Incremental Semantic Materialization: Maintaining Integrated Views under Schema and Data Change

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

23 July 2026

Abstract

An integrated view can change even when no new source row arrives. A source schema can change, a source-to-domain mapping can gain a new conversion, an identity decision can be reversed, or a provenance policy can require evidence that an old materialization did not retain. Conventional incremental view maintenance provides mature techniques for data deltas. Those techniques alone do not decide whether old state still has the same meaning after a semantic change.

We give a provisional semantics and systems contract for incremental materialization in CatDB. Its authoritative object is an immutable materialization definition that binds schema, mapping, query, reconciliation, provenance, scalar-semantics, coverage, and consistency revisions. A physical realization also records a plan, per-source frontiers, auxiliary state, a dependency graph, and a checkpoint. For a deliberately small positive bag algebra, we state the established first-order delta laws. We then state the stronger CatDB obligation: normalized data and provenance after incremental maintenance must equal independent full recomputation at the same semantic definition and declared source frontier. Some data changes may admit deltas. Schema, mapping, reconciliation, provenance-policy, and capability changes instead pass through a typed dependency and invalidation decision. Reuse is permitted only by a named adaptation rule; otherwise the exact treatment is rebuild or rejection.

This version contributes a falsifiable design, not an implemented result. Classical incremental maintenance, self-maintainable views, lazy refresh, view adaptation, DBToaster, differential dataflow, DBSP, Noria, change-data capture, transactional streaming, distributed snapshots, and provenance semirings are established prior art. The current repository has no `catdb-materialization` crate, full or incremental evaluator, P6 fixture family, source-change connector, crash failpoint harness, Haskell oracle, Lean P6 declaration, or benchmark run. All CatDB equivalence, cursor safety, recovery, provenance, and performance claims therefore remain OPEN or OBSTRUCTED. In

particular, this paper reports no exactly-once guarantee, no transactional cursor result, no crash-recovery result, no formal proof of a CatDB maintenance algorithm, and no performance result.

Contents

1 Introduction

1.1 A corrected order is the easy case

Consider two source systems that contribute orders to a canonical `CustomerRevenue` view. The PostgreSQL source reports amounts in cents. A periodic CSV snapshot stores decimal text. Versioned mappings convert both representations, while a reconciliation policy connects source-local customer identifiers to canonical customers. The view groups accepted orders by canonical identity.

A correction changes one PostgreSQL order from 2,500 to 3,000 cents. A local delta is straightforward: subtract the old contribution and add the new one. That rule no longer settles the matter if the correction is replayed after a crash, the mapping changes from gross to net revenue, the reconciliation policy splits a customer that had been merged, or the provenance profile begins requiring the origin of a newly used refund field. Each event changes the maintenance problem differently. A row-level merge statement cannot tell the system which old state remains valid.

There is also no single obvious meaning of "now." PostgreSQL may be applied through one log sequence number while the CSV source remains at `snapshot-8`. Together they form a vector of observations, not evidence for a simultaneous distributed snapshot. A display that says only "refreshed at 14:03" hides whether a source is missing, a log gap exists, a new definition is rebuilding, or the data is current while required provenance is not.

This leaves a narrow question:

> How should a mapped view respond to data, schema, mapping, identity, provenance-policy, coverage, and execution-capability changes without letting its physical cache become the authoritative definition of meaning?

1.2 Thesis and boundary

The candidate thesis is:

> For an explicitly supported operator fragment, a versioned semantic definition, a declared source-capture profile, and a recoverable checkpoint protocol, CatDB can compile a materialization strategy that updates affected state after admitted

data changes and conservatively invalidates or rebuilds state after semantic changes. Correctness requires agreement with independent full recomputation for normalized data and provenance at the same declared source frontier.

This thesis is an ENGINEERING HYPOTHESIS, not evidence of an implementation. Incremental view maintenance is established and covers a broad range of cases [11–13]. Higher-order and automatically derived deltas, partially ordered incrementality, and shared partial state are also established [1, 4, 9, 18]. The possible CatDB delta lies in binding those execution techniques to typed, immutable semantic revisions and using explicit dependencies to decide when a data-delta plan remains applicable.

1.3 Research questions

The paper organizes the program around five questions:

1. Which finite query and change fragment admits an independent full-recomputation oracle and exact delta maintenance?
2. How should dependency information connect source schemas, mappings, queries, reconciliation, provenance, coverage, capabilities, and physical state?
3. Which source, sink, cursor, and checkpoint contracts prevent a published position from outrunning durable effects?
4. Which semantic changes admit validated adaptation, and which must invalidate, rebuild, or block?
5. Under which workload regions, if any, does incremental maintenance reduce work or latency after auxiliary state, provenance, recovery, and source-history costs are included?

1.4 Contributions of this author pass

This manuscript contributes:

1. an immutable materialization-definition model that separates denotation from a replaceable physical realization;
2. a source-change envelope, per-source frontier model, and explicit status vocabulary;
3. a bounded positive bag-algebra fragment with stated delta and auxiliary-state obligations;
4. a typed dependency graph and a conservative decision lattice for data and semantic changes;
5. a cursor/checkpoint safety condition, a crash matrix, and snapshot-to-stream cutover contract;
6. a multidimensional freshness envelope;
7. a prior-art and novelty boundary; and

8. implementation, formalization, and experimental plans with named falsifiers.

These are specification artifacts. No item is presented as a working P6 system result.

1.5 Exact nonclaims

CatDB does not currently claim that it eliminates ETL, data movement, full recomputation, staleness, source coordination, or distributed-systems tradeoffs. It does not claim to incrementalize every query or arbitrary schema and mapping change. It does not infer missing before-images or treat independent source positions as a global snapshot. It does not claim generic exactly-once delivery or an atomic transaction across unrelated sources and sinks. It does not claim that materialization is free, that provenance is free, that retained state always admits reuse, or that category theory proves an execution protocol correct.

2 Evidence vocabulary and prerequisite boundary

2.1 Labels

Table 1 fixes how claims are read. A result supported in a different system remains prior literature, not a CatDB result. A decoded record is not evidence that its intended state transition works.

Table 1: Evidence labels used by this manuscript.

Label	Meaning
SUPPORTED BY PRIOR LITERATURE	A primary paper, standard, or official system document establishes the stated result or capability within its own assumptions.
ENGINEERING HYPOTHESIS	A proposed CatDB behavior that still needs implementation and experiment.
OPEN CONJECTURE	A mathematical statement without a completed proof for the declared fragment.
OPEN	Required definitions, code, fixtures, experiments, or review are incomplete.
OBSTRUCTED	A prerequisite semantic boundary is missing, so a claim cannot yet be implemented or tested honestly.

No P6 claim in this manuscript is labelled proved, machine-checked, computational, or experimentally supported. DBSP's mechanized core does not verify a future CatDB compiler [4]. PostgreSQL logical decoding does not make a future CatDB checkpoint atomic [22]. Equality of materialized rows would not by itself establish equality of provenance.

2.2 Blocking paper contracts

P6 depends on four other CatDB papers:

- P2 must define logical-query denotation, normalization, capabilities, and the semantic-to-physical plan boundary.
- P4 must define read-direction mapping semantics, coverage, partiality, information loss, conversion, and composition.
- P5 must define source observations, connector capabilities, source coverage, snapshot policy, partial availability, and result reconciliation.
- P13 must define normalized data provenance, retention, completeness, and equality for full and incremental paths.

Those contracts are not frozen. Consequently, data and provenance equivalence, safe mapping adaptation, and strategy interchangeability are OBSTRUCTED. A P6 implementation can begin around isolated source and checkpoint types, but it cannot close the central claim until these dependencies are explicit.

2.3 Current repository audit

The repository contains typed schema, path, mapping, and a small migration slice across Rust, Haskell, Lean, and shared fixtures. Those are prerequisites only. At the time of this author pass, the repository does not contain:

- a `catdb-materialization` crate;
- a logical query evaluator or P6 full-recomputation oracle;
- a change envelope, dependency extractor, delta compiler, invalidator, or auxiliary aggregate store;
- PostgreSQL logical-slot or SQLite session-extension integration;
- checkpoint publication, applied-effect state, or process-kill failpoints;
- initial snapshot and change-stream cutover;
- freshness responses or a materialization API;
- incremental provenance semantics;
- P6 Haskell modules or Lean declarations;
- P6 fixtures, raw runs, or benchmarks; or
- terminal P6 paper review.

The current system gate `SYS-MAT-01` and claim `CR-063` therefore remain OPEN and unresolved respectively.

3 Semantic model

3.1 Authoritative definition

Definition 3.1 (Materialization definition)**.** *A materialization definition is the immutable tuple*

$$\mathcal{D} = (S, M, Q, R, P, \Lambda, A),$$

where S is a vector of source-schema revisions, M is a versioned read-mapping bundle, Q is a typed canonical query, R is a reconciliation and authority policy, P is a provenance and retention profile, Λ fixes scalar and error semantics, and A fixes source coverage, availability, and consistency policy.

The definition identifier is a deterministic content digest:

$$\mathsf{DefId} = H(S, M, Q, R, P, \Lambda, A).$$

The future repository must specify the serialization and hash domain. The equation is a design requirement, not evidence that content addressing is implemented.

Connector capability manifest C and physical plan L are versioned execution inputs. They may change how $\mathcal{D}$ is evaluated. They may not silently change what Q means. Every result binds both the semantic definition and the execution revisions used to compute it.

3.2 Source observation and full semantics

Definition 3.2 (Source frontier)**.** *A source frontier f_s identifies one stream, a source-specific position, its completeness class, and the time at which* CatDB *observed it. A multi-source frontier is the vector $f = \{s \mapsto f_s\}$.*

Positions can be PostgreSQL log sequence numbers, Kafka partition offsets, SQLite change-set identities, API delta tokens, file-snapshot digests, or other declared values. Comparison is valid only within the same source, stream, and ordering profile. The vector does not imply a distributed transaction or simultaneous snapshot. Distributed snapshot algorithms require explicit channel and process assumptions [6].

For definition $\mathcal{D}$ and the actual input observation I_f, the authoritative reference is

$$\mathsf{Full}(\mathcal{D}, I_f) = \mathsf{Normalize}\left(\mathsf{DataEval}(\mathcal{D}, I_f), \mathsf{ProvEval}(\mathcal{D}, I_f)\right).$$

Normalization must fix bag or set semantics, duplicates, null and missing values, decimals, time zones, collation, errors, row identity, source-token identity, provenance normalization, completeness, and any relevant ordering. The independent full evaluator may not call the same delta compiler as the incremental path.

3.3 Replaceable realization

Definition 3.3 (Materialization realization)**.** *A physical realization is*

$$\mathcal{X} = (\mathsf{DefId}, \mathsf{PlanId}, f, V, U, G, K, \sigma),$$

where V is materialized output, U is auxiliary incremental state, G is a dependency-graph revision, K is the latest durable checkpoint, and σ is the status.

The realization may be deleted, rebuilt, or replaced if retained sources and the immutable definition permit reproduction. Generated SQL, task logs, indexes, cached batches, and physical tables are execution artifacts. If a manual patch adds semantic logic only to generated SQL, the realization has left the managed correctness boundary.

3.4 Incremental correctness obligation

Let J be a candidate incremental strategy, X its old state, and Δ an admitted change. The post-change definition and frontier are $\mathcal{D}'$ and f'. Exact incremental mode requires:

$$\mathsf{Normalize}(\mathsf{Observe}(J(X, \Delta))) = \mathsf{Full}(\mathcal{D}', I_{f'}).$$

The equality includes normalized data, provenance, definition identity, frontier, completeness, and status.

Obligation 3.4 (Data equivalence)**.** *Every admitted incremental history must produce the same normalized bag result as independent full recomputation at the same definition and frontier.*

Obligation 3.5 (Provenance equivalence)**.** *Every admitted incremental history must produce the same normalized P13 provenance as full recomputation. Matching row values alone is insufficient.*

Obligation 3.6 (Strategy separability)**.** *Federation, full recomputation, eager delta maintenance, lazy maintenance, partition rebuild, and validated reuse may implement one semantic definition only where their normalized result envelopes agree.*

All three obligations are OBSTRUCTED. There is no current evaluator, provenance profile, fixture corpus, or cross-strategy comparison.

3.5 Status vocabulary

Table 2: Candidate realization statuses.

Status	Meaning
`building`	Initial snapshot or full recomputation is incomplete.
`fresh`	All required declared frontiers are applied under the active definition.
`stale`	Known source or semantic changes are not applied.
`partial`	Only a declared subset of required sources or evidence is present.
`invalid`	The realization cannot answer under its bound definition.
`rebuilding`	A replacement realization is being computed.
`recovering`	Replay or checkpoint repair is in progress.
`blocked`	Missing history, capability, schema, or policy prevents repair.
`unknown`	Progress or completeness cannot be established.

"Fresh" is relative to a definition and frontier. It does not mean merely that a worker ran recently.

4 Change model

4.1 Change envelopes

An admitted source event needs more structure than an operation name and row body. The candidate record is:

```
ChangeEnvelopeV1 {
  source_id
  source_schema_revision
  capture_profile_revision
  stream_or_snapshot_id
  source_transaction_id?
  source_position
  event_ordinal?
  event_id?
  operation
  relation_id
  key_before?
  key_after?
  row_before?
  row_after?
  source_commit_time?
  observed_time
  provenance_token
}
```

Optional fields determine which computations are possible. A delete without an old key cannot reliably retract a keyed join contribution. An update that changes a key is not equivalent to an update of a non-key attribute. An event identifier can support deduplication, but it does not make the output update, auxiliary state, provenance, and cursor publication atomic.

The initial operation vocabulary includes snapshot boundaries and reads, insert, delete, update, truncate, source transaction boundaries, heartbeat, schema marker, and gap. A heartbeat establishes liveness only within its connector profile. A gap states that required incremental history is no longer available. Exact mode then stops and reboots from a new declared snapshot or returns a blocked state.

4.2 Capture profile

Each connector supplies an immutable capture profile:

```
CaptureProfileV1 {
  source
  position_kind
  ordering
  delivery
  transaction_boundaries
  row_images
  ddl_capture
  gap_detection
}
```

The conservative v1 profile is at-least-once delivery, complete source transaction order, permitted duplicates, no transaction fragmentation, and fatal gaps. Out-of-order and late events require an explicit buffer, repair, rejection, or logical-time policy. Arrival order is not a semantic ordering rule.

PostgreSQL makes old-row availability depend on replica identity and can replay recent changes after a crash because slot position persistence is checkpointed [22, 23]. SQLite changesets preserve keyed old and new values under documented conditions, while patchsets retain less information [25]. The ordinary SQLite update hook omits important cases and has no suitable general CDC contract [26]. These facts justify connector-specific profiles. They do not establish that CatDB has implemented either profile.

4.3 Data changes and semantic changes

Table 3 separates data events from changes that alter the view's meaning or the evidence required to justify it.

Table 3: Candidate change classification and conservative treatment.

Class	Example	Default treatment	Condition for reuse
data insert	new order	delta	admitted operator and complete keyed event
data delete	removed order	delta or rebuild	old key and every required old value available
data update	amount or key changes	old retraction plus new insertion, or declared direct delta	before/after and key semantics fixed
truncate	relation cleared	bulk invalidation or explicit truncate delta	exact source coverage and dependency scope
duplicate	replayed transaction	deduplicate or idempotently reapply	stable effect identity and one sink transaction
out of order	earlier source position arrives late	reorder, repair, reject, or use an admitted time domain	source ordering profile supports the choice
source schema	field add, drop, type, key, or equation change	revalidate, usually invalidate	dependency proof and conversion rule
mapping	gross-to-net conversion	invalidate affected dependency slice	named adaptation rule and full oracle
query	new filter or grouping	adapt or rebuild	retained state is sufficient and equality passes
reconciliation	identity merge or split	affected-identity recomputation	complete reverse dependency index
provenance policy	require field origin	provenance rebuild, possibly data rebuild	retained evidence derives the stricter profile
coverage or authority	source becomes optional or nonauthoritative	new definition and invalidation	exact subtraction of old contribution
capability	before-images disappear or pushdown changes	stop/replan/reboot	logical semantics and in-flight boundary preserved

Class	Example	Default treatment	Condition for reuse
runtime	new delta compiler	shadow verify before promotion	deterministic fixture and oracle equivalence

The size of a text diff is not a safe impact estimate. A one-character unit change can alter every output value. Conversely, an added nullable source field may be irrelevant if the capture, wildcard, provenance, mapping, and query dependencies all exclude it.

5 A bounded differential fragment

5.1 Signed bags

Classical IVM already handles inserts, deletes, updates, duplicates, joins, aggregation, and broad SQL or Datalog fragments [11–13]. This paper chooses a smaller initial profile so that a separate full evaluator and cross-language fixtures remain feasible.

Represent a bag relation by a finitely supported integer-valued function:

$$R : \text{Tuple} \to \mathbb{Z}.$$

A signed change ΔR updates it by

$$R' = R + \Delta R.$$

Intermediate deltas may contain negative multiplicities. Every published database state must have nonnegative final multiplicities. A negative published count is a diagnostic for an invalid history, a source gap, or an implementation defect.

For query Q, state I, and admitted change ΔI, the established data-level target is

$$Q(I + \Delta I) = Q(I) + \Delta Q(I, \Delta I).$$

This identity is SUPPORTED BY PRIOR LITERATURE for classical incremental maintenance. It is not a proof that a future CatDB compiler derives the right ΔQ.

5.2 Selection, projection, and union

For deterministic predicate p,

$$\Delta(\sigma_p R) = \sigma_p(\Delta R).$$

The predicate is excluded if it depends on a volatile clock, unversioned model, remote mutable lookup, locale not fixed by Λ, or opaque code.

For bag projection,

$$\Delta(\pi_A R) = \pi_A(\Delta R).$$

Projection must preserve multiplicity. A set-valued result instead needs support counts so that removing one derivation does not remove an output still supported by another derivation.

For bag union,

$$\Delta(R \uplus S) = \Delta R \uplus \Delta S.$$

Set union again needs counts of alternative support. Counting is established practice, not a CatDB novelty [11, 13].

5.3 Inner equijoin

For a simultaneous batch against a declared pre-state convention:

$$\Delta(R \bowtie S) = (\Delta R \bowtie S) \uplus (R \bowtie \Delta S) \uplus (\Delta R \bowtie \Delta S).$$

The cross term matters when both sides change in the same logical batch. Applying source events one at a time without a fixed pre-state, post-state, or staging convention can omit or double-count that term.

At the provenance level, a positive-algebra join multiplies joint derivations and union adds alternatives in semiring-style models [10]. Retractions and canonical provenance normalization need the exact P13 profile. The preceding data identity therefore does not close the provenance obligation.

5.4 Initial aggregate boundary

Table 4: Auxiliary state and retraction concerns for candidate aggregates.

Aggregate	Auxiliary state	Retraction concern
`COUNT`	count per group	subtract multiplicity and remove empty group under the declared zero policy
`SUM`	sum per group	subtract old contribution under a fixed numeric and overflow profile
`AVG`	sum and count	define zero-count result and maintain both components
`MIN/MAX`	ordered support multiset	deleting the extremum needs the next candidate
`COUNT DISTINCT`	support count per distinct value	state can approach or exceed the input projection

The proposed v1 admits `COUNT`, `SUM`, and `AVG`. It excludes `MIN`, `MAX`, and `COUNT DISTINCT` until memory, provenance, numeric, and deletion semantics are specified. An implementation may later support them, but this manuscript does not rely on that possibility.

5.5 Unsupported exact mode

The first exact profile rejects arbitrary difference and negation, outer joins, recursive queries, correlated subqueries, volatile functions, opaque user code, external side effects, approximate vector search, unstable top-k, floating-point rewrites without an equivalence profile, and any operator whose provenance delta is undefined.

Richer fragments are possible and already appear in the literature. DBToaster recursively materializes higher-order delta queries and uses costed materialization [1]. Differential dataflow maintains indexed differences across partially ordered versions [18]. DBSP provides algebraic automatic incrementalization for a rich language with mechanized core results [4]. A smaller CatDB fragment is an engineering choice that makes its own oracle and failure boundary auditable.

5.6 Auxiliary state is part of the cost

An incremental plan must declare every required base lookup, index, support count, aggregate component, retained input, provenance circuit, applied-effect record, and pending batch. Self-maintainable-view results show that avoiding base access depends on the view and retained information [14]. Incrementality may therefore trade update latency for memory, write amplification, source-log retention, and recovery work. Those resources belong in the evaluation, not in an implementation footnote.

6 Dependency, invalidation, and adaptation

6.1 Typed dependency graph

The candidate graph contains more than relations and columns. Its nodes include source and canonical schema revisions, fields, paths, capture profiles, mapping revisions and clauses, query revisions and logical operators, reconciliation and identity decisions, authority and coverage policies, provenance and retention profiles, capability manifests, physical plans, auxiliary state, checkpoints, result partitions, and both definition and realization identities.

Edges state roles such as:

```
reads types_against
maps_through depends_on_clause
uses_identity uses_authority
requires_coverage requires_provenance
requires_capability
compiled_to maintains
derived_from checkpointed_at
invalidates can_adapt_from
```

One untyped `depends-on` edge is too weak for selective invalidation. The system needs to distinguish whether a field contributes a displayed value, a filter, a join key, a group key, an identity rule, a coverage decision, provenance only, capture, or physical pushdown only. Dependencies must be extracted from typed IR. SQL text search cannot reliably find stable semantic identity, composed mapping clauses, or an opaque dependency. An opaque operator yields a wildcard edge or rejection.

6.2 Soundness before minimality

For change c and graph G, the first obligation is:

$$\mathsf{SemanticAffected}(c) \subseteq \mathsf{Impact}(c, G).$$

The impact set may initially over-approximate. Extra recomputation is a cost; a missed dependency is an incorrect result. Minimality is a later optimization objective:

$$\mathsf{Impact}(c, G) \approx \mathsf{SemanticAffected}(c).$$

Obligation 6.1 (Impact soundness)**.** *For the frozen finite IR, every output whose normalized data, provenance, status, or definition binding can change must be reachable from the change through the dependency graph.*

This obligation is an OPEN CONJECTURE for a future formal fragment and an ENGINEERING HYPOTHESIS for a runtime. It is OPEN because no P6 graph or invalidation algorithm exists. A single mutation-generated false negative would falsify it.

6.3 Decision lattice

A refresh decision selects one of:

```
retain
replan_only
delta_update
adapt_from_old_state
invalidate_partition
invalidate_all
rebootstrap_source
blocked
```

The record stores the old and new definition identities, change identity, affected graph slice, rule or proof identifier, needed source history, freshness and provenance effects, fallback, and a reviewer-readable rationale. Correctness filters candidate actions before a cost model ranks them.

6.4 Reuse obligation

Old state $X_{\mathcal{D}}$ may be transformed for a new definition $\mathcal{D}'$ only if a named transformer T satisfies:

$$\mathsf{Normalize}(\mathsf{Observe}(T(X_{\mathcal{D}}))) = \mathsf{Full}(\mathcal{D}', I_f).$$

Selected SQL view redefinitions can reuse old materializations and retained extra information [15]. That result does not authorize universal reuse across categorical mappings, identity splits, authority changes, or provenance profiles.

When no validated rule applies, exact mode marks the realization stale or invalid, computes a replacement under $\mathcal{D}'$, checks it against the oracle, atomically publishes the new reference, and retains or removes the old artifact according to policy. Full recomputation is a safe strategy, not a failure of the semantic model.

6.5 Partition recomputation

Partition-level repair needs a conservative affected-key function:

$$\mathsf{KeysWhoseOutputChanges}(c, \mathcal{D}) \subseteq \mathsf{AffectedKeys}(c, \mathcal{D}).$$

Candidate keys include canonical entity, tenant, source, time range, provenance domain, and workspace. Identity merges and splits can fan out across joins and aggregates, so a customer-local source update does not imply a customer-local semantic impact. If reverse reachability is incomplete, the correct treatment is a larger rebuild.

7 Checkpoint and recovery contract

7.1 Checkpoint state

A candidate checkpoint is:

```
MaterializationCheckpointV1 {
  checkpoint_id
  definition_id
  dependency_graph_id
  plan_id
  compiler_runtime_id
  parent_checkpoint_id?
  input_frontiers
  applied_batch_ids
  data_artifact_id
  auxiliary_state_artifact_ids
  provenance_artifact_id
  state_digest
  status
  staged_at
  committed_at?
}
```

It is valid only when output, auxiliary state, provenance, and applied source positions name one durable boundary. Async snapshots and stateful streaming systems provide important precedents [2, 5]. A future CatDB protocol must still state its own source and sink assumptions.

7.2 Cursor safety

For every published position p:

$$\mathsf{Published}(p) \Rightarrow \mathsf{Durable}(\mathsf{EffectsThrough}(p)).$$

The converse need not hold at every crash point. Durable effects may exist while the published cursor lags. Recovery can then detect a staged effect or replay idempotently.

The candidate external-source sequence is:

1. durably ingest the event or retain a replayable source position;
2. derive a stable batch and effect identity;
3. stage output, auxiliary, and provenance updates;
4. validate local invariants;
5. commit effects and the local applied-position record together;
6. publish the new realization reference;
7. acknowledge upstream under the connector contract; and

8. tolerate replay from the last acknowledged position.

This is a proposed protocol. No transactional cursor test has run. When source, sink, provenance, and cursor share one transactional database, one implementation may commit them atomically. Across PostgreSQL, Kafka, SQLite, files, and APIs there is no generic transaction. Kafka can atomically combine Kafka output writes and consumer offsets inside its own transaction model, but external sinks require participation or a separate protocol [3]. This paper makes no exactly-once claim.

7.3 Effect identity and replay

A candidate stable effect identity is:

$$\mathsf{EffectId} = H(\mathsf{DefId}, \mathsf{SourceId}, \mathsf{StreamId}, \mathsf{TxId}, \mathsf{Position}, \mathsf{Ordinal}).$$

At the observable checkpoint boundary, replay should satisfy:

$$\mathsf{Apply}(e, \mathsf{Apply}(e, X)) = \mathsf{Apply}(e, X).$$

Signed deltas themselves are additive and would normally double-apply. The runtime therefore needs an applied-effect guard in the same sink transaction as the effects it protects. The Debezium outbox pattern places source state and an event record in one source transaction, and its event identity helps deduplication; it does not solve downstream semantic maintenance [7].

7.4 Crash matrix

Table 5: Required crash observations and recovery behavior.

Failure point	Potential durable state	Required safe recovery
before ingest	no new local record	reread from source
after durable ingest	input log only	resume the batch
during staged delta	partial invisible stage	rollback or discard stage
after validation, before commit	complete invisible stage	commit once or discard and replay
between data and provenance updates	forbidden published state	use one transaction or invisible artifacts
after effects commit, before cursor publication	effects present, cursor lags	detect effect or replay harmlessly
after cursor publication, before source acknowledgement	complete local state	upstream replay is harmless
after acknowledgement, before durability	forbidden ordering	protocol must prevent it

Failure point	Potential durable state	Required safe recovery
during full rebuild	old realization visible	publish replacement only after validation
during definition change	old and new artifacts coexist	every read binds one immutable definition

The future failpoint harness must terminate the process, reopen durable stores, recover, and compare with full recomputation. Throwing and catching an exception in one process is not a durability test. No row in Table 5 is a reported CatDB result.

8 Bootstrap, source gaps, and freshness

8.1 Initial snapshot and stream cutover

Let p_0 be the source position associated with an initial snapshot. A correct bootstrap represents

$$I_{p_0} \quad \text{followed by} \quad \Delta I_{(p_0, p_1]}.$$

No change between snapshot and stream may disappear, and a row visible through both paths may not be counted twice.

A candidate protocol binds a replayable stream, records a cut position, reads a snapshot under a declared isolation mode, persists completion, replays committed changes after the cut, resolves collisions by key and position, and validates at a later frontier before publication. PostgreSQL's initial copy and ordered catch-up are a concrete precedent [21]. Debezium documents consistent snapshots, log-sequence streaming, offsets, transaction metadata, and duplicate cases [8]. Neither is evidence that CatDB has a working cutover.

PostgreSQL logical replication does not carry DDL as row replication [24]. A P6 connector therefore needs a separate schema-observation channel and a safe response to a mismatch between observed DDL and the capture stream. If WAL, offsets, delta tokens, or required snapshots expire, the materialization records the gap, stops exact incremental repair, and reboots or blocks. It must not advance a freshness marker across missing history.

8.2 Freshness envelope

The candidate response is:

```
FreshnessEnvelopeV1 {
  definition_id
  status
  required_sources
  per_source_observed_frontier
  per_source_applied_frontier
  source_commit_time_range?
  last_successful_checkpoint
  known_pending_batches
```

```
  known_gap?
  completeness
  measured_at
}
```

The envelope reports position lag where positions are comparable, source-time lag where source commit time exists, processing lag, definition lag, provenance lag, source completeness, and whether each quantity is known, bounded, partial, or unknown. A single wall-clock refresh time cannot replace these fields.

8.3 Consistency profiles

Table 6: Candidate materialization consistency profiles.

Profile	Promise
`local-atomic`	One local checkpoint exposes output, auxiliary state, provenance, and applied positions together.
`source-prefix`	Output reflects a complete prefix of each declared source stream.
`single-source-snapshot`	Output matches one source snapshot or position.
`vector-frontier`	Output names an independently observed position for each source.
`coordinated-snapshot`	Output is cross-source consistent only under an external protocol with recorded evidence.
`eventual-repair`	Known changes converge if sources and definitions eventually stop changing and retained history remains available.
`bounded-staleness`	A measured bound is enforced or its violation is surfaced.

`vector-frontier` is the realistic heterogeneous default. A read policy then decides whether stale, partial, invalid, or recovering states are rejected, returned with metadata, waited on, recomputed, federated, or answered from an earlier valid definition. A cost optimizer cannot override a strict consistency or completeness policy.

Stale-view cleaning studies incorrect, missing, and superfluous records and can provide approximate repair [17]. P6 exact mode must keep approximation separate from freshness and must never return an approximate or partial result as complete.

9 Semantic changes

9.1 Source schema changes

The invalidator compares typed old and new schema revisions. An unused nullable field may permit retention after capture, wildcard, mapping, query, identity, and provenance dependencies are checked. A used field, dropped field, nullability change, key change, type narrowing, table split, or path equation change generally requires revalidation and rebuild. A rename is reusable only when a stable semantic identity or explicit rename artifact establishes what remained the same.

A replica-identity change affects future delete and update evidence even if the query text is unchanged. The connector must stop at a safe checkpoint, reconfigure or reboot, and record the stream and profile change.

9.2 Mapping and query changes

A mapping edit can alter conversion, object construction, path meaning, enum normalization, identity keys, coverage, information loss, or provenance. Default treatment is reachability-based invalidation from the changed mapping clause. The paper does not claim that mapping composition always localizes repair.

Selected query changes may admit adaptation. An added filter may be computed from an old broader view. A projected column may be available in retained auxiliary state. A coarser grouping may be derivable from finer aggregates. A new union branch may be computed independently. Each rule has assumptions, retained-state requirements, and an oracle comparison. View-adaptation prior art supplies algorithms for selected SQL cases [15]; the CatDB program must identify rather than assume its applicable subset.

9.3 Identity and authority changes

An identity merge, split, reversal, contradiction, supersession, or authority change may move many facts between canonical groups. The dependency graph needs reverse indexes from reconciliation decisions to canonical entities, joined facts, aggregates, provenance evidence, and consumer materializations. If the index is incomplete, the system rebuilds the larger affected view. Vector similarity or a small textual edit does not bound this impact.

9.4 Provenance and retention changes

A weaker provenance policy may allow audited redaction or compaction. A stricter policy can reuse data only when retained raw evidence can derive and validate the required provenance. Without that evidence, old row values may still be usable while the exact provenance profile is partial or invalid. Changing the derivation semantics may require rebuilding both.

This distinction is why P6 requires data and provenance equivalence. Semiring provenance gives a compositional basis for positive algebra [10], but retention, redaction, source-token identity, definition revisions, and retractions still need the P13 profile.

9.5 Capabilities and physical replans

A capability change may remove predicate pushdown, supply transaction metadata, remove before-images, weaken ordering, or change numeric behavior. If logical equivalence remains established, a physical replan can retain output. If capture evidence becomes insufficient for future deltas, the system stops at a safe boundary and switches to full recomputation or rebootstrap.

An in-flight batch binds the capability and plan revisions under which it was compiled. Recovery may not reinterpret that batch under a new plan without validation. Physical plan caching is subordinate to semantic definition and checkpoint identity.

10 Materialization as an execution strategy

The intended compilation chain is:

immutable semantic definition
$\Downarrow$
logical maintenance specification
$\Downarrow$
physical refresh strategy
$\Downarrow$
replaceable materialized realization.

Every arrow records compiler, input, output, and revision identities. The last artifact is not the definition of the first.

These layers allow a planner to choose live federation, full local recomputation, eager or lazy delta maintenance, partition invalidation, partial or complete materialization, cached execution, or validated state reuse. Lazy view maintenance has established task-log, row-version, on-demand catch-up, ordering, and recovery precedents [27]. Oracle's materialized-view documentation likewise illustrates mature refresh and freshness choices [20]. CatDB does not invent those choices.

Any strategy that copies or materializes data still moves data. Generated ETL still needs credentials, scheduling, retries, monitoring, storage, and remediation. The proposed change is narrower: integration meaning stays in versioned mappings and semantic definitions, while the transfer program becomes a generated, inspectable realization.

Materialization selection is also established as a cost and dependency problem, including aggregate-view lattices [16]. P14 will own a broader cost model. P6 supplies correctness-filtered alternatives and the measurements needed to price update, checkpoint, source, state, and provenance costs.

11 Related work and novelty boundary

11.1 Classical incremental view maintenance

Gupta, Mumick, and Subrahmanian describe incremental algorithms for inserts, deletes, and updates over SQL and Datalog features, including counting treatments for set and duplicate semantics [13]. Gupta and Mumick organize materialized-view maintenance by view class, available resources, modification type, and applicability [12]. Griffin and Libkin provide an algebraic treatment of duplicates [11]. These sources establish that deltas, bag changes, counts, and broad operator support are prior art.

Self-maintainable views characterize when selected integration views can be maintained from old state and change information without source access [14]. Materialized-view adaptation shows that selected SQL redefinitions can reuse old state [15]. P6 therefore cannot presume either self-maintainability or adaptation. It must derive required state and reject reuse outside a named rule.

Lazy maintenance defers work while retaining ordered tasks and row versions [27]. Stale-view cleaning studies repair and approximation [17]. The CatDB terminology distinguishes eager, lazy, stale, partial, approximate, invalid, and blocked states so that execution policy does not silently weaken semantics.

11.2 Compiled and differential systems

DBToaster derives higher-order deltas, recursively materializes delta queries, and uses a costed choice between materialization and lazy evaluation [1]. Differential dataflow retains indexed differences over partially ordered logical times and supports iterative computations [18]. Naiad provides the timely dataflow substrate and low-latency iterative execution precedent [19]. DBSP gives a general algebraic incrementalization framework, a compiler/runtime direction, and a machine-checked theoretical core [4]. Noria studies shared partial state, on-demand reconstruction, and online dataflow-graph changes [9].

A fair P6 evaluation must compare equivalent fragments, disclose unsupported features, record compile time and retained state, and avoid attributing an established delta method to CatDB. DBSP is the strongest direct baseline for generic automatic IVM. Differential dataflow is the logical-time and retained-trace baseline. DBToaster is the higher-order compiler baseline. Noria is the partial-state and graph-change baseline.

The proposed P6 question is different: when does a data-delta strategy remain valid after a versioned schema, mapping, reconciliation, provenance, coverage, or capability change? A differential timestamp does not by itself answer that question, and adding a version label would not constitute a contribution without a validated dependency and recovery property.

11.3 Snapshots, streams, and capture

Distributed snapshots require explicit process and channel assumptions [6]. Lightweight asynchronous snapshots apply barrier techniques to stateful dataflows [5]. MillWheel combines persistent per-key state, logical time, low watermarks, and framework-scoped fault tolerance [2]. Kafka transactions can atomically couple Kafka offsets and Kafka outputs under their documented model [3]. These results reject any vague claim that a cursor, checkpoint, or "exactly once" label is new or universal.

PostgreSQL logical decoding, replica identity, initial copy, and replication restrictions define one source profile [21–24]. SQLite sessions and update-hook limits define different embedded boundaries [25, 26]. Debezium documents a practical snapshot and streaming connector plus outbox support [7, 8]. P6 must implement and test its own integration with these mechanisms.

11.4 Plausible CatDB contribution

The publication-safe candidate is the combination of:

1. one immutable definition binding schema, mapping, query, reconciliation, provenance, scalar, coverage, and consistency revisions;

2. one typed dependency model spanning semantic definitions, source observations, capabilities, plans, auxiliary state, checkpoints, and realizations;
3. a change classifier that separates admitted data deltas from semantic invalidation and adaptation;
4. normalized full-versus-incremental equality for both data and provenance;
5. a cursor/effect protocol verified by process-kill recovery; and
6. a freshness envelope that exposes heterogeneous frontiers, completeness, definition lag, and provenance lag.

Every item is currently OPEN or OBSTRUCTED. The combination is only plausibly novel. If it reduces to conventional IVM with version columns, if the graph misses dependencies, or if no validated cross-layer rule exists, the novelty thesis fails.

12 Implementation roadmap

12.1 Profile freeze

The first gate is joint ownership, not code generation. P2, P4, P5, and P13 must freeze:

- the finite logical query and normalization profile;
- read-direction mapping and conversion semantics;
- source observation, coverage, snapshot, and failure semantics; and
- normalized provenance, retention, and completeness semantics.

The resulting schemas cover materialization definition, capture profile, change envelope, checkpoint, and freshness envelope. Deterministic serialization, normative examples, rejection cases, and terminology review are required before an implementation can claim conformance.

12.2 Hand-worked fixtures and Haskell oracle

The first fixture contains two keyed relations, one read mapping, one inner equijoin, one grouped sum, one positive provenance circuit, insert/update/ delete/duplicate events, one mapping change that forces rebuild, one source frontier, and one crash after effects but before cursor publication. Two independent human derivations must agree before either program uses the expected output.

An independent Haskell oracle then evaluates full semantics, interprets the small delta fragment, normalizes bags and P13 provenance, classifies invalidation, and emits JSON observations. It must not share Rust implementation logic or read golden expected values as computed results. Haskell agreement would be bounded computational evidence, not a proof.

12.3 Rust runtime boundary

The proposed Rust crate owns definition references, change and frontier handling, dependency and invalidation mechanics, maintenance state, checkpoint/recovery, freshness, and realization lifecycle. It does not own schema truth, mapping truth, query denotation, connector capability truth, identity authority, or provenance meaning.

An initial module split is:

```
definition.rs change.rs frontier.rs
dependency.rs delta.rs state.rs
invalidation.rs checkpoint.rs recovery.rs
freshness.rs error.rs
```

SQLite is the first local durable store for checkpoints, applied-effect IDs, output, auxiliary state, and provenance references. The session extension can be evaluated as a local source profile. PostgreSQL work adds publication and replica-identity preflight, slot lifecycle, snapshot/stream cutover, LSN and transaction metadata, schema polling, gap detection, backpressure, and connector-specific recovery. Dropping or advancing a source slot is a consequential operation and must be logged.

12.4 Recovery and observability

Named failpoints terminate the process before ingest, after ingest, during state and provenance staging, before and after sink commit, around cursor publication, around upstream acknowledgement, during rebuild, and during old artifact cleanup. Recovery reopens every store and compares with the independent oracle.

The API exposes definition and realization identities, status, per-source frontiers, pending batches, last checkpoint, invalidation reason, selected strategy, recovery history, data and provenance completeness, and wait, rebuild, or federated-fallback operations. A later MDReader view can inspect these fields, but UI display is not a substitute for the underlying invariants.

12.5 Formalization roadmap

Lean work begins with finite bags and signed changes, followed by selection, projection, union, join, and basic aggregate delta laws. It should identify and reuse the precise theoretical delta already available from DBSP where appropriate rather than restating existing formal work as a CatDB result.

The high-value new targets are:

1. impact soundness for one frozen finite semantic IR;
2. checkpoint-state safety, $\Box(\mathsf{cursorPublished}(p) \Rightarrow \mathsf{effectsDurable}(p))$;
3. provenance equivalence for one P13 positive-algebra profile; and
4. selected adaptation laws, such as an unused-field addition, a rename with stable semantic identity, a stronger selection over retained rows, a coarser grouping over finer state, and an equivalent physical replan.

No such P6 declaration exists. These are OPEN CONJECTURE or OBSTRUCTED targets, not formal results.

13 Experimental and reproducibility contract

13.1 Evaluation order

Correctness precedes performance. For each generated history, the harness:

1. creates a source state and records the initial frontier;
2. runs independent full recomputation;
3. initializes full, invalidate-on-read, eager-delta, lazy, and selected external baseline treatments;
4. applies the same source transactions and semantic changes;
5. checkpoints after every batch;
6. kills selected runs at declared failpoints;
7. recovers and completes;
8. independently recomputes at comparison frontiers;
9. normalizes data, provenance, definitions, and status;
10. compares exact values, multiplicities, tokens, and envelopes; and
11. preserves the first mismatch and a minimized reproducer.

Any correctness mismatch stops performance interpretation for that treatment. Failed histories and raw store snapshots remain part of the artifact.

13.2 Baselines

Every external baseline needs an exact revision, build, configuration, capability statement, and validation fixture. A published number from another machine is context, not a co-measured result. Unsupported features are reported rather than silently translated.

13.3 Correctness and change workloads

Data histories vary cardinality, selectivity, join fanout, duplicates, groups, batch size, insert/delete/update mix, key changes, source count, skew, duplicate and out-of-order rates, and retained-history window. Semantic mutations add, drop, rename, and change fields; alter keys and equations; edit mapping clauses; change filters, projections, groups, and joins; merge and split identities; change authority or coverage; tighten provenance and retention; remove a connector capability; and change consistency policy.

Table 7: Preregistered baseline roles.

Baseline	Role
full recomputation	correctness oracle and reference cost
invalidate on read	conservative cache treatment
classical first-order IVM	simplest delta compiler baseline
DBToaster	higher-order delta and costed materialization baseline
differential dataflow	partially ordered incremental computation baseline
DBSP	automatic rich-query incrementalization baseline
Noria lineage	partial-state and reuse baseline where reproducibly buildable
PostgreSQL treatment	native backend reference
lazy task log	update/read latency and staleness tradeoff
CatDB candidate	dependency-aware strategy under the same admitted semantics

For invalidation, the oracle computes the true affected output. The experiment records predicted nodes, false negatives, false positives, retained and rebuilt state, decision time, and time to validated freshness. One false negative fails impact soundness.

13.4 Crash and bootstrap workloads

Each crash run records source positions, sends an actual termination signal, reopens durable state, inspects checkpoint and applied-effect records, resumes the source, and compares with full recomputation. It counts replayed and deduplicated events and verifies that no published cursor leads durable effects. Filesystem, database durability, isolation, and acknowledgement settings are part of the result.

Bootstrap tests run concurrent writes around the snapshot cut. They vary source isolation, snapshot chunk count, row-key collisions, restarts, DDL during copy, log retention, and source failover. The final bag must have no missing or duplicate multiplicity, and every observed and applied frontier is retained.

13.5 Freshness and recovery truthfulness

The harness injects source delay, source outage, connector pause, backlog, failed checkpoint, definition change, provenance rebuild, lost history, and partial availability. It checks that status matches ground truth and strict reads do not consume inadmissible state. A stale or partial response returned as exact fresh output is a correctness failure.

13.6 Performance measurements

Only correct treatments enter performance analysis. Measurements include:

- compilation, delta-plan generation, and dependency-impact time;
- refresh CPU, wall time, and p50/p95/p99 update-to-visible latency;

- source rows and bytes read, output rows and bytes changed;
- auxiliary, provenance, deduplication, and retained-history bytes;
- checkpoint size and duration, recovery time, and full-recompute time;
- source load, stale duration, and lazy read delay; and
- cost-estimate error and crossover curves by update fraction and batch size.

The central performance claim succeeds only if a declared workload region shows lower work or latency without unacceptable storage, source, provenance, or recovery cost. It fails if no such region exists. No measurements are reported in this paper.

13.7 Artifact retention

Each run records the Git commit and dirty-tree digest, operating system, hardware, storage, filesystem, Rust/Haskell/Lean/database/connector versions, topology, durability and isolation settings, random seeds, fixture and source digests, definition and plan IDs, exact commands, timestamps, raw output, crash signal and failpoint, correctness verdict, and exclusion reason.

Confirmatory workloads and metrics are preregistered. Runs repeat after warm-up and report distributions and intervals, not means alone. Timeouts, out-of-memory runs, and correctness failures remain visible. Exact and approximate treatments are never pooled.

14 Claim register and present results

14.1 Claim anchors

Table 8: P6 claim anchors and current evidence state.

ID	Claim	Required witness	Falsifier	Status
P6-C01	incremental data equals normalized full recomputation	independent oracle, fixtures, randomized histories, backend comparison	one admitted mismatch	OBSTRUCTED
P6-C02	incremental provenance equals P13-normalized full provenance	provenance oracle and adversarial derivations	missing, extra, stale, or mis-versioned contributor	OBSTRUCTED
P6-C03	published cursor never leads durable data, state, and provenance	state model and process-kill matrix	one crash skips an unapplied change	OPEN
P6-C04	replaying a committed batch leaves normalized checkpoint unchanged	duplicate/crash histories per connector	duplicate row, count, token, or effect	OPEN
P6-C05	dependency impact contains every semantically affected output	formal fragment, mutations, wildcard behavior	one missed output	OPEN
P6-C06	named semantic changes can adapt old state exactly	rules, oracle checks, cost comparison	one admitted adapted mismatch	OBSTRUCTED
P6-C07	users can distinguish freshness and failure states	API, fixtures, failure tests, UI review	stale or partial returned as exact	OPEN
P6-C08	execution strategies preserve one immutable definition	cross-strategy data/provenance equality	strategy changes denotation	OBSTRUCTED
P6-C09	incremental mode benefits a declared workload region	preregistered baselines, raw runs, crossover curves	no measured beneficial region	OPEN
P6-C10	realization is disposable and reproducible	delete/rebuild exercise with equal output and provenance	logic exists only in mutable execution state	OPEN

14.2 What this paper establishes

This author pass establishes only artifact facts. It provides a LaTeX manuscript, source bibliography, paper-local ledgers, candidate definitions, the supported-fragment boundary, falsification conditions, implementation and formalization roadmaps, and an experimental contract. Its algebraic delta identities and cited source behaviors are SUPPORTED BY PRIOR LITERATURE.

14.3 What remains absent

No P6 source event has been applied by CatDB. No full and incremental outputs have been compared. No provenance circuit has been maintained. No dependency-impact set has been computed. No materialization has been invalidated or adapted. No cursor has been published by a P6 runtime. No process-kill recovery, initial cutover, source-gap repair, freshness response, formal check, baseline run, or performance measurement exists.

The paper therefore reports no runtime, crash-safety, formal, provenance, or performance result. The overall P6 state is OBSTRUCTED because central semantic prerequisites and executable evidence are missing.

15 Limitations and threats to validity

15.1 Semantic limitations

The proposed operator fragment omits negation, outer join, recursion, volatility, approximate search, and several common aggregates. It assumes a normalization profile that has not yet been frozen. SQL nulls, missing values, collation, time zones, decimal overflow, floating point, and error behavior can break equivalence if left implicit.

The dependency model is presently prose. Dynamic SQL, unregistered source logic, opaque functions, external models, and human decisions can escape a graph unless the system rejects them or uses conservative wildcard edges. Impact minimality may be unattainable for practical policies; sound over-approximation remains the first requirement.

15.2 Connector and recovery limitations

At-least-once capture, old row images, transaction boundaries, DDL observation, order, gap detection, and retention differ by source. A protocol valid for one PostgreSQL/sink topology may not apply to Kafka, SQLite, files, or APIs. A local atomic checkpoint does not imply upstream acknowledgement or downstream side effects are atomic.

Process-kill tests cover chosen failpoints and durability settings. They do not prove behavior under every hardware, kernel, filesystem, database, network, or operator failure. Liveness needs assumptions about source availability, retry, retention, and eventual quiescence.

15.3 Provenance limitations

Positive semiring provenance is not a complete operational provenance model. Retractions, duplicate derivations, identity revisions, plan history, checkpoint recovery, redaction, retention, and trust policy all need explicit normalization. Data may remain correct after evidence needed for a stricter profile has expired. The system must report that separation rather than reconstruct an explanation it cannot justify.

15.4 Experimental limitations

A finite fixture corpus and two initial backends do not represent all workloads, schemas, update distributions, organizational policies, or failure modes. Property-based tests can expose counterexamples but do not prove an unbounded implementation. A positive workload region does not imply that incremental maintenance is generally faster. A negative result may show that full recomputation is the better strategy for the measured region.

External baseline builds can differ in language coverage, tuning, state retention, and durability. Comparisons must disclose those differences. A translation that weakens a baseline or silently excludes unsupported input is not fair.

15.5 Novelty limitations

Almost every component has direct prior art. The candidate contribution is a cross-layer contract. Combination alone is weak novelty. The program needs evidence that typed semantic versions change invalidation correctness, recovery accountability, or planner behavior in a way not provided by a conventional IVM system plus ordinary metadata.

16 Open questions

1. What is the smallest P2 logical language that supports useful joins and aggregates plus an independent evaluator?
2. Which P4 mapping clauses expose exact reverse dependencies and affected keys?
3. Should v1 admit only one totally ordered cursor per source?
4. How should cross-source transactions be represented when no global transaction exists?
5. Which PostgreSQL logical-decoding plugin and version should be pinned?
6. Is the SQLite session extension sufficient, or does embedded capture need an application-owned outbox?
7. How should truncate, key changes, partition movement, and source failover appear in a portable change envelope?
8. How long must source history be retained for lazy repair and recovery?
9. Which aggregates enter v1 without unbounded auxiliary state?

10. Can the P13 representation retract provenance support without exponential expansion?
11. Which evidence survives retention and redaction, and how is incompleteness reported?
12. Which schema and mapping changes deserve proved adaptation rules?
13. How are versioned opaque functions and learned models invalidated?
14. Can a capability-only replan preserve an in-flight checkpoint?
15. How can materializations be shared across workspaces with different identity, authority, or provenance policies?
16. Which read policy applies while a new definition rebuilds?
17. How is bounded staleness expressed when a source exposes no commit timestamp?
18. What prevents a slow consumer from retaining unbounded source logs?
19. How is stream reset distinguished from duplicate or failover replay?
20. What measured benefit justifies auxiliary and provenance state?

17 Conclusion

In CatDB, incremental materialization is one physical realization of an immutable semantic definition. Admitted data changes may use established delta techniques. Before old state can be reused, changes to schemas, mappings, reconciliation, provenance, coverage, or capabilities need a typed dependency decision. Exact mode compares normalized data and provenance with independent full recomputation at the same declared source frontier. If the evidence is insufficient, the system must invalidate, rebuild, block, or label the result.

This remains a research target. The current repository has no P6 runtime, oracle, formalization, connector, crash harness, or benchmark. The manuscript therefore makes no exactly-once, transactional cursor, recovery, equivalence, proof, or performance claim. Its next evidence gate is a joint P2/P4/P5/P13 profile freeze, followed by one small full-versus-delta fixture computed independently in Haskell and Rust. The paper should report a CatDB incremental result only after that gate passes.

References

[1] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. DBToaster: Higher-order delta processing for dynamic, frequently fresh views. *Proceedings of the VLDB Endowment*, 5(10):968–979, 2012. doi: 10.14778/2336664.2336670. URL https://www.vldb.org/pvldb/vol5/p968_yanifahmad_vldb2012.pdf.

[2] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. Millwheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):

1033–1044, 2013. doi: 10.14778/2536222.2536229. URL https://research.google.com/pubs/archive/41378.pdf.

[3] Apache Kafka. Design: Message delivery and transactions. Official Apache Kafka 4.1 documentation, 2026. URL https://kafka.apache.org/41/design/design/. Accessed 23 July 2026.

[4] Mihai Budiu, Tej Chajed, Frank McSherry, Leonid Ryzhyk, and Val Tannen. DBSP: Automatic incremental view maintenance for rich query languages. *Proceedings of the VLDB Endowment*, 16(7):1601–1614, 2023. doi: 10.14778/3587136.3587137. URL https://www.vldb.org/pvldb/vol16/p1601-budiu.pdf.

[5] Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows, 2015. URL https://arxiv.org/abs/1506.08603.

[6] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985. doi: 10.1145/214451.214456. URL https://lamport.azurewebsites.net/pubs/chandy.pdf.

[7] Debezium Project. Outbox event router. Official Debezium documentation, 2026. URL https://debezium.io/documentation/reference/stable/transformations/outbox-event-router.html. Accessed 23 July 2026.

[8] Debezium Project. Debezium connector for postgresql. Official Debezium documentation, 2026. URL https://debezium.io/documentation/reference/stable/connectors/postgresql.html. Accessed 23 July 2026.

[9] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. Noria: Dynamic, partially-stateful data-flow for high-performance web applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, pages 213–231, 2018. URL https://www.usenix.org/conference/osdi18/presentation/gjengset.

[10] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems*, pages 31–40, 2007. doi: 10.1145/1265530.1265535. URL https://www.cs.ucdavis.edu/~green/papers/pods07.pdf.

[11] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 328–339, 1995. URL https://sigmodrecord.org/1995/06/06/incremental-maintenance-of-views-with-duplicates/.

[12] Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995. URL https://www.vldb.org/dblp/db/journals/debu/GuptaM95.html.

[13] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 157–166, 1993. URL https://sigmodrecord.org/1993/06/03/maintaining-views-incrementally/.

[14] Ashish Gupta, H. V. Jagadish, and Inderpal Singh Mumick. Data integration using self-maintainable views. In *Advances in Database Technology—EDBT 1996*, pages 140–144, 1996. doi: 10.1007/BFb0014149. URL https://doi.org/10.1007/BFb0014149.

[15] Ashish Gupta, Inderpal Singh Mumick, Jun Rao, and Kenneth A. Ross. Adapting materialized views after redefinitions: Techniques and a performance study. *Information Systems*, 26(5):323–362, 2001.

[16] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 205–216, 1996. URL https://sigmodrecord.org/1996/06/24/implementing-data-cubes-efficiently/.

[17] Sanjay Krishnan, Jiannan Wang, Michael J. Franklin, Ken Goldberg, and Tim Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. *Proceedings of the VLDB Endowment*, 8(12):1370–1381, 2015. doi: 10.14778/2824032.2824049. URL https://www.vldb.org/pvldb/vol8/p1370-krishnan.pdf.

[18] Frank McSherry, Derek Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow. In *Conference on Innovative Data Systems Research*, 2013. URL https://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper111.pdf.

[19] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 439–455, 2013. doi: 10.1145/2517349.2522738. URL https://doi.org/10.1145/2517349.2522738.

[20] Oracle Corporation. Basic materialized views. Oracle Database 26 documentation, 2026. URL https://docs.oracle.com/en/database/oracle/oracle-database/26/dwhsg/basic-materialized-views.html. Accessed 23 July 2026.

[21] PostgreSQL Global Development Group. Logical replication architecture. PostgreSQL 18 documentation, 2026. URL https://www.postgresql.org/docs/current/logical-replication-architecture.html. Accessed 23 July 2026.

[22] PostgreSQL Global Development Group. Logical decoding concepts. PostgreSQL 18 documentation, 2026. URL https://www.postgresql.org/docs/current/logicaldecoding-explanation.html. Accessed 23 July 2026.

[23] PostgreSQL Global Development Group. Logical replication publication and replica identity. PostgreSQL 18 documentation, 2026. URL https://www.postgresql.org/docs/current/logical-replication-publication.html. Accessed 23 July 2026.

[24] PostgreSQL Global Development Group. Logical replication restrictions. PostgreSQL 18 documentation, 2026. URL https://www.postgresql.org/docs/current/logical-replication-restrictions.html. Accessed 23 July 2026.

[25] SQLite Project. The session extension. Official SQLite documentation, 2026. URL https://sqlite.org/sessionintro.html. Accessed 23 July 2026.

[26] SQLite Project. Data change notification callbacks. Official SQLite C interface documentation, 2026. URL https://sqlite.org/c3ref/update_hook.html. Accessed 23 July 2026.

[27] Jingren Zhou, Per-Åke Larson, and Hicham G. Elmongui. Lazy maintenance of materialized views. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 231–242, 2007. URL https://www.vldb.org/conf/2007/papers/research/p231-zhou.pdf.