

# The Living Knowledge Layer

Knowledge-as-Code for Governed LLM-Driven Enterprise Analytics

Olivier Nachba

Operator and enterprise data architecture practitioner

May 2026 — Industrial position paper, version 2.6 (final)

## Abstract

Most enterprise data stacks now make data accessible. They do not make it safe to interpret.

That distinction is where many modern analytics failures live. In production, the expensive mistakes are often not broken pipelines or failed SQL queries. They are plausible answers generated from incomplete operational knowledge. The SQL runs. The dashboard renders. The meeting happens. The decision is wrong.

This paper introduces the *Living Knowledge Layer* (LKL), a governed, versioned, and injectable repository of operational knowledge for enterprise analytics. The LKL captures rules, contexts, traps, mappings, join patterns, exceptions, precedence rules, and cross-domain relationships that are usually stored in people, meetings, wiki pages, tickets, and undocumented analyst habits.

The LKL is designed for LLM-driven analytics, but it is not just prompt context. A mature LKL acts as a control plane. Validated entries are retrieved and injected before an LLM generates SQL or explanations. Mandatory entries can also compile into validators, forbidden-pattern checks, table-selection constraints, refusal conditions, and audit trails. The goal is not to trust the model more. The goal is to reduce how much trust the model needs.

The semantic layer remains necessary. It standardizes metrics, dimensions, entities, and joins. The claim of this paper is narrower: semantic layers do not, by themselves, capture the full operational knowledge required to prevent silent analytical failures. The LKL complements the semantic layer by making operational knowledge explicit, reviewable, testable, and applicable at runtime.

This paper makes five contributions. First, it reframes persistent enterprise analytical silos as knowledge-silo failures rather than data-centralization failures. Second, it reports a field observation from a twelve-month industrial deployment in a large multinational organization. Third, it defines the LKL as an architectural pattern with taxonomy, lifecycle, governance, security, runtime behavior, and automated controls. Fourth, it proposes an evaluation protocol for measuring whether curated operational knowledge improves text-to-SQL correctness, rule adherence, and silent-failure prevention. Fifth, it describes the operating model required to maintain the layer over time.

The deployment produced 235 knowledge entries, including 73 expert-curated entries and 23 join patterns. In production observation, LKL injection appeared to materially improve SQL reliability on intermediate-complexity questions. These observations are feasibility evidence, not yet a controlled benchmark. The next step is formal evaluation against schema-only, semantic-layer-only, raw-documentation RAG, LKL, and LKL-plus-control baselines.

**Keywords:** enterprise data architecture; semantic layer; knowledge layer; generative AI; large language models; text-to-SQL; data governance; Knowledge-as-Code; RAG; GraphRAG; business glossary; organizational learning; industrial case study.

# 1 Introduction

## 1.1 Data access is not interpretation safety

For more than twenty years, enterprise data programs have promised to break organizational silos.

Data warehouses centralized transactional and analytical data. Data lakes expanded storage scope. Cloud migrations made compute elastic. The modern data stack made ingestion, transformation, testing, and BI more modular. Lakehouse architectures attempted to unify warehouse and lake patterns.

These investments were useful. They made data easier to store, process, govern, and query. They improved analytics inside individual domains. Finance, Supply Chain, Merchandising, Marketing, Operations, Risk, and Customer teams became faster inside their own boundaries.

But the core silo problem survived.

The reason is simple: most architectures centralized data, not interpretation. They moved tables into common platforms, but they did not move the tacit knowledge required to use those tables correctly. A finance analyst and an operations analyst can query the same table and produce two results that are both technically valid and mutually incompatible.

The remaining silo is cognitive.

## 1.2 The expensive failure mode is plausible wrongness

Modern data systems are good at detecting technical failure. Broken pipelines, failed jobs, missing columns, schema drift, permission errors, and syntax errors are visible. They create alerts.

The more expensive failure mode is different. It is plausible wrongness.

A query runs. It returns rows. The numbers look reasonable. The dashboard renders. No monitoring system raises an alarm. Only domain knowledge would reveal that the wrong table was used, a join multiplied revenue, a date field meant dispatch rather than arrival, or a month-to-date comparison was being made against a full prior-year month.

This is the failure class this paper addresses: the *silent analytical failure*.

## 1.3 The proposed response

The Living Knowledge Layer is a structured way to capture and apply the knowledge that prevents these failures.

An LKL entry is not a wiki note. It is a typed object with metadata, status, owner, scope, evidence, references, tests, and security classification. Entries are stored in readable files, reviewed through pull requests, validated by domain owners, indexed for retrieval, injected into LLM workflows, and used to test generated outputs.

The practical idea is simple:

*When the organization learns that a table, rule, join, field, metric, mapping, or business context is dangerous to misunderstand, that learning should become durable operational infrastructure.*

## 1.4 Contributions

This paper makes five contributions.

1. **Problem formulation.** It reframes persistent enterprise analytical silos as knowledge-silo failures rather than data-centralization failures.
2. **Field evidence.** It reports observations from a twelve-month industrial deployment in a large multinational organization, including entry counts, failure patterns, and operational lessons.
3. **Architectural pattern.** It defines the Living Knowledge Layer as a structured, living, governed, injectable, and testable layer of operational knowledge.
4. **Runtime model.** It describes how the LKL acts not only as prompt context, but also as a control plane for SQL validation, conflict detection, security filtering, observability, and refusal.
5. **Evaluation protocol.** It proposes a controlled methodology to test whether LKL injection improves SQL correctness, rule adherence, and silent-failure prevention compared with relevant baselines.

## 1.5 Scope and non-goals

The LKL is not a replacement for warehouses, lakehouses, data catalogs, semantic layers, BI tools, or knowledge graphs. It is an overlay.

The LKL is not a claim that LLMs understand the business. LLMs remain unreliable without retrieved context, validation, guardrails, and observability.

The LKL is not an attempt to formalize all tacit knowledge. Some expertise remains human, situated, and hard to encode. The target is narrower: capture the formalizable operational knowledge whose absence causes repeated analytical errors.

The LKL is not a first investment for immature data organizations. It becomes useful after the organization has enough data-stack maturity for interpretation errors to become a bottleneck.

## **2 Field observation: what triggered the LKL**

### **2.1 The production context**

The case behind this paper comes from a twelve-month deployment in a large multinational organization operating across multiple business domains. Exact names, internal tables, geography, and financial details are anonymized for confidentiality.

The environment had the usual characteristics of a mature enterprise data estate: a cloud data warehouse; more than one hundred billion rows across analytical projects; multiple legacy systems and migration artifacts; several functional domains with their own reporting habits; competent analysts and engineers; business experts with deep local knowledge; and an LLM-based conversational interface for analytical access.

The data was not inaccessible. The problem appeared after access improved.

As more users asked questions through the conversational interface, the system surfaced a pattern: technically valid answers were sometimes wrong because the model, and often the analyst, lacked operational knowledge that lived outside the formal data stack.

### **2.2 The recurring scene**

The recurring scene was simple. A result was shown to a domain expert. The expert reacted immediately: “Those numbers are wrong.” “You used the wrong table.” “That field does not mean what you think.” “This site is a special case.” “That changed after the migration.” “You cannot compare this month to last year that way.”

Each sentence pointed to knowledge that existed in the organization but was not available to the analytical system at the moment of use.

The original response was manual: investigate, correct the query, explain the issue, move on. The LKL emerged from a different response: investigate, correct the query, capture the learning as an entry, validate it, index it, inject it next time, and test for it.

### **2.3 Failure gallery**

The failures below are anonymized, but the pattern is representative.

### **2.4 What made the failures invisible**

The failures shared five properties: (1) the data was accessible; (2) the SQL executed; (3) the output looked plausible; (4) existing monitoring did not detect the error; (5) a domain expert could detect it quickly.

This is the operational signature of a knowledge silo.

| Failure                        | Why it looked correct                                 | Hidden knowledge  | LKL entry created              | Prevented |
|--------------------------------|---|---|--------------------------------|-----------|
| Partial operational-flow table | Name and fields matched the question                  | A migration had made another table canonical; the old table covered only part of the volume | Mandatory canonical-table rule | Yes       |
| Invalid month comparison       | The prior-year month existed and the query executed   | Current month was partial; comparison had to be MTD vs MTD at equal date                    | Equal-date comparison rule     | Yes       |
| Supplier revenue omission      | The parent supplier code looked like the right filter | Commercial activity was split across several sub-codes                                      | Supplier mapping entry         | Yes       |
| Join amplification             | The join returned plausible rows                      | Upstream key was duplicated; deduplication required before aggregation                      | Join pattern and SQL check     | Yes       |
| Channel stock invisibility     | Stock table was technically correct for one channel   | Another fulfillment path existed but was not visible in that reporting scope                | Network-context entry          | Yes       |

**Table 1:** Failure gallery from production. None of these failures was a beginner mistake. Each was a normal failure in a complex enterprise where operational knowledge had not been made machine-usable.

## 2.5 The operator lesson

The field lesson was not that the model was bad. The model was doing what analysts often do under the same conditions: infer too much from schema and too little from operational context.

The lesson was more structural:

*If operational knowledge is not captured as infrastructure, every analyst and every model will rediscover it through mistakes.*

The LKL is the attempt to stop paying that cost repeatedly.

## 3 Problem formulation: enterprise silos as knowledge silos

### 3.1 Definition

A **data silo** exists when data required for an analysis is inaccessible.

A **knowledge silo** exists when the data is accessible, but the knowledge required to interpret

it correctly is trapped inside a team, individual, tool, meeting, or informal practice.

Many mature organizations have reduced data silos without reducing knowledge silos. This is why centralization can coexist with inconsistent analytics.

### 3.2 Silent analytical failure

A **silent analytical failure** has four properties: (1) the query runs successfully; (2) the result looks plausible; (3) no technical monitoring system flags the result as wrong; (4) only domain knowledge would reveal the error.

Silent failures are more dangerous than visible failures. A failed query forces investigation. A plausible wrong query enters dashboards, decisions, forecasts, negotiations, replenishment logic, customer promises, and incentive systems.

### 3.3 Four cognitive dimensions of enterprise data

Enterprise analytical knowledge usually falls into four cognitive dimensions: *vocabulary*, *rule*, *context*, and *trap*.

**Vocabulary.** The same word can mean different things across domains. “Department,” “active customer,” or “revenue” may each carry several incompatible interpretations. No column name resolves this ambiguity by itself.

**Rule.** Business rules often sit outside the database. A return may count for Operations but not for Finance. A transaction may be valid for Marketing attribution but excluded from margin reporting. A site may be eligible for fulfillment only under specific category and margin constraints.

**Context.** Some facts require operational context. A warehouse may be physically a store, legally a warehouse, and operationally a dark store. A country may be rolled up under another entity for historical or tax reasons. A migration may have split logging across two tables. A fiscal month may not align with a calendar month.

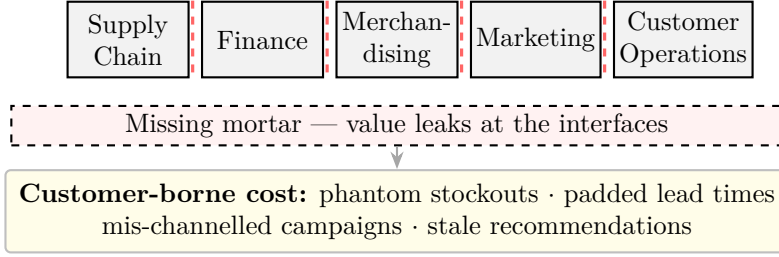
**Trap.** Mature databases contain traps. A field looks populated but is unreliable. A table looks canonical but covers only part of the process. A join looks obvious but duplicates rows. A code table contains two historical coding systems. Traps are often the most valuable knowledge in an enterprise because they prevent silent failures.

### 3.4 The wall metaphor

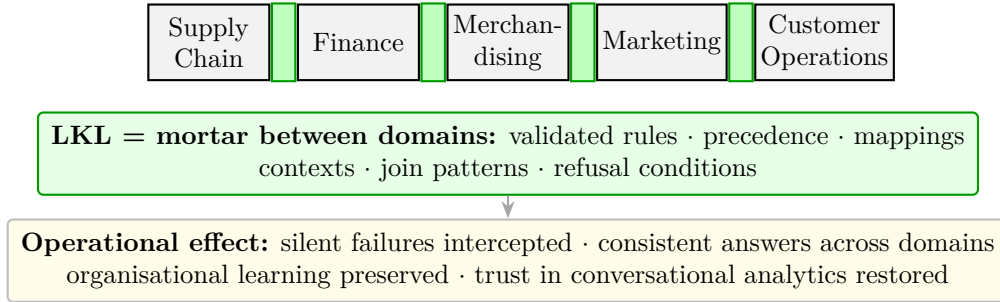
A mature enterprise is a wall. Each silo is a stone: Supply Chain, Finance, Merchandising, Marketing, Operations, Customer, Risk. Twenty years of investment have produced remarkable stones. Pipelines run. Dashboards render. Local analytics are fast.

But a wall does not hold because its stones are polished. It holds because mortar binds them.

The missing mortar is cross-domain operational knowledge: what one domain knows that another domain needs in order to interpret data safely.



**Figure 1:** The wall of the data organization: polished stones, missing mortar. Each silo (Supply Chain, Finance, Merchandising, Marketing, Customer Operations) has been optimized inside itself by twenty years of investment. The mortar between them — the cross-domain knowledge that should bind them — has never been laid. The residual cost surfaces at the interfaces, and is largely borne by the end customer.



**Figure 2:** The wall with mortar applied: a Living Knowledge Layer between domains. The stones are unchanged — the same warehouses, semantic layers, dashboards, and pipelines. What changes is that the cross-domain operational knowledge has been captured, validated, and is applied at the moment a question is asked. The contrast with Figure 1 is the contrast between “the data is accessible” and “the data is safe to interpret.”

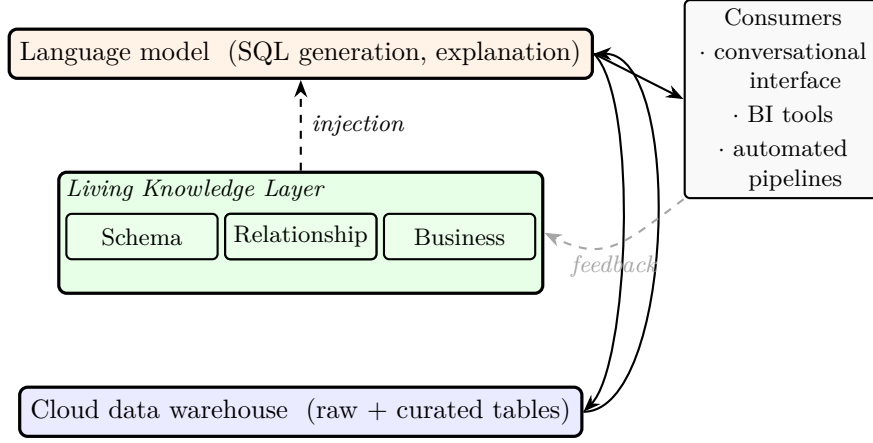
The two states shown in Figures 1 and 2 summarize a recurring pattern across enterprise data programs. In State A (Figure 1), the stones are polished but the bonds are absent: definitions collide at the interfaces, operational context is lost between domains, and the visible consequences — phantom stockouts, padded lead times, wrong campaigns, stale recommendations, bad promises, weak negotiations — accumulate outside the boundary of any single dashboard.

In State B (Figure 2), the stones are unchanged. The same warehouses, semantic layers, dashboards, and pipelines remain in place. What changes is that the cross-domain operational knowledge has been captured as validated rules, precedence entries, mappings, contexts, join patterns, and refusal conditions, and that this knowledge is applied at the moment a question is asked. The contrast between the two states is the contrast between “the data is accessible” and “the data is safe to interpret.”

The Living Knowledge Layer is one way to produce that mortar deliberately: not as side documentation, but as governed, versioned, and injectable knowledge that travels with the data at the moment of analysis. The remainder of this paper specifies how.

### 3.5 Requirements for a solution

A solution to knowledge silos must satisfy eight requirements: capture operational knowledge (R1); make it structured (R2); make it living (R3); make it injectable (R4); govern validation



**Figure 3:** Conceptual architecture of the Living Knowledge Layer. The LKL is layered between the data and its language-model consumer. It injects contextual knowledge (rules, contexts, traps, mappings) into each prompt before the model acts, and absorbs feedback from downstream consumers to grow over time. Its three sub-layers — schema intelligence, relationship intelligence, business intelligence — correspond to three distinct questions an analyst asks of the data.

(R5); handle conflict (R6); preserve auditability (R7); respect security (R8). The Living Knowledge Layer is designed around these requirements.

## 4 The Living Knowledge Layer

### 4.1 Definition

A **Living Knowledge Layer** is a structured, governed, versioned, and programmatically injectable repository of operational knowledge required to use enterprise data correctly.

The definition is deliberately operational, not theoretical. It is what an LKL has to be in order to survive contact with a real enterprise: not a wiki of good intentions, but an artifact that can be reviewed, tested, and applied at the moment a question is asked.

The definition has four key properties. **Structured:** entries are typed objects with metadata, owners, scopes, references, lifecycle states, and tests. **Living:** entries change as systems, processes, definitions, and organizational knowledge evolve. **Governed:** binding entries require expert validation. Disagreement is visible and handled through explicit processes. **Injectable:** entries are retrieved and applied in analytical workflows at runtime, especially LLM-driven workflows.

### 4.2 Relationship to existing layers

The LKL sits above data infrastructure and alongside, not instead of, the semantic layer.

The semantic layer answers: *what are the official metrics, dimensions, entities, and joins?* The LKL answers: *what must be known to use those objects correctly in this context?*



### 4.3 Three sub-layers

**Schema intelligence** answers *what is this object really?* It enriches technical schema metadata with operational interpretation: table purpose and canonical status; field reliability and fill rate; nested-field access patterns; misleading labels; historical changes; data freshness constraints; safe and unsafe use cases.

**Relationship intelligence** answers *how does this connect?* It documents how entities and tables should be joined in practice: correct join keys; required type casts; deduplication requirements; many-to-many amplification risks; temporal join validity; partition and cost warnings; wrong-but-plausible joins; validated SQL patterns. This layer prevents joins that execute successfully but change the meaning of aggregates.

**Business intelligence** answers *what does this mean in the business?* It captures knowledge that cannot be inferred from schema alone: operational role of a site; exception rules; business thresholds; country or brand conventions; cross-domain dependencies; migration history; strategic decisions that changed interpretation; metric precedence across domains; known limitations of reports or models. This is usually the most valuable and least documented knowledge.

### 4.4 Entry taxonomy

An LKL entry has a type. The following taxonomy is a practical starting point.

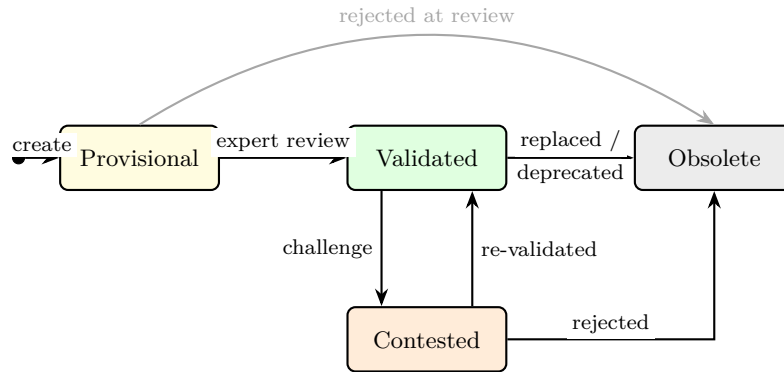
| Type              | Purpose                               | Example  |
|-------------------|---------------------------------------|--|
| <b>metric</b>     | Quantitative definition               | Net revenue definition for Finance.  |
| <b>glossary</b>   | Business term definition              | Meaning of “active customer” by domain.  |
| <b>rule</b>       | Binding analytical constraint         | Always deduplicate by transaction ID before aggregating.                         |
| <b>parameter</b>  | Configurable business threshold       | Minimum margin gate for cross-border fulfillment.                                |
| <b>mapping</b>    | Code-to-meaning translation           | Site codes to operational regions.   |
| <b>context</b>    | Background needed for interpretation  | Why one country rolls up under another entity.                                   |
| <b>cookbook</b>   | Validated analytical pattern          | SQL pattern for seven-day flow analysis.   |
| <b>schema</b>     | Enriched table or field documentation | Canonical status and limitations of a table.                                     |
| <b>module</b>     | Code or pipeline documentation        | Transformation module behavior.  |
| <b>join</b>       | Validated relationship pattern        | Correct join between sales and product hierarchy.                                |
| <b>precedence</b> | Conflict-resolution rule              | Finance revenue rule overrides Marketing attribution when reporting net revenue. |
| <b>test</b>       | Automated validation artifact         | Rule adherence test for generated SQL.   |

**Table 2:** Entry taxonomy. The list is meant to be extensible. What matters is that entries are typed, queryable, governed, and testable.

### 4.5 Lifecycle

Each entry has a lifecycle state: **provisional** (proposed but not validated), **validated** (approved by domain owner, eligible for runtime injection), **contested** (challenged and awaiting arbitration, removed from binding injection), **obsolete** (preserved for history but no longer

valid), or **rejected** (not accepted after review). The lifecycle is what makes the layer trustworthy enough to be applied automatically.



**Figure 4:** Lifecycle of a knowledge entry. Every entry is born *provisional* (suggested by an analyst or auto-extracted). Domain-expert review promotes it to *validated*, after which it is injected into model prompts. A challenge moves it back to *contested*; arbitration either re-validates it or marks it *obsolete*. Obsolete entries remain readable as archive but are no longer injected. This explicit lifecycle is what makes the LKL operationally trustworthy.

## 4.6 Knowledge-as-Code

The LKL should be maintained as code. A practical implementation stores entries as Markdown files with YAML frontmatter in a Git repository. This gives the LKL a familiar engineering workflow: creating an entry is a pull request; validation is a merge by a domain owner or CODEOWNER; disagreement is an issue or challenge request; deprecation is a commit; audit history comes from Git; completeness checks run in CI; production deployment updates the retrieval index.

This approach prevents the LKL from becoming another wiki. A wiki is read manually. The LKL is reviewed, tested, indexed, and applied.

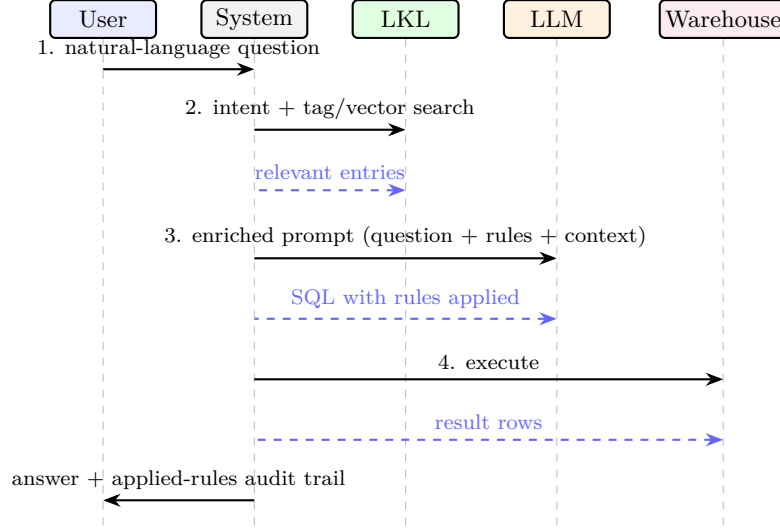
## 4.7 Minimum metadata and applicability scope

Every entry should include at least an `id`, `type`, `status`, `title`, `domain`, `owner`, `created_at`, `updated_at`, an `enforcement` level (`mandatory`, `recommended`, or `informational`), and a `security_classification`. Validated entries also require a `validator`. Recommended fields include `valid_from`, `valid_to`, `references_tables`, `references_entries`, `evidence_level`, and `tests`.

A common failure mode in knowledge systems is over-application. A true rule in one context becomes false when generalized. Each entry should therefore define an **applicability** scope: where it applies and where it does not. Applicability metadata is essential for safe injection.

## 4.8 When an LKL is premature

An LKL should not be the first investment of an immature data organization. It is premature when source data is unstable; when no shared warehouse or lakehouse exists; when business



**Figure 5:** The four-step contextual injection mechanism. A user question is parsed for intent; the LKL is queried for relevant entries; the LLM receives an enriched prompt that combines the question with the applicable rules and context; the resulting SQL is executed against the warehouse and the answer returned to the user with an auditable trail of which rules were applied.

experts are unavailable; when data permissions are unclear; when no one owns governance; or when the organization cannot name repeated silent failures. In these conditions, the LKL becomes documentation theater. The priority remains data foundations, ownership, access control, and basic data quality.

## 5 From prompt context to control plane

### 5.1 Overview

The first version of an LKL improves prompts. The mature version constrains outputs.

Mandatory entries should compile into validators, SQL checks, forbidden-pattern rules, table-selection constraints, cost controls, and refusal conditions. Otherwise, the system still relies too much on the model obeying instructions.

The critical design point is that retrieval, permission filtering, prompt composition, and validation all happen before the answer is trusted.

### 5.2 Runtime steps

The runtime pipeline has eight steps: (1) receive the user question in natural language or a structured form; (2) extract intent (domain, analysis type, entities, time window, concepts); (3) retrieve LKL entries by exact tag, table reference, domain, vector similarity, dependency expansion, and enforcement-priority ranking; (4) filter by user permission, including knowledge classification, before any prompt is composed; (5) compose the prompt with structured sections for mandatory rules, business context, join patterns, mappings, and the user question; (6) generate SQL or analytical output through the LLM; (7) validate the generated SQL through

syntax, table-permission, forbidden-table, mandatory-rule, join-pattern, cost, and dry-run checks; (8) execute, explain, and log, with an audit trail listing injected entries, applied rules, detected conflicts, assumptions, query version, and model version.

A simple principle should govern step 4:

*No user should receive knowledge context that grants more business visibility than their data permissions allow.*

### 5.3 Token-budget policy

Context windows are finite. The LKL must define what happens when too many candidate entries are retrieved. A deterministic priority cascade is safer than ad hoc truncation: mandatory rules are never evicted; security and access rules are applied outside the prompt composer; precedence rules are included whenever cross-domain conflict is possible; join patterns are included when relevant to candidate SQL paths; business contexts follow by relevance score and scope match; mappings, cookbooks, and glossary entries follow last and only when compact or directly relevant. The system should log both injected and evicted entries.

### 5.4 LKL as executable governance

We have learned this the hard way: a model that has been told a rule is not a model that follows the rule. Operationally, “told” and “enforced” are different system properties.

The LKL should not only inform prompts. It should also test outputs. A rule entry can carry generated-SQL checks that compile into runtime guards:

```
id: rule.flow_canonical_not_legacy
type: rule
status: validated
enforcement: mandatory
tests:
  - id: no_legacy_flow_table
    type: generated_sql_check
    forbidden_patterns:
      - "fact_flow_legacy"
    required_patterns:
      - "fact_flow_canonical"
  - id: uses_flow_date
    type: generated_sql_check
    required_patterns:
      - "flow_date"
```

The control plane should support at least six control types: *required pattern* (generated SQL must contain a specific table or column); *forbidden pattern* (generated SQL must not contain a specific table or column); *required filter* (a domain-specific filter must be present, e.g. Finance revenue excluding non-sale transactions); *join guard* (specific joins require deduplication or type-cast steps); *refusal condition* (if two mandatory rules conflict and no precedence rule

exists, refuse to answer); and *permission guard* (do not inject entries above the user’s knowledge classification). This is where the LKL becomes more than better documentation.

## 5.5 Conflict detection and arbitration

Cross-domain disagreement is not a bug. It is a central reality of enterprise data. Finance may define revenue net of late returns after a six-week window; Marketing may attribute completed transactions regardless of late returns; a user may ask for marketing-attributed revenue net of late returns. Both definitions are valid in their domains. Together, they create ambiguity.

The LKL handles this through three mechanisms. *Automatic collision detection* flags semantically overlapping constraints. *Precedence entries* explicitly resolve known conflicts. *Escalation* applies when no precedence rule exists: the system pauses, surfaces the conflict, and routes the issue to a knowledge steward or domain owners. The arbitration result becomes a new LKL entry.

## 5.6 Security model

The LKL can contain sensitive knowledge. Security cannot be an afterthought. At minimum, the architecture should include role-based access control, security classification on every entry, alignment between data permissions and knowledge permissions, prompt-time filtering before model calls, redaction of secrets, audit logs for accessed entries, prevention of prompt injection from untrusted documents, separate treatment of binding LKL entries and untrusted RAG context, and reviews for entries containing commercial, pricing, margin, legal, or operationally sensitive logic. Suggested classifications are **internal**, **restricted**, **confidential**, and **regulated**.

## 5.7 Observability and model portability

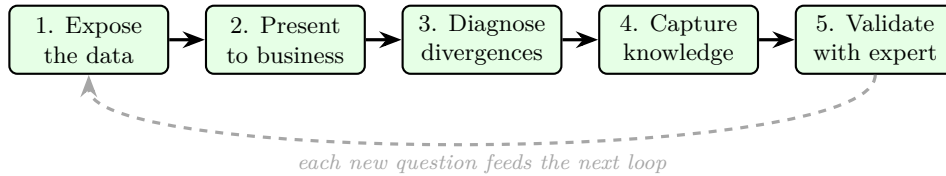
Every LKL-assisted answer should emit a structured log including: interaction identifier; user identifier hash; question hash; intent (domain, concepts); model (provider, name, version); retrieval (injected and evicted entries with version hash and enforcement); permissions (policy version, denied entries count); prompt hash; SQL (generated SQL hash, validation status, failed checks); execution (warehouse job identifier, row count, result hash); answer (applied rules, assumptions, warnings); and feedback (user rating, challenge identifier). Without this log, debugging becomes guesswork. With it, hallucinations and wrong answers become reproducible incidents.

The LKL provides knowledge portability across model generations because the knowledge itself is stored outside the model in readable formats. Runtime performance, however, depends on retrieval quality, context-window size, prompt design, tool-calling behavior, SQL generation ability, instruction-following reliability, latency, cost, and the security posture of the model provider. The precise claim is therefore that the LKL provides knowledge portability; runtime performance remains model-dependent and must be evaluated per model.

## 6 Construction process and operating model

### 6.1 Discovery loop

The LKL is not built once. It grows through repeated discovery.



**Figure 6:** The five-step discovery loop. Each iteration takes a divergence reported by a domain expert (step 2), investigates its cause (step 3), converts the diagnosis into structured knowledge entries (step 4), and validates them (step 5). The loop closes: next time a similar question is asked, the system applies the captured rule automatically. A mature LKL has traversed this loop dozens of times.

The loop has five steps: *expose* (make data queryable through dashboards, SQL, reports, or conversational interfaces); *present* (show results to domain experts and listen for objections); *diagnose* (investigate why the result conflicts with expert knowledge); *capture* (convert the diagnosis into one or more entries); *validate* (have domain owners approve entries before production injection).

The useful signal is often a sentence like: “You used the wrong table.” That sentence is a candidate control. In our deployment, almost every mandatory rule started as a short, exasperated objection from a domain expert in a meeting. The LKL is the discipline of treating those sentences as infrastructure inputs rather than as informal feedback.

### 6.2 Where to start

Start where three conditions overlap: high *expert density* (the domain has people who know the traps); *repeated questions* (captured rules will be reused often); and *high silent-failure cost* (wrong answers have visible consequences). Good initial domains often include Supply Chain, Finance, Operations, Risk, Compliance, and customer-promise logic. Avoid starting in a politically contested domain unless there is an executive sponsor and a clear arbitration mechanism.

### 6.3 First interviews, governance, and cadence

The first entries should come from structured expert interviews, not blank-page writing. Three questions are usually enough to seed a useful base of 15 to 30 entries: *What are the five worst analytical mistakes you have seen in this domain?* (output: traps, rules, cookbooks); *What are the five words or metrics people misunderstand most often?* (output: glossary, metrics, precedence); *What do you know that exists in no table?* (output: contexts, mappings, parameters).

Governance roles include *entry author*, *domain validator*, *knowledge steward*, *data engineer*, *security owner*, and *executive sponsor*. The key organizational shift is that business experts

become contributors to analytical infrastructure, not just consumers of reports.

A reasonable cadence is weekly review of provisional and contested entries; quarterly audits of validated entries and security classifications; monthly coverage-gap reviews; daily review of automated-test failures. An unmaintained LKL becomes dangerous: it can preserve wrong knowledge with more authority than a wiki.

## 6.4 Budget implications and role implications

The LKL shifts budget rather than simply reducing it. BI tool usage may decline in some use cases as conversational analytics absorbs ad hoc consumption. LLM inference becomes a recurring operational cost. Retrieval infrastructure adds a small line. Human validation is a real cost, not optional. Analyst repetitive work declines; error investigation declines in covered domains; governance effort increases. The business case should not rely only on license reduction. The larger value is fewer silent failures, faster decisions, and retained organizational knowledge.

The center of gravity of data roles shifts. The data engineer still builds pipelines but moves toward capturing the operational knowledge discovered while building and debugging them. The analyst moves from repetitive query production toward validation, diagnosis, and knowledge capture. The business expert becomes a validator and contributor — a real commitment that requires ownership, time, and accountability. The data steward expands from data quality to knowledge quality: lifecycle, review, freshness, conflict handling, and audit. In active domains, a dedicated knowledge steward may be needed.

# 7 Industrial case study

## 7.1 Context and deployed architecture

The case study comes from a twelve-month deployment in a large multinational organization operating across multiple business domains. Exact internal names, table names, geography, and financial details are anonymized for confidentiality. The analytical estate included a mature cloud data warehouse; more than one hundred billion rows across analytical projects; multiple operational domains; legacy systems and migration artifacts; business experts with deep local knowledge; and an LLM-based conversational interface for analytical access.

The deployed architecture combined a cloud data warehouse foundation; an LKL repository stored as structured files; a retrieval index over validated entries; LLM-based intent extraction and SQL generation; validation and SQL checks derived from mandatory entries; a conversational interface for business users; and a feedback loop from business validation into new entries.

At the time of reporting, the LKL contained 235 total entries, of which 73 expert-curated and 139 auto-extracted or enriched schema entries; 23 join patterns; 23 mandatory rules; 17 documented technical traps; 9 cookbooks; 11 configurable business parameters; covering 8 functional domains.

## 7.2 Definition of correct SQL

The phrase “correct SQL” is dangerous unless defined. In this paper, **correct SQL** means that all four conditions hold: (1) the query executes successfully; (2) it selects the intended canonical tables and fields; (3) it applies mandatory business rules and validated join constraints; (4) the result is judged semantically correct by a domain expert or validated reference answer.

Execution alone is not correctness.

## 7.3 Observed analytical reliability

In production observation, the LLM-based inference engine produced correct SQL on roughly 40 percent of intermediate-complexity questions without LKL injection. With systematic LKL injection, the observed rate exceeded 85 percent.

This result is important but must be interpreted carefully. It was not yet produced under the controlled protocol described in Section 8. The exact question set, sample size, judging method, and confidence intervals must be formalized before the number can be treated as a research benchmark.

The conservative interpretation:

*LKL injection appeared to materially improve SQL reliability in production, especially for questions requiring table choice, join constraints, business rules, and operational context.*

## 7.4 Representative discovery and impact chain

One discovery illustrates the pattern. An operational-flow dashboard was shown to a domain head. The expert immediately objected: volumes for a main operational site were too low. Investigation showed that the queried table captured only about 30 percent of actual volume. The remaining flow existed in another table that had become canonical after a historical migration, but this change was not documented where analysts could discover it. The investigation produced three LKL entries: a mandatory rule requiring the canonical table for operational-flow analysis; a rule clarifying date-field semantics; and a context entry explaining network topology and migration history.

The direct diagnostic effort was about two hours. The durable benefit was that future analysis in that scope no longer depended on the memory of a single expert.

The business impact was not created by the LKL — it was revealed by it. The chain ran: historical migration artifact → wrong table used for operational flow → understated capacity and volume → wrong operational decisions → phantom stockouts, inflated lead times, mis-sized buffers, SLA pressure → customer and margin impact. This pattern repeated across other documented traps. Each was small at the technical level and large at the operational level.



## 7.5 Identified loss pools and recovery distinction

The deployment identified operational loss pools whose annualized modeled value was in the hundreds of millions of USD. This must be stated precisely:

- the LKL did not create the value;
- the LKL did not recover the value;
- the LKL made previously hidden loss pools visible;
- recovery required separate operational action;
- theoretical actionability is not operational recovery.

A simple estimation frame is:

```
identified loss pool = affected volume
                      x estimated error rate
                      x unit margin or cost impact
                      x annualization factor
```

The exact figures may remain confidential, but the estimation chain should be auditable.

## 7.6 Cost reporting and qualitative effects

The deployment reported marginal external and infrastructure cost below USD 20,000, excluding internal team time. That exclusion matters. Internal time is likely the main cost of an LKL. Future reporting should separately track LLM inference, embedding and retrieval, warehouse execution, repository and CI/CD, expert interviews, data-engineering diagnosis, validation and arbitration, maintenance audits, and security review. A credible ROI model should include both marginal spend and human time.

Four qualitative effects were observed. *Faster business response*: non-trivial business questions that previously required days could often be answered in minutes within covered domains. *Fewer repeated mistakes*: known silent failures became less frequent once mandatory rules were validated and injected. *Shorter analyst onboarding*: new analysts could inspect the LKL instead of learning traps only through mistakes and mentorship. *Better business–tech collaboration*: business experts became contributors to durable analytical infrastructure rather than knowledge sources merely consumed in meetings.

## 7.7 What this case proves and does not prove

This case proves that an LKL can be built in a real enterprise context and can become operationally useful. It does not yet prove universal performance gains. It does not isolate the LKL from all other improvements. It does not provide a public reproducible benchmark. It does not prove that all organizations can achieve the same impact. Those claims require the evaluation protocol in Section 8.

## 8 Evaluation protocol

### 8.1 Why evaluation matters

Industrial observations are useful but are not enough. A paper claiming that the LKL improves analytical reliability must compare the LKL against baselines under a defined protocol. Without this, the strongest claims remain plausible but unproven. The industrial case in this paper should therefore be read in two ways: as feasibility evidence for building and operating an LKL, and as motivation for the controlled evaluation protocol below.

### 8.2 Research questions, baselines, and metrics

The protocol formulates six research questions: whether LKL injection improves SQL correctness compared with schema-only prompting; whether it improves rule adherence compared with a semantic layer alone; whether it reduces silent failures compared with raw documentation RAG; how LKL coverage affects performance across domains and complexity; what latency and cost trade-offs LKL injection introduces; and how often conflicts, missing entries, or permission filters prevent a safe answer.

The LKL should be compared against at least five baselines: a competent human analyst (B0); schema only (B1); semantic layer only (B2); raw documentation RAG (B3); LKL injection (B4); and LKL plus controls (B5).

Metrics should include execution accuracy, semantic correctness, rule adherence, silent-failure rate, conflict detection rate, refusal quality, latency, cost, explainability, and coverage.

### 8.3 Question set and judging

The evaluation set should be built before running the experiment, balanced along domain, complexity, knowledge requirement, failure risk, and LKL coverage. A minimum credible evaluation uses 100 to 150 questions; a stronger evaluation uses 300 or more. Questions should be written or validated by domain experts and must include adversarial cases where the obvious SQL path is wrong.

A robust judging protocol pairs every question across all baselines; uses the same model, temperature, and SQL execution environment for non-human baselines; hides baseline identity from judges where possible; uses at least two reviewers per question (one data expert, one domain expert); records disagreements and arbitrates through a third reviewer; reports confidence intervals, not only point estimates; and publishes prompts, retrieved entry IDs, SQL, and anonymized evaluation labels where confidentiality allows.

### 8.4 Statistical reporting and threats to validity

For each metric, the protocol reports sample size; mean and median where appropriate; confidence interval; per-domain breakdown; per-complexity breakdown; coverage sensitivity; and failure examples. The headline should not be only “accuracy improved.” It should be:

*On which questions, under which coverage conditions, against which baseline, with which uncertainty?*

Expected threats to validity include proprietary data limits on reproducibility; expert-judge bias toward existing practice; entry-writing leakage if not controlled; model changes during evaluation; prompt-engineering bias toward one baseline; overrepresentation of known traps; and confidentiality limits on full evidence publication. These threats do not invalidate the approach. They must be disclosed.

## 9 Related work

### 9.1 Enterprise data architecture

Classical data warehouse practice emphasized integrated, subject-oriented analytical data and dimensional modeling. Later data lake and lakehouse architectures expanded the range of stored data and separated storage from compute. The modern data stack added modular tooling: managed ingestion, SQL-based transformation, version-controlled modeling, testing, orchestration, cataloging, and BI. These developments improved data engineering practice but did not remove the need for business interpretation. In practice, a well-modeled warehouse still leaves many questions unresolved: which table should be used for a specific business intent; which edge cases are excluded by convention; which entity hierarchy applies for a given decision; which historical artifact must be ignored; which domain definition takes precedence when two definitions conflict. The LKL addresses this unresolved layer of interpretation.

### 9.2 Semantic layers

Semantic layers centralize business definitions such as metrics, dimensions, entities, and join relationships. They are present in BI tools, metrics layers, and emerging AI/BI platforms. Recent implementations are becoming more AI-aware: dbt’s Semantic Layer defines semantic models, metrics, dimensions, and entities in YAML [dbt-semantic-layer]; Snowflake Cortex Analyst uses semantic model YAML to help generate SQL from natural language, and Snowflake supports custom instructions for SQL generation in Cortex Analyst and semantic views [snowflake-cortex; snowflake-custom-instructions; snowflake-semantic-views]; Databricks AI/BI and Genie Spaces rely on business semantics and user feedback for natural-language interaction with enterprise data [databricks-ai-bi; databricks-genie].

This evolution matters. The LKL should not be positioned as “semantic layer versus knowledge layer.” A better framing is:

*The semantic layer defines shared analytical objects. The Living Knowledge Layer captures the operational knowledge required to use those objects safely across domains.*

In some platforms, parts of the LKL may eventually be absorbed into the semantic layer. The architectural distinction still matters because the lifecycle, governance, security, testing, and collision-handling requirements are broader than metric definition.

### 9.3 Text-to-SQL and LLM database interfaces

LLM-based text-to-SQL systems are promising but fragile in real enterprise settings. Spider helped the field evaluate semantic parsing over relational schemas [spider]. BIRD emphasized larger databases, dirty values, external knowledge, efficiency, and real-world professional domains [bird]. BIRD is especially relevant because it highlights a core point of this paper: schema alone is insufficient. Real-world text-to-SQL requires database values, external knowledge, and domain context. In enterprise settings, this context is often proprietary and cannot be learned from public pretraining. The LKL can be understood as a controlled way to provide this missing enterprise context at runtime.

### 9.4 Retrieval-augmented generation and GraphRAG

Retrieval-augmented generation combines language models with retrieved external knowledge [rag]. The LKL is mechanically a RAG pattern, but its corpus is different from ordinary documentation RAG. A typical RAG system indexes unstructured chunks from documents. The LKL indexes typed, reviewed, versioned, access-controlled knowledge entries with explicit lifecycle states.

GraphRAG extends RAG by constructing graph-based representations of entities and relationships, then using community summaries or graph traversal to answer broader questions [graphrag]. GraphRAG is complementary to the LKL. It can help discover implicit relationships, suggest missing links, and identify coverage gaps. But validated operational rules should not be treated as ordinary retrieved text. They require ownership, precedence, evidence, and tests.

### 9.5 Knowledge graphs, catalogs, glossaries, and organizational learning

Knowledge graphs represent entities and relationships in a structured graph [knowledge-graphs]. They are powerful but often expensive to design, populate, and maintain. Enterprise data catalogs document datasets, ownership, schemas, lineage, and quality metadata. Business glossaries define terms. The LKL overlaps with all three but has a narrower operational design center: lighter than a full knowledge graph; more governed and executable than a wiki; more contextual than a catalog; more operationally prescriptive than a glossary; more dynamic and AI-consumable than many semantic-layer deployments.

The LKL also belongs to the tradition of organizational learning. Organizations lose knowledge when experts leave, teams reorganize, systems migrate, and documentation rots. The LKL turns repeated analytical learning into an organizational asset. Its value is not only that it helps an LLM generate better SQL. Its deeper value is that it preserves how the organization learned to interpret its own data.

## 10 Limitations and risks

**Wrong knowledge becomes systematically wrong.** A wrong validated entry can be worse than no entry because it is applied repeatedly and with authority. Controls include an

explicit lifecycle, a challenge mechanism, periodic review, automated tests, impact monitoring, conservative enforcement levels, and expert accountability.

**Governance cost is real.** The LKL requires time from domain experts, data teams, and stewards. Organizations unwilling to fund maintenance should not deploy binding LKL injection in critical workflows.

**Culture can block the system.** If business experts do not trust the data team, refuse validation ownership, or lack time, the LKL will not grow. Tooling cannot fix a broken business–tech relationship.

**Tacit knowledge has a capture ceiling.** Not all expert judgment can be encoded. Some decisions require discussion, context, and human accountability. The LKL should reduce repetitive mistakes, not replace expert reasoning.

**LLMs still fail.** The model may ignore instructions, miscompose SQL, misunderstand ambiguity, overgeneralize rules, or fail on complex multi-step queries. LKL injection reduces one class of failure. It does not make LLM analytics safe by default.

**Security risk increases with useful knowledge.** The more useful the LKL becomes, the more sensitive it becomes. An entry about a margin gate, fraud rule, operational weakness, or supplier mapping may be more sensitive than a table schema.

**Knowledge drift.** Business rules change. Systems migrate. Sites open and close. Legal entities restructure. Entries age. A stale LKL becomes a source of errors.

**Incentive mismatch.** Domain experts may not be rewarded for capturing knowledge. Analysts may see the LKL as extra documentation work. Managers may count delivered dashboards but not prevented failures. The operating model must reward contribution.

## 11 Future work

The highest priority is a controlled benchmark using the protocol of Section 8, comparing schema-only, semantic-layer-only, raw documentation RAG, LKL, and LKL plus controls.

Future work should also include ablation studies isolating which entry types drive performance (rules, join patterns, contexts, mappings, glossary entries, cookbooks, precedence rules), since high-value knowledge may be concentrated in a small number of mandatory entries.

GraphRAG and knowledge-graph methods could help detect missing cross-domain links, identify isolated entries, suggest conflicting rules, cluster entries into domains, and surface stale or redundant knowledge.

A shared open schema for LKL entries would make tools interoperable. Sectoral libraries — common SQL anti-patterns, warehouse-specific performance traps, retail metrics glossary, finance reporting patterns, healthcare data safety rules, manufacturing downtime taxonomies — could grow on top of such a schema, provided governance prevents them from drifting into untrusted generic documentation.

Open research questions in runtime safety include how to prove a model obeyed mandatory rules; how to detect when retrieved context is insufficient; how to combine LKL entries with

raw RAG safely; how to quantify silent-failure reduction; and how to handle contradictions between expert-approved entries.

## 12 Conclusion

Enterprise data programs have made data more accessible. They have not made interpretation equally accessible. The persistent silo problem is therefore not only a data architecture problem. It is a knowledge architecture problem.

The Living Knowledge Layer addresses this gap. It captures operational knowledge as a governed, versioned, testable, and injectable asset. It complements the semantic layer rather than replacing it. It gives LLM-driven analytics context and controls required to avoid many plausible wrong answers. It also gives organizations a way to preserve analytical learning beyond individual experts.

The strongest form of the claim is not that every enterprise needs an LKL immediately. The claim is narrower and more testable:

*In mature data organizations where silent analytical failures are caused by uncaptured operational knowledge, a governed and injectable knowledge layer should improve analytical reliability, reduce repeated errors, and preserve organizational learning.*

The industrial case shows feasibility. The next step is controlled evaluation.

The next bottleneck in enterprise analytics is not access to data. It is access to validated operational knowledge at the moment of analysis. The Living Knowledge Layer is one way to make that knowledge operational.

## References

**Semantic layer and enterprise data architecture.** [dbt-semantic-layer] dbt Labs. “Semantic Layer configurations.” dbt Developer Hub, 2026. <https://docs.getdbt.com/reference/semantic-layer-reference>. [dbt-semantic-models] dbt Labs. “Semantic models.” dbt Developer Hub, 2026. <https://docs.getdbt.com/docs/build/semantic-models>. [snowflake-cortex] Snowflake. “Cortex Analyst.” Snowflake Documentation, 2026. <https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst>. [snowflake-custom-instructions] Snowflake. “Custom instructions in Cortex Analyst.” Snowflake Documentation, 2026. <https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst/custom-instructions>. [snowflake-semantic-views] Snowflake. “Specifying custom instructions in semantic views.” Snowflake release notes, 2026. [databricks-ai-bi] Databricks. “Databricks AI/BI.” Databricks Documentation, 2026. <https://docs.databricks.com/aws/en/ai-bi/>. [databricks-genie] Databricks. “What is a Genie Space?” Databricks Documentation, 2026. <https://docs.databricks.com/aws/en/genie/>. Kimball, R., and Ross, M. *The Data Warehouse Toolkit*, 3rd ed. Wiley, 2013. Inmon, W. H. *Building the Data Warehouse*, 4th ed. Wiley, 2005.

**Text-to-SQL and LLM database interfaces.** [bird] Li, J., Hui, B., Qu, G., et al. “Can LLM Already Serve as A Database Interface? A BIG Bench for Large-Scale Database Grounded Text-to-SQLs.” arXiv:2305.03111, 2023. [spider] Yu, T., Zhang, R., Yang, K., et al. “Spider: A Large-Scale Human-

Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task.” EMNLP, 2018.

**RAG, GraphRAG, and knowledge graphs.** [rag] Lewis, P., Perez, E., Piktus, A., et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.” NeurIPS, 2020. [graphrag] Edge, D., Trinh, H., Cheng, N., et al. “From Local to Global: A Graph RAG Approach to Query-Focused Summarization.” arXiv:2404.16130, 2024. [knowledge-graphs] Hogan, A., Blomqvist, E., Cochez, M., et al. “Knowledge Graphs.” ACM Computing Surveys, 2021. Noy, N., Gao, Y., Jain, A., et al. “Industry-scale Knowledge Graphs: Lessons and Challenges.” ACM Queue, 2019.

**Organizational learning and analytics culture.** Nonaka, I., and Takeuchi, H. *The Knowledge-Creating Company*. Oxford University Press, 1995. Senge, P. *The Fifth Discipline*. Doubleday, 1990. Davenport, T. H., and Harris, J. G. *Competing on Analytics*. Harvard Business Press, 2007.