A Typed Carrier Algebra for Unified Query Execution

Relational, Text, Vector, Graph, and Probabilistic Retrieval in One Runtime

Jaepil Jeong
Cognica, Inc.
Email: jaepil@cognica.io
Date: August 3, 2026

To my father, who taught me that different things may remain themselves and still belong together.

Abstract

Systems that combine SQL, full-text retrieval, vector search, graph traversal, and probabilistic ranking are often described as "unified" because they share an API or because every intermediate result is forced into one container. Neither condition is a sufficient mathematical foundation. A single container can carry several observably different structures: set membership, tuple multiplicity, scores, positional payloads, graph-match context, and rank order. Algebraic laws valid for one of these structures can be unsound for another.

This paper develops an implementation-grounded framework in which unification occurs through a **typed family of carriers** and a common planning and execution calculus. Finite document sets carry Boolean algebra; finite-support relations carry semiring addition and multiplication; decorated postings carry explicitly non-Boolean collision policies; ranked views own ordering and top-k; generalized relations preserve join-tuple identity; graph postings pair document support with invariant-checked graph context; and aggregate states carry monoids. Operators compose only when their input and output carriers agree, while explicit adapters mark every intentional loss or encoding.

Four consequences are developed in detail. First, it states exactly which rewrites—idempotence, absorption, filter pushdown, threshold merging, join reordering, pattern fusion, and aggregate decomposition—are valid under which observations. Second, it separates raw retrieval scores, prior-free evidence, priors, and posterior probabilities, giving an exact single-prior Bayesian fusion rule while distinguishing robust ranking heuristics. Third, it integrates graph traversal and regular path queries without claiming an isomorphism between arbitrary graphs and document sets; a versioned Φ codec is proved only as a lossless representation change over a constrained graph-posting carrier. Fourth, it connects the algebra to a working Rust engine whose statements lower to one typed plan hierarchy and whose specialized access paths retain their native carriers.

The result is a broad but bounded claim: relational, text, vector, graph, and probabilistic retrieval can share a principled optimizer and runtime without erasing the semantic distinctions on which correctness depends.

Keywords: query algebra, semiring relations, information retrieval, vector search, graph query, probabilistic fusion, query optimization, embedded database

1. Introduction

Modern applications increasingly ask one query to cross several data paradigms. A relational predicate narrows a corpus; full-text retrieval identifies lexical evidence; vector search supplies semantic candidates; a graph traversal adds structural context; ranking and aggregation produce the final answer. Deploying a separate engine for each stage makes the application responsible for identity mapping, transaction boundaries, score interpretation, and cross-system optimization.

A unified runtime is attractive, but the mathematical difficulty is not syntax. The difficulty is that the paradigms expose different observable structures:

- SQL commonly evaluates bags of typed rows and must preserve null and duplicate semantics [9].

- Boolean retrieval reasons about membership in a finite document universe.
- Weighted retrieval associates values with documents and may combine them additively or multiplicatively.
- Posting indexes carry positions, fields, bounds, and scores whose merge policy is observable.
- Top-k depends on order and can discard support.
- Graph matches carry vertex, edge, name, path, and temporal context.
- Joins produce tuples rather than single document identifiers.
- Probabilistic values are meaningful only when priors and evidence domains are not mixed.

Treating all of these as one undifferentiated "posting-list algebra" overstates the common structure. In particular, a posting list with payloads is not bijective with a set of documents: projecting to document identifiers loses information. A payload merge that adds scores is not idempotent. A right-biased field merge is not commutative. A ranked top-k is not a Boolean homomorphism. An arbitrary graph is not isomorphic to a document set.

These observations do not defeat unification. They locate it more precisely. We propose that a heterogeneous query system is unified when:

1. every intermediate value has an explicit carrier and observation;
2. every operator has a typed signature between carriers;
3. conversions are named and their information loss or round-trip law is stated;
4. optimizer rewrites are indexed by the laws of the carrier they transform; and
5. all statement forms participate in one plan and execution hierarchy.

This formulation is both broader and more defensible than container uniformity. It supports cross-paradigm composition while permitting each domain to retain the structure needed for correct execution.

1.1 Contributions

This paper makes the following contributions.

1. **A carrier-indexed query algebra.** We define finite Boolean support, semiring-valued relations, decorated postings, ranked views, tuple relations, graph postings, row bags, and aggregate states as separate but composable carriers.
2. **A law-preserving support projection.** We show that decorated posting merges project to document-set union and intersection while proving why the posting values themselves need not satisfy Boolean idempotence or commutativity.
3. **Typed joins and aggregations.** We express relational and cross-paradigm joins over tuple identities and state the exact monoid condition required for parallel aggregation.
4. **A score-domain calculus.** We separate raw BM25 scores, evidence logits, prior logits, and posterior probabilities. Under explicit conditional-independence assumptions, signed evidence adds and the prior enters exactly once.
5. **A graph extension without a universal-isomorphism claim.** Graph query results retain graph context in a dedicated carrier. A versioned Φ codec has a constrained round-trip theorem; graph operations and regular path queries remain graph-native.
6. **A law-indexed optimizer and unified plan model.** We connect formal laws to executable rewrite guards and to a runtime in which relational, retrieval, graph, and command plans share one exhaustive planning and execution path.
7. **An evidence discipline.** Algebraic correctness, empirical calibration, approximate-index recall, persistence, and performance are treated as different claims with different validation methods.

1.2 Scope

The model concerns finite database snapshots and an embeddable single-node runtime. It includes relational SQL, text and vector retrieval, graph queries, ranking, joins, aggregation, and transactional persistence. It does not claim distributed query execution, universal graph/document equivalence, calibration without labeled evaluation, or performance superiority without reproducible measurements.

1.3 Relation to the Earlier Unified-Algebra Formulation

This manuscript consolidates and revises the original unified-query-algebra formulation [17] and its graph-data extension [18] in light of the executable system. It retains their central objective—one compositional framework for relational, textual, vector, and graph queries—and preserves the useful Boolean-support, cross-paradigm join, aggregate, hierarchical-value, traversal, and regular-path constructions. It changes the foundation where implementation exposed stronger semantic requirements.

The principal revision is to replace the proposed universal posting-list representation with a typed family of carriers. The earlier document-set/posting-list bijection is retained only as the support/lift round trip for default-decorated postings; it is not asserted for postings with observable payload. Likewise, the graph extension's broad graph/document isomorphism is replaced by a lossless, versioned codec theorem over the invariant-checked graph-posting carrier. Score combination is divided into exact single-prior Bayesian evidence fusion and explicitly heuristic pooling, while optimizer identities are stated relative to the observation under which they are sound. These are not reductions in the system's compositional ambition. They are the conditions that let the broader claim survive contact with tuple identity, bags, payload collisions, ranking, graph context, and persistent execution.

1.4 Related work

The framework sits at the intersection of four established lines of work, and its contribution is the boundary discipline between them rather than any single one of them.

Annotated relations. Provenance semirings [15] establish that set, bag, and probabilistic relational semantics are instances of one K-annotated model, and FAQ [2] and AJAR [21] extend the treatment to aggregation over such relations. Sections 4.2 and 5.1 use that model directly. What is added here is the observation that a semiring annotation is one carrier among several: it does not subsume rank order, payload collision policy, or graph context, and the optimizer therefore cannot treat semiring laws as globally available.

Matrix and semiring formulations of graph computation. GraphBLAS [22] shows that a small set of semiring matrix operations expresses a wide class of graph algorithms. That formulation is complementary to Section 7: it unifies graph *algorithms* over a chosen semiring, whereas the concern here is preserving graph *match context*—matched vertices, matched edges, and graph name—as first-class result structure when a graph operator composes with retrieval and relational operators.

Graph query languages. Foundational treatments of graph patterns and navigational expressions [3, 5, 23, 27] supply the pattern and regular-path semantics assumed in Section 7.4. No new expressive power is claimed; the contribution is the typed boundary at which those results enter a shared optimizer.

Top-k and threshold algorithms. Instance-optimal threshold algorithms [11] and index-level pruning [7, 10] establish when ranked retrieval may terminate early. Section 4.4 restates the corresponding algebraic fact: ranking is a separate carrier, and truncation is sound only against a specific parent observation.

Nested collections. The hierarchical-value treatment of Section 5.4 follows the nested relational calculus tradition [8], which already distinguishes value-level projection from multiplicity-changing unnesting.

2. Why One Universal Posting Carrier Is Insufficient

Let U be a finite universe of document identifiers and let Π be a payload domain. A decorated posting value can be modeled as a finite partial map

$$P : U \rightharpoonup \Pi,$$

and we write $P_{\Pi}(U)$ for the set of all such maps.

Its support is

$$\operatorname{supp}(P) = \{d \in U \mid P(d) \text{ is defined}\}.$$

The map $\operatorname{supp} : P_{\Pi}(U) \to \mathcal{P}(U)$ is generally many-to-one. Two posting values with the same identifiers but different positions, scores, or fields have the same support. If $\operatorname{lift}(D)$ assigns a default payload to every $d \in D$, then

$$\operatorname{supp}(\operatorname{lift}(D)) = D,$$

but in general

$$\operatorname{lift}(\operatorname{supp}(P)) \neq P.$$

Thus document sets are a **retract** of decorated postings, not a representation isomorphic to all of them. Equivalently, $\iota = \operatorname{lift} \circ \operatorname{supp}$ is an idempotent on $P_{\Pi}(U)$ whose splitting is $\mathcal{P}(U)$ [24]. This is not a formality: the fixed points of ι are exactly the posting values on which Boolean reasoning is sound, which is the semantic content of the membership-only guard used by the optimizer in Section 8.2. Proposition 4.4 makes the identification precise.

This distinction matters operationally. Suppose a collision policy m on payloads adds scores, unions positions, and gives the right operand precedence for colliding fields; write $\widetilde{\cup}_m$ for the induced posting union defined in Section 4.3. For a posting P containing a scored document,

$$P \mathbin{\widetilde{\cup}_m} P \neq P$$

because its score is doubled. For two postings P, Q with different values for the same field,

$$P \mathbin{\widetilde{\cup}_m} Q \neq Q \mathbin{\widetilde{\cup}_m} P$$

under full payload equality. Nevertheless their supports can still satisfy ordinary set union. The optimizer must therefore know whether it is observing membership or the full decorated value.

The same problem appears elsewhere:

- a SQL bag cannot be reduced to a set without losing multiplicity;
- a join result cannot be reduced to one document identifier without losing tuple identity;
- a graph match cannot be reduced to document support without losing matched vertices and edges; and
- a rank order cannot be reconstructed from an unordered support set.

The corrected foundation is consequently many-sorted: common structure is shared where it exists, and explicit boundaries are retained where it does not.

3. Snapshot and Carrier Model

3.1 Database snapshots

A database snapshot is written

$$\Sigma = (U, \mathcal{T}, \mathcal{I}, \mathcal{G}, \Theta, \mathcal{C}),$$

where U is the finite identifier universe, $\mathcal{T}$ is relational and document data, $\mathcal{I}$ is the family of physical indexes, $\mathcal{G}$ is the named-graph state, Θ is scoring and model metadata, and $\mathcal{C}$ is catalog state. A read operator is pure relative to a pinned Σ. A state-changing operator is modeled separately in Section 9.

3.2 Core carriers

The logical and physical framework uses the following carrier family.

Carrier	Mathematical form	Primary observation
Document support D_U	$\mathcal{P}(U)$	identifier membership
K-relation $R_K(U)$	finite $r : U \to K$	identifier-value pairs
Decorated posting $P_\Pi(U)$	finite $U \rightharpoonup \Pi$	identifiers and complete payloads
Ranked view $V_s(P)$	deterministic order over a posting	score order and top-k
SQL row bag B_Γ	$\mathbb{N}^{(\mathrm{Row}_\Gamma)}$	schema, multiplicity, and values; a sequence when `ORDER BY` is present
Tuple relation $J_{\Gamma_1,\ldots,\Gamma_n}$	finite relation over identifier tuples	complete tuple identity
Graph posting $GP(U)$	posting plus graph-context side map	support, payload, and graph context
Aggregate state A_M	element of a monoid M	finalized aggregate value

Here $\mathbb{N}^{(\mathrm{Row}_\Gamma)}$ denotes finitely supported multiplicity functions. A bag carries no intrinsic order; when a statement requests one, the observation at that boundary is the induced sequence rather than the multiset, and the two must not be interchanged in a rewrite.

The list is intentionally not a subtyping hierarchy. A value may be converted only through a named map whose contract is known.

Three symbols are reused across the literature this paper draws on, and are fixed here to avoid collision: K always denotes a semiring, $\mathcal{K}[\cdot]$ always denotes an evaluation context, and κ always denotes a vector candidate-pool size.

3.3 Typed operators

Each operator has a signature. Representative examples are:

$$
\begin{aligned}
\mathrm{term}_{f,q} &: \Sigma \to P_\Pi(U),\\
\mathrm{filter}_p &: D_U \to D_U,\\
\mathrm{vector}_{f,v,\tau} &: \Sigma \to P_\Pi(U),\\
\mathrm{knn}_{f,v,k} &: \Sigma \to V_s(P_\Pi(U)),\\
\mathrm{rank}_s &: P_\Pi(U) \to V_s(P_\Pi(U)),\\
\mathrm{top}_k &: V_s(P_\Pi(U)) \to P_\Pi(U),\\
\mathrm{join}_\theta &: J_A \times J_B \to J_{A \cup B},\\
\mathrm{traverse}_R &: \mathcal{G} \to GP(U),\\
\mathrm{aggregate}_h &: B_\Gamma \to A_M.
\end{aligned}
$$

The snapshot Σ is not a carrier. For a pinned Σ, an access operator such as $\mathrm{term}_{f,q}$ is read as a morphism $\mathbf{1} \to P_\Pi(U)$ from the terminal object, so that the whole pure fragment lives in one category over the carrier objects alone.

An operator composition $g \circ f$ exists only when the output carrier of f is the input carrier of g, or when an explicit adapter is inserted. The pure fragment therefore forms a category UQA: carriers are objects, total deterministic operators are morphisms, identities are carrier identities, and composition is ordinary function composition. This modest categorical statement is sufficient. A value-only adapter is not called a functor unless an operator mapping exists and identity and

composition preservation have been established [24].

Two classes of operator are deliberately outside UQA. Merges that may reject their arguments—graph-name conflicts under an unresolving policy (Section 7.2)—and effectful commands (Section 9.3) are partial or state-changing. Both are modeled as Kleisli morphisms $A \to \mathcal{T}(B)$ for an error or state-and-error monad $\mathcal{T}$, and a rewrite that moves such an operator must preserve its error behavior, not merely its successful values. The theorems of Sections 4, 5, and 7 are stated for the total fragment; where a construction is partial, this is said explicitly.

3.4 Carrier-relative equivalence

For every carrier C, let obs_C expose the values that are semantically visible at that boundary. Queries $q_1, q_2 : \Sigma \times X \to C$ are equivalent under snapshot Σ when

$$q_1 \equiv_C q_2 \iff \mathrm{obs}_C(q_1(\Sigma, x)) = \mathrm{obs}_C(q_2(\Sigma, x)) \quad \text{for every valid } x.$$

Equality of support is therefore weaker than equality of decorated postings, and equality of an unordered tuple set is weaker than equality of an ordered SQL result. This indexed equivalence is the basis of safe rewriting.

3.5 Observations are ordered, and the order is not linear

The observations of Section 3.2 are not independent. Say that $\mathrm{obs}_{C'}$ **refines** obs_C, written $\mathrm{obs}_C \sqsubseteq \mathrm{obs}_{C'}$, when there is a function u with $\mathrm{obs}_C = u \circ \mathrm{obs}_{C'}$: everything the coarser observation can see is recoverable from the finer one.

Lemma 3.1 (Refinement weakens equivalence). If $\mathrm{obs}_C \sqsubseteq \mathrm{obs}_{C'}$, then $q_1 \equiv_{C'} q_2$ implies $q_1 \equiv_C q_2$.

Proof. Apply u to both sides of the equality of $\mathrm{obs}_{C'}$ values. □

The lemma is one line, and it is the reason the rewrite table of Section 8.2 is a structure rather than a list: a rewrite proved sound at a finer observation is automatically sound at every coarser one, and never the converse. Concretely,

$$\mathrm{obs}_{\mathsf{supp}} \sqsubseteq \mathrm{obs}_{\mathsf{payload}} \sqsubseteq \mathrm{obs}_{\mathsf{ranked}}, \qquad \mathrm{obs}_{\mathsf{supp}} \sqsubseteq \mathrm{obs}_{\mathsf{graph}},$$

since supp forgets payload, a ranked view forgets its order under re-materialization, and a graph posting forgets its side map.

The refinement order is a partial order and not a chain. SQL bag multiplicity and graph match context are incomparable: neither is recoverable from the other. An optimizer that assumes a single linear "amount of information" axis will therefore admit unsound rewrites at exactly the cross-paradigm boundaries this framework exists to protect.

4. Algebraic Laws by Carrier

4.1 Finite Boolean algebra of document support

Fix a finite universe U. For $A, B \subseteq U$, define

$$A \vee B = A \cup B, \qquad A \wedge B = A \cap B, \qquad \neg_U A = U \setminus A.$$

Theorem 4.1 (Finite support algebra). $(D_U, \vee, \wedge, \neg_U, \varnothing, U)$ is a complete Boolean algebra.

Proof. The power set of any fixed set is a complete Boolean algebra under union, intersection, and relative complement; arbitrary families have suprema and infima given by union and intersection [4]. □

Completeness does not depend on finiteness. What finiteness supplies is different and equally necessary in practice: D_U is atomic and finite, so $\neg_U$ is materializable by enumeration rather than being merely well defined [26].

The qualification “fixed universe” is essential. A complement is not an intrinsic property of a posting; it is relative to the snapshot or relation being queried. In SQL, complement lowering also requires a two-valued predicate. A nullable comparison under NOT remains in the three-valued row evaluator rather than being rewritten as set complement.

4.2 Finite-support semiring relations

Let $K = (K, \oplus, \otimes, 0, 1)$ be a semiring. Relative to finite U, define

$$R_K(U) = \{r : U \to K\}$$

with pointwise operations

$$(r \oplus s)(d) = r(d) \oplus s(d), \qquad (r \otimes s)(d) = r(d) \otimes s(d).$$

Theorem 4.2 (Pointwise relation semiring). $R_K(U)$ is a semiring under the pointwise operations, with the constant-zero and constant-one relations as identities.

Proof. Every semiring axiom holds at each $d \in U$, so it holds extensionally for the functions. □

Sparse storage omits entries equal to 0. For the Boolean semiring

$$K_{\mathbb{B}} = (\{\bot, \top\}, \vee, \wedge, \bot, \top),$$

the characteristic map $\chi : D_U \to R_{K_{\mathbb{B}}}(U)$ satisfies

$$\chi(A \cup B) = \chi(A) \oplus \chi(B), \qquad \chi(A \cap B) = \chi(A) \otimes \chi(B).$$

For the log semiring over $\mathbb{R} \cup \{-\infty\}$, $0_K = -\infty$, $1_K = 0$, addition is log-sum-exp, and multiplication is ordinary addition in log space; its elements are logarithms of non-negative weights. This supports stable combination of such weights without attributing those laws to a payload container. The K-annotation model and its recovery of set, bag, and probabilistic semantics are due to provenance semirings [15]; semiring formulations of graph computation in the same spirit are surveyed in [22].

Support projection from a general semiring relation needs two familiar qualifications:

$$\operatorname{supp}(r \oplus s) = \operatorname{supp}(r) \cup \operatorname{supp}(s)$$

when K is zero-sum-free, and

$$\operatorname{supp}(r \otimes s) = \operatorname{supp}(r) \cap \operatorname{supp}(s)$$

when K has no zero divisors [12]. The Boolean and log semirings used here satisfy both conditions: $\log(e^a + e^b) = -\infty$ only when $a = b = -\infty$, and $a + b = -\infty$ only when $a = -\infty$ or $b = -\infty$. A semiring admitting additive cancellation, such as $(\mathbb{R}, +, \cdot)$, satisfies neither, and support projection is then unsound.

4.3 Decorated postings and support homomorphisms

Let $m : \Pi \times \Pi \to \Pi$ be a collision policy. Define decorated support-union and support-intersection by

$$(P \mathbin{\widetilde{\cup}_m} Q)(d) = \begin{cases} m(P(d), Q(d)), & d \in \operatorname{supp}(P) \cap \operatorname{supp}(Q), \\ P(d), & d \in \operatorname{supp}(P) \setminus \operatorname{supp}(Q), \\ Q(d), & d \in \operatorname{supp}(Q) \setminus \operatorname{supp}(P), \end{cases}$$

and by retaining only the first case for $\widetilde{\cap}_m$.

Proposition 4.3 (Support preservation).

$$\operatorname{supp}(P \mathbin{\widetilde{\cup}_m} Q) = \operatorname{supp}(P) \cup \operatorname{supp}(Q),$$
$$\operatorname{supp}(P \mathbin{\widetilde{\cap}_m} Q) = \operatorname{supp}(P) \cap \operatorname{supp}(Q).$$

Proof. The constructors include exactly the identifiers in the corresponding set operation; m changes only the payload at an already included identifier. □

No Boolean law for the full posting value follows from Proposition 4.3. If m adds scores, then $m(p, p) \neq p$ in general. If it gives one side field precedence, then $m(p, q) \neq m(q, p)$ in general. Consequently:

- membership-only subtrees may use Boolean idempotence and absorption;
- score-bearing or field-bearing subtrees may not be deduplicated merely because they are structurally equal; and
- set-level optimizer proofs must name support equality rather than silently using full-value equality.

The first bullet is usually stated informally. It has an exact characterization, which matters because the optimizer must decide it mechanically. Let $\pi_0 \in \Pi$ be the default payload used by lift, and let

$$P_\Pi^{\mathsf{def}}(U) = \{P \in P_\Pi(U) \mid P(d) = \pi_0 \text{ for all } d \in \mathrm{supp}(P)\}.$$

Proposition 4.4 (Membership-only values are the fixed points of the retraction). For $P \in P_\Pi(U)$, the following are equivalent: (i) $P \in P_\Pi^{\mathsf{def}}(U)$; (ii) $\mathrm{lift}(\mathrm{supp}(P)) = P$. If in addition $m(\pi_0, \pi_0) = \pi_0$, then $P_\Pi^{\mathsf{def}}(U)$ is closed under $\widetilde{\cup}_m$ and $\widetilde{\cap}_m$, and

$$\left(P_\Pi^{\mathsf{def}}(U), \widetilde{\cup}_m, \widetilde{\cap}_m\right) \;\cong\; (\mathcal{P}(U), \cup, \cap)$$

as lattices, with supp and lift mutually inverse.

Proof. (i) ⇒ (ii): lift assigns π_0 on exactly $\mathrm{supp}(P)$, which by (i) reproduces P. (ii) ⇒ (i): the value of $\mathrm{lift}(\mathrm{supp}(P))$ at any d in its domain is π_0 by construction. Closure: the three cases of $\widetilde{\cup}_m$ yield $m(\pi_0, \pi_0) = \pi_0$, π_0, and π_0 respectively. The lattice isomorphism is then Proposition 4.3 restricted to $P_\Pi^{\mathsf{def}}(U)$, where supp is injective. □

Idempotence, commutativity, associativity, and absorption therefore hold on $P_\Pi^{\mathsf{def}}(U)$ and are transported from $\mathcal{P}(U)$ rather than assumed. The engineering consequence is stated as a proof obligation in Section 8.2: the syntactic membership-only predicate must be a sound under-approximation of $P_\Pi^{\mathsf{def}}(U)$. Rejecting a value that happens to lie in $P_\Pi^{\mathsf{def}}(U)$ costs an optimization; accepting one that does not costs correctness.

4.4 Ranking is a separate algebra

For a posting P, a scoring function $s : U \times \Pi \to \mathbb{R}$, and a deterministic tie key, $\mathrm{rank}_s(P)$ totally orders $\mathrm{supp}(P)$ by $s(d, P(d))$. The selection top_k returns at most k entries and then may be re-materialized in identifier order.

Top-k is deterministic, but it is not monotone with respect to support inclusion: if $A \subseteq B$, a document in $\mathrm{top}_k(A)$ may be evicted in $\mathrm{top}_k(B)$ by higher-scoring members of $B \setminus A$.

Whether truncation may be pushed below a merge depends on the merge, and the distinction is exactly the payload distinction of Section 4.3. Call a union **score-preserving** when the score of d in the result depends only on d and not on which operands contained it.

Proposition 4.5 (Pushdown under a score-preserving union). If $\widetilde{\cup}_m$ is score-preserving and the tie key is a function of the identifier alone, then

$$\mathrm{top}_k(A \,\widetilde{\cup}_m\, B) = \mathrm{top}_k\Big(\mathrm{top}_k(A) \,\widetilde{\cup}_m\, \mathrm{top}_k(B)\Big).$$

Proof. Write $S = \mathrm{top}_k(A) \,\widetilde{\cup}_m\, \mathrm{top}_k(B)$. Let $d \in \mathrm{top}_k(A \,\widetilde{\cup}_m\, B)$ and suppose $d \in \mathrm{supp}(A)$, the other case being symmetric. Score preservation gives d the same key in A as in the merge, and $\mathrm{supp}(A) \subseteq \mathrm{supp}(A \,\widetilde{\cup}_m\, B)$, so d is among the k largest keys of A and survives its truncation. Hence

$$\mathrm{top}_k(A \,\widetilde{\cup}_m\, B) \subseteq S \subseteq A \,\widetilde{\cup}_m\, B,$$

with keys unchanged throughout. The k largest keys of S are therefore exactly the k largest of $A \,\widetilde{\cup}_m\, B$, and $\mathrm{top}_k(S)$ equals the left side. □

The proposition fails as soon as m combines scores. Let $k = 1$, $A = \{d_1 \mapsto 3,\ d_2 \mapsto 2\}$, $B = \{d_2 \mapsto 2\}$, and let m add scores. Then $A \mathbin{\widetilde{\cup}_m} B = \{d_1 \mapsto 3,\ d_2 \mapsto 4\}$, while $\mathrm{top}_1(A) = \{d_1 \mapsto 3\}$ and $\mathrm{top}_1(B) = B$, so

$$\mathrm{top}_1(A \mathbin{\widetilde{\cup}_m} B) = \{d_2\}, \qquad \mathrm{top}_1\Big(\mathrm{top}_1(A) \mathbin{\widetilde{\cup}_m} \mathrm{top}_1(B)\Big) = \{d_1\},$$

because truncating A first discards the d_2 contribution that the merge would have amplified. Fusion parents behave the same way: a document below the local threshold in every input can be raised above it by their combination.

Pushing top-k below such a parent therefore requires a separate threshold algorithm and a valid upper bound on the unfinished contribution, not a rewrite. In the implementation described here, exact WAND [7] and Block-Max WAND [10] are used only where the complete score is monotone in safe per-term bounds; duplicate query terms remain distinct, and a Boolean or fusion parent blocks leaf truncation. The general instance-optimality results for threshold algorithms over sorted access apply to this setting [11].

5. Joins, Bags, Aggregates, and Hierarchical Values

5.1 Tuple identity and semiring joins

Let schemas A and B share compatible attributes. A K-annotated relation over schema A is a finite function

$$R : \mathrm{Tuple}(A) \to K.$$

The natural join of R over A and S over B is the K-annotated relation over $A \cup B$ given by

$$(R \bowtie S)(t) = R(t|_A) \otimes S(t|_B), \qquad t \in \mathrm{Tuple}(A \cup B).$$

Every tuple over $A \cup B$ restricts to both schemas, so no separate compatibility case is required; absence is carried by the annotation 0 rather than by a partial domain. Projection to schema $C \subseteq A$ sums annotations:

$$(\pi_C R)(u) = \bigoplus_{t : t|_C = u} R(t).$$

This recovers set semantics with $K = \mathbb{B}$ and bag semantics with $K = \mathbb{N}$; both are instances of the annotated-relation model of [15], and the relational baseline is standard [1, 9]. For S and T over the *same* schema B, it also gives the distributive law

$$R \bowtie (S \oplus T) = (R \bowtie S) \oplus (R \bowtie T)$$

by semiring distributivity. The schema condition is not decorative: $\oplus$ is defined only between relations of one schema, so a rewrite must check it rather than infer it from the shape of the expression.

The result identity is the full tuple t, not an invented scalar document identifier. A generalized posting representation can therefore sort and deduplicate identifier tuples while retaining their payload. In a SQL row pipeline, multiplicities remain in the bag carrier; in cross-paradigm access joins, the generalized tuple carrier provides set-valued tuple identity. The two are connected at an explicit materialization boundary rather than conflated.

Inner equijoins are associative and commutative up to schema-preserving tuple reordering. Outer joins, lateral dependencies, null-extension, order-sensitive operations, and volatile expressions do not inherit those reorderings. A join optimizer may flatten and reorder only an inner region whose predicates and tuple projections meet the necessary conditions.

5.2 Cross-paradigm joins

Text, vector, and graph joins are predicates over typed tuple components:

$$\theta_{\text{text},\tau} = \{(l, r) \mid \text{sim}_{\text{text}}(l.f, r.g) \geq \tau\},$$
$$\theta_{\text{vec},\tau} = \{(l, r) \mid \text{sim}_{\text{vec}}(l.v, r.w) \geq \tau\},$$
$$\theta_{\text{graph}} = \{(l, r) \mid r.id \in \text{traverse}(G, l.id, R)\}.$$

(The symbol θ is used for join predicates throughout; J remains reserved for the tuple-relation carrier of Section 3.2.)

Their shared structure is tuple formation and predicate evaluation; their candidate generation and cost are domain-specific. Preserving $(l.id, r.id)$ lets later filters, projections, and ranking identify both sides without encoding an enumeration position as a document.

5.3 Mergeable aggregation

An aggregate is parallelizable when it factors through a monoid

$$(M, \odot, e).$$

Let $h : X \to M$ lift one input and define, on *sequences*,

$$\text{fold}(x_1, \ldots, x_n) = h(x_1) \odot \cdots \odot h(x_n), \qquad \text{fold}(\varepsilon) = e.$$

Theorem 5.1 (Contiguous partition decomposition). If a sequence σ splits as the concatenation $\sigma_1\sigma_2$, then

$$\text{fold}(\sigma) = \text{fold}(\sigma_1) \odot \text{fold}(\sigma_2).$$

Proof. Reassociate, using e for an empty part. □

Associativity alone gives the contiguous case. It does not give the case the executor actually needs, because a physical partition of a bag imposes no order relation between its parts.

Corollary 5.2 (Bag partition decomposition). If M is commutative, then fold is well defined on bags—independent of the enumeration chosen—and for every disjoint bag partition $X = X_1 \uplus X_2$,

$$\text{fold}(X) = \text{fold}(X_1) \odot \text{fold}(X_2).$$

Proof. Commutativity makes the value of a fold invariant under permutation, so any enumeration of X may be permuted into one enumerating X_1 before X_2; then apply Theorem 5.1. □

The separation is the point. An aggregate declared order-insensitive and executed over an unordered partition requires a commutative monoid; associativity is not enough, and a monoid that is associative but not commutative—string concatenation, for instance—will produce a plan-dependent answer if the two conditions are conflated.

COUNT and SUM use commutative additive monoids. AVG uses the product state $(\text{sum}, \text{count})$ and finalizes only after partial states are merged. Ordered-set aggregates are not covered unless their state explicitly preserves the required order, in which case only Theorem 5.1 applies and the partition must be contiguous. Aggregation over annotated relations in this style is treated in depth in [2, 21].

5.4 Hierarchical values

Hierarchical data is modeled recursively:

$$\text{Value} = \text{Atom} + \text{List}(\text{Value}) + \text{Map}(\text{String}, \text{Value}).$$

A path p defines a partial evaluation

$$\text{eval}_p : \text{Value} \to \text{Option}(\text{Value}).$$

This is the standard complex-value model of the nested relational calculus [8]. A deterministic path filter distributes over bag union because it evaluates each row independently. Path projection and unnesting have different carriers: projection changes row values, while unnesting changes multiplicity and therefore belongs to the row-bag algebra. Idempotence of projection is claimed only for a normalized representation whose output paths are stable under repeated evaluation; it is not inferred merely from the word "projection."

6. Retrieval Scores and Probabilistic Evidence

The score-domain treatment builds on the Bayesian hybrid-search framework [19] while narrowing its claims at implementation boundaries: unlabeled monotone mappings remain score transforms until labeled validation, and exact signed-evidence fusion remains distinct from robust positive-evidence pooling.

6.1 Raw text scores

For query terms q, a BM25-style score has the additive form

$$S_{\mathrm{BM25}}(q,d) = \sum_{t \in q} \mathrm{IDF}(t) \frac{f(t,d)(k_1+1)}{f(t,d) + k_1\left(1 - b + b\,|d|/\,\mathrm{avgdl}\right)}.$$

The implementation uses a numerically stable equivalent and a Robertson–Spärck Jones IDF [31]. The complete raw query score belongs to the carrier

$$S_{\mathrm{raw}} \in \mathbb{R},$$

not to a probability domain. Term contributions are summed before a query-level calibration transform is applied. This order matters: applying a sigmoid independently to every term and then adding the outputs is not equivalent.

For parameters $\alpha > 0$ and $\beta \in \mathbb{R}$, the monotone transform

$$c_{\alpha,\beta}(S) = \sigma(\alpha(S - \beta))$$

maps a complete raw score into $[0,1]$. Monotonicity preserves within-query order and permits an upper bound B on the raw score to be mapped to $c_{\alpha,\beta}(B)$. It does not, by itself, prove empirical calibration. The same one-dimensional logistic form is standard for post-hoc calibration [30], and the modern evidence is that fitting it does not by itself make an output reliable [16]. If α and β are estimated without relevance labels, the result is a bounded score transform until reliability is evaluated on held-out judgments.

The public mathematical boundary therefore distinguishes:

$$\begin{aligned} \mathrm{RawBm25Score} &\subset \mathbb{R}, \\ \mathrm{EvidenceLogit} &\subset \mathbb{R}, \\ \mathrm{PriorLogit} &\subset \mathbb{R}, \\ \mathrm{PosteriorProbability} &\subset [0,1]. \end{aligned}$$

The wrappers have the same machine representation but different admissible operations.

6.2 Exact single-prior fusion

Let $R \in \{0,1\}$ denote relevance. Signals $x_1, \ldots, x_n$ are **class-conditionally independent** when

$$p(x_1, \ldots, x_n \mid R = r) = \prod_{i=1}^{n} p(x_i \mid R = r) \qquad \text{for both } r \in \{0,1\}.$$

Both conditions are needed. Independence given $R = 1$ alone factors the numerator of the posterior odds and leaves the denominator intact, so the log-odds do not decompose. For such signals define prior-free likelihood-ratio evidence

$$e_i = \log \frac{p(x_i \mid R = 1)}{p(x_i \mid R = 0)}$$

and a prior logit

$$\lambda_0 = \log \frac{p(R = 1)}{1 - p(R = 1)}.$$

Theorem 6.1 (Bayesian evidence fusion). Under class-conditional independence,

$$p(R = 1 \mid x_1, \dots, x_n) = \sigma\left(\lambda_0 + \sum_{i=1}^{n} e_i\right).$$

Proof. Bayes' rule gives the posterior odds as the prior odds times the joint likelihood ratio. Class-conditional independence factors that ratio into $\prod_i p(x_i \mid R = 1)/p(x_i \mid R = 0)$. Taking logarithms turns the product into $\lambda_0 + \sum_i e_i$; applying the logistic inverse returns the probability. □

Three consequences are operationally important.

1. Zero evidence is neutral.
2. Negative evidence must remain negative; discarding it changes the model.
3. The prior enters once. Feeding posterior probabilities that already contain the same prior into the sum double-counts it.

If a calibrated signal output p_i contains a base-rate logit λ_i, its prior-free evidence is $\operatorname{logit}(p_i) - \lambda_i$. The type boundary should make this conversion explicit.

6.3 Robust positive-evidence pooling

Applications sometimes prefer a robust ranking rule in which a weak matching signal cannot lower a document below the pool prior. One useful family is

$$h(p_1, \dots, p_n) = \sigma\left(\lambda_0 + n^{\alpha} \frac{1}{n} \sum_{i=1}^{n} g(\operatorname{logit}(p_i))\right),$$

where g may be softplus, ReLU, sigmoid, or a learned gate, and optional normalized weights or per-signal bounds may be added. The factor n^{α}/n is $n^{\alpha-1}$; with $\alpha \in [0, 1]$, the endpoint $\alpha = 1$ recovers the plain sum, $\alpha = 0$ the mean, and $\alpha = \frac{1}{2}$ the central-limit scaling.

The stated property—that a weak signal cannot pull a document below the pool prior—holds precisely when $g \geq 0$. Softplus, ReLU, and the sigmoid satisfy this; a learned gate does not unless it is constrained to. An unconstrained gate is admissible, but then the non-negativity guarantee must be dropped from the operator's contract rather than assumed from the family it belongs to.

This family is intentionally not called exact Bayesian fusion. Gating, confidence scaling, query-pool normalization, and learned weights change the likelihood-ratio calculus. It is a bounded ranking heuristic whose usefulness is empirical. Keeping it under a distinct operator name prevents a desirable ranking policy from being mistaken for a posterior theorem.

6.4 Vector scores, calibration, and approximation

Cosine similarity, Euclidean distance, and index-specific scores are not probabilities. A reusable vector calibration model must identify at least the corpus, physical index, embedding model, dimensionality, candidate-pool size κ, and model/schema version. Applying it to a different target is a model mismatch, not a harmless implementation detail.

Exact brute-force vector search and approximate IVF or HNSW [25] search also make different claims. A calibration result does not prove approximate recall, and a recall experiment does not prove probability calibration. The physical-index contract therefore separates:

- distance or similarity correctness;
- candidate provenance and κ;
- approximate recall against exact search;
- persistence and mutation behavior; and
- held-out reliability, expected calibration error, Brier score [6], and log loss [16].

7. Graph Queries as a Native Carrier

7.1 Property graph state

A named property graph is

$$G = (V, E, s, t, \ell_V, \ell_E, \rho_V, \rho_E),$$

where V and E are finite vertex and edge sets, $s, t : E \to V$ are endpoints, ℓ_V, ℓ_E are labels, and ρ_V, ρ_E are property maps; this is the standard property-graph model [3, 5]. Named graphs may share stored entities while retaining separate membership.

Graph query results need more than document support. Define a graph payload

$$\gamma(d) = (V_d, E_d, n_d, \hat{s}_d),$$

containing the matched vertex set, matched edge set, graph name, and an optional score override. A graph posting is

$$GP = (P, \gamma)$$

with invariant

$$\mathrm{dom}(\gamma) \subseteq \mathrm{supp}(P).$$

The invariant prevents graph metadata from referring to a result that the underlying posting does not contain.

7.2 Explicit graph merge policies

When two graph results overlap on document d, ordinary posting payloads and graph context are different contracts. The graph component uses an explicit policy

$$\mu \in \{\mathrm{Union}, \mathrm{Intersection}, \mathrm{PreferLeft}, \mathrm{PreferRight}\}.$$

Graph-name conflicts are errors unless a chosen policy resolves them. Union and intersection of subgraph vertex and edge sets are therefore intentional graph operations, not accidental consequences of a generic map precedence rule.

Because a conflict is an error rather than a value, a graph merge under an unresolving policy is a partial operation and does not live in the total category UQA of Section 3.3. It is a Kleisli morphism for the error monad, and a rewrite that reorders or duplicates such a merge must preserve which inputs raise—not only which values are produced when none does. A rewrite that turns a raising plan into a succeeding one has changed the query, even if every non-raising execution agrees.

7.3 The versioned Φ codec

Let $P_\Phi(U) \subset P_\Pi(U)$ be the decorated postings whose reserved fields contain a recognized versioned graph envelope. The codec

$$\Phi : GP(U) \to P_\Phi(U)$$

stores graph payloads in reserved posting fields while preserving the original score and any values that already occupied those fields.

Theorem 7.1 (Constrained graph-posting round trip). Assume:

- **(H1) Reserved-namespace disjointness.** The reserved field namespace is not writable through ordinary payload construction, so no posting that was never encoded can present a well-formed envelope.
- **(H2) Version rejection.** A decoder that meets an envelope whose version it does not recognize raises an error; it does not fall back to generic payload handling.

Then Φ is injective on valid graph postings, Φ^{-1} is total on $\Phi(GP(U))$ and undefined elsewhere, and for every valid graph posting $g = (P, \gamma)$,

$$\Phi^{-1}(\Phi(g)) = g, \qquad \operatorname{supp}(\Phi(g)) = \operatorname{supp}(P).$$

Proof sketch. Encoding writes a versioned envelope containing the base score, exact optional score-override state, graph name, vertex and edge identifiers, and displaced reserved values. Decoding recognizes the version, restores the displaced values and base score, and reconstructs the side map under the support invariant. Encoding neither inserts nor removes posting identifiers. □

Neither hypothesis is bookkeeping. Without (H1), the recognition step in decoding is unsound: a user payload that happens to occupy a reserved field in envelope shape is decoded as graph context, and the result is well-typed and wrong. Without (H2), version skew across a persisted store degrades silently—an unrecognized envelope decoded as an opaque payload satisfies neither equation, and the failure surfaces later as missing graph context rather than as a decode error at the boundary where it occurred. (H2) is the reason Φ^{-1} is specified as a partial function rather than a best-effort one.

This theorem is deliberately narrower than an isomorphism between arbitrary graphs and document collections. Φ is a lossless representation change for graph-posting values. It does not claim that every graph is determined by a document set or that generic posting merge is a graph-algebra homomorphism.

7.4 Traversal, patterns, and regular paths

A regular path expression over edge labels is generated by

$$R ::= a \;\mid\; R_1 \cdot R_2 \;\mid\; R_1 + R_2 \;\mid\; R^*,$$

where a ranges over edge labels, $\cdot$ is concatenation, $+$ is alternation, and * is Kleene star. Alternation is written $+$ rather than $|$ so that it does not collide with the BNF separator. Compilation produces an automaton $A_R = (Q, \delta, q_0, F)$. Evaluating an RPQ is reachability in the product graph

$$G \times A_R,$$

whose states are $(v, q) \in V \times Q$. For a fixed automaton and one start set, traversal is linear in the reachable portion of the product, bounded by

$$O(|Q|(|V| + |E|))$$

before output materialization. This is the cost of *reachability*; enumerating witnessing paths is a different problem, and for simple-path semantics it is intractable in general [27]. Parser, NFA, and DFA size limits are physical safeguards against state explosion. Path queries with data-value comparisons require a strictly richer formalism than the one above [23].

A graph pattern is better modeled as a relation of variable assignments than as an opaque "subgraph object." If patterns P_1 and P_2 share variables, then matching their merged constraints is equivalent to a natural join of assignment relations when:

1. shared variables have the same domain and identity semantics;
2. all vertex, edge, direction, label, property, and temporal constraints are retained; and
3. multiplicity or duplicate-elimination policy is the same on both sides.

Under these conditions,

$$\mathrm{Match}(P_1 \sqcup P_2, G) \equiv \mathrm{Match}(P_1, G) \bowtie \mathrm{Match}(P_2, G).$$

General subgraph isomorphism remains computationally hard, and pattern semantics differ across languages in ways that change the answer set—homomorphism, no-repeated-edge, and injective matching are not interchangeable [3]. The presence of a unified algebra does not remove either boundary; it enables the optimizer to choose filters, indexes, traversal direction, cached patterns, or bounded RPQs while preserving whichever semantics the statement declared.

8. Unified Planning and Law-Indexed Optimization

8.1 One plan hierarchy, several execution carriers

The implementation realizes the typed framework with the plan sum

$$\mathrm{UnifiedPlan} = \mathrm{Query}(\mathrm{QueryPlan}) + \mathrm{Command}(\mathrm{CommandPlan}).$$

A query plan owns CTEs and a relational root. Every query block carries an access decision:

$$\mathrm{AccessPath} = \mathrm{Row} + \mathrm{OperatorTree} + \mathrm{Hybrid}.$$

The row path retains SQL bag, null, scalar-expression, window, and ordering semantics. The operator-tree path handles document support, retrieval, graph, scoring, fusion, and cross-paradigm operators. The hybrid path evaluates posting candidates followed by a row-level residual. Specialized execution returns the typed sum

$$\mathrm{OperatorOutput} = \mathrm{Posting} + \mathrm{Graph} + \mathrm{Generalized}.$$

Thus "unified" does not mean that every node returns the same container. It means that every statement is lowered, optimized, and executed through one exhaustive hierarchy, and that carrier changes are explicit in that hierarchy.

The pipeline is:

$$\mathrm{Statement} \longrightarrow \mathrm{UnifiedPlan} \longrightarrow \text{plan-native optimizer} \longrightarrow \mathrm{UnifiedPlanExecutor}.$$

Within a retrieval access child:

$$\mathrm{ScalarExpr} \longrightarrow \mathrm{OperatorTree} \longrightarrow \mathrm{QueryOptimizer} \longrightarrow \mathrm{PlanExecutor}.$$

Unsupported retrieval lowering returns control to the enclosing relational node. It does not invoke an independent top-level dispatcher with different semantics.

8.2 Law-indexed rewrite criterion

Theorem 8.1 (Contextual rewrite soundness). Let $e_1, e_2 : \Sigma \times X \to C$ be pure expressions with $e_1 \equiv_C e_2$. Let $\mathcal{K}[\cdot]$ be a deterministic context of result carrier Y that **observes only** obs_C, by which is meant that it factors as

$$\mathcal{K} = \overline{\mathcal{K}} \circ \mathrm{obs}_C \qquad \text{for some } \overline{\mathcal{K}}.$$

Then $\mathcal{K}[e_1] \equiv_Y \mathcal{K}[e_2]$.

Proof. For every valid input, $\mathrm{obs}_Y(\mathcal{K}[e_i](\Sigma, x)) = \mathrm{obs}_Y\left(\overline{\mathcal{K}}(\mathrm{obs}_C(e_i(\Sigma, x)))\right)$, and the two inner arguments are equal by hypothesis. □

Corollary 8.2 (Rewrites descend the refinement order). If $\mathrm{obs}_C \sqsubseteq \mathrm{obs}_{C'}$ and $e_1 \equiv_{C'} e_2$, then the conclusion holds in every context that observes only obs_C. Combine Lemma 3.1 with Theorem 8.1. □

Stating the observation restriction as a factorization is what makes the theorem usable rather than circular. "Observes only obs_C" is not a property a node can inspect about itself; it is a property of its parent, and it must be either declared per operator or derived. The implementation declares it. Deriving it—a top-down analysis that propagates a demanded observation level from the plan root, the dual of required-column analysis in a compiler—would admit rewrites that per-node classification cannot express, since a score-bearing subtree whose parent demands only support recovers Boolean idempotence. This is listed as open work in Section 11.

The theorem is elementary; its engineering consequence is substantial. A rewrite implementation must establish:

1. the carrier C at the rewritten boundary;
2. the equality observed by its parent;
3. purity and error behavior;
4. the side conditions of the algebraic law; and
5. any ordering, null, multiplicity, or snapshot constraints.

The current rewrite families can be summarized as follows. The second column names the finest observation at which the two sides of the rewrite are equivalent; by Corollary 8.2, each entry is then sound in every context whose demanded observation is that one or coarser.

Rewrite	Preserves	Required carrier and side conditions
$A \wedge A \to A$, $A \vee A \to A$	support	membership-only values in the sense of Proposition 4.4
absorption	support	membership-only values and structural equivalence
complement	support	explicit universe and a null-free, two-valued predicate
filter pushdown	payload	deterministic predicate whose fields belong to the child; no changed null-extension
vector threshold merge	payload	identical field and query vector; threshold predicate with $\max(\tau_1, \tau_2)$
top-k pushdown	rank order	score-preserving merge and identifier-only tie key (Proposition 4.5)
inner-join reordering	tuple identity	associative inner region; predicates preserved; no outer or lateral boundary
pattern-filter fusion	graph context	constraint representable inside the graph pattern
pattern-join fusion	graph context	compatible shared variables and identical assignment multiplicity
aggregate decomposition	aggregate value	commutative monoid state for unordered partitions (Corollary 5.2); contiguous partition otherwise

Truncation below a score-combining merge is deliberately absent from the table. It is not a rewrite at all: by the counterexample of Section 4.4 no observation is preserved, and early termination there requires a threshold algorithm with admissible bounds (Section 9.2), which is a physical refinement justified separately.

The table also explains why a globally enabled "Boolean simplifier" is unsafe. Idempotence preserves support and nothing finer, so a parent that observes scores may not use it; the refinement order runs one way only. The implementation classifies operator-tree variants exhaustively with an is-membership-only predicate, and Proposition 4.4 states the obligation that predicate discharges: every variant it accepts must produce values in $P_{\Pi}^{\mathsf{def}}(U)$ on all inputs. Soundness requires under-approximation, not exactness—rejecting a membership-only value forfeits an optimization, while accepting a payload-bearing one forfeits correctness. A new variant is therefore ineligible for Boolean idempotence until this obligation is discharged, which makes the review a proof step rather than a convention.

8.3 Cost is not correctness

Cardinality and cost estimates select plans; they do not define query results. A simple independence estimate such as

$$\widehat{|A \cap B|} = \frac{|A||B|}{|U|}\, c_{A,B}$$

is useful only together with its model for the correlation factor $c_{A,B}$. Likewise, a graph estimate based on degree or edge-label frequency is a heuristic.

Probability bounds must state their confidence parameter. Let $X = \sum_{i=1}^{n} X_i$ be a sum of independent Bernoulli variables with $\mu = \mathbb{E}[X]$. The multiplicative Chernoff bound gives $\Pr[|X - \mu| \geq \epsilon\mu] \leq 2\exp(-\mu\epsilon^2/3)$ for $0 < \epsilon \leq 1$, so a two-sided failure probability of at most δ requires

$$\epsilon \geq \sqrt{\frac{3\ln(2/\delta)}{\mu}}$$

in that regime [28]. Two qualifications travel with the formula and are easy to drop. First, μ is the expected count $\mathbb{E}[X]$, not a per-trial mean. Second, independence is a hypothesis, and selectivity estimates over correlated predicates violate it—which is the same objection this section raises against writing $1/\sqrt{\mu}$ with no confidence parameter attached. An unqualified expression of either kind is not a high-probability guarantee.

Join enumeration uses DPccp [29] inside reorderable inner regions and retains the selected physical strategy. Approximate vector-index recall, estimator error, and runtime latency remain empirical properties reported with corpus, parameters, executable identity, and measurement method.

9. Physical Execution, State, and Persistence

9.1 Pull execution and materialization

Relational execution uses a pull protocol over row-oriented batches in the Volcano style [13]. Blocking operators—sort, distinct, set operations, windows, grouping, and joins—must either account for memory and spill or document a bounded-input contract. A materialized SQL API and a streaming cursor API may expose different delivery mechanisms while preserving the same schema-ordered row observation.

The current physical row representation is dynamic and positional. It is not claimed to be a fully vectorized typed-column engine. Logical names and hidden `(qualifier, column)` identities map to physical slots at the schema level, duplicate labels retain distinct in-flight values, and columnar result batches are produced without converting intermediate rows to maps.

9.2 Exact top-k as a physical refinement

WAND [7] and Block-Max WAND [10] are physical refinements of exhaustive scoring only when they return the same ranked result. Let $S(d)$ be the complete monotone score and B_i admissible upper bounds for unfinished contributions. Skipping is sound when the sum of relevant bounds cannot exceed the current k-th score.

For Bayesian BM25, the sigmoid is applied once after the complete raw sum. Its monotonicity permits the raw upper bound to be transformed safely. Persisted block bounds include a fingerprint of scorer parameters and field statistics; a write invalidates them. If validity changes after planning, execution falls back to exact WAND rather than using stale metadata.

9.3 Stateful semantics

An effectful command is modeled as

$$f : \mathcal{S} \times X \to \mathrm{Result}(\mathcal{S} \times Y, \mathcal{E}),$$

where $\mathcal{S}$ includes durable catalog state, graph state, indexes, models, caches, and session-local state, and $\mathcal{E}$ is the error type. (E remains the edge set of Section 7.1.) This is the Kleisli form anticipated in Section 3.3; commands are outside the total category UQA by construction. For a transaction T, atomic visibility requires [14]:

$$\mathrm{publish}(T) \text{ occurs only after every durable component of } T \text{ succeeds.}$$

On failure, the externally visible state is observationally equivalent to the pre-transaction state. This condition is more than storage durability: in-memory graph registries, index metadata, scoring models, and caches must roll back with the catalog.

Logical sessions own transaction affinity, variables, search path, prepared plans, cancellation, sequence state, and statement serialization. Published generation counters may be shared, but mutable transaction state may not leak between sibling sessions.

9.4 Reopen invariance

For deterministic reads, a persistent representation is correct when closing and reopening a committed snapshot preserves the observable query result:

$$\mathrm{obs}_C\left(q(\mathrm{reopen}(\mathrm{persist}(\Sigma)))\right) = \mathrm{obs}_C(q(\Sigma)).$$

This is an implementation refinement theorem under assumptions that serialization is complete, migrations preserve semantics, and external nondeterminism is absent. It is validated separately for documents, schemas, indexes, graphs, models, statistics, transactions, and encrypted storage.

10. Implementation Correspondence and Validation

10.1 Correspondence to UQA Engine

The UQA Engine 0.1.0 preproduction implementation [20] maps the theory to concrete boundaries.

Formal component	Implementation component	Enforced contract
D_U	DocSet	finite support, explicit-universe complement, Boolean property tests
$R_K(U)$	Relation<K>, Semiring	sparse nonzero entries, pointwise plus and times
$P_{\Pi}(U)$	PostingList, Payload	sorted unique identifiers and explicit collision policy
$V_s(P)$	RankedView	descending score, deterministic identifier tie-break, separate materialization
J	GeneralizedPostingList	lexicographically ordered identifier tuples
$GP(U)$	GraphPostingList	side-map support invariant and explicit subgraph policies
Φ	GraphPostingCodec	versioned payload-preserving round trip
score domains	RawBm25Score, EvidenceLogit, PriorLogit, PosteriorProbability	explicit construction and conversion
typed access IR	OperatorTree, OperatorOutput	exhaustive output-carrier dispatch
unified statement IR	UnifiedPlan, QueryPlan, CommandPlan	exhaustive lowering and execution
physical refinement	TextTopKPlan, WAND, Block-Max WAND	differential equality with exhaustive scoring

The Rust workspace separates core carriers, analysis, storage, scoring, fusion, operators, graph processing, joins, planning, physical execution, SQL, engine integration, machine learning, command-line access, APIs, protocol support, and language bindings into 21 crates. This modularity is an ownership boundary, not evidence that the runtime is fragmented: the executable plan hierarchy composes them.

10.2 Validation layers

The artifact uses different evidence for different kinds of claim.

1. **Algebraic properties.** Property tests exercise Boolean idempotence, commutativity, associativity, distributivity, complements, De Morgan laws, and support round trips over DocSet, over ‹N_1› generated cases per law. Counterexample tests verify that decorated posting merges are not falsely treated as idempotent or commutative.
2. **Carrier and codec invariants.** Constructors reject graph payload keys outside support. Versioned Φ round trips preserve scores, optional overrides, graph names, vertex and edge identifiers, and pre-existing reserved fields across ‹N_2› randomized graph postings, and unrecognized envelope versions are asserted to raise rather than degrade, per hypothesis (H2) of Theorem 7.1.
3. **Optimizer equivalence.** Structural tests verify that membership-only expressions simplify while scored branches remain distinct. Randomized tests compare optimized plans with unoptimized or exhaustive execution over ‹N_3› generated plans.
4. **Top-k exactness.** WAND and Block-Max WAND are compared against exhaustive ranking over ‹N_4› query/corpus configurations, including duplicate terms, field-scoped statistics, scorer changes, writes, and reopen cycles.
5. **Compatibility.** SQL and graph behavior uses golden fixtures and differential probes against the declared external semantics, including PostgreSQL-oriented cases.

6. **Persistence.** Reopen and rollback tests cover catalog objects, relational data, indexes, tensors, graphs, scoring parameters, models, views, routines, sequences, and encrypted or compressed stores.
7. **Performance methodology.** Thirty-two Rust benchmark entrypoints are tracked by a machine-checked coverage manifest. A benchmark's presence is not itself a speed claim; published comparisons require same-machine artifacts, executable hashes, fixtures, warmup, sample count, and ratio gates.

(The placeholders ‹N_1›–‹N_4› are to be filled from the coverage manifest before release. A layer named without a count states that a test exists; a layer with a count states what the test has ruled out, and only the second is evidence.)

The implementation is heavily tested but not formally verified. The value of the carrier model is that tests and review can be attached to precise laws rather than to a vague assertion of universal equivalence.

11. Limits and Open Problems

The framework leaves several important boundaries explicit.

1. **Finite snapshots.** The Boolean complement and completeness result is relative to a finite, fixed universe. Streaming and continuously changing universes need temporal or incremental semantics.
2. **SQL coverage and row representation.** The implementation has a broad PostgreSQL-oriented surface, not a proof of complete PostgreSQL equivalence. Its internal row carrier is dynamic rather than fully typed and vectorized.
3. **Bag and tuple boundaries.** SQL bags and generalized set-valued identifier tuples are distinct. A complete semiring treatment of every physical SQL operator remains future work.
4. **Approximate indexes.** IVF and HNSW trade exactness for recall and latency. The algebra types their results, but only differential experiments establish recall for a parameterization and corpus.
5. **Calibration.** A probability-shaped output is not evidence of calibration. Distribution shift, candidate-pool changes, and embedding-model changes require renewed held-out evaluation.
6. **Graph complexity.** General pattern matching can be exponential, and unrestricted path enumeration can produce unbounded output. Automata limits and bounded traversal are physical safeguards, not changes to complexity theory.
7. **Effects and user functions.** Volatile or externally effectful functions restrict rewrite and retry safety. Their properties must be declared and enforced at registration.
8. **Distributed execution.** Partitioned state, network failure, distributed joins, and consistency across nodes are outside the current core.
9. **Security.** Encryption and authenticated storage have separate threat models. Logical algebra does not prove confidentiality, key management, or rollback resistance.
10. **Per-node observation classification.** Theorem 8.1 requires the parent's observation as a hypothesis, and the implementation supplies it by per-operator declaration. A rewrite is therefore rejected whenever the *node* is payload-bearing, even when the *parent* demands only support. The classification is sound but not complete.

Promising future work includes a top-down demanded-observation analysis that propagates the required obs level from the plan root and discharges the hypothesis of Theorem 8.1 by derivation rather than declaration; an effect system for volatility and transaction behavior; a fully positional typed row carrier; semiring annotations over generalized tuples; incremental view and graph maintenance; proof-producing rewrite registration; and distributed carrier-preserving execution.

12. Conclusion

Heterogeneous query execution does not require a false choice between one universal container and a collection of unrelated engines. A typed carrier algebra provides a stronger middle ground.

Document support is Boolean. Weighted relations are semiring-valued. Decorated postings preserve operational payloads without inheriting laws they do not satisfy. Ranking owns order and truncation. SQL retains bags and null semantics. Joins retain tuples. Graph results retain graph context. Probabilistic fusion distinguishes prior-free evidence from priors and separates exact inference from useful heuristics.

These carriers can still participate in one system because their operators are typed, their adapters are explicit, their rewrites are law-indexed, and their plans share one exhaustive execution hierarchy. The current UQA Engine implementation demonstrates that this is not merely a taxonomy: the distinctions determine concrete optimizer guards, score types, graph codecs, output variants, top-k fallbacks, transaction boundaries, and validation tests.

The central claim is therefore substantial but precise: a single embedded runtime can compose relational, lexical, vector, graph, and probabilistic queries under one mathematical and physical planning framework, provided that unification preserves rather than erases the semantics of each carrier.

References

1. Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.
2. Abo Khamis, M., Ngo, H. Q., and Rudra, A. (2016). FAQ: Questions asked frequently. In *Proceedings of PODS 2016*, 13–28. https://doi.org/10.1145/2902251.2902280
3. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J. L., and Vrgoč, D. (2017). Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5), Article 68. https://doi.org/10.1145/3104031
4. Birkhoff, G. (1967). *Lattice Theory*, 3rd ed. American Mathematical Society.
5. Bonifati, A., Fletcher, G., Voigt, H., and Yakovets, N. (2018). *Querying Graphs*. Morgan & Claypool.
6. Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1), 1–3.
7. Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. (2003). Efficient query evaluation using a two-level retrieval process. In *Proceedings of CIKM 2003*, 426–434.
8. Buneman, P., Naqvi, S., Tannen, V., and Wong, L. (1995). Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1), 3–48. https://doi.org/10.1016/0304-3975(95)00024-Q
9. Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387.
10. Ding, S., and Suel, T. (2011). Faster top-k document retrieval using block-max indexes. In *Proceedings of WSDM 2011*, 993–1002.
11. Fagin, R., Lotem, A., and Naor, M. (2003). Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4), 614–656. https://doi.org/10.1016/S0022-0000(03)00026-6
12. Golan, J. S. (1999). *Semirings and Their Applications*. Kluwer Academic Publishers.
13. Graefe, G. (1994). Volcano—an extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering*, 6(1), 120–135.
14. Gray, J., and Reuter, A. (1992). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann.
15. Green, T. J., Karvounarakis, G., and Tannen, V. (2007). Provenance semirings. In *Proceedings of PODS 2007*, 31–40.
16. Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of ICML 2017*, 1321–1330.
17. Jeong, J. (2023). *A Unified Mathematical Framework for Query Algebras Across Heterogeneous Data Paradigms*. OSF Preprints. https://doi.org/10.31219/osf.io/f56j2_v2
18. Jeong, J. (2024). *Extending the Unified Mathematical Framework to Support Graph Data Structures*. OSF Preprints. https://doi.org/10.31219/osf.io/cgfae_v1
19. Jeong, J. (2026). *A Unified Bayesian Framework for Hybrid Search: Calibration and Log-Odds Fusion of Lexical and Vector Retrieval*. Zenodo. https://doi.org/10.5281/zenodo.20768747
20. Jeong, J. and Cognica, Inc. (2026). UQA Engine 0.1.0 preproduction software artifact. https://github.com/cognica-io/uqa-engine

21. Joglekar, M. R., Puttagunta, R., and Ré, C. (2016). AJAR: Aggregations and joins over annotated relations. In *Proceedings of PODS 2016*, 91–106. https://doi.org/10.1145/2902251.2902293
22. Kepner, J., Aaltonen, P., Bader, D. A., Buluç, A., Franchetti, F., Gilbert, J. R., Hutchison, D., Kumar, M., Lumsdaine, A., Meyerhenke, H., McMillan, S., Yang, C., Owens, J. D., Zalewski, M., Mattson, T. G., and Moreira, J. E. (2016). Mathematical foundations of the GraphBLAS. In *Proceedings of IEEE HPEC 2016*, 1–9. https://doi.org/10.1109/HPEC.2016.7761646
23. Libkin, L., Martens, W., and Vrgoč, D. (2016). Querying graphs with data. *Journal of the ACM*, 63(2).
24. Mac Lane, S. (1998). *Categories for the Working Mathematician*, 2nd ed. Springer.
25. Malkov, Y. A., and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824–836.
26. Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
27. Mendelzon, A. O., and Wood, P. T. (1995). Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6), 1235–1258.
28. Mitzenmacher, M., and Upfal, E. (2017). *Probability and Computing*, 2nd ed. Cambridge University Press.
29. Moerkotte, G., and Neumann, T. (2006). Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *Proceedings of VLDB 2006*, 930–941.
30. Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, 61–74. MIT Press.
31. Robertson, S. E., and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4), 333–389.