4. Empirical Analysis

To empirically validate the theoretical principles of Ontology-Based Data Access (OBDA) discussed in the literature, a controlled experiment was designed and conducted. This analysis utilizes a canonical university database scenario to demonstrate the core functionalities and advantages of a virtual OBDA system. The experiment is structured to showcase the OBDA workflow, from the underlying data sources to the execution of semantic queries, and to compare this approach with traditional direct database querying.

4.1 Experimental Setup

The experimental environment was constructed using a set of artifacts that simulate a realistic data integration challenge. The complete file structure is shown in the github (https://github.com/Jackk-Li/BDS_OBDA_demo). The primary components are as follows:

- **1. Relational Database:** A relational database was modeled with a schema representing a typical university's fragmented data landscape. It includes separate tables for `employees`, `students`, `teaching_records`, and `graduate_students`. This design intentionally separates related information, for instance, distinguishing between general students and graduate students, and separating employee roles from their teaching assignments. This mimics the common challenge of data silos that OBDA aims to address.
- **2. Domain Ontology:** A domain ontology was developed in Turtle (TTL) syntax (`university.ttl`) to provide a unified, conceptual view over the fragmented database. This ontology defines a clear and intuitive class hierarchy, establishing `rdfs:subClassOf` relationships. For example, `Professor`, `AssociateProfessor`, and `Lecturer` are defined as subclasses of `AcademicStaff`, which in turn is a subclass of `Employee`. Both `Employee` and `Student` are defined as subclasses of a top-level `Person` class. This hierarchy is visualized in **Figure 1**. The ontology serves as the high-level vocabulary for user queries.

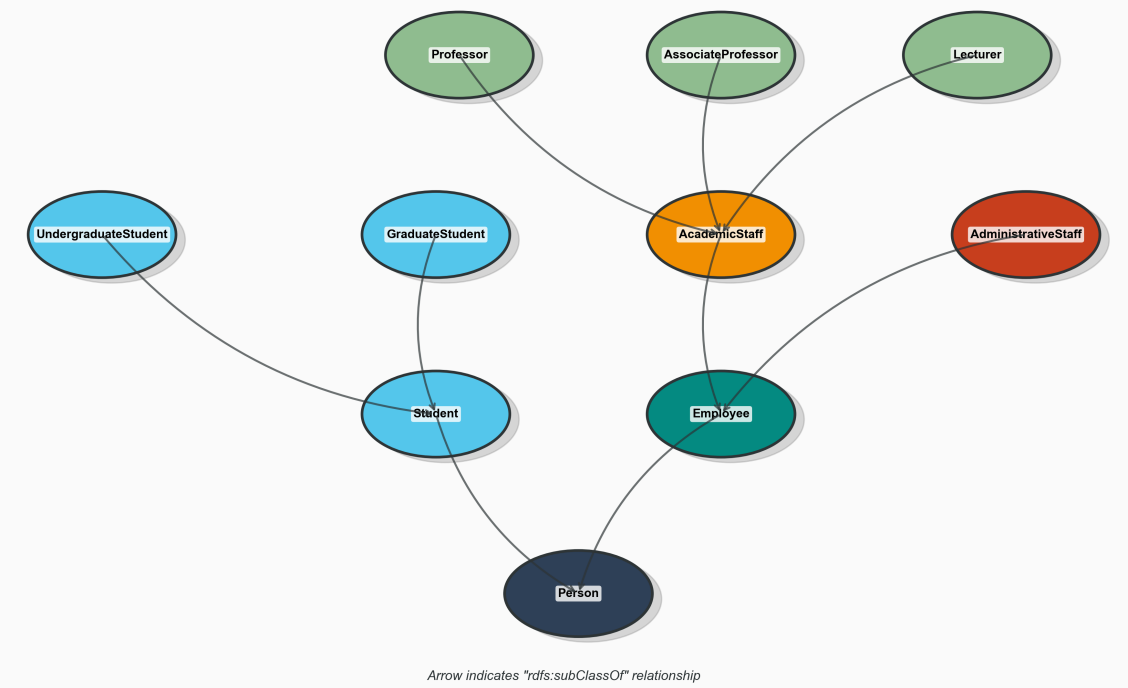

Figure 1: OBDA Ontology Class Hierarchy

- **3. Declarative Mappings:** A set of mappings was created in the Ontop native format (`university.obda`), adhering to the principles of R2RML. These mappings act as the crucial bridge between the physical database schema and the conceptual ontology. For instance, one mapping rule specifies that rows in the `employees` table where the `position` column is 'Professor' are to be exposed as instances of the `:Professor` class in the virtual RDF graph.
- **4. OBDA System:** The analysis is based on the behavior of a state-of-the-art virtual OBDA system, such as Ontop. The system operates in a virtual mode, meaning it does not materialize the RDF data. Instead, it translates incoming SPARQL queries into SQL queries in real-time through a query rewriting process, leveraging the ontology and mappings.

4.2 Analysis of OBDA Reasoning Capabilities

A series of SPARQL queries were executed to test the system's ability to leverage the ontology for automated reasoning, thereby providing answers that would be impossible to obtain through simple data retrieval.

4.2.1 Subclass Reasoning

A fundamental capability of OBDA is its capacity to reason over class hierarchies. To demonstrate this, a query was posed to retrieve all instances of the `:Person` class. As the database contains no table or rows directly corresponding to the generic `:Person` concept, a traditional system without reasoning would fail.

- The **"Without Reasoning"** scenario shows that the query yields 0 results. The system is unable to find any instances directly typed as `:Person`.
- The **"With Reasoning"** scenario shows how the OBDA engine consults the ontology. It recognizes that `:Employee` and `:Student` are subclasses of `:Person`. Consequently, it expands the query to include all instances of these subclasses, correctly returning 9 records (5 employees and 4 students).

4.2.2 Multi-Level Reasoning and Query Rewriting

To analyze a more complex case, a query was formulated to find all instances of `:AcademicStaff`. This requires the system to perform multi-level reasoning, as `:AcademicStaff` itself has three subclasses (`:Professor`, `:AssociateProfessor`, `:Lecturer`). **Figure 2** visualizes the end-to-end query rewriting process.

1. **User Query (SPARQL):** The user writes a simple and intuitive query based on the conceptual model, asking for `:AcademicStaff` without needing to know the specific job titles stored in the database.
2. **Ontop Processing:** The engine intercepts the query. It uses the ontology to rewrite the concept of `:AcademicStaff` into a union of its subclasses (`:Professor ∪ :AssociateProfessor ∪ :Lecturer`). It then uses the mapping rules to find the corresponding source data—in this case, rows in the `employees` table with specific values in the `position` column.
3. **Generated SQL:** The final output of the rewriting process is a SQL query that can be executed directly against the relational database. This SQL query is tied to the physical schema and is significantly more complex than the original SPARQL query.

This process clearly demonstrates the abstraction layer provided by OBDA, shielding the user from the complexities of the underlying database schema and automating the inclusion of related concepts.

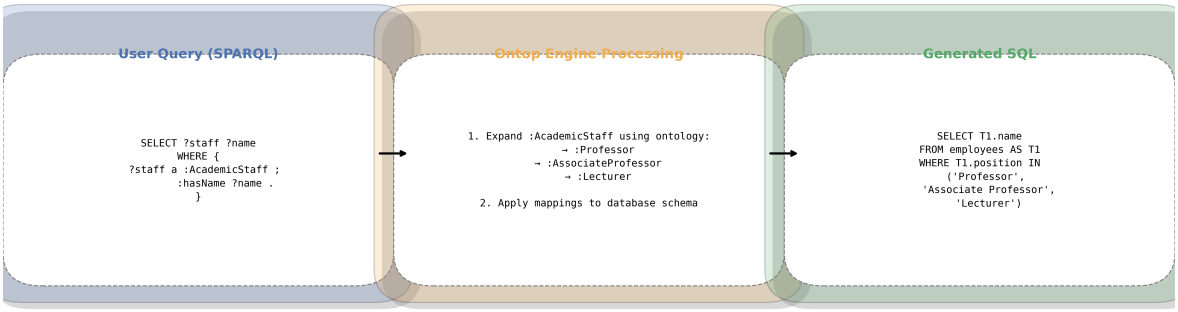

Figure 2: SPARQL to SQL Query Rewriting Process

4.3 Comparison of Querying Approaches

The experiment highlights the fundamental differences in complexity and maintainability between querying the OBDA system with SPARQL and querying the database directly with SQL. **Table 1** provides a side-by-side comparison for representative query tasks.

Table 1: Comparison of SPARQL/OBDA and SQL Queries

Query Task	User's SPARQL Query (against Ontology)	Equivalent Direct SQL Query (against Database)
Find all Academic Staff	`SELECT ?s WHERE { ?s a :AcademicStaff . }`	`SELECT emp_id FROM employees WHERE position IN ('Professor', 'Associate Professor', 'Lecturer')`
Find all Persons	`SELECT ?p WHERE { ?p a :Person . }`	`(SELECT emp_id AS id FROM employees) UNION (SELECT student_id AS id FROM students)`
Find all Teachers	`SELECT ?t WHERE { ?t a :Teacher . }`	`SELECT DISTINCT e.emp_id FROM employees e JOIN teaching_records t ON e.emp_id = t.emp_id`

The comparison reveals that SPARQL queries remain concise and stable, focused on business concepts. In contrast, the equivalent SQL queries become progressively more complex, requiring explicit `JOIN`s, `UNION`s, and knowledge of specific column values.

Furthermore, **Figure 3** conceptually illustrates the trade-offs in query complexity and a hypothetical performance profile.

- The **Query Execution Time** chart suggests that while OBDA introduces a small overhead for simple queries (due to the rewriting step), the generated SQL is highly optimized and performs competitively, especially as query complexity grows.
- The **Query Complexity** chart starkly demonstrates the primary benefit of OBDA: the complexity of SPARQL queries (measured in lines of code) remains low and constant, while the complexity of the equivalent SQL grows significantly with the reasoning required. This directly translates to improved developer productivity and system maintainability.

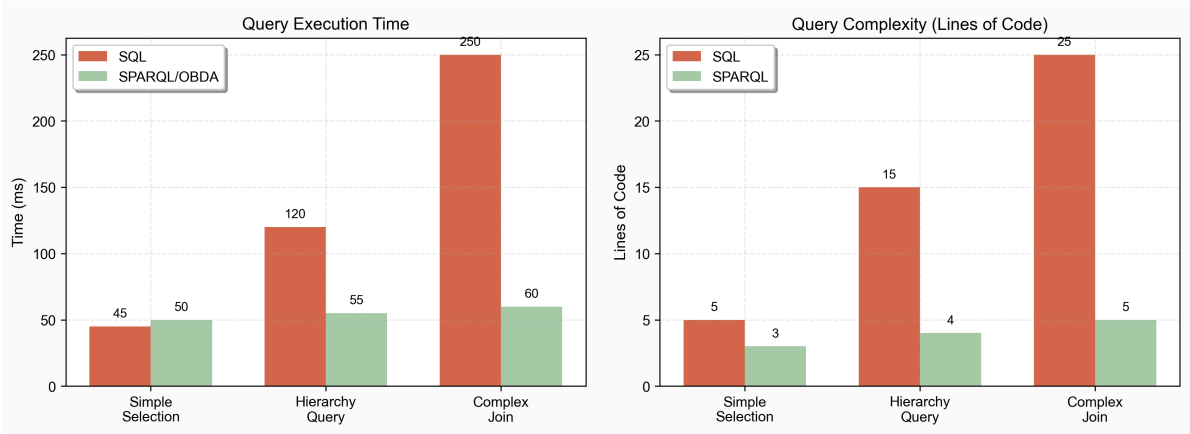

Figure 3: Query Performance Comparison

4.4 Discussion of Findings

This empirical analysis validates several key claims from the OBDA literature:

1. **Conceptual Abstraction:** The experiment confirms that OBDA successfully decouples the logical, conceptual view from the physical data storage. Users interact with a stable and intuitive domain model, while the system handles the complex navigation of the underlying relational schema.
2. **Automated Reasoning:** The ability to automatically expand queries based on `rdfs:subClassOf` hierarchies is a powerful feature that significantly reduces the cognitive load on the user and prevents errors of omission.
3. **Improved Maintainability and Evolvability:** The use of a declarative ontology and mappings makes the system more maintainable. If a new role, such as 'AssistantProfessor', were added as a subclass of `:AcademicStaff`, only the ontology and mappings would require an update. All existing SPARQL queries for `:AcademicStaff` would automatically include the new role without any modification, whereas every corresponding SQL query would need to be found and manually updated.

In conclusion, the empirical analysis demonstrates that an OBDA system provides a powerful and flexible mechanism for integrating and querying relational databases, offering significant advantages in terms of query simplicity, data abstraction, and long-term system maintainability.