Bidirectional Schema Mappings and Safe Update Propagation in CatDB

Typed write profiles, explicit refusal, and the boundary of reconciliation

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

23 July 2026

Provisional evidence status: `OBSTRUCTED`. This paper specifies an engineering hypothesis and falsification program. CatDB does not currently implement the write profile, putback evaluator, authority gate, database transaction path, P8 fixture suite, P8 Haskell oracle, P8 Lean theory, or P8 benchmark described here. The current bounded Δ_F implementation is read-side migration evidence only.

Abstract

A canonical view is easy to read and dangerously easy to mistake for a safe place to write. The read mapping may discard fields, combine records, hide origin, cross ownership boundaries, or depend on source snapshots that have already changed. Classical view-update theory, lenses, putback-first languages, and native database views provide precise tools for parts of this problem, but none turns an arbitrary integration mapping into an authorized, transactionally sound write path.

This paper proposes a narrower contract for CatDB. A versioned mapping may carry a typed write profile that names its update language, putback semantics, authority and identity requirements, constraint obligations, write footprint, source-version checks, connector capabilities, and transaction policy. The profile classifies the mapping as exactly one of `fully_writable`, `partially_writable`, `read_only`, or `reconciliation_dependent`. The classification is relative to a declared update language. Each attempted update still returns an operation-level decision: accepted, rejected, conflict, requires-reconciliation, or unsupported.

The main result is a design and falsification boundary, not a completed system. We give the proposed formal model, relational cases, API semantics, multi-source refusal rules, implementation roadmap, claim table, and counterexample plan. Universal safe write-back is rejected. The current status is `OBSTRUCTED` because the relevant CatDB profile, implementation, proofs, database execution, and experiments do not yet exist.

Contents

1 Introduction

Suppose a canonical customer record presents a name from a CRM, a credit limit from an ERP system, an email-consent flag from a consent service, and a segment computed by analytics. Reading the record is an integration problem. Writing it is four different problems hidden behind one form. The name and credit limit have different owners. Changing consent may require a domain-specific API call rather than a column update. The segment may have no meaningful inverse at all. A batch that edits all four fields may also request an all-or-nothing result that the sources cannot provide.

The tempting shortcut is to reverse the read mapping. That shortcut fails even before distributed execution enters the picture. A projection has forgotten fields. A join offers several possible source effects. An untagged union has lost branch origin. A full name can be split in more than one reasonable way. An average specifies a desired output without choosing the underlying rows. An exact mathematical inverse, when one exists, still says nothing about who may write, whether the target record is the intended identity, or whether the source supports a conditional update.

Database and programming-language research has treated these questions for decades. Classical work formalized update translators, complements, and correctness conditions [2, 6, 5, 20, 12]. Lenses made get and put behavior compositional and exposed round-trip laws [10, 3]. Symmetric and delta lenses made private state and explicit change more visible [15, 7, 17]. Putback-first systems let programmers state backward behavior instead of guessing it from a query [21, 22, 26, 25]. Schema-mapping inversion has its own existence and expressiveness boundaries [8, 9]. These are foundations and baselines for P8, not gaps that CatDB can claim to have discovered.

The proposal in this paper is operationally conservative. A read mapping and a write profile are different artifacts. The profile says which canonical updates are admitted, which source effects implement them, which evidence justifies the choice, and which checks must hold before execution. When policy or evidence cannot choose a source effect, the result is not a guessed write. It is a structured reconciliation request. When no write semantics exists, the mapping is read-only. Both are successful safety outcomes.

This position leads to two levels of answer. At the mapping level, the profile has exactly one of four labels: `fully_writable`, `partially_writable`, `read_only`, or `reconciliation_dependent`. At the operation level, a request may be accepted, rejected, conflicted, escalated to reconciliation, or marked unsupported. A normally writable operation can still encounter a stale source version. A partially writable mapping can accept one field, reject another, and escalate a third.

The contribution claimed here is limited to a `ENGINEERING HYPOTHESIS`: a versioned systems contract connecting established lens and view-update semantics to CatDB mapping identity, authority, identity evidence, constraints, effects, source capabilities, transactions, conflicts, and provenance. The four-way classification and normalized refusal surface are proposed. They have not been implemented or experimentally validated. The paper therefore serves as an exact specification for what would have to be built and falsified.

Questions. The paper asks:

1. What information must accompany a read mapping before a canonical update can be considered?
2. How should writeability be classified without treating "invertible" as a Boolean shortcut?
3. Which laws are semantic, and which obligations belong to policy, identity, transactions, and source capabilities?
4. When must a multi-source request stop at reconciliation?
5. What evidence would justify promoting the proposal beyond `OBSTRUCTED`?

Answer in brief. A mapping is writable only relative to an immutable write profile and a declared update language. A putback decision must be typed, partial, and able to refuse. Round-trip laws are required for the fragment that claims them, but they are not sufficient for execution. Authority, exact identity, constraint preservation, write footprint, source-version checks, connector capabilities, and a matching transaction contract are separate obligations. Ambiguity in business meaning is preserved as a versioned reconciliation decision.

Paper organization. Section 2 records the evidence boundary. Section 3 defines the model and profile. Section 4 defines the four classes. Section 5 separates lens laws from operational obligations. Section 6 develops relational cases. Sections 7 to 9 cover authority, identity, conflicts, and multi-source ambiguity. Section 10 states what the categorical layer does and does not provide. Section 11 gives the proposed API and execution semantics. Sections 12 and 13 record implementation status and the falsification plan. Related work, limitations, open questions, and exact nonclaims follow.

2 Evidence and status contract

2.1 Evidence vocabulary

This manuscript uses two independent dimensions. An evidence label says what kind of support a statement has. An operational status says where the CatDB research workflow currently stands. They are not interchangeable.

Table 1: Evidence and operational labels used by P8.

Label	Meaning
`SUPPORTED BY PRIOR LITERATURE`	A primary paper, standard, or official implementation document establishes a result within its stated assumptions. It does not establish a CatDB implementation or novelty claim.
`ENGINEERING HYPOTHESIS`	A precise proposed CatDB behavior has named assumptions and a falsification test, but lacks sufficient implementation and experimental evidence.

Label	Meaning
`OPEN CONJECTURE`	A precise mathematical or systems statement remains unproved.
`COMPUTATIONAL`	A named implementation computed a named result under a recorded command. The current P8 write path has no such result.
`STRUCTURAL-ONLY`	An artifact decoded, normalized, or round-tripped without executing the claimed write semantics.
`OPEN`	Definitions, fixtures, code, experiments, or review remain incomplete.
`OBSTRUCTED`	A prerequisite semantic or operational boundary is missing, so the claim cannot yet be implemented or tested honestly.

No CatDB P8 claim currently has proof, checker, or experimental support. The following closed-vocabulary labels do not apply:

`PROVED`, `MACHINE-CHECKED`, or
`EXPERIMENTALLY SUPPORTED`.

Established view-update and lens results are `SUPPORTED BY PRIOR LITERATURE`. The proposed P8 contract is an `ENGINEERING HYPOTHESIS` with overall status `OBSTRUCTED`.

2.2 Dependency boundary

P8 depends on immutable schema and mapping identity, migration and information loss, exact source-to-domain mapping semantics, and explicit identity and authority decisions. In the program dependency graph these are supplied by P3, P4, and P7. Distributed ordering and consistency belong to P10 and P11. Provenance-complete write history belongs to P13. This provisional paper may state the interface it needs from those papers. It may not pretend those interfaces are frozen or implemented.

The dependency has practical consequences:

- P8 cannot infer a putback from a read mapping whose instance semantics are not defined.
- P8 cannot turn a probable identity match into a physical target.
- P8 cannot use a provenance record as authorization.
- P8 cannot call a cross-source batch atomic without a mechanism that supplies the stated scope.

2.3 Current repository evidence

The shared repository has bounded finite schema, path, mapping, and instance artifacts; Rust and Haskell finite-instance validation; a bounded Δ_F candidate; selected Lean path and

mapping laws; and stable diagnostic conventions. Those artifacts are evidence for their own accepted slices. They do not implement P8.

The P8-specific absences are material: no closed write-profile schema, no canonical-update IR, no putback evaluator, no classifier, no authority gate, no identity-strength enforcement, no reconciliation request type, no version preconditions, no effect analysis, no SQLite or PostgreSQL write compiler, no P8 fixtures, no independent P8 Haskell semantics, no P8 Lean definitions, no P8 experiment, and no AGY paper review. General Σ_F and Π_F also remain outside the current executable profile.

Binding status statement. No current CatDB mapping is classified as fully writable or partially writable. The four-way classifier is proposed but absent. Universal safe write-back is rejected.

3 Mathematical framework

3.1 States, views, updates, and contexts

Definition 3.1 (Source and canonical state)**.** Let S be a source-state type and V a canonical view-state type. Let $C_S : S \to \{\mathsf{true}, \mathsf{false}\}$ and $C_V : V \to \{\mathsf{true}, \mathsf{false}\}$ be their declared validity predicates. A state is valid exactly when its corresponding predicate holds.

Definition 3.2 (Read interpretation)**.** A read interpretation is a declared function

$$g : S \longrightarrow V.$$

Its revision is immutable and is named by the write profile. This definition does not imply injectivity, surjectivity, or an inverse.

Definition 3.3 (Canonical update language)**.** For one immutable profile revision P, let $\mathcal{U}_V$ be a versioned, typed language of canonical updates. The proposed first language contains insertion, deletion, scalar replacement, relation and unrelation, and a batch constructor with an explicit atomicity requirement. Arbitrary SQL, opaque user code, schema changes, aggregates, vector-neighbor mutations, and unfrozen bulk predicates are outside the first language.

The candidate sum type is:

```
CanonicalUpdate =
    Insert { object, proposed_id, fields }
  | Delete { object, canonical_id }
  | Replace { object, canonical_id, path, before, after }
  | Relate { relation, from, to }
  | Unrelate{ relation, from, to }
  | Batch { atomicity, updates }
```

Definition 3.4 (Write context)**.** A write context K contains the actor, workspace, canonical snapshot, source snapshots, record or predicate version tokens, identity evidence, approvals, consistency requirement, and idempotency key used to decide and execute one request.

```
WriteContext {
  actor
  workspace
  canonical_snapshot
  source_snapshots
  expected_versions
  identity_evidence
  approvals
  consistency_requirement
  request_id
}
```

A snapshot reference is source-specific. A file digest, PostgreSQL snapshot, HTTP ETag, and opaque API version token do not provide the same guarantee. The profile must say which one protects which read set.

3.2 Partial putback

Definition 3.5 (Profile-relative putback)**.** For an immutable write profile P, the proposed semantic decision function is

$$\mathrm{put}_P : S \times \mathcal{U}_V \times K \longrightarrow R,$$

where R is a normalized result with one of five constructors:

$$\text{Accepted}, \quad \text{Rejected}, \quad \text{Conflict}, \quad \text{RequiresReconciliation}, \quad \text{Unsupported}\,.$$

The function is partial in the semantic sense that not every canonical update has an executable source effect. Partiality is represented by a typed result, not by an undocumented exception.

Definition 3.6 (Pure state-based lens kernel)**.** When a profile claims state-based lens laws, its pure kernel has the partial shape

$$p_P : S \times V \rightharpoonup S.$$

For an admitted operational update $u \in \mathcal{U}_V$, let $v' = \mathrm{apply}_V(g(s), u)$. If the operational translator accepts a source delta δs, let $s' = \mathrm{apply}_S(s, \delta s)$. The profile must state and verify the correspondence between $p_P(s, v') = s'$ and the accepted normalized decision. The state kernel does not contain authority, connector, transaction, or audit effects unless the profile explicitly extends its equality and effect model.

The accepted case carries a semantic delta, a physical-plan candidate, discharged obligations, pending runtime checks, affected footprint, and expected post-view digest. A rejected case names failed obligations and evidence that no physical effect occurred. A conflict reports observed and expected versions. A reconciliation result retains alternatives and required authority. Unsupported identifies the boundary of the profile.

```
WriteDecision =
    Accepted {
      semantic_delta,
      physical_plan,
      discharged_obligations,
      pending_runtime_checks,
      affected_footprint,
      expected_post_view_digest
    }
  | Rejected {
      code,
      failed_obligations,
      no_effect_proof_or_receipt
    }
  | Conflict {
      observed_versions,
      expected_versions,
      conflict_class,
      retryability
    }
  | RequiresReconciliation {
      alternatives,
      evidence,
      required_authority,
      reversible_decision_template
    }
  | Unsupported {
      feature,
      profile_boundary
    }
```

3.3 Inverse classes are not write classes

Four mathematical relationships recur in the literature and in proposed CatDB mappings.

Exact inverse. A mapping h satisfies the relevant two-sided recovery equations on a declared class of instances.

Quasi-inverse. Recovery is relative to an equivalence induced by schema mapping or data exchange. Existence and expressiveness are restricted [8, 9].

Partial inverse. A function is defined only on a named subdomain, for example

$$h(g(s)) = s \quad (s \in D_S) \qquad \text{or} \qquad g(h(v)) = v \quad (v \in D_V).$$

Lens-like relationship. A read function and putback satisfy selected round-trip laws on explicit domains, often using a source or separate record as complement [10, 3].

None of these relationships specifies actor authority, path ownership, connector capability,

isolation, retry, or the intended choice among several semantically meaningful inverse images. The write profile therefore records an *inverse class* and a *write class* as separate fields.

3.4 Typed write profile

The proposed profile is a closed, versioned representation:

```
WriteProfile {
  profile_id
  profile_version
  mapping_id
  mapping_revision
  source_schema_revision
  canonical_schema_revision
  read_semantics_revision

  update_language
  put_semantics
  inverse_class:
      Exact | Quasi | Partial | LensLike | None
  write_class:
      FullyWritable
    | PartiallyWritable
    | ReadOnly
    | ReconciliationDependent

  path_policies
  authority_policy
  identity_policy
  source_capabilities
  constraint_obligations
  law_obligations
  effect_policy
  conflict_policy
  transaction_policy
  provenance_policy
}
```

Changing a default, owner, delete strategy, constraint, target, law, or conflict rule creates a new profile revision. A mutable profile would make an old approval or cached plan impossible to interpret reliably.

3.5 Path policy

Writeability may differ by canonical path. Each path policy declares allowed operations, candidate targets, deterministic target selection, conversion, missing-value behavior, delete behavior, footprint, constraints, authority, and identity strength.

```
PathWritePolicy {
  canonical_path
  allowed_operations
  candidate_targets
  target_selection
  conversion
  missing_source_value_policy
  delete_policy
  write_footprint
  required_constraints
  required_authority
  required_identity_strength
}
```

Target selection must be a result of versioned policy and evidence. Connector order and map iteration order are not semantic priorities.

4 Four-way writeability classification

4.1 Definition

Let $\mathcal{U}_V$ be the profile's declared update language. Let $D_P \subseteq \mathcal{U}_V$ contain updates that can proceed without a new semantic decision, and let $R_P \subseteq \mathcal{U}_V$ contain updates that the profile deliberately sends to reconciliation. Require

$$D_P \cap R_P = \varnothing.$$

The mapping-level class is assigned in the following order.

Definition 4.1 (Fully writable)**.** P is `fully_writable` exactly when

$$D_P = \mathcal{U}_V.$$

Every well-typed update in the declared language has a decision procedure and every update satisfying the published preconditions is accepted without a new reconciliation decision.

Definition 4.2 (Reconciliation-dependent)**.** P is `reconciliation_dependent` exactly when

$$D_P \neq \mathcal{U}_V \quad \text{and} \quad R_P \neq \varnothing.$$

At least one declared intended update requires new identity, authority, conflict, atomicity, or business-semantic input.

Definition 4.3 (Partially writable)**.** P is `partially_writable` exactly when

$$\varnothing \subset D_P \subset \mathcal{U}_V \quad \text{and} \quad R_P = \varnothing.$$

The strict admitted subset must be decidable and published as

$$\text{admissible}_P(u, K).$$

Definition 4.4 (Read-only)**.** P is `read_only` exactly when

$$D_P = R_P = \varnothing.$$

No canonical update is admitted.

Updates outside $D_P \cup R_P$ are rejected or unsupported. A runtime Conflict is orthogonal to the static mapping class.

Proposition 4.5 (Four-way partition, status `OPEN`)**.** *Assume a well-formed profile has a declared language $\mathcal{U}_V$ and decidable, disjoint subsets $D_P, R_P \subseteq \mathcal{U}_V$. The ordered definitions above select exactly one mapping-level class.*

Manuscript case analysis. If $D_P = \mathcal{U}_V$, disjointness implies $R_P = \varnothing$, so the first case applies. Otherwise $D_P \neq \mathcal{U}_V$. If $R_P \neq \varnothing$, the second case applies. If $R_P = \varnothing$ and $D_P \neq \varnothing$, then D_P is a nonempty strict subset and the third case applies. The remaining case has $D_P = R_P = \varnothing$ and is read-only. The conditions used by later cases contradict those of earlier cases. □

The case analysis does not promote P8-C01. The profile is not frozen, no classifier exists, and no finite Lean theorem represents these definitions. The claim remains `OPEN`.

4.2 What fully writable means

`fully_writable` is always relative to $\mathcal{U}_V$. It does not mean every imaginable mutation of the canonical schema is writable. The class requires total decision coverage on the declared language, deterministic translation up to declared plan equivalence, satisfaction of all named laws and constraints, sufficient authority and identity, present source capabilities, and no in-domain reconciliation requirement.

A profile that admits only scalar replacements can be fully writable for that language while rejecting insertion because insertion is not in the language. The update-language revision is therefore part of the claim.

4.3 What partially writable means

`partially_writable` is not permission to discover the boundary by attempting writes. The admission predicate must be inspectable before connector mutation. Examples include base columns writable while a computed column is read-only, selection-preserving replacements, projection replacements with a retained complement, and one owned side of a key-preserving join.

4.4 Read-only is intentional

`read_only` can be permanent. Aggregates without an allocation policy, historical snapshots, vector-similarity results, externally governed sources, and connectors without mutation capability are legitimate read-only cases. The classifier must refuse before a physical mutation is attempted.

4.5 Reconciliation is a refusal to guess

`reconciliation_dependent` does not mean "retry later." It means the current policy and evidence cannot choose a source effect. The result retains the alternatives, evidence, required decision role, expiration, and reversal history. An approval creates a new decision artifact. It does not rewrite the old profile or silently make its prior ambiguity disappear.

5 Law profile and operational obligations

5.1 GetPut

For a valid source state s, the classic stability law is

$$p_P(s, g(s)) = s.$$

For the operation translator, the corresponding no-op canonical update must produce an accepted empty semantic delta or an explicitly equivalent no-effect result. Operational timestamps, audit records, generated transaction identifiers, and source version tokens must be projected out before exact semantic equality is claimed. If those effects are inside the claim, the profile needs an effect-aware law.

5.2 PutGet

For an admitted requested view v', if putback accepts and produces s',

$$p_P(s, v') = s' \quad \Longrightarrow \quad g(s') = v'.$$

This is the exact-view condition. Bag semantics, nulls, ordering, generated identities, and provenance may require a named equality other than byte equality. A policy that permits amendments must state a weaker relation rather than call it equality.

5.3 PutPut

For deterministic state semantics, history ignorance is often stated as

$$p_P(p_P(s, v_1), v_2) = p_P(s, v_2)$$

when both sides are defined under compatible context. P8 does not impose this law universally. Approval histories, monotone audit state, external effects, and version tokens can make literal operational equality inappropriate. A profile must state whether it claims semantic PutPut, effect-aware PutPut, a delta-composition law, or no PutPut law.

Putback-first work provides a direct precedent for making backward behavior explicit and checking well-behavedness [21, 22, 14].

5.4 Constraint preservation

For an accepted update:

$$\begin{aligned} C_S(s) \wedge C_V(v') \wedge \mathrm{put}_P(s, u, K) = \mathrm{Accepted}(\delta s, \ldots) \\ \wedge \ s' = \mathrm{apply}_S(s, \delta s) \wedge v' = \mathrm{apply}_V(g(s), u) \\ \Longrightarrow C_S(s') \wedge C_V(g(s')). \end{aligned}$$

The constraint set may include scalar domains, keys, foreign keys, cardinality, path equations, source-native checks, canonical invariants, tenancy, identity consistency, temporal validity, and materialization dependencies.

5.5 Footprint preservation

If W is the declared source footprint:

$$\mathrm{restrict}_{\neg W}(s') = \mathrm{restrict}_{\neg W}(s).$$

This is the no-unadvertised-side-effect condition. A weaker profile may allow additional effects only when it predicts and exposes them before execution.

5.6 Authority preservation

Every physical mutation requires an authority derivation:

$$\mathrm{authorized}(K.\mathit{actor}, \mathit{target}, \mathit{operation}, K) = \mathit{allow}.$$

A database privilege check can be necessary while remaining insufficient. The credential holder and the semantic owner are not necessarily the same.

5.7 Version preservation

For every protected record or predicate range r:

$$\mathrm{observedVersion}(r) = \mathrm{expectedVersion}(r),$$

or the result is Conflict. The backend contract must identify the protected read set. A row token is not enough for every predicate write, join, or cross-row constraint.

5.8 Idempotency and replay

If a profile claims idempotent replay under key k:

$$\mathrm{exec}(k, \mathrm{exec}(k, s)) \simeq \mathrm{exec}(k, s).$$

This requires persisted deduplication or a source guarantee. Deterministic plan generation does not imply idempotent execution, and timeout does not prove failure.

5.9 Why laws are not a safety bundle

GetPut and PutGet describe a semantic relation between source and view. They do not prove authorization, exact identity, transaction isolation, source capabilities, retry safety, or acceptable side effects. A lawful pure lens can still choose the wrong customer record, write a field owned by another system, race with a concurrent edit, or call an API that committed before a timeout. Each obligation needs separate evidence.

6 Relational view-update cases

The first useful P8 fragment should be finite and relational. It should cover identity and renaming, total bijective scalar conversion, selection with a check condition, projection with a complement or constructor policy, and one key-preserving join fragment. This section explains why each operator needs a different write policy. Classical operator-specific algorithms and relational lenses are the direct baselines [20, 3].

6.1 Identity and renaming

Identity and pure renaming are the easiest candidates, but they are not automatic. Consider a canonical field `Customer.displayName` mapped to `crm.customer_name`. A replacement can translate to one source-field update only if the profile also knows the source-local key, authorizes the actor, validates the new scalar, protects the source version, and executes in a transaction with an auditable result.

For a bounded replacement-only language, this can be a fully writable candidate. It becomes read-only when the connector offers no mutation, and it returns a runtime conflict when the protected source version changed.

6.2 Selection

Let

$$V = \sigma_P(S).$$

Suppose a view contains active accounts. Changing the status field from `active` to `suspended` makes the row disappear. That may be the requested business action, or it may violate the user's expectation that an edited row remains visible. The query alone cannot decide.

One profile can use check-option behavior and reject any update that falsifies P. Another can allow disappearance and use a weaker effect relation. Native PostgreSQL updatable views and `CHECK OPTION` provide a concrete backend baseline, with documented restrictions and trigger caveats [24]. Fixtures must cover replacements that preserve and violate P, satisfying and violating inserts, deletion, and a source change between read and execution.

6.3 Projection

Let

$$V = \pi_A(S).$$

An existing source row can act as a complement. If the canonical view exposes name and email but hides a required tax code, replacing the email can preserve the tax code from the exact old source row. Insertion is different. It needs a default, generator, or constructor for every omitted required field.

Projection also loses identity when A is not a key. Two source rows can collapse to one view row. A write profile must state the stable source key, complement location and revision, insert constructor, duplicate behavior, and view equality. If the complement is missing or stale, the request cannot use it as if it were durable evidence.

6.4 Key-preserving join

Let

$$V = R \bowtie_K T.$$

A joined row contains fields from two source relations. If the view edits a field owned by R and functionally determined by K, one bounded profile can update R only. Updating a shared key, synthesizing a missing referenced row, or deleting a joined entity is outside that fragment.

Many-to-many joins expose a sharper failure. One source mutation can change several view rows. If the profile promises an exact one-row effect, the predicted fan-out must block execution. A link-table profile may safely delete an association while refusing deletion of either endpoint.

6.5 Union

Let

$$V = R \cup T.$$

An origin tag can support a branch-specific policy. Without it, an inserted or deleted tuple has no unique destination. If the tuple is present in both branches, deleting it from one branch may leave the view unchanged.

The safe untagged-union response is a reconciliation request containing the R and T alternatives and their predicted effects. Choosing the first connector or the cheaper write would be physically deterministic but semantically arbitrary. Collaborative update-exchange work already treats trust, mappings, provenance, and side effects as first-class concerns [13, 19].

6.6 Computed fields

A forward function is writable only when the profile provides a domain-bounded inverse or putback. Exact Celsius-to-Fahrenheit conversion is a reasonable candidate over a declared

numeric domain. Full-name splitting is not: "Ada Lovelace" admits several conventions once titles, particles, and multi-part family names enter the source.

Normalization can also destroy lexical information. A phone formatter may be invertible only when a country code and extension are retained as a complement. Hashes, counts, ranks, averages, and embeddings have no generic putback. A deterministic forward computation does not change that.

6.7 Aggregation

Changing an average from 7 to 8 does not identify which rows to edit. Allocation or optimization policies can be supplied by an application, but they are new business semantics. The first P8 profile returns `read_only` or RequiresReconciliation for aggregates. It never fabricates row updates.

6.8 Deletion

Deletion raises its own provenance and side-effect choices. A deletion from a join or union may have several source explanations, and finding a minimal or preferred source deletion can be nontrivial [4]. P8 must name whether the operation deletes a source entity, removes a relationship, marks a tombstone, or creates a proposal. A generic "delete what made this row" rule is not acceptable.

Table 2: Candidate status by relational construct. No row is a current CatDB result.

Construct	Candidate class	Required boundary
Identity or rename	fully writable candidate	Exact identity, type equality, authority, version check, and transaction.
Bijective scalar conversion	fully writable candidate	Total inverse on the declared domain and canonical lexical rules.
Selection	partially writable candidate	Predicate-preserving update or explicit check-option/effect policy.
Projection	partially writable candidate	Stable key, exact complement, and an insert constructor if insertion is admitted.
Key-preserving join	partially writable candidate	Functional dependency, side ownership, delete policy, and footprint.
Tagged union	partially writable candidate	Stable origin and branch-specific authority.
Untagged union	reconciliation-dependent	Versioned branch choice; silent tie-breaking is forbidden.
Computed field	partial or reconciliation candidate	Unique domain-bounded inverse or retained complement.

Construct	Candidate class	Required boundary
Aggregate	read-only initially	No generic inverse; an allocation policy would be new semantics.
Deduplication or quotient	reconciliation-dependent	Known identity or approved merge with reversible decision history.
Vector-similarity result	read-only	Similarity may propose candidates but cannot establish target identity.
Historical snapshot	read-only	Mutation requires a separate fork or branch operation.
Multi-source batch	reconciliation-dependent initially	Owner selection plus a matching atomicity or compensation contract.

7 Authority, identity, and ownership

7.1 Authority as versioned data

Authority is a relation over source, object type, canonical path, operation, tenant or workspace, actor or role, valid time, system time, and delegation evidence. Its possible decisions are `allow`, `deny`, `require_approval`, and `allow_with_obligations`. No result defaults to allow.

An authoritative source is not simply the first mapping, newest timestamp, highest similarity score, lowest-latency connector, last writer, or source whose credential permits mutation. Authority is business ownership or delegation represented as policy.

7.2 Path-level ownership example

Return to the customer record:

```
Customer.name -> CRM
Customer.creditLimit -> ERP
Customer.emailOptIn -> Consent service
Customer.segment -> Derived analytics
```

The useful decision is path-specific. A name update may target the CRM. A credit-limit change may require ERP approval. Consent may require a dedicated operation that records legal context. Segment remains read-only because it is derived. A batch spanning name and credit limit may be individually well-defined while still lacking the cross-source atomicity the caller asks for.

7.3 Approval binding

An approval references the exact semantic delta, affected records, profile and policy revisions, actor and approver, evidence shown, expiration, and constraints checked. If any of those

inputs drift before execution, the approval must be revalidated. A detached Boolean approval would authorize a different request after the evidence changed.

7.4 Identity strength

Each path policy names its required identity strength:

```
source_local_exact
canonical_known_equivalence
approved_merge
probable_match_not_writable
possible_match_not_writable
contradicted_not_writable
```

A probable or possible match can produce a proposal. It cannot produce a silent write target. Vector similarity remains candidate-generation evidence, not identity or authority.

7.5 Merge, supersession, and generated identity

If two source records have been merged canonically, the policy must state whether both remain authoritative, one supersedes the other, one becomes historical, updates fan out, or every write requires reconciliation. A reversible identity decision does not make an already committed external write reversible.

Insertion must also say where identity is minted: a source sequence, a CatDB UUID, a remote API, or an approved identity service. A canonical to source correspondence is not final until the source confirms creation. Generated-key return, retry, and partial failure belong to the plan.

8 Conflicts, constraints, and transactions

8.1 Conflict is not ambiguity

Ambiguity exists before execution because the desired source effect is underspecified. Conflict exists when a formerly valid plan no longer matches observed state or when concurrent intentions are incompatible. Keeping them separate improves diagnostics and prevents blind retry of a semantic question.

The proposed conflict classes are:

```
ConflictClass =
    SourceVersionChanged
  | CanonicalVersionChanged
  | MappingRevisionChanged
  | AuthorityPolicyChanged
  | IdentityDecisionChanged
  | ConstraintChanged
```

```
  | CapabilityChanged
  | ConcurrentSamePath
  | ConcurrentRelatedPath
  | PartialCommitObserved
  | RetryOutcomeUnknown
```

8.2 Compare-and-set boundary

For one row, an expected-version predicate can supply a useful compare-and-set contract. Predicate updates, joins, and cross-row constraints may need a protected predicate range or transaction snapshot. The profile must identify the read set whose stability justifies the decision.

Allowed conflict policies include reject and reread, a proven commutative merge, reconciliation, revalidated retry of an idempotent operation, or a versioned compensating action after recorded partial commit. `last_write_wins` is never the default. If a bounded field uses it, the policy must be explicit and its lost-update behavior measured.

8.3 Constraint layers

An accepted update may rely on static proof, a preflight query, a transaction-time constraint, or asynchronous validation. These are different guarantees. The decision records which layer discharged each obligation.

Native database constraints are useful execution guards but do not replace semantic validation. A PostgreSQL foreign key cannot validate an external API relation. A trigger can have hidden effects. Deferred constraints move the failure point. Collation and null semantics can change a selection predicate. Every relied-on native check needs a lowering report.

8.4 Exact effects and side-effect reports

For requested view delta δv and source delta δs, the profile may require

$$g(s \oplus \delta s) = g(s) \oplus \delta v.$$

The equality must name set or bag semantics, nulls, order, and error values. Before execution, the decision reports requested paths, predicted canonical paths, source fields, cascades, generated values, rows becoming invisible, affected materializations, and unknown effects. An unknown effect blocks any profile that promises exact footprint.

8.5 Local transaction contract

For one SQLite or PostgreSQL backend, the first profile should require:

1. one transaction at declared isolation;
2. precondition checks and mutations in that transaction;
3. typed mapping of constraint failure;

4. rollback on any failed statement;
5. generated-key capture and deterministic statement order; and
6. a postcondition query or returned-row check when required.

SQLite is the program's first local execution target; PostgreSQL is second. Neither compiler exists for P8.

8.6 APIs, files, and retry

An HTTP API may offer an ETag but no multi-request transaction. A local file may support atomic rename without a stable remote lock. The capability profile must distinguish conditional update, idempotency key, batch atomicity, read-your-write, returned version, rollback, compensation, rate limit, and unknown-outcome handling.

A retry is permitted only when the operation is proven idempotent under the source contract, the source persists an accepted idempotency key, or the previous attempt is known not to have committed. A timeout is an unknown outcome unless the source proves otherwise.

9 Multi-source ambiguity and reconciliation

9.1 Ambiguity classes

Multi-source ambiguity is not one error code. The profile distinguishes:

Table 3: Ambiguity classes.

Class	Meaning
Target	More than one source can receive the update.
Decomposition	One canonical value has several source decompositions.
Identity	The intended source record is uncertain.
Authority	Ownership policies disagree or are incomplete.
Constraint	Several constraint-preserving translations remain.
Temporal	The read combined incompatible source snapshots.
Conflict	Concurrent updates express incompatible intent.
Atomicity	No mechanism matches the requested all-or-nothing result.
Side effect	Alternatives differ in additional canonical effects.

Each class may require a different decision role. A data steward can resolve identity while a service owner resolves mutation authority and an application owner changes an atomicity requirement.

9.2 Reconciliation request

```
ReconciliationRequest {
  request_id
  canonical_update
  profile_revision
  alternatives: [
    {
      semantic_delta
      physical_targets
      predicted_view_effect
      predicted_side_effects
      constraints
      authority_evidence
      identity_evidence
      atomicity
    }
  ]
  contradictions
  required_decision_role
  expiration
}
```

An approved alternative creates a new decision artifact. If policy permits, it may motivate a more specific future profile rule. It does not mutate the old profile. Reversal changes future decision state without rewriting history.

9.3 No arbitrary tie-breaking

Lexicographic source identifiers, connector order, map iteration order, lowest estimated cost, latest timestamp, and first successful write can make a plan deterministic. They become semantic resolution only when a versioned policy explicitly says so. Otherwise the planner must preserve the alternatives.

9.4 Cross-source execution

A logical write is atomic only within the scope established by its mechanism. Possible mechanisms include a distributed transaction, source-specific workflow, saga with compensation, outbox with asynchronous propagation, tentative workspace proposal, or refusal. Each exposes different visibility and failure behavior.

Dejima and update-exchange systems show that shared views, peer autonomy, affected-peer discovery, trust, side effects, and transactional propagation already have substantial systems precedent [13, 19, 1]. A CatDB contribution must compare exact semantics rather than claim that multi-source update propagation is new.

The first P8 profile rejects cross-source atomic execution. If two independent local updates cannot meet the caller's consistency requirement, the mapping is `reconciliation_dependent` for that batch. A saga can be offered only as a different contract. Compensation is not

rollback.

10 Categorical relationship and composition

10.1 What typed mappings contribute

A typed schema mapping can provide source and target typing, path substitution, compositional structure, equation-preservation obligations, and immutable mapping identity. It gives the write profile an inspectable object to reference. Category-theoretic lens work also provides relevant precedent for universal translations under explicit assumptions [18].

10.2 What a functor does not provide

A functor does not automatically provide an inverse functor, instance-level putback, complement, choice among inverse images, source authority, authorization, transaction atomicity, or conflict resolution. Formal coherence can expose incompatibility. It cannot infer intended business meaning.

10.3 Delta, Sigma, and Pi

The current repository computes a bounded finite Δ_F candidate. Δ_F is reindexing or pullback migration on instances. It is not a generic inverse of a read view and not a write translator.

General Σ_F and Π_F are unsupported in the current executable profile. Even when adjoints exist, their units and counits do not choose a user update policy. They may help state information-loss or recovery properties, but P8 still needs operation-level laws, authority, identity, effects, and execution evidence.

10.4 Natural transformations, pullbacks, and pushouts

A natural transformation can represent a coherent data change between functors. That mathematical change is distinct from a proposed semantic delta, a validated write decision, and a committed physical operation.

Pullbacks and pushouts may model agreement or combination in a stated category. They do not infer ownership, canonical identity, or a commit protocol. Their P8 use remains `OPEN` until the category, existence conditions, and operational interpretation are supplied.

10.5 Composition obligations

For profiles

$$P_1 : S \leftrightarrow V, \qquad P_2 : V \leftrightarrow W,$$

write composition is admitted only when:

1. intermediate state domains and equality notions agree;

2. the output update domain of P_2 is accepted by P_1;
3. constraint and law obligations align;
4. footprints and authority requirements compose;
5. effects do not duplicate or reorder unsafely;
6. source-version preconditions can be enforced together; and
7. the composite satisfies every law it declares.

Read-side mapping composition does not discharge these obligations. An individually lawful pair may compose to a narrower domain or to refusal.

Proposition 10.1 (Selected profile composition, status `OPEN CONJECTURE`)**.** *If two pure finite profiles satisfy an explicit compatibility predicate over domains, equality, constraints, laws, footprints, and effects, then selected GetPut and conditional PutGet laws may lift to the composite.*

No proof or implementation currently establishes this statement for P8. Arbitrary writeability composition is explicitly not claimed.

11 Proposed API and execution semantics

11.1 Semantic operations instead of raw mutation

The proposed agent and service surface exposes semantic stages rather than an unrestricted row-write endpoint:

```
describeWriteProfile(mapping_revision) -> WriteProfileSummary
proposeCanonicalUpdate(update, context) -> Proposal
validateWrite(proposal) -> WriteDecision
explainDecision(decision_id) -> EvidenceAndObligations
requestReconciliation(decision_id) -> ReconciliationRequest
approveAlternative(request_id, alternative_id, approval) -> DecisionRevision
executeAccepted(decision_id, execution_context) -> ExecutionReceipt
inspectConflict(decision_id) -> ConflictReport
compensate(receipt_id, policy_revision) -> CompensationDecision
```

These are semantic contracts, not implemented endpoints. An API transport may use HTTP, a local library, or an agent tool, but the normalized decision is the cross-language acceptance surface.

11.2 Decision state machine

The key type boundary is P8-C02: nonaccepting decisions cannot reach connector mutation. Revalidation is required if the mapping, profile, authority, identity, capability, constraint, or protected source revision changed. The current repository has no state machine that enforces this claim.

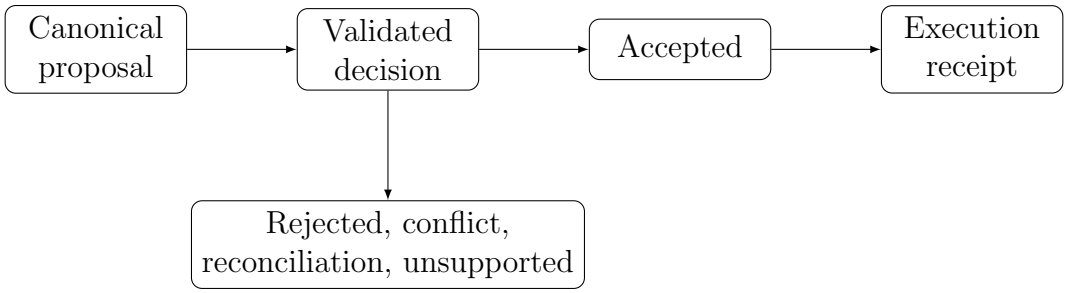

Figure 1: Proposed fail-closed decision state machine. Only an accepted decision can reach physical execution.

11.3 Compilation pipeline

The proposed write path is:

$$\begin{aligned}\text{canonical update} &\longrightarrow \text{semantic delta} \longrightarrow \text{profile decision} \\ &\longrightarrow \text{obligation discharge} \longrightarrow \text{backend plan} \longrightarrow \text{receipt}.\end{aligned}$$

The static phase checks mapping and schema typing, inverse or lens fragment, update-language membership, path ownership, static constraints, required capabilities, law references, statement templates, and expected footprint. The request phase evaluates actor, approval, identity, current capability, source versions, data-dependent constraints, ambiguity, transaction feasibility, and predicted effects. The execution phase records statements or calls without secrets, affected resources, returned versions and generated identities, constraint outcomes, partial failure, compensation, postcondition, and provenance.

11.4 Plan cache invalidation

A cached plan becomes invalid when any relevant schema, mapping, write profile, authority, identity, connector capability, dialect, or constraint revision changes. Data versions remain request-specific. Reusing a cached SQL template is different from reusing an authorization or stale snapshot.

11.5 Physical plan equivalence

Several SQL statement sequences may implement the same semantic delta. P8 therefore needs a declared normalization or equivalence relation for physical plans. Equivalence cannot ignore statement ordering when triggers, generated keys, constraints, or external effects make order observable.

12 Implementation and formalization status

12.1 Current status matrix

Table 4: P8 status at manuscript drafting.

Layer	Status	Reason
Classical view-update and lens theory	`SUPPORTED BY PRIOR LITERATURE`	Primary sources establish translators, complements, laws, relational restrictions, putback-first approaches, and delta variants.
Four-way CatDB classification	`ENGINEERING HYPOTHESIS`	No frozen profile or classifier exists.
Exact inverse detection	`OPEN`	No admitted mapping fragment or decision procedure exists.
Partial putback semantics	`OBSTRUCTED`	P4 and P7 prerequisites are not frozen.
Authority policy	`OPEN`	Shared design claim only; no P8 evaluator.
Conflict detection	`OPEN`	No source-version contract or write execution.
Multi-source write	`OBSTRUCTED`	Identity, authority, transaction, and reconciliation semantics are absent.
Rust runtime	`OPEN`	No P8 module or fixture evaluation.
Independent Haskell semantics	`OPEN`	No P8 reference evaluator or properties.
Lean laws	`OPEN`	No P8 definitions or theorems.
SQLite and PostgreSQL compiler	`OPEN`	No accepted semantic plan to lower.
API and file write-through	`OPEN`	No connector mutation capability profile.
Experiments	`OPEN`	No P8 run artifact exists.
AGY paper review	`OPEN`	Explicitly outside this authoring task.

12.2 Strongest first implementation target

The first publishable result should remain small:

- finite relational source and canonical states;
- immutable schema, mapping, instance, and profile revisions;
- required total scalar attributes;
- insert, delete, and scalar replace by stable source-local identity;
- identity, rename, total bijective conversion, checked selection, projection with complement or default, and one key-preserving join;
- exact accepted and rejected decisions;
- GetPut and PutGet for every admitted pure case;
- explicit PutPut status;

- constraint, footprint, authority, version, and capability obligations;
- local SQLite and PostgreSQL transactions; and
- multi-source cases that stop at reconciliation.

This target can establish a useful safe subset. It cannot establish universal write-back.

12.3 Implementation sequence

P8-I0: freeze the profile. Define a closed schema, diagnostics, update language, equality model, capability vocabulary, and writeability predicates. Unknown fields fail closed.

P8-I1: hand-worked fixtures. Populate accepted, rejected, unsupported, conflict, and reconciliation cases. Each accepted case includes a hand-computed source post-state, recomputed view, and physical footprint.

P8-I2: independent Haskell semantics. Implement update membership, pure putback, law checks, classification, obligations, and normalized decisions independently from Rust. Property tests remain computational evidence.

P8-I3: Rust semantic kernel. Add the write-profile and putback logic at owning crate boundaries rather than creating a crate that mirrors this paper's sections. Compare every closed fixture with Haskell.

P8-I4: policy and identity gate. Implement authority evaluation, identity-strength checking, approval binding, reconciliation requests, and no-plan guarantees for nonaccepting decisions.

P8-I5 and P8-I6: local database compilers. Compile the supported fragment first to SQLite, then PostgreSQL. Differential tests must cover native state, recomputed view, rollback, generated identity, constraints, stale versions, and trigger behavior.

P8-I7 and later: reconciliation, provenance, and connectors. Link proposals, decisions, plans, executions, conflicts, and compensation. Only then add one ETag-capable API, one file profile, and a cross-source proposal that remains reconciliation-dependent unless its consistency requirement can be met.

12.4 Formalization sequence

The initial Lean scope should contain only pure finite semantics:

1. valid source/view states, updates, constraints, and extensional equality;
2. asymmetric get and partial put;
3. GetPut, conditional PutGet, optional PutPut, and closure;
4. decidable admission and four-way classification;

5. selected constraint and footprint preservation;
6. compatible composition for selected fragments; and
7. selected state/delta equivalence.

SQL isolation, triggers, authentication, network failures, distributed transactions, effectful retry, human authority, probabilistic identity, cryptographic provenance, and performance remain systems evidence. A Lean theorem over the pure finite model cannot prove backend execution.

13 Experimental and falsification plan

13.1 Research questions

1. Does the classifier reject every preregistered unsafe or ambiguous case?
2. For admitted cases, does compiled execution match the pure oracle and recomputed canonical view?
3. What fraction of reviewed canonical updates falls in each mapping class and operation result?
4. Which constraint, authority, identity, and conflict checks dominate latency?
5. What overhead does CatDB add over equal-semantics native PostgreSQL views and hand-written transactions?
6. Does delta putback reduce work without changing semantic results?
7. Which missing evidence most often causes multi-source reconciliation?
8. Are stale reads, hidden effects, unknown outcomes, and partial commits detected without false success?

13.2 Correctness baselines

The required baselines are hand-computed fixtures, an independent Haskell evaluator, full canonical-view recomputation, native PostgreSQL updatable views where applicable, explicit trigger or rule policies, BIRDS for supported relational strategies, equivalent hand-written transactions, and expected no-write or reconciliation outcomes. Incremental relational lenses are the direct change-propagation implementation baseline [17]. Language-integrated lenses are a type-system and ergonomics comparison [16].

13.3 Safety metrics

The target is zero for:

- accepted unsafe updates;
- reconciliation cases incorrectly executed;
- normalized Rust/Haskell disagreements;
- post-view mismatches;
- post-success constraint violations;
- undeclared footprint changes;

- unauthorized target mutations;
- stale-version lost updates;
- duplicate replay effects;
- partial commits reported as success; and
- unsupported cases silently approximated.

Any nonzero safety count is a correctness failure, not statistical noise.

13.4 Failure injection

The preregistered schedule changes the source between read and execution, changes mapping or profile revisions, revokes authority, reverses an identity decision, fails the second local statement, times out before or after remote commit, duplicates delivery, downgrades capability, adds a trigger after plan caching, creates deadlock or serialization failure, and fails compensation. Every trace and minimized counterexample is retained.

13.5 Statistical boundary

Correctness is exhaustive over the closed fixture corpus. Performance uses fresh independent units, recorded seeds, fixed warmups, at least 30 measured invocations per unit, randomized paired blocks, explicit cache treatments, median and p95, dispersion, and cluster-bootstrap intervals. Comparisons require equal semantic result, isolation, constraints, provenance mode, and failure semantics.

The shared 10% cached-execution noninferiority margin is applicable only where the native and CatDB contracts are equivalent. Baselines with different trigger effects or isolation are reported separately.

13.6 Direct falsifiers

Table 5: Condensed falsification plan.

ID	Claim under test	Direct falsifier
F1	Admission is bounded by the declared language.	An out-of-language update is accepted.
F2	Fully writable is total on the declared language.	One well-typed, precondition-satisfying update lacks an accepted translation.
F3	Partial writeability is inspectable.	The profile does not publish a decidable strict admitted subset.
F4	Accepted pure updates satisfy declared PutGet.	Recomputed $g(s')$ differs under the named equality.
F5	Claimed GetPut is stable.	An unchanged view changes semantic source state.

ID	Claim under test	Direct falsifier
F6	Claimed PutPut ignores history.	Two histories produce different semantic states under the declared equality.
F7	Footprints are enforced.	A source, relation, tenant, field, or identity outside the footprint changes.
F8	Constraints are preserved.	An accepted post-state violates a relied-on source or canonical constraint.
F9	Ambiguity is not guessed.	Several semantic translations exist and one is chosen without policy or reconciliation.
F10	Identity thresholds gate execution.	A probable, possible, or contradicted identity becomes a write target.
F11	Authority is non-bypassable.	A denied, absent, or drifted authority derivation still commits.
F12	Version checks prevent stale overwrite.	A protected concurrent change is overwritten without typed conflict.
F13	Capabilities are honest.	The system relies on unavailable transaction, rollback, version, or idempotency support.
F14	Atomicity labels match execution.	A partial commit is reported as successful atomic execution.
F15	Retry is safe.	A non-idempotent external effect is duplicated after retry.
F16	Read-only refuses before mutation.	Connector mutation occurs before or during a read-only decision.
F17	Schema inverse is separate from putback.	A quasi-inverse is executed as policy without operation-level evidence.
F18	Categorical constructions are not write proof.	Δ_F, Σ_F, Π_F, a natural transformation, pullback, or pushout is treated as sufficient authorization and execution evidence.
F19	Composition is checked.	Two profiles compose while violating a domain, constraint, authority, effect, version, or law obligation.
F20	Exact-effect deletion is honest.	Deletion has unrequested canonical side effects under an exact profile.
F21	Runtime conditions remain visible.	A conditional profile is reported statically without unresolved conditions.

ID	Claim under test	Direct falsifier
F22	Independent evaluators agree.	Rust and Haskell emit different normalized decisions for a closed fixture.
F23	Formal claims match encodings.	The prose exceeds Lean assumptions or omits axioms.
F24	Performance baselines are fair.	Results compare different semantics, isolation, constraints, or failure modes.
F25	Novelty is specific.	The proposed mechanism reduces to an established lens or SQL view mechanism with no validated CatDB delta.

14 Claim table

Table 6: P8 manuscript claim anchors.

Anchor	Candidate claim	Status	Evidence required
P8-C01	The four mapping classes are total and mutually exclusive for a well-formed profile.	`OPEN`	Frozen D_P/R_P, classifier tests, and finite Lean theorem.
P8-C02	A nonaccepting decision cannot reach connector mutation.	`OPEN`	Type or state-machine boundary, misuse tests, and receipts.
P8-C03	Every accepted pure v1 update satisfies the declared round-trip laws.	`OBSTRUCTED`	Frozen putback, fixtures, generated cases, and selected proof.
P8-C04	Accepted v1 updates preserve all declared source and canonical constraints.	`OBSTRUCTED`	Constraint semantics, mutation tests, and database conformance.
P8-C05	Accepted exact-footprint plans change nothing outside the declared footprint.	`OPEN`	Effect analysis, trigger audit, and full before-after diff.
P8-C06	No physical target is mutated without a current authority derivation.	`OBSTRUCTED`	Policy IR, revision checks, and non-bypass tests.
P8-C07	Identity-sensitive writes require the declared identity strength.	`OBSTRUCTED`	P7 contract, negative fixtures, and approval binding.

Anchor	Candidate claim	Status	Evidence required
P8-C08	Protected stale reads become typed conflicts rather than lost updates.	`OPEN`	Source-version contract and scheduled concurrency tests.
P8-C09	Profile composition requires domain, law, constraint, authority, version, and effect compatibility.	`OBSTRUCTED`	Compatibility algorithm, counterexamples, and selected proof.
P8-C10	SQLite and PostgreSQL plans match the reference semantic post-state and decision class.	`OPEN`	Differential fixtures and native recomputation.
P8-C11	Rust and independent Haskell evaluators agree on every closed P8 fixture.	`OPEN`	Shared manifest, independent implementations, and comparison report.
P8-C12	Cached semantic planning is noninferior to an equal-semantics native baseline on named workloads.	`OPEN`	Accepted runtime and preregistered paired benchmark.

The anchors are independent. Evidence for one row does not promote a neighboring row. In particular, a pure lens proof does not promote database authority or transaction claims.

15 Related work and novelty boundary

15.1 Classical view-update theory

Update translators, constant complements, correctness criteria, operator algorithms, and consistent-view programs are established [2, 6, 5, 20, 12]. A generic claim that CatDB solves the view-update problem would be false as novelty and unsupported as a result.

15.2 Lenses and incremental bidirectionality

Asymmetric lenses provide get and put with round-trip laws and compositional operators [10]. Relational lenses establish useful selection, projection, and join fragments under dependencies and policy [3]. Symmetric lenses retain private information on both sides [15]. Delta and incremental lenses make changes explicit and reduce recomputation [7, 17]. P8 must prove any selected state/delta correspondence rather than assuming it.

15.3 Putback-first systems

BiGUL and axiomatic putback-first programming make backward behavior primary and check well-behavedness [22, 21]. BIRDS specifies relational view-update strategies in Datalog and compiles practical PostgreSQL behavior [26, 25]. The proposed CatDB profile is closer to this tradition than to automatic inversion.

15.4 Schema inverses

Schema mappings do not all have inverses, and quasi-inverse existence remains relative to an instance class and equivalence [8, 9]. A quasi-inverse can inform recovery. It does not supply user intent, authority, or a transaction. Constraint- and determinacy-relative accounts further reinforce the need for fragment-specific claims [11].

15.5 Collaborative update exchange

Update exchange, bidirectional data exchange, and Dejima already address networks of mappings, trust, peer autonomy, side effects, and propagation [13, 19, 1]. Recent partial-state lens work also addresses partially specified intentions and competing multi-view updates [23]. P8 must compare its reconciliation and authority boundary directly with these systems.

15.6 Native SQL views

PostgreSQL automatically updates restricted simple views, exposes mixed writable and read-only behavior, supports check options, and permits explicit rules or triggers [24]. This is a capability and differential baseline. It is not proof that a CatDB mapping has the intended putback, authority, or side-effect contract.

15.7 Plausible CatDB delta

The defensible prospective contribution is not new lens theory. It is a versioned, operation-relative systems contract that connects:

- read-mapping identity and explicit put semantics;
- four mapping-level writeability classes and operation-level decisions;
- authority, identity strength, constraints, and source capabilities;
- exact refusal and reconciliation diagnostics;
- local backend compilation with equal-semantics comparison; and
- immutable decision and execution provenance.

This delta remains `OBSTRUCTED`. It becomes a contribution only after the stated implementation, formal, database, experimental, and review gates pass.

16 Limitations and open questions

16.1 Limitations

This manuscript has no P8 execution evidence. Its profile is a candidate contract, not an accepted schema. The relational fragment has not been fixed, the equality model is unsettled, and no fixture shows that the proposed decision surface is usable. The paper can reveal missing obligations now, but it cannot tell us how often real updates will be accepted or how expensive the checks will be.

The first scope also leaves out hard cases on purpose. General aggregation, recursive views, vector search, arbitrary user code, bulk predicate mutation, general Σ_F and Π_F, distributed transactions, and cross-source atomic write-back are excluded. A later system may add policies for some of them. That would create new profile revisions and new proof and failure obligations, not extend the v1 result for free.

Formal assurance will remain layered. Lean can check a pure finite model, but it cannot certify an undocumented trigger or a remote service's timeout behavior. Database tests can exercise rollback and isolation, but they cannot prove business authority. Rust and Haskell agreement can find specification drift, but two implementations can share a mistaken oracle. The evidence needs to remain separate.

The four mapping labels are summaries. They do not replace per-path and per-operation decisions. A profile may also depend on runtime capabilities and versions that static analysis cannot settle. The API must expose those conditions rather than promise more certainty than the sources provide.

16.2 Open questions

1. What is the smallest useful canonical update language for v1?
2. Should the user see classification at mapping, path, operation, or all three levels?
3. Which equality supports bags, nulls, order, generated identities, and provenance?
4. Is semantic PutPut required in v1, or only recorded where available?
5. Which projection constructors and functional-dependency fragments are portable?
6. How should source-generated identities appear before execution?
7. Which source-version guard protects a predicate update?
8. Can trigger-induced effects be discovered and cached safely?
9. Which connector capabilities can be tested instead of self-declared?
10. What makes two physical plans semantically equivalent?
11. When can a reconciliation decision become a reusable profile rule?
12. How are approvals invalidated by source, identity, or policy drift?
13. Which commutative field operations can merge concurrent edits?
14. How should composition narrow admitted domains?
15. Can footprint analysis be complete for the first relational fragment?
16. What overhead makes semantic write-through impractical?

16.3 Obstructed questions

General write-back through Σ_F or Π_F is obstructed by unsupported migration semantics and missing operation policies. Cross-source writes are obstructed by identity, authority, and later consistency semantics. Profile composition is obstructed by the absence of frozen read transformation and effect languages. Reconciliation execution is obstructed by the missing identity-decision model. Provenance-complete history is obstructed by the absent P13 runtime. Database comparison and performance claims are obstructed until a write compiler exists and all baselines have equal contracts.

17 Exact nonclaims

This paper makes the following nonclaims:

1. CatDB does not solve the view-update problem in general.
2. Not every categorical schema mapping is invertible.
3. Not every invertible mapping is operationally writable.
4. A quasi-inverse does not establish intended source effects.
5. A functor, adjunction, natural transformation, pullback, or pushout does not authorize a write.
6. Δ_F, Σ_F, and Π_F are not universal write-back operators.
7. Round-trip laws do not establish authority, identity, isolation, retry safety, or acceptable side effects.
8. Category theory does not choose business ownership or conflict policy.
9. CatDB will reject many canonical updates.
10. `fully_writable` is relative to a bounded, versioned update language.
11. `partially_writable` must expose its admission predicate.
12. `read_only` is a valid permanent classification.
13. `reconciliation_dependent` requires a new decision and cannot fall back to an arbitrary choice.
14. Vector similarity cannot establish identity or write authority.
15. Provenance does not grant authority or prove correctness.
16. Database credentials do not establish semantic ownership.
17. Native SQL view updatability does not prove CatDB correctness.
18. A trigger is code requiring review, not a proof.
19. A local transaction does not provide cross-source atomicity.
20. Compensation is not rollback.
21. Timeout does not prove failure.
22. Retrying a generated plan is not necessarily idempotent.
23. Federation does not imply write federation.
24. Materialization does not become authoritative because it is local.
25. Source-local identity is not automatically canonical identity.
26. Probable equivalence is not writable identity.
27. Deterministic tie-breaking is not semantic reconciliation.
28. Static classification cannot erase runtime version conditions.

29. Constraint checks can be incomplete when sources hide state or effects.
30. A post-write row count does not prove exact canonical effect.
31. Rust/Haskell agreement is computational evidence, not a theorem.
32. Lean laws over a pure finite model do not prove backend execution.
33. No current CatDB P8 runtime or formal claim is accepted.
34. The current bounded Δ_F slice is not write-back evidence.
35. CatDB does not eliminate migration, ETL, reconciliation, or distributed-systems tradeoffs.

18 Conclusion

Safe write propagation starts by admitting how often a read mapping is not enough. The old source may contain the missing complement. A join may offer several targets. A source may own one field but not another. The record may be only a probable identity match. A timeout may hide a commit. None of these problems is repaired by calling the mapping bidirectional.

The proposed CatDB answer is a versioned write profile with a typed refusal surface. It separates inverse class from write class, and it classifies each mapping as fully writable, partially writable, read-only, or reconciliation-dependent relative to a declared update language. Every operation still receives its own decision and may fail on authority, identity, constraints, source versions, capabilities, effects, or transactions.

That design is useful only if the refusal path is real. A nonaccepting decision must never reach a connector. An ambiguous union must preserve both branches. A probable identity must remain a proposal. A cross-source batch must not report atomic success when the mechanism cannot provide it. These are testable conditions, and the paper records their falsifiers.

The current result stops there. The profile is not frozen, no P8 evaluator or database compiler exists, and no P8 proof or experiment has run. The status therefore remains `OBSTRUCTED`. The next honest step is to freeze the small relational profile and build counterexamples before adding a write endpoint. Universal safe write-back remains rejected.

A Detailed fixture plan

Every fixture should carry immutable profile and schema revisions, source state and version tokens, canonical update, actor and identity context, expected normalized decision, expected post-state for accepted cases, exact footprint, discharged and failed obligations, and the expected diagnostic.

A.1 Fully writable candidates

Fixture	Expected oracle
`write-identity-replace-ok`	One field changes; GetPut, PutGet, constraints, authority, and version pass.
`write-rename-insert-ok`	One source insert with exact renamed fields and caller-specified identity.
`write-bijective-scalar-ok`	Inverse conversion round-trips over the declared exact domain.
`write-batch-local-atomic-ok`	Two local changes commit together or both roll back.

A.2 Partially writable cases

Fixture	Expected oracle
`write-selection-preserve-ok`	Replacement keeps the selection predicate true.
`write-selection-check-reject`	Predicate violation returns typed rejection with no effect.
`write-projection-replace-ok`	Omitted source fields are preserved from the exact complement.
`write-projection-insert-missing-error`	A required omitted field has no constructor.
`write-join-owned-left-ok`	Only the declared left-side field changes.
`write-join-shared-key-reject`	Shared-key update lies outside the profile.
`write-computed-column-read-only`	Base columns are writable while the computed path is rejected.

A.3 Read-only and reconciliation cases

Fixture	Expected oracle
`write-aggregate-read-only`	No plan exists for average replacement.
`write-vector-match-read-only`	Similarity cannot establish target identity.
`write-historical-snapshot-read-only`	Mutation is refused before connector invocation.
`write-connector-no-mutation`	Capability failure is normalized without a write attempt.
`write-untagged-union-branch`	Both branch alternatives are retained; no physical plan is emitted.

Fixture	Expected oracle
`write-full-name-ambiguous-split`	Several decompositions are retained.
`write-probable-identity`	Candidate records are returned, but none is writable.
`write-dual-authority-conflict`	Contradictory ownership policies are surfaced.
`write-cross-source-no-atomicity`	Staged, compensated, and reject alternatives appear; no success is reported.
`write-concurrent-intentions`	Incompatible path updates require reconciliation.

A.4 Constraint, side-effect, and failure cases

Fixture	Expected oracle
`write-unique-violation`	The transaction is rejected and rolled back.
`write-foreign-key-missing`	Preflight or native transaction rejects with a typed layer.
`write-path-equation-break`	Semantic validation rejects before execution.
`write-join-fanout-side-effect`	Predicted extra view rows block an exact-footprint profile.
`write-union-delete-still-visible`	The requested effect is not achieved, so the operation is rejected.
`write-trigger-hidden-side-effect`	Backend conformance detects a footprint mismatch.
`write-stale-row-version`	Typed source-version conflict; no mutation.
`write-mapping-revision-drift`	The cached plan is invalidated.
`write-authority-revision-drift`	Approval must be revalidated.
`write-timeout-outcome-unknown`	No blind retry occurs.
`write-idempotency-replay-ok`	One effect commits and the receipt is stable.
`write-local-second-statement-fails`	The local transaction fully rolls back.
`write-cross-source-partial-commit`	Partial outcome and compensation state are explicit; success is impossible.

A.5 Composition cases

Fixture	Expected oracle
`write-compose-identity-ok`	Composite and direct semantic results agree.
`write-compose-domain-narrowing`	The composite becomes partially writable.
`write-compose-authority-obstructed`	An intermediate target lacks authority.
`write-compose-constraint-counterexample`	Individually valid local plans are rejected as a composite.
`write-compose-effect-duplication`	A repeated non-idempotent effect blocks composition.

B Generative properties

Independent generators should check:

1. GetPut over valid source states;
2. PutGet over admitted updates;
3. PutPut only for profiles that declare it;
4. source and canonical constraint preservation;
5. footprint preservation;
6. classifier totality over normalized profiles;
7. no accepted decision with an undischarged blocking obligation;
8. no write through insufficient identity evidence;
9. deterministic target selection from policy evidence;
10. accepted plan versus recomputed-view equivalence;
11. transaction rollback under injected statement failure;
12. Rust/Haskell normalized-decision agreement; and
13. shrinking to one semantic fault without changing the expected class.

Random passing tests would provide computational evidence. They would not constitute a proof.

C Planned verification commands

The following commands are preregistered gates. Their targets do not currently exist, so they are not current verification results.

```
python scripts/validate-fixtures.py \
  --schema fixtures/write-profile-v1.schema.json \
  --cases fixtures/cases/write-profile-v1

python scripts/audit-fixture-manifest.py \
  fixtures/write-profile-v1-manifest.json

cabal build all --ghc-options="-Wall␣-Werror"
```

```
cabal test catdb-reference:p08-write-profile-v1
cabal test catdb-reference:p08-write-properties

cargo fmt --all --check
cargo clippy --workspace --all-targets --all-features -- -D warnings
cargo test --workspace --all-features

cargo run -p catdb-cli -- write fixtures verify \
  fixtures/write-profile-v1-manifest.json

cargo run -p catdb-cli -- write fixtures compare \
  --left rust --right haskell \
  fixtures/write-profile-v1-manifest.json

lake build
python scripts/audit-lean.py

cargo test -p catdb-storage-sqlite --test write_profile_v1
cargo test -p catdb-storage-postgres --test write_profile_v1
cargo test -p catdb-storage-postgres --test write_conflicts
cargo test -p catdb-storage-postgres --test write_rollback
```

D Reader-facing decision examples

D.1 A selection update that remains visible

An analyst changes an account's display label while the account remains active. The profile recognizes a scalar replacement, finds one authoritative source field, checks the expected row version, and predicts no additional view effect. If every obligation passes, the semantic decision can be accepted. The database transaction still has to confirm the version and constraints.

D.2 A projection insertion with missing source data

A user inserts a canonical employee with name and department. The source also requires a payroll region, but the view does not expose it and the profile has no default or constructor. The system returns a typed rejection before building a physical mutation. Guessing a region would satisfy a database constraint while inventing business data.

D.3 An untagged union deletion

A canonical contact appears in two sources that use the same projected shape. The user asks to delete it. Removing the row from the CRM, the support system, or both has a different

meaning. The profile returns a reconciliation request with those alternatives and predicted view effects. It does not use connector order as intent.

D.4 A stale authority approval

A manager approves a credit-limit change against profile revision 14. Before execution, ownership moves to another ERP team and the authority policy becomes revision 15. The approval is no longer valid for the current decision. The request must be revalidated, even if the SQL statement and database credential would still work.

D.5 A remote timeout

An API call times out after the request leaves CatDB. Without an idempotency receipt or source query that proves the outcome, the result is unknown. Blind retry could duplicate a non-idempotent effect. The conflict report preserves that uncertainty and does not say the write failed.

D.6 A multi-source batch without atomicity

A user edits a CRM name and an ERP credit limit in one batch and requests atomic execution. Both path updates may be valid on their own. If the sources offer no common transaction and the caller does not accept compensation, the batch is reconciliation-dependent. Reporting two local successes as one atomic success would falsify the contract.

References

[1] Asano et al. Flexible framework for data integration and update propagation: System aspect. In *2019 IEEE International Conference on Big Data and Smart Computing*, 2019.

[2] François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[3] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: A language for updatable views. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 338–347, 2006.

[4] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002.

[5] Stavros S. Cosmadakis and Christos H. Papadimitriou. Updates of relational views. *Journal of the ACM*, 31(4):742–760, 1984.

[6] Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 7(3):381–416, 1982.

[7] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From state- to delta-based bidirectional model transformations: The asymmetric case. *Journal of Object Technology*, 10(1):6:1–25, 2011.

[8] Ronald Fagin. Inverting schema mappings. *ACM Transactions on Database Systems*, 32(4), 2007.

[9] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Quasi-inverses of schema mappings. *ACM Transactions on Database Systems*, 33(2), 2008.

[10] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems*, 29(3), 2007.

[11] Enrico Franconi and Paolo Guagliardo. The view update problem revisited. *arXiv preprint arXiv:1211.3016*, 2012.

[12] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, 1988.

[13] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 675–686, 2007.

[14] Xiao He and Zhenjiang Hu. Putback-based bidirectional model transformations. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018.

[15] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. Symmetric lenses. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 371–384, 2011.

[16] Rudi Horn, Simon Fowler, and James Cheney. Language-integrated updatable views. *arXiv preprint arXiv:2003.02191*, 2020.

[17] Rudi Horn, Roly Perera, and James Cheney. Incremental relational lenses. *Proceedings of the ACM on Programming Languages*, 2(ICFP), 2018.

[18] Michael Johnson and Robert Rosebrugh. Lenses, fibrations and universal translations. *Mathematical Structures in Computer Science*, 22(1):25–42, 2012.

[19] Grigoris Karvounarakis and Zachary G. Ives. Bidirectional mappings for data and update exchange. In *Proceedings of the 11th International Workshop on the Web and Databases*, 2008.

[20] Arthur M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 154–163, 1985.

[21] Hsiang-Shang Ko and Zhenjiang Hu. An axiomatic basis for bidirectional programming. *Proceedings of the ACM on Programming Languages*, 2(POPL), 2018.

[22] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. BiGUL: A formally verified core language for putback-based bidirectional programming. In *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, 2016.

[23] Kazutaka Matsuda, Manh Nguyen, and Meng Wang. Lenses for partially-specified states. In *Programming Languages and Systems*, 2026.

[24] PostgreSQL Global Development Group. *CREATE VIEW*. PostgreSQL Global Development Group, 2026. Official documentation; experimental work must pin the tested PostgreSQL version.

[25] Dung Tran, Hiroyuki Kato, and Zhenjiang Hu. BIRDS: Programming view update strategies in datalog. *Proceedings of the VLDB Endowment*, 13(12):2897–2900, 2020.

[26] Dung Tran, Hiroyuki Kato, and Zhenjiang Hu. Programmable view update strategies on relations. *Proceedings of the VLDB Endowment*, 13(5):726–739, 2020.