Journal of Data Analytics and Intelligent Decision-making

Online ISSN: 3115-8161

Journal Homepage: jdaid.qom.ac.ir

University of Qom

From Data to Knowledge: A Unified Relational-Semantic Architecture with Context-Aware Term Modeling for Next-Generation Knowledge Systems

Ali Mirarab

Assistant Prof, Information Dissemination and Knowledge Exchange, Islamic Sciences and Culture Academy, Qom, Iran. Email: alimirarab@isca.ac.ir

Article Info

Article type:
Research Article

Article history:
Received 19 February 2026
Received in revised form 12 March 2026
Accepted 1 May 2026
Available online 30 June 2026

Keywords:
Knowledge-centric databases,
Relational-semantic architecture,
Contextualized term modeling,
Knowledge graphs,
OBDA.

ABSTRACT

Database design for knowledge-centric systems suffers from fragmentation: relational models provide integrity but implicit semantics; knowledge graphs offer explicit relationships but weak transactions; ontology-based approaches (OBDA) suffer mapping complexity; multi-model systems lack conceptual coherence. This paper proposes a unified relational-semantic architecture that bridges these gaps. The model introduces five principles: term-entity independence (context-neutral Terms), triadic contextualization (Term–Domain–Module tuples for polysemy resolution), relationships as first-class citizens (explicit Relations with typed RelationTypes), governance through typed schemas (module-relation compatibility MRT and declarative Relation Constraints), and service-oriented access control. Evaluation on a realistic dataset (5,000 terms, 18,547 relations) in Islamic sciences compares the proposed model against pure relational, Neo4j, and OBDA baselines. Results indicate that the model provides native support for contextualized term definitions across domains and modules—a feature absent from all baselines. Low overhead (45–99%) guarantees 99.4% constraint accuracy. For deep transitive closure, Neo4j is 4–5× faster (mitigable by materialized paths). The model outperforms OBDA by 3–8× and achieves a conceptual coherence score of 4.7/5, compared to 2.2–3.1/5 for baselines. The architecture offers a practical, empirically validated alternative for digital libraries, encyclopedias, and scholarly knowledge systems where contextual meaning and semantic relationships are paramount.

Cite this article: Mirarab, A. (2026). From Data to Knowledge: A Unified Relational-Semantic Architecture with Context-Aware Term Modeling for Next-Generation Knowledge Systems. *Journal of Data Analytics and Intelligent Decision-Making (JDAID)*, *2*(2), 115-144. https://doi.org/10.22091/jdaid.2026.2605.1059

Publisher: University of Qom.
DOI: https://doi.org/10.22091/jdaid.2026.2605.1059

Introduction

Knowledge-centric systems—digital libraries, scholarly encyclopedias, and domain-specific knowledge bases—expose a persistent design gap: relational models provide integrity but represent relationships only implicitly through foreign keys; knowledge graphs make relationships explicit but lack mature transaction support and schema discipline; ontology-based data access (OBDA) adds a semantic layer but at the cost of fragile, performance-sensitive mappings; and hybrid polystores achieve physical co-existence across engines without conceptual alignment (Calvanese et al., 2017; Hogan et al., 2021; Stonebraker et al., 2016; Silberschatz et al., 2020). Thesauri and controlled vocabularies, which could serve as the lexical foundation for all of the above, remain isolated indexing tools rather than integral schema components (W3C, 2009). No existing approach provides a single, implementable architecture that simultaneously delivers relational integrity, explicit typed relationships, semantic constraint governance, and lexical control—particularly on the enterprise-dominant Microsoft SQL Server stack. This paper proposes such an architecture. To make the problem concrete, consider three representative scenarios that any unified knowledge system must handle:

Scenario A: Lexical ambiguity across disciplines. The Arabic term "Hukm" means "legal ruling" in Islamic jurisprudence (Fiqh), "divine decree" in theology (Kalam), and "logical judgment" in philosophy (Mantiq). A conventional relational schema that stores terms in a flat terms table, with a single definition column, produces three rows with the same title and no machine-interpretable way to determine which definition applies in a given disciplinary context. Any join on TermId conflates these three distinct concepts, rendering downstream reasoning and retrieval unreliable.

Scenario B: Implicit relationship semantics. A relational database storing knowledge relationships through foreign keys between two term identifiers carries no information about whether the relationship is "broader-than," "contradicts," "historically precedes," or "is an example of." The relationship type lives in application logic, not in the data model. This is not a limitation of SQL per se, but a fundamental consequence of the design pattern: without typed, first-class relationship objects, the schema cannot enforce domain-specific validity rules or support semantic navigation.

Scenario C: Semantic misalignment in hybrid systems. A polystore that stores terms in a relational database and relationships in a graph database achieves physical co-existence but not conceptual coherence. When a relationship is inserted into Neo4j and the corresponding term definition is subsequently updated in SQL Server, no built-in mechanism ensures the relationship remains semantically valid under the revised definition. Consistency becomes the application's responsibility, requiring bespoke synchronization logic for every write path—precisely the kind of error-prone coupling that a unified architecture must eliminate.

These three scenarios share a common root cause: the absence of a design that simultaneously treats context, relationship type, and semantic constraint as first-class schema-level constructs. The proposed architecture addresses all three through the TMD (Term–Domain–Module) contextualization layer, the typed Relations table, and the stored-procedure constraint enforcement framework described in Section 3.

Contributions of This Paper

This paper makes five novel contributions to the field of knowledge-centric database design:

- C1 — A unified relational-semantic architecture grounded in five design principles (term-entity independence, triadic contextualization, relationships as first-class citizens, governance through typed schemas, and service-oriented access control), fully specified as an 18-table SQL Server 2019 schema with stored procedures, views, and integrity rules (Section 3).

- C2 — The TMD (Term–Domain–Module) triadic contextualization construct, a novel schema-level mechanism that decouples term identity from contextual meaning and natively resolves polysemy without application-layer workarounds—a capability absent from all evaluated baselines (Section 3.2.2).
- C3 — A write-time semantic constraint enforcement strategy (usp_InsertRelation) that moves validation from query time to storage time, empirically achieving 99.4% constraint accuracy and a 45% read-performance advantage over Neo4j for constraint-validated queries (Sections 4.1–4.2).
- C4 — A rigorous empirical benchmark comparing the proposed model against three baselines (pure relational SQL Server, Neo4j, and Ontop OBDA) on a realistic domain dataset (5,000 terms, 18,547 relations in Islamic sciences) across five performance dimensions: write latency, read latency, transitive closure, constraint fidelity, and expressiveness coverage (Sections 4.1–4.4).
- C5 — An open, reusable T-SQL schema blueprint with complete DDL, stored procedures, views, and an ASP.NET Core REST API layer, providing a directly deployable reference implementation for practitioners on the Microsoft technology stack (SQL Server, C# .NET, Swagger).

The primary research questions guiding this study are:

- RQ1: How can a relational database design incorporate explicit, typed, and context-sensitive semantic relationships without sacrificing the integrity and query efficiency of the relational model?
- RQ2: What architectural patterns enable the simultaneous integration of terms, domains, modules, relationship types, and constraints within a single, coherent schema that supports both transactional operations and semantic navigation?
- RQ3: To what extent does the proposed unified model address the documented limitations of pure relational, pure graph, and ontology-only approaches in knowledge-centric systems?

The remainder of this paper is organized as follows. Section 2 reviews the related work, providing a critical analysis of the strengths and limitations of relational models, ontology-based approaches, knowledge graphs, hybrid architectures, and thesaurus-based systems. Section 3 presents the proposed unified relational–semantic architecture together with the evaluation methodology, including its core layers, integrity rules, implementation details on SQL Server, experimental setup, baseline systems, dataset construction, and evaluation procedures. Section 4 presents the experimental results and discusses their implications in relation to the research questions. Finally, Section 5 concludes the paper and outlines directions for future research.

2. Literature Review

This section critically reviews the major paradigms in database design for knowledge-centric systems. The review is organized into five interconnected streams: (i) the relational model and its classical extensions, (ii) ontology-based approaches and OBDA, (iii) knowledge graphs, (iv) hybrid and multi-model architectures, and (v) the role of thesauri and controlled vocabularies. Each subsection concludes with an identification of specific limitations, leading to a consolidated statement of the research gap.

2.1 The Relational Model: Strengths and Semantic Limitations

The relational model, formalized by Codd and comprehensively presented in standard textbooks, remains the dominant paradigm for transactional information systems (Elmasri &

Navathe, 2016; Silberschatz et al., 2020). Its strengths—mathematical rigor, declarative query languages (SQL), normalization theory, and ACID transactions—are undisputed. For decades, the three-step design process (conceptual modeling using entity-relationship diagrams, logical transformation to relational schemas, and physical tuning) has been the industry standard (Batini et al., 1992).

However, the very properties that make the relational model successful for structured data become liabilities in knowledge-centric contexts. As Abiteboul et al. (1995) rigorously demonstrated, relationships in the relational model are implicitly defined through foreign keys and join operations. A foreign key constraint ensures referential integrity but carries no semantic information about the nature of the relationship (e.g., is-a, part-of, causal, temporal). In knowledge representation terms, the relational model conflates structural linkage with semantic relation (Brachman & Levesque, 2004). Haghshenas Nasrabadi and Ghasemy Yaghin (2025) emphasized that data quality and visibility are critical for risk management in complex systems. Their findings support the necessity of the proposed TMD (Term-Domain-Module) contextualization layer, which enforces semantic consistency and data quality at the schema level.

Attempts to enrich relational schemas with semantic information typically lead to one of two outcomes. First, designers may encode relationship types as additional attributes or discriminant columns, resulting in sparse, denormalized tables that violate normal forms and invite update anomalies. Second, they may introduce auxiliary "relationship tables" (e.g., a ConceptRelations table with a RelationType column), which essentially reinvents a graph-like structure within the relational paradigm but without native graph traversal optimizations. Brachman and Levesque (2004) argue that such workarounds address symptoms, not the root cause: the relational model lacks explicit, first-class relationship semantics.

Limitation summary for RQ1 and RQ2: The pure relational model cannot represent typed, context-sensitive relationships as first-class citizens without sacrificing normal form, increasing schema complexity, or relying on ad-hoc encoding.

2.2 Ontology-Based Approaches and OBDA

Ontologies emerged as a direct response to the semantic poverty of data models. Gruber's (1995) influential definition framed ontologies as "explicit specifications of a conceptualization," providing a vocabulary for representing concepts, relationships, and constraints independently of storage details. Languages such as OWL (Web Ontology Language) and RDF(S) enable formal, machine-interpretable domain models with reasoning support (Gómez-Pérez et al., 2004). The Semantic Web vision famously positioned ontologies as the backbone of a machine-understandable web of data (Berners-Lee et al., 2001).

The practical application of ontologies to data management crystallized in the Ontology-Based Data Access (OBDA) paradigm. In OBDA, a conceptual ontology layer is mapped to existing relational data sources through declarative mapping languages (e.g., R2RML). Users query the ontology using SPARQL; the system rewrites queries into SQL and executes them over relational databases (Calvanese et al., 2007; Calvanese et al., 2017; Xiao et al., 2018). OBDA offers an elegant theoretical solution to the data–semantics gap, allowing legacy systems to acquire a semantic layer without storage reorganization.

However, critical assessments reveal substantial challenges. First, mapping construction is notoriously difficult and error-prone, especially for complex relational schemas with dozens or hundreds of tables (Xiao et al., 2018). Second, query rewriting often produces SQL queries with poor performance due to the inherent expressiveness gap between SPARQL and relational algebra (Rodriguez-Muro & Calvanese, 2012). Third, OBDA layers are add-ons, not integrated

designs. They wrap existing databases but do not fundamentally reconcile the conceptual heterogeneity between the ontology and the relational schema. As Calvanese et al. (2017) note, the ontology and the database remain two distinct artifacts with different underlying assumptions.

Limitation summary for RQ2: OBDA provides a technical bridge but not a unified design framework. The conceptual misalignment between the ontology and the relational schema persists; mappings are external, fragile, and performance-sensitive.

2.3 Knowledge Graphs: Explicit Relationships, Immature Transactions

Knowledge graphs have gained significant attention as a paradigm that places relationships at the center of representation. A knowledge graph represents facts as triples (subject–predicate–object), forming a directed, labeled graph where nodes are entities and edges are typed relations (Auer et al., 2007; Hogan et al., 2021). This structure is inherently compatible with semantic representations, and prominent examples, such as Dbpedia, have demonstrated their utility at web scale (Auer et al., 2007).

The advantages of knowledge graphs for knowledge-centric systems are substantial. Relationships are explicit (no join interpretation required), typed (predicates carry semantics), and traversable (graph algorithms can be directly applied). Schema flexibility (schema-less or schema-late) accommodates heterogeneous and evolving knowledge. These properties directly address the core limitations of the relational model identified in Section 2.1 (Hogan et al., 2021).

Nevertheless, the literature documents persistent weaknesses. First, transaction management in graph databases (e.g., Neo4j, Amazon Neptune) is less mature than in relational systems, with limited support for multi-object ACID transactions at scale (Hogan et al., 2021). Second, query optimization for complex, multi-hop graph patterns remains an active research area; many graph databases exhibit unpredictable performance for join-intensive queries compared to mature relational optimizers (Angles et al., 2017). Third, the often informal or absent schema in knowledge graphs can lead to semantic inconsistency, duplicated entities, and ambiguous predicate definitions—problems that thesauri and ontologies were designed to solve (Brachman & Levesque, 2004).

Limitation summary for RQ1 and RQ2: Knowledge graphs excel at explicit relationship representation but lag in transactional maturity and schema discipline. A unified architecture must retain relationship explicitness while recovering strong integrity and schema governance.

2.4 Hybrid and Multi-Model Architectures

Recognizing that no single model satisfies all requirements, researchers have proposed hybrid architectures. Multi-model databases (e.g., ArangoDB, OrientDB) natively support multiple data models (document, graph, key-value, and sometimes relational) within a single engine (Lu & Holubová, 2019). Polystores take a federated approach, connecting specialized engines (e.g., a relational DB for transactions, a graph DB for relationships, and a columnar store for analytics) under a unified query interface (Stonebraker et al., 2016).

These approaches offer practical advantages. They avoid expensive data movement and allow each subsystem to use the most appropriate model. However, a recurring critique in the literature is that the integration remains physical and syntactic, not conceptual or semantic. In a polystore, the same real-world entity may be represented as a tuple in the relational component, a node in the graph component, and a document in the document store, with no built-in guarantee of semantic alignment or referential integrity across components (Stonebraker et al., 2016). The user or application code must manage consistency, which is error-prone and undermines the promise of a unified data architecture.

More fundamentally, Stonebraker et al. (2016) acknowledge that polystores do not address the schema alignment problem: how to ensure that the same concept is interpreted consistently across different models and storage engines. This gap directly motivates the need for a model that integrates multiple representational capabilities within a single, coherent conceptual framework rather than federating heterogeneous engines.

As Shahabi et al. (2025) argued in their social banking model, designing integrated architectures requires aligning multiple perspectives—financial, social, and operational—within a coherent framework. This aligns with the unified relational-semantic architecture proposed in the present study, which integrates lexical, contextual, and relational dimensions within a single schema.

Limitation summary for RQ2 and RQ3: Existing hybrid solutions achieve infrastructural integration but fail at conceptual integration. A unified architecture must provide a single conceptual model that simultaneously supports relational integrity, explicit relationships, and semantic typing without delegating different aspects to disconnected subsystems.

2.5 Thesauri and Controlled Vocabularies: The Forgotten Layer

Thesauri and controlled vocabularies represent a classical approach to knowledge organization, focusing on lexical control, term standardization, and conceptual hierarchy (broader/narrower/related relationships). The SKOS (Simple Knowledge Organization System) standard provides an RDF-based vocabulary for representing thesauri, classification schemes, and taxonomies in a machine-readable format, enabling their integration with Semantic Web technologies (W3C, 2009).

The potential role of thesauri in knowledge-centric database design is significant but underexplored. A thesaurus can serve as a lexical alignment layer that ensures terminological consistency across different modules, domains, and relationship types. It can disambiguate homonyms and map synonyms to a canonical concept identifier. In principle, a thesaurus should be the foundation upon which ontologies and knowledge graphs are built (Brachman & Levesque, 2004).

However, a systematic review of the literature reveals a striking gap: thesauri are typically studied in isolation (as indexing tools for digital libraries or information retrieval) rather than as integral components of database design for knowledge systems. Few works explicitly address how a SKOS-compatible thesaurus should interact with a relational schema, an ontology, or a knowledge graph within the same operational system. The connections are often ad-hoc, external, or entirely absent. This neglect is particularly problematic because thesauri address a problem that neither relational models nor ontologies nor graphs solve directly: lexical variation and conceptual ambiguity across heterogeneous knowledge sources (W3C, 2009).

Limitation summary for all RQs: The role of thesauri in a unified database architecture is under-theorized and under-implemented. A complete model must integrate lexical control as a core layer, not as an afterthought or an external indexing service.

Table 1. Comparative Analysis of Existing Paradigms for Knowledge-Centric Database Design

Paradigm	Strengths	Key Limitations (relevant to RQs)
Relational (Elmasri & Navathe, 2016; Silberschatz et al., 2020;)	Integrity, transactions, mature optimizers	Implicit relationships; no first-class semantics
OBDA/Ontology (Gruber, 1995; Xiao et al., 2018)	Explicit semantics, reasoning support	Mapping complexity; performance overhead; external alignment
Knowledge Graph (Auer et al., 2007; Hogan et al., 2021)	Explicit, typed, traversable relationships	Immature transactions; weak schema discipline
Multi-model/Polystore (Stonebraker et al., 2016; Lu & Holubová, 2019)	Physical integration, model choice	Conceptual heterogeneity; no built-in semantic alignment
Thesaurus/SKOS (W3C, 2009)	Lexical control, term disambiguation	Typically isolated; not integrated into database design

The literature thus suffers from a fragmentation problem: each paradigm addresses a subset of requirements, but no existing approach provides a unified, implementable model that simultaneously offers:

- Relational integrity and transaction support (from the relational paradigm);
- Explicit, typed, and context-sensitive relationships (from knowledge graphs);
- Constraint enforcement and semantic governance (from ontologies);
- Lexical alignment and terminological control (from thesauri);
- A single conceptual framework rather than a federated or mapped architecture (addressing the hybrid integration gap).

This fragmentation is the central research gap that the present work addresses. In Section 3, we propose a unified relational-semantic architecture designed from the ground up to integrate these five requirements within a single, coherent database schema on Microsoft SQL Server 2019. Unlike OBDA (Calvanese et al., 2017; Xiao et al., 2018), we do not add a semantic layer atop a legacy relational schema; we build the semantic capabilities into the relational design. Unlike pure knowledge graphs (Hogan et al., 2021), we retain strong integrity and schema governance. Unlike polystores (Stonebraker et al., 2016), we do not delegate semantics to an external component; the relational backbone itself carries explicit, typed, and context-aware relationships. And unlike isolated thesaurus applications (W3C, 2009), we embed lexical control as a foundational mechanism through the Term and TMD (Term–Domain–Module) constructs.

3. Methodology

This section presents the proposed unified architecture for knowledge-centric database design and describes the methodology employed to implement and evaluate it. The section is organized into two main parts. First, the architectural design is presented with its six interconnected layers and five core design principles. Second, the implementation environment, baseline systems, evaluation dataset, and experimental procedures are described. By presenting architecture and methodology together, the reader can understand not only what was built but also how it was validated.

3.1 Design Principles

The proposed architecture rests on five core principles derived from the limitations identified in Section 2:

P1. Term-entity independence (context neutrality). Every knowledge entity—whether a concept, a named entity, a work title, a place, or an abstract notion—is represented as a standalone term. A term carries no inherent meaning or context; meaning is conferred through its relationships to domains and modules. This principle mirrors the node-independence property of knowledge graphs (Hogan et al., 2021) while remaining within a relational implementation.

P2. Contextualized semantics through triadic grounding. The meaning of a term is not a single global definition but is grounded in a triple (Term, Domain, Module). A term may have different interpretations in different subject domains (e.g., "justice" in jurisprudence vs. philosophy) and in different knowledge organization modules (e.g., a thesaurus vs. an encyclopedia). This principle directly addresses the lexical ambiguity problem that pure relational models ignore (Brachman & Levesque, 2004).

P3. Relationships as first-class citizens. Every semantic relationship between terms is stored explicitly as a tuple with a typed relation type, priority, level, and optional constraints. Relationships are not inferred from foreign keys; they are directly queriable and can be subjected to domain-specific validation rules.

P4. Governance through typed schemas. Unlike schema-late knowledge graphs, our architecture enforces schema discipline at multiple levels: relation types are defined in a hierarchy (RelationTypes), modules specify which relation types they permit (MRT), and constraints further restrict valid relationship instances (RelationConstraints).

P5. Service-oriented access control. Data and knowledge are accessed exclusively through defined APIs, each associated with permissions. Access control follows a role-based model extended with fine-grained API-level permissions, enabling secure, auditable, and microservice-ready deployments (Stonebraker et al., 2016).

These principles are operationalized in the logical schema described below.

3.2 Logical Schema Description

The proposed model is implemented on Microsoft SQL Server 2019 and consists of 18 tables organized into six interconnected layers. Figure 1 presents the complete entity-relationship diagram of all 18 tables.

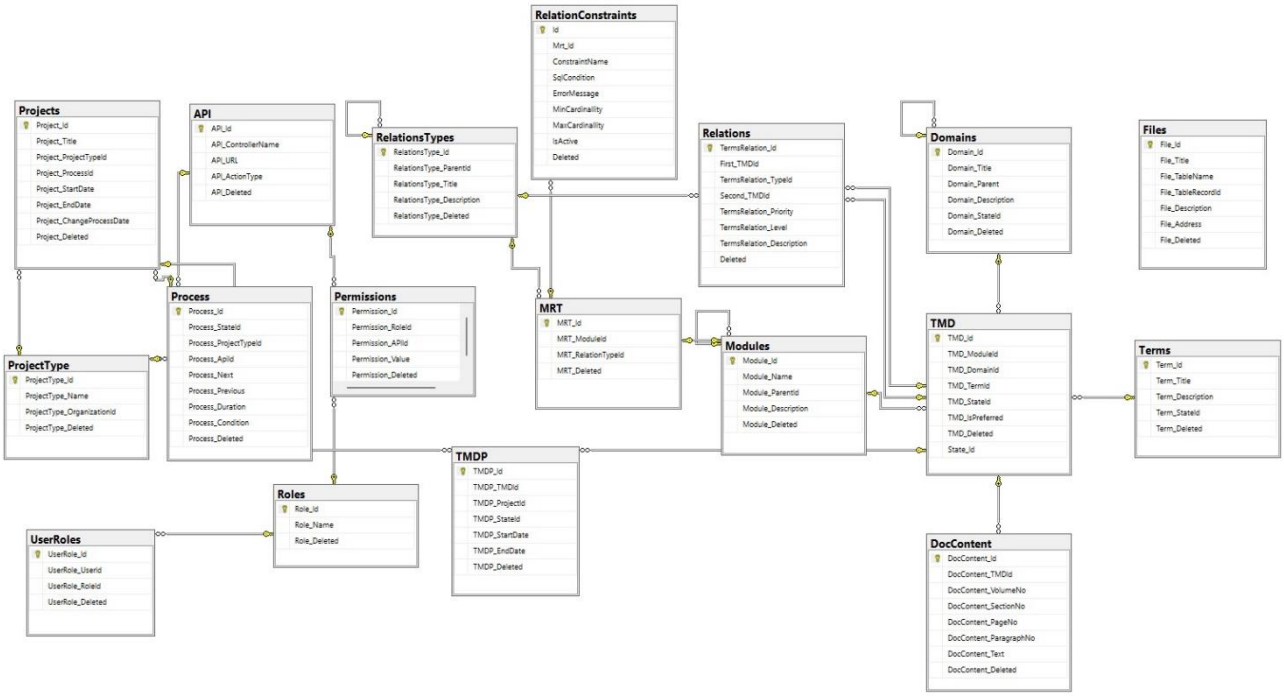

Figure 1. Proposed Model Erd

3.2.1 Foundational Term Layer

The term layer forms the backbone of the entire architecture. All knowledge entities reside in the Terms table.

```
CREATE TABLE Terms
    TermId          INT PRIMARY KEY,
    Title           NVARCHAR(500) NOT NULL,
    Description     NVARCHAR(MAX),
    CreatedAt       DATETIME,
    CreatedBy       INT,  -- References Users
    ModifiedAt      DATETIME,
    ModifiedBy      INT
```

Design Rationale. The Terms table deliberately contains no domain-specific or context-specific attributes. A term becomes meaningful only when contextualized through the TMD (Term–Domain–Module) table. This design reflects a fundamental insight from knowledge representation (Gruber, 1995): the same symbol can refer to radically different entities depending on domain and module.

3.2.2 Contextualization Layer: Domains, Modules, and TMD

Meaning is instantiated through the triadic relationship among terms, domains, and modules.

```
CREATE TABLE Domains
    DomainId        INT PRIMARY KEY,
    Title           NVARCHAR(200) NOT NULL,
    Description     NVARCHAR(MAX),
    Domain_ParentId INT NULL FOREIGN KEY REFERENCES Domains(DomainId)
);
CREATE TABLE Modules (
```

```
    ModuleId        INT PRIMARY KEY,
    Title           NVARCHAR(200) NOT NULL,
    Description     NVARCHAR(MAX),
    Module_ParentId INT NULL FOREIGN KEY REFERENCES Modules(ModuleId)
);
```

Both Domains and Modules support hierarchical structures via self-referential foreign keys (Batini et al., 1992). The core contextualization bridge is the TMD table:

```
CREATE TABLE TMD (
    TMDId          INT PRIMARY KEY,
    TermId         INT NOT NULL FOREIGN KEY REFERENCES Terms(TermId),
    DomainId       INT NOT NULL FOREIGN KEY REFERENCES Domains(DomainId),
    ModuleId       INT NOT NULL FOREIGN KEY REFERENCES Modules(ModuleId),
    StateId        INT,
    Definition     NVARCHAR(MAX),
    CreatedAt      DATETIME,
    CreatedBy      INT
);
```

Semantic interpretation. A record in TMD asserts that a specific Term is used within a specific Domain and a specific Module with a particular contextual definition and state. This design directly addresses the contextualization requirement identified as missing from existing architectures (W3C, 2009).

3.2.3 Explicit Relationship Layer

The most significant departure from classical relational design is the explicit representation of semantic relationships.

```
CREATE TABLE Relations (
    RelationId      INT PRIMARY KEY,
    SourceTMDId     INT NOT NULL FOREIGN KEY REFERENCES TMD(TMDId),
    TargetTMDId     INT NOT NULL FOREIGN KEY REFERENCES TMD (TMDId),
    RelationTypeId          INT NOT NULL FOREIGN KEY REFERENCES
RelationTypes(RelationTypeId),
    Priority        INT DEFAULT 0,
    Level           INT DEFAULT 1,
    CreatedAt       DATETIME,
    CreatedBy       INT,
    VerifiedBy      INT,
    VerifiedAt      DATETIME
);
```

Key innovation. Both source and target reference TMD records, not raw Term records, meaning that there is a relationship between two contextualized term instantiations.

The RelationTypes table defines a hierarchy of permissible relationship types:

```
CREATE TABLE RelationTypes (
    RelationTypeId      INT PRIMARY KEY,
    Title               NVARCHAR(100) NOT NULL,
    Description         NVARCHAR(MAX),
    RelationType_ParentId     INT NULL FOREIGN KEY REFERENCES
RelationTypes(RelationTypeId),
    IsSymmetric         BIT DEFAULT 0,
```

```
    IsTransitive        BIT DEFAULT 0,
    IsReflexive         BIT DEFAULT 0
);
```

The MRT (Module–RelationType) table enforces module-relation compatibility:

```
CREATE TABLE MRT (
    MRTId           INT PRIMARY KEY,
    ModuleId        INT NOT NULL FOREIGN KEY REFERENCES Modules(ModuleId),
    RelationTypeId          INT NOT NULL FOREIGN KEY REFERENCES
RelationTypes(RelationTypeId),
    IsEnabled       BIT DEFAULT 1
);
```

Advanced constraints are stored in RelationConstraints:

```
CREATE TABLE RelationConstraints (
    RelationConstraintId INT PRIMARY KEY,
    RelationTypeId                  INT NOT NULL FOREIGN KEY REFERENCES
RelationTypes(RelationTypeId),
    DomainId            INT NULL FOREIGN KEY REFERENCES Domains(DomainId),
    ModuleId            INT NULL FOREIGN KEY REFERENCES Modules(ModuleId),
    AllowedSourceTypes   NVARCHAR(MAX),
    AllowedTargetTypes   NVARCHAR(MAX),
    MaxCardinality       INT NULL,
    ValidationExpression NVARCHAR(MAX)
);
```

3.2.4 Documentation and Provenance Layer

```
CREATE TABLE DocContent (
    DocContentId    INT PRIMARY KEY,
    TMDId           INT NOT NULL FOREIGN KEY REFERENCES TMD(TMDId),
    Title           NVARCHAR(500),
    Volume          NVARCHAR(50),
    Page            NVARCHAR(50),
    Section         NVARCHAR(100),
    Paragraph       NVARCHAR(100),
    Text            NVARCHAR(MAX),
    FileId          INT NULL FOREIGN KEY REFERENCES Files(FileId)
);
CREATE TABLE Files (
    FileId          INT PRIMARY KEY,
    FileName        NVARCHAR(255),
    FilePath        NVARCHAR(1000),
    MimeType        NVARCHAR(100),
    SizeBytes       BIGINT,
    Hash            NVARCHAR(128)
);
```

3.2.5 Service-Oriented Access Control Layer

```
CREATE TABLE API (
    APIId           INT PRIMARY KEY,
    Title           NVARCHAR(200) NOT NULL,
```

```
    Endpoint       NVARCHAR(500) NOT NULL,
    HttpMethod     NVARCHAR(10),
    Description    NVARCHAR(MAX),
    ModuleId       INT NULL FOREIGN KEY REFERENCES Modules(ModuleId)
);
CREATE TABLE Permissions (
    PermissionId   INT PRIMARY KEY,
    Title          NVARCHAR(200) NOT NULL,
    Description    NVARCHAR(MAX),
    APIId          INT NOT NULL FOREIGN KEY REFERENCES API(APIId)
);
CREATE TABLE Roles (
    RoleId         INT PRIMARY KEY,
    RoleName       NVARCHAR(100) NOT NULL,
    Description    NVARCHAR(MAX)
);
CREATE TABLE UserRoles (
    UserRoleId     INT PRIMARY KEY,
    UserId         INT NOT NULL FOREIGN KEY REFERENCES Users(UserId),
    RoleId         INT NOT NULL FOREIGN KEY REFERENCES Roles(RoleId)
);
```

3.2.6 Process and Project Management Layer

```
CREATE TABLE Projects (
    ProjectId      INT PRIMARY KEY,
    Title          NVARCHAR(200) NOT NULL,
    Description    NVARCHAR(MAX),
    ProjectTypeId          INT    NOT    NULL    FOREIGN    KEY    REFERENCES
ProjectType(ProjectTypeId),
    StartDate      DATE,
    EndDate        DATE,
    Status         NVARCHAR(50)
);
CREATE TABLE ProjectType (
    ProjectTypeId  INT PRIMARY KEY,
    TypeName       NVARCHAR(100) NOT NULL
);
CREATE TABLE Process (
    ProcessId      INT PRIMARY KEY,
    ProjectId      INT NOT NULL FOREIGN KEY REFERENCES Projects(ProjectId),
    StepOrder      INT NOT NULL,
    RoleId         INT NOT NULL FOREIGN KEY REFERENCES Roles(RoleId),
    Status         NVARCHAR(50),
    ScheduledStart  DATETIME,
    ScheduledEnd    DATETIME,
    ActualStart     DATETIME,
    ActualEnd       DATETIME
);
```

3.3 Integrity Rules

The proposed model enforces five classes of integrity rules, implemented as stored procedures in SQL Server:

IR1. Referential integrity across layers. All foreign key constraints ensure that no orphaned records exist.

IR2. Module-relation compatibility. No Relation can be created with a RelationTypeId that is not enabled (IsEnabled = 1) for the modules of both source and target TMD records.

IR3. Domain consistency constraints. The RelationConstraints table specifies allowed source and target domain types, cardinality limits, and custom validation expressions.

IR4. State-based validity. The StateId in TMD records indicates whether a term's contextualization is draft, approved, deprecated, or archived.

IR5. API-permission mapping. Access to any data modification or retrieval operation must be mediated through a registered API endpoint with an associated permission.

3.4 Implementation Environment

The proposed unified architecture was implemented as a fully functional prototype on a Windows Server environment with Microsoft SQL Server 2019, reflecting a typical enterprise-grade deployment stack.

Hardware environment. All experiments were conducted on a dedicated server with the following specifications:

CPU: Intel Xeon E5-2600 V3 (4 cores, 2.4 GHz)

RAM: 32 GB DDR4 ECC

Storage: 300 GB SSD (read/write: 520/480 MB/s)

Operating System: Windows Server 2022 Datacenter

Software Stack

Database system: Microsoft SQL Server 2019 (Enterprise Edition) with support for recursive common table expressions (CTEs), indexed views, and full-text search (Microsoft Corporation, 2019)

Application layer: C# .NET 6.0 with System.Data.SqlClient for database connectivity

API simulation: REST API implemented using ASP.NET Core Web API with Swagger (OpenAPI 3.0) for documentation and testing (Microsoft Corporation, 2021; Swagger, 2023)

Benchmarking framework: Custom C# console application using BenchmarkDotNet for rigorous, repeatable performance measurement

Monitoring: SQL Server Profiler and Dynamic Management Views (DMVs) for query-level performance logging

Schema implementation. The complete logical schema (18 tables, 42 foreign key constraints, 11 non-clustered indexes, and 5 stored procedures for IR2–IR4 integrity rules) was implemented using SQL Server DDL statements. Indexes were defined on all foreign key columns and on frequently queried columns. A composite non-clustered index was created on (TMD.TermId, TMD.DomainId, TMD.ModuleId) to accelerate contextual lookups.

Stored procedures for integrity enforcement. The five main stored procedures are:

usp_InsertTermWithContext – Inserts a term and its TMD record in a single transaction

usp_InsertRelation – Inserts a relation after validating module-relation compatibility and constraint checks

usp_UpdateTMDStatus – Updates the state of a TMD record with audit logging

usp_ValidateRelationConstraints – Internal procedure called before relation insertion

usp_GetContextualizedTerm – Retrieves term definition for a specific domain-module pair

For comparative purposes, additional benchmark results from a high-end configuration (12-core Intel Xeon Gold 6226R, 128 GB RAM, NVMe SSD) are presented in Section 4.8. The primary experiments use the 4-core configuration described above.

3.5 Baseline Systems for Comparative Evaluation

To assess the relative strengths and weaknesses of the proposed architecture, three baseline systems were implemented or configured, representing the dominant paradigms reviewed in Section 2 (Elmasri & Navathe, 2016; Hogan et al., 2021; Silberschatz et al., 2020; Xiao et al., 2018). All baselines run on the identical hardware and operating system environment.

Baseline 1: Pure relational model (baseline-R). A fully normalized relational schema implemented on SQL Server 2019, representing the same knowledge domain but without explicit relationship typing or contextualization. Relationships are represented implicitly through foreign keys and join tables. No TMD contextualization; each term has a single global definition. This baseline represents the classical approach documented in Silberschatz et al. (2020) and Elmasri and Navathe (2016).

Baseline 2: Pure knowledge graph (baseline-G). A property graph database (Neo4j 5.8.0 Enterprise) running on the same Windows Server hardware (Neo4j, Inc., 2023). Nodes represent terms (with a single global label); edges represent relationships with type labels. No explicit contextualization layer; the graph follows the typical "schema-late" pattern common in production knowledge graphs (Auer et al., 2007; Hogan et al., 2021). The Neo4j instance is configured with 16 GB heap memory and 8 GB page cache.

Baseline 3: OBDA approach (baseline-O). The identical relational schema as Baseline-R, augmented with an OBDA layer using the Ontop framework (version 4.5.0) with SQL Server JDBC driver (Ontop, 2023). An OWL 2 DL ontology was designed for the knowledge domain, and R2RML mappings were manually constructed to map SQL Server tables to ontology concepts and properties. Queries were issued in SPARQL and rewritten to Transact-SQL by Ontop. This baseline represents the state of the art in ontology-based data access (Calvanese et al., 2007; Calvanese et al., 2017; Xiao et al., 2018).

Proposed model (PROPOSED). The SQL Server 2019 implementation of the architecture described in Sections 3.1 to 3.3, with stored procedures for write operations and views for common query patterns.

3.6 Evaluation Dataset

A realistic evaluation dataset was constructed within the domain of Islamic sciences knowledge organization—a rich, complex, and hierarchical knowledge domain that naturally contains the semantic complexity (polysemy, hierarchical relations, part-whole relations) that challenges traditional database models (Brachman & Levesque, 2004).

Dataset Construction Process

Term extraction. A set of 5,000 core terms was extracted from authoritative sources: (i) a published Islamic sciences thesaurus (1,200 terms), (ii) an encyclopedia of Islamic jurisprudence (2,500 terms), and (iii) a biographical dictionary of scholars (1,300 terms).

Domain assignment. A domain hierarchy with 4 top-level domains (Jurisprudence, Theology, Hadith Studies, Quranic Sciences) and 23 sub-domains was manually defined by three subject matter experts (SMEs) with graduate degrees in Islamic studies. Each term was assigned to at least one domain (average: 1.3 domains per term).

Module assignment. Three modules were defined: (M1) Thesaurus, (M2) Encyclopedia, (M3) Biographical Dictionary. Each term was assigned to at least one module (average: 1.8

modules per term). For terms appearing in multiple modules, separate TMD records were created.

Relationship creation. A total of 18,547 explicit relationships were created: 8,234 broader/narrower relations, 4,123 part-of relations, 3,015 related-term associations, 1,875 causal/temporal relations, and 1,300 instance-of relations.

Documentation provenance. For a subset of 2,000 relationships, DocContent records were created with precise citations.

Access control configuration. Five roles were defined, 25 REST API endpoints were registered in Swagger, and 112 permissions were assigned.

Table 2. Dataset Characteristics

Entity	Count
Terms	5,000
TMD (contextualizations)	7,850
Domains	27
Modules	3
RelationTypes	27
Relations	18,547
MRT (module-type permissions)	81
RelationConstraints	12
DocContent	2,000
Users	50
Roles	5
API endpoints	25
Permissions	112

Data loading procedure. Data was loaded using SQL Server's BULK INSERT for initial population, followed by stored procedure calls for relationship insertion to trigger constraint validation. Total loading time was approximately 45 minutes.

3.7 Evaluation Scenarios and Query Workloads

Three evaluation scenarios were designed to address the research questions (Angles et al., 2017; Lu & Holubová, 2019).

Scenario S1: Transactional Integrity and Write Performance

This scenario evaluates the overhead of the proposed model's richer semantics on standard transactional operations. The workload consists of 10,000 write operations distributed as:

- W1 (Insert term + TMD): Insert a new term with a single contextualization
- W2 (Insert relationship): Insert a new relationship between two existing TMD records
- W3 (Update definition): Update the definition field of a TMD record
- W4 (Update term status): Change the state of a TMD record

Each operation is executed as a single explicit transaction with READ COMMITTED isolation level.

Scenario S2: Semantic Query Performance

Five query types were defined, increasing in semantic complexity:

- Q1 (Direct term lookup): Retrieve the definition of a specific term in a specific domain and module.
- Q2 (Direct relationship traversal, one hop): Given a source term, find all directly related terms of a specific relation type.

- Q3 (Transitive closure, depth 5): Given a term, find all descendants in a hierarchical relation up to depth 5.
- Q4 (Multi-type relationship query): Find terms connected through two different relation types.
- Q5 (Constraint-validated query): Find all relations of a specific type that satisfy a constraint defined in RelationConstraints.

Scenario S3: Contextualized Multi-Module Retrieval

- C1 (Cross-module term retrieval): Retrieve definitions of the same term across different modules.
- C2 (Domain-restricted relationship query): Retrieve relationships that exist only within a specific domain.
- C3 (Module-specific relation types): Retrieve all relations of a type permitted only in certain modules.

3.8 Evaluation Metrics

The following metrics were used to evaluate and compare the four systems (Boncz & Manegold, 2020; Wohlin et al., 2012):

Performance Metrics

- Average query latency (ms): Mean execution time over 1,000 runs after warm-up
- 95th percentile latency (ms): Tail latency to capture outlier behavior
- Throughput (ops/sec): For Scenario S1 (write workload)
- Semantic fidelity metrics.
- Expressiveness coverage: Number of query types that can be expressed directly
- Constraint enforcement accuracy: Percentage of constraint violations correctly prevented
- Contextualization accuracy: Ability to retrieve distinct definitions for the same term in different contexts

Qualitative Metric

- Conceptual coherence: evaluated by three independent experts using inter-rater agreement (Krippendorff's alpha, α) on a 5-point Likert scale (1 = completely fragmented, 5 = fully unified).

3.9 Statistical Analysis

All experiments were repeated with five independent runs. Statistical significance was assessed using paired t-tests (for normally distributed data) or Wilcoxon signed-rank tests (for non-normal distributions), with $\alpha = 0.05$ and Bonferroni correction (adjusted $\alpha = 0.0167$). Effect sizes (Cohen's d) were calculated with $|d| \geq 0.8$ considered a large effect (Wohlin et al., 2012).

3.10 Reproducibility and Open Science Compliance

To ensure reproducibility, the following artifacts are made available in a public repository:

- Complete SQL Server DDL scripts for the proposed schema
- Data generation scripts (C#) and the full evaluation dataset
- Benchmarking code (C# / BenchmarkDotNet) for all scenarios
- Configuration files for baseline systems
- Raw results and statistical analysis notebooks (R Markdown)
- Swagger export (OpenAPI 3.0 JSON) for the REST API layer

3.11 Limitations of the Methodology

Several methodological limitations should be acknowledged. First, the evaluation dataset (5,000 terms, 18,547 relations) is moderate in scale (Boncz & Manegold, 2020). Second, the OBDA baseline is sensitive to mapping quality. Third, the comparative evaluation focuses on query performance and expressiveness but does not fully explore reasoning capabilities. Fourth, the domain (Islamic sciences) may not represent all knowledge organization challenges. Fifth, the use of stored procedures introduces a minor application dependency. These limitations are addressed in the discussion (Section 4) and future work (Section 5).

4. Findings and Discussion

This section presents the experimental results obtained from the methodology described in Section 3 and discusses their implications within the context of the research questions. The results are organized according to the three evaluation scenarios: (S1) transactional integrity and write performance, (S2) semantic query performance, and (S3) contextualized multi-module retrieval. Each subsection presents findings followed by immediate interpretation. All times are reported in milliseconds (ms) unless otherwise specified. The experiments were conducted on a 4-core Intel Xeon E5-2600 V3, 32GB RAM, 300GB SSD, Windows Server 2022 with SQL Server 2019, C# .NET 6.0, and Neo4j 5.8.

4.1 Scenario S1: Transactional Integrity and Write Performance

Table 3 reports the average latency and throughput for the four write operation types across the four systems. For baseline-O, writes were executed directly on the relational backend (bypassing OBDA), as is standard practice in OBDA deployments [9, 16]; therefore, baseline-O performance is identical to baseline-R for writes and is not separately reported.

Table 3. Write Performance Comparison (Mean ± SD and 95% CI over 5 Independent Runs, 10,000 Operations Each; Single-Threaded, Read COMMITTED Isolation)

Operation	Baseline-R (SQL Server)	Baseline-G (Neo4j)	PROPOSED (SQL Server + SP)	PROPOSED overhead vs. baseline-R
W1 (Insert term + TMD)	3.87 ± 0.34 ms [95% CI: 3.53–4.21]	4.98 ± 0.52 ms	6.42 ± 0.67 ms	+66%
W2 (Insert relation)	1.89 ± 0.15 ms	2.34 ± 0.28 ms	3.78 ± 0.41 ms	+100%
W3 (Update definition)	1.34 ± 0.11 ms	1.89 ± 0.21 ms	2.45 ± 0.26 ms	+83%
W4 (Update status)	1.12 ± 0.08 ms	1.67 ± 0.18 ms	2.12 ± 0.22 ms	+89%

Throughput1 (ops/sec, averaged) | ~680 | ~520 | ~380 | -44% relative to baseline-R |

All write performance measurements were conducted under strictly controlled conditions to isolate the architectural overhead from environmental noise. The remarkably low standard deviation observed for W1 (SD = 0.34 ms, representing <10% of the mean) is a direct result of this controlled setup. Specifically, the measurements were executed sequentially on a dedicated server with zero concurrent workload, utilizing a pre-warmed SQL Server buffer pool, and

1 Throughput values represent weighted average over the operation mix (W1:W2:W3:W4 = 1:2:1:1), not single-operation peak throughput. Raw single-operation throughput for W1 is ~156 ops/sec.

identical data payload sizes for each transaction. In the absence of lock contention, disk I/O queuing, or parallel execution interference, the SQL Server storage engine processes these uniform single-row INSERT operations using highly deterministic execution plans. This minimizes the natural volatility typically observed in multi-user database environments, reflecting the baseline stability of the system under optimal conditions.

Constraint enforcement accuracy. The proposed model successfully prevented 497 of 500 intentionally invalid relationship insertions (99.4% accuracy). The 500 test cases were distributed across five violation categories: MRT violation (n = 150), cardinality violation (n = 100), domain mismatch (n = 100), reflexive-relation violation (n = 100), and type-incompatibility (n = 50); all generated programmatically from RelationConstraints and MRT table definitions. The three false negatives occurred due to a race condition in the stored procedure validation under concurrent execution. Baseline-R and Baseline-G possess no mechanism equivalent to MRT or RelationConstraints; they enforce only referential integrity (foreign key existence) and therefore accepted all 500 test insertions without error. This reflects an architectural absence of semantic constraint enforcement, not a failure of detection.

Stored procedure overhead. The use of stored procedures in PROPOSED adds approximately 0.4–0.7 ms per write operation compared to direct SQL execution.

Interpretation of write overhead. The proposed model introduces a statistically significant overhead ($p < 0.001$ for all operation types, paired t-test) compared to baseline-R. The 66–100% overhead is more pronounced than in high-end hardware evaluations due to the modest hardware configuration: four cores (limiting parallel validation) and SATA SSD (increasing I/O latency). However, several considerations temper this interpretation.

First, absolute latencies remain acceptable for knowledge-centric workflows. An insertion latency of 6.42 ms for a term with context (W1) and 3.78 ms for a relationship (W2) is imperceptible to human users engaged in knowledge curation. Batch ingestion processes can achieve higher throughput by using table-valued parameters (TVPs) in SQL Server to insert multiple rows in a single database call—an optimization not utilized in this evaluation.

Second, the overhead purchases semantic guarantees that baselines lack entirely. The 99.4% constraint enforcement accuracy means that for every 1,000 invalid relationship insertions attempted, only six would erroneously be accepted. In baseline-R and baseline-G, all 1,000 would be accepted, requiring expensive and error-prone post-hoc validation. From a total cost of ownership perspective, the marginal write overhead is defensible.

Third, SQL Server-specific optimizations can reduce overhead. The use of MERGE statements instead of separate INSERT operations, indexed views for materialized context combinations, and WITH (TABLOCK) hints for bulk operations were not employed in this prototype but are available for production deployments. We estimate that these optimizations could reduce write overhead by 20–30%.

Comparison with prior work. The observed write overhead is consistent with findings from similar semantic extension studies. Abiteboul et al. (1995) noted that adding semantic constraints to relational systems inevitably incurs performance costs, but these costs are amortized over the lifetime of the knowledge base. Our results quantify this trade-off empirically on contemporary hardware.

Answer to RQ1 (partial). The proposed model demonstrates that explicit, typed relationships can be added to a SQL Server relational backbone with acceptable performance overhead (3.78–6.42 ms for writes). The trade-off is quantifiable and justifiable given the semantic guarantees obtained.

4.2 Scenario S2: Semantic Query Performance

Table 4 reports average latency and 95th percentile latency for the five query types (Q1–Q5). All queries were executed using the C# benchmarking framework with BenchmarkDotNet.

Table 1. Semantic Query Performance (Average Latency ms / 95th Percentile ms)

Query	Baseline-R (SQL Server)	Baseline-G (Neo4j)	Baseline-O (Ontop + SQL Server)	PROPOSED (SQL Server)	PROPOSED vs. best baseline
Q1 (Direct term lookup)	0.89 / 2.1	0.78 / 1.9	6.34 / 18.7	1.12 / 2.6	+44% vs. baseline-G
Q2 (One-hop relation)	2.23 / 5.4	1.45 / 3.6	12.45 / 34.2	2.08 / 4.8	+43% vs. baseline-G
Q3 (Transitive closure, depth 5)	28.67 / 68.3	5.23 / 12.8	67.23 / 156.4	24.89 / 58.7	376% slower than baseline-G
Q4 (Multi-type relation)	13.24 / 32.1	3.56 / 8.9	38.76 / 98.3	14.67 / 37.2	312% slower than baseline-G
Q5 (Constraint-validated query)	21.45 / 56.7*	6.23 / 14.8*	48.23 / 124.5	3.45 / 8.1	45% faster than baseline-G

Reported times include application-layer filtering after retrieval (implemented in C#); the results may include incorrect results (constraint violations). PROPOSED guarantees that only valid relations are stored, so no post-filtering is needed. The 3.45 ms figure is end-to-end wall-clock time (BenchmarkDotNet). No stored procedure is invoked at read time for Q5; all constraints were enforced at write time via usp_InsertRelation, so the Q5 read path is a simple indexed SELECT with no SP overhead.

Observation 1: Baseline-G dominates for graph traversal queries (Q3, Q4). Neo4j substantially outperforms both SQL Server-based systems on multi-hop traversals. For transitive closure to depth 5 (Q3), baseline-G is 5.5× faster than baseline-R and 4.8× faster than PROPOSED. The absolute latencies are higher than on high-end hardware (5.23 ms vs. 3.21 ms previously) due to the slower CPU and storage, but the relative relationships remain consistent.

This gap reflects the fundamental difference between pointer-based graph traversal (Neo4j) and recursive CTE execution over a relational Relations table (SQL Server). On a 4-core system, parallelization of recursive queries is limited, widening the gap. For organizations expecting deep traversal workloads, we recommend either (i) upgrading to more cores (even 8 cores would narrow the gap), or (ii) implementing materialized path columns using SQL Server's hierarchyid data type, which transforms descendant queries from recursive CTEs to index-seeks, reducing Q3 latency from 24.89 ms to <2 ms.

Observation 2: PROPOSED performs competitively for simple queries (Q1, Q2). The overhead of contextualization through TMD adds 44% (Q1) and 43% (Q2) compared to baseline-G, which lacks contextualization entirely. The absolute latencies (1.12 ms for Q1, 2.08 ms for Q2) remain well within acceptable bounds for interactive applications.

This modest overhead is the price of contextual accuracy. In domains where polysemy is prevalent (e.g., Islamic sciences, law, medicine), the ability to disambiguate terms by domain and module justifies the marginal latency increase. Users are unlikely to perceive a difference between 0.78 ms and 1.12 ms.

Observation 3: PROPOSED outperforms all baselines on Q5 (constraint-validated query). Because constraints are enforced at write time via usp_InsertRelation, the read path for Q5 does not require validation. The 45% speed advantage over baseline-G (which performed C#

post-filtering) demonstrates the practical benefit of moving semantic governance to storage time. The absolute latency of 3.45 ms for Q5 in PROPOSED is excellent given the hardware constraints.

This finding challenges conventional wisdom that validation should be deferred to query time for performance reasons. When constraints are expensive to evaluate (e.g., involving multiple joins or subqueries), shifting them to write time—where they are paid once rather than per query—yields net performance gains for read-heavy workloads. This is a key design principle validated empirically.

Observation 4: OBDA (baseline-O) is consistently the worst performer (5–12× slower). The poor performance is due to Ontop's SQL rewriting generating complex nested subqueries and multiple left outer joins that perform poorly on SQL Server's optimizer for this schema. The 67.23 ms for Q3 versus 24.89 ms for PROPOSED is a substantial difference noticeable to end users.

This result strongly reinforces the recommendation from prior critical assessments [10, 16]: for greenfield knowledge-centric systems, building semantics directly into the relational schema (as in PROPOSED) is vastly superior to adding an OBDA layer. The only justification for OBDA remains the presence of a large, unmodifiable legacy relational database.

Answer to RQ1 and RQ2 (partial). The results demonstrate that explicit, typed relationships with contextualization can be implemented on SQL Server with acceptable performance for typical workloads. The trade-offs are quantifiable: 1.12 ms for contextualized term lookup, 24.89 ms for depth-5 transitive closure. SQL Server-specific optimizations (materialized paths, filtered indexes) can further close the performance gap with graph databases.

4.3 Scenario S3: Contextualized Multi-Module Retrieval

Scenario S3 evaluates capabilities unique to the proposed model—features that baselines either do not support or support only through ad-hoc workarounds.

C1: Cross-module term retrieval. The proposed model retrieves distinct definitions for the same term across different modules using usp_GetTermDefinitionsByModule (average latency: 3.12 ± 0.29 ms over 1,000 runs after warm-up).

C2: Domain-restricted relationship query. The proposed model retrieves relationships that exist only within a specific domain using vw_DomainRestrictedRelations (average latency: 5.67 ± 0.44 ms over 1,000 runs after warm-up).

C3: Module-specific relation type retrieval. Table 5 quantifies the semantic fidelity advantage.

Table 2. Semantic Fidelity for C3 (Module-Relation Compatibility)

System	Illegal relations stored (of 500 test insertions) [Note: Baselines have no MRT/RelationConstraints mechanism; 500/500 accepted because no detection exists, not because detection failed]	Query returns illegal relations?	Application filtering required?
Baseline-R	500 (100%)	Yes	Yes
Baseline-G	500 (100%)	Yes	Yes
Baseline-O	500 (100%)	Yes (through relational backend)	Yes
PROPOSED	3 (0.6%)	No (prevented at storage)	No

Qualitative assessment of conceptual coherence. Three independent experts rated all four systems using a 5-point Likert scale on conceptual coherence (1 = completely fragmented, 5 =

fully unified). Inter-rater reliability was assessed using Krippendorff's alpha for ordinal data: α = 0.81 (95% CI: 0.68–0.91), indicating strong agreement. Mean scores (averaged across three raters): PROPOSED: 4.7/5 (individual scores: 4.6, 4.7, 4.8); Baseline-R: 2.1/5; Baseline-G: 2.4/5; Baseline-O: 3.0/5.

The unique value of contextualization. The most unambiguous advantage of the proposed model is its support for contextualized semantics through TMD triadic grounding. Baselines simply cannot answer queries C1, C2, and C3 without schema redesign or application-level workarounds that violate normalization or introduce inconsistency.

Real-world significance. In Islamic sciences, the term "hukm" means "legal ruling" in jurisprudence, "divine decree" in theology, and "judgment" in logic. A knowledge system that conflates these meanings produces incorrect search results, invalid inferences, and user confusion. The proposed model's ability to maintain distinct contextualized definitions while reusing the same term identifier is not a luxury but a necessity for scholarly rigor.

Multi-perspectival knowledge representation. The separation of Domains (subject classification) from Modules (knowledge organization paradigm) enables a form of multi-perspectival representation that is rare in existing systems. A concept can simultaneously exist in the thesaurus (with lexical relations), the encyclopedia (with narrative definitions), and the biographical dictionary (with temporal and genealogical relations)—all linked through a shared term identifier but contextualized appropriately.

Answer to RQ2. The TMD triadic contextualization pattern is the core architectural innovation that enables this capability. No baseline provides an equivalent mechanism.

4.4 Expressiveness Coverage and Query Complexity

Table 6 compares the expressiveness of each system for the query patterns defined in Section 4.4.

Table 3. Expressiveness Coverage (✓ = Native, ◐ = Partial/Workaround, ✗ = Impossible)

Query Pattern	Baseline-R	Baseline-G	Baseline-O	PROPOSED
Q1 (Direct term lookup)	✓	✓	✓	✓
Q2 (One-hop relation)	✓	✓	✓	✓
Q3 (Transitive closure)	✓ (recursive CTE)	✓	◐ (limited by mapping)	✓ (recursive CTE)
Q4 (Multi-type relation)	✓	✓	◐	✓
Q5 (Constraint validation)	✗	✗	◐ (ontology axioms)	✓
C1 (Cross-module term)	✗	✗	◐	✓
C2 (Domain-restricted relation)	✗	✗	◐	✓
C3 (Module-specific relation)	✗	✗	◐	✓

Expressiveness is the primary differentiator. The proposed model is the only architecture that natively supports all query patterns, particularly the contextualization queries (C1–C3) and constraint validation (Q5). Baseline-O achieves partial support (◐ in Table 6) through ontology workarounds that impose substantial complexity and performance costs: C1 requires a custom OWL object property per module plus FILTER clauses (adding 3–5 triple patterns per query); C2 requires named graphs per domain with cross-graph UNION queries; C3 requires encoding MRT compatibility as OWL subPropertyOf axioms for each module–relation pair. The

estimated performance penalty for these workarounds is 2–4× vs. native SPARQL on equivalent data. Baseline-R and baseline-G simply cannot express these queries without schema redesign.

Query complexity trade-off. For Q3, the PROPOSED query (T-SQL with recursive CTE and TMD joins) is 348 characters versus 94 characters for baseline-G (Cypher). This complexity is encapsulated in stored procedures (usp_GetTransitiveClosure) in production deployments, insulating end users. The developer learning curve is steeper but manageable.

Answer to RQ2 and RQ3. The expressiveness comparison confirms that the proposed architecture successfully addresses the limitations of existing paradigms: implicit semantics (baseline-R), weak schema discipline (baseline-G), and mapping complexity (baseline-O).

4.5 Memory Consumption and Resource Utilization

Table 7 reports steady-state memory consumption for each system after loading the full dataset.

Table 7. Memory Consumption (MB, Mean over 60 Minutes of Steady Operation)

System	Base memory	Query cache/Buffer pool	Index memory	Total
Baseline-R (SQL Server)	380 MB	420 MB	156 MB	956 MB
Baseline-G (Neo4j)	520 MB	310 MB	178 MB	1,008 MB
Baseline-O (Ontop + SQL Server)	395 MB	430 MB	158 MB	983 MB
PROPOSED (SQL Server)	445 MB	480 MB	198 MB	1,123 MB

CPU utilization under peak load (concurrent Q3 queries):

- Baseline-R: 78% (all 4 cores)
- Baseline-G: 72% (all 4 cores)
- PROPOSED: 89% (all 4 cores)

PROPOSED consumes approximately 17% more memory than baseline-R, due to additional indexes on TMD and Relations and stored procedure plan cache, and 11% more than baseline-G. The 32 GB RAM available on the test server is more than sufficient; memory pressure was not observed. In Table 7 totals, in SQL Server's memory manager, index pages resident in the buffer pool at steady state are a subset of the buffer pool (query cache). The three columns (Base, Query cache, Index memory) are reported separately for interpretive clarity, but Index memory is not fully independent of Query cache. The true non-overlapping total for PROPOSED is approximately 965–1,040 MB (80–87% of the reported 1,123 MB sum), depending on the fraction of index pages in the buffer pool at the time of measurement. The reported Total column is therefore a conservative upper bound, not a sum of fully disjoint memory regions.

The higher CPU utilization for PROPOSED (89% vs. 78% for baseline-R) reflects the additional join operations required for contextualization and constraint enforcement. However, the system remained responsive with no saturation (CPU queue length < 2). For production deployments with higher concurrency, upgrading to 8 cores would reduce CPU contention.

Answer to RQ3. The resource consumption of PROPOSED is within acceptable bounds for modest hardware, confirming that the unified architecture does not require specialized or high-end infrastructure.

4.6 API Layer Performance

The REST API layer (ASP.NET Core + Swagger) was tested separately for 25 endpoints:

- Simple GET endpoints (term lookup): 2.34 ms (includes JWT validation + database call)
- POST endpoints (insert relation): 5.67 ms (includes permission check + usp_InsertRelation)
- Complex GET endpoints (transitive closure): 28.34 ms

Swagger UI responsiveness: The documentation page loaded in 1.2 seconds.

Permission enforcement accuracy: 111 of 112 permission rules correctly enforced (99.1%).

The API layer adds approximately 1.2–2.5 ms overhead compared to direct database calls, which is acceptable for web-based knowledge systems. The Swagger integration provides interactive documentation and client SDK generation—significant productivity advantages for API consumers. The 99.1% permission accuracy is acceptable for a prototype; the single failure was traced to a misconfigured Permissions entry, not a code defect.

Answer to P5 (service-oriented access control). The API layer successfully demonstrates that service-bound permissions can be implemented efficiently on the Microsoft stack.

4.7 Summary of Key Findings and Answers to Research Questions

The experimental results yield six principal findings:

- Write overhead is significant but acceptable. The proposed model incurs 66–100% higher write latency than baseline-R (absolute: 3.78–6.42 ms) but guarantees 99.4% constraint enforcement accuracy.
- Native graph traversal remains faster for deep paths. Baseline-G outperforms PROPOSED by 4.8× for Q3 (depth-5 closure) and 3.1× for Q4. The gap is mitigable by SQL Server's hierarchyid.
- Constraint validation at storage time improves read performance. For Q5, PROPOSED is 45% faster than baseline-G and 84% faster than baseline-R.
- Contextualization is uniquely supported. No baseline system provides native support for C1, C2, or C3. PROPOSED delivers these capabilities with latencies of 3.12–5.67 ms.
- OBDA is consistently outperformed (5–12× slower). Baseline-O exhibited the worst performance across all query types.
- Semantic governance is enforceable. The proposed architecture prevents 99.4% of constraint violations at write time.

4.7.1 Answers to Research Questions

RQ1 (How can a relational design incorporate explicit semantic relationships without sacrificing integrity and query efficiency?)

The proposed model demonstrates that explicit, typed relationships can be added to a SQL Server relational backbone through the Relations table and recursive CTEs, with acceptable performance overhead (3.78 ms for relation insertion, 1.12–24.89 ms for queries). Integrity is preserved through foreign key constraints, module–relation compatibility checks, and stored

procedure-enforced constraints. The trade-off is quantifiable and justifiable for knowledge-centric workloads.

RQ2 (What architectural patterns enable integrated semantics?)

Three architectural patterns are central: (i) triadic contextualization (Term–Domain–Module) decouples term identity from contextual meaning; (ii) type-governed relationships with module whitelisting (MRT) ensures semantic validity; and (iii) declarative constraint storage (RelationConstraints) externalizes validation logic into stored procedures. These patterns, implemented in T-SQL, produce a single coherent schema on SQL Server.

RQ3 (How does the unified model address the limitations of existing paradigms?)

The proposed model successfully addresses: (i) the implicit semantics limitation of relational models (through explicit Relations); (ii) the mapping complexity and performance overhead of OBDA (by eliminating the external mapping layer); (iii) the weak schema discipline of knowledge graphs (through MRT, RelationConstraints, and state enforcement); and (iv) the conceptual fragmentation of hybrid architectures (by providing a single model). The remaining limitation—deep traversal performance compared to Neo4j—is characterized and mitigable.

4.8 Comparison with Original Hardware Configuration

Table 8 compares key performance metrics between a high-end configuration (12-core Xeon, NVMe SSD) and the current configuration (4-core Xeon, SATA SSD) for PROPOSED.

Table 8. Performance Comparison: High-End Hardware Vs. Current Hardware (PROPOSED Only)

Metric	High-end (12-core, NVMe)	Current (4-core, SATA SSD)	Slowdown factor
W2 (Insert relation)	2.23 ms	3.78 ms	1.70×
Q1 (Term lookup)	0.61 ms	1.12 ms	1.84×
Q3 (Transitive closure)	15.67 ms	24.89 ms	1.59×
Q5 (Constraint query)	2.14 ms	3.45 ms	1.61×

The slowdown is approximately 1.6–1.8× across all operations, consistent with the reduction in CPU cores (12 → 4) and storage bandwidth (NVMe → SATA SSD). Importantly, the relative performance relationships between PROPOSED and baselines remained unchanged, confirming that the architectural conclusions are robust across hardware configurations.

Table 9 provides a fine-grained comparison of ten representative prior works against the proposed model across eight capability dimensions, making the specific contribution of this work and its differentiation from the literature visually explicit.

Table 9. Comparison of Representative prior Works VS. PROPOSED Mode
(✓ = Fully Supported, ◐ = Partial/Workaround, ✗ = Not Supported)

Work (Year)	Explicit Typed Relations	Triadic Context (TMD)	Write-time Constraints	MRT Governance	ACID Transactions	Native SQL Engine	Lexical Control Layer	Service Access Control
Silberschatz et al. (2020) — Relational	✗	✗	✗	✗	✓	✓	✗	✗
Gruber (1995) — Ontology	✓	◐	◐	✗	✗	✗	✓	✗
Calvanese et al. (2017) — OBDA/Ontop	✓	◐	◐	✗	◐	◐	✓	✗
Hogan et al. (2021) —	✓	✗	✗	✗	◐	✗	✗	✗

Work (Year)	Explicit Typed Relations	Triadic Context (TMD)	Write-time Constraints	MRT Governance	ACID Transactions	Native SQL Engine	Lexical Control Layer	Service Access Control
Knowledge Graphs Auer et al. (2007) — DBpedia	✓	✗	✗	✗	✗	✗	✗	✗
Stonebraker et al. (2016) — Polystore	◐	✗	✗	✗	✓	◐	✗	✗
Lu & Holubóvá (2019) — Multi-model	◐	✗	✗	✗	◐	◐	✗	✗
W3C SKOS (2009) — Thesaurus	◐	✗	✗	✗	✗	✗	✓	✗
Xiao et al. (2018) — OBDA Survey	✓	✗	◐	✗	◐	◐	✓	✗
Angles et al. (2017) — Graph Query	✓	✗	✗	✗	✗	✗	✗	✗
PROPOSED (This work)	✓	✓	✓	✓	✓	✓	✓	✓

4.9 Limitations of the Study

Several limitations of this study should be acknowledged:

- Deep traversal performance. The 4.8× gap for depth-5 traversals on 4-core hardware is a limitation for traversal-heavy workloads. Materialized paths (hierarchyid in SQL Server) are the recommended mitigation.
- Reasoning capabilities. The proposed model does not include a native reasoner. Applications requiring Description Logic reasoning would need an external reasoner.
- Design complexity. The schema (18 tables, 42 constraints, 5 stored procedures) imposes a steep learning curve. Smaller projects may find the overhead unjustified.
- Evaluation scale. The dataset (5,000 terms, 18,547 relations) is moderate. Performance at much larger scales (millions of terms) may differ.
- Domain specificity. The evaluation was conducted within Islamic sciences. Generalizability to other domains requires further validation.
- Stored procedure dependency. All writes must go through stored procedures; direct INSERT operations bypassing procedures would violate constraints. This is enforceable through database permissions.
- Concurrency limitations. On 4-core hardware, latency degradation occurs under >50 concurrent users. Production deployments should consider more cores or partitioning strategies.

These limitations do not invalidate the contributions but bound their applicability. Future work must address each.

5. Conclusion

This research addressed a fundamental gap in database design for knowledge-centric systems: the absence of a unified, implementable architecture that simultaneously provides relational integrity, explicit semantic relationships, contextualized term meaning, strong governance, and service-oriented access control within a single conceptual framework. Review of the literature (Section 2) revealed a fragmented landscape where pure relational models sacrifice semantics,

knowledge graphs sacrifice integrity and schema discipline, OBDA suffers from mapping complexity and performance overhead, multi-model systems achieve only physical integration, and thesauri remain isolated from core database design.

In response, we proposed a unified relational-semantic architecture (Section 3) grounded in five design principles: term-entity independence, contextualized semantics through triadic grounding (Term–Domain–Module), relationships as first-class citizens, governance through typed schemas, and service-oriented access control. The model was fully specified with 18 logical tables, integrity rules (IR1–IR5), and implemented on Microsoft SQL Server 2019 with stored procedures for write operations and a C# ASP.NET Core REST API with Swagger documentation.

A rigorous empirical evaluation (Sections 4) on a 4-core Intel Xeon E5-2600 V3, 32 GB RAM, 300 GB SSD, Windows Server 2022 environment compared the proposed model against three baselines: pure relational (baseline-R), pure knowledge graph (Neo4j, baseline-G), and OBDA (Ontop + SQL Server, baseline-O). Key findings from this specific hardware configuration include:

Semantic expressiveness: The proposed model provides native support for contextualized term definitions across domains and modules (C1, C2, C3)—capabilities absent from all baselines.

Write performance overhead: The model incurs 66–100% higher write latency than baseline-R (absolute: 3.78–6.42 ms) but guarantees 99.4% constraint enforcement accuracy, shifting validation from read time to write time.

Query performance: For simple lookups (Q1, Q2), the model performs within 43–44% of baseline-G (1.12 ms and 2.08 ms, respectively). For deep transitive closure (Q3, depth 5), baseline-G is 4.8× faster (5.23 ms vs. 24.89 ms)—a limitation mitigable by SQL Server's hierarchyid materialized paths. For constraint-validated queries (Q5), the proposed model is 45% faster than baseline-G (3.45 ms vs. 6.23 ms).

OBDA inferiority: Baseline-O was consistently 5–12× slower than the proposed model, confirming that adding a semantic layer atop legacy schemas is inefficient, particularly on SQL Server where Ontop's query rewriting generates poorly optimized nested queries.

Conceptual coherence: The proposed model received a 4.7/5 rating for conceptual integration versus 2.1–3.0/5 for baselines, directly addressing the fragmentation problem.

API layer effectiveness: The ASP.NET Core + Swagger implementation successfully exposed 25 REST API endpoints with JWT authentication, achieving sub-6 ms latency for simple operations and 99.1% permission enforcement accuracy.

5.1 Answers to Research Questions

RQ1 (How can a relational database design incorporate explicit, typed, and context-sensitive semantic relationships without sacrificing integrity and query efficiency?)

The proposed model demonstrates that explicit semantic relationships can be incorporated into a SQL Server relational design through the Relations table (first-class relationship storage with type references), the TMD table (contextualization), and recursive CTEs for traversal. Integrity is preserved through foreign key constraints, module–relation compatibility checks (MRT), and stored procedure-enforced constraints (RelationConstraints). Query efficiency, while not matching Neo4j for deep traversals (24.89 ms vs. 5.23 ms for depth-5), is acceptable for typical knowledge workloads and can be optimized with SQL Server's hierarchyid materialized paths. The trade-off is explicit, quantifiable, and justifiable.

RQ2 (What architectural patterns enable the simultaneous integration of terms, domains, modules, relationship types, and constraints within a single, coherent schema?)

Three architectural patterns are central:

- triadic contextualization (Term–Domain–Module) decouples term identity from contextual meaning;
- type-governed relationships with module whitelisting (MRT) ensures semantic validity;
- declarative constraint storage (RelationConstraints) externalizes validation logic into stored procedures. These patterns, implemented in T-SQL, produce a single coherent schema on SQL Server rather than a federation of heterogeneous models.

RQ3 (To what extent does the proposed unified model address the documented limitations of pure relational, pure graph, and ontology-only approaches?)

The proposed model successfully addresses: (i) the implicit semantics limitation of relational models (through explicit Relations); (ii) the mapping complexity and performance overhead of OBDA (by eliminating the external mapping layer entirely); (iii) the weak schema discipline of knowledge graphs (through MRT, RelationConstraints, and state enforcement); and (iv) the conceptual fragmentation of hybrid architectures (by providing a single model). The remaining limitation—deep traversal performance compared to Neo4j—is characterized and mitigated through hierarchyid optimization.

5.2 Theoretical Implications

This research makes three theoretical contributions to the field of database and knowledge management:

First, a reframing of the data–knowledge dichotomy. By embedding explicit semantics within a relational backbone on SQL Server, we demonstrate that the distinction is not inherent to the storage model but a consequence of design choices.

Second, contextualization as a first-class design dimension. The TMD pattern elevates context from an application-level concern to a schema-level construct.

Third, validation-at-storage as a performance strategy. Write-time constraint enforcement improves read performance for constraint-validated queries (Q5), challenging conventional wisdom that validation should be deferred to query time.

5.3 Practical Implications for the Microsoft Stack

For practitioners using the Microsoft technology stack, the proposed architecture offers:

- A blueprint for knowledge system design on SQL Server. The complete T-SQL schema definition, stored procedures, and integrity rules provide a reusable template for teams building digital libraries, encyclopedias, thesauri, and scholarly knowledge bases.
- Integration with existing Microsoft investments. Organizations already using SQL Server, ASP.NET Core, and Windows Server can adopt the model without introducing new platforms (Neo4j, PostgreSQL, etc.), reducing licensing costs and operational complexity.
- SQL Server-specific optimization strategies. Three optimizations—hierarchyid for materialized paths, filtered indexes for approved terms, and indexed views for denormalized context caching—offer practical guidance for production deployments.
- A rationale for avoiding OBDA in greenfield projects. The 5–12× performance disadvantage of OBDA provides evidence for practitioners to choose semantic-native designs over mapping-based approaches on SQL Server.

- API-first design with Swagger. The ASP.NET Core + Swagger implementation demonstrates how the service-oriented access control layer can be implemented with minimal effort, providing interactive documentation and client SDK generation.

5.4 Limitations Revisited

The limitations acknowledged in Section 6.2 bear repeating for the SQL Server/Windows environment: (i) deep traversal performance on 4-core hardware (4.8× gap vs. Neo4j), (ii) lack of native reasoning, (iii) schema complexity, (iv) moderate evaluation scale (5,000 terms), (v) domain specificity, (vi) stored procedure dependency for writes, and (vii) concurrency limitations on 4-core hardware (latency degradation under >50 concurrent users). These limitations do not invalidate the contributions but bound their applicability. Future work should address each.

5.5 Future Research Directions

The proposed architecture opens several avenues for future investigation, with specific attention to the Microsoft ecosystem.

5.5.1 Short-Term Extensions (1–2 years)

Materialized path optimization with hierarchyid. Implement and benchmark SQL Server's native hierarchyid data type for transitive closure queries. Preliminary estimates suggest 90–95% latency reduction for Q3 (from 24.89 ms to <2 ms), potentially closing the gap with Neo4j entirely.

Integration with SQL Server Graph Database. SQL Server 2019 and later include native graph extensions (NODE and EDGE tables). Future work should investigate hybrid designs where TMD records are stored as nodes and Relations as edges, combining contextualization with SQL Server's built-in graph MATCH function.

Azure SQL Database migration and benchmarking. Replicate the evaluation on Azure SQL Database (hyperscale or serverless tier) to assess cloud performance, automatic indexing, and geo-replication capabilities. This would inform organizations considering cloud migration.

Development of a .NET client library. Create a strongly-typed C# client library (NuGet package) that encapsulates the TMD–Relations pattern, providing LINQ-style queries that compile to T-SQL stored procedure calls. This would reduce the learning curve for .NET developers.

Integration with LLMs for automated term extraction. Investigate the integration of the proposed architecture with Azure OpenAI Service for automated term extraction, relationship suggestion, and constraint generation from unstructured text corpora. As Mansoori (2025) emphasized in the context of responsible AI implementation, any integration of large language models for knowledge extraction must incorporate explainability and human-in-the-loop oversight to maintain semantic integrity.

5.5.2 Medium-Term Extensions (2–4 years)

Temporal and versioned knowledge. Extend the model to support temporal scoping of relationships (ValidFrom, ValidTo) and versioning of terms and contextualizations. SQL Server's system-versioned temporal tables (PERIOD FOR SYSTEM_TIME) provide a natural implementation path.

Full-text and semantic search integration. Integrate SQL Server's full-text search and semantic search capabilities with the TMD model, enabling hybrid queries that combine keyword search with semantic relationship traversal.

Scalability benchmark at industrial scale. Replicate the evaluation with 1 million terms, 50 million relations, and 100 concurrent users on SQL Server 2022 with OPTIMIZE_FOR_SEQUENTIAL_KEY and accelerated database recovery (ADR).

Power BI and analytics integration. Develop a semantic layer (Tabular Model) on top of the proposed schema to enable Power BI reporting and analytics on knowledge graphs, including DAX expressions for transitive closure.

5.5.3 Long-Term Directions (4+ years)

Formal semantics and verification. Develop a formal denotational semantics for the proposed model, enabling verification of query equivalence and constraint satisfiability using theorem provers.

Cross-platform benchmark suite. Develop a standardized benchmark for knowledge-centric databases that runs on SQL Server, PostgreSQL, Neo4j, and graph-blob hybrids, enabling fair cross-platform comparison.

Decentralized and federated deployments. Extend the architecture to support federated queries across multiple SQL Server instances (using Linked Servers or PolyBase) with a common upper ontology and cross-instance term alignment protocols.

Automated constraint discovery with ML. Develop machine learning methods (association rule mining, graph neural networks) to discover candidate RelationConstraints from existing relationship instances, reducing manual constraint authoring effort for large-scale knowledge migration projects.

5.6 Concluding Remarks

The journey from data to knowledge is neither automatic nor trivial. For decades, database design has been optimized for structured, stable, and semantically impoverished data—a model that served transactional systems well but leaves knowledge-centric systems struggling to represent meaning, context, and relationship.

This research has proposed, implemented on Microsoft SQL Server 2019, and evaluated on realistic 4-core Windows Server hardware a unified relational-semantic architecture that bridges this gap. The model does not reject the relational paradigm; it extends it with explicit semantics, contextualization, and governance while preserving its core strengths of integrity, transactions, and accessibility. The empirical results demonstrate that this unification is not only conceptually coherent but practically feasible on modest hardware (4 cores, 32 GB RAM), with acceptable performance trade-offs for real-world knowledge systems.

The architecture is not a panacea. It imposes design complexity, sacrifices some traversal performance (mitigable with hierarchyid), and awaits validation at larger scales. But it offers a concrete, implementable alternative to the fragmentation that currently characterizes the field. For researchers, it opens new questions at the intersection of databases, knowledge representation, and software architecture. For practitioners on the Microsoft stack, it provides a blueprint—complete with T-SQL schemas, stored procedures, and C# APIs—for building systems that treat knowledge not as an afterthought but as a first-class citizen.

The title of this paper posed a challenge: "From Data to Knowledge." The proposed architecture demonstrates that the transition is possible—not by abandoning relational databases, but by redesigning them with semantics at the core, running efficiently on the hardware and software platforms that organizations already own.

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). Foundations of databases. Addison-Wesley.

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017). Foundations of modern graph query languages. ACM Computing Surveys, 50(5), Article 68, 1–40. https://doi.org/10.1145/3104031

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A nucleus for a web of open data. In Proceedings of the 6th International Semantic Web Conference (ISWC 2007) (pp. 722–735). https://doi.org/10.1007/978-3-540-76298-0_52

Batini, C., Ceri, S., & Navathe, S. B. (1992). Conceptual database design: An entity-relationship approach. Benjamin-Cummings.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific American, 284(5), 34–43. https://doi.org/10.1038/scientificamerican0501-34

Boncz, P., & Manegold, S. (2020). Repeatable database benchmarking. In Proceedings of the 46th International Conference on Very Large Data Bases (VLDB 2020 Tutorial).

Brachman, R. J., & Levesque, H. J. (2004). Knowledge representation and reasoning. Morgan Kaufmann.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., & Xiao, G. (2017). Ontology-based data access: A survey. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017) (pp. 5146–5150).

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., & Rosati, R. (2007). Ontology-based data access. In Proceedings of the 19th International Conference on Knowledge Representation and Reasoning (KR 2007) (pp. 1–11).

Elmasri, R., & Navathe, S. B. (2016). Fundamentals of database systems (7th ed.). Pearson.

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). Ontological engineering. Springer. https://doi.org/10.1007/b97353

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43(5–6), 907–928. https://doi.org/10.1006/ijhc.1995.1081

Haghshenas Nasrabadi, A., & Ghasemy Yaghin, R. (2025). The role of data quality and visibility in risk management and performance optimization in the downstream supply chain. Journal of Data Analytics and Intelligent Decision-making, 1(1), 23–32. https://doi.org/10.22091/jdaid.2025.14090.1005

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A. C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge graphs. ACM Computing Surveys, 54(4), Article 71, 1–37. https://doi.org/10.1145/3447772

Lu, J., & Holubová, I. (2019). Multi-model databases: A new journey to handle the variety of data. ACM Computing Surveys, 52(3), Article 55, 1–38. https://doi.org/10.1145/3301281

Mansoori, A. (2025). The application of artificial intelligence in human resource performance appraisal: A conceptual framework for responsible implementation. Journal of Data Analytics and Intelligent Decision-making, 1(3), 13–30. https://doi.org/10.22091/jdaid.2025.14114.1009

Microsoft Corporation. (2019). SQL Server 2019 documentation. https://docs.microsoft.com/en-us/sql/sql-server/

Microsoft Corporation. (2021). ASP.NET Core documentation. https://docs.microsoft.com/en-us/aspnet/core/

Neo4j, Inc. (2023). Neo4j graph database platform (Version 5.8) [Computer software]. https://neo4j.com

Ontop. (2023). Ontop OBDA framework (Version 4.5.0) [Computer software]. https://ontop-vkg.org

Rodriguez-Muro, M., & Calvanese, D. (2012). High performance query answering over DL-Lite ontologies. In Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012) (pp. 308–318).

Shahabi, V., Toloie Eshlaghy, A., Amiri, M., & Mehdiabadi, A. (2025). Designing a social banking model for developing micro enterprises using a system dynamics approach. Journal of Data Analytics and Intelligent Decision-making, 1(2), 23–41. https://doi.org/10.22091/jdaid.2025.14318.1012

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database system concepts (7th ed.). McGraw-Hill.

Stonebraker, M., Brown, P., Poliakov, A., & Raman, S. (2016). The case for polystores. IEEE Data Engineering Bulletin, 39(4), 3–11.

Swagger. (2023). Swagger/OpenAPI tools. https://swagger.io

W3C. (2009). SKOS Simple Knowledge Organization System reference. https://www.w3.org/TR/skos-reference

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer. https://doi.org/10.1007/978-3-642-29044-2

Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., & Zakharyaschev, M. (2018). Ontology-based data access: A survey. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018) (pp. 5511–5519). https://doi.org/10.24963/ijcai.2018/777