Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Ontology and Machine Learning Based SQL Query Rewriting and Optimization

Meftah Lakehal[1,2], Mustapha Bourahla[1,2]
[1]*Computer Science Department, University of M'Sila, University Pole, Road Bordj Bou Arreridj, M'sila 28000, Algeria*
[2]*Laboratory of Informatics and its Application, University of M'Sila, University Pole, Road Bordj Bou Arreridj, M'sila 28000, Algeria*
Corresponding Author: meftah.lakehal@univ-msila.dz

ARTICLE INFO

Received: 30 Dec 2024

Revised: 12 Feb 2025

Accepted: 26 Feb 2025

ABSTRACT

In this work, we present a hybrid approach that integrates supervised machine learning with semantic reasoning through ontology to optimize SQL queries in relational databases. The relational schema is first mapped to an OWL ontology, where domain knowledge and business constraints are encoded as TBox axioms. A labeled dataset of SQL queries is extracted from query logs and used to train a Random Forest Classifier to distinguish between simple and complex queries based on selected features such as number of joins, presence of WHERE clauses, and execution time. Complex queries are then rewritten using semantic rules defined in the ontology, allowing for more efficient execution by constraining unnecessary data retrieval. Experimental results demonstrate high accuracy and effectiveness of the proposed system, showing significant performance gains in query execution. This work highlights the value of combining knowledge representation with machine learning for intelligent and context-aware data access strategies.

Keywords: Query Rewriting, Optimization, SQL, Ontologies, Machine Learning, Database Performance.

INTRODUCTION

Optimizing SQL queries is a critical challenge in database management systems (DBMS), particularly as data grow in size and complexity. Traditional query optimization techniques primarily focus on cost-based strategies, indexing, and heuristic rules to minimize execution time and resource usage [1]. However, these approaches often neglect semantic correctness, meaning that while queries may be executed efficiently, they might return results that do not fully align with the business logic or domain-specific constraints. For instance, in an e-commerce database, retrieving all orders without filtering outdated transactions may lead to misleading analytics and inaccurate decision-making. Addressing this issue requires an approach that integrates both performance optimization and semantic enrichment of queries.

To bridge this gap, we propose an ontology-based query optimization framework enhanced with machine learning techniques. Ontologies, expressed in Description Logic (DL), formally define the domain knowledge, specifying constraints such as minimum customer age, valid order dates, or stock availability. These constraints ensure that queries produce meaningful and domain-consistent results. By incorporating machine learning (ML)-based prediction, we can automatically detect complex queries that are likely to be inefficient or incomplete. Subsequently, a query rewriting mechanism modifies these queries to include the necessary semantic constraints derived from the ontology, ensuring both optimized execution and data integrity.

Existing works in ontology-based data access (OBDA) have demonstrated the effectiveness of integrating ontologies with relational databases [2]. OBDA systems map high-level ontology-based queries (e.g., SPARQL) into SQL queries while ensuring compliance with the domain model. However, traditional OBDA approaches do not incorporate machine learning-driven optimization or automatic query rewriting. Other studies, such as in [3], Trivela et al. explore query transformation techniques but rely on predefined rule-based systems, which lack adaptability to dynamic query workloads. Our approach extends this research by combining ontology-driven reasoning with ML-based query complexity prediction to dynamically enhance SQL queries.

Our proposed method consists of three key components: (1) a domain ontology defining semantic constraints, (2) an ML model trained on synthetic SQL query logs to predict query complexity, and (3) an automated query rewriting module. The ontology provides a formal semantic layer, ensuring compliance with business rules. The ML model predicts whether a query is complex based on structural features such as JOIN operations, WHERE clauses, and

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

estimated execution time. If a query is deemed complex, it is passed to the rewriting module, which applies missing constraints to improve performance and semantic correctness.

This work contributes to the fields of semantic query optimization, machine learning-based database optimization, and ontology-based data access. Our novel integration of ML and ontologies for SQL query enhancement extends previous work on both traditional query optimization [4] and ontology-driven reasoning [5]. Unlike static rule-based optimizations, our approach dynamically adapts to query variations and evolving data constraints, making it particularly useful in domains such as finance, healthcare, and e-commerce, where data quality and compliance are critical.

RELATED WORKS

The integration of ontologies and machine learning into SQL query optimization has been the focus of research in recent years, leading to the development of innovative approaches that enhance both the efficiency and semantic accuracy of database queries.

Ontology-Based Data Access (OBDA) frameworks have been pivotal in aligning query processing with domain-specific knowledge. These frameworks utilize ontologies to provide a conceptual layer over relational databases, ensuring that queries adhere to predefined semantic constraints. Calvanese et al. discuss the process of query reformulation in OBDA, emphasizing the elimination of redundant self-joins through semantic optimization techniques [6]. Further advancements in OBDA have led to the development of algorithms that optimize query rewriting by addressing combinatorial explosions and redundancies, thereby enhancing query performance.

The application of machine learning (ML) to query optimization has opened new avenues for automating and refining query execution plans. Yang's dissertation [7], introduces Balsa, a deep reinforcement learning agent that learns to optimize SQL queries through trial-and-error, outperforming traditional optimizers like PostgreSQL. Similarly, IBM's Db2 leverages ML to enhance query performance, achieving up to tenfold improvements in execution speed by analyzing historical query data to inform optimization strategies.

Recent studies have explored the synergy between ontologies and ML to further enhance query optimization. For instance, research by Lanti et al. [8] proposes a cost-driven OBDA approach that incorporates data statistics into the query translation process, resulting in more efficient query execution. Additionally, the integration of ML techniques for tuning query parallelism has been investigated, demonstrating that models can accurately predict query performance and assist in selecting optimal execution strategies.

The incorporation of artificial intelligence (AI) into query optimization has led to significant improvements in data retrieval speeds and analytical accuracy. A study published in the International Journal of Intelligent Systems and Applications in Engineering [9] proposes a machine learning model that accelerates query processing in large-scale data analytics, highlighting the potential of AI to revolutionize traditional query optimization methods.

METHODOLOGY

In figure 1, we depict the proposed pipeline of our methodology. This pipeline consists of two main parts: the machine learning component and the ontology processing component. We recall that our work aims to integrate ontology and machine learning to enhance query optimization in relational databases.

First, a log file serves as the main input of our pipeline. This file stores the history of transactions performed on relational databases within a given system. By transactions, we specifically refer to operations that involve querying data from the database. The language used for these queries is SQL (Structured Query Language), with the primary command being the SELECT statement.

We use supervised machine learning to determine whether a query is simple or complex. To achieve this, we train our model using data extracted from the database log file. These data points help identify the features used in training the model.

For this purpose, we consider several features, including the execution time of each query and the number of JOIN operations in the query. Additionally, we use a WHERE flag, which indicates whether the query contains a WHERE clause. Another key feature is the complexity score, which is calculated based on the number of JOIN operations and the presence of a WHERE clause. The complexity score cs can be computed as follows:

$$cs = 1.0 + 1.5 * j + 1.0 * w \qquad (1)$$

Where j represents the number of JOIN operations, and w denotes the presence of WHERE clauses.

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

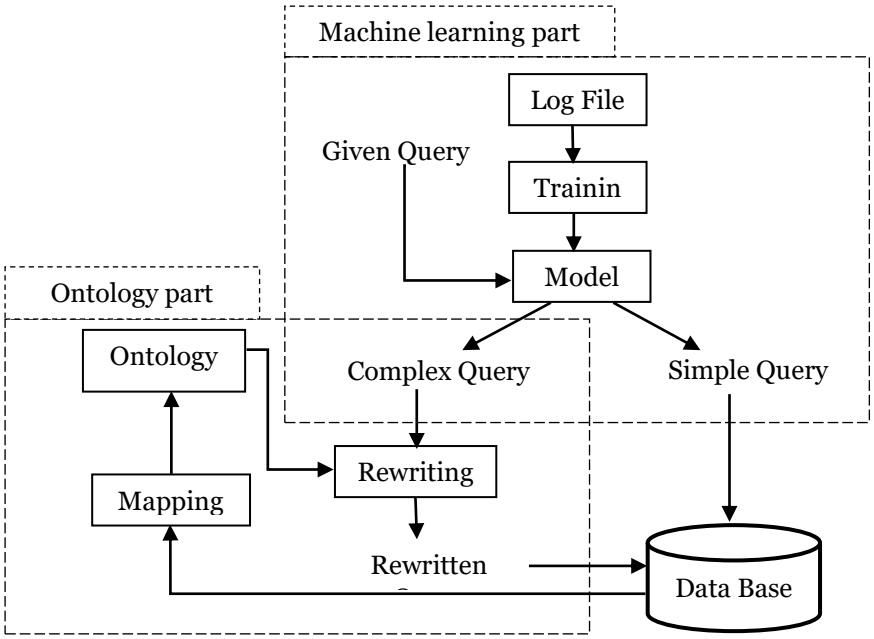

Figure 1. Pipeline of the Proposed Methodology.

The trained model determines whether a query is simple or complex. If classified as complex, the query will be rewritten based on the axioms defined in the ontology. Query rewriting helps reduce unnecessary data retrieval and optimizes performance. By integrating ontology-based reasoning, the system enhances query accuracy and relevance. For example, if a query is structured as follows:

$$SELECT * FROM\ Orders \quad (2)$$

And, if we have an axiom in the ontology given by :

$$Order \sqsubseteq \exists hasOrderDate.(2022-08-01) \quad (3)$$

The previous query in (2) will be rewritten as :

$$SELECT * FROM\ Orders\ WHERE\ Orders.OrderDate > '01-08-2022' \quad (4)$$

The query in (2) retrieves all order data, including old orders, which can negatively impact execution time and memory usage. The rewriting process 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
```

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

In our approach, the ontology acts as a semantic layer that regulates queries using axioms. The axiom in (3) specifies that only orders placed after '01-08-2022' should be considered and eliminating outdated records. By applying this axiom, the query returns a smaller relevant data and reduces computational overhead.

This optimization enhances execution speed and improves memory efficiency. Ontologies allow semantic filtering, ensuring that queries align with business constraints and data relevance. By enforcing domain knowledge through axioms, query execution becomes faster and more efficient. Additionally, memory consumption decreases as fewer records are processed. This approach ensures that database queries are both optimized and semantically accurate.

While ontology provides a semantic layer within a given production system, it is essential to clearly define how the ontology is designed and integrated. Generally, ontology can be derived from a relational database schema through a dedicated mapping process as given in the procedure billow:

```
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
```

Since each system has its own distinct database schema, it naturally results in the creation of a corresponding ontology tailored specifically to that system. The relationship connecting ontology and the underlying relational database schema is commonly referred to as mapping. Such mappings translate relational data structures into semantic constructs, making the underlying data understandable at a conceptual level. This allows users and applications to interact with the data using domain-specific concepts, rather than dealing directly with technical details of relational databases. Consequently, ontology acts as a conceptual abstraction over structured database content.

A relational database typically contains structured data organized into tables interconnected via relationships defined by foreign keys. On the other hand, ontology comprises concepts or classes interconnected through semantic properties, often enriched by axioms that enable reasoning and inference. This semantic richness differentiates ontologies significantly from traditional database schemas, enhancing the potential for sophisticated querying and reasoning capabilities. Establishing effective mappings between the relational schema and ontology can be performed manually, semi-automatically, or automatically by dedicated tools. Choosing the appropriate method of mapping depends on system complexity, the availability of domain experts, and specific project requirements. Thus, an accurate and efficient mapping ensures the effectiveness of the query rewriting process, enhancing semantic correctness and query optimization.

EXPERIMENTATION AND RESULTS

To test our approach, we applied it within an e-commerce scenario, illustrated in Figure 2. The scenario is represented as an entity-association model containing four entities: Customer, Order, OrderDetail, and Product. In this model, a customer can place multiple orders, each order can include multiple order details, and each order detail references one product. Conversely, each product may appear in multiple order details. For each entity, we specify its properties, noting those that serve as a primary key (PK) or a foreign key (FK).

Based on the previous model, a relational database can be created and queried using Structured Query Language (SQL). As described in the methodology section, queries can be either simple or complex, depending on the machine

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

learning model, which will be explained in the following paragraphs. Complex queries require optimization; to achieve this, corresponding ontologies are needed and can be generated through a mapping process.

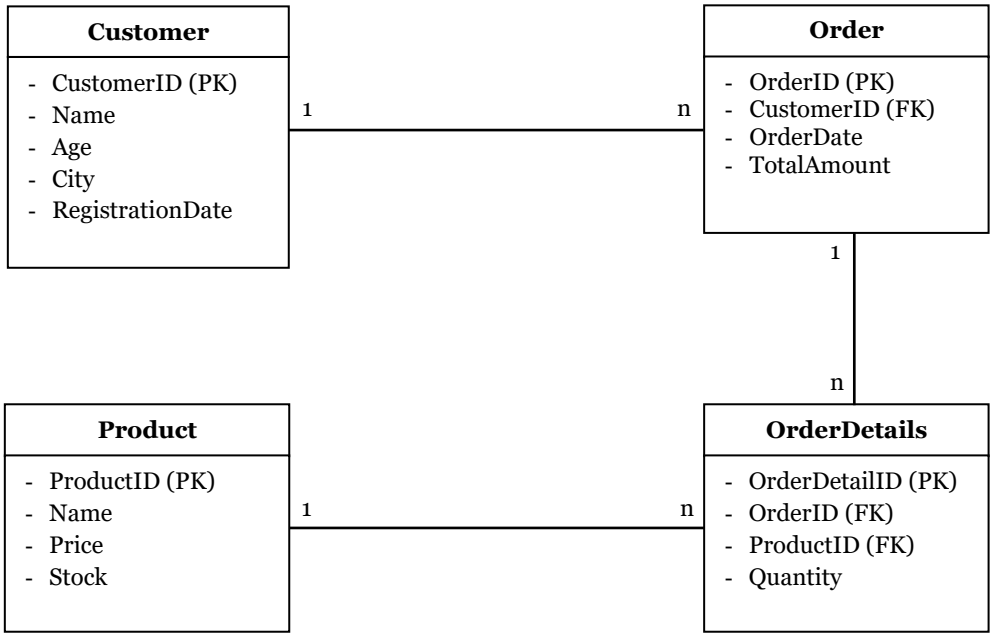

Figure 2. E-commerce scenario entity association model.

Based on the previous model, a relational database can be created and queried using Structured Query Language (SQL).

Table 1 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 2. 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 presents our ontology as extracted from the Protégé[1] system, it shows all the defined classes and the associated 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)

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

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Table 1. TBox terminology of the corresponding ontology.

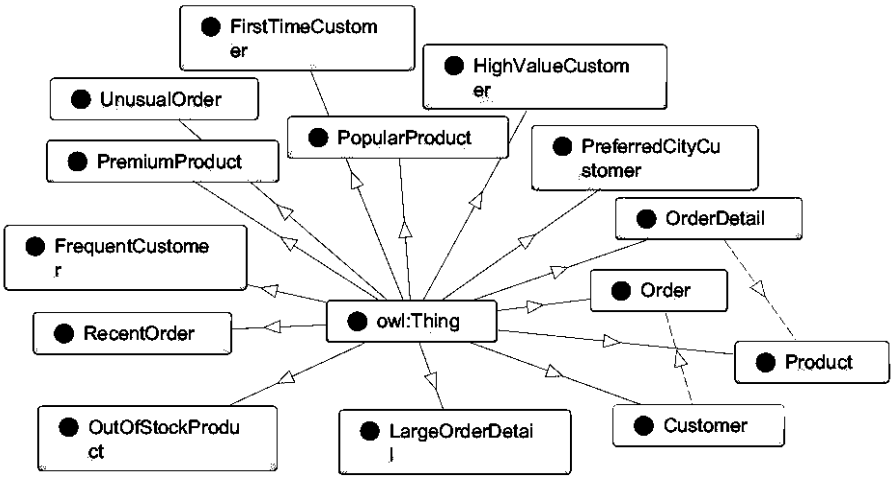

Figure 3. Visual representation of our ontology

We use the previously defined ontology to rewrite complex SQL queries, based on the model obtained through the machine learning process. As described in the methodology section, a supervised machine learning approach is employed to train a model that predicts whether a query is simple or complex. To achieve this goal, we adopted supervised learning, where the model is trained on labeled data extracted from SQL query logs. Each instance in the training dataset includes the SQL query itself, a set of extracted features (as detailed in the methodology section), and a label indicating whether the query is simple or complex.

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

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

SELECT Name FROM Customer WHERE City = 'London';	0	1	20	0.5	Simple

Table 2. A sample of trained data.

For classification, we used the Random Forest Classifier, a powerful ensemble learning algorithm that builds multiple decision trees and combines their outputs through majority voting to make final predictions. This algorithm is well-suited for both classification and regression tasks and is chosen for its robustness and ability to handle a mixture of numerical and categorical features. In our implementation, we used the RandomForestClassifier from the scikit-learn library, configuring it with 100 trees (n_estimators=100). The decision trees were permitted to expand without a predefined maximum depth, thereby allowing the model to fully capture complex patterns in the data. To ensure the reproducibility of results, the random_state parameter was fixed at 42. Moreover, to mitigate the effects of class imbalance in the dataset, the class_weight parameter was set to 'balanced', enabling the model to assign appropriate importance to each class during training.

To train our model, we utilized data extracted from a log file that records all SQL queries executed over a relational database, whose corresponding entity-relationship model is provided above. The dataset comprises 3,000 SELECT queries, which is deemed sufficient to achieve reliable results for a classification task of this nature. Table 2 presents a representative sample of queries used in the training process. Each entry in the table includes: the SQL query text, the number of joins involved, a binary flag indicating the presence of a WHERE clause (0: absent, 1: present), the execution time of the query, a computed complexity score (as detailed in the methodology section), and the final classification label (either simple or complex). To ensure the robustness and generalizability of our model, we adopted an approach with an 80/20 train-test split. This technique divides the dataset such that 80% of the data is used to train the machine learning model, while the remaining 20% is held out for validation and evaluation. This separation allows the model to learn patterns from majority of the dataset while reserving unseen data to objectively assess predictive performance. The dataset used to build and validate the classifier, leveraging key features such as described above.

To evaluate the performance of the trained model, we utilized in figure 4 a comprehensive set of classification metrics: accuracy, precision, recall, F1-score, AUC score, and the confusion matrix. Accuracy represents the overall correctness of the model, measuring the proportion of true results (both true positives and true negatives) among the total number of cases examined. Precision and recall provide a more nuanced view of performance, especially in imbalanced datasets. Precision measures the proportion of correctly identified complex queries out of all queries classified as complex, while recall indicates how many of the actual complex queries were successfully identified. The F1-score, as the harmonic mean of precision and recall, balances both metrics, providing a single metric that considers both false positives and false negatives.

	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%

c. Feature Importance.

Accuracy	90%
AUC Score	0.93

d. Accuracy and AUC Score.

Figure 4. Metrics of our trained model

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The classification report further breaks down the performance of the model by class (simple vs. complex), offering insights into how well each category is handled. It includes per-class precision, recall, and F1-score, helping identify any class imbalance or model bias. In addition, the Area Under the Receiver Operating Characteristic Score (AUC-Score) was used to assess the model's ability to distinguish between the two classes across all classification thresholds. A high AUC score close to 1 indicates that the model effectively separates simple and complex queries, which is critical in optimizing only those queries that truly benefit from ontology-based rewriting.

To understand the contribution of each feature in the classification process, we conducted a feature importance analysis using the inherent interpretability of the Random Forest classifier. This analysis ranks the features based on their influence on the final decision. In our findings, the execution time and number of joins emerged as the most influential predictors, reflecting the complexity and performance cost associated with SQL queries. The complexity score, which combines structural elements such as joins and the presence of WHERE clauses, also showed significant weight, validating its usefulness as a composite metric. This insight not only guides further feature engineering but also supports explainability in real-world deployments.

The results obtained through training and validation demonstrate that the proposed model is highly effective for classifying SQL queries in terms of complexity. With strong performance across key metrics and a transparent understanding of contributing factors, this model can be seamlessly integrated into production systems. Its output enables dynamic decision-making, where complex queries are identified and rewritten using ontology-driven rules, enhancing performance and ensuring adherence to semantic constraints. The combination of data-driven learning and rule-based reasoning underscores the strength of our hybrid approach to query optimization in relational database systems.

CONCLUSION

In this study, we proposed a hybrid approach combining ontology-driven semantic reasoning with supervised machine learning to enhance SQL query optimization in relational database systems. Our methodology integrates a semantic layer—modeled as an OWL ontology—derived from the database's entity-relationship model. The ontology provides domain knowledge in the form of TBox axioms and semantic rules, which are then leveraged to rewrite complex queries for improved performance. We also developed a machine learning model, specifically a Random Forest Classifier, trained on features extracted from SQL logs to automatically classify queries as simple or complex. This classification determines whether a query should be rewritten according to the defined optimization rules embedded in the ontology.

Experimental evaluations on a dataset of 3,000 SQL queries demonstrated the effectiveness of our approach. Key evaluation metrics, including accuracy, precision, recall, F1-score, and AUC-ROC, showed strong predictive performance, confirming the robustness of our classifier. Moreover, the integration of ontology allowed for meaningful semantic query transformations, leading to reduced execution time and enhanced system efficiency. The work emphasizes the practical benefits of combining AI techniques with knowledge representation to address performance bottlenecks in data-intensive environments. This solution opens new directions for intelligent data management systems that are both scalable and semantically enriched.

REFRENCES

[1] Ramakrishnan, R., & Gehrke, J. (2002). Database Management Systems (3rd ed.). McGraw-Hill.
[2] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2013). Ontology-Based Data Access and Integration. In A. Polleres & C. d'Amato (Eds.), Reasoning Web. Semantic Technologies for Intelligent Data Access (pp. 1–24). Springer.
[3] Trivela, D., Calbimonte, J. P., Corcho, O., & Ruckhaus, E. (2019). A Benchmarking Framework for SQL Query Rewriting Systems. Journal of Web Semantics, 58, 100472.
[4] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., & Price, T. G. (1979). Access Path Selection in a Relational Database Management System. Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, 23–34.
[5] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (2003). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.
[6] Calvanese, D. (2018). Ontology-based data access: A tutorial on query reformulation and optimization. Presented at ONTOBRAS 2018.

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[7] Yang, Z., Chiang, W.-L., Luan, S., Mittal, G., Luo, M., & Stoica, I. (2022). Balsa: Learning a query optimizer without expert demonstrations. Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data, 2046–2059.
[8] Lanti, D., Xiao, G., & Calvanese, D. (2017). Cost-driven ontology-based data access (extended version). arXiv preprint arXiv:1707.06974.
[9] Veeraiah, V., Ravikumar, G. K., Gupta, N., Singh, D., Rathod, V. M., & Yellapragada, R. K. (2023). Diagnosing of disease using machine learning in healthcare by Internet of Things. International Journal of Intelligent Systems and Applications in Engineering, 11(10s), 954–973.