrow \text{Rust normalized result} \\ &\longrightarrow \text{Lean declaration where available.} \end{aligned}$$

The manifest records claim anchor, assumptions, expected status, Rust and Haskell tests, optional Lean declaration, unsupported boundary, and source revision. A blank Lean field means "not machine checked."

Open conjecture 9.1 (Overlay/materialized equivalence). For every workspace, query, and operation in the frozen finite profile, the optimized base-plus-overlay evaluator returns the same normalized data and provenance as query evaluation over the canonical materialized workspace.

Evidence status. OBSTRUCTED. This is P9-C02 and `SYS-WS-01`; no evaluator or fixture profile exists.

Open conjecture 9.2 (Pure workspace noninterference). Given explicit disjoint workspace contexts, evaluation in W_A is independent of unimported overlay operations and branch-local decisions in W_B.

Evidence status. OPEN CONJECTURE for a pure evaluator and OBSTRUCTED for the storage and authorization system.

10 Implementation roadmap

10.1 Architecture boundary

The implementation sequence refines the shared Slice 12 plan. It does not replace it. The semantic model belongs in immutable IR and workspace interfaces; SQLite is the first durable realization. Storage may persist artifacts and references, but it does not decide schema validity, mapping lawfulness, identity authority, or provenance completeness.

10.2 Stage I0: freeze prerequisites and profile

P3, P4, P6, P7, and P13 must publish compatible finite contracts. P9 then freezes artifact classes, identity layers, operation kinds, conflict classes, authority, provenance, materialization boundaries, unsupported operations, fixture version, diagnostic codes, threat model, and crash model.
Evidence status. OBSTRUCTED.

10.3 Stage I1: canonical artifacts, commits, and refs

Implement logical and revision IDs, domain-separated digests, typed dependencies, semantic commit bodies, mutable refs with generation, canonical serialization, closed diagnostics, and profile versioning. Rust and Haskell must produce identical canonical bytes and identifiers under permutation fixtures.
Evidence status. OPEN.

10.4 Stage I2: immutable store and atomic refs

Implement put-if-absent artifact and commit storage, digest verification, reachability audit, transactional SQLite reference storage, compare-and-swap publication, recovery receipts, and

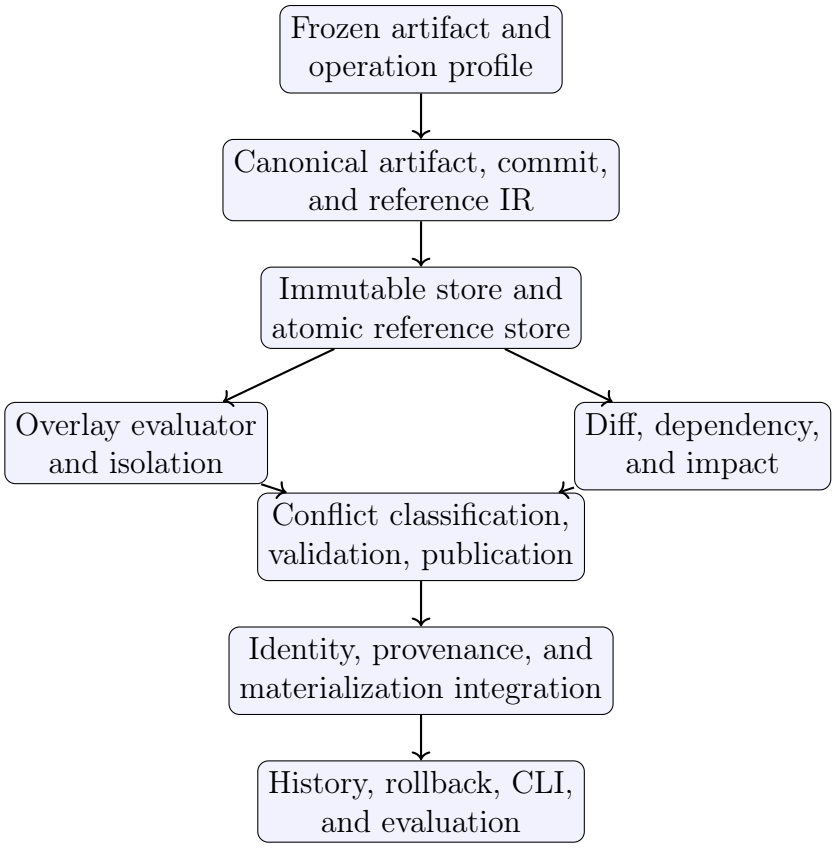

Figure 1: Proposed local implementation order. The cross-object layers enter only after commit, overlay, diff, and publication contracts are fixed.

a retention-aware garbage-collection design. The crash matrix covers every boundary before and after artifact staging, commit persistence, reference update, and acknowledgement.
Evidence status. OPEN.

10.5 Stage I3: workspace and overlay

Create `catdb-workspace` with the state machine, typed operation application, deterministic normalization, validation-head invalidation, explicit query context, canonical materialized-state evaluator, and optimized overlay evaluator. Concurrency schedules must test isolation from branch-local rows, decisions, validation state, approvals, and cache effects.
Evidence status. OBSTRUCTED.

10.6 Stage I4: diff, dependencies, and impact

Implement all five diff layers, exact separation of recorded and inferred changes, typed impact closure, compiler and user edge provenance, wildcard dependencies, and conservative unknown handling. Mutation tests remove dependency edges and validators to confirm that the oracle catches omissions.
Evidence status. OBSTRUCTED.

10.7 Stage I5: merge and publication

Implement merge-base resolution, the reviewed pair table, footprint expansion, precondition and authority checks, typed conflicts, resolution operations, complete candidate validation, approval coverage over the exact digest, and atomic publication through Stage I2.
Evidence status. OBSTRUCTED.

10.8 Stages I6 through I11

I6, cross-object integration.
Add branch-local identity projections, contradictions, reversal, result provenance, completeness and retention, materialization invalidation, reuse, and rebuild.

I7, history and rollback.
Add explicit temporal cuts, reproduction envelopes, degraded outcomes, ref rollback, revert, and compensation.

I8, SQLite realization.
Persist append-only artifacts and commits, transactional refs, workspace operations, dependencies, conflicts, validations, approvals, and failpoint receipts.

I9, CLI.
Expose create, apply, validate, diff, query, merge, resolve, publish, history, reproduce, and audit through closed machine-readable envelopes.

I10, hardening.
Add model-based tests, fuzzing, mutation tests, concurrency schedules, failpoints, authorization time-of-check/time-of-use cases, digest confusion, profile-crossing, malicious cycles, resource limits, and redaction noninterference.

I11, D7 harness.
Implement the common relational subset and baseline adapters, preserve failed observations, and run correctness before performance.

All six stages are OPEN or OBSTRUCTED.

10.9 Command-line surface

The planned deterministic commands are:

```
catdb workspace create --base <commit>
catdb workspace apply --workspace <id> --ops <file>
catdb workspace validate --workspace <id>
catdb workspace diff <from> <to>
catdb workspace query --workspace <id> --query <file>
catdb workspace merge --base <id> --target <id> --source <id>
catdb workspace resolve --conflict <id> --resolution <file>
catdb workspace publish --workspace <id> --ref <name> --expect <generation>
catdb workspace history --ref <name>
catdb workspace reproduce --result <id>
catdb workspace audit --commit <id>
```

The CLI must not infer a default workspace for branch-local or historical queries. Human-readable output is a rendering of the closed envelope, not an alternative semantics.

11 Evaluation and falsification plan

11.1 Research questions

ID	Question
RQ1	Does base-plus-overlay query equal canonical materialization for every admitted operation and query case?
RQ2	Are workspace reads isolated under concurrent schedules and shared caches?
RQ3	How sound and precise are semantic diff, impact closure, and conflict classification on seeded changes?
RQ4	Does automatic merge remain deterministic, conservative, and globally valid?
RQ5	Is merge publication atomic across every declared crash point and ref race?
RQ6	What storage, checkout, diff, merge, and historical-query cost does CatDB add on the shared relational subset?
RQ7	Does cross-object metadata detect faults or reduce diagnosis and repair effort beyond data-only versioning?
RQ8	Under what retained inputs can historical results be reproduced, reconstructed, degraded, or not recovered?
RQ9	Can alternative interpretations remain isolated while supporting explicit, provenance-bearing comparison?

Table 9: P9 evaluation questions.

11.2 Correctness gate

Performance measurements stop if any run exhibits:

- canonical commit mismatch;
- overlay/materialized data or provenance mismatch;
- workspace leakage;
- omitted preregistered impact;
- incorrect automatic merge or untyped supported conflict;
- incomplete merged-state validation;
- unauthorized identity decision;
- stale materialization reported as current;
- partial target-reference publication;

- historical reproduction misclassification; or
- raw-artifact integrity failure.

A slower correct execution remains evidence. A faster incorrect execution is rejected.

11.3 Baselines and fairness

Baseline	Fair comparison boundary
Decibel	relational branch, checkout, difference, merge, version query, and storage; released artifact if reproducible, otherwise labelled reconstruction
ORPHEUSDB	relational version storage, retrieval, query, and version operations under the same artifact/reconstruction rule
full-copy SQL	one table copy per version in SQLite or PostgreSQL
row-delta SQL	parent-linked keyed row deltas without CatDB semantics
Git files	canonical JSON storage and text workflow only; a negative semantic baseline, not database-query equivalence
CatDB data-only ablation	commit engine with cross-object features disabled to isolate their overhead and value

Table 10: Evaluation baselines. CatDB-only semantics cannot be used to claim that a prior system fails requirements outside its scope.

Published performance numbers are not executable baselines by themselves. If an original artifact cannot run in the pinned environment, the evaluation uses a reviewed reconstruction and reports the difference.

11.4 Workload dimensions

Generate version graphs across:

- graph depth, branch width, merge density, and changed fraction;
- data size, row width, key skew, and hot-key concentration;
- schema object, path, equation, and constraint counts;
- mapping count, path depth, coverage, loss, and ambiguity;
- dependency fan-out and wildcard use;
- identity evidence, decision, contradiction, and reversal count;
- provenance density, completeness, redaction, and retention;
- materialization count and source-frontier skew;
- policy and approval count; and

- conflict mix: independent, commuting, direct conflict, reconciliation-required, globally invalid, and unsupported.

Every generated history retains its seed and canonical manifest. Workload families cover the relational shared subset, schema/data evolution, mapping evolution, identity evolution, provenance and retention, materialization evolution, alternative interpretations, crash recovery, and historical reproduction.

11.5 Correctness metrics

Report:

- exact normalized data and provenance agreement;
- workspace leakage count;
- diff reconstruction;
- changed-artifact and impact recall;
- impact precision;
- conflict-class precision, recall, and per-class confusion;
- automatic-merge false positives;
- unsupported-pair and global-validator escape rates;
- unauthorized-decision acceptance;
- stale-materialization false-current count;
- crash outcome membership and lost acknowledged publications;
- historical-status confusion; and
- missing interpretation context.

Deterministic correctness invariants require zero observed violations within the reported workload bounds. Zero failures do not prove correctness outside those bounds.

11.6 Performance and storage metrics

Measure commit canonicalization and validation, artifact and commit bytes, metadata/index amplification, version-storage amplification, branch creation, checkout, overlay and materialized query latency, diff latency by layer, impact closure, merge classification, full validation, publication, history query, reproduction replay, invalidation and rebuild work, memory, CPU, and bytes read or written.

Report medians, tail percentiles, dispersion, cold and warm cache, hardware, database settings, and confidence intervals. Do not collapse every operation into one "workspace overhead" number.

11.7 Seeded fault and reviewer study

The blinded fault corpus includes true independent edits, direct conflicts, cross-object conflicts, global-only violations, unauthorized decisions, stale caches, missing provenance, retention failures, unknown operation pairs, and benign physical-only differences. Database/versioning and dependency-domain reviewers freeze the oracle before running the classifier.

If the paper claims reduced review or repair effort, a preregistered study compares:

1. Git/file diff plus relational data diff;
2. data-only version control;
3. CatDB structural diff; and
4. the complete CatDB impact, validation, and conflict explanation.

Participants identify affected artifacts, merge safety, conflict class, missing evidence or authority, required rebuild, and historical reproducibility. The study reports accuracy, time, confidence calibration, repair steps, expertise, order effects, and inter-reviewer agreement. A convenience sample supports only an exploratory claim.

11.8 Crash protocol

For each failpoint:

1. begin from a verified old head;
2. stage a deterministic candidate;
3. kill the process at the instrumented boundary;
4. restart in a fresh process;
5. verify reachable digests;
6. read the target ref and generation;
7. audit commit completeness, approval, validation, and provenance; and
8. preserve the database, log, and observed outcome.

The target ref may name the old complete commit or the new complete commit. It may not name a partial state. Complete but unreachable staged artifacts are allowed and reported.

11.9 Historical reproduction protocol

For each sampled result, freeze the envelope, execute and store normalized data/provenance digests, rerun in the pinned environment, rerun with a semantically permitted different plan, remove or alter one dependency, and require the status classifier to explain the outcome. An independent clean-room run preserves both successes and failures.

11.10 Statistical and reporting plan

Primary metrics, comparisons, thresholds, and exclusions are frozen before confirmatory execution. All runs, failures, and timeouts remain in the artifact. Paired comparisons use identical generated histories when possible. Skewed distributions use preregistered bootstrap intervals or a robust alternative. The report gives effect sizes and raw distributions, not only p-values. Noninferiority or usability thresholds come from pilot data and external requirements before confirmatory outcomes are inspected.

11.11 Claim-level falsification

Table 11: Primary claim anchors and falsifiers.

Anchor	Candidate claim	Immediate falsifier
P9-C01	deterministic commit identity across Rust and Haskell	different canonical bytes or IDs for one valid body/profile
P9-C02	overlay query equals canonical materialization	any normalized data or provenance mismatch
P9-C03	workspace isolation	one leaked row, decision, validation, approval, or cache effect
P9-C04	typed diff reconstructs admitted descendant	state mismatch, omitted artifact, invented intent, or unstable order
P9-C05	conservative impact closure	one affected registered dependent is absent
P9-C06	conservative automatic merge	order dependence, authority breach, global violation, or missing rule
P9-C07	atomic publication	published dangling ref, mixed state, or lost acknowledged update
P9-C08	cross-object conflict classification	supported conflict auto-merges or unknown receives permissive default
P9-C09	reconciliation-bearing merge	decision lacks evidence/authority or reversal leaves stale dependents
P9-C10	evidence-based history status	exact reproduction reported after a required input is substituted or lost
P9-C11	alternative-interpretation isolation	context leakage or unsupported compatibility inference
P9-C12	rollback preserves audit boundary	silent history rewrite or false claim that external effects were undone
P9-C13	value beyond relational versioning	no added fault/review value or unacceptable false-positive burden

Every anchor is OPEN or OBSTRUCTED. None is a result of this manuscript.

12 Related work revisited

12.1 Relational version-control systems

Decibel is the closest direct baseline for branch semantics and relational three-way merge [19]. It makes the distinction from line-oriented file versioning concrete: relations are unordered, primary keys matter, versions share data, and users query across histories. ORPHEUSDB shows that version control can be layered over a mature relational engine and that physical representation and partitioning dominate important performance tradeoffs [16]. DataHub and its versioning work place those operations in collaborative analysis [3, 4].

P9 adopts immutable commits, branch references, ancestry, working overlays, and three-way comparison. Its proposed difference is the commit atom and conflict boundary across semantic artifacts. That difference must be measured. Merely adding metadata files to a relational version graph would not establish it.

12.2 Schema and mapping evolution

PRISM, PRISM++, and PRIMA demonstrate that schema evolution can be represented through operators, mappings, query rewriting, and histories rather than treated only as ordered DDL [8, 9, 20]. Mapping-adaptation work shows that an evolved mapping may require composition, pruning, or a user decision [27–29]. InVerDa demonstrates coexisting schema versions over one dataset [15].

P9's schema/data co-versioning and mapping-aware merge are therefore extensions of established lines, not a new research category. The proposed systems contribution must be the exact compatibility bundle, conservative cross-object impacts, and atomic publication.

12.3 Model management and categorical structure

Model management supplies a broad operator vocabulary for models and mappings [2]. Algebraic model management asks when those operators admit algebraic or categorical descriptions [24]. Functorial data migration supplies a precise categorical foundation for selected schema mappings and instance transformations [26]. These works motivate typed paths and compositional transformations in CatDB. They also create strong novelty pressure.

The operational workspace includes evidence, authority, partiality, source availability, materialization state, and storage publication. Category theory may prove laws for a selected pure fragment; it does not discharge those operational obligations by itself.

12.4 Temporal data and reproducibility

Temporal databases separate system and application time [17]; P9 adds commit ancestry, source frontiers, and branch context rather than replacing temporal dimensions. Scientific archiving and dynamic data citation show that later verification needs stable history and the

query that selected a subset [5, 22, 23]. VQuel demonstrates that provenance and versioning can share a query language [6].

P9's contribution, if validated, would be a more explicit reproduction envelope for CatDB's semantic artifacts. It cannot claim that versioned provenance or query-based citation is new.

12.5 Provenance and materialized state

W3C PROV gives a standard interchange vocabulary [18]; provenance semirings give compositional annotations for positive relational queries [13]; PROV-IDEA joins schema and data provenance during evolution [21]. P9 relies on P13 to define the exact internal provenance profile and any external projection.

Materialized-view adaptation establishes that selected changed definitions can reuse earlier realizations [14]. P9 adds branch and cross-artifact context, but must still prove or test each reuse rule.

13 Limitations and exact nonclaims

The current design has several hard boundaries.

13.1 Semantic limitations

The proposed stable IDs require governance. They do not prove that two revisions retain the same meaning, and they do not resolve independently assigned IDs across administrative domains. The initial data diff assumes finite keyed data. Records without stable keys remain ambiguous or unsupported. The first operation-pair table is closed and will classify many real changes as manual or unsupported.

Mapping semantics can be partial, lossy, ambiguous, and direction-sensitive. A semantic merge cannot reconstruct stakeholder intent that was never recorded. A state-derived diff reconstructs a state relation, not the historical operation or the reason for it. Inferred explanations remain hypotheses.

Branch-local validation does not imply merge compatibility. Pairwise commutativity does not imply that the final state satisfies global constraints. The complete validator and dependency registry become part of the trusted computing base for automatic merge.

13.2 Systems limitations

The local proposal does not supply distributed convergence, causal consistency, consensus, or a CAP result. Those questions depend on P10 and P11 after P9 freezes the operation algebra. SQLite failpoint evidence would not establish PostgreSQL, object-store, or distributed durability. A process-kill matrix cannot simulate every filesystem, kernel, power, or hardware failure.

Atomic reference publication does not atomically reverse or coordinate external source effects. Write-through and compensation require P8 and backend-specific protocols. Garbage

collection conflicts with retention, legal deletion, and audit requirements; the design does not solve that policy problem.

13.3 Formal limitations

The proposed Lean fragment covers finite pure representations and selected laws. It excludes codecs, cryptographic assumptions, storage, authorization, external provenance, probabilistic reconciliation, and performance. Haskell property tests would supply computational evidence, not proof. A kernel checked theorem would not verify the Rust implementation unless a separately verified correspondence were established.

13.4 Experimental limitations

Decibel and ORPHEUSDB may require reconstructed baselines. Generated version histories may not reflect real organizational conflicts. Stable keys make the first profile easier than general entity resolution. A richer conflict classifier may produce too many false positives. Human participants may learn the classifier or differ sharply in expertise. A dependency oracle could repeat the implementation's assumptions. One backend, graph shape, or conflict mix cannot support a universal performance claim.

13.5 Exact nonclaims

The paper does not claim:

1. that Git file diff is a semantic database merge;
2. invention of database branching, dataset version control, schema versioning, mapping adaptation, temporal query, provenance, or data citation;
3. that stable logical IDs prove semantic continuity;
4. that a digest proves equivalence, truth, authorship, authority, completeness, or availability;
5. recovery of unrecorded intent from a typed diff;
6. that an inferred change explanation is authoritative history;
7. safety of unknown operation pairs;
8. automatic resolution of every semantic conflict;
9. total, lossless, unambiguous, invertible, or writable mappings;
10. that vector similarity establishes canonical identity;
11. that provenance proves the correctness of a source, mapping, decision, or result;
12. that stored materialization bytes are authoritative meaning;
13. reproducibility from a semantic commit ID alone;

14. equivalence of state reconstruction, query replay, and exact result reproduction;
15. permanent recovery after lawful deletion or expired retention;
16. reversal of external effects through ref rollback;
17. compatibility or equal credibility of alternative branches;
18. that every workspace merge is a categorical pushout;
19. distributed convergence or elimination of coordination tradeoffs;
20. that SQLite evidence transfers to every backend;
21. that Haskell tests are Lean proofs;
22. that finite Lean theorems verify the production runtime;
23. that reconstructed baselines inherit published performance;
24. that CatDB-only cases reveal defects in systems that never claimed those semantics; or
25. any current publishable P9 implementation or performance result.

14 Open questions

14.1 Profile and identity

- Which P3 migration operators can appear in the first overlay profile?
- What is P4's canonical equivalence for mappings, and how does it encode ambiguity?
- Which P6 materialization statuses and reuse rules are authoritative inputs?
- What exact P7 decision and reversal projection enters a commit?
- What completeness and retention structure does P13 expose?
- When may a logical identity cross a namespace or administrative domain?
- How are forked assignments of one logical ID reconciled?

The first five questions are OBSTRUCTED by prerequisite papers. The remaining questions are OPEN.

14.2 Diff and dependency

- Can impact soundness be proved for the selected dependency language?
- How are inferred dependencies shown without making them authoritative?
- What precision keeps impact review and recomputation usable?
- Which query and connector properties affect semantic identity rather than only reproducibility?

14.3 Merge

- What is the complete first operation-pair table?
- Which rules are proved, dynamically checked, or only tested?
- How is a merge base selected when ancestry has several candidates?
- Which parent roles make merge intentionally asymmetric?
- How are policy changes authorized during merge?
- What happens when the validation procedure changes between branches?
- What should the result be when both branches validate independently but violate a global trust or retention rule when combined?

14.4 Storage, history, and evaluation

- Should the first store use normalized relations, content-addressed blobs, or a hybrid?
- What durability assumptions are made about SQLite and the filesystem?
- How long are unreachable complete commits retained after failed publication?
- Which operations have inverses, compensation, or no safe reversal?
- How can lawful deletion coexist with content-addressed history?
- Which backend-version changes preserve semantic equivalence?
- Can original Decibel and ORPHEUSDB artifacts run in a pinned current environment?
- Which cross-object cases occur often enough to support external validity?

15 Conclusion

Database version control already has serious systems work behind it. Schema evolution, mapping adaptation, temporal query, provenance, and dynamic data citation do too. CatDB should not rename those results and call the bundle new.

The remaining question is narrower and testable. Can a database commit bind schemas, mappings, migrations, data observations, identity decisions, provenance requirements, policies, queries, and materialization definitions closely enough to support conservative diff and merge? The design in this paper answers with a partial contract. It separates five identity layers and five diff layers. It requires typed dependency impact, closed merge rules, complete candidate validation, explicit reconciliation, and atomic reference publication. It also keeps reconstruction, replay, and reproduction apart.

None of that behavior exists in the current CatDB repository. CR-069 remains PROPOSED; `SYS-WS-01`, `SYS-WS-02`, and `SYS-WS-03` remain OPEN; the first formal and experimental slices remain OBSTRUCTED. The next useful step is not stronger prose. It is a

frozen finite profile followed by adversarial fixtures, independent Haskell and Rust results, selected Lean obligations, an atomic SQLite reference store, and a fair D7 evaluation.

If those gates pass, CatDB may support a defensible systems claim: alternative semantic interpretations can coexist under explicit context, and some cross-object changes can be diffed and merged safely under a reviewed finite profile. If the gates fail, the claim must narrow. The design is meant to make either outcome visible.

A Fixture and counterexample families

Workspace cases require a new immutable fixture-schema version. Version 1 must remain unchanged. Expected outcomes are closed: `ok`, `conflict`, `reconciliation-required`, `blocked`, `error`, `unsupported`, or `crash-recovery`. Free-form exceptions and skipped cases fail the fixture.

Table 12: Representative fixture families.

Family	Case	Expected result
`WS-COMMIT`	permuted set-like members; changed profile; missing inherited artifact; digest mismatch	stable canonical ID; domain separation; completeness/integrity errors
`WS-ID`	explicit rename; remove/add lookalike; copy; split; merge; content-equal import	declared continuity only; new IDs otherwise; missing-policy block
`WS-OV`	independent inserts; repeated updates; add/retract; rekey/update; identity merge with aggregate	overlay and materialized data/provenance equality or typed conflict
`WS-ISO`	branch-local row, decision, validation, approval, and shared-cache schedules	no cross-workspace observation without explicit import
`WS-DIFF`	retained operations; state-only reconstruction; rename versus remove/add; wildcard query; removed dependency edge	exact reconstruction, evidence separation, and mutation-detected omission
`WS-MRG`	disjoint additions; path/path conflict; delete/use; local commutativity with global uniqueness failure; stale ref; unknown pair	validated merge, typed conflict, CAS failure, or exact unsupported outcome
`WS-REC`	confirmed same versus different; unauthorized decision; supersession; reversal after lossy aggregate	reconciliation, authority rejection, preserved history, partial compensation

Family	Case	Expected result
`WS-PROV`	stricter profile; missing evidence; retention expiry; redaction; equal data with different provenance	blocked, incomplete, unavailable, redacted, or non-exact reproduction
`WS-MAT`	unrelated change; mapping value change; schema rename; frontier skew; stale cache advertised current	reuse, adaptation, invalidation, stale status, or validation failure
`WS-HIST`	retained environment; new plan; current query on old commit; missing API response; changed collation; deleted restricted input	exact, reconstructed, explicit mismatch, unavailable, or nondeterministic
`WS-RB`	ref move; total inverse; lossy operation; abandoned draft; external API write	recorded rollback, new revert, compensation required, or external effect preserved
`WS-CRASH`	kill before/after staging, commit, CAS, and acknowledgement; concurrent publishers	old complete or new complete ref only; one concurrent winner
`WS-ALT`	gross/net mapping; same/different identity; strict/permissive trust; explicit comparison and promotion	isolated context-labelled results and authority-bearing promotion

Acceptance first validates the closed representation, then compares canonical envelopes, Haskell, Rust, and any Lean-linked law. Mutation tests remove dependency, authority, validation, or provenance checks. Crash fixtures run in disposable databases. Every failure is preserved under a run ID.

B Operation-pair policy sketch

Table 13 is illustrative. It is not the frozen first profile.

Table 13: Illustrative operation-pair outcomes.

Left operation	Right operation	Required classification
add object A	add object B	candidate independent if IDs, constraints, and dependencies are disjoint; full validation still required

Left operation	Right operation	Required classification
rename object A	add unrelated object B	candidate independent only when rename continuity is explicit and no wildcard dependency observes labels
remove attribute f	replace mapping image with path using f	schema/mapping conflict
retype attribute f	keyed row update to f	revalidate value-domain conversion; conflict or migration required
add equation	add equation	candidate merge, followed by global consistency and data validation
confirm same x, y	confirm different x, y	reconciliation-required contradiction
confirm same x, y	refresh dependent aggregate	order-dependent unless refresh is recomputed after effective decision
strengthen provenance profile	expire supporting source	retention/provenance conflict
change query definition	reuse old materialization	validated adaptation, rebuild, or invalid
physical plan change	equal semantic definition	execution-strategy choice, not semantic conflict
opaque transform	any semantic structural change	unsupported/manual unless the exact pair gains a reviewed rule

C Claim-to-evidence matrix

Table 14: Claim anchors, evidence, and present status.

Anchor	Program link	Required evidence	Present status
P9-C01	commit identity	cross-language canonical fixtures	OPEN
P9-C02	CR-069, SYS-WS-01	equivalence fixtures and selected proof	OBSTRUCTED
P9-C03	CR-069, SYS-WS-02	concurrent isolation schedules	OPEN
P9-C04	semantic diff	reconstruction and mutation tests	OPEN
P9-C05	dependency impact	formal fragment and seeded faults	OBSTRUCTED
P9-C06	automatic merge	closed exhaustive pair table and property tests	OBSTRUCTED

Anchor	Program link	Required evidence	Present status
P9-C07	CR-045, SYS-WS-03	storage argument and failpoint recovery	OPEN
P9-C08	conflict taxonomy	blinded seeded corpus and confusion matrix	OBSTRUCTED
P9-C09	P7/P9/P13 integration	decision, contradiction, and reversal fixtures	OBSTRUCTED
P9-C10	historical query	retained and missing-input replay runs	OBSTRUCTED
P9-C11	interpretation branches	paired scenarios and provenance audit	OBSTRUCTED
P9-C12	rollback	semantic, storage, and compensation tests	OPEN
P9-C13	systems value	fair baselines and preregistered reviewer study	OPEN

D Reproducibility artifact schema

A confirmatory run should contain:

```
benchmarks/runs/<run_id>/
  manifest.json
  preregistration.md
  source-lock.json
  environment.json
  baseline-status.json
  fixtures/
  raw/
  workspace-results.jsonl
  crash-results.jsonl
  reproduction-results.jsonl
  classifier-confusion.json
  statistics.json
  failures/
  checksums.sha256
```

The manifest records repository revision and dirty state, fixture and profile versions, Rust/Haskell/Lean versions, database and operating-system versions, baseline source or reconstruction status, hardware, seeds, execution order, warmup and repetition policy, exclusions, failed runs, analysis digest, and review sign-off.

Before a future manuscript promotes a claim:

- dependency contracts and the workspace profile must be frozen;
- immutable fixtures and counterexamples must be committed;
- Rust and Haskell results must agree;
- Lean coverage must name only kernel-checked declarations;
- `SYS-WS-01` through `SYS-WS-03` must pass;
- every unknown pair must remain explicit;
- materialization and provenance invalidation must be audited;
- historical status must be tested with missing inputs;
- baseline artifact status must be explicit; and
- all failures, raw observations, statistics, and checksums must reproduce.

References

[1] Philip A. Bernstein and Nathan Goodman. Multiversion concurrency control: Theory and algorithms. *ACM Transactions on Database Systems*, 8(4):465–483, 1983. doi: 10.1145/319996.319998.

[2] Philip A. Bernstein and Sergey Melnik. Model management 2.0: Manipulating richer mappings. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2007. doi: 10.1145/1247480.1247482. URL https://doi.org/10.1145/1247480.1247482.

[3] Anant Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya G. Parameswaran. DataHub: Collaborative data science and dataset version management at scale, 2014. URL https://arxiv.org/abs/1409.0798.

[4] Souvik Bhattacherjee, Amit Chavan, Silu Huang, Amol Deshpande, and Aditya Parameswaran. Principles of dataset versioning: Exploring the recreation/storage tradeoff. *Proceedings of the VLDB Endowment*, 8(12):1346–1357, 2015. doi: 10.14778/2824032.2824035. URL https://www.vldb.org/pvldb/vol8/p1346-bhattacherjee.pdf.

[5] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. Archiving scientific data. *ACM Transactions on Database Systems*, 29(1):2–42, 2004. doi: 10.1145/974750.974752. URL https://doi.org/10.1145/974750.974752.

[6] Amit Chavan, Silu Huang, Amol Deshpande, Aaron Elmore, Samuel Madden, and Aditya Parameswaran. Towards a unified query language for provenance and versioning, 2015. URL https://arxiv.org/abs/1506.04815. Theory and Practice of Provenance; introduces VQuel.

[7] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Automating the database schema evolution process. *The VLDB Journal*, 22(1):73–98, 2013. doi: 10.1007/s00778-012-0302-x. URL https://doi.org/10.1007/s00778-012-0302-x.

[8] Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. Graceful database schema evolution: The PRISM workbench. *Proceedings of the VLDB Endowment*, 1(1):761–772, 2008. doi: 10.14778/1453856.1453939. URL https://www.vldb.org/pvldb/vol1/1453939.pdf.

[9] Carlo A. Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++. *Proceedings of the VLDB Endowment*, 4(2):117–128, 2010. doi: 10.14778/1921071.1921078. URL https://www.vldb.org/pvldb/vol4/p117-curino.pdf.

[10] Cristina de Castro, Fabio Grandi, and Maria Rita Scalas. Schema versioning for multitemporal relational databases. *Information Systems*, 22(5):249–290, 1997. doi: 10.1016/S0306-4379(97)00017-3.

[11] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989. doi: 10.1016/0022-0000(89)90034-2.

[12] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems*, 30(4):994–1055, 2005. doi: 10.1145/1114244.1114249. URL https://doi.org/10.1145/1114244.1114249.

[13] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 31–40, 2007. doi: 10.1145/1265530.1265535. URL https://doi.org/10.1145/1265530.1265535.

[14] Ashish Gupta, Inderpal S. Mumick, Jun Rao, and Kenneth A. Ross. Adapting materialized views after redefinitions: Techniques and a performance study. *Information Systems*, 26(5):323–362, 2001. doi: 10.1016/S0306-4379(01)00024-2.

[15] Kai Herrmann, Hannes Voigt, Andreas Behrend, Jonas Rausch, and Wolfgang Lehner. Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1101–1116, 2017. doi: 10.1145/3035918.3064046. URL https://arxiv.org/abs/1608.05564.

[16] Silu Huang, Liqi Xu, Jialin Liu, Aaron J. Elmore, and Aditya Parameswaran. ORPHEUSDB: Bolt-on versioning for relational databases. *Proceedings of the VLDB Endowment*, 10(10):1130–1141, 2017. doi: 10.14778/3115404.3115417. URL https://www.vldb.org/pvldb/vol10/p1130-huang.pdf.

[17] Krishna Kulkarni and Jan-Eike Michels. Temporal features in SQL:2011. *ACM SIGMOD Record*, 41(3):34–43, 2012. doi: 10.1145/2380776.2380786. URL https://sigmodrecord.org/2012/09/30/temporal-features-in-sql2011/.

[18] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. PROV-O: The PROV ontology. W3C recommendation, World Wide Web Consortium, 2013. URL https://www.w3.org/TR/prov-o/.

[19] Michael Maddox, David Goehring, Aaron J. Elmore, Samuel Madden, Aditya Parameswaran, and Amol Deshpande. Decibel: The relational dataset branching system. *Proceedings of the VLDB Endowment*, 9(9):624–635, 2016. doi: 10.14778/2947618.2947619. URL https://www.vldb.org/pvldb/vol9/p624-maddox.pdf.

[20] Hyun J. Moon, Carlo A. Curino, Alin Deutsch, Chien-Yi Hou, and Carlo Zaniolo. Managing and querying transaction-time databases under schema evolution. *Proceedings of the VLDB Endowment*, 1(1):882–895, 2008. doi: 10.14778/1453856.1453952. URL https://www.vldb.org/pvldb/vol1/1453952.pdf.

[21] Beatriz Pérez, Ángel Luis Rubio Garcia, and María A. Zapata. PROV-IDEA: Supporting interoperable schema and data provenance within database evolution. *ACM Transactions on Software Engineering and Methodology*, 34(3):1–27, 2025. doi: 10.1145/3697008. URL https://doi.org/10.1145/3697008.

[22] Andreas Rauber, Ari Asmi, Dieter van Uytvanck, and Stefan Pröll. Data citation of evolving data: Recommendations of the RDA working group on data citation. Technical report, Research Data Alliance, 2016. URL https://doi.org/10.15497/RDA00016.

[23] Andreas Rauber, Ari Asmi, Dieter van Uytvanck, and Stefan Pröll. Identification of reproducible subsets for data citation, sharing and re-use, 2016. URL https://doi.org/10.5281/zenodo.4048304.

[24] Patrick Schultz, David I. Spivak, and Ryan Wisnesky. Algebraic model management: A survey. In *Recent Trends in Algebraic Development Techniques*, volume 10644 of *Lecture Notes in Computer Science*, pages 56–69. Springer, 2017. doi: 10.1007/978-3-319-72044-9_5. URL https://doi.org/10.1007/978-3-319-72044-9_5.

[25] Roee Shraga and Renée J. Miller. Explaining dataset changes for semantic data versioning with Explain-Da-V. *Proceedings of the VLDB Endowment*, 16(6):1587–1600, 2023. doi: 10.14778/3583140.3583169. URL https://www.vldb.org/pvldb/vol16/p1587-shraga.pdf.

[26] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012. doi: 10.1016/j.ic.2012.05.001. URL https://arxiv.org/abs/1009.1166.

[27] Yannis Velegrakis, Renée J. Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 584–595. Morgan Kaufmann, 2003. doi: 10.1016/B978-012722442-8/50058-6. URL https://research.ibm.com/publications/mapping-adaptation-under-evolving-schemas.

[28] Yannis Velegrakis, Renée J. Miller, and Lucian Popa. Preserving mapping consistency under schema changes. *The VLDB Journal*, 13(3):274–293, 2004. doi: 10.1007/s00778-004-0136-2. URL https://doi.org/10.1007/s00778-004-0136-2.

[29] Cong Yu and Lucian Popa. Semantic adaptation of schema mappings when schemas evolve. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1006–1017, 2005. URL https://www.vldb.org/archives/website/2005/program/paper/fri/p1006-yu.pdf.