Object-relational database

Erick Gonzalez Parada 178145
Antonio Gutiérrez Blanco 177442
Andre Francois Duhamel Gutierrez 177315
Emiliano Ruiz Plancarte 177478

February 7, 2025

1 Objective

To design, implement, and evaluate an object-relational database system by defining complex object types, creating relational tables with nested collections, and establishing relationships between objects. The goal is to demonstrate the effective use of Object Query Language (OQL) for managing object-relational data structures, inserting sample data, and performing queries to validate the schema's functionality and integrity.

2 Introduction

There is an EXTRA section at the end
please don't forget

Object Query Language (OQL) is a standardized query language designed for querying and manipulating object-oriented databases. Unlike traditional SQL, which is tailored for relational databases, OQL is specifically built to handle complex object-oriented data structures, such as inheritance, polymorphism, and nested collections. OQL allows developers to interact with object-relational databases in a way that aligns with the principles of object-oriented programming, making it a powerful tool for modern database systems[2].
OQL revolves around its ability to query objects and their relationships directly, rather than relying on flat, tabular structures. This is particularly useful

in scenarios where data is represented as objects with attributes, methods, and relationships to other objects[1].

3 Literature Review

Traditional SQL (Structured Query Language) is designed for relational databases, where data is stored in flat, tabular structures. The philosophy of SQL revolves around the use of **joins** to combine data from multiple tables, enabling users to retrieve related information. For example, to retrieve data about an employee and their department, SQL requires joining the `Employee` and `Department` tables using a common key. While this approach is effective for relational data, it becomes cumbersome and inefficient when dealing with complex, hierarchical, or object-oriented data structures[3].

In contrast, Object Query Language (OQL) is specifically designed for object-oriented databases, where data is represented as objects with attributes, methods, and relationships. OQL eliminates the need for explicit joins by allowing developers to **directly traverse object relationships**. For instance, in an object-relational database, an `Employee` object might have a direct reference to a `Department` object. With OQL, querying the department of an employee is as simple as navigating the object hierarchy (e.g., `employee.department`), without the need for complex joins or intermediate tables[1].

While SQL can simulate some object-oriented behavior—such as using foreign keys and nested queries to represent relationships—it lacks the native support for object-oriented principles like **inheritance**, **polymorphism**, and **nested collections**. For example, simulating inheritance in SQL requires creating multiple tables and using complex joins to represent parent-child relationships. Similarly, representing nested collections (e.g., a list of phone numbers for an employee) often involves creating additional tables and performing multiple joins, which can lead to inefficient queries and increased complexity[2].

OQL, on the other hand, natively supports these object-oriented features. It allows developers to define **complex object types**, **inheritance hierarchies**, and **nested collections** directly in the database schema. This makes OQL more intuitive and efficient for querying object-oriented data, as it aligns with the natural structure of the data. For example, querying a nested collection of phone numbers in OQL is straightforward (e.g., `employee.phones`), whereas in SQL, it requires joining multiple tables and aggregating results[1].

In summary, while SQL remains a powerful tool for relational data, its reliance on joins and lack of native support for object-oriented principles make it less suitable for modern, object-oriented applications. OQL addresses these limitations by providing a query language that is inherently designed for object-relational databases, enabling more efficient and intuitive data retrieval and manipulation.

4 Methodology

5 Result Analysis

6 Conclusions

This work demonstrated the effectiveness of Object Query Language (OQL) in managing object-relational databases. **However we as a team struggled a ton to get this tools working due to the lack of documentation that is online.** By defining complex object types, nested collections, and relationships, we showed that OQL simplifies querying object-oriented data compared to SQL, which relies on complex joins. OQL's native support for object-oriented principles, such as inheritance and polymorphism, makes it a superior choice for modern applications. This highlights OQL's potential for advancing database technologies in object-oriented contexts.

7 EXTRA

References

[1] OQL. (n.d.). Mendix Documentation. Retrieved February 7, 2025, from https://docs.mendix.com/refguide/oql/

[2] Tivoli Network Manager IP edition 4.2.0. (2025, January 21). Ibm.com. https://www.ibm.com/docs/en/networkmanager/4.2.0?topic=reference-object-query-language

[3] What is SQL. (n.d.). W3schools.com. Retrieved February 7, 2025, from https://www.w3schools.com/whatis/whatis_sql.asp