



CANONICAL LIVING OVERVIEW GUIDE

Relational, Document, Graph, Vector, and Time-Series Databases

A database engineering guide comparing data models, transactions, indexing, query planning, replication, sharding, search, vector similarity, temporal data, operations, and polyglot persistence.

PERMANENT PUBLICATION IDENTITY

Relational, Document, Graph, Vector, and Time-Series Databases

Document ID
MYTHOS-FG-DAT-0001

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL L1-L4 Foundational through frontier	CLASS FIELD GUIDE Practical and educational	CADENCE QUARTERLY Interim updates as needed	EVIDENCE SOURCE-BOUND Limitations remain visible
--	--	--	---

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-SWE; MYTHOS-CLD; MYTHOS-AI
Audience	Data, software, platform, AI, analytics, and reliability engineers.
Prerequisites	Data modeling, SQL, distributed systems, storage, APIs, and application architecture.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

- Choose database models according to invariants, access paths, scale, consistency, and operations.
- Design schemas and indexes for real query workloads and lifecycle changes.
- Operate replication, partitioning, backup, restore, migration, and failure recovery.
- Use vector and graph retrieval without sacrificing authority, filtering, and transactional data.
- Govern polyglot persistence through ownership, contracts, evidence, and exit paths.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path

New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

How to use this guide	3
Contents	3
Chapter 01 - Database selection and workload modeling	5
Chapter 01 laboratory and review	10
Chapter 02 - Relational data and transactional integrity	11
Chapter 02 laboratory and review	16
Chapter 03 - Document databases and aggregate models	17
Chapter 03 laboratory and review	22
Chapter 04 - Graph databases and relationship-intensive state	23
Chapter 04 laboratory and review	28
Chapter 05 - Vector databases and similarity retrieval	29
Chapter 05 laboratory and review	34

Chapter 06 - Time-series and operational measurements	35
Chapter 06 laboratory and review	40
Chapter 07 - Replication, partitioning, and availability	41
Chapter 07 laboratory and review	46
Chapter 08 - Polyglot persistence and lifecycle governance	47
Chapter 08 laboratory and review	52
Integrated learning roadmap	53
Source authority register	54
Limitations, freshness, and revision control	55

CHAPTER 01

Database selection and workload modeling

Select data systems from concrete access, state, and failure requirements.

Chapter operating position

Select data systems from concrete access, state, and failure requirements.



1.1 Data and relationship shape

Data and relationship shape must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Identify entities, documents, edges, vectors, measurements, events, and temporal history. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for data and relationship shape.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating data and relationship shape as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for data and relationship shape.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.2 Access patterns and constraints

Access patterns and constraints must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define reads, writes, joins, traversals, nearest neighbors, ranges, aggregations, and invariants. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for access patterns and constraints.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating access patterns and constraints as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for access patterns and constraints.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.



1.3 Scale and distribution

Scale and distribution must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Estimate volume, velocity, cardinality, retention, locality, growth, partitions, and concurrency. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for scale and distribution.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating scale and distribution as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for scale and distribution.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.4 Operational and organizational fit

Operational and organizational fit must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Assess skills, tooling, support, licensing, portability, cloud, sovereignty, and cost. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for operational and organizational fit.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating operational and organizational fit as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for operational and organizational fit.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 01 laboratory and review

Practical laboratory

Create a workload model and select primary and secondary stores without starting from product preference.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore adaptive storage systems that choose representations and indexes per workload while preserving contracts.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Relational data and transactional integrity

Use schemas, constraints, SQL, and transactions to encode durable business invariants.

Chapter operating position

Use schemas, constraints, SQL, and transactions to encode durable business invariants.

2.1 Tables, keys, and normalization

Tables, keys, and normalization must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Design identities, references, normal forms, denormalization, generated values, and temporal columns. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for tables, keys, and normalization.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating tables, keys, and normalization as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for tables, keys, and normalization.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.2 Transactions and isolation

Transactions and isolation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Understand atomicity, consistency, isolation levels, locks, MVCC, anomalies, and retries. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for transactions and isolation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating transactions and isolation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transactions and isolation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.3 SQL and query planning

SQL and query planning must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use joins, subqueries, windows, statistics, plans, prepared statements, and execution evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for sql and query planning.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating sql and query planning as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for sql and query planning.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.4 JSON and hybrid relational models

JSON and hybrid relational models must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Combine typed columns with JSON documents, paths, indexes, and validation deliberately. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for json and hybrid relational models.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating json and hybrid relational models as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for json and hybrid relational models.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 02 laboratory and review

Practical laboratory

Model and test a transactional domain under concurrent updates, deadlocks, retries, and schema changes.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore declarative invariants spanning relational state, events, and external systems.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Document databases and aggregate models

Store flexible aggregates while preserving identity, atomicity, and evolution.

Chapter operating position

Store flexible aggregates while preserving identity, atomicity, and evolution.

3.1 Document boundaries

Document boundaries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose embedding or references according to atomic updates, size, locality, reuse, and lifecycle. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for document boundaries.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating document boundaries as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for document boundaries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.2 Schema validation and evolution

Schema validation and evolution must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Version fields, defaults, optionality, migrations, mixed versions, and readers. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for schema validation and evolution.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating schema validation and evolution as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for schema validation and evolution.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.3 Indexes, queries, and aggregation

Indexes, queries, and aggregation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Design compound, multikey, text, geospatial, and aggregation pipelines from workloads. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for indexes, queries, and aggregation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating indexes, queries, and aggregation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for indexes, queries, and aggregation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.4 Transactions and change streams

Transactions and change streams must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use multi-document transactions selectively and consume changes with resumability limits. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for transactions and change streams.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating transactions and change streams as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transactions and change streams.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 03 laboratory and review

Practical laboratory

Design a document aggregate, then test growth, concurrent updates, migration, and change-stream resume.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore verifiable document histories and schema-aware offline synchronization.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Graph databases and relationship-intensive state

Model and query connected domains where path structure is first-class.

Chapter operating position

Model and query connected domains where path structure is first-class.



4.1 Nodes, relationships, and properties

Nodes, relationships, and properties must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define identity, labels, types, direction, constraints, and temporal or weighted edges. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for nodes, relationships, and properties.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating nodes, relationships, and properties as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for nodes, relationships, and properties.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.2 Traversal and pattern queries

Traversal and pattern queries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use bounded paths, shortest paths, subgraphs, aggregation, and query-plan inspection. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for traversal and pattern queries.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating traversal and pattern queries as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for traversal and pattern queries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.



4.3 Causal clustering and consistency

Causal clustering and consistency must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle leaders, followers, routing, bookmarks, failover, and read-your-writes. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for causal clustering and consistency.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating causal clustering and consistency as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for causal clustering and consistency.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.4 Graph analytics and knowledge systems

Graph analytics and knowledge systems must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate transactional workloads from large graph algorithms, embeddings, and AI retrieval. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for graph analytics and knowledge systems.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating graph analytics and knowledge systems as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for graph analytics and knowledge systems.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 04 laboratory and review

Practical laboratory

Build a graph model for ownership and dependencies, then validate causal reads and failure behavior.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore temporal knowledge graphs carrying provenance and confidence on every edge.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Vector databases and similarity retrieval

Operate vector search as indexed approximate computation with metadata and authority.

Chapter operating position

Operate vector search as indexed approximate computation with metadata and authority.

5.1 Embeddings and distance

Embeddings and distance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose model, dimensions, metric, normalization, version, and compatibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for embeddings and distance.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating embeddings and distance as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for embeddings and distance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

⊕ 5.2 ANN indexes and tuning

ANN indexes and tuning must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Understand HNSW-style graphs, construction, search effort, recall, latency, memory, and rebuild. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for ann indexes and tuning.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating ann indexes and tuning as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for ann indexes and tuning.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.3 Payloads and filtering

Payloads and filtering must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Apply tenant, permissions, time, source, language, type, and domain filters before or during search. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for payloads and filtering.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating payloads and filtering as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for payloads and filtering.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.4 Updates and distributed operation

Updates and distributed operation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle insert, update, delete, compaction, replicas, shards, snapshots, and consistency. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for updates and distributed operation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating updates and distributed operation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for updates and distributed operation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 05 laboratory and review

Practical laboratory

Benchmark filtered vector retrieval under index, parameter, embedding, and distribution changes.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore hybrid late-interaction and graph-vector databases with source-authority-aware ranking.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Time-series and operational measurements

Store timestamped values while preserving identity, units, resolution, and retention.

Chapter operating position

Store timestamped values while preserving identity, units, resolution, and retention.

6.1 Series and cardinality

Series and cardinality must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define metric identity, labels, dimensions, tags, fields, and cardinality budgets. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for series and cardinality.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating series and cardinality as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for series and cardinality.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

⊕ 6.2 Ingestion and ordering

Ingestion and ordering must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle timestamps, clock skew, late data, duplicates, batches, compression, and backpressure. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for ingestion and ordering.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating ingestion and ordering as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for ingestion and ordering.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.3 Retention and downsampling

Retention and downsampling must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use partitions, chunks, tiers, rollups, continuous aggregates, and deletion. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for retention and downsampling.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating retention and downsampling as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retention and downsampling.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.4 Queries and alerting

Queries and alerting must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Support windows, rates, percentiles, seasonality, anomalies, and correlation with events. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for queries and alerting.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating queries and alerting as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for queries and alerting.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 06 laboratory and review

Practical laboratory

Design a telemetry schema and measure cardinality, compression, query cost, and late-data behavior.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore evidence-preserving telemetry that links aggregates to retained raw samples and provenance.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Replication, partitioning, and availability

Scale data while understanding consistency and failure domains.

Chapter operating position

Scale data while understanding consistency and failure domains.

7.1 Leader, quorum, and replica models

Leader, quorum, and replica models must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare synchronous, asynchronous, multi-leader, consensus, lag, and failover. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for leader, quorum, and replica models.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating leader, quorum, and replica models as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for leader, quorum, and replica models.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.2 Sharding and partition keys

Sharding and partition keys must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose distribution, routing, rebalancing, hotspots, locality, and tenant isolation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for sharding and partition keys.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating sharding and partition keys as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for sharding and partition keys.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.3 Consistency and client semantics

Consistency and client semantics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Specify linearizable, serializable, causal, snapshot, monotonic, and eventual expectations. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for consistency and client semantics.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating consistency and client semantics as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for consistency and client semantics.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.4 Failure and repair

Failure and repair must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle partitions, split brain, replica loss, corrupt state, resync, and disaster recovery. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for failure and repair.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating failure and repair as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for failure and repair.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 07 laboratory and review

Practical laboratory

Run a database failure matrix across replica loss, partition, lag, hotspot, and stale client routing.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore cross-region sovereign data fabrics with policy-bound replicas and verifiable consistency.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Polyglot persistence and lifecycle governance

Use multiple stores without creating unowned duplication and inconsistent truth.

Chapter operating position

Use multiple stores without creating unowned duplication and inconsistent truth.

8.1 System of record and derived stores

System of record and derived stores must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Identify authoritative writes, projections, search, caches, vectors, graphs, and analytics. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for system of record and derived stores.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating system of record and derived stores as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for system of record and derived stores.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.2 Change propagation

Change propagation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use CDC, outbox, streams, replay, reconciliation, and repair with stable identities. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for change propagation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
data_requirement:
  invariant: account_balance_never_below_reserved_floor
workload:
  writes_per_second: 1200
  read_patterns: [by_account, recent_transactions, fraud_graph]
stores:
  system_of_record: relational
  search_projection: distributed-search
  relationship_projection: graph
  semantic_projection: vector
rebuild:
  source: transaction-log-and-outbox
```

Failure patterns

Treating change propagation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for change propagation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.



8.3 Migrations and exit

Migrations and exit must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Move schemas, data, indexes, clients, workloads, and historical evidence safely. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for migrations and exit.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating migrations and exit as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for migrations and exit.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.4 Decision and review

Decision and review must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Record workload evidence, tradeoffs, limitations, owner, costs, security, and review triggers. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the domain invariant, data model, write, transaction, index, query, replication, backup, recovery, and application outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for decision and review.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating decision and review as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for decision and review.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 08 laboratory and review

Practical laboratory

Design a system using relational, search, graph, and vector stores with explicit ownership and rebuild paths.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop self-describing data platforms whose stores can be regenerated from authoritative state and evidence.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy

Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. PostgreSQL - PostgreSQL 18 Documentation

Role: Relational transactions, SQL, JSON, replication, indexing, and operations.

Verified: 2026-07-26

Official source: <https://www.postgresql.org/docs/current/>

02. MongoDB - MongoDB Database Manual

Role: Document model, transactions, replication, sharding, indexing, and operations.

Verified: 2026-07-26

Official source: <https://www.mongodb.com/docs/manual/>

03. MongoDB - Change Streams

Role: Real-time change events, resume tokens, filtering, and deployment scope.

Verified: 2026-07-26

Official source: <https://www.mongodb.com/docs/manual/changestreams/>

04. Neo4j - Neo4j Operations Manual - Clustering

Role: Graph clustering, causal consistency, routing, leadership, and operational management.

Verified: 2026-07-26

Official source: <https://neo4j.com/docs/operations-manual/current/clustering/>

05. Qdrant - Qdrant Documentation

Role: Vector collections, payloads, filtering, HNSW search, persistence, and distributed deployment.

Verified: 2026-07-26

Official source: <https://qdrant.tech/documentation/overview/>

06. OpenSearch - OpenSearch Documentation

Role: Distributed search, indexing, snapshots, lifecycle, observability, and recovery.

Verified: 2026-07-26

Official source: <https://docs.opensearch.org/latest/>

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-SWE; MYTHOS-CLD; MYTHOS-AI
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	databases, relational, document, graph, vector database, time series, PostgreSQL, MongoDB, Neo4j, Qdrant
Canonical route	/library/field-guides/mythos-fg-dat-0001/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.