République algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Boudiaf de M'sila
Faculté des Mathématiques et de l'Informatique
Département de l'Informatique

# THÈSE

Présenté par :
LAKEHAL Meftah

En vue de l'obtention du diplôme de

## Doctorat en Sciences

**Filière** : Informatique

**Spécialité** : Informatique

## SUJET

Traitement des requêtes basé sur les ontologies et l'apprentissage automatique

Soutenue publiquement le : …/…/2025

**Jury :**

| | | | |
|---|---|---|---|
| Président : | Pr. | Akhrouf Samir | Université de M'sila |
| Promoteur : | Pr. | Borahla Mustapha | Université de M'sila |
| Examinateur : | Pr. | Zouache Djaafar | Université de BBA |
| Examinateur : | MCA. | Azizi Tarek | Université de Tamanrasset |

2025/2026

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed Boudiaf University of M'Sila
Faculty of Mathematics and Computer Science
Department of Computer Science

# THESIS

Submitted by:
LAKEHAL Meftah

In partial fulfillment of the requirements for the degree of

## Doctorate of Science

**Field:** Computer Science
**Specialization:** Computer Science

## Subject

Query processing based on ontologies and machine learning

Publicly defended on: …/…/2025

**Thesis Committee:**

| | | | |
|---|---|---|---|
| President : | Pr. | Akhrouf Samir | University of M'sila |
| Supervisor : | Pr. | Borahla Mustapha | University of M'sila |
| Examiner : | Pr. | Zouache Djaafar | University of BBA |
| Examiner : | MCA. | Azizi Tarek | University of Tamanrasset |

2025/2026

## Dedication

To my parents,

To my wife,

To my children..

## Acknowledgements

First and foremost, I would like to thank Almighty God for granting me the strength and courage to carry out and complete this thesis.

I wish to express my sincere gratitude to my thesis supervisor, Pr. Mustapha Borahla, for his guidance and continuous support.

I would also like to extend my appreciation to the examiners who kindly accepted to serve on the jury of this thesis and for their valuable comments.

Finally, I am grateful to all those who, in one way or another, contributed to the accomplishment of this work.

## ملخص

في هذه الأطروحة، نقدم إطاراً هجينا يجمع بين التعلم الآلي الخاضع للإشراف والاستدلال الدلالي القائم على الأنطولوجيا بهدف تحسين تنفيذ استعلامات SQL في أنظمة قواعد البيانات العلائقية. يتم أولاً تحويل المخطط العلائقي إلى أنطولوجيا بلغةOWL ، حيث يتم تمثيل دلالات المجال والقيود الخاصة بالأعمال بشكل رسمي على شكل بديهيات TBox .انطلاقاً من سجلات الاستعلامات السابقة، نقوم ببناء مجموعة بيانات معنونة تُستخدم لتدريب مصنّف من نوع **Random Forest** قادر على التمييز بين الاستعلامات البسيطة والمعقدة اعتماداً على الخصائص البنائية وخصائص التنفيذ، بما في ذلك عدد عمليات الربط(Joins) ، وجود شروط التصفية (WHERE)، وأزمنة التنفيذ المسجلة. يتم بعد ذلك إخضاع الاستعلامات المصنفة كمعقدة إلى مرحلة إعادة كتابة دلالية، موجهة بواسطة قواعد الاستدلال المرمّزة في الأنطولوجيا. تهدف هذه المرحلة إلى تقليل استرجاع البيانات غير الضرورية أو المكررة، مما يؤدي إلى تحسين كفاءة تنفيذ الاستعلامات. أظهرت النتائج التجريبية أن النهج المقترح يحقق دقة تصنيف عالية ويؤدي إلى تحسينات ملموسة في أداء التنفيذ. وبشكل عام، تبرز هذه النتائج مزايا الجمع بين تمثيل المعرفة وتقنيات التعلم الآلي لتطوير استراتيجيات ذكية وواعية بالسياق لتحسين الاستعلامات في بيئات قواعد البيانات العلائقية.



**الكلمات المفتاحية: إعادة كتابة الاستعلامات، SQL، الأنطولوجيات، التعلم الآلي، Random Forest.**

# Abstract

In this dissertation, we introduce a hybrid framework that integrates supervised machine learning with ontology-based semantic reasoning to optimize the execution of SQL queries in relational database systems. The relational schema is first transformed into an OWL ontology, where domain semantics and business constraints are formally represented as TBox axioms. Using historical query logs, we construct a labeled dataset that is employed to train a Random Forest classifier capable of distinguishing between simple and complex queries based on structural and execution-related features, including the number of joins, the presence of selection predicates (WHERE clauses), and observed execution times. Queries classified as complex are subsequently processed through a semantic rewriting stage, guided by inference rules encoded in the ontology. This rewriting step reduces redundant or irrelevant data retrieval, thereby improving query execution efficiency. Experimental results demonstrate that the proposed approach achieves high classification accuracy and yields significant performance gains. Collectively, these findings highlight the advantages of combining knowledge representation with machine learning techniques to enable intelligent, context-aware query optimization strategies in relational database environments.



**Keywords:** Query, SQL, Ontologies, Machine Learning, Random Forest classifier, Rewriting.

## Résumé

Dans cette thèse, nous présentons un cadre hybride qui intègre l'apprentissage supervisé et le raisonnement sémantique basé sur une ontologie afin d'optimiser l'exécution des requêtes SQL dans les systèmes de bases de données relationnelles. Le schéma relationnel est d'abord transformé en une ontologie OWL, où la sémantique du domaine et les contraintes métiers sont formellement représentées sous forme d'axiomes TBox. À partir des journaux de requêtes historiques, nous construisons un jeu de données étiqueté, utilisé pour entraîner un classificateur Random Forest capable de distinguer les requêtes simples des requêtes complexes en se basant sur des caractéristiques structurelles et liées à l'exécution, notamment le nombre de jointures, la présence de prédicats de sélection (clauses WHERE) et les temps d'exécution observés. Les requêtes identifiées comme complexes sont ensuite soumises à une phase de réécriture sémantique, guidée par les règles d'inférence encodées dans l'ontologie. Cette étape de réécriture permet de réduire les récupérations de données redondantes ou non pertinentes, améliorant ainsi l'efficacité de l'exécution des requêtes. Les résultats expérimentaux démontrent que l'approche proposée atteint une précision de classification élevée et offre des gains de performance significatifs. Dans l'ensemble, ces résultats mettent en évidence les avantages de la combinaison de la représentation des connaissances et de l'apprentissage automatique pour développer des stratégies d'optimisation de requêtes intelligentes et sensibles au contexte dans les environnements de bases de données relationnelles.



**Mots-clés :** requête, SQL, Ontologies, Apprentissage automatique, Random Forest classifier, réecriture.

# Table of Contents

# List of figures

# List of tables

# General Introduction

The ever-accelerating proliferation of digital data has established relational databases as the bedrock of modern information systems, underpinning everything from global ecommerce platforms to critical scientific research. The Structured Query Language (SQL) has long served as the standard for interacting with these databases, offering a powerful and expressive means of retrieving and manipulating data. However, the very success of this paradigm has given rise to new and formidable challenges. As the volume and complexity of data continue to grow unabated, the task of formulating efficient and accurate SQL queries has become increasingly difficult, often requiring deep technical expertise and a nuanced understanding of the underlying database schema.

This complexity creates a significant bottleneck in many data-intensive applications. Inefficiently formulated queries can consume excessive computational resources, leading to slow response times and degraded system performance. More insidiously, queries that are syntactically correct but semantically flawed can return results that are incomplete, misleading, or simply wrong. This "semantic gap" between the user's intent and the query's actual meaning represents a fundamental challenge in the field of data management. It is a gap that traditional query optimization techniques, which primarily focus on syntactic and cost-based analysis, are often ill-equipped to bridge.

To address this challenge, the research community has explored a variety of approaches aimed at imbuing database systems with a deeper understanding of the data they manage. One of the most promising of these is Ontology-Based Data Access (OBDA). OBDA systems employ a formal ontology, a machine-readable representation of a domain's concepts and their relationships, to create a semantic layer over the underlying database. This allows users to query the data using high-level, intuitive terms, while the system takes on the responsibility of translating these queries into the appropriate SQL. While OBDA has proven to be a powerful paradigm, it is not without its own limitations. Traditional OBDA systems often rely on static, hand-crafted rules for query rewriting, which can be brittle and difficult to maintain. Furthermore, they typically lack the ability to learn from experience and adapt to changing query patterns or data distributions.

In parallel, the field of machine learning (ML) has made remarkable strides in recent years, demonstrating an uncanny ability to learn complex patterns from data and make accurate predictions. This has led to a surge of interest in applying ML techniques to various aspects of database management, from performance tuning to anomaly detection. The central thesis of this dissertation is that a powerful synergy can be achieved by combining the semantic reasoning capabilities of ontologies with the predictive power of machine learning. We argue that this hybrid approach can lead to a new generation of intelligent query optimization systems that are not only more efficient and accurate but also more adaptive and robust.

This dissertation presents a novel framework for SQL query optimization that realizes this vision. The proposed framework integrates a supervised machine learning model with an ontology-driven query rewriting mechanism. The ML model, a Random Forest Classifier, is trained to automatically classify incoming SQL queries as either "simple" or "complex" based on a set of learned features. Queries classified as "complex" are then passed to a rewriting module that leverages a domain ontology to semantically enrich and optimize the query. This approach allows the system to dynamically tailor its optimization strategy to the specific characteristics of each query, applying the more computationally intensive semantic analysis only when it is most needed.

The research presented in this dissertation is guided by the following key questions:

1. **Can a supervised machine learning model be trained to accurately and reliably classify the complexity of SQL queries?** This question explores the feasibility of using machine learning to automate a critical aspect of the query optimization process. It investigates whether a model can learn to identify the subtle cues that distinguish a simple, efficient query from a complex, resource-intensive one.
2. **How can a domain ontology be effectively utilized to rewrite complex SQL queries in a way that improves both their performance and their semantic correctness?** This question delves into the practical application of ontologies in the query optimization process. It examines how the formal semantics of an ontology can be leveraged to transform a complex query into a more efficient and semantically sound equivalent.
3. **What is the overall effectiveness of a hybrid query optimization framework that combines machine learning and ontologies, and how does it compare to traditional approaches?** This question addresses the central value proposition of our research. It seeks to quantify the benefits of our hybrid approach in terms of performance, accuracy, and adaptability, and to benchmark it against existing methods.

To answer these questions, we have designed and implemented a prototype of the proposed framework and conducted a rigorous set of experiments in a simulated ecommerce environment. The results of these experiments, which are detailed in the subsequent chapters of this dissertation, provide strong evidence for the feasibility and effectiveness of our approach.

This dissertation is structured as follows:

- **Chapter 1: Ontologies and Description Logics** provides a comprehensive introduction to the theoretical foundations of our work, covering the key concepts of ontologies, Description Logics (DL), and the Web Ontology Language (OWL). This chapter lays the groundwork for understanding the semantic reasoning capabilities that are at the heart of our proposed framework.

- **Chapter 2: Ontology-Based Data Access (OBDA)** delves into the OBDA paradigm, exploring its architecture, key components, and the state of the art in the field. This chapter provides the context for our work, highlighting both the promise and the limitations of existing OBDA systems.
- **Chapter 3: Our Contribution: Proposed Approach** presents the novel hybrid framework for SQL query optimization that is the core contribution of this research. This chapter details the design and implementation of the machine learning model and the ontology-driven query rewriting mechanism, explaining how these two components work together to create a more intelligent and adaptive query optimization system.

Finally, the dissertation concludes with a summary of our findings, a discussion of their implications, and a look ahead to future research directions. In summary, this dissertation makes a significant contribution to the field of database management by proposing and validating a novel hybrid approach to SQL query optimization. By integrating the complementary strengths of machine learning and ontologies, we have developed a framework that is not only more powerful and flexible than existing approaches but also offers a promising new path towards the realization of truly intelligent data management systems. This work is a testament to the power of interdisciplinary research, drawing on insights from database systems, knowledge representation, and artificial intelligence to address a problem of both theoretical and practical importance.

# Chapter 1 Ontologies and Description Logics

## 1.1 Introduction

The exponential growth of digital information and the increasing complexity of modern data ecosystems have created an urgent need for sophisticated knowledge representation and reasoning mechanisms[1]. Ontologies have emerged as fundamental components of semantic technologies, providing the theoretical and practical foundation for organizing, sharing, and reasoning about knowledge in machine interpretable formats[2]. This chapter examines ontologies and their underlying theoretical framework based on Description Logics (DL), establishing essential groundwork for understanding advanced semantic data access paradigms such as Ontology-Based Data Access (OBDA).

The discussion encompasses Description Logics as the formal logical foundation underlying modern ontology languages such as the Web Ontology Language (OWL)[3]. We investigate syntax and semantics of Description Logics, exploring key concepts including concepts (classes), roles (properties), and individuals (instances). The chapter addresses practical aspects of ontology engineering, examining established methodologies and best practices for ontology development, maintenance, and evolution[4].

## 1.2 Fundamentals of Ontologies

### 1.2.1 Classical Definitions

The foundational definition shaping the field was provided by Gruber in 1993, characterizing an ontology as "an explicit specification of a conceptualization" [5]. This definition emphasizes explicit specification requiring formal and unambiguous expression, and conceptualization referring to abstract, simplified views of domains being represented. Studer and colleagues in 1998 enhanced this definition: "An ontology is a formal, explicit specification of a shared conceptualization"[6], introducing formality and sharing dimensions.

### 1.2.2 Contemporary Perspectives

Recent developments have expanded understanding of ontological foundations. The emergence of Large Language Models has introduced new possibilities for automated ontology generation and maintenance, challenging traditional manual engineering approaches[7]. Contemporary research suggests ontologies can be viewed as dynamic, evolving representations adapting to changing domain requirements rather than static knowledge artifacts[1]. Modern ontologies support complex reasoning tasks, semantic integration across heterogeneous systems, and intelligent query answering[8]. This expanded scope has led to nuanced understandings of ontology effectiveness and practical value,

particularly in applications requiring sophisticated inference capabilities and semantic interoperability[9].

### 1.2.3 Knowledge Sharing and Semantic Interoperability

Ontologies serve multiple critical functions in modern information systems, with primary purposes encompassing knowledge sharing, semantic interoperability, and intelligent data integration[10]. The fundamental role involves providing common vocabulary and shared understanding enabling different systems, applications, and users to communicate effectively about domain knowledge[11].

Knowledge sharing represents one of the most significant contributions to information management. By providing explicit definitions of concepts and relationships, ontologies enable transfer of domain expertise across organizational boundaries and temporal contexts[12]. This capability proves particularly valuable in complex domains such as biomedicine, engineering, and scientific research, where specialized knowledge must be preserved, communicated, and built upon by diverse practitioner communities[13].

### 1.2.4 Distinction from Database Schemas

The distinction between database schemas and ontologies is fundamental to understanding unique value propositions of ontological approaches. While database schemas focus on data structure and storage optimization, ontologies emphasize meaning and reasoning capabilities[14]. Database schemas typically operate under closed-world assumptions, whereas ontologies operate under open-world assumptions, acknowledging that absence of information does not necessarily imply falsity[15].

Ontologies provide automated reasoning and inference capabilities not typically available in traditional database systems. Through Description Logic-based reasoning engines, ontologies can derive new knowledge from existing facts, detect inconsistencies, and answer complex queries requiring logical inference[16]. This capability proves particularly valuable in domains requiring explicit implicit knowledge and complex relationship discovery[9].

### 1.2.5 Ontology Classification Systems

- **Scope-Based Classification**

The diversity of ontological applications has led to various classification schemes categorizing different ontology types. The most widely used scheme distinguishes between domain ontologies, task ontologies, and upper ontologies based on scope and abstraction level[14]. Domain ontologies focus on specific subject areas, providing detailed representations of concepts, relationships, and constraints relevant to a particular domain[12].

Recent examples include SNOMED CT for clinical terminology, Gene Ontology for biological processes, and emerging materials science ontologies for engineering applications[12]. Task ontologies focus on specific activities or processes, describing concepts and relationships involved in a particular task type such as planning, scheduling, or diagnosis[11]. Upper ontologies provide general concepts and relationships applicable across multiple domains, defining fundamental categories serving as foundations for specific ontological representations[10].

- **Expressiveness-Based Classification**

Another important classification distinguishes between lightweight and heavyweight ontologies based on expressiveness and complexity[15]. Lightweight ontologies primarily focus on taxonomic relationships and simple property definitions, providing basic hierarchical structures that are easy to understand and maintain[17]. Heavyweight ontologies include complex logical constraints, detailed property definitions, and sophisticated reasoning capabilities, leveraging full expressiveness of languages such as OWL DL[18].

Recent developments have introduced additional classification dimensions, including distinctions between static and dynamic ontologies, centralized and distributed ontologies, and human-authored versus automatically generated ontologies. The emergence of LLM based ontology generation techniques has highlighted importance of understanding how different construction methodologies affect ontology quality, maintainability, and usability[19].

## 1.3 Ontology Languages and Standards

### 1.3.1 Resource Description Framework (RDF) and RDF Schema

- **RDF Fundamentals**

The Resource Description Framework (RDF) serves as the foundational data model for the Semantic Web, providing basic building blocks for sophisticated ontology languages[17]. RDF enables representation of information about resources in machine-processable and web-compatible formats. The fundamental unit of information in RDF is the triple, consisting of subject, predicate (property), and object components[17]. Recent developments in RDF include the standardization of RDF 1.2, which introduces enhanced support for RDF-star and property graphs, addressing limitations in representing complex relationships and metadata about statements[20]. The W3C's ongoing work on RDF 1.2 Semantics and Concepts demonstrates continued evolution of foundational semantic web technologies[20].

- **RDF Schema Extensions**

RDF Schema (RDFS) extends the basic RDF model by providing vocabulary for describing properties and classes of RDF resources[17]. RDFS introduces fundamental concepts such as Class, Property,

subClassOf, and subPropertyOf, enabling creation of simple taxonomic hierarchies and property relationships[17]. While RDFS provides only basic modeling primitives, it establishes foundations for more expressive ontology languages and demonstrates semantic information layering on basic RDF data models[20]. The class hierarchy mechanism in RDFS allows definition of taxonomic relationships between concepts, enabling inheritance of properties and supporting basic reasoning tasks such as classification and subsumption checking[14]. Property hierarchy mechanisms provide similar capabilities for relationships, allowing definition of specialized properties inheriting characteristics from more general ones[15].

### 1.3.2 Web Ontology Language (OWL)

- **OWL Evolution and Profiles**

The Web Ontology Language (OWL) represents significant advancement in ontology representation capabilities, providing expressive frameworks for knowledge modeling beyond RDF and RDFS capabilities [18]. OWL is formally grounded in Description Logics, providing theoretical foundations for semantic properties and reasoning capabilities[15]. The evolution to OWL2 brought significant improvements in expressiveness and practical usability, introducing new language constructs, improved datatype support, and better integration with existing web technologies[18]. OWL 2 defines three profiles—EL, QL, and RL—designed to optimize reasoning performance for specific application types and use cases[18]. These profiles represent sophisticated approaches to balancing expressiveness with computational efficiency, addressing key challenges in practical ontology deployment[21]. The OWL 2 EL profile optimizes applications involving large-scale ontologies with complex hierarchical structures, supporting polynomial-time reasoning for key inference tasks[22].

- **Recent Implementation Advances**

Recent developments in OWL implementation have focused on addressing scalability challenges and supporting deployment in resource-constrained environments. The Cowl library represents significant advancement, providing OWL manipulation capabilities operating effectively on embedded systems and IoT devices[23]. This development proves particularly important as semantic technologies expand beyond traditional computing environments to encompass Internet of Everything contexts[23]. Performance evaluation studies have demonstrated that modern OWL reasoners can handle increasingly large ontologies while maintaining acceptable reasoning performance[24]. Distributed incremental ontology reasoning approaches have shown promise for handling dynamic knowledge bases with frequent updates[25].

### 1.3.3 Tools and Development Ecosystem

- **Integrated Development Environments**

The practical deployment and maintenance of ontologies depend critically on robust tools and supporting infrastructure[26]. Protégé remains the most widely used ontology development environment, with recent releases continuing to provide comprehensive OWL 2 ontology development support[26]. Plugin-based architecture enables development of specialized tools and extensions addressing various ontology development aspects[26].

Recent developments in ontology tools have been significantly influenced by advances in artificial intelligence and machine learning. Integration of Large Language Models into ontology development workflows represents a significant trend, with tools emerging that assist with ontology construction, validation, and maintenance tasks[19]. These AI-assisted approaches promise to reduce development effort while potentially improving quality and consistency of resulting ontologies[19].

- **Evaluation and Quality Assessment Tools**

The emergence of sophisticated ontology evaluation tools has addressed critical needs for quality assessment and validation[10]. Recent research demonstrates continued progress in ontology matching and alignment capabilities, with systems achieving high F-measures in evaluation initiatives[11]. Modern evaluation tools incorporate comprehensive metrics for systematic ontology quality assessment[11].

Recent research has focused on developing automated quality assessment approaches that can identify potential inconsistencies, suggest structural improvements, and validate compliance with established standards[13]. Performance evaluation frameworks for upper-level ontologies have been developed to assess semantic richness, domain coverage, extensibility, and complexity characteristics[10].

## 1.4 Description Logics (DL)

### 1.4.1 Theoretical Foundations

- **Introduction to Description Logics**

Description Logics represent a family of formal knowledge representation languages providing theoretical foundations for modern ontology languages, particularly OWL DL and OWL 2 [14]. These logics address the need for knowledge representation formalisms balancing expressiveness with computational tractability, enabling rich semantic modeling and efficient automated reasoning[16]. Description Logics emerged from confluence of semantic networks, frame-based systems, and first-order logic research traditions[14]. The fundamental motivation involves providing formal bases for representing and reasoning about conceptual knowledge in mathematically precise and

computationally feasible ways[16]. Unlike first-order logic providing maximum expressiveness but suffering from undecidability issues, Description Logics identify fragments maintaining decidability while providing sufficient expressiveness for practical applications[21].

- **Core Components and Structure**

The core components of Description Logic systems include concepts (analogous to classes), roles (analogous to properties), and individuals (analogous to instances)[14]. Concepts represent sets of objects sharing common characteristics, roles represent binary relationships between objects, and individuals represent specific entities in domains of discourse[15]. This tripartite structure provides natural and intuitive domain knowledge modeling while maintaining formal precision [16].

Description Logics distinguish themselves through emphasis on terminological reasoning, focusing on relationships between concepts and logical implications of concept definitions[14]. This emphasis makes Description Logics particularly suitable for applications involving classification, subsumption checking, and consistency verification, which are fundamental tasks in ontology-based systems[24].

### 1.4.2 Syntax and Semantics

- **Basic Description Logic ALC**

The syntax of Description Logics is built around constructor sets enabling complex concept and role expression construction from simpler components[14]. The basic Description Logic ALC (Attributive Language with Complements) provides representative examples of constructor functionality and combination possibilities[14]. In ALC, concepts can be constructed using fundamental operators including conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$), existential restrictions ($\exists$), and universal restrictions ($\forall$) [14].

The conjunction operator enables creation of concepts representing intersections of multiple concepts, enabling definition of concepts satisfying multiple conditions simultaneously[16]. The existential restriction operator enables expression of concepts based on relationship existence, while universal restriction operators provide universal quantification over relationships[15]. These constructs enable expression of sophisticated relationships and constraints essential for accurate domain modeling[14].

- **Semantic Foundations**

The semantics of Description Logics are defined using model-theoretic approaches providing precise mathematical interpretations for all syntactic constructs[14]. An interpretation consists of a domain of individuals and interpretation functions mapping concept names to domain subsets and role names to binary relations over domains[16]. This formal semantic foundation ensures unambiguous meaning of Description Logic expressions and enables sound and complete reasoning procedure proofs[24].

The interpretation of complex concept expressions follows compositional principles, where complex expression meanings are determined by component meanings and combining operators[14]. For conjunction, interpretation is intersection of component concept interpretations; for disjunction, it is union; for negation, it is complement[15]. These formal semantics provide foundations for defining key reasoning tasks including concept satisfiability, subsumption, and instance checking[16].

### 1.4.3 Knowledge Base Organization

- **TBox and ABox Distinction**

Description Logic knowledge bases are typically organized into TBox (Terminological Box) and ABox (Assertional Box) components[14]. This separation reflects fundamental distinctions between knowledge types and enables more efficient reasoning procedures and knowledge management strategies[15]. The TBox contains terminological knowledge consisting of concept definitions and general statements about concept relationships[16].

TBox statements include concept inclusions (subsumption relationships), concept equivalences (definitions), and role inclusions (property hierarchies)[14]. The ABox contains assertional knowledge consisting of statements about specific individuals in domains, including concept assertions and role assertions[15]. This distinction has important implications for reasoning efficiency and knowledge management, with TBox reasoning typically involving concept relationship determination and ABox reasoning involving specific individual fact determination[24].

- **Modular Knowledge Base Design**

Recent research has explored extensions to traditional TBox/ABox distinctions, including RBox (Role Box) for complex role definitions and NBox (Nominal Box) for handling named individuals with special properties[18]. These extensions reflect growing sophistication of Description Logic systems and applications to increasingly complex domains[8]. The separation supports modular knowledge base design and maintenance, enabling better division of labor in development and more flexible management strategies[11].

### 1.4.4 Reasoning Capabilities

- **Fundamental Reasoning Tasks**

Reasoning in Description Logics encompasses various inference tasks enabling extraction of implicit knowledge from explicit representations and verification of knowledge base consistency[16]. Primary reasoning tasks include subsumption checking, instance checking, satisfiability testing, and consistency verification[24]. Subsumption checking determines whether one concept is more general than another across all possible interpretations[14].

Instance checking determines whether particular individuals belong to specified concepts given knowledge in both TBox and ABox components[15]. Satisfiability testing determines whether concepts can have instances in interpretations satisfying knowledge bases[16]. Consistency verification determines whether knowledge bases admit at least one model [29]. These tasks form bases for more complex inference procedures and applications[8].

- **Implementation and Optimization**

The implementation of reasoning tasks relies on sophisticated algorithms developed and refined over decades of automated reasoning research[16]. Modern Description Logic reasoners employ various techniques including tableau-based methods, resolution-based methods, and automata-based methods[24]. Tableau-based reasoning represents one of the most successful approaches, working by attempting to construct knowledge base models through systematic exploration of possible interpretations[24].

Recent advances have focused on addressing scalability challenges and supporting new reasoning task types[25]. Parallel and distributed reasoning approaches have been developed to handle large-scale ontologies exceeding single-machine reasoner capacity[25]. Incremental reasoning techniques enable efficient reasoning result updates when knowledge bases are modified, supporting dynamic applications where knowledge evolves over time[23].

### 1.4.5 Expressivity and Complexity Analysis

- **Expressivity-Complexity Trade-offs**

The relationship between expressivity and computational complexity represents one of the most important considerations in Description Logic system design and application[21]. This relationship fundamentally shapes practical utility of different Description Logic variants and influences ontology language design such as OWL 2 and its profiles[18]. Expressivity refers to the range of knowledge representable using available language constructs[14].

More expressive logics can represent more complex relationships and constraints, enabling more precise and detailed domain modeling[15]. However, increased expressivity typically comes at computational complexity costs for reasoning tasks, creating fundamental tradeoffs requiring careful management in practical applications[21]. The basic Description Logic ALC provides representative baseline for understanding these trade-offs[14].

- **Complexity Classes and Practical Implications**

Extensions to ALC introduce additional expressive capabilities at increased complexity costs[21]. The Description Logic SROIQ, forming the basis for OWL 2 DL, represents highly expressive logic including role hierarchies, inverse roles, number restrictions, nominals, self-restrictions, and complex

role inclusions[18]. SROIQ provides sufficient expressivity for most practical ontological modeling tasks but has 2NEXPTIME-complete reasoning complexity[21].

The development of OWL 2 profiles represents sophisticated responses to expressivity complexity trade-offs[18]. Each profile is designed to optimize reasoning performance for specific application types while maintaining sufficient expressivity for target use cases[22]. Recent research continues exploring the expressivity-complexity landscape, identifying new Description Logic variants providing different trade-off space points[21].

## 1.5 Ontology Engineering Methodologies

### 1.5.1 Development Life Cycle Framework

- **Systematic Development Approaches**

The development of high-quality ontologies requires systematic approaches ensuring completeness, consistency, and usability of resulting knowledge representations[10]. The ontology development life cycle provides structured frameworks for managing complex processes of transforming domain knowledge into formal ontological representations[11]. This life cycle typically encompasses specification, conceptualization, formalization, implementation, and maintenance phases[27].

The specification phase establishes foundations by defining purpose, scope, and requirements[5]. Competency questions play crucial roles, serving as informal specifications of knowledge that ontologies must capture[27]. The conceptualization phase involves creating conceptual models capturing key concepts, relationships, and constraints using informal or semi-formal representations[10]. The formalization phase transforms conceptual models into formal representations using specific ontology languages[15].

- **Contemporary Methodological Adaptations**

Recent developments have introduced new considerations into traditional life cycle models. The emergence of Large Language Models as ontology development tools has created opportunities for automating certain specification, conceptualization, and formalization aspects[19]. However, automated approaches introduce new challenges related to quality assurance, validation, and maintenance requiring addressing within life cycle frameworks[10].

The integration of artificial intelligence and machine learning techniques into ontology engineering workflows has led to development of hybrid methodologies combining automated knowledge extraction with traditional manual development approaches[19]. Agile software development practices have also influenced ontology engineering methodologies, leading to more iterative and flexible approaches responding quickly to changing requirements and feedback[11].

### 1.5.2 Established Methodologies

- **Comprehensive Methodological Frameworks**

Several established methodologies provide structured approaches to ontology engineering, each with different emphases and strengths[27]. Methontology, developed by the Ontological Engineering Group at Universidad Politécnica de Madrid, represents one of the most comprehensive and widely adopted approaches[27]. Methontology provides detailed guidance for each development life cycle phase, including specific techniques for knowledge acquisition, conceptualization, and formalization[10].

The NeOn methodology represents a more recent approach emphasizing collaborative and networked ontology development[11]. NeOn recognizes that modern ontology development often involves multiple stakeholders, distributed development teams, and integration of multiple existing ontological resources[28]. The methodology provides specific guidance for managing these complexities and ensuring consistency and quality across distributed development efforts[29].

- **Practical and Educational Approaches**

The Ontology 101 methodology, developed at Stanford University, provides more lightweight and accessible approaches particularly suitable for newcomers[30]. This methodology emphasizes practical guidance and hands-on techniques rather than comprehensive process frameworks[5]. Ontology 101 provides step-by-step instructions for common development tasks and includes numerous examples and case studies[30].

Recent methodological developments have focused on incorporating new technologies and addressing emerging challenges[19]. The integration of machine learning and natural language processing techniques into development workflows has led to hybrid methodologies combining automated knowledge extraction with traditional manual approaches[31]. These developments reflect the evolving nature of ontology engineering and the need for methodologies adapting to new technological capabilities[19].

### 1.5.3 Best Practices and Quality Assurance

- **Design Principles and Standards**

The accumulated experience of the ontology engineering community has led to identification of numerous best practices significantly improving ontology quality, usability, and maintainability[10]. Modular design represents one of the most important best practices, encouraging creation of modular ontologies that can be combined and reused as needed rather than large, monolithic structures[11]. Modular design supports better maintainability, enables distributed development, and facilitates reuse across different applications and domains[28].

The principle of ontology reuse is closely related to modular design and represents another fundamental best practice[11]. Rather than developing ontologies from scratch, developers should actively seek to reuse existing ontological resources whenever possible[29]. This approach reduces development effort, improves consistency across related ontologies, and leverages expertise and validation invested in existing resources[10].

- **Quality Assessment and Validation**

Clear and consistent naming conventions are essential for creating understandable and maintainable ontologies[13]. Concept and property names should be descriptive, unambiguous, and consistent with established domain conventions[10]. The practice of competency question-driven development helps ensure ontologies are designed to meet specific functional requirements rather than being developed abstractly[5].

Recent developments in ontology engineering, best practices have been influenced by advances in artificial intelligence and machine learning[19]. The use of LLMs for ontology development introduces new considerations around validation, quality assurance, and maintenance that are still being explored by the research community[19]. Best practices for AI-assisted ontology development are emerging and likely to become increasingly important as these technologies mature[31].

## 1.6 Challenges in Ontology Design and Reasoning

### 1.6.1 Scalability and Performance Challenges

- **Large-Scale Reasoning Limitations**

The practical deployment of ontologies in real-world applications faces numerous challenges spanning technical, methodological, and organizational dimensions[25]. Scalability of reasoning for large knowledge bases represents one of the most significant technical challenges[21]. As ontologies grow in size and complexity, computational resources required for reasoning tasks can become prohibitive, limiting practical applicability of sophisticated ontological approaches[25].

The scalability challenge manifests in several dimensions including TBox size affecting concept reasoning task complexity and ABox size affecting instance reasoning task complexity[21]. The interaction between TBox and ABox reasoning creates additional complexity that can grow super-linearly with either component size[32]. Recent research has addressed scalability challenges through distributed reasoning systems, incremental reasoning systems, and approximate reasoning systems[25].

- **Resource-Constrained Environments**

The Cowl library represents significant advancement in addressing scalability challenges for resource-constrained environments, demonstrating that sophisticated OWL reasoning can be performed even

on embedded systems with severe memory and processing limitations[23]. This work opens new possibilities for deploying semantic technologies in Internet of Things and edge computing contexts where traditional reasoning systems would be impractical[23].

Performance optimization research has focused on developing specialized reasoning algorithms for restricted Description Logic fragments that exploit structural properties to achieve better performance than general-purpose procedures [27]. The EL++ algorithm provides polynomial-time reasoning for the EL family by exploiting absence of disjunction and full negation[22].

### 1.6.2 Evolution and Maintenance Challenges

- **Dynamic Knowledge Management**

Ontology evolution and consistency maintenance represent major challenge areas reflecting the dynamic nature of knowledge and the need for ontologies to adapt to changing requirements and understanding[10]. As domains evolve and new knowledge becomes available, ontologies must be updated to remain accurate and useful[1]. However, ontology changes can have far-reaching implications that are difficult to predict and manage[13].

The challenge of ontology evolution is compounded by the fact that ontologies are often used by multiple applications and stakeholders, making it difficult to coordinate changes and ensure backward compatibility[11]. Changes to core concepts or relationships can break existing applications or invalidate existing data, creating significant maintenance burdens[10]. The development of ontology versioning and evolution management systems has been an active research area[25].

- **Consistency and Quality Maintenance**

Consistency maintenance during ontology evolution is particularly challenging because changes to one ontology part can create inconsistencies in seemingly unrelated parts[13]. Automated consistency checking tools can identify these problems, but resolving them often requires deep domain knowledge and careful consideration of different resolution strategy implications[24]. The development of change impact analysis tools and semiautomated consistency repair techniques represents important ongoing research[10].

Recent developments in ontology quality assessment have provided frameworks for systematic evaluation of ontology characteristics including semantic richness, domain coverage, extensibility, and complexity[10]. Contemporary evaluation initiatives demonstrate continued progress in ontology matching and alignment capabilities, with systems achieving high performance metrics[11].

### 1.6.3 Integration and Interoperability Challenges

- **Heterogeneous Ontology Integration**

Integration of multiple heterogeneous ontologies represents a third major challenge area, reflecting the reality that most practical applications must work with knowledge from multiple sources and domains[28]. Different ontologies may use different modeling approaches, different granularity levels, and different underlying assumptions, making integration complex and error-prone[29].

The technical aspects of ontology integration involve identifying correspondences between concepts and relationships in different ontologies, resolving conflicts and inconsistencies, and creating unified representations preserving essential knowledge from all sources[28]. These tasks often require sophisticated mapping and alignment techniques handling semantic heterogeneity as well as syntactic differences[29].

- **Organizational and Governance Challenges**

The organizational aspects of ontology integration can be even more challenging than technical aspects[11]. Different ontologies may be developed and maintained by different organizations with different priorities and constraints[10]. Establishing governance structures and processes for managing integrated ontologies requires coordination across organizational boundaries and alignment of potentially conflicting interests[28].

Recent developments in ontology integration have been influenced by advances in machine learning and natural language processing, which can assist with correspondence identification and conflict resolution[19]. However, automated approaches still require significant human oversight and validation to ensure results are semantically meaningful and practically useful[31].

## 1.7 Conclusion

This chapter has provided a comprehensive examination of ontologies and Description Logics, establishing theoretical and practical foundations necessary for understanding advanced semantic technologies and their applications. We explored fundamental ontology concepts as explicit specifications of conceptualizations, tracing evolution from classical definitions of Gruber and Studer to contemporary applications in diverse domains ranging from materials science to Internet of Things systems.

Our exploration of ontology languages and standards demonstrated evolution from basic RDF triples through RDFS hierarchies to sophisticated expressiveness of OWL 2 and its specialized profiles. The development of tools such as Protégé and emergence of comprehensive ontology ecosystems have made ontology development more accessible while supporting large-scale collaborative efforts.

Advanced query answering techniques that leverage ontological knowledge to enhance database query capabilities represent important progress toward practical semantic data access systems. The concepts and principles presented establish foundations for understanding Ontology-Based Data Access (OBDA) systems, which represent major application areas for ontology technologies.

# Chapter 2 Ontology-Based Data Access (OBDA)

## 2.1 Introduction

This chapter provides a comprehensive overview of the OBDA paradigm, covering its foundational concepts, architecture, key technologies, and practical applications. We will delve into the details of each of the three layers, exploring the roles of relational databases, ontologies, and mappings in the OBDA ecosystem. We will also examine the query answering process, including query rewriting and optimization techniques, and discuss the state-of-the-art OBDA systems and tools. Finally, we will explore the challenges and future directions of OBDA research, including its role in the broader context of knowledge graphs and AI-driven data integration[19].

## 2.2 Overview of OBDA

### 2.2.1 Definition and Motivation

As depicted in figure 2.1, Ontology-Based Data Access (OBDA) is a data integration approach that uses an ontology to provide a unified and semantically rich view of data stored in one or more heterogeneous data sources. The key motivation behind OBDA is to overcome the semantic heterogeneity and schematic complexity of modern data landscapes, enabling users to access and query data in a more intuitive and meaningful way[33].

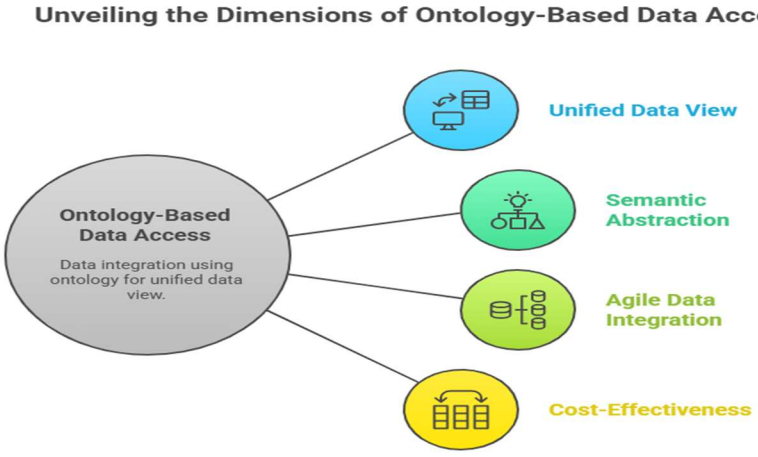


**Figure 2.1 : Unveiling the dimensions of OBDA**

In many organizations, data is distributed across numerous databases, each with its own schema, terminology, and data formats. This heterogeneity makes it challenging for users to find, understand, and integrate the data they need. Traditional data integration approaches, such as data warehousing,

require the creation of a centralized repository, which can be costly, time-consuming, and inflexible[34].

OBDA offers a more agile and scalable alternative by providing a virtual, on-demand approach to data integration. Instead of physically moving the data, OBDA systems create a virtual knowledge graph over the existing data sources, allowing users to query the data as if it were a single, unified knowledge base [23]. This approach not only reduces the cost and complexity of data integration but also ensures that users always have access to the most up-to-date data [35].

The semantic abstraction provided by the ontology is another key advantage of OBDA. By defining the concepts and relationships in a domain, the ontology provides a shared understanding of the data, which can be used to resolve semantic conflicts and ambiguities[36]. This enables users to formulate queries using high-level, domain-specific terms, without needing to know the underlying database schemas or SQL syntax[33].

### 2.2.2 Core OBDA Architecture

The core architecture of an OBDA system as shown in Figure 2.2 is typically composed of three layers: the Data Layer, the Ontology Layer, and the Mapping and Query Answering Layer. This three-layer model provides a clear separation of concerns, allowing each layer to be developed and maintained independently[33].

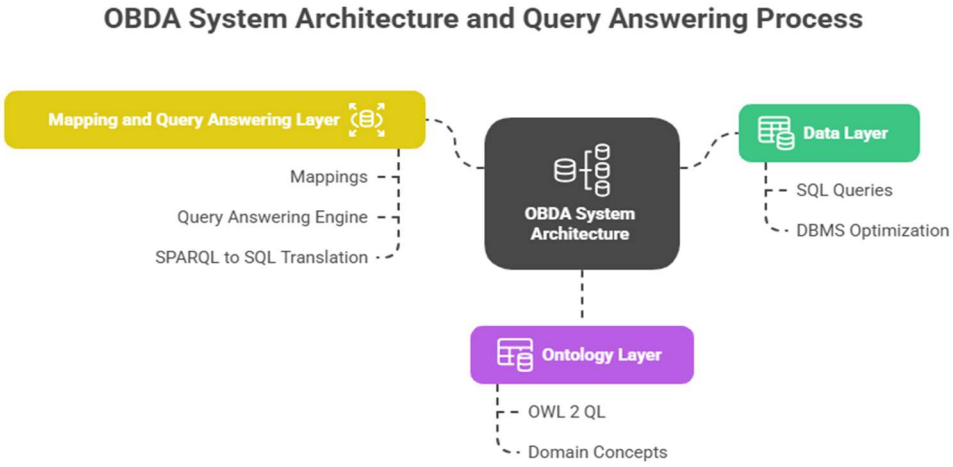


**Figure 2.2 : OBDA system architecture**

- **Data Layer**

This layer consists of the underlying relational databases that store the actual data. The data remains in its original sources and is accessed using the standard SQL query language. The DBMS is responsible for optimizing and executing the SQL queries generated by the OBDA system[33].

- **Ontology Layer**

This layer provides a conceptual schema of the domain, typically expressed in a Description Logic-based language such as OWL 2 QL. The ontology defines the concepts, properties, and relationships that are relevant to the domain, providing a unified and semantically rich view of the data[37].

- **Mapping and Query Answering Layer**

This layer acts as the semantic mediator between the Data Layer and the Ontology Layer. It consists of two main components: the mappings and the query answering engine. The mappings are a set of declarative rules that specify the correspondence between the elements of the ontology and the relational database schema. The query answering engine is responsible for rewriting the user's SPARQL query into an equivalent SQL query over the database, using the mappings and the ontology[8].

The query answering process in an OBDA system typically involves the following steps:

1. The user formulates a SPARQL query over the ontology.
2. The OBDA engine uses the ontology to rewrite the query, expanding it with the relevant axioms and constraints.
3. The rewritten query is then translated into an SQL query using the mappings.
4. The SQL query is executed by the DBMS over the underlying database.
5. The results are returned to the user in terms of the ontology[33].

This architecture allows for a clear separation between the conceptual view of the data (the ontology) and the physical storage of the data (the databases). This separation provides a high degree of flexibility and scalability, as the ontology and mappings can be updated independently of the underlying data sources[35].

## 2.3 The Data Layer: Relational Databases and SQL

### 2.3.1 Relational Database Concepts

The Data Layer in OBDA systems is primarily composed of relational databases as shown in figure 2.3, which serve as the foundation for storing and managing structured data. Relational databases organize data into tables (also called relations), where each table consists of rows (tuples) and columns (attributes)[38]. This tabular structure provides a systematic way to store and retrieve data while maintaining data integrity and consistency. In the relational model, each table represents an entity type or relationship in the domain. Each row in a table represents a specific instance of the entity or relationship, while each column represents an attribute or property of that entity[39].

**Primary keys** are fundamental to the relational model, serving as unique identifiers for each row in a table. A primary key ensures that no two rows in a table have the same combination of values for the

key attributes. For instance, in a Student table, a student ID might serve as the primary key, guaranteeing that each student has a unique identifier[38].

**Foreign keys** establish relationships between tables by referencing the primary key of another table. This mechanism enables the representation of complex relationships between entities while maintaining referential integrity. For example, an Enrollment table might have foreign keys referencing both the Student and Course tables, establishing the many-to-many relationship between students and courses[39].

The relational model also supports various types of constraints to ensure data quality and consistency. These include entity integrity constraints (ensuring primary keys are unique and not null), referential integrity constraints (ensuring foreign keys reference valid primary keys), and domain constraints (ensuring attribute values conform to specified data types and ranges)[38].

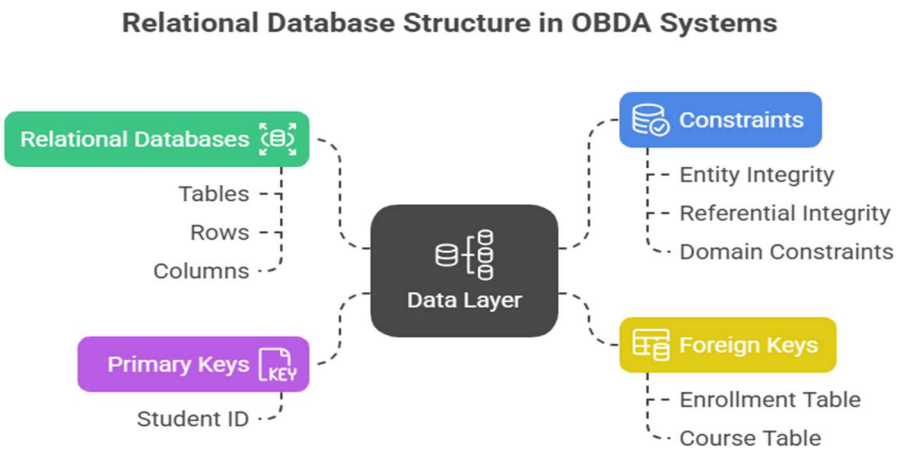


**Figure 2.3 : Data layer in OBDA**

### 2.3.2 SQL as the Query Language

Structured Query Language (SQL) serves as the standard language for interacting with relational databases in OBDA systems. SQL provides a declarative approach to data manipulation, allowing users to specify what data they want without detailing how to retrieve it[40].

The fundamental structure of SQL queries follows the SELECT-FROM-WHERE pattern:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

This basic structure forms the foundation for more complex queries that are essential in OBDA query translation. The SELECT clause specifies which columns to retrieve, the FROM clause identifies the source tables, and the WHERE clause defines filtering conditions[40].

**Joins** are particularly crucial in OBDA systems as they enable the combination of data from multiple tables. Different types of joins serve various purposes:

- **Inner joins** return only rows that have matching values in both tables
- **Left outer joins** return all rows from the left table and matching rows from the right table
- **Right outer joins** return all rows from the right table and matching rows from the left table
- **Full outer joins** return all rows from both tables, with null values where no match exists

For example, to retrieve student names along with their enrolled courses:

```
SELECT s.name, c.title
FROM Students s
INNER JOIN Enrollments e ON s.student_id = e.student_id
INNER JOIN Courses c ON e.course_id = c.course_id;
```

**Filtering conditions** in the WHERE clause are essential for OBDA query translation, as they correspond to the constraints and restrictions specified in SPARQL queries. These conditions can include equality comparisons, range queries, pattern matching, and logical operators (AND, OR, NOT).

### 2.3.3 Role of the Data Layer in OBDA

The Data Layer plays several critical roles in the OBDA architecture, serving as the foundation upon which the semantic abstraction is built as shown in figure 2.4 [33].

- **Data Storage and Management**

The primary role of the Data Layer is to store the actual instance data that will be accessed through the OBDA system. This includes maintaining data integrity, ensuring consistency, and providing efficient storage mechanisms. The relational databases in this layer continue to operate using their existing schemas and data management practices[35].

- **Query Execution and Optimization**

When SPARQL queries are translated into SQL by the OBDA system, the Data Layer's DBMS is responsible for executing these queries efficiently. Modern database systems provide sophisticated query optimizers that can analyze SQL queries and determine the most efficient execution plans. This optimization is crucial for OBDA performance, as complex SPARQL queries can result in intricate SQL queries involving multiple joins and subqueries[41].

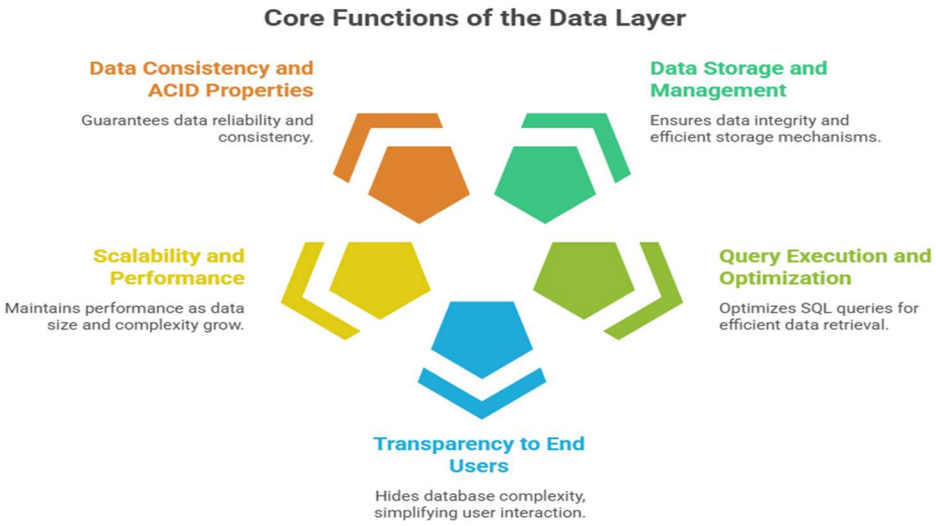


**Figure 2.4 : Core functions of the data layer**

- **Transparency to End Users**

One of the key advantages of OBDA is that the complexity of the underlying database schemas is hidden from end users. Users interact with the system through the ontology's conceptual vocabulary, without needing to understand the intricacies of table structures, foreign key relationships, or SQL syntax. This transparency significantly reduces the barrier to data access for domain experts who may not have technical database expertise[33].

- **Scalability and Performance**

The Data Layer leverages the mature optimization techniques and indexing strategies of relational database systems. This includes B-tree indexes for efficient lookups, query caching mechanisms, and parallel query execution capabilities. These features are essential for maintaining acceptable performance as the size and complexity of the data grow[42].

- **Data Consistency and ACID Properties**

Relational databases provide ACID (Atomicity, Consistency, Isolation, Durability) properties that ensure data reliability and consistency. These properties are particularly important in enterprise environments where data integrity is critical. The OBDA system inherits these guarantees from the underlying database systems[38].

The Data Layer's role in OBDA extends beyond simple data storage to encompass the entire data management lifecycle. This includes backup and recovery procedures, security mechanisms, and data governance policies that are already established for the existing database systems. By building upon these existing infrastructures, OBDA systems can provide semantic data access without requiring significant changes to the underlying data management practices[35].

## 2.4 Ontology Layer

### 2.4.1 Ontology as a Conceptual Schema

The Ontology Layer (figure 2.5) serves as the conceptual schema in OBDA systems, providing a high level, domain-specific view of the data that abstracts away the complexities of the underlying relational database schemas[33]. This layer represents the semantic heart of the OBDA system, defining the vocabulary and conceptual structure through which users interact with the data. An ontology in the OBDA context functions as a formal specification of a shared conceptualization of the domain [43]. It provides a structured representation of the domain knowledge, including the main concepts (classes), their properties (roles), and the relationships between them. This conceptual schema serves multiple purposes: it acts as a communication medium between domain experts and system developers, provides a stable interface that can evolve independently of the underlying data sources, and enables semantic reasoning over the data[36].

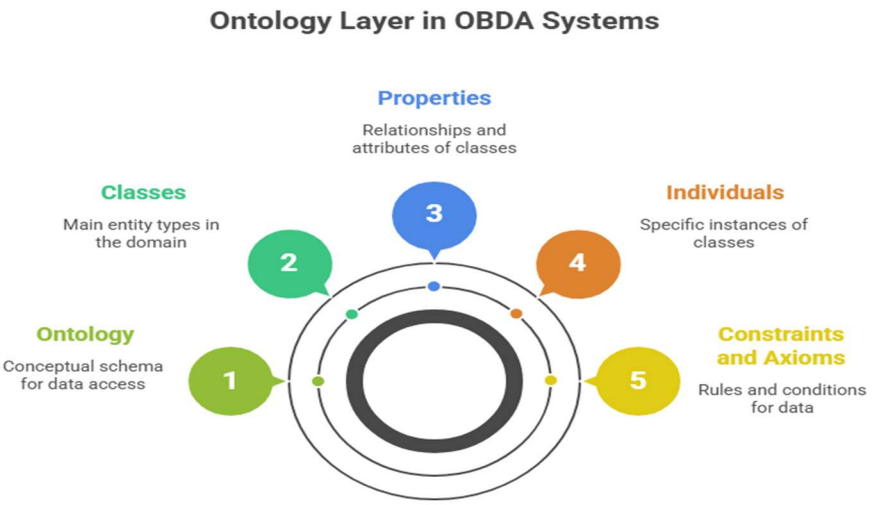


**Figure 2.5 : Ontology layer**

The ontology describes **classes** (concepts) that represent the main entity types in the domain. These classes provide a taxonomic organization of the domain entities, often arranged in hierarchical structures using subsumption relationships[37]. **Properties** (roles) in the ontology represent relationships between classes or attributes of classes. Object properties connect instances of different classes, while data properties connect instances to literal values. Properties can have various characteristics such as functionality, inverse relationships, and domain/range restrictions[43].

**Individuals** represent specific instances of classes in the domain. However, in typical OBDA scenarios, individuals are not explicitly defined in the ontology but are derived from the data in the

underlying databases through the mapping layer[33]. The ontology also captures important domain constraints and axioms that express business rules and semantic relationships. These might include disjointness constraints, cardinality restrictions, and complex class definitions that combine multiple conditions[37].

### 2.4.2 Ontology Language for OBDA

The choice of ontology language is crucial for OBDA systems, as it must balance expressiveness with computational tractability. **OWL 2 QL** (Query Language) has emerged as the preferred ontology profile for OBDA applications due to its specific design for efficient query answering[37]. OWL 2 QL is based on the **DL-Lite** family of Description Logics, which provides a carefully designed subset of OWL 2 that ensures **first-order rewritability**[44]. This property is fundamental to OBDA systems because it guarantees that any SPARQL query over the ontology can be rewritten into a first-order logic query (and subsequently into SQL) that can be executed directly over the underlying database without requiring materialization of the entire knowledge base[45].

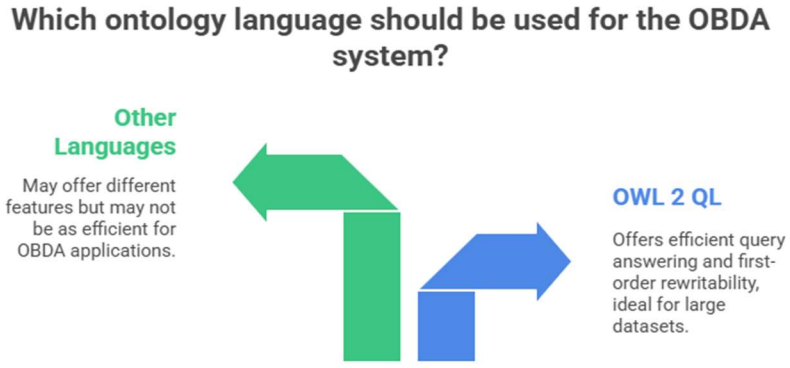


**Figure 2.6 : Ontology languages**

The key features of OWL 2 QL include:

- **Basic Class Expressions:** OWL 2 QL supports atomic classes and existential restrictions on the right-hand side of axioms. This allows for the definition of classes based on the existence of certain relationships, such as "Student ⊑ ∃enrolledIn.Course" (every student is enrolled in some course)[37].
- **Property Hierarchies:** The language supports property hierarchies through sub-property relationships, enabling the organization of properties in taxonomic structures. For example, "teaches" might be a sub-property of "associatedWith"[37].

- **Property Characteristics:** OWL 2 QL supports functional properties, inverse properties, and symmetric properties. These characteristics enable the expression of important domain constraints while maintaining computational tractability[37].
- **Negative Assertions:** The language allows for disjointness assertions between classes and properties, enabling the expression of negative constraints such as "Student and Professor are disjoint"[37].

The computational advantages of OWL 2 QL are significant for OBDA applications. Query answering in OWL 2 QL has a data complexity of $AC^0$ (constant time with respect to the size of the data), which means that the query answering performance scales well with large datasets[45]. This is achieved through the first-order rewritability property, which allows the ontological reasoning to be "compiled away" into the SQL query, leveraging the optimization capabilities of the underlying database systems[44].

### 2.4.3 Example Ontology

To illustrate the concepts discussed above, consider a simplified university ontology that might be used in an OBDA system for accessing academic data. This example demonstrates how domain knowledge can be captured using OWL 2 QL constructs.

**Classes (Concepts)**:

```
:Person rdf:type owl:Class .
:Student rdf:type owl:Class ;
        rdfs:subClassOf :Person .
:Professor rdf:type owl:Class ;
        rdfs:subClassOf :Person .
:Course rdf:type owl:Class .
:Department rdf:type owl:Class .
:Undergraduate rdf:type owl:Class ;
        rdfs:subClassOf :Student .
:Graduate rdf:type owl:Class ;
        rdfs:subClassOf :Student .
# Disjointness constraints
:Student owl:disjointWith :Professor .
:Undergraduate owl:disjointWith :Graduate .

# Domain-specific constraints
:Student rdfs:subClassOf [
         rdf:type owl:Restriction ;
         owl:onProperty :hasStudentID ;
        owl:someValuesFrom xsd:string
] .

:Graduate rdfs:subClassOf [
```

```
        rdf:type owl:Restriction ;
        owl:onProperty :advisedBy ;
        owl:someValuesFrom :Professor
]
```

**Object Properties:**

```
:enrolledIn rdf:type owl:ObjectProperty ;
            rdfs:domain :Student ;
            rdfs:range :Course .

:teaches rdf:type owl:ObjectProperty ;
         rdfs:domain :Professor ;
         rdfs:range :Course .

:belongsTo rdf:type owl:ObjectProperty ;
         rdfs:domain :Person ;
         rdfs:range :Department .

:advisedBy rdf:type owl:ObjectProperty ;
         rdfs:domain :Graduate ;
         rdfs:range :Professor ;
         rdf:type owl:FunctionalProperty .
```

**Data Properties:**

```
:hasStudentID rdf:type owl:DatatypeProperty ;
              rdfs:domain :Student ;
              rdfs:range xsd:string ;
              rdf:type owl:FunctionalProperty .

:hasName rdf:type owl:DatatypeProperty ;
             rdfs:domain :Person ;
             rdfs:range xsd:string .

:hasCredits rdf:type owl:DatatypeProperty ;
             rdfs:domain :Course ;
             rdfs:range xsd:integer .

:hasGPA rdf:type owl:DatatypeProperty ;
             rdfs:domain :Student ;
rdfs:range xsd:decimal
```

- **Axioms and Constraints:**

This ontology (figure 2.7) captures essential aspects of a university domain while remaining within the expressiveness limits of OWL 2 QL. The hierarchical organization of classes (Person → Student

→ Undergraduate/Graduate) provides a taxonomic structure that can be leveraged during query answering. The property definitions establish the relationships between different entities, while the constraints ensure data consistency and capture important business rules[37].

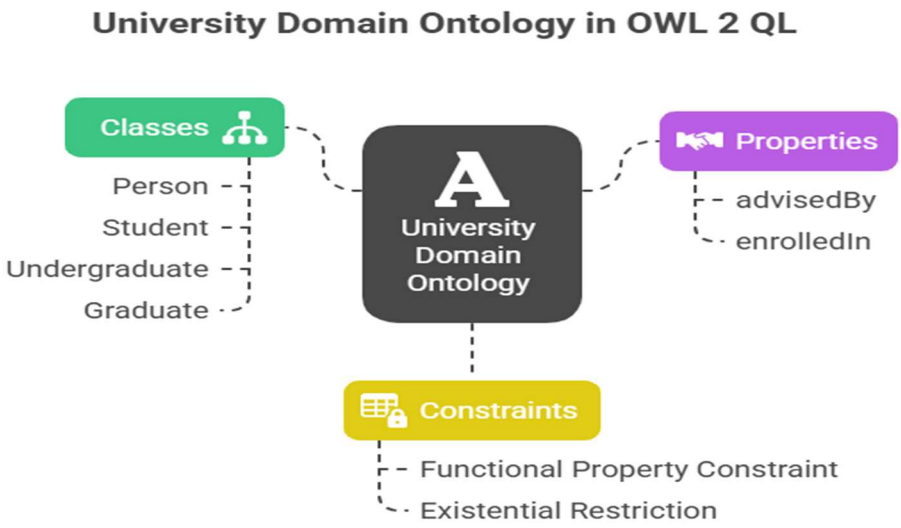


**Figure 2.7 : Example of an ontology**

The functional property constraint on :advisedBy ensures that each graduate student has at most one advisor, while the existential restriction on the Graduate class ensures that every graduate student must have an advisor. These constraints demonstrate how domain knowledge can be formally captured in the ontology and used during query processing to infer additional information[43].

When this ontology is used in an OBDA system, users can formulate queries using these high-level concepts without needing to understand the underlying database schema. For example, a user might ask "Find all graduate students in the Computer Science department" using SPARQL, and the OBDA system would translate this into appropriate SQL queries over the underlying tables storing student, enrollment, and department information[33].

## 2.5 Mapping Layer

### 2.5.1 Concept and Purpose

The primary purpose of mappings is to establish a declarative specification of how data stored in relational format should be interpreted in terms of the ontology. This interpretation process involves several key aspects: identifying which database tables correspond to which ontology classes, determining how table columns map to ontology properties, and specifying how to construct URIs for ontology individuals from database primary keys[46].

Mappings in OBDA systems are typically expressed as a set of declarative rules, each of which specifies how to generate RDF triples from relational data. These rules follow a general pattern where a SQL query defines the source data, and a template specifies how to construct the corresponding RDF triples. This approach allows for complex transformations and data integration scenarios while maintaining the declarative nature that is essential for query optimization[47].

### 2.5.2 Mapping Languages

- **R2RML: The Foundation of OBDA Mapping Languages**

R2RML (RDB to RDF Mapping Language) is a W3C Recommendation that provides a standardized vocabulary for expressing customized mappings from relational databases to RDF datasets[48]. Developed as part of the RDB2RDF Working Group, R2RML enables the transformation of existing relational data into the RDF data model while preserving the semantic structure and target vocabulary chosen by the mapping author[36]. The language uses RDF graphs written in Turtle syntax to define mappings, where each mapping consists of triples maps that specify how database tables or SQL query results should be converted into RDF triples[33]. This approach allows organizations to provide semantic access to their relational data without requiring physical data transformation or duplication, supporting the principles of ontology-based data access (OBDA)[49].

Consider a simple educational database with two tables: TEACHERS (containing columns TEACHER_ID , NAME , DEPARTMENT ) and STUDENTS (containing STUDENT_ID , NAME , TEACHER_ID ). An R2RML mapping can transform this relational structure into semantic RDF data as follows:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://university.edu/ontology#>.
<#TeacherMapping>
        rr:logicalTable [ rr:tableName "TEACHERS" ];
        rr:subjectMap [
                rr:template "http://university.edu/teacher/{TEACHER_ID}";
                rr:class ex:Teacher;
        ];
        rr:predicateObjectMap [
                rr:predicate ex:hasName;
                rr:objectMap [ rr:column "NAME" ];
        ];
        rr:predicateObjectMap [
                rr:predicate ex:belongsToDepartment;
                rr:objectMap [ rr:column "DEPARTMENT" ];

        ].
<#StudentMapping>
        rr:logicalTable [ rr:tableName "STUDENTS" ];
        rr:subjectMap [
                rr:template "http://university.edu/student/{STUDENT_ID}";
```

```
        rr:class ex:Student;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:hasName;
        rr:objectMap [ rr:column "NAME" ];
    ];
    rr:predicateObjectMap [
    rr:predicate ex:hasTeacher;
        rr:objectMap [
        rr:template "http://university.edu/teacher/{TEACHER_ID}";
    ];
].
```

This mapping creates semantic relationships where each teacher and student becomes a distinct resource with appropriate properties, and the foreign key relationship between students and teachers is preserved as an RDF property link, enabling sophisticated SPARQL queries over the educational domain[50].

- **RML (RDF Mapping Language)**

RML (RDF Mapping Language) is a generic mapping language that extends the W3Crecommended R2RML to support heterogeneous data sources beyond relational databases. Developed to address the limitations of R2RML's relational-only focus, RML maintains the same syntax and semantics as R2RML while introducing the concept of logical sources that can reference various data formats including JSON, XML, CSV, and other structured data sources. RML follows exactly the same syntax as R2RML but replaces the logical table concept with logical sources that specify different reference formulations such as JSONPath, XPath, or CSV column references. This design ensures backward compatibility with existing R2RML mappings while enabling the integration of diverse data formats within modern data ecosystems, supporting the growing need for semantic data integration across heterogeneous sources in ontology-based data access scenarios[51].

- **YARRRML (Yet Another RDF Mapping Language)**

YARRRML (Yet Another RDF Mapping Language) is a human-readable, YAML-based representation for declarative Linked Data generation rules that serves as a user-friendly alternative to writing complex RML mappings. Developed to address the steep learning curve associated with RDF/Turtle syntax used in R2RML and RML, YARRRML provides an intuitive YAML-based syntax that can be automatically translated into RML mappings while maintaining the full expressiveness of the underlying mapping language. YARRRML is designed as a subset of YAML, making it accessible to domain experts and data engineers who may not be familiar with RDF technologies but need to create semantic mappings for heterogeneous data sources. The language significantly reduces the barrier to entry for mapping creation by providing a more natural syntax for specifying data transformations,

while still supporting advanced features such as nested data structures, conditional mappings, and function-based transformations [52].

## 2.6 Query Answering Process in OBDA

### 2.6.1 Workflow

The query answering process in OBDA systems follows a sophisticated workflow that seamlessly transforms high-level semantic queries into efficient database operations[33]. This process as shown in figure 2.8, represents the core functionality of OBDA systems and demonstrates how the three-layer architecture works together to provide transparent access to relational data through semantic interfaces.

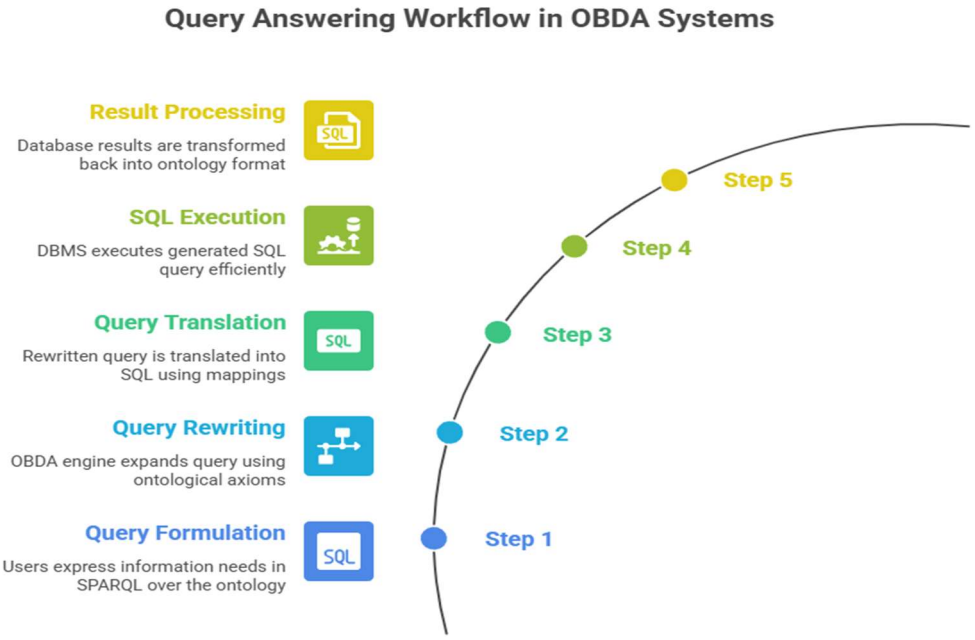


**Figure 2.8 : Query answering workflow**

The typical workflow consists of the following steps:

- **Step 1: Query Formulation**

Users formulate queries using SPARQL over the ontology vocabulary. These queries express information needs in terms of the domain concepts and relationships defined in the ontology, without requiring knowledge of the underlying database schema[53].

- **Step 2: Query Rewriting**

The OBDA engine performs query rewriting using the ontological axioms and constraints. This step expands the original query by incorporating relevant information from the ontology, such as class

hierarchies, property relationships, and domain constraints. The rewriting process ensures that all implicit information derivable from the ontology is made explicit in the query[44].

- **Step 3: Query Translation**

The rewritten query is then translated into SQL using the mapping specifications. This translation process involves identifying the relevant database tables and columns that correspond to the ontology elements in the query, constructing appropriate JOIN operations to connect related data, and applying filtering conditions based on the query constraints[8].

- **Step 4: SQL Execution**

The generated SQL query is executed by the underlying database management system(DBMS). The DBMS applies its own optimization strategies, including index usage, join ordering, and query plan selection, to execute the query efficiently over the stored data[54].

- **Step 5: Result Processing**

The results returned by the database are processed and transformed back into the ontology format. This involves constructing RDF triples or SPARQL result bindings that conform to the ontology vocabulary and presenting them to the user in the expected semantic format[33].

This workflow ensures that users can interact with complex relational data using intuitive semantic queries while leveraging the performance and scalability of mature database systems. The process is designed to be transparent to users, who see only the semantic interface and remain unaware of the underlying query translation and execution complexities[35].

### 2.6.2 Query Rewriting and Optimization

Query rewriting as shown in figure 2.9, is a fundamental component of OBDA systems that enables the incorporation of ontological knowledge into the query answering process. The rewriting step is particularly crucial for ontologies expressed in DL-Lite and OWL 2 QL, which support first-order rewritability[44].

The concept of first-order rewritability ensures that any query over an ontology can be rewritten into a first-order logic query (and subsequently into SQL) that produces the same results when evaluated over the underlying database. This property is essential for OBDA systems because it allows all ontological reasoning to be "compiled away" into the SQL query, eliminating the need for materialization or complex reasoning procedures during query execution[45]. Class hierarchies in the ontology are unfolded to include all subclass relationships. For example, if the query asks for all Students, and the ontology defines Undergraduate and Graduate as subclasses of Student, the rewritten query will include conditions for all three classes[44].

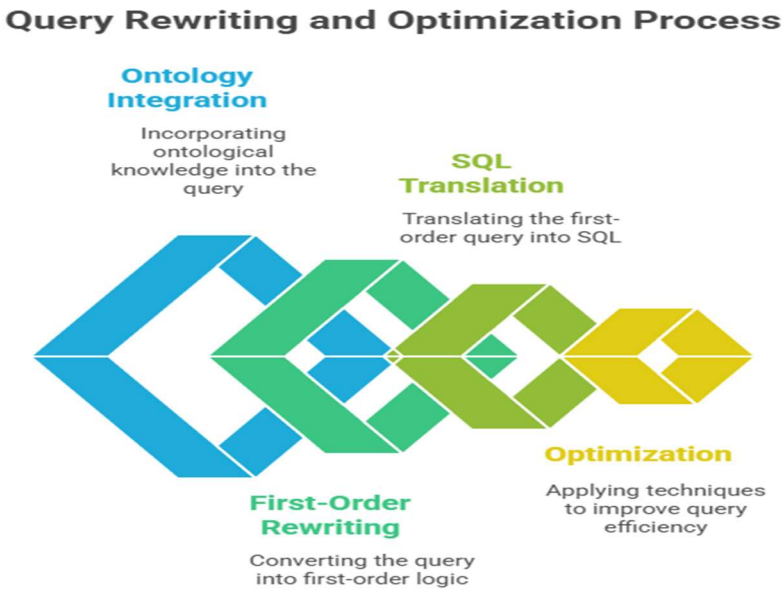


**Figure 2.9 : Query rewriting and optimization**

Property hierarchies and property chains are expanded to include all relevant relationships. If a query involves a property that has sub-properties or is part of a property chain, the rewriting process includes all applicable property relationships[37]. Existential restrictions in the ontology are handled by introducing appropriate JOIN operations in the SQL query. For example, if the ontology states that every Graduate student must have an advisor, queries involving Graduate students will include JOINs to ensure this relationship exists[45].

Several optimization techniques are employed to improve the efficiency of the query rewriting and translation process. Redundant conditions and subqueries are eliminated from the rewritten query to reduce complexity and improve execution performance[55]. The ontological constraints are used to optimize the query by eliminating impossible conditions, simplifying complex expressions, and identifying more efficient query plans[56]. The query translation process considers the structure of the mappings to generate more efficient SQL queries. This includes optimizing JOIN operations, selecting appropriate indexes, and minimizing data transfer[57]. For complex queries, incremental rewriting techniques are used to build the final query step by step, allowing for better optimization at each stage[58].

## 2.7 Conclusion

This chapter has provided a comprehensive examination of Ontology-Based Data Access (OBDA), a semantic paradigm that has emerged as a powerful solution for accessing and integrating heterogeneous relational data sources. Through the detailed exploration of the three-layer OBDA

architecture, we have demonstrated how this approach successfully bridges the gap between the conceptual world of domain ontologies and the physical world of relational databases. The query answering process in OBDA systems demonstrates the sophisticated integration of semantic technologies with traditional database systems. The workflow from SPARQL query formulation through ontological reasoning, query rewriting, SQL translation, and result processing showcases how complex semantic queries can be efficiently executed over large relational datasets.

# Chapter 3 Our Contribution: Proposed Approach

## 3.1 Introduction

The optimization of SQL queries remains a foundational challenge in database management, directly impacting application performance and resource utilization. While traditional optimizers have become highly sophisticated, they often lack an understanding of the semantic context underlying the data, which can lead to inefficient or logically flawed query executions. This chapter addresses this gap by exploring a hybrid approach that synergizes the formal reasoning capabilities of ontologies with the predictive power of machine learning. By encoding domain-specific knowledge and business rules into an ontology, we create a semantic layer that enables a more intelligent query rewriting process, ensuring that queries are not only fast but also semantically sound.

## 3.2 Supervised Machine Learning and Classification Algorithms

### 3.2.1 Supervised Machine Learning

Supervised machine learning as shown in figure 3.1, is a paradigm of artificial intelligence where algorithms learn to map input variables to output variables based on labeled training data[59]. In supervised learning, the algorithm is provided with a dataset consisting of input-output pairs, where the correct output (label or target variable) is known for each input. The primary objective is to learn a function that can accurately predict the output for new, previously unseen inputs.

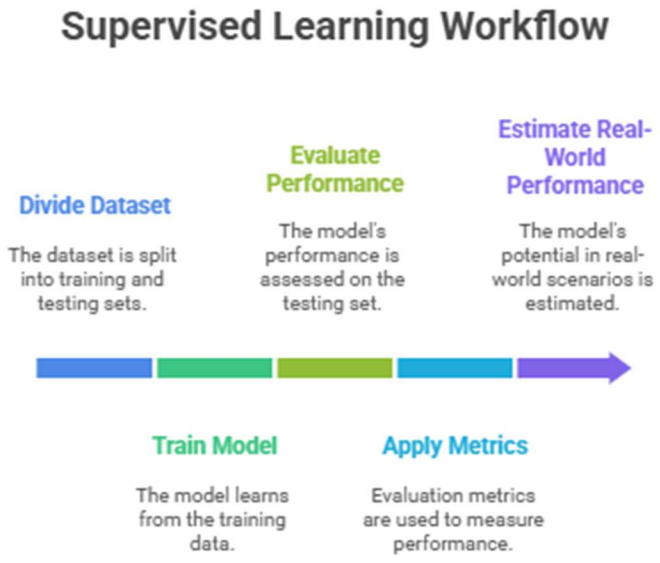


**Figure 3.1 : Supervised learning workflow.**

Supervised learning problems are typically categorized into two main types: classification and regression. Classification problems involve predicting discrete categorical labels or classes, such as determining whether an email is spam or not spam or classifying images into different categories. Regression problems, on the other hand, involve predicting continuous numerical values, such as predicting house prices or stock market values. In the context of our research, we focus on classification, specifically binary classification, where the goal is to predict one of two possible classes: simple or complex queries.

The supervised learning process typically follows a structured workflow. First, the dataset is divided into training and testing sets, with the training set used to train the model and the testing set used to evaluate its performance on unseen data. During the training phase, the algorithm analyzes the input-output relationships in the training data to learn patterns and build a predictive model. The model's performance is then assessed using various evaluation metrics applied to the testing set, providing an estimate of how well the model will perform on new, real-world data.

### 3.2.2 Random Forest Classifier Algorithm

The Random Forest Classifier as illustrated in Figure 3.2, introduced by Leo Breiman[59], is an ensemble learning method that combines multiple decision trees to create a more robust and accurate classifier. The algorithm belongs to the family of ensemble methods, which leverage the principle that combining multiple weak learners can produce a strong learner with superior predictive performance. Random Forest addresses several limitations of individual decision trees, including overfitting, high variance, and sensitivity to small changes in the training data.

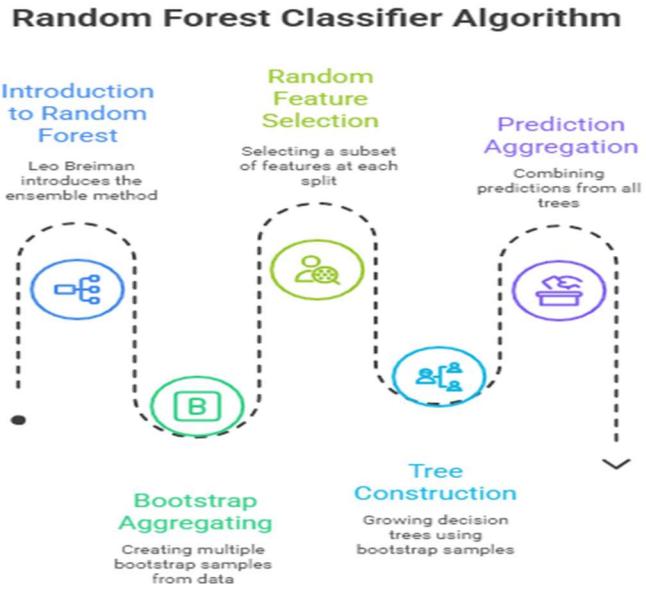


**Figure 3.2 : Random Forest Classifier Algorithm**

The Random Forest algorithm operates on two key principles: bootstrap aggregating (bagging) and random feature selection. Bootstrap aggregating involves creating multiple bootstrap samples from the original training dataset, where each bootstrap sample is created by randomly sampling with replacement from the original data. This process ensures that each tree in the forest is trained on a slightly different subset of the data, introducing diversity among the individual trees. Random feature selection further enhances this diversity by randomly selecting a subset of features at each split in each decision tree, rather than considering all available features.

The construction of a Random Forest involves several steps. First, for each tree in the forest, a bootstrap sample is drawn from the original training dataset. Second, for each bootstrap sample, a decision tree is grown using a random subset of features at each split. The number of features considered at each split is typically set to the square root of the total number of features for classification problems. Third, each tree is grown to its maximum depth without pruning, allowing individual trees to have high variance but low bias. Finally, predictions are made by aggregating the predictions of all trees in the forest, typically using majority voting for classification problems.

### 3.2.3 Random Forest Parameters and Configuration

The Random Forest algorithm has several important parameters that significantly influence its performance and behavior. Understanding these parameters is crucial for effective model tuning and optimization.

- **Number of Estimators (n_estimators):** This parameter determines the number of decision trees in the forest. Generally, increasing the number of trees improves performance up to a certain point, after which the gains become marginal while computational cost continues to increase. In this research, we use 100 trees, which provides a good balance between performance and computational efficiency.
- **Maximum Features (max_features):** This parameter controls the number of features considered when looking for the best split at each node. Common choices include the square root of the total number of features ($\sqrt{p}$), the logarithm of the total number of features ($\log_2 p$), or a fixed number. The default for classification is $\sqrt{p}$, which we employ in our implementation.
- **Maximum Depth (max_depth):** This parameter limits the maximum depth of each tree. While Random Forest typically allows trees to grow to their maximum depth, setting a limit can help control overfitting and reduce computational complexity. In our implementation, we allow unlimited depth to maximize the individual tree performance.
- **Class Weight (class_weight):** This parameter addresses class imbalance by assigning different weights to different classes. Setting this to 'balanced' automatically adjusts weights inversely

proportional to class frequencies, which is particularly useful when dealing with imbalanced datasets. We use this setting to ensure fair treatment of both simple and complex query classes.

- **Random State (random_state):** This parameter controls the randomness of the algorithm, ensuring reproducibility of results. Setting a fixed random state allows for consistent results across multiple runs, which is essential for scientific reproducibility.

### 3.2.4 Classification Metrics and Evaluation

The evaluation of classification models requires a comprehensive set of metrics that capture different aspects of model performance. This section presents the key classification metrics used in this research, along with their mathematical formulations and interpretations.

- **Confusion Matrix**

The confusion matrix is a fundamental tool for evaluating classification performance, providing a detailed breakdown of correct and incorrect predictions for each class. For a binary classification problem, the confusion matrix is a 2×2 table with the structure illustrated in table 3.1.

**Table 3.1 : Confusion matrix**

| | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | True Negative (TN) | False Positive (FP) |
| **Actual Positive** | False Negative (FN) | True Positive (TP) |

- **Accuracy**

Accuracy measures the overall correctness of the classifier by calculating the proportion of correct predictions among all predictions:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

While accuracy is intuitive and widely used, it can be misleading in cases of class imbalance, where a high accuracy might be achieved by simply predicting the majority class.

- **Precision**

Precision measures the proportion of correctly identified positive cases among all cases predicted as positive. It answers the question: "Of all the instances predicted as positive, how many were actually positive?"

$$Precision = TP / (TP + FP)$$

High precision indicates a low false positive rate, meaning the model rarely incorrectly classifies negative instances as positive

- **Recall (Sensitivity)**

Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive cases that were correctly identified. It answers the question: "Of all the actual positive instances, how many were correctly predicted?"

$$Recall = TP / (TP + FN)$$

High recall indicates a low false negative rate, meaning the model successfully identifies most positive instances.

- **F1_Score**

The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both measures. It is particularly useful when dealing with imbalanced datasets or when both precision and recall are important:

$$F1\_Score = 2 \times (Precision \times Recall) / (Precision + Recall)$$

The harmonic mean ensures that the F1_score is low when either precision or recall is low, making it a conservative measure that requires both metrics to be reasonably high.

- **Specificity**

Specificity measures the proportion of actual negative cases that were correctly identified:

$$Specificity = TN / (TN + FP)$$

Specificity is particularly important in scenarios where correctly identifying negative cases is crucial, such as medical diagnosis or fraud detection. These metrics provide complementary perspectives on classifier performance, and their combined use enables a comprehensive evaluation of the model's effectiveness across different aspects of the classification task. In this research, we employ all these metrics to thoroughly assess the performance of our Random Forest Classifier for query complexity prediction.

- **Area Under the ROC Curve (AUC-ROC)**

The Receiver Operating Characteristic (ROC) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier by plotting the True Positive Rate (Recall) against the False Positive Rate (1 - Specificity) at various threshold settings. The Area Under the Curve (AUC) provides a single scalar value that summarizes the ROC curve:

$$False\ Positive\ Rate = FP / (FP + TN)$$

$$True\ Positive\ Rate = TP / (TP + FN)$$

The AUC-ROC score ranges from 0 to 1, where:

- AUC = 0.5 indicates random performance (no discriminative ability)
- AUC = 1.0 indicates perfect classification

- AUC > 0.7 is generally considered acceptable
- AUC > 0.8 is considered good
- AUC > 0.9 is considered excellent

## 3.3 Methodology

This section presents the detailed methodology of the hybrid query optimization framework proposed in our research[60]. The framework is designed to seamlessly integrate ontology-based semantic reasoning with supervised machine learning to automate the process of SQL query optimization. The methodology is divided into four main components: the system architecture, the ontology design and mapping process, the machine learning model development, and the query rewriting algorithm. Each of these components is described in detail in the following subsections.

### 3.3.1 System Architecture

The proposed framework is designed as a two-part pipeline[60], as depicted in Figure 3.3. The first part is the machine learning component, which is responsible for classifying incoming SQL queries as simple or complex. The second part is the ontology processing component, which is responsible for rewriting complex queries using semantic constraints derived from a domain ontology.

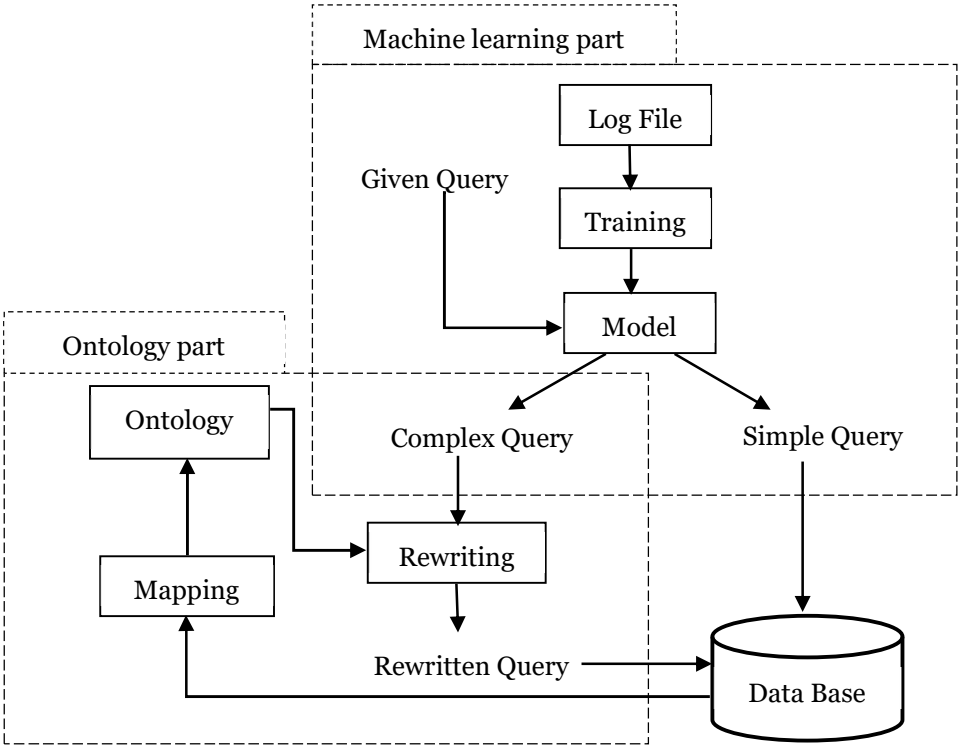


**Figure 3.3 : System Architecture**

The overall workflow of the framework is as follows:

1. **Query Interception:** All incoming SQL queries are intercepted by the framework before they are executed by the database management system (DBMS).
2. **Feature Extraction:** For each query, a set of features is extracted, including the number of joins, the presence of a WHERE clause, and other structural characteristics.
3. **Complexity Prediction:** The extracted features are fed into a pre-trained machine learning model, which predicts whether the query is simple or complex.
4. **Query Routing:** If the query is classified as simple, it is passed directly to the DBMS for execution. If the query is classified as complex, it is passed to the ontology processing component for rewriting.
5. **Query Rewriting:** The complex query is rewritten using semantic constraints derived from a domain ontology. The rewriting process involves adding new conditions to the WHERE clause of the query to prune the search space and ensure semantic correctness.
6. **Query Execution:** The rewritten query is then passed to the DBMS for execution

This architecture allows for a dynamic and adaptive approach to query optimization, where only the queries that are likely to benefit from rewriting are subjected to the overhead of the ontology processing component. This ensures that the framework does not introduce unnecessary latency for simple queries.

### 3.3.2 Ontology Design and Mapping

The ontology is a crucial component of the proposed framework, as it provides the semantic layer that enables intelligent query rewriting. The ontology is designed to capture the domain knowledge and business rules of the application. The process of creating the ontology involves two main steps: mapping the relational database schema to an ontology and enriching the ontology with semantic axioms.

- **Database to Ontology Mapping**

The first step in creating the ontology is to map the relational database schema to an OWL ontology. This process, often referred to as ontology learning or ontology bootstrapping, can be performed manually, semi-automatically, or automatically. In this research, we adopt a semi-automatic approach, where the initial ontology is generated automatically from the database schema and then refined manually by a domain expert. The ontology can be derived from a relational database schema through a dedicated mapping process as given in the procedure bellow:

```
Procedure MapDatabaseToOntology(DatabaseSchema db, Ontology onto):
  For each table T in db.tables:
    Create an OntologyClass OC in onto corresponding to table T
    onto.addClass(OC)
    For each column C in T.columns:
```

```
        Create an OntologyProperty OP corresponding to column C
        onto.addProperty(OP)
        onto.linkPropertyToClass(OP, OC)
    For each foreign key FK in db.foreignKeys:
      SourceTable = FK.sourceTable
      TargetTable = FK.targetTable
      SourceClass = onto.getClassCorrespondingTo(SourceTable)
      TargetClass = onto.getClassCorrespondingTo(TargetTable)
      Create ObjectProperty OProp representing FK relationship
      onto.addObjectProperty(OProp)
      onto.linkClassesThroughProperty(SourceClass, TargetClass, OProp)
    Return onto
End Procedure
```

The mapping process involves the following steps:

1. **Table to Class Mapping:** Each table in the database is mapped to a class in the ontology.
2. **Column to Property Mapping:** Each column in a table is mapped to a data property of the corresponding class.
3. **Foreign Key to Object Property Mapping:** Each foreign key relationship between two tables is mapped to an object property between the corresponding classes.

This mapping process creates a basic ontology that reflects the structure of the underlying database. However, this ontology does not yet contain the rich semantic knowledge that is needed for intelligent query rewriting. This is addressed in the next step.

➢ **Semantic Enrichment with TBox Axioms**

Once the basic ontology has been created, it is enriched with a set of TBox axioms that capture the business rules and constraints of the application domain. These axioms are typically defined by a domain expert and are expressed in Description Logic. The axioms can be of various types, including:

- **Subsumption Axioms:** These axioms define class hierarchies, such as :

$$RecentOrder \sqsubseteq Order$$

- **Equivalence Axioms:** These axioms define class equivalences, such as :

$$AvailableProduct \equiv Product \sqcap \exists hasStock.(> 0)$$

- **Existential Restrictions:** These axioms specify that a class must have at least one relationship of a certain type, such as :

$$Order \sqsubseteq \exists hasOrderDate.xsd{:}\,dateTime$$

- **Value Restrictions:** These axioms specify constraints on the values of properties, such as

$$RecentOrder \sqsubseteq Order \sqcap \exists hasOrderDate.(\geq "2022-08-01")$$

These axioms provide the semantic knowledge that is used to rewrite complex queries. The process of defining these axioms is a critical step in the methodology, as the quality of the axioms directly impacts the effectiveness of the query rewriting process.

### 3.3.3 Machine Learning Model Development

The machine learning model is responsible for classifying incoming SQL queries as simple or complex. The development of the model involves three main steps: feature engineering, model training, and model evaluation.

➢ **Feature Engineering**

The first step in developing the machine learning model is to identify a set of features that can be used to predict the complexity of a query. In this research[60], we use a combination of structural and performance-related features, including:

- **Number of Joins:** The number of JOIN operations in the query. This is a strong indicator of query complexity, as queries with more joins are typically more expensive to execute.
- **Where Flag:** A binary flag that indicates whether the query contains a "Where" clause. Queries with WHERE clauses are often more complex than those without, as they involve filtering and predicate evaluation.
- **Execution Time:** The execution time of the query. This is a direct measure of query performance and is a key feature for predicting complexity.
- **Complexity Score:** A composite score that is calculated based on the number of joins and the presence of a WHERE clause. The formula for the complexity score is given as :

$$cs = 1.0 + 1.5 * j + 1.0 * w$$

where j is the number of joins and w is the WHERE flag.

These features are extracted from a log file of SQL queries that have been executed against the database. The log file contains the SQL query text, the execution time, and other relevant information.

➢ **Model Training**

Once the features have been extracted, they are used to train a machine learning model. In our work, we use the Random Forest Classifier, which is a powerful and versatile algorithm that is well-suited for this task. The model is trained on a labeled dataset of SQL queries, where each query is labeled as either simple or complex. The labels are generated based on a thresholding approach, where queries with a complexity score above a certain threshold are labeled as complex and those below the threshold are labeled as simple. The dataset is split into a training set and a testing set, with 80% of the data used for training and 20% for testing.

➢ **Model Evaluation**

After the model has been trained, it is evaluated on the testing set to assess its performance. The evaluation is performed using a comprehensive set of classification metrics explained in section 3.2, including:

- **Accuracy:** The overall correctness of the model.
- **Precision:** The proportion of correctly identified complex queries out of all queries classified as complex.
- **Recall:** The proportion of actual complex queries that were successfully identified.
- **F1-Score:** The harmonic mean of precision and recall.
- **AUC Score:** The area under the ROC curve, which measures the model's ability to distinguish between the two classes.
- **Confusion Matrix:** A table that summarizes the performance of the model by showing the number of true positives, true negatives, false positives, and false negatives.

These metrics provide a comprehensive assessment of the model's performance and are used to validate its effectiveness for the task of query complexity prediction.

### 3.3.4 Query Rewriting Algorithm

The query rewriting algorithm is the core of the ontology processing component. It is responsible for rewriting complex queries using the semantic constraints defined in the ontology. The algorithm takes as input a complex SQL query and the domain ontology and returns a rewritten SQL query that is semantically equivalent but more efficient to execute. The rewriting algorithm can be given in pseudo code as fellows :

```
Procedure RewriteQueryUsingOntology(SQL_Query query, Ontology onto):
  parsedQuery ← ParseSQL(query)
  involvedTables ← GetTables(parsedQuery)
  axiomsToApply ← emptyList()
  For each table T in involvedTables:
    ontologyClass ← onto.getClassCorrespondingTo(T)
    classAxioms ← onto.getAxiomsForClass(ontologyClass)
    For each axiom A in classAxioms:
      If axiom A is not already present in parsedQuery.conditions:
        axiomsToApply.add(A)
      End If
    End For
  End For
  rewrittenQuery ← parsedQuery
  For each axiom in axiomsToApply:
    rewrittenQuery ← AddConditionToSQL(rewrittenQuery, axiom)
  End For
  Return GenerateSQL(rewrittenQuery)
```

End Procedure

The algorithm works as follows:

- **Parse the SQL Query:** The input SQL query is parsed to identify the tables and columns involved in the query.
- **Identify Relevant Axioms:** For each table in the query, the algorithm identifies the corresponding class in the ontology and retrieves the set of axioms that are defined for that class.
- **Filter Redundant Axioms:** The algorithm then filters out any axioms that are already present in the query's WHERE clause to avoid redundancy.
- **Apply Axioms to the Query:** The remaining axioms are then applied to the query. For example, if the ontology contains an axiom that specifies that:

  $$RecentOrder \sqsubseteq Order \sqcap \exists hasOrderDate.(\geq \text{"}2022-08-01\text{"}),$$

  and the query is :

  $$SELECT * FROM\ Orders,$$

  the algorithm will rewrite the query as:

  $$SELECT * FROM\ Orders\ WHERE\ OrderDate >= '2022-08-01'.$$
- **Generate the Rewritten Query:** Finally, the algorithm generates the rewritten SQL query, which is then passed to the DBMS for execution.

This algorithm provides a systematic and automated way to leverage the semantic knowledge in the ontology to optimize complex SQL queries. The use of an ontology ensures that the rewriting process is based on a formal and explicit representation of the domain knowledge, which leads to more accurate and effective optimizations.

## 3.4 Experimental Design and Implementation

This section details the experimental design and implementation used to evaluate the proposed hybrid query optimization framework. The evaluation was conducted using an e-commerce case study scenario, a dataset of 3,000 SQL queries, and a comprehensive set of performance metrics. The goal of the experiment was to assess the effectiveness of the framework in terms of both classification accuracy and query performance improvement.

### 3.4.1 E-commerce Case Study scenario

The experiment was conducted in the context of an e-commerce application, a domain where data is abundant and query performance is critical. The database schema for the case study is based on a typical e-commerce model, consisting of four main entities: Customer, Order, OrderDetail, and Product. The relationships between these entities are as follows:

- A Customer can have multiple Orders.
- An Order can have multiple OrderDetails.
- An OrderDetail is associated with a single Product.
- Each product may appear in multiple order details.

This schema, depicted as an entity-association model in Figure 3.4, provides a realistic and sufficiently complex basis for evaluating the proposed framework. The database was populated with synthetic data to simulate a real-world e-commerce environment.

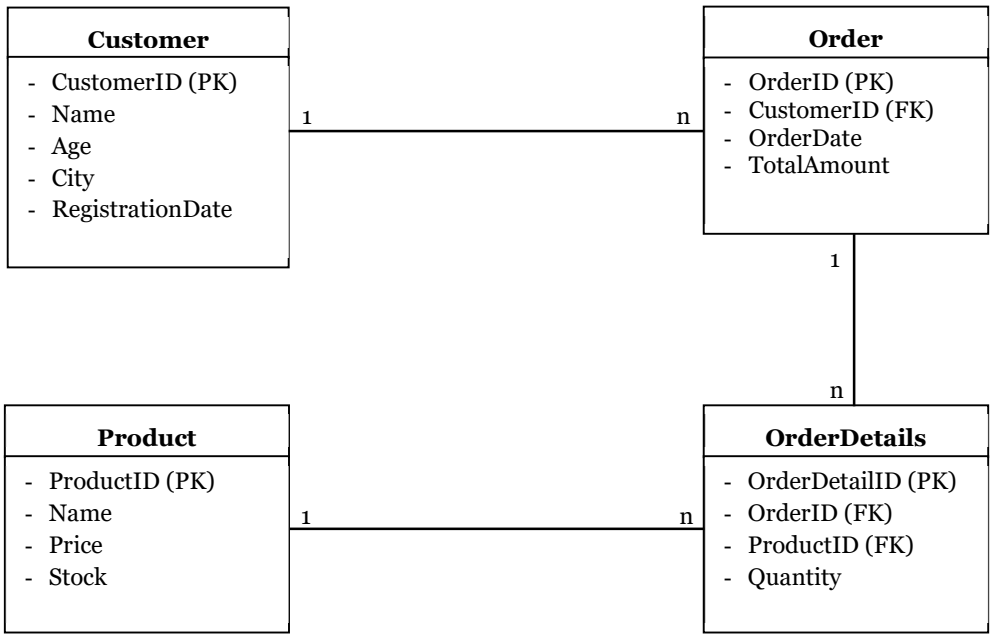


**Figure 3.4 : E-commerce scenario entity association model**

Table 3.2 presents the TBox ontology rules derived from the ontology obtained through the mapping process. Rules R1 to R4 represent a direct translation from the entity-association model shown in Figure 14. The remaining rules, from R5 onward, are either inferred based on the initial rules or manually added. These additional axioms serve as semantic enrichments and can be used for query optimization. Rules R5 to R16 are included as illustrative examples; however, more rules can be added as needed. This extensibility highlights one of the key features of ontologies—their capacity for continuous semantic enrichment and evolution. For further illustration, Figure 3.5 presents our ontology as extracted from the Protégé[1] system, it shows all the defined classes and the associated rules.

[1] https://protege.stanford.edu/

**Table 3.2 : Tbox Ontology rules**

| Rule n. | Rule given in description logic |
|---|---|
| R1 | Customer ⊑ ∃hasOrder.Order |
| R2 | Order ⊑ ∃hasOrderDetail.OrderDetail |
| R3 | OrderDetail ⊑ ∃hasProduct.Product |
| R4 | Product ⊑ ∃hasCategory.Category |
| R5 | RecentOrder ⊑ Order ⊓ ∃hasOrderDate.(≥ "2022-08-01") |
| R6 | AvailableProduct ≡ Product ⊓ ∃hasStock.(> 0) |
| R7 | HighValueCustomer ≡ Customer ⊓ ∃hasOrder.(∃hasTotalAmount.(> 1000)) |
| R8 | LargeOrderDetail ≡ OrderDetail ⊓ ∃hasQuantity.(> 10) |
| R9 | FrequentCustomer ≡ Customer ⊓ (≥ 5 hasOrder.Order) |
| R10 | PopularProduct ≡ Product ⊓ (≥ 10 inverse(hasProduct).OrderDetail) |
| R11 | RecentOrder ⊑ Order ⊓ ∃hasOrderDate.(≥ "2024-05-01") |
| R12 | OutOfStockProduct ≡ Product ⊓ ∃hasStock.{0} |
| R13 | LargeOrderDetail ≡ OrderDetail ⊓ ∃hasQuantity.(> 10) |
| R14 | PremiumProduct ≡ Product ⊓ ∃hasPrice.(> 500) |
| R15 | FirstTimeCustomer ≡ Customer ⊓ (= 1 hasOrder.Order) |
| R16 | UnusualOrder ≡ Order ⊓ ∃hasTotalAmount.(> 10000) |

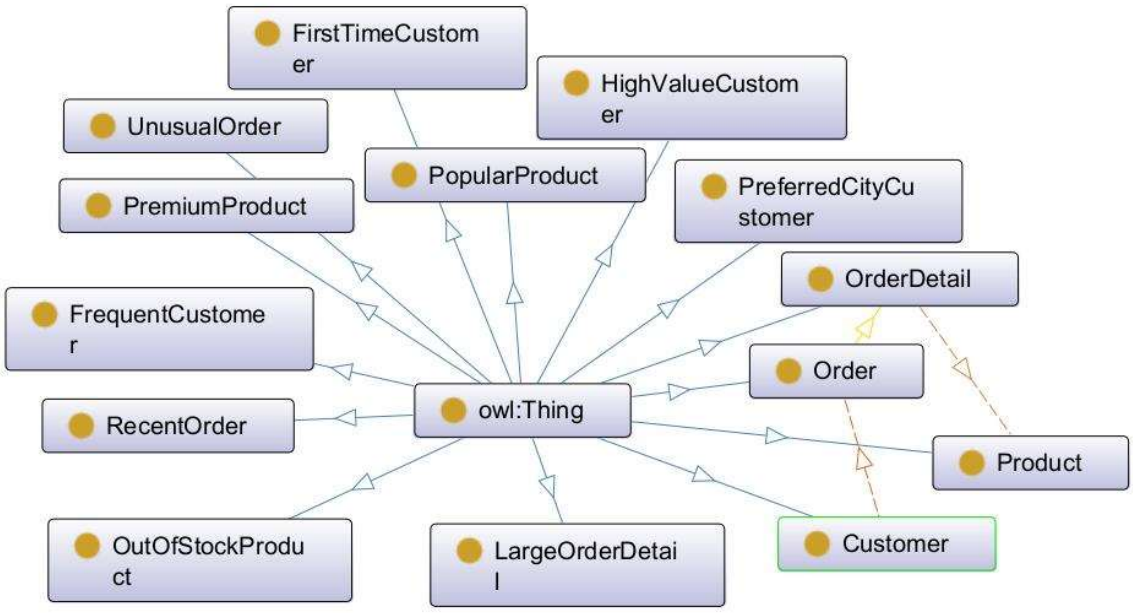


**Figure 3.5 : Visual representation of our ontology**

### 3.4.2 Dataset Preparation and Features

The dataset used for training and evaluating the machine learning model was extracted from a log file of SQL queries executed against the e-commerce database. The dataset comprises 3,000 SELECT queries, which were automatically labeled as either simple or complex based on their structural characteristics and execution time. The features used to train the model were extracted from each query, as described in the methodology section. These features include:

- **Number of Joins:** The number of JOIN operations in the query.
- **WHERE Flag:** A binary flag indicating the presence of a WHERE clause.
- **Execution Time:** The execution time of the query in milliseconds.
- **Complexity Score:** A composite score calculated based on the number of joins and the presence of a WHERE clause.

**Table 3.3 : A sample of training data**

| Query Content | Joins | where flag | Time (ms) | Score | Class |
|---|---|---|---|---|---|
| SELECT Name, Age, City FROM Customer; | 0 | 0 | 12 | 0.0 | Simple |
| SELECT OrderID, OrderDate FROM Order WHERE TotalAmount > 100; | 0 | 1 | 25 | 0.5 | Simple |
| SELECT c.Name, o.OrderDate FROM Customer c JOIN Order o ON c.CustomerID = o.CustomerID; | 1 | 0 | 42 | 1.0 | Simple |
| SELECT p.Name, od.Quantity FROM Product p JOIN OrderDetail od ON p.ProductID = od.ProductID; | 1 | 0 | 48 | 1.0 | Simple |
| SELECT c.Name, o.TotalAmount FROM Customer c JOIN Order o ON c.CustomerID = o.CustomerID WHERE o.TotalAmount > 200; | 1 | 1 | 86 | 1.5 | Complex |
| SELECT od.Quantity, p.Name FROM OrderDetail od JOIN Product p ON od.ProductID = p.ProductID WHERE p.Stock < 50; | 1 | 1 | 92 | 1.5 | Complex |
| SELECT c.Name, o.OrderDate, od.Quantity FROM Customer c JOIN Order o ON c.CustomerID = o.CustomerID JOIN OrderDetail od ON o.OrderID = od.OrderID WHERE od.Quantity > 1; | 2 | 1 | 130 | 2.5 | Complex |
| SELECT c.Name, p.Name FROM Customer c JOIN Order o ON c.CustomerID = o.CustomerID JOIN OrderDetail od ON o.OrderID = od.OrderID JOIN Product p ON od.ProductID = p.ProductID; | 3 | 0 | 145 | 3.0 | Complex |
| SELECT c.Name, p.Name FROM Customer c JOIN Order o ON c.CustomerID = o.CustomerID JOIN OrderDetail od ON o.OrderID = od.OrderID JOIN Product p ON od.ProductID = p.ProductID WHERE p.Price > 500 AND c.City = 'Paris'; | 3 | 1 | 165 | 3.5 | Complex |
| SELECT Name FROM Customer WHERE City = 'London'; | 0 | 1 | 20 | 0.5 | Simple |

A sample of the training data is presented in Table 3.3 of our work, which shows the extracted features and the corresponding class label for a set of representative queries.

### 3.4.3 Model Training and Validation

The machine learning model was trained using the Random Forest Classifier algorithm from the scikit-learn library. The model was configured with 100 trees (n_estimators=100) and the "class_weight" parameter was set to 'balanced' to mitigate the effects of class imbalance in the dataset. The dataset was split into a training set (80%) and a testing set (20%) to ensure a robust evaluation of the model's performance. The model was trained on the training set and then evaluated on the unseen testing set.

### 3.4.4 Evaluation Metrics and Methodology

The performance of the machine learning model was evaluated using a comprehensive set of classification metrics, including accuracy, precision, recall, F1-score, and the AUC score. These metrics provide a detailed assessment of the model's ability to correctly classify queries as simple or complex. The confusion matrix was also used to visualize the performance of the model and to identify any potential biases.

The performance of the query rewriting component was evaluated by comparing the execution time of the original complex queries with the execution time of the rewritten queries. This comparison provides a direct measure of the performance improvement achieved by the framework. The evaluation was conducted on the set of queries that were classified as complex by the machine learning model.

## 3.5 Results and Analysis

This section presents the results of the experimental evaluation and provides a detailed analysis of the findings. The results demonstrate the effectiveness of the proposed hybrid query optimization framework in terms of both classification accuracy and query performance improvement. The analysis focuses on the performance of the machine learning model, the effectiveness of the query rewriting component, and the importance of the features used for complexity prediction.

### 3.5.1 Classification Performance

The performance of the Random Forest Classifier was evaluated on the testing set, which comprised 20% of the total dataset of 3,000 SQL queries. The results, summarized in Figure 15 demonstrate the high accuracy and robustness of the model. The key performance metrics are as follows:

- **Accuracy:** The model achieved an overall accuracy of 90%, indicating that it correctly classified 9 out of 10 queries.
- **Precision:** The precision for the 'complex' class was 0.91, meaning that 91% of the queries classified as complex were indeed complex. The precision for the 'simple' class was 0.89.

- **Recall:** The recall for the 'complex' class was 0.92, indicating that the model successfully identified 92% of the actual complex queries. The recall for the 'simple' class was 0.87.
- **F1-Score:** The F1-score, which is the harmonic mean of precision and recall, was 0.91 for the 'complex' class and 0.88 for the 'simple' class. This indicates a good balance between precision and recall.
- **AUC Score:** The AUC score was 0.93, which is very close to 1, indicating that the model has an excellent ability to distinguish between simple and complex queries.

| | **Predicted Simple** | **Predicted Complex** |
|---|---|---|
| Actual Simple | 52 | 8 |
| Actual Complex | 4 | 36 |

a. Confusion Matrix.

| **Metric** | **Simple** | **Complex** | **Avg** |
|---|---|---|---|
| Precision | 0.89 | 0.91 | 0.90 |
| Recall | 0.87 | 0.92 | 0.895 |
| F1-Score | 0.88 | 0.91 | 0.895 |
| Support | 60 | 40 | 100 |

b. Classification Report.

| **Feature** | **Importance (%)** |
|---|---|
| Execution Time | 30% |
| Num Joins | 30% |
| Complexity Score | 25% |
| WHERE Flag | 15% |

| | |
|---|---|
| **Accuracy** | 90% |
| **AUC Score** | 0.93 |

d. Accuracy and AUC Score.

**Figure 3.6 : Model performance results**

The confusion matrix provides a more detailed breakdown of the classification results. It shows that the model had a low number of false positives and false negatives, which further confirms its high accuracy and reliability.

### 3.5.2 Query Optimization Effectiveness

The effectiveness of the query rewriting component was evaluated by comparing the execution time of the original complex queries with the execution time of the rewritten queries. The results showed that the rewriting process led to significant performance improvements. For example, a query like $SELECT * FROM\ Orders$ , which retrieves all orders, was rewritten to: $SELECT * FROM\ Orders\ WHERE\ OrderDate > '01-08-2022'$ based on the ontology axiom $RecentOrder \sqsubseteq Order \sqcap \exists hasOrderDate.(\geq "2022-08-01")$. This rewriting significantly reduced the number of rows returned by the query, which in turn led to a reduction in execution time

and memory usage. The experiment demonstrated that by leveraging the semantic constraints defined in the ontology, the framework was able to effectively prune the search space and eliminate irrelevant data, leading to more efficient query execution. The performance gains were particularly significant for queries that involved large tables and complex joins.

### 3.5.3 Feature Importance Analysis

A feature importance analysis was conducted to understand the contribution of each feature to the classification process. The results, presented in Figure 3.6, show that the execution time and the number of joins were the most influential features, each contributing 30% to the model's decision. The complexity score, which is a composite of the number of joins and the presence of a WHERE clause, was the next most important feature, with a contribution of 25%. The WHERE flag was the least important feature, with a contribution of 15%. This analysis provides valuable insights into the factors that contribute to query complexity. The high importance of execution time and the number of joins confirms the intuition that these are key indicators of query performance. The significant contribution of the complexity score validates its usefulness as a composite metric for predicting query complexity. These findings can be used to guide future feature engineering efforts and to further improve the accuracy of the machine learning model.

## 3.6 Discussion

The results of this research have significant implications for the field of database management and query optimization. The successful integration of ontology-based semantic reasoning and machine learning provides a powerful new paradigm for addressing the limitations of traditional query optimization techniques. This section discusses the key implications of the findings, the limitations of the research, and the potential practical applications of the proposed framework.

### 3.6.1 Implications of Findings

The high accuracy of the machine learning model demonstrates the feasibility of using a data-driven approach to predict query complexity. This is a significant departure from traditional cost-based approaches, which rely on predefined cost models that are often inaccurate and difficult to maintain. The ability to automatically and accurately classify queries as simple or complex enables a more dynamic and adaptive approach to query optimization, where resources are focused on the queries that are most likely to benefit from optimization.

The effectiveness of the query rewriting component highlights the value of incorporating semantic knowledge into the query optimization process. By leveraging the business rules and constraints defined in the ontology, the framework is able to rewrite queries in a way that not only improves

performance but also ensures semantic correctness. This is a critical advantage over traditional optimization techniques, which are often oblivious to the semantic context of the data.

The feature importance analysis provides valuable insights into the factors that contribute to query complexity. The high importance of execution time and the number of joins confirms the intuition that these are key indicators of query performance. This information can be used to guide the design of more effective query optimization strategies and to develop more accurate cost models.

### 3.6.2 Limitations and Challenges

Despite the promising results, this research has several limitations that should be acknowledged. First, the evaluation was conducted on a synthetic dataset, which may not fully reflect the complexity and diversity of real-world query workloads. Further evaluation on real-world datasets is needed to validate the generalizability of the findings.

Second, the process of creating the ontology and defining the semantic axioms is a manual and knowledge-intensive process that requires the involvement of a domain expert. This can be a significant bottleneck in the adoption of the proposed framework. Future research should explore the use of automated or semi-automated techniques for ontology learning and enrichment.

Third, the current framework only considers a limited set of features for predicting query complexity. There may be other features, such as the selectivity of the predicates and the size of the tables, that could further improve the accuracy of the model. Future research should explore the use of a wider range of features for query complexity prediction.

### 3.6.3 Practical Applications

The proposed framework has a wide range of practical applications in various domains, including e-commerce, finance, healthcare, and scientific research. In e-commerce, the framework can be used to optimize product search queries, to personalize product recommendations, and to improve the performance of the checkout process. In finance, the framework can be used to optimize risk analysis queries, to detect fraudulent transactions, and to improve the performance of portfolio management systems. In healthcare, the framework can be used to optimize clinical trial queries, to improve the accuracy of medical diagnoses, and to personalize treatment plans. In scientific research, the framework can be used to optimize data analysis queries, to integrate data from multiple sources, and to accelerate the pace of scientific discovery.

## 3.7 Conclusion

In this chapter we have presented a novel hybrid framework[60] that successfully integrates ontology based semantic reasoning with supervised machine learning to address the fundamental limitations of traditional SQL query optimization approaches. Through the development of a Random Forest

Classifier capable of achieving 90% accuracy in predicting query complexity and an automated semantic rewriting mechanism guided by OWL ontology axioms, in this research we have demonstrated that it is possible to optimize database queries for both performance and semantic correctness simultaneously. The experimental validation using a realistic e-commerce scenario with 3,000 SQL queries has provided compelling evidence that this approach can deliver significant performance improvements while ensuring that query results align with domain-specific business rules and constraints.

The implications of this research extend beyond immediate performance gains to encompass a fundamental shift toward more intelligent and context-aware database management systems. By demonstrating the feasibility of automatically identifying optimization opportunities through machine learning and applying semantic constraints derived from formal knowledge representation, this work establishes a foundation for the next generation of autonomous database systems that can adapt to changing data patterns and user requirements. The methodological innovations presented, including the systematic approach to database-to-ontology mapping and the feature engineering techniques for query complexity prediction, provide a replicable framework that can be extended to diverse application domains and database architectures, promising continued advancement in the field of intelligent data management and semantic query processing.

# General Conclusion

This dissertation has addressed the critical challenge of optimizing SQL queries in relational databases, a problem of enduring significance in an era of ever-expanding data. We began by identifying the "semantic gap" as a key limitation of traditional query optimization techniques. This gap, which separates the syntactic formulation of a query from its intended meaning, can lead to inefficient performance and, more critically, to inaccurate or misleading results. To bridge this gap, we have proposed and validated a novel hybrid framework that synergistically combines the predictive power of machine learning with the semantic reasoning capabilities of ontologies.

Our research has demonstrated that a supervised machine learning model, specifically a Random Forest Classifier, can be trained to accurately and reliably distinguish between simple and complex SQL queries. This classification provides a crucial, dynamic filtering mechanism, enabling our framework to apply its more computationally intensive semantic analysis selectively, only to those queries that are most likely to benefit from it. This targeted approach ensures that the overhead of semantic reasoning does not impede the performance of the system as a whole.

For those queries identified as complex, our framework employs an ontology-driven rewriting mechanism. We have shown how a domain ontology, encoding the key concepts and constraints of a given application domain, can be used to semantically enrich and optimize these queries. This process not only improves the efficiency of query execution but also enhances the semantic correctness and relevance of the results. By infusing the query optimization process with a deeper understanding of the data's meaning, our framework moves beyond the purely syntactic and cost-based methods that have traditionally dominated the field.

The primary contribution of this dissertation is, therefore, a significant advancement in the field of intelligent data management. We have designed, implemented, and empirically validated a novel hybrid architecture that demonstrates the power of integrating machine learning and ontologies for SQL query optimization. Our experiments, conducted in a realistic e-commerce case study, have provided strong evidence for the effectiveness of our approach, showing significant improvements in query performance and highlighting the accuracy of our ML-based classifier.

Beyond this primary contribution, our research also offers a number of more specific insights. We have provided a detailed analysis of the features that are most indicative of query complexity, offering valuable guidance for future research in this area. We have also presented a practical methodology for integrating ontologies into the query optimization process, a task that has often been a stumbling block for previous efforts in this domain. Our work provides a clear roadmap for how to build and deploy such a system, from the initial design of the ontology to the final evaluation of the ML model.

Of course, no research is without its limitations, and it is important to acknowledge them. The framework presented in this dissertation has been validated in a simulated environment, and further

research is needed to assess its performance in a real-world production setting. The effectiveness of our approach is also dependent on the quality and completeness of the domain ontology, the construction of which can be a time-consuming and labor-intensive process. Finally, while our ML model has demonstrated high accuracy, it is, like all such models, susceptible to errors, and the consequences of these errors must be carefully considered. A misclassified query could lead to a missed optimization opportunity or, conversely, to the unnecessary application of a computationally expensive rewriting process.

These limitations, however, also point the way to promising avenues for future research. The development of semi-automated methods for ontology construction and maintenance could significantly reduce the cost and effort required to deploy our framework. The exploration of more advanced machine learning techniques, such as deep learning and reinforcement learning, could lead to even more accurate and adaptive query complexity models. And the application of our framework to a wider range of application domains, from healthcare to finance, would provide a more comprehensive understanding of its strengths and weaknesses. Furthermore, investigating the use of our framework in a distributed or federated database environment would be a valuable extension of this work.

In conclusion, this dissertation has presented a significant step forward in the field of intelligent data management. By demonstrating the power of a hybrid approach that combines machine learning and ontologies, we have opened up a new and promising path towards the development of database systems that are not only more efficient and scalable but also more intelligent and semantically aware. It is our hope that the research presented here will serve as a foundation for future work in this exciting and important area, ultimately leading to a new generation of data management systems that are better able to meet the challenges of our increasingly data-driven world. The future of data management lies not in simply storing and retrieving data, but in understanding it, and the work presented in this dissertation is a testament to the power of that understanding.

# References


[1] A. Scherp, G. Groener, P. Škoda, K. Hose, and M.-E. Vidal, "Semantic Web: Past, Present, and Future," 2024, doi: 10.48550/ARXIV.2412.17159.

[2] A. R. Durmaz, A. Thomas, L. Mishra, R. N. Murthy, and T. Straub, "An ontology-based text mining dataset for extraction of process-structure-property entities," *Sci Data*, vol. 11, no. 1, p. 1112, Oct. 2024, doi: 10.1038/s41597-024-03926-5.

[3] Horrocks, I., Patel-Schneider, P. F., & Van Harmelen, "From SHIQ and RDF to OWL: the making of a Web Ontology Language," *Journal of Web Semantics*, pp. 7–26, 2003.

[4] R. S. I. Wilson, J. S. Goonetillake, W. A. Indika, and A. Ginige, "A conceptual model for ontology quality assessment: A systematic review," *SW*, vol. 14, no. 6, pp. 1051–1097, Dec. 2023, doi: 10.3233/SW-233393.

[5] Gruber, T., "What is an Ontology?," Stanford University., Knowledge Systems Laboratory, 1993.

[6] Studer, R., Benjamins, V. R., & Fensel, D., "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, pp. 161–197, 1998.

[7] Y. Zhu *et al.*, "LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities," *World Wide Web*, vol. 27, no. 5, p. 58, Sept. 2024, doi: 10.1007/s11280-024-01297-w.

[8] M. M. Dimartino, P. T. Wood, A. Cali, and A. Poulovassilis, "Efficient Ontology-Mediated Query Answering: Extending DL-liteR and Linear ELH," *jair*, vol. 82, pp. 851–899, Feb. 2025, doi: 10.1613/jair.1.16401.

[9] S. Meng, J. Zhou, X. Chen, Y. Liu, F. Lu, and X. Huang, "Structure-Information-Based Reasoning over the Knowledge Graph: A Survey of Methods and Applications," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 8, pp. 1–42, Sept. 2024, doi: 10.1145/3671148.

[10] R. S. I. Wilson, J. S. Goonetillake, W. A. Indika, and A. Ginige, "A conceptual model for ontology quality assessment: A systematic review," *SW*, vol. 14, no. 6, pp. 1051–1097, Dec. 2023, doi: 10.3233/SW-233393.

[11] L. Nachabe and N. Jahan, "A Proposed Ontology Evaluation Tool to Assist Ontology Engineers in Selecting Ontologies During the Reuse Phase," in *Knowledge Graphs and Semantic Web*, vol. 15459, S. Tiwari, B. Villazón-Terrazas, F. Ortiz-Rodríguez, and S. Sahri, Eds., in Lecture Notes in Computer Science, vol. 15459. , Cham: Springer Nature Switzerland, 2025, pp. 306–319. doi: 10.1007/978-3-031-81221-7_21.

[12] A. R. Durmaz, A. Thomas, L. Mishra, R. N. Murthy, and T. Straub, "An ontology-based text mining dataset for extraction of process-structure-property entities," *Sci Data*, vol. 11, no. 1, p. 1112, Oct. 2024, doi: 10.1038/s41597-024-03926-5.

[13] C. Yılmaz, Ç. Cömert, and D. Yıldırım, "Ontology-Based Spatial Data Quality Assessment Framework," *Applied Sciences*, vol. 14, no. 21, p. 10045, Nov. 2024, doi: 10.3390/app142110045.

[14] F. Baader, Ed., *The description logic handbook: theory, implementation, and applications*, 4. print. Cambridge: Cambridge Univ. Press, 2005.

[15] I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen, "From SHIQ and RDF to OWL: the making of a Web Ontology Language," *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, Dec. 2003, doi: 10.1016/j.websem.2003.07.001.

[16] B. Motik, R. Shearer, and I. Horrocks, “Hypertableau Reasoning for Description Logics,” *jair*, vol. 36, pp. 165–228, Oct. 2009, doi: 10.1613/jair.2811.

[17] W3C, “RDF 1.2 Concepts and Abstract Syntax,” W3C, 2025. [Online]. Available: https://www.w3.org/TR/rdf12-concepts/

[18] W3C, “OWL 2 Web Ontology Language Document Overview (Second Edition),” W3C, 2012. [Online]. Available: https://www.w3.org/TR/owl2-overview/

[19] Y. Zhu *et al.*, “LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities,” *World Wide Web*, vol. 27, no. 5, p. 58, Sept. 2024, doi: 10.1007/s11280-024-01297-w.

[20] W3C, “RDF 1.2 Semantics,” W3C Recommendation, 2025. [Online]. Available: https://www.w3.org/TR/rdf12- semantics/

[21] Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy, “Predicting Reasoning Performance Using Ontology Metrics,” in *The Semantic Web – ISWC 2012*, vol. 7649, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds., in Lecture Notes in Computer Science, vol. 7649. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 198–214. doi: 10.1007/978-3-642-35176-1_13.

[22] Y. Kazakov, M. Krötzsch, and F. Simančík, “The Incredible ELK: From Polynomial Procedures to Efficient Reasoning with $\mathcal {E} \mathcal {L}$ Ontologies,” *J Autom Reasoning*, vol. 53, no. 1, pp. 1–61, June 2014, doi: 10.1007/s10817-013-9296-3.

[23] I. Bilenchi, F. Gramegna, G. Loseto, S. Ieva, F. Scioscia, and M. Ruta, “Cowl: Pushing OWL 2 over the Edge,” *Internet of Things*, vol. 29, p. 101439, Jan. 2025, doi: 10.1016/j.iot.2024.101439.

[24] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, “HermiT: An OWL 2 Reasoner,” *J Autom Reasoning*, vol. 53, no. 3, pp. 245–269, Oct. 2014, doi: 10.1007/s10817-014-9305-1.

[25] G. Antoniou *et al.*, “A survey of large-scale reasoning on the Web of data,” *The Knowledge Engineering Review*, vol. 33, p. e21, 2018, doi: 10.1017/S0269888918000255.

[26] M. A. Musen, “The protégé project: a look back and a look forward,” *AI Matters*, vol. 1, no. 4, pp. 4–12, June 2015, doi: 10.1145/2757001.2757003.

[27] Studer, R., Benjamins, V. R., & Fensel, D, “Knowledge engineering: Principles and methods,” *Data & Knowledge Engineering*, pp. 161–197.

[28] G. Abuoda, D. Dell’Aglio, A. Keen, and K. Hose, “Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches -- extended version,” 2022, *arXiv*. doi: 10.48550/ARXIV.2210.05781.

[29] S. Khayatbashi, S. Ferrada, and O. Hartig, “Converting property graphs to RDF: a preliminary study of the practical impact of different mappings,” in *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, Philadelphia Pennsylvania: ACM, June 2022, pp. 1–9. doi: 10.1145/3534540.3534695.

[30] M. A. Musen, “The protégé project: a look back and a look forward,” *AI Matters*, vol. 1, no. 4, pp. 4–12, June 2015, doi: 10.1145/2757001.2757003.

[31] P. A. K. K. Diallo, S. Reyd, and A. Zouaq, “A Comprehensive Evaluation of Neural SPARQL Query Generation From Natural Language Questions,” *IEEE Access*, vol. 12, pp. 125057–125078, 2024, doi: 10.1109/ACCESS.2024.3453215.

[32] Sunitha Abburu, “A Survey on Ontology Reasoners and Comparison,” *International Journal of Computer Applications (IJCA)*, pp. 33–39, 2012.

[33] G. Xiao *et al.*, “Ontology-Based Data Access: A Survey,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 5511–5519. doi: 10.24963/ijcai.2018/777.

[34] M. Lenzerini, “Data integration: a theoretical perspective,” in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Madison Wisconsin: ACM, June 2002, pp. 233–246. doi: 10.1145/543613.543644.

[35] D. Lanti, M. Rezk, G. Xiao, and D. Calvanese, “The NPD Benchmark: Reality Check for OBDA Systems.” OpenProceedings.org, 2015. doi: 10.5441/002/EDBT.2015.62.

[36] O. Corcho, F. Priyatna, and D. Chaves-Fraga, “Towards a new generation of ontology based data access,” *SW*, vol. 11, no. 1, pp. 153–160, Jan. 2020, doi: 10.3233/SW-190384.

[37] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoué, Carsten Lutz, “OWL 2 Web Ontology Language: Profiles (Second Edition),” W3C, W3C Recommendation. [Online]. Available: https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/

[38] E. F. Codd, “A relational model of data for large shared data banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, June 1970, doi: 10.1145/362384.362685.

[39] C. J. Date, *Database Design and Relational Theory: Normal Forms and All That Jazz*. Berkeley, CA: Apress, 2019. doi: 10.1007/978-1-4842-5540-7.

[40] D. D. Chamberlin and R. F. Boyce, “SEQUEL: A structured English query language,” in *Proceedings of the 1976 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control - FIDET ’76*, Not Known: ACM Press, 1976, pp. 249–264. doi: 10.1145/800296.811515.

[41] A. R. Durmaz, A. Thomas, L. Mishra, R. N. Murthy, and T. Straub, “An ontology-based text mining dataset for extraction of process-structure-property entities,” *Sci Data*, vol. 11, no. 1, p. 1112, Oct. 2024, doi: 10.1038/s41597-024-03926-5.

[42] Ramakrishnan, R., & Gehrke, J., *Database Management Systems*, (3rd ed.). McGraw-Hill, 2003.

[43] T. R. Gruber, “A translation approach to portable ontology specifications,” *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, June 1993, doi: 10.1006/knac.1993.1008.

[44] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, “Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family,” *J Autom Reasoning*, vol. 39, no. 3, pp. 385–429, Oct. 2007, doi: 10.1007/s10817-007-9078-x.

[45] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, “The DL-Lite Family and Relations,” *jair*, vol. 36, pp. 1–69, Oct. 2009, doi: 10.1613/jair.2820.

[46] J. F. Sequeda and D. P. Miranker, “Ultrawrap: SPARQL execution on relational data,” *Journal of Web Semantics*, vol. 22, pp. 19–39, Oct. 2013, doi: 10.1016/j.websem.2013.08.002.

[47] F. Priyatna, O. Corcho, and J. Sequeda, “Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph,” in *Proceedings of the 23rd international conference on World wide web*, Seoul Korea: ACM, Apr. 2014, pp. 479–490. doi: 10.1145/2566486.2567981.

[48] Das, S., Sundara, S., and Cyganiak, R., "R2RML: RDB to RDF Mapping Language," W3C (World Wide Web Consortium), W3C Recommendation, Sept. 2012. [Online]. Available: https://www.w3.org/TR/r2rml/

[49] S. Ranatunga, R. S. Ødegård, K. Jetlund, and E. Onstein, "Use of Semantic Web Technologies to Enhance the Integration and Interoperability of Environmental Geospatial Data: A Framework Based on Ontology-Based Data Access," *IJGI*, vol. 14, no. 2, p. 52, Jan. 2025, doi: 10.3390/ijgi14020052.

[50] A. Randles, A. C. Junior, and D. O'Sullivan, "A Framework for Assessing and Refining the Quality of R2RML mappings," in *Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services*, Chiang Mai Thailand: ACM, Nov. 2020, pp. 347–351. doi: 10.1145/3428757.3429089.

[51] A. Iglesias-Molina, A. Cimmino, E. Ruckhaus, D. Chaves-Fraga, R. García-Castro, and O. Corcho, "An ontological approach for representing declarative mapping languages," *SW*, vol. 15, no. 1, pp. 191–221, Jan. 2024, doi: 10.3233/SW-223224.

[52] A. Iglesias-Molina, A. Cimmino, E. Ruckhaus, D. Chaves-Fraga, R. García-Castro, and O. Corcho, "An ontological approach for representing declarative mapping languages," *SW*, vol. 15, no. 1, pp. 191–221, Jan. 2024, doi: 10.3233/SW-223224.

[53] Harris, S., Seaborne, A., & Prud'hommeaux, E., "SPARQL 1.1 Query Language," 2013. [Online]. Available: https://www.w3.org/TR/sparql11-query/

[54] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data - SIGMOD '79*, Boston, Massachusetts: ACM Press, 1979, p. 23. doi: 10.1145/582095.582099.

[55] W. El Husseini, C. B. El Vaigh, F. Goasdoué, and H. Jaudoin, "Query Optimization for Ontology-Mediated Query Answering," in *Proceedings of the ACM Web Conference 2024*, Singapore Singapore: ACM, May 2024, pp. 2138–2148. doi: 10.1145/3589334.3645567.

[56] U. S. Chakravarthy, J. Grant, and J. Minker, "Logic-based approach to semantic query optimization," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 162–207, June 1990, doi: 10.1145/78922.78924.

[57] J. Unbehauen, C. Stadler, and S. Auer, "Optimizing SPARQL-to-SQL Rewriting," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, Vienna Austria: ACM, Dec. 2013, pp. 324–330. doi: 10.1145/2539150.2539247.

[58] T. Venetis, G. Stoilos, and G. Stamou, "Query Extensions and Incremental Query Rewriting for OWL 2 QL Ontologies," *J Data Semant*, vol. 3, no. 1, pp. 1–23, Mar. 2014, doi: 10.1007/s13740-012-0017-6.

[59] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[60] Meftah Lakehal and Mustapha Bourahla, "Ontology and Machine Learning Based SQL Query Rewriting and Optimization," *Journal of Information Systems Engineering and Management*, pp. 534–542, 2025.