Schema Migration as a Typed Mapping: A Functorial Model of Database Evolution

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

23 July 2026

Abstract

A database migration script specifies commands, but it can leave the meaning of the change implicit. It need not record which old concept corresponds to which new concept, whether a conversion is total, what information is discarded, which ambiguity a person resolved, or whether a compatibility view and a data rewrite should return the same result. This paper gives a provisional contract for recording those facts before execution. A CatDB migration candidate relates two immutable schema revisions through a typed semantic relation. It also records direction, coverage, determinacy, conservative loss evidence, synthesis, ambiguity, authority, typed obligations, provenance, and execution intent. The intent may request eager, lazy, virtual, hybrid, or mixed-version execution, but it does not change the declared meaning.

The mathematical fragment admitted here is deliberately small. For a lawful total mapping $F : S \to T$ and a valid finite target instance J, pullback has the contravariant direction $\Delta_F : T\text{-}\mathsf{Inst} \to S\text{-}\mathsf{Inst}$. Under the finite-total assumptions inherited from the P1 dossier, we give complete paper proofs that pullback preserves instance validity and respects identity and contravariant composition. These are PROVED IN P1 DOSSIER results, not machine checks of migration code. A ‘Customer‘/‘Order‘ revision chain shows a stable-identity field-to-path read, a stronger value-domain obligation, and the point at which a split or merge needs construction or identity policy beyond a functor-only map.

The systems part remains a hypothesis. Accepted prerequisite artifacts compute a bounded schema/path/mapping profile in Rust and Haskell, and they machine-check selected path and mapping declarations in Lean. They do not execute migration. The four migration, two information-loss, and two schema-diff envelopes are STRUCTURAL-ONLY. There is no migration executor, loss detector, diff engine, physical compiler, backend runner, or benchmark result. This manuscript therefore makes no claim about a physical plan, online cutover, performance, or a backend. The candidate contribution combines an immutable typed migration package, bounded diagnostics, explicit semantic/physical separation, dependency impact, and layered evidence. Functorial

migration, schema-evolution operators, data exchange, lenses, and compatible versions are established prior work. So are online and lazy migration, registries, upcasters, and database versioning. The evaluation contract asks whether the narrower combination adds anything.

Contents

1 Introduction

1.1 The change request that is not yet a migration

Consider an application with a `Customer` table and an `Order` table. One change request asks to rename a customer field, add a required historical value, split a customer into a person and an account, and reject orders whose monetary value falls outside a stronger domain. An ordered DDL file can contain `ALTER TABLE`, `UPDATE`, and `CREATE VIEW` statements without answering the semantic questions.

For example, did the rename preserve one declaration's identity, or did one field disappear while another received a similar name? What value should the required field have for older rows? The split raises more questions. Does it create two objects for every old customer, or only when a discriminator matches? Which identifier relates those objects, and what happens when that identifier or discriminator is missing? The monetary conversion still needs a choice among rejection, rounding, clamping, and overflow. An old client might also need to write through a view. Finally, a physical rollback may stop short of restoring information that has already been discarded.

SQL remains a suitable implementation language. The problem is that a physical command sequence does not fully state the meaning of the change. A rewrite, an on-read conversion, a compatibility view, or a protocol for coexisting old and new clients could all implement the same semantic change. Conversely, similar-looking DDL can encode different choices about identity, loss, and failure.

This paper proposes the following ordering discipline:

state a typed, versioned semantic contract first

$$\Downarrow$$

choose and validate a physical realization second

This is an ENGINEERING HYPOTHESIS, not a claim that category theory removes locks, backfills, retries, data movement, downtime, or coordination.

1.2 Why a mapping is necessary but insufficient

Functorial data migration gives precise mathematical operators induced by a schema functor [21, 22]. FQL and CQL implementations demonstrate that these operators are executable

[4, 5, 24]. Unlike a name-based diff, a typed mapping can check whether mapped paths have the right endpoints, whether every source generator has an image, and whether accepted source equations remain equal after mapping.

Even a lawful total mapping may identify two source objects. It may have no inverse. It need not decide how to create target-only objects. It says nothing by itself about old-client writes, backend locks, online version skew, or recovery. For that reason, the phrase "migration is a functor" is too strong for the systems object considered here. P3 begins with a mapping in one admitted fragment and places it inside a larger migration package.

1.3 Research question and claimed scope

The research question is:

> What semantic information must be stated before executing a schema migration?

The proposed answer is an immutable package that contains exact old and new schema revisions, a typed relation, direction, coverage, result determinacy, loss evidence, synthesis and identity policy, unresolved ambiguity, preconditions, postconditions, authority, provenance, and non-semantic execution intent. The paper contributes a specification and formal boundary, a migration taxonomy, a concrete revision chain, a prior-art comparison, and a falsifiable implementation and experiment contract.

It does not contribute a migration runtime. It does not report a generated physical plan. It does not establish a new categorical migration operator. The central package has not yet been implemented or evaluated, so the paper remains provisional.

1.4 Contributions and noncontributions

The author-side contributions are:

1. a versioned migration-package definition that keeps semantic meaning distinct from execution intent;
2. six orthogonal classification axes for coverage, determinacy, information, reversibility, writeability, and execution;
3. a taxonomy that states representability, required policy, obligations, physical choices, current status, and a falsifier for each change class;
4. a correct contravariant 'delta' contract with two complete finite-total paper proofs inherited from P1;
5. a dependency-impact and provenance contract; and
6. a preregistered implementation and evaluation matrix designed to reject overbroad claims.

The paper does not claim that functorial migration, schema-change operators, model management, universal solutions, lenses, compatibility views, co-existing schemas, online or lazy migration, migration histories, schema registries, event upcasting, or database branching are new. Those exclusions define the scope of the argument.

2 Evidence vocabulary and current implementation status

2.1 Labels

Every result-like statement in this manuscript belongs to one of the evidence classes in Table 1. The labels prevent a representation test from being narrated as a semantic result and prevent a mathematical argument from being narrated as a program proof.

Table 1: Binding evidence vocabulary.

Label	Meaning
SUPPORTED PRIOR LITERATURE	A cited primary paper or official document supports the statement within its own assumptions.
PROVED IN P1 DOSSIER	The P1 dossier contains a human-readable proof under explicit finite-total assumptions; this is not machine checking.
COMPUTATIONAL	A named program computed a named result at a recorded revision.
STRUCTURAL-ONLY	A closed representation decoded, canonicalized, or round-tripped; the semantic operation was not computed.
ENGINEERING HYPOTHESIS	Proposed behavior requiring implementation and experiment.
OPEN CONJECTURE	A proposed property without an accepted proof or sufficient experiment.
OBSTRUCTED	Required definitions or equality/decidability boundaries are not fixed.
UNSUPPORTED	The current profile represents or names the request but deliberately rejects it.

One subject can have different labels at different layers. The pullback law is PROVED IN P1 DOSSIER. A JSON request carrying `operation: delta` may be STRUCTURAL-ONLY. A generated PostgreSQL compatibility view is an ENGINEERING HYPOTHESIS. No label promotes the others.

2.2 Accepted prerequisite evidence

The accepted P1 prerequisite boundary is narrow:

- Rust computes the bounded schema, typed-path, lawful-mapping, identity-mapping, and mapping-composition operations in the selected profile.
- An independent Haskell model agrees on 16 semantic outcomes and 8 structural-only outcomes, with zero recorded disagreements at the accepted revision.
- Lean checks exact declarations covering `PATH-02` through `PATH-04` and `MAP-03` through `MAP-05`. Those declarations concern typed paths and lawful mappings.

Evidence status. COMPUTATIONAL prerequisite evidence, bounded to P1..

These artifacts do not validate finite instances, compute Δ, detect information loss, compute a schema diff, compile a compatibility view, or run a backend migration. In particular, the Lean declarations do not machine-check Δ, instance validity, loss, diff, physical plans, or online execution.

2.3 P3 artifact audit

The current P3 representation contains four migration envelopes, two information-loss envelopes, and two schema-diff envelopes. Their expected outcomes describe future acceptance tests. Present evidence shows only that the values are admitted by a closed IR and can be canonicalized or round-tripped.

Table 2: Current P3 semantic-operation boundary.

Artifact class	Count and representation	Current status
Migration validation	four envelopes: valid pullback, schema mismatch, unsupported Σ, unsupported Π	STRUCTURAL-ONLY; no semantic result computed
Information-loss detection	two envelopes: object identification and incomplete mapping	STRUCTURAL-ONLY; no witness computed
Schema diff	two envelopes: entity addition and duplicate-ID error	STRUCTURAL-ONLY; no diff computed
Physical migration	no compiler, plan, or runner	absent
Evaluation	no backend or human-study run	absent

There is no `catdb-migration` crate, migration Haskell oracle, migration Lean theorem, physical-plan compiler, SQLite/PostgreSQL migration runner, or benchmark result. Structural round trips are useful IR evidence, but they are not migration results.

3 Prior art and novelty boundary

3.1 Functorial migration and categorical systems

Spivak defines schemas categorically, instances as set-valued functors, and the functor-induced Σ, Δ, and Π operators [21]. Spivak and Wisnesky connect this foundation to relational query languages and composition [22]. FQL and CQL case studies show executable categorical migrations and integration, including nontrivial type-side behavior [4, 5, 24].

These are SUPPORTED PRIOR LITERATURE results. CatDB cannot claim invention of mappings, pullback, left/right migration, or mapping composition.

The direct relevance is also a warning. A Σ may need generated values and quotient-like behavior; a type algebra can be nontrivial. A mapping's existence does not imply that a forward construction is deterministic, null-free, or information preserving. P3 therefore supports only the restricted pullback direction and names Σ and Π as UNSUPPORTED.

3.2 Schema evolution and model management

PRISM already links Schema Modification Operators, derived mappings, information-preservation conditions, composed evolution histories, legacy query support, and generated views or migration scripts [7]. PRISM++ adds integrity-constraint modification, impact analysis, and legacy query/update rewriting in a restricted operator language [8]. These systems are the most direct comparison for P3's taxonomy and compatibility claims.

Model Management 2.0 treats models and mappings as first-class and organizes match, compose, diff, merge, translation, inversion-related operations, and generated transformations [1]. Algebraic model management connects such operators with algebraic and categorical structure [20]. Accordingly, neither "mappings are first-class" nor "schema changes compose" is an acceptable novelty statement.

3.3 Data exchange and mapping-language boundaries

Data exchange gives semantics to source instances, dependencies, target solution spaces, universal solutions, chase construction, and certain answers [11]. This framework is adjacent to future CatDB synthesis and bounded Σ, especially when target values must be created or several solutions are possible. P3 profile v1 instead assumes a finite target instance for pullback and does not construct general target solutions.

Composition is language-relative. Ordinary source-to-target dependencies are not generally closed under composition, while second-order dependencies have a closure result under their specified language [12]. P3 therefore never uses "composable" without naming endpoint agreement, the admitted mapping fragment, coverage, and the policy artifacts that must accumulate.

3.4 Lenses and co-existing versions

Relational lenses establish well-behaved updateable views only under restrictions and round-trip laws [3]. InVerDa/BiDEL generates view/trigger delta code so several schema versions can coexist over one data set, while decoupling logical evolution from physical design [13]. Compatibility views and simultaneous versions are thus established. P3 uses them as baselines and reserves write propagation for a separate writeability analysis.

3.5 Physical transition systems and production tools

F1 demonstrates that asynchronous online change can corrupt data when dangerous transitions are not decomposed into safe intermediate states under bounded version skew [18]. BullFrog

demonstrates lazy relational evolution with conversion on access, concurrency control, and an eager/lazy comparison [2]. These works make online and lazy migration prior art and show why a semantic relation is not a cutover protocol.

PostgreSQL documents operation-specific locks, scans, `USING` conversions, dependency behavior, staged constraint validation, and possible table rewrites [16]. Its view documentation states restricted automatic-updateability conditions [17]. Backend documentation supports statements about PostgreSQL; it is not evidence that CatDB generates a correct plan.

Flyway provides ordered, once-only, checksum-tracked versioned scripts and schema history [19]. Alembic autogenerate produces candidate operations requiring review and documents that table/column renames are not inferred, appearing as add/drop pairs [23]. They are required workflow baselines.

3.6 Compatibility policies, upcasters, and versioned data

Confluent Schema Registry supplies configurable backward, forward, full, and transitive policies for message schemas [6]. EventSourcingDB documents immutable event versions and on-read normalization in application or projection code [10]. These are useful compatibility and lazy-conversion analogues, but neither defines arbitrary relational split, backfill, or CatDB mapping laws.

Relational schema versioning predates CatDB [9]. Decibel and ORPHEUSDB demonstrate relational dataset branching/version control with physical storage, query, and migration tradeoffs [14, 15]. Immutable versions are necessary here, not novel by themselves.

3.7 Plausible delta

The narrow candidate contribution is the combination of (i) one immutable package tying exact schema revisions to a typed relation, loss, partiality, synthesis, ambiguity, obligations, authority, and execution intent; (ii) bounded deterministic diagnostics; (iii) explicit semantic/physical separation; (iv) a CatDB-wide dependency impact set; and (v) layered Rust/Haskell/Lean evidence. Each component has strong precedents. The combination is an ENGINEERING HYPOTHESIS until implementation and direct evaluation show a meaningful difference.

4 Immutable schema revisions and migration packages

Definition 4.1 (Immutable schema revision)**.** A schema revision is a record

$$\mathsf{SchemaRevision} = (\mathsf{id}, \mathsf{semanticBodyHash}, \mathsf{parentRefs}, \mathsf{sourceDigest}, \mathsf{profileVersion}).$$

The semantic-body hash covers a canonical typed schema body. Source location and formatting are provenance but do not define semantic identity. Mutable branch names may point to immutable revisions; they are not revision IDs.

Two independently declared but categorically isomorphic schemas remain distinct revisions unless a checked equivalence artifact connects them. This choice prevents a mathematical

existence statement from silently rewriting historical identity. A stable declaration ID similarly records continuity only when an authorized revision preserves it; reuse after a meaning change must be rejected or represented as a new ID.

Definition 4.2 (Migration package)**.** For schema revisions S_0 and S_1, a migration candidate is

$$\mathcal{M}_{01} = (S_0, S_1, R, \mathsf{direction}, \mathsf{coverage}, \mathsf{loss}, \mathsf{synthesis}, \mathsf{ambiguity}, \mathsf{obligations}, \mathsf{executionIntent}).$$

The package also carries its own immutable ID/body hash, authority decisions, and provenance references. Initially, R may be a lawful total mapping in the finite 'delta' profile.

The fields have independent jobs:

S_0, S_1 exact endpoint revisions, not branch aliases;

R the admitted semantic relation;

direction the input instance category and required result category;

coverage total, partial on a named domain, or unsupported;

loss named conservative checks, concrete witnesses, inconclusive checks, and their scope;

synthesis defaults, total expressions, fresh-identity rules, or external decisions needed to construct values;

ambiguity alternatives, selection policy, authority, and unresolved cases;

obligations typed preconditions and postconditions with evidence;

execution intent a request for a physical mode that does not alter the semantic result.

The physical plan is a separately versioned artifact. A backend compiler may regenerate it for a new capability manifest without changing $\mathcal{M}_{01}$, but only if it preserves the declared result contract. Changing a converter, identity rule, failure policy, or accepted result is a semantic package change, not a harmless optimizer choice.

4.1 Orthogonal classification axes

A package can be total and lossy, deterministic and noninvertible, virtual and read-only, or semantically defined and physically unsupported. "Lossless" appears in this paper only as `lossless_under_checked_criteria`. The phrase makes incompleteness visible.

4.2 Obligations

An admitted result names, rather than infers, each discharged obligation:

1. schemas well formed;
2. mapping total on the declared domain;

Table 3: Axes that a migration result must not collapse.

Axis	Values	Question answered
coverage	total; partial with domain; unsupported	Is a semantic action defined for every required declaration/input?
result determinacy	deterministic; policy-selected; solution set	Does the package specify one result?
information	lossy; lossless under named checked criteria; unknown	What do the named conservative checks establish?
reversibility	invertible; partial inverse; noninvertible; unproved	Is a semantic inverse established?
writeability	read-only; partially writable; fully writable; reconciliation-dependent	Can updates propagate safely?
execution	semantic-only; virtual; lazy; eager; online mixed-version; unsupported	Which physical realization is supported?

3. mapping paths typed;
4. mapping attributes typed;
5. source equations preserved;
6. input instance valid;
7. output instance valid;
8. loss analysis complete for the declared check set;
9. synthesis policy total;
10. identity policy authorized;
11. target constraints satisfied;
12. compatibility-view result equivalent;
13. physical-plan capability supported;
14. mixed-version write protocol safe; and
15. rollback boundary recorded.

Success at one layer does not discharge another. A lawful mapping does not validate an input instance. A result-equivalent view does not establish writeability. A correct eager copy does not establish mixed-version safety.

5 Mapping direction and the restricted delta contract

5.1 Finite-total profile

Let a finite schema S have finitely many objects, generating arrows, attributes to scalar types, and path equations. All arrows and attributes in profile v1 denote total functions. A valid finite instance $I : S \to \mathbf{FinSet}$ assigns a finite set to each object, a total function to each arrow, a total scalar-valued function to each attribute, and satisfies every schema equation extensionally.

A lawful total mapping $F : S \to T$ maps every source object to a target object, every source arrow to a well-typed target path, and every source attribute to a well-typed target path followed by a target attribute. It preserves source equations in the accepted target equality fragment.

Definition 5.1 (Restricted pullback)**.** For a lawful total mapping $F : S \to T$ and valid finite target instance J, the pullback instance $\Delta_F J$ is defined by

$$(\Delta_F J)(A) = J(F(A)),$$
$$(\Delta_F J)(f) = J(F(f)),$$
$$(\Delta_F J)(a) = \mathsf{eval}_J(F(a)).$$

Its direction is

$$\Delta_F : T\text{-}\mathsf{Inst} \longrightarrow S\text{-}\mathsf{Inst}.$$

This variance matters operationally. If S is an old schema, T a new schema, and $F : S \to T$, then $\Delta_F J$ can provide an old-shaped read over a new target instance J. Calling it a forward construction from old data to new data reverses the operator.

5.2 Validity preservation

Proposition 5.2 (Finite pullback validity, inherited as MIG-01)**.** *Let $F : S \to T$ be a lawful total mapping in the finite profile, and let $J : T \to \mathbf{FinSet}$ be a valid finite target instance. Then $\Delta_F J$ is a valid finite S-instance.*

Proof. For each source object A, the definition assigns the finite set $J(F(A))$. For each source arrow $f : A \to B$, lawfulness gives a typed target path $F(f) : F(A) \to F(B)$. Because J interprets every target generator as a total function, composing the finitely many functions along that path yields a total function $J(F(A)) \to J(F(B))$. For each source attribute $a : A \to \tau$, lawfulness gives a typed target path followed by a target attribute of scalar type τ; its interpretation is therefore a total τ-valued function on $J(F(A))$.

It remains to check equations. Let $p = q$ be any source path equation. Mapping lawfulness states that $F(p)$ and $F(q)$ are equal in the accepted target equality fragment. Since J is a valid target instance, it interprets those equal target paths as extensionally equal functions. By the definition of pullback path interpretation, $(\Delta_F J)(p) = J(F(p)) = J(F(q)) = (\Delta_F J)(q)$. Thus every source equation holds. All required components are finite and total, so $\Delta_F J$ is a valid finite S-instance. □

Evidence status. PROVED IN P1 DOSSIER. The proof is complete for the stated mathematical profile. It is not a Lean declaration and does not verify Rust..

5.3 Identity and contravariant composition

Proposition 5.3 (Identity and composition, inherited as MIG-02)**.** *For every valid finite S-instance I, $\Delta_{1_S} I$ is extensionally equal to I. For lawful total mappings $F : S \to T$ and $G : T \to U$, and every valid finite U-instance K,*

$$\Delta_{G \circ F} K \quad \textit{is extensionally equal to} \quad \Delta_F(\Delta_G K).$$

Proof. For identity, 1_S maps each object to itself, each arrow to its one-edge path, and each attribute to itself. Hence the definitions of $\Delta_{1_S} I$ and I agree on every object set, generating-arrow function, and attribute function. They are extensionally equal.

For composition, fix a source object A. The left side assigns $K((G \circ F)(A)) = K(G(F(A)))$. The right side assigns $(\Delta_G K)(F(A)) = K(G(F(A)))$. Now fix a source arrow f. Mapping composition substitutes G's target paths into $F(f)$. Interpreting the substituted path in K is, by associativity of ordinary function composition, the same function as first interpreting G to form $\Delta_G K$ and then interpreting $F(f)$. The same substitution argument holds for a source attribute's path followed by its terminal attribute. Thus the two instances agree on all objects, arrows, and attributes, which is extensional equality. □

Evidence status. PROVED IN P1 DOSSIER. No direct-versus-staged physical migration equivalence is inferred from this semantic law..

5.4 Lawfulness is not losslessness

Mapping lawfulness checks typed total images and equation preservation. It does not imply injectivity or surjectivity on declarations or data, an inverse, preservation of keys/cardinalities outside the profile, a canonical Σ, safe update propagation, physical recovery, or acceptable cost. Two source objects may map to one target object while the mapping remains total. Such an identification is exactly why information loss needs a separate witness language.

6 Running example: three immutable revisions

6.1 S_0: old customer-shaped reads

This example explains the contract; it is not an executed fixture. Revision S_0, with ID `schema:customer-order:v0` and an illustrative semantic body hash h_0, contains objects `Customer` and `Order`; a total arrow

$$\mathsf{placedBy} : \mathsf{Order} \to \mathsf{Customer};$$

and required attributes

$$\begin{array}{ll} \mathsf{customerId} : \mathsf{Customer} \to \mathsf{UUID}, & \mathsf{regionCode} : \mathsf{Customer} \to \mathsf{String}, \\ \mathsf{orderId} : \mathsf{Order} \to \mathsf{UUID}, & \mathsf{cents} : \mathsf{Order} \to \mathsf{Int64}. \end{array}$$

The finite profile treats all four attributes as total functions. Although some names suggest identifiers, the profile does not represent SQL null, optionality, or a key constraint.

6.2 S_1: stable identity and a field-to-path mapping

Revision S_1, ID `schema:customer-order:v1` with hash h_1, keeps the stable semantic identities of `Customer`, `Order`, `placedBy`, and the three unaffected attributes. It introduces object `Region`, arrow $\mathsf{home} : \mathsf{Customer} \to \mathsf{Region}$, and attribute $\mathsf{code} : \mathsf{Region} \to \mathsf{String}$. The old `regionCode` declaration is represented through the typed expression $\mathsf{home}; \mathsf{code}$.

The lawful mapping

$$F_{01} : S_0 \longrightarrow S_1$$

maps the two stable objects to themselves, maps `placedBy` to itself, maps unaffected attributes to themselves, and maps

$$F_{01}(\mathsf{regionCode}) = \mathsf{home}; \mathsf{code}.$$

For a valid S_1-instance J_1, $\Delta_{F_{01}} J_1$ is an S_0-shaped read. In particular,

$$(\Delta_{F_{01}} J_1)(\mathsf{regionCode})(c) = J_1(\mathsf{code})(J_1(\mathsf{home})(c)).$$

This is a mathematical compatibility-read expression in the admitted fragment. No CatDB view compiler has generated SQL for it, and the paper does not present a physical plan.

Stable identity distinguishes this field change from a name-based remove/add. It does not prove that an independently declared `Region` object has the intended business meaning. The old-shaped read is not automatically writeable either. Changing an old region string would require a policy for choosing or creating a `Region` row and reconciling shared references.

6.3 S_2: a stronger value-domain obligation

Revision S_2, ID `schema:customer-order:v2` with hash h_2, asks that order amounts belong to a stronger domain, such as the values accepted by an explicit nonnegative, bounded monetary conversion. A complete package would have to state the source and target scalar domains, the totality domain, failure behavior, collision/injectivity checks, precision and overflow rules, round-trip status, and a backend lowering.

Profile v1 has no arbitrary scalar expression language or refinement constraint for this request. The proposed $S_1 \to S_2$ transition is OBSTRUCTED at the semantic-definition layer. Writing PostgreSQL `USING` syntax would not prove totality. Existing rows could violate the new domain, and enforcement for new writes is distinct from validation or repair of historical rows.

6.4 Rejected split and guarded merge

A later request proposes splitting each `Customer` into `Person` and `Account`. The current object map sends one source object to one target object; it cannot produce two correlated target objects. The split therefore needs a richer construction language, a total discriminator or creation rule, fresh or derived identifiers, and a relationship rule. It is UNSUPPORTED in profile v1. A missing discriminator is a falsifying counterexample to any claim of a total split.

Mapping two source objects, for example `Customer` and `Guest`, to one target `Party` is syntactically possible. The mapping alone does not decide whether equal email addresses

denote one person, how collisions are handled, or which source is authoritative. A conservative `OBJECT_IDENTIFICATION` witness would classify the mapping as `lossy` unless a separately authorized tagged-union or reconciliation policy preserves identity. That witness is specified, not currently computed.

6.5 One package, several physical choices

For the valid $S_0 \to S_1$ compatibility read, a future planner could:

- eagerly populate a region relation before cutover;
- lazily create or resolve regions when customer rows are accessed;
- leave storage unchanged and expose an old-shaped virtual view;
- combine eager work for hot data with virtual access for cold data; or
- run old and new clients under a mixed-version protocol.

These modes differ in locks, scans, failure behavior, freshness, and write-routing. Each would need to be compared with the same semantic result before acceptance. The current artifact has no such physical plan, so the example stops at the semantic contract.

6.6 Impact set

Changing `regionCode` and the amount domain should produce a versioned impact set containing every registered dependent query, mapping, constraint, materialization, policy, and export. For example:

$$\begin{aligned}\mathsf{impact}(\mathcal{M}_{01}) = \{&\mathsf{query} : \mathsf{customerSummary},\ \mathsf{mapping} : \mathsf{crmExport},\\ &\mathsf{constraint} : \mathsf{orderAmountDomain},\ \mathsf{materialization} : \mathsf{regionalSales},\\ &\mathsf{policy} : \mathsf{regionAccess},\ \mathsf{export} : \mathsf{legacyCustomerCSV}\}.\end{aligned}$$

This is an illustrative set. Dependency-closure computation does not yet exist. Any omitted registered dependency would falsify the future completeness claim.

7 Migration taxonomy

The taxonomy in Tables 4 and 5 separates representation, semantic data, risk, obligations, physical choices, status, and falsifiers. "Represented" never means "implemented."

Table 4: Migration taxonomy, part I.

Class	v1 representation	Required semantic data	Loss or ambiguity	Validation obligations	Physical choices / status	Falsifying counterexample
Additive entity or relation	Entity can appear in target; new total arrow needs synthesis	Population rule or proof that no population is required	Missing target objects or relationship values	Total synthesis and target validity	Eager/lazy/virtual; ENGINEERING HYPOTHESIS	Existing row needs an arrow value but no rule supplies one
Add required attribute	Shape representable; historical construction is not automatic	Typed default, total expression, lookup, or rejection	Invalid default or missing historical value	Synthesis totality, type, target constraint	Backfill/default/view; OBSTRUCTEDbeyond simple expression	One old row lies outside the default/expression domain
Stable-ID rename	Yes when identity is preserved or mapping is explicit	Stable declaration ID and authorized rename	Low semantic-loss risk; possible external name dependency	ID uniqueness and dependency impact	Metadata, view, or physical rename; diff not computed	Classification changes under declaration permutation
Destructive drop	Not as an omitted generator in a strict total old-to-new map	Partial domain, archive/tombstone, projection direction, and authorized loss	Direct omission and dependency loss	Loss witness, authority, impact, rollback boundary	View omission or physical drop; obstructed or proposed	Dropped value is later required but neither archived nor witnessed
Field-to-path	Yes: target path followed by target attribute	Typed path and terminal scalar attribute	Shared target or write ambiguity	Endpoint/type checking, instance validity	Virtual/eager/lazy; semantic read proved on paper, not executed	Path endpoint or scalar type differs
Value-domain conversion	No arbitrary scalar transforms in v1	Domains, totality, failure, precision, collision, round trip	Parse failure, overflow, rounding, many-to-one collision	Counterexample search and target validation	Backend expression/backfill; OBSTRUCTED	A declared input causes error or two inputs collide against a claimed injection
Object split	No: one source object has one target-object image	Multiple target constructors, keys, discriminator, relationship rules	Ambiguous classification, missing key, duplicated/lost identity	Total construction and referential consistency	Eager/lazy/virtual; UNSUPPORTED	A source row has no discriminator or produces inconsistent correlated IDs

Table 5: Migration taxonomy, part II.

Class	v1 representation	Required semantic data	Loss or ambiguity	Validation obligations	Physical choices / status	Falsifying counterexample
Object merge	Schema-object identification is syntactically representable; data consolidation is not	Tagged union or reconciliation/quotient, representative, authority, collision policy	Identity collapse and conflicting attributes	Authorized identity policy and provenance	Future bounded Σ; currently UNSUPPORTED	Two distinct source entities collide without an authorized representative
Cardinality change	No optional or multivalued arrows/bounds in v1	Minimum, maximum, selection or construction rule	Dropped multiplicity, invented relationship, violating rows	Existing-data and new-write validation	Constraint, view, or backfill; obstructed	Two target rows survive a requested functional restriction
Constraint strengthening	Direct path equations only; general constraints outside v1	New law, validation, repair/rejection, enforcement order	Existing violations and rejected writes	Separate historical and future-write obligations	Staged validate/backfill; ENGINEERING HYPOTHESIS	One historical row violates the new law after reported success
Constraint weakening	Some direct-equation deletion structurally expressible	Explicit old-write compatibility policy	Old clients may assume a law no longer enforced	Legacy read/write contract	View/trigger or client cutoff; ENGINEERING HYPOTHESIS	Old write accepted although it breaks a required old invariant
Identity-rule change	Names/attributes exist; key meaning not represented	Identity and reference-rewrite policy with authority	Merge/split identities, stale references, provenance discontinuity	Global reference and provenance checks	Reconciliation plus physical update; OBSTRUCTED	Two old IDs become one without resolving foreign references
Enum add/rename	Scalar enum semantics not in v1; stable identity can state rename intent	Open/closed policy and value map	Rename ambiguity or unhandled consumers	Exhaustiveness and compatibility	Metadata/converter; OBSTRUCTED	A closed-world old consumer receives a new value
Enum merge/split/delete	Not in v1	Total value classifier or map, loss and authority policy	Many-to-one loss, contextual ambiguity, deletion	Domain-totality and collision witnesses	Conversion, backfill, or view; OBSTRUCTED	One old value has two context-dependent targets with no context rule

8 Loss, partiality, ambiguity, synthesis, and reversibility

8.1 Conservative witness discipline

Information loss is not the negation of mapping validity. A future analyzer returns one of:

$$\mathsf{lossy}(W), \qquad \mathsf{lossless\_under\_checked\_criteria}(C), \qquad \mathsf{unknown}(C, U),$$

where W is a nonempty set of concrete witnesses, C names completed checks, and U names inconclusive checks. A concrete witness dominates an unknown finding. Empty findings justify only the qualified `lossless_under_checked_criteria`, never universal losslessness.

Candidate syntactic witnesses include object identification, arrow collapse, attribute omission, and declared scalar collision. Their soundness goals must be proved separately. Completeness is not assumed. A reported witness must reference declarations that exist in the exact endpoint revisions.

8.2 Partiality

A partial migration has a machine-readable input domain. It is not encoded as a total function that throws an undocumented exception. For a conversion, the domain may be a predicate over scalar values; for a structural operation, it may require a discriminator or a populated relationship. The result must record every rejected input or a replayable counterexample and must not call a partial operation total because the observed sample happened to fit.

8.3 Synthesis and fresh identity

Adding required values or constructing target objects needs a synthesis policy. The policy states whether a value is a literal default, a typed total expression, a lookup into a versioned artifact, a fresh identifier, or a human decision. It also states determinism and authority. Fresh identifiers need a naming rule that yields compatible results under eager, lazy, and virtual realizations. Otherwise physical mode changes meaning.

8.4 Ambiguity and authority

An ambiguity record contains alternatives, evidence, the selecting authority, and unresolved cases. Similar column names are candidate evidence, not an authorized rename. Equal email strings are not an entity-resolution theorem. A migration package may be deterministic because an authorized policy selects one alternative; that does not make the choice logically necessary.

8.5 Reversibility and rollback

Semantic reversibility asks whether an inverse relation or partial inverse is established under stated laws. Physical rollback asks whether the deployed system retains enough old

representation and log/history to restore a checkpoint. These are independent. A lossy semantic mapping may be physically rolled back while an archive is retained. An invertible read mapping can still have an unsafe online cutover or update policy.

9 Compatibility views and historical queries

For $F : S_{\text{old}} \to S_{\text{new}}$ and valid new instance J, $\Delta_F J$ is the candidate old-shaped result. A future compiled view is accepted only if

$$\mathsf{eval}_{\text{backend}}(\mathsf{compileView}(F), J) \approx \Delta_F J$$

under a declared snapshot, bag/set model, null policy, scalar semantics, collation, timezone, error behavior, and ordering boundary.

This equation is a test contract, not a current result. It does not imply the view is updateable, efficient, natively constrained, or stable under later schema revision. PostgreSQL automatically updates only restricted classes of views [17]; relational lens results likewise impose conditions [3]. Every P3 compatibility view is therefore read-only until a separate writeability result exists.

A historical query needs more than a schema version. Reproducibility requires the schema revision, mapping revision, source snapshot or operation interval, identity/reconciliation revision, execution-plan revision, capability manifest, materialization/checkpoint, scalar/null semantics, and provenance policy. A branch alias is not enough because it can move.

Legacy write support is harder than legacy reading. A source-shaped update may have several target realizations or may need to create target-only objects. The package must classify it as read-only, partially writable, fully writable, or reconciliation-dependent. No inference from read equivalence to write safety is allowed.

10 Eager, lazy, virtual, hybrid, and mixed-version execution

10.1 Physical compilation boundary

For one migration package, a future compiler may compute

$$\mathsf{compile}(\mathcal{M}, C, W, P) \longrightarrow \mathsf{PhysicalPlan},$$

where C is a backend capability manifest, W workload and deployment context, and P physical policy. The plan references the immutable migration package and states which result-equivalence contract it is expected to preserve. Changing backend or indexes may change the plan; changing the accepted semantic result requires a new package.

10.2 Eager

Eager mode copies or rewrites all affected physical data before cutover. A plan must expose scans, table/index rewrites, lock levels, backfill order, validation queries, write freeze or dual-write requirements, cutover transaction, retained old representation, and rollback boundary.

Eager is not automatically safe: a complete rewrite faithfully executing a wrong mapping is still wrong.

10.3 Lazy

Lazy mode exposes the new version before every row is physically converted. Its plan must specify version-marker grain, idempotence or exactly-once behavior, concurrency control, conversion failures, mixed converted and unconverted query semantics, index/constraint visibility, background completion, crash recovery, and old-writer behavior. BullFrog is the direct systems baseline [2]; event upcasting is a narrower application-layer analogue [10].

10.4 Virtual

Virtual mode leaves a base representation in place and presents another schema through a view or query rewrite. A plan must include the view definition, result equivalence, push-down/materialization boundary, null/multiplicity/scalar policy, permissions and row security, dependency invalidation, and writeability classification. A virtual plan can avoid a rewrite while increasing query cost or restricting writes. "No rewrite" is not "no migration cost."

10.5 Hybrid

A hybrid may eagerly convert an indexed hot region, lazily convert cold rows, and virtualize rare historical access. Every region needs a recorded state and transition rule. Hybrid execution is an optimizer choice only if all regions preserve the same declared result.

10.6 Online mixed-version protocol

Online execution with old and new clients is a protocol envelope rather than just another storage choice. It needs intermediate states, bounded version skew, write routing, index/constraint maintenance, safe transitions, and recovery. F1 shows the corruption risk of unsafe asynchronous change [18]. No online-safety claim follows from a typed mapping or a generated SQL plan. P3 currently has no such protocol.

11 Dependency impact and provenance

11.1 Stable-ID diff

A semantic diff compares immutable revisions by stable declaration ID and canonical body. It may report addition, removal, stable-ID rename, retype, path/equation change, or metadata-only change. Input permutation must not change the result. A name similarity heuristic may suggest a rename in an exploratory tool, but the confirmatory artifact requires stable identity or explicit authorization.

A diff is not a migration plan. It does not establish a converter, loss status, dependency closure, cutover, or rollback. Alembic's documented non-inference of renames provides a concrete baseline for testing whether stable IDs improve the workflow [23].

11.2 Registered dependency closure

Impact analysis traverses typed edges from changed declarations to registered queries, mappings, constraints, materializations, policies, exports, and physical plans. The future completeness claim is relative to this registered graph; unregistered application code is outside the proof boundary and must be reported as a threat.

The impact result is immutable and deterministic. Each edge records the source artifact version and dependency kind. Reviewers must be able to trace why an item appears. A stale cache or omitted edge that hides one registered dependent artifact falsifies completeness.

11.3 Provenance

A migration result should retain:

- endpoint schema and package hashes;
- input snapshot or operation interval;
- mapping, synthesis, identity, ambiguity, and authority revisions;
- validator, compiler, and backend versions;
- capability manifest and physical-plan hash;
- discharged and open obligations;
- loss witnesses and inconclusive checks;
- generated and human-edited artifacts, distinguished;
- execution observations, failures, retries, and exclusions; and
- materialization/checkpoint and rollback boundary.

Provenance does not make a bad decision correct. It makes the decision and its evidence inspectable and historical results reproducible.

12 Formal properties and proof obligations

12.1 What the two propositions establish

Propositions 5.2 and 5.3 establish validity preservation, identity, and contravariant composition for the finite-total pullback model. They are reused from the P1 dossier with complete written proofs. They do not establish instance-validator implementation, information-loss analysis, Σ, Π, physical equivalence, updateability, or online safety.

12.2 Loss witness goals

For each admitted witness class w, a future soundness proposition must have the form:

$$\mathsf{reports}(w, S, T, R) \Longrightarrow \mathsf{namedInformationDifference}(w, S, T, R).$$

The right side needs a precise semantic definition. Object identification, arrow collapse, attribute omission, and scalar collision need different definitions; a generic "loss" bit is insufficient. Completeness is a separate and likely stronger question. Until these definitions and proofs exist, the classifier is an ENGINEERING HYPOTHESIS.

12.3 Composition obligations

A semantic composite may be formed only when endpoints match, both relations belong to a closed admitted fragment, coverage domains compose, and synthesis, identity, and authority policies compose or are reapproved. Loss witnesses and ambiguity must accumulate rather than disappear. The P1 pullback composition proof supplies one semantic law; it does not prove that physical plan composition equals compilation of the semantic composite.

Open conjecture (direct versus staged execution). Within a future deterministic fragment with fixed scalar, null, identity, and snapshot semantics, direct execution of an accepted composed package and staged execution of its factors should be extensionally equivalent while the staged result accumulates the same obligations, loss, and ambiguity. Status: OPEN CONJECTURE. One counterexample rejects or narrows the fragment.

12.4 Obstructed definitions

The following remain OBSTRUCTED: a canonical representative for bounded Σ; any useful restricted Π; null/missing/error/partial-value semantics; keys and cardinality beyond total arrows; arbitrary scalar expressions; object construction and split; equality beyond the direct fragment; and a theorem connecting semantic and physical-plan composition.

13 Fixture and implementation plan

13.1 Existing envelopes

The existing cases remain normative future tests:

- `migration-delta-pullback-ok`;
- `migration-delta-schema-mismatch-error`;
- `migration-sigma-unsupported`;
- `migration-pi-unsupported`;
- `information-loss-object-identification-ok`;
- `information-loss-incomplete-mapping-error`;
- `schema-diff-entity-added-ok`; and
- `schema-diff-duplicate-before-error`.

Today, all eight are STRUCTURAL-ONLY. The expected value is comparison data, not an implementation. Rust and Haskell must compute each outcome independently.

13.2 Current-fragment additions

The first semantic slice should add path-attribute pullback, multi-arrow path pullback, invalid-instance totality, equation violation, identity and composition projections, arrow-collapse loss, attribute-omission loss, inconclusive required-target analysis, no-witness qualified status, mixed witness precedence, stable-ID retype, equation addition, permutation-stable diff, and a rename-not-inferred case.

Diagnostics must be stable and specific. Examples include:

- `INSTANCE_FUNCTION_NOT_TOTAL`;
- `INSTANCE_EQUATION_VIOLATION`;
- `ARROW_COLLAPSE`;
- `ATTRIBUTE_OMISSION`; and
- `ANALYSIS_INCONCLUSIVE`.

The implementation must not read expected outputs to produce observations.

13.3 Future-profile cases

A new fixture-schema version is required for required-field synthesis, authorized destructive drop, explicit rename, partial/colliding conversion, object split, merge identity policy, tagged union, cardinality strengthening, staged validation, compatibility-view cardinality, eager/virtual equivalence, lazy retry, and unsafe online version skew. These features cannot be smuggled into profile v1 through paper-only conventions.

13.4 Generative properties

Property tests should check pullback validity under preconditions; identity and composition by extensional equality; witness references; qualified losslessness only when no finding exists; deterministic diff under permutation; application of the admitted diff algebra; accumulation of loss and ambiguity; rejection outside conversion domains; physical-mode result agreement; and replayable shrinking with seed and fixture hash.

14 Experimental evaluation contract

Correctness precedes performance. A fast migration with the wrong result is a failure, not a tradeoff. Table 6 preregisters the experiment families and rejection conditions.

Table 6: Evaluation matrix.

ID	Hypothesis	Baseline/input	Primary measures	Reject or narrow when
E1 fixture conformance	Validators compute deterministic typed outcomes	Independent Haskell model; shared fixtures	Exact outcomes, diagnostics, disagreement count	Any unexplained disagreement or nondeterminism
E2 delta laws	Implementation matches finite P1 laws	Generated lawful small models	Validity and extensional identity/composition	One accepted counterexample
E3 loss classifier	Concrete witnesses are sound and unknown is not promoted	Hand-audited taxonomy cases	Precision by witness, unknown rate	A false witness or unqualified lossless result
E4 diff/impact	Stable-ID diff and closure find supported changes/dependencies	SQL/ORM diff; registered D5 graph	Recall, false positives, actionable set	One registered omission or unusable false-positive load
E5 compatibility view	Compiled view equals semantic result	Hand-written native view; fixed snapshot	Normalized rows, multiplicity, scalar behavior, plan	Any declared semantic mismatch
E6 eager plan	Physical result equals semantic target	Expert SQL and ORM on SQLite and PostgreSQL	Result, constraints, locks, movement, recovery	Incorrect result or hidden destructive effect
E7 lazy plan	Completion equals eager result	BullFrog-inspired baseline	Throughput, latency, completion, replay/recovery	Divergence, duplicate/lost conversion, unbounded failure
E8 virtual versus materialized	Modes preserve one semantic result	Hand-written view or materialization	Result, freshness, maintenance, cost	Semantic divergence
E9 mixed-version	Protocol protects concurrent writes and constraints	F1-style state protocol	Anomalies, availability, skew boundary	One admitted corrupting history
E10 composition	Direct and staged results/obligations agree	Staged execution	Extensional result and accumulated warnings	Mismatch or erased warning
E11 seeded-fault review	Typed package improves pre-execution review	Ordered SQL and mainstream ORM	Faults found, false positives, time, rollback, confidence	No net benefit after training/tool overhead

ID	Hypothesis	Baseline/input	Primary measures	Reject or narrow when
E12 backend cost	Semantic compilation exposes physical work	Native SQLite and PostgreSQL	Scans, bytes, locks, downtime, CPU, elapsed time	A claimed cheap/online plan is materially worse or unsafe

14.1 D5 and D5-H

The D5 corpus must include additive field, stable-ID rename, object split and merge, cardinality and constraint change, value-domain conversion, mapping revision, source deletion, reconciliation reversal, and provenance-policy revision. D5-H gives reviewers the same target state, backend, and recovery requirements under ordered SQL, one mainstream ORM, and CatDB.

Seeded faults include silent truncation, invalid default, lost identity, broken foreign-key meaning, ambiguous split, incomplete merge, stronger-constraint violations, and a compatibility view that changes cardinality. Reviewability is supported only if the controlled study finds improved fault detection or impact correctness after training and tool cost are counted. One study would not establish universal migration safety.

14.2 Evidence retention

Every run retains environment identity, source and fixture hashes, tool versions, random seeds, raw observations, repetitions, exclusions, and failed runs. Backend runs record native plans, locks, scans, bytes moved, transaction boundaries, and recovery outcomes. Rust and Haskell observations are produced independently before comparison.

15 Results and current artifact status

15.1 What exists

This author pass produces a LaTeX manuscript, a primary-source BibTeX file, related-work and novelty ledgers, a disputed-claim ledger, a primary-source bibliography, a reproducibility record, a review contract, and a Humanizer scope marked not run. The manuscript contains the package specification, taxonomy, illustrative revision chain, two complete finite-total paper proofs, and future experiment contract.

Accepted P1 prerequisite evidence establishes bounded schema/path/mapping computation in Rust and Haskell and selected path/mapping declarations in Lean. Those artifacts may reduce ambiguity about the relation used by a future migration engine. They are not P3 migration evidence.

15.2 What does not exist

Semantic migration execution remains absent. Specifically:

- no valid or rejected Δ request has been computed by a P3 migration executor;
- no Σ or Π implementation exists;
- no information-loss witness has been computed;
- no schema diff or impact closure has been computed;
- no compatibility view or physical migration plan has been generated;

- no eager, lazy, virtual, hybrid, or mixed-version engine exists;
- no SQLite or PostgreSQL migration has been run;
- no result-equivalence, recovery, performance, or human-review experiment has been run; and
- the Humanizer gate has passed, but no terminal AGY paper review has been run.

The four migration, two loss, and two diff envelopes are STRUCTURAL-ONLY. Their round trips are not migration, loss-analysis, or diff results. This section therefore reports an artifact-status result, not a database-migration result.

15.3 Current verdict

The central systems thesis remains ENGINEERING HYPOTHESIS. Direct-versus-staged execution is OPEN CONJECTURE. General Σ, useful Π, arbitrary conversion, cardinality, split, and physical composition are OBSTRUCTED or UNSUPPORTED as stated earlier. The manuscript is provisional.

16 Threats to validity and limitations

16.1 Model limitations

Profile v1 includes finite objects, total arrows, required total attributes, and a restricted equation checker. It does not model SQL nulls, missing/error values, optional relationships, one-to-many relations, cardinality bounds, keys, general scalar expressions, aggregation, target construction, or backend bags. Results inside the profile do not extend to those features.

The categorical pullback model is cleaner than production SQL. A useful notion of result equivalence still needs explicit semantics for bags and sets, duplicates, nulls, collations, timezones, decimals, overflow, and errors. Until those semantics are fixed, cross-backend equivalence is obstructed.

16.2 Analysis limitations

Conservative syntactic checks may miss semantic loss, and a qualified no-witness result is not an invertibility proof. Stable IDs can also be misused. Preserving an identifier while changing its meaning defeats the intended rename signal. Dependency closure can be complete only over registered artifacts, so dynamic SQL and external code may remain invisible.

Questions about identity and ambiguity often require domain authority. Category theory does not decide whether two customers are the same person or which source wins a conflict. Recording a human decision as a versioned policy improves the audit trail, but it does not prove that the policy is correct.

16.3 Systems limitations

There is no physical compiler or backend result. PostgreSQL and SQLite differ in DDL, locking, constraints, conversions, null behavior, and transactions. Backend documentation cannot substitute for CatDB runs. Evidence for online safety would need concurrency histories and protocol tests, not just a generated plan.

Lazy conversion can shift cost into unpredictable reads and complicate indexes, constraints, and recovery. Virtual access may change performance and permissions, while eager conversion can lock or rewrite data. Hybrid modes add more states to analyze. The semantic package records these concerns; it does not solve them.

16.4 Evidence limitations

The paper proofs are not machine checks. Agreement between the P1 Rust and Haskell implementations is bounded computational evidence, not verification of P3. P1 Lean declarations cover only path and mapping laws. Fixture expectations are not observations. No benchmark or human study supports a claim about reviewability, cost, availability, or safety.

16.5 External-validity limitations

The proposed D5 corpus and two initial backends cannot represent every database, data distribution, review practice, or outage constraint. Participant expertise and learning may affect a controlled review study. A positive result would support a bounded workflow claim, not universal safety. A negative result must be retained and may require a narrower package.

17 Open questions

1. What smallest schema extension represents optionality, keys, and cardinality without destabilizing the finite-total profile?
2. Should destructive change use a partial-map type, an explicit discard object, or a separate construction language?
3. What quotient and representative policy makes bounded Σ deterministic and useful?
4. Which scalar expression language is portable, totality-checkable, and capable of returning counterexamples?
5. How should synthesized objects and fresh identifiers be named so eager, lazy, and virtual modes agree?
6. Which equation fragment balances sound preservation checking with useful expressiveness?
7. How should keys, foreign keys, cardinalities, and policies accumulate under composition?
8. When may loss, synthesis, identity, or authority decisions compose without reapproval?

9. What normalized result model covers SQL bags, nulls, collations, timezones, decimals, and errors?
10. Which online protocols can be generated from semantic metadata, and which require backend-specific coordination?
11. Can registered dependency closure be complete and still precise enough to act on?
12. After comparison with PRISM++, CQL, InVerDa, BullFrog, and current migration tools, what empirical CatDB delta remains?

18 Conclusion

A schema change needs a meaning before it needs a deployment strategy. The proposed migration package records exact revisions, a typed relation, direction, coverage, determinacy, conservative loss evidence, synthesis, ambiguity, authority, obligations, provenance, and execution intent. Eager, lazy, virtual, hybrid, and mixed-version plans can then be treated as different realizations of one stated result, rather than as unrecorded changes of meaning.

The mathematical core is bounded. Pullback has the contravariant direction $\Delta_F : T\text{-}\mathsf{Inst} \to S\text{-}\mathsf{Inst}$; under finite-total assumptions it preserves validity and satisfies identity and composition. Those complete paper arguments are inherited P1 results. They do not show that migrations are lossless, invertible, writable, cheap, recoverable, or safe under concurrent versions.

The systems case has not yet been demonstrated. Current P3 envelopes are STRUCTURAL-ONLY, and there is no migration execution, loss detection, diff, impact analysis, physical compilation, backend run, benchmark, or human study. The candidate contribution must survive direct comparison with the substantial prior art and the falsifiers defined here. Until then, "schema migration as a typed mapping" is a research program, not a completed migration system.

References

[1] 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.

[2] Souvik Bhattacherjee, Gang Liao, Michael Hicks, and Daniel J. Abadi. BullFrog: Online schema evolution via lazy evaluation. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, 2021. doi: 10.1145/3448016.3452842.

[3] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: A language for updatable views. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems*, pages 338–347, 2006. doi: 10.1145/1142351.1142399.

[4] Kristopher S. Brown, David I. Spivak, and Ryan Wisnesky. Categorical data integration for computational science. *Computational Materials Science*, 164:127–132, 2019. doi: 10.1016/j.commatsci.2019.04.002.

[5] CategoricalData project. Categorical query language (CQL) documentation and tutorial. https://categoricaldata.net/CQL/, 2026. Official project documentation; accessed 22 July 2026.

[6] Confluent. Schema evolution and compatibility. https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html, 2026. Official documentation; accessed 22 July 2026.

[7] 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.

[8] 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.

[9] 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.

[10] EventSourcingDB. Versioning events. https://docs.eventsourcingdb.io/best-practices/versioning-events/, 2026. Official documentation; accessed 22 July 2026.

[11] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005. doi: 10.1016/j.tcs.2004.10.033.

[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.

[13] 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 SIGMOD International Conference on Management of Data*, 2017. doi: 10.1145/3035918.3064046.

[14] 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.

[15] 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.

[16] PostgreSQL Global Development Group. PostgreSQL 18 documentation: ALTER TABLE. https://www.postgresql.org/docs/current/sql-altertable.html, 2026. Official documentation; accessed 22 July 2026.

[17] PostgreSQL Global Development Group. PostgreSQL 18 documentation: CREATE VIEW. https://www.postgresql.org/docs/current/sql-createview.html, 2026. Official documentation; accessed 22 July 2026.

[18] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek. Online, asynchronous schema change in F1. *Proceedings of the VLDB Endowment*, 6(11):1045–1056, 2013. doi: 10.14778/2536222.2536230.

[19] Redgate. Flyway documentation: Versioned migrations. https://documentation.red-gate.com/fd/versioned-migrations-273973333.html, 2026. Official documentation; accessed 22 July 2026.

[20] 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. 2017. doi: 10.1007/978-3-319-72044-9_5.

[21] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012. doi: 10.1016/j.ic.2012.05.001.

[22] David I. Spivak and Ryan Wisnesky. Relational foundations for functorial data migration. In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 21–28, 2015. doi: 10.1145/2815072.2815075.

[23] SQLAlchemy project. Alembic documentation: Auto generating migrations. https://alembic.sqlalchemy.org/en/latest/autogenerate.html, 2026. Version 1.18.5 official documentation; accessed 22 July 2026.

[24] Ryan Wisnesky, David I. Spivak, Patrick Schultz, and Eswaran Subrahmanian. Functorial data migration: From theory to practice, 2015.