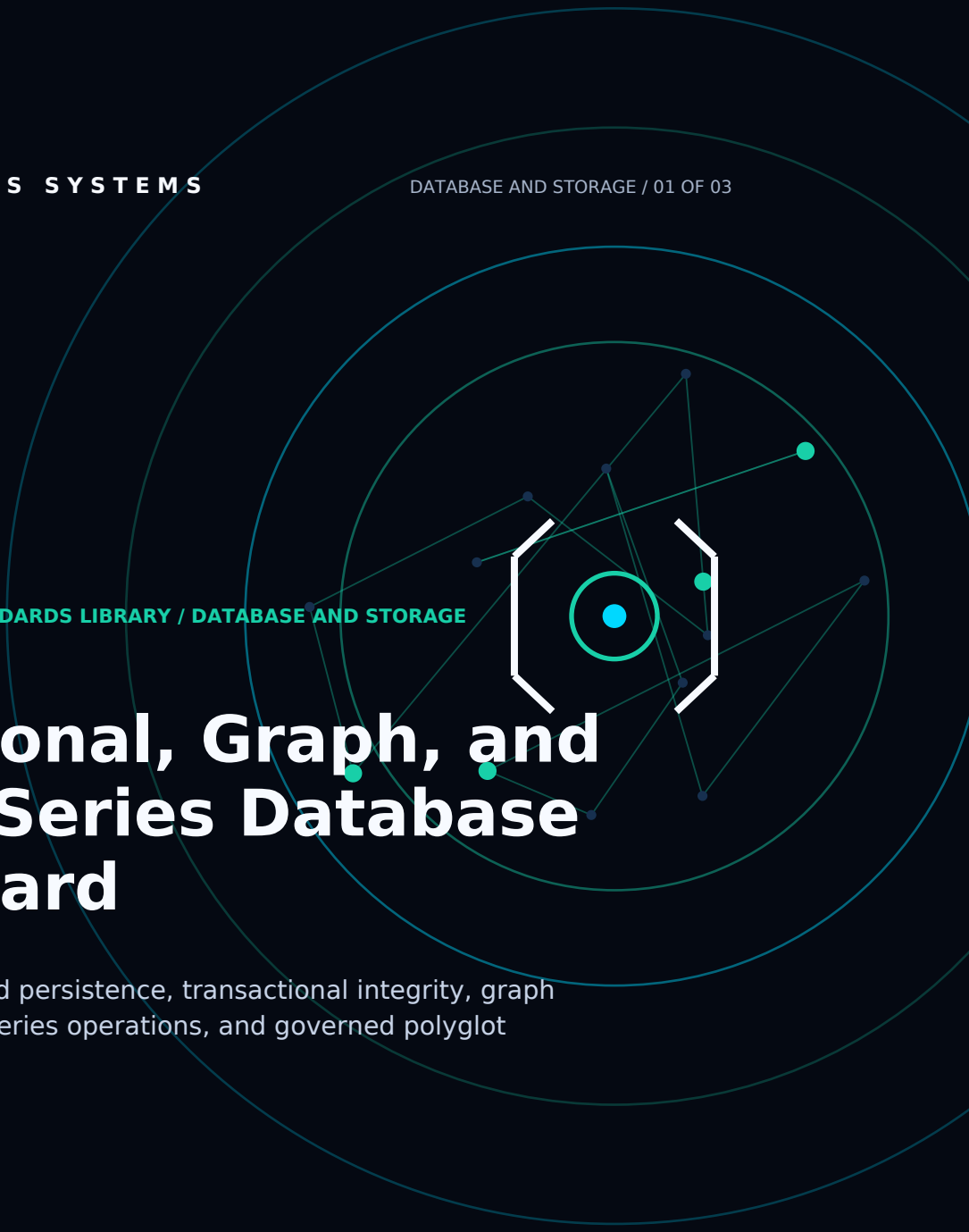


ENGINEERING STANDARDS LIBRARY / DATABASE AND STORAGE

Relational, Graph, and Time-Series Database Standard

Workload-aligned persistence, transactional integrity, graph traversal, time-series operations, and governed polyglot persistence.

PERMANENT DOCUMENT ID

MYTHOS-DBS-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Relational, Graph, and Time-Series Database Standard

DOCUMENT ID
MYTHOS-DBS-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

12	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable guidance or requirements for its defined scope. Interpretation must preserve the permanent document identity, recorded evidence, controlled exceptions, and formal revision history.

DOMAIN CONTROL SYSTEM

MYTHOS-DBS-0001 inside the database and storage standard constellation

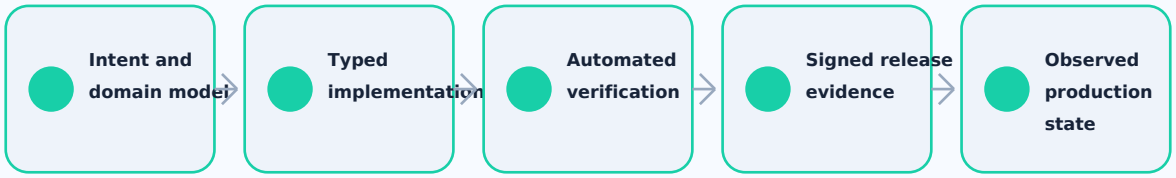
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Relational, Graph, and Time-Series Database Standard

The architecture of data persistence has failed.

Evidence-bearing delivery path



Assurance maturity spectrum

0	1	2	3	4	5
Absent	Ad hoc	Repeatable	Controlled	Measured	Adaptive

CONTENTS

Publication navigation

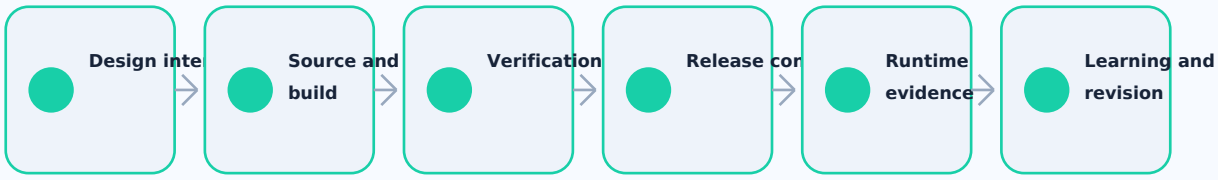
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
Polyglot Data Modeling (Weight: 20%)	1
Connection Pooling and Proxying (Weight: 20%)	1
High Availability and Failover (Weight: 15%)	1
Scalability and Sharding (Weight: 15%)	1
Tenant Isolation (RLS) (Weight: 15%)	1
Backup and Point-in-Time Recovery (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Polyglot Data Modeling (Weight: 20%)	Data Shape Alignment
2	Connection Pooling and Proxying (Weight: 20%)	Resource Management
3	High Availability and Failover (Weight: 15%)	Resilience
4	Scalability and Sharding (Weight: 15%)	Write Throughput
5	Tenant Isolation (RLS) (Weight: 15%)	Security Boundaries
6	Backup and Point-in-Time Recovery (Weight: 15%)	Data Durability
7	Suite Composition	MDBS-Pool
8	Suite Composition	MDBS-Isolation
9	Additional Evaluation Dimension	Persistence Dimensions
10	Additional Evaluation Dimension	The Database Doctrine
11	Additional Evaluation Dimension	Scoring Model
12	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The architecture of data persistence has failed.

Organizations force highly connected social graph data into relational tables, attempting deep traversals via recursive JOINs that cripple the CPU. They push high-cardinality IoT telemetry into standard relational databases, watching the index bloat until the system grinds to a halt. They allow microservices to open thousands of direct database connections, causing "connection storms" that exhaust database memory and crash the cluster. Furthermore, they rely on single-node databases with manual backup scripts, accepting hours of data loss (RPO) when a disk fails.

This standard exists because:

1. One Database Cannot Serve All Data Shapes: A relational database is optimized for set-based operations, not deep relationship traversals. A time-series database is optimized for high-write throughput and time-bucketed aggregations, not ACID transactions. Systems MUST adopt polyglot persistence, matching the data shape to the database engine.
2. Direct Connections are a Memory DoS: A microservice cluster spinning up 100 pods with 10 connection pools each will attempt to open 1,000 direct connections to the database. The database will spend 80% of its CPU managing connection state instead of executing queries. Systems MUST implement connection pooling proxies (e.g., PgBouncer, ProxySQL) to multiplex thousands of application connections into tens of database connections.
3. Single-Node Scaling is a Dead End: A monolithic database that scales vertically (bigger AWS instance) hits a hard hardware ceiling. Systems MUST implement horizontal scalability via sharding or distributed SQL (e.g., Vitess, Citus, CockroachDB) for write-heavy workloads.
4. Tenant Isolation Must Be Enforced by the Database: Trusting the application layer to filter data by `tenant_id` guarantees a cross-tenant data breach when a developer forgets a `WHERE` clause. Systems MUST enforce Row-Level Security (RLS) at the database engine level.

This document establishes the Mythos Database Standard (MDBS), a comprehensive, evidence-based methodology for evaluating database architectures against polyglot modeling, connection hygiene, and horizontal resilience.

Scope and Applicability

In Scope

This standard covers the evaluation of:

- Polyglot Persistence: Relational (PostgreSQL), Graph (Neo4j), Time-Series (InfluxDB)
- Connection pooling and proxy architectures (PgBouncer, Odyssey, ProxySQL)
- Horizontal scalability: Sharding, distributed SQL, and read replicas
- High Availability (HA): Synchronous replication, automated failover (Patroni)
- Tenant isolation: Row-Level Security (RLS) and database-level multi-tenancy
- Backup and recovery: Point-in-Time Recovery (PITR) and immutable backups

Out of Scope

- Vector databases and local-first sync (covered in MYTHOS-DAT-0001)
- Event sourcing and CQRS patterns (covered in MYTHOS-DAT-0004)
- General API and service mesh security (covered in MYTHOS-DST-0002)

Audience

- Database administrators (DBAs) and data architects
- Platform engineers managing stateful Kubernetes workloads
- Backend engineers designing microservice data stores
- Security engineers evaluating tenant isolation and data exfiltration risks
- Standards bodies seeking a reference database methodology

Definitions and Taxonomy

Persistence Dimensions

Dimension	Definition
Polyglot Persistence	The architectural paradigm of using different database technologies to handle different data storage needs within a single system (e.g., Postgres for transactions, Neo4j for relationships, InfluxDB for metrics).
Connection Multiplexing	The use of a proxy (e.g., PgBouncer) to maintain a small pool of active database connections, routing thousands of lightweight client requests over them to prevent database memory exhaustion.
Sharding	The process of horizontally partitioning large datasets into smaller, distributed blocks (shards) across multiple database nodes to parallelize write and read throughput.
Row-Level Security (RLS)	A database feature that restricts which rows a user can access based on a policy defined within the database itself, rather than relying on application logic.
Point-in-Time Recovery (PITR)	A backup strategy that restores a database to its state at a specific moment in time using continuous write-ahead log (WAL) archiving.

The Database Doctrine

> A recursive JOIN is not a graph traversal. An unbounded connection pool is a self-inflicted DoS. An application-level `tenant_id` filter is a breach waiting to happen. If the database cannot scale horizontally, the application cannot scale at all.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; monolithic DB, direct connections, single-node, no RLS
1	Basic read replicas, but relies on vertical scaling and app-level tenancy
2	Functional polyglot setup, but lacks connection pooling or sharding
3	Solid production performance; Polyglot, PgBouncer, automated HA/replicas, RLS enforced
4	Strong performance; Distributed SQL/Sharding, PITR backups, formally verified RLS
5	Best-in-class; sets the standard for mathematically verified, zero-downtime data persistence

Weighting

Category	Weight	Rationale
Polyglot Data Modeling	20%	Forcing all data into SQL causes catastrophic performance failures
Connection Pooling & Proxying	20%	Direct connections from microservices exhaust database memory
High Availability & Failover	15%	Single-node databases are a critical single point of failure
Scalability & Sharding	15%	Vertical scaling hits hardware limits; horizontal is required
Tenant Isolation (RLS)	15%	App-level filtering guarantees cross-tenant data leaks
Backup & Point-in-Time Recovery	15%	Snapshot backups guarantee data loss between backups

Total: 100%

A system MUST score at least 3.0 in Connection Pooling and High Availability to be considered for production use.

Evaluation Categories

Polyglot Data Modeling (Weight: 20%)

Data Shape Alignment

Is the database engine matched to the data structure and access pattern?

Score	Criteria
0	All data (transactions, relationships, telemetry) forced into a single MySQL/Postgres instance
1	Graph data stored in relational tables with heavy JOINs; metrics stored in SQL

Score	Criteria
2	Time-series data offloaded to InfluxDB, but graphs remain in SQL
3	Full polyglot persistence: PostgreSQL (ACID), Neo4j (Graph), InfluxDB (Telemetry)
4	Event-sourced aggregates (CQRS) utilized to pre-compute complex reads
5	Perfect modeling: formally verified data routing, zero performance-degrading access patterns, mathematical proof of optimal engine selection

Connection Pooling and Proxying (Weight: 20%)

Resource Management

How do microservices connect to the database without exhausting resources?

Score	Criteria
0	Microservices open direct TCP connections to the database; connection limits frequently hit
1	Application-level connection pooling (e.g., HikariCP), but 100 pods still open 1000 connections
2	Connection proxy (PgBouncer/ProxySQL) deployed, but configured with static, unoptimized limits
3	Transaction-level pooling via PgBouncer; database connection count strictly bounded (< 100) regardless of app scale
4	Dynamic pool sizing based on real-time database load and CPU metrics
5	Perfect pooling: formally verified connection bounds, zero connection storms possible, mathematical proof of optimal multiplexing

High Availability and Failover (Weight: 15%)

Resilience

Can the database survive the loss of its primary node without data loss or downtime?

Score	Criteria
0	Single-node database; manual recovery required on crash
1	Asynchronous replication to a standby node; manual promotion required
2	Automated failover via Patroni/Stolon, but accepts some data loss (async)
3	Synchronous replication ensuring zero data loss (RPO=0); automated, sub-minute failover (RTO < 60s)
4	Multi-region active-active or distributed SQL (CockroachDB) surviving a region loss
5	Perfect resilience: formally verified consensus algorithms, zero split-brain risk, mathematical proof of continuous availability

Scalability and Sharding (Weight: 15%)

Write Throughput

How does the database handle write loads that exceed a single machine's capacity?

Score	Criteria
0	Vertical scaling only (buying a bigger EC2 instance)
1	Read replicas used to offload reads, but writes bottlenecked on primary
2	Manual application-level sharding (hard to maintain)
3	Distributed SQL (CockroachDB) or transparent sharding (Citus/Vitess) handling writes horizontally
4	Dynamic resharding without downtime; hotspots automatically rebalanced
5	Perfect scaling: formally verified data distribution, zero write bottlenecks, linear horizontal scalability

Tenant Isolation (RLS) (Weight: 15%)

Security Boundaries

How is cross-tenant data access prevented?

Score	Criteria
0	Application code filters by `tenant_id` (guarantees eventual breach)
1	Database views used to filter data, but views can be bypassed
2	Database roles created per tenant, but application manages switching roles
3	Row-Level Security (RLS) enforced at the database engine level; application cannot bypass policies even with raw SQL
4	Context-aware RLS: database reads session variables (e.g., `app.current_tenant`) set by the connection proxy
5	Perfect isolation: formally verified RLS policies, zero ability to query cross-tenant data, cryptographic proof of tenant boundaries

Backup and Point-in-Time Recovery (Weight: 15%)

Data Durability

Can the database be restored to any specific second in the past?

Score	Criteria
0	Nightly snapshot backups only; up to 24 hours of data loss possible
1	WAL archiving enabled, but restoration is manual and untested
2	Point-in-Time Recovery (PITR) automated and tested quarterly
3	Continuous WAL archiving to immutable, write-once storage (S3 Object Lock)
4	Automated, daily restoration tests in an isolated environment verifying backup integrity
5	Perfect durability: formally verified recovery bounds, zero data loss (RPO=0), cryptographic attestation of backup integrity

The Mythos Database Suite (MDBS)

Traditional benchmarks measure TPS (Transactions Per Second). The MDBS evaluates connection storm survival, tenant isolation, and zero-data-loss failover.

Suite Composition

MDBS-Pool

1. Connection Storm Test: 100+ tests spawning 5,000 concurrent microservice connections, verifying the PgBouncer proxy bounds the database connections to < 100 without dropping transactions.
1. Failover Test: 50+ tests physically killing the primary database node during peak write load, verifying Patroni promotes a standby with zero data loss (synchronous) and sub-minute recovery.

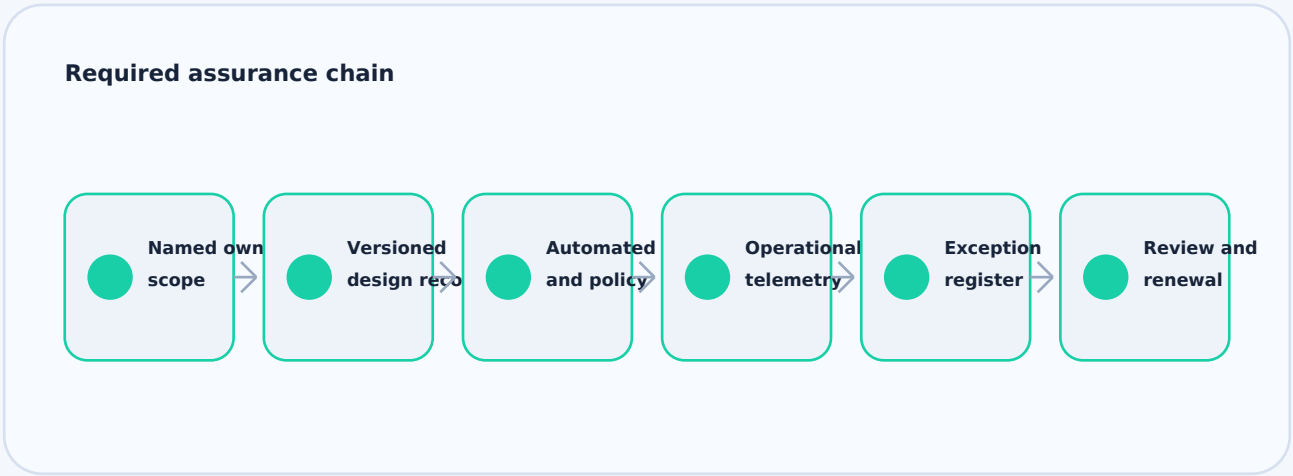
MDBS-Isolation

1. Tenant Bypass Test: 100+ tests attempting to execute raw SQL queries without tenant context or with a spoofed tenant ID, verifying the database RLS policies block the queries.
1. PITR Test: 50+ tests restoring the database to a random timestamp 3 hours in the past, verifying the WAL archive is continuous and the restoration is byte-perfect.

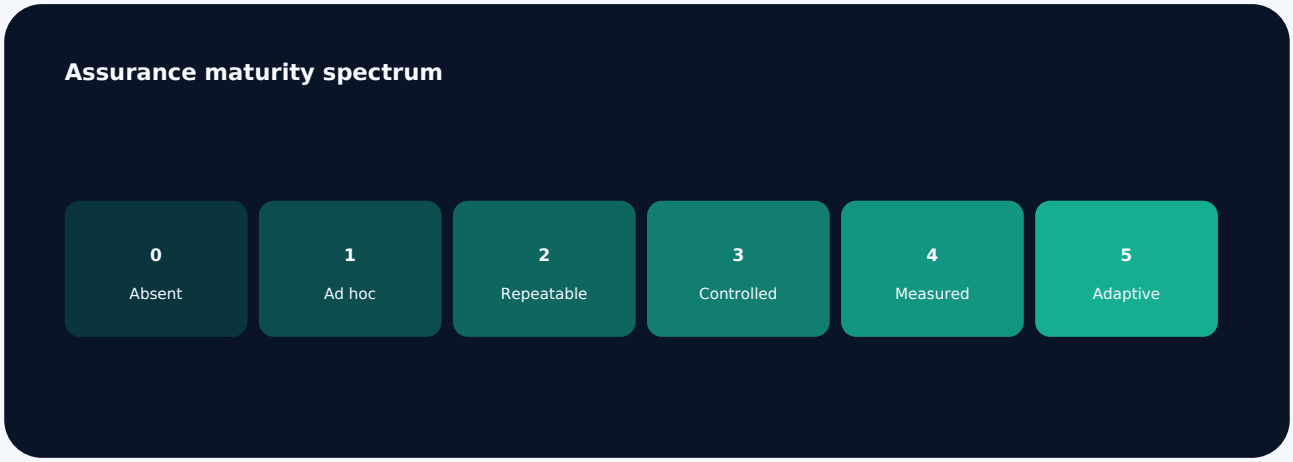
Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.



Current authority register

Authority	Version / snapshot	Applicability	Review by
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22
PostgreSQL Global Development Group: PostgreSQL Documentation	Rolling official documentation snapshot	High	2026-10-22

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.