Object-relational database

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

February 17, 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

To implement and evaluate the object-relational database system, we used **Oracle Database Express Edition (XE)** as the primary database management system. The general solution involved the following steps:

1. **Setting Up Oracle Database Express**: - Oracle Database XE was downloaded and installed on a local machine. During installation, the default settings were used, and a system password was set for the administrative user (`SYS`). - The database was configured to allow remote connections, and the necessary ports (e.g., 1521 for Oracle) were opened to enable access from external tools.

2. **Configuring Visual Studio Code (VSCode)**: - The **Oracle Developer Tools for VSCode** extension was installed to facilitate database connectivity and query execution. - A connection profile was created in VSCode, specifying the host (e.g., `localhost`), port (e.g., `1521`), service name (e.g., `XE`), and credentials (e.g., `SYS` and the system password).

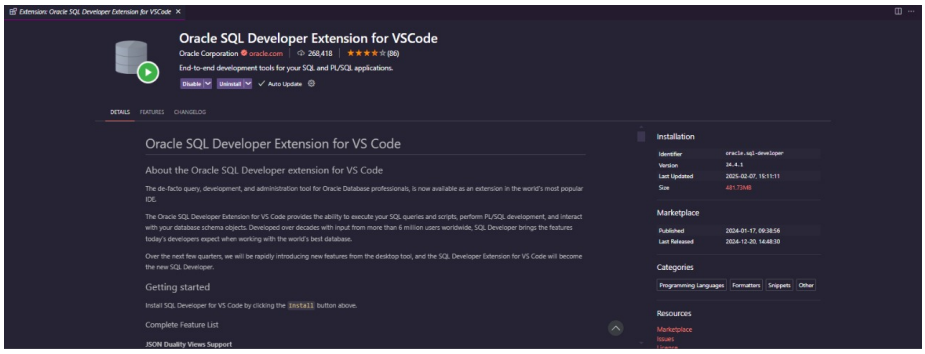

Figure 1: VSCode extension

3. **Writing and Executing OQL Scripts**: - OQL scripts were written to define complex object types, create relational tables with nested collections, and establish relationships between objects. These scripts were saved as `.sql` files for execution. - The scripts included: - **Object Type Definitions**: Creating object types such as `employee_t`, `worker_t`, and `manager_t` with attributes and methods. - **Table Creation**: Defining tables with nested collections (e.g., `telephone_t` as a nested table of phone numbers). - **Data Insertion**: Inserting sample data into the tables to populate the database. - **Query Execution**: Writing and executing queries to retrieve and validate data from the object-relational schema.

4. **Challenges Faced**: - One of the major challenges was the **lack of comprehensive online documentation** for OQL, especially in the context of Oracle Database. This made it difficult to troubleshoot issues and find examples of advanced OQL features. - Additionally, configuring the Oracle extension in VSCode required trial and error, as some settings (e.g., connection parameters) were not well-documented.

5. **Validation and Testing**: - After executing the OQL scripts, the database schema and data were validated by running queries to ensure that object relationships, nested collections, and inheritance hierarchies were func-

tioning as expected. - The results were compared with the expected outcomes to verify the correctness and efficiency of the implemented schema.

5 Result Analysis

The following figures will show us the whole work done for our solution and how the oql compiled successfully:

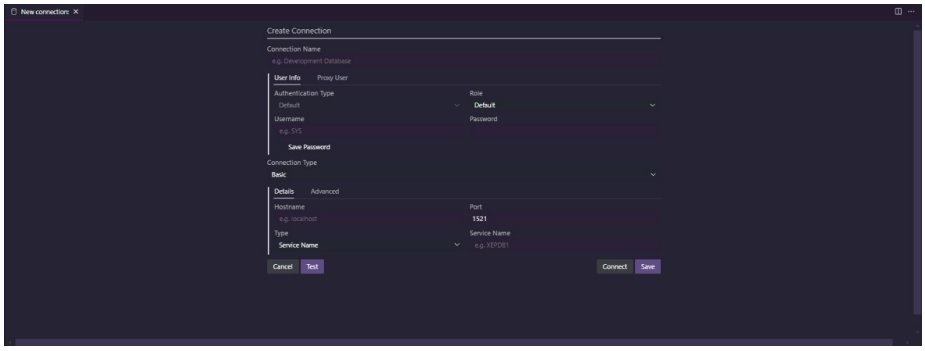

Figure 2: Create connection

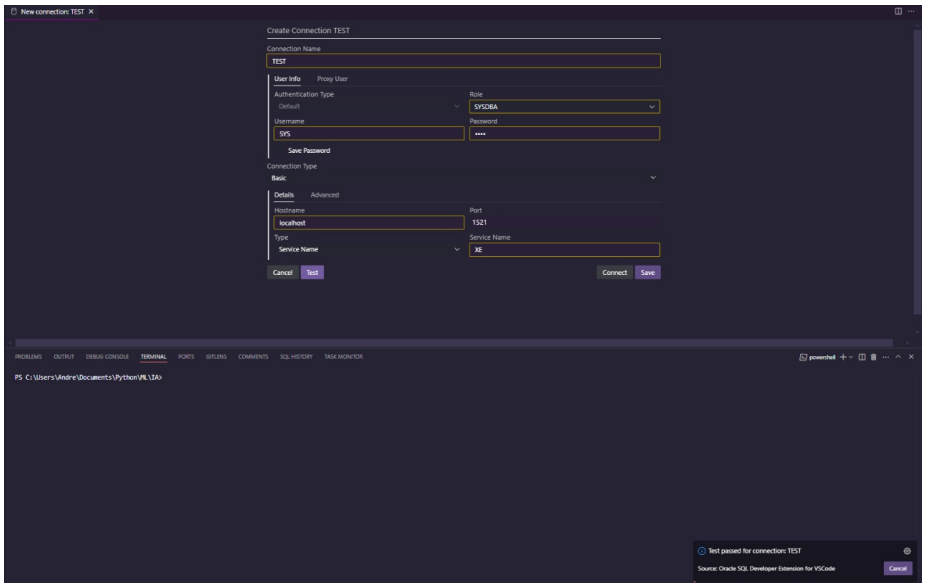

Figure 3: Setting connection

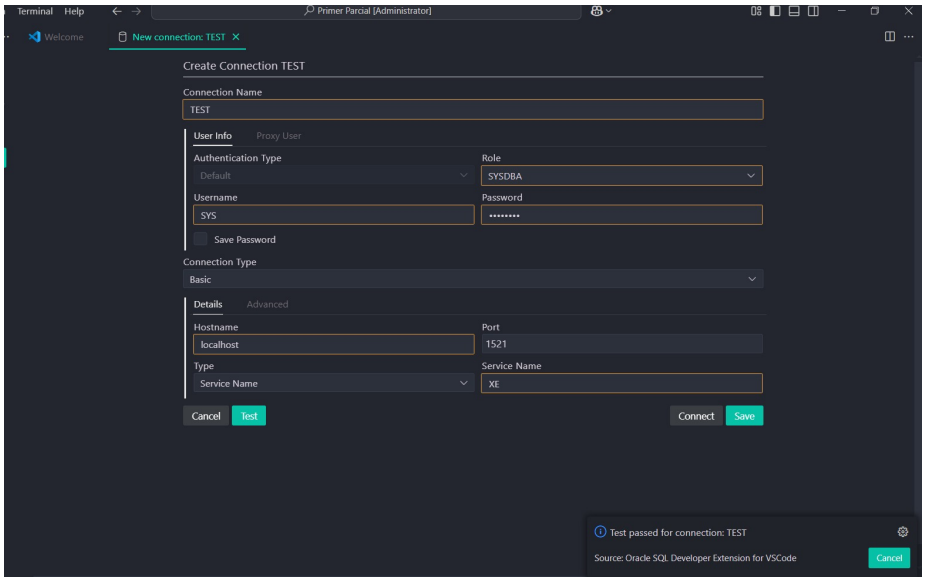

Figure 4: Setting connection

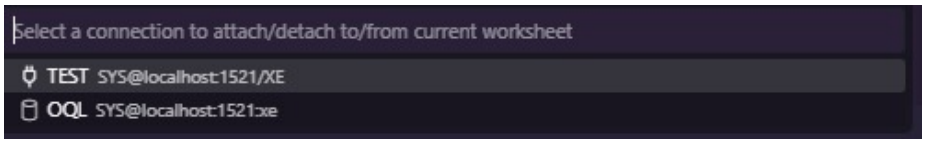

Figure 5: Connection selection

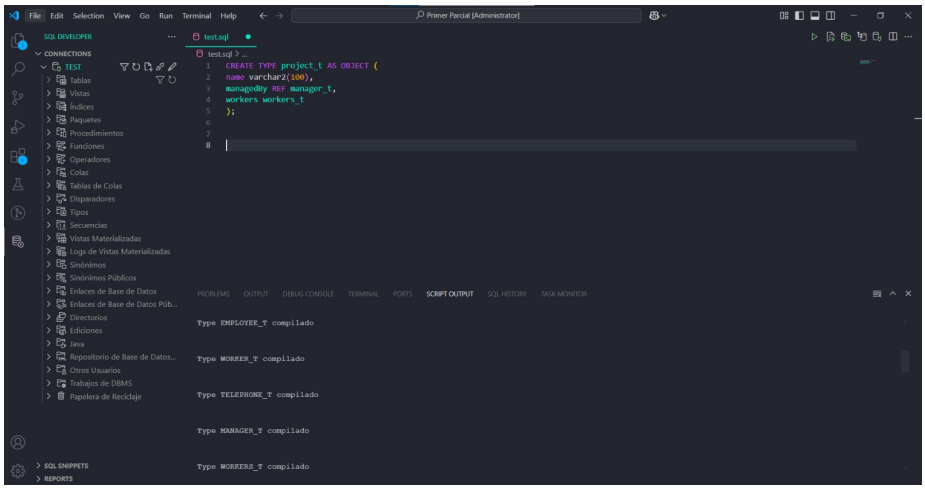

Figure 6: Create query

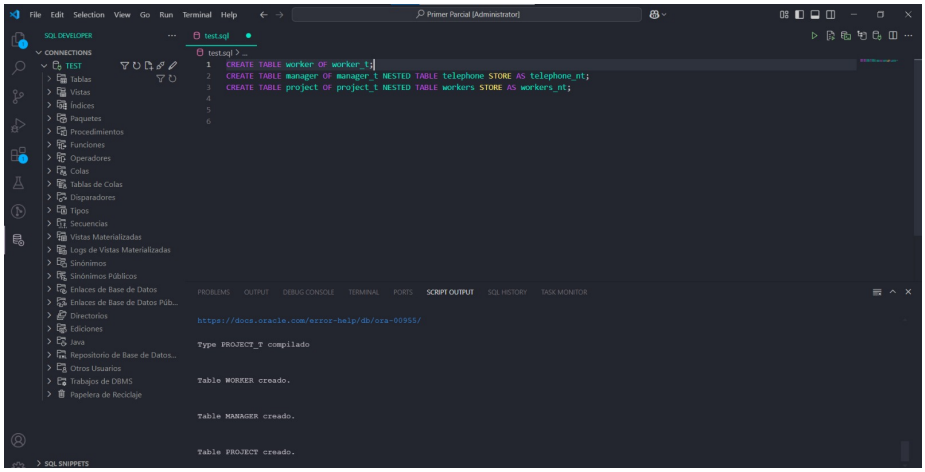

Figure 7: Create query

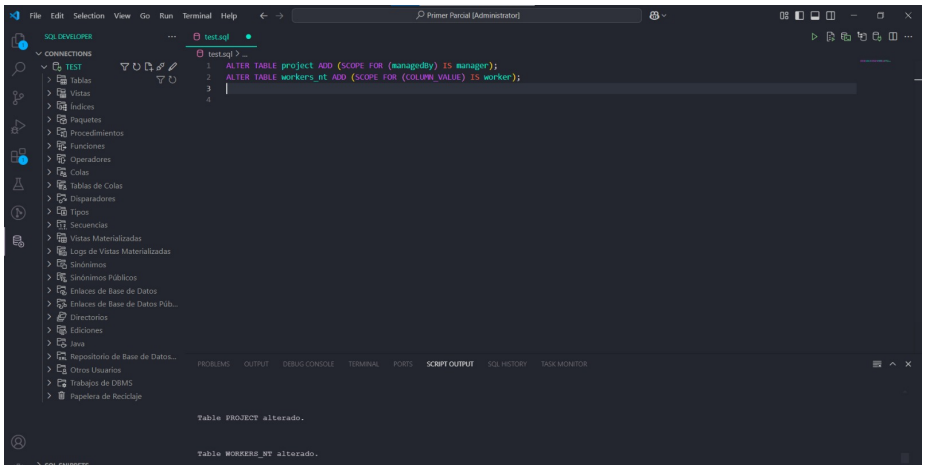

Figure 8: Alter query

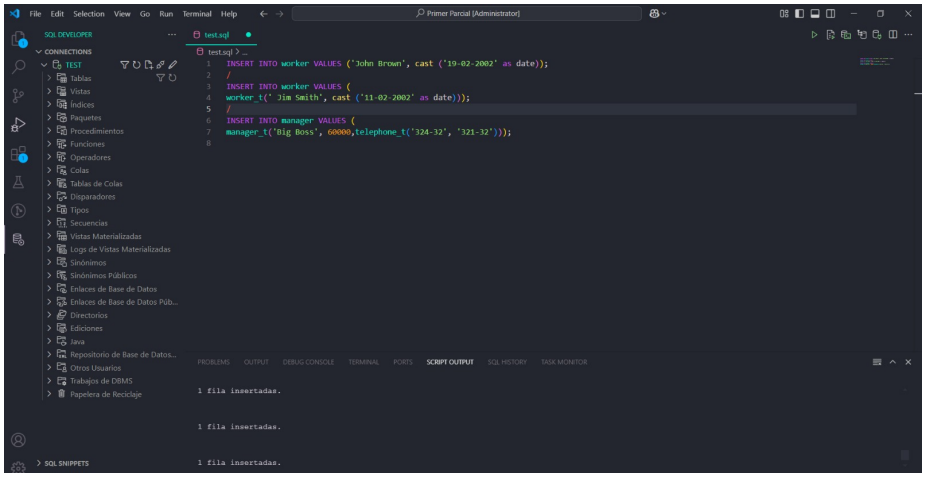

Figure 9: Insert query

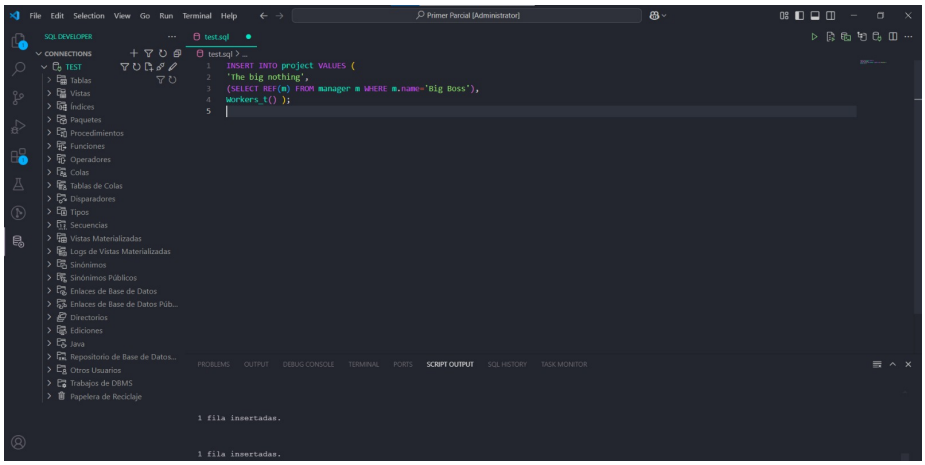

Figure 10: Insert query

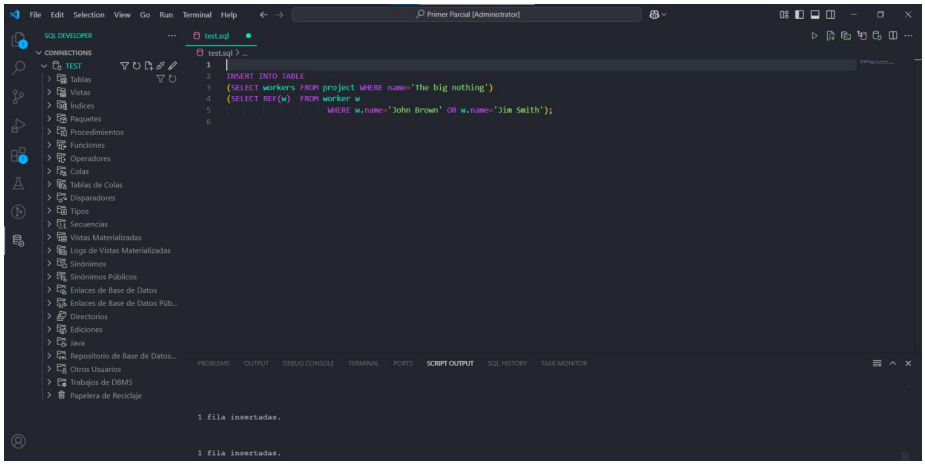

Figure 11: Insert query

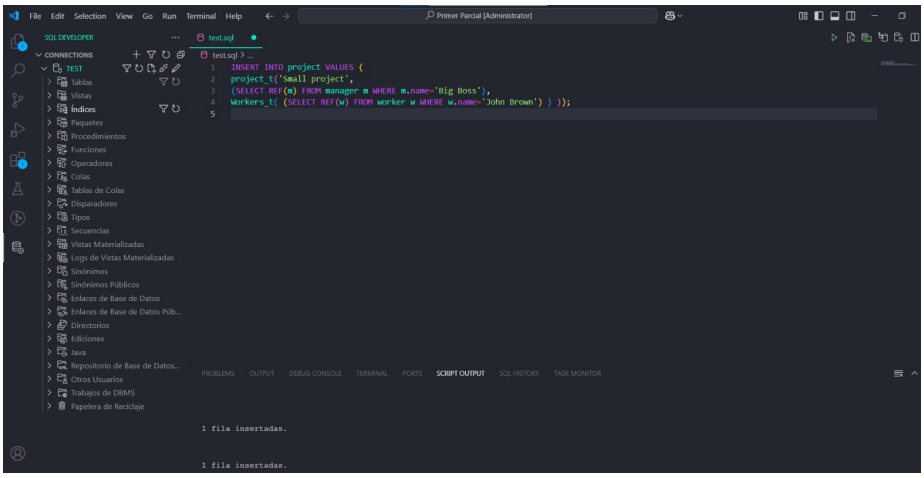

Figure 12: Insert query

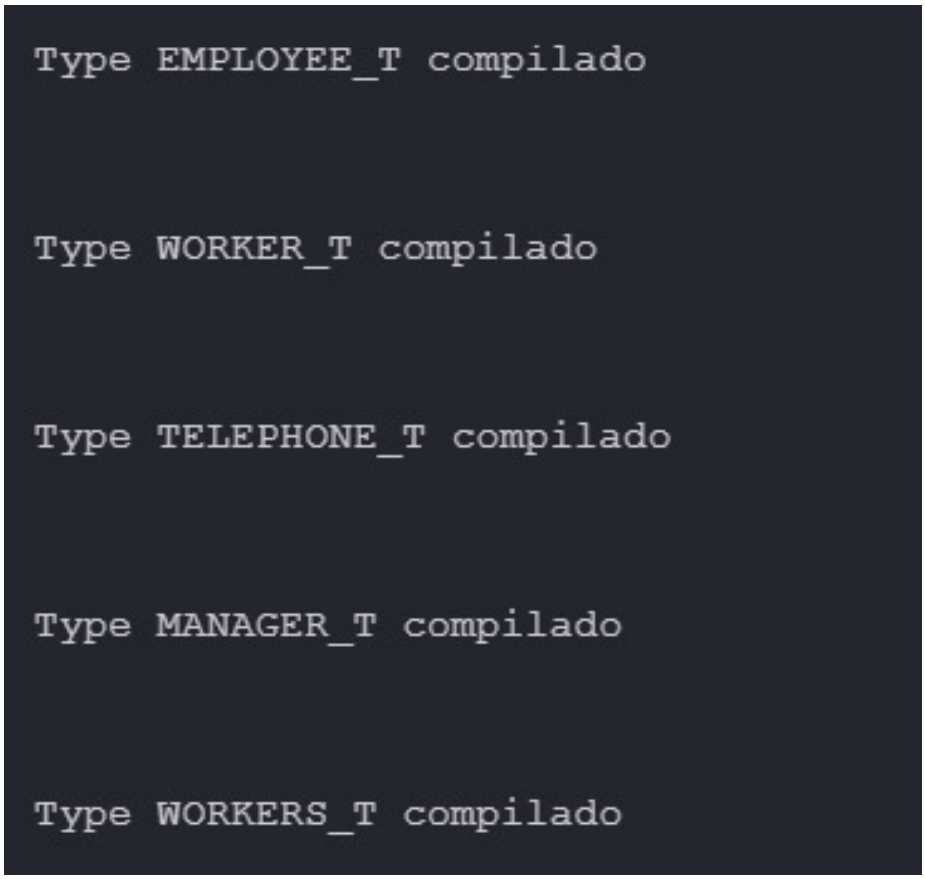

Figure 13: compilation successfully

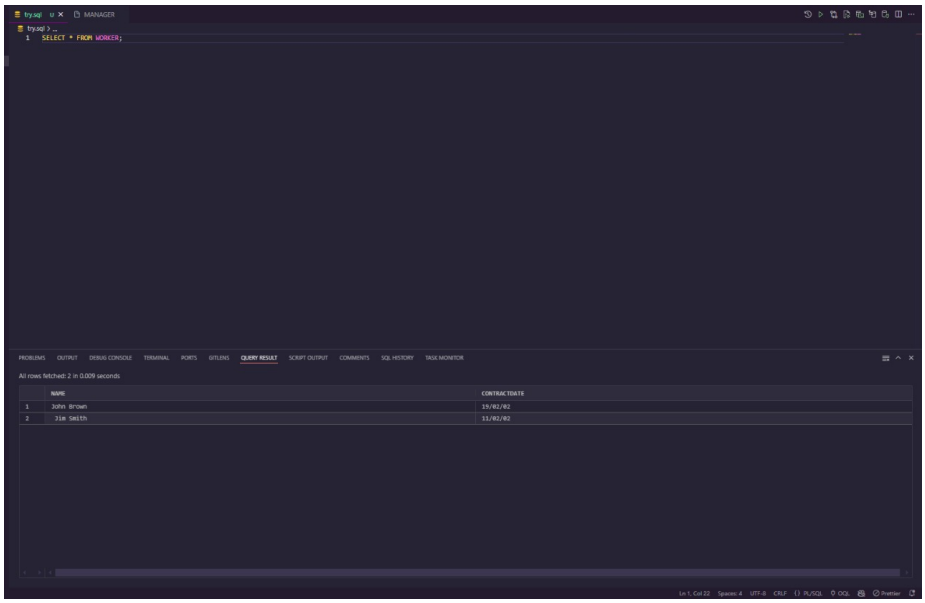

Figure 14: Select query

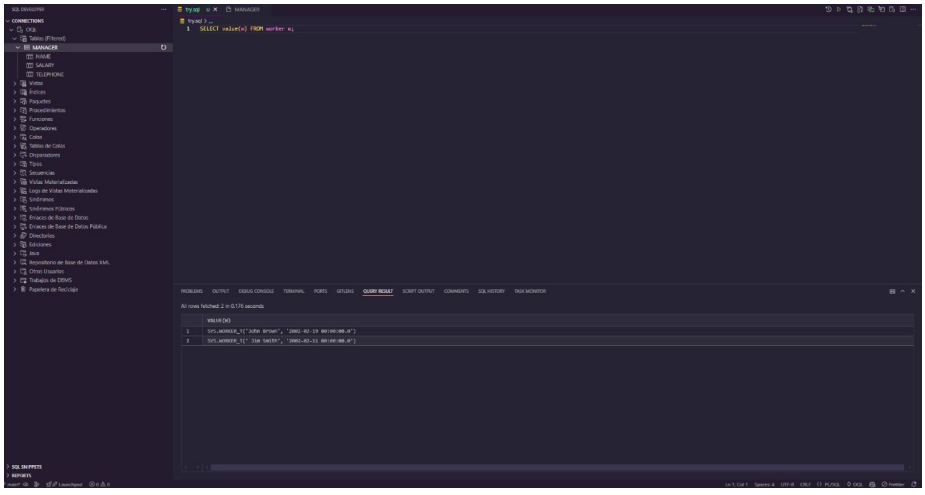

Figure 15: Select query

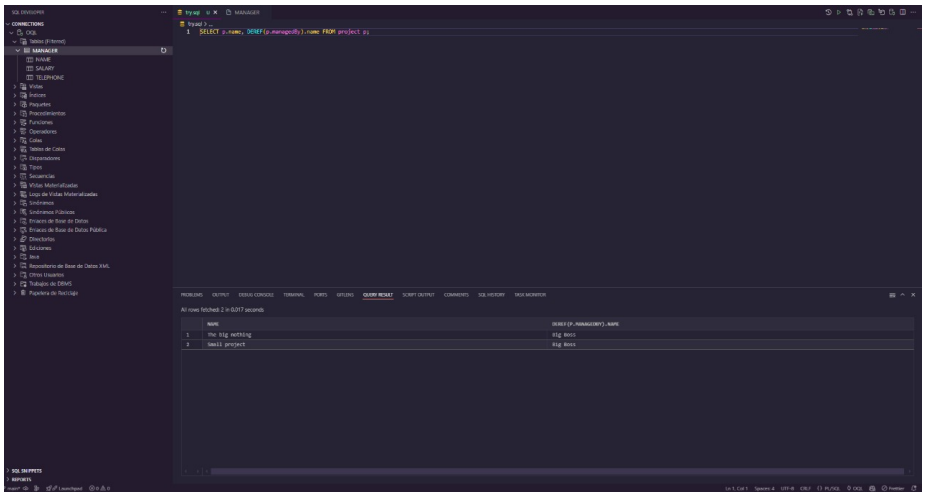

Figure 16: Select query

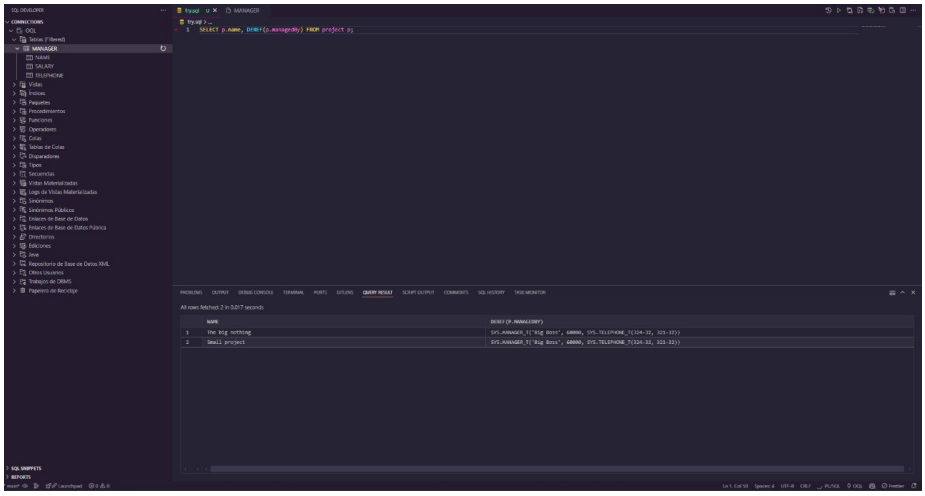

Figure 17: Select query

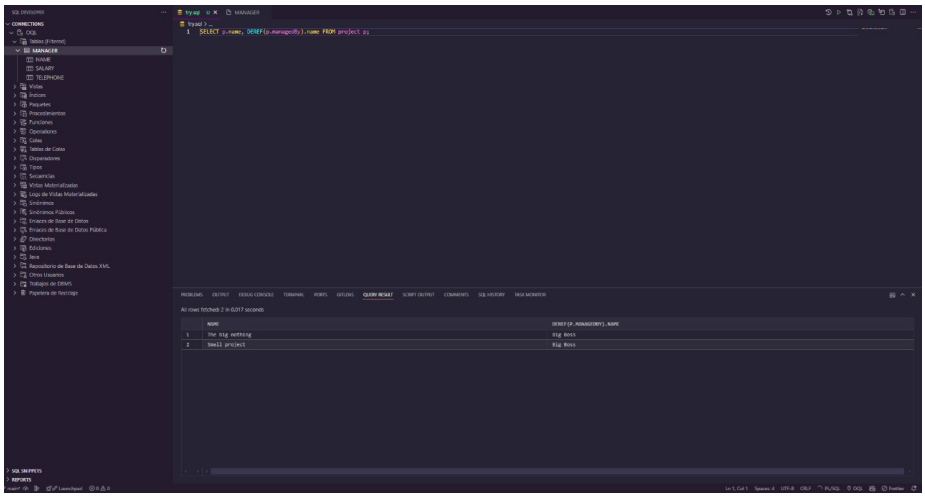

Figure 18: Select query

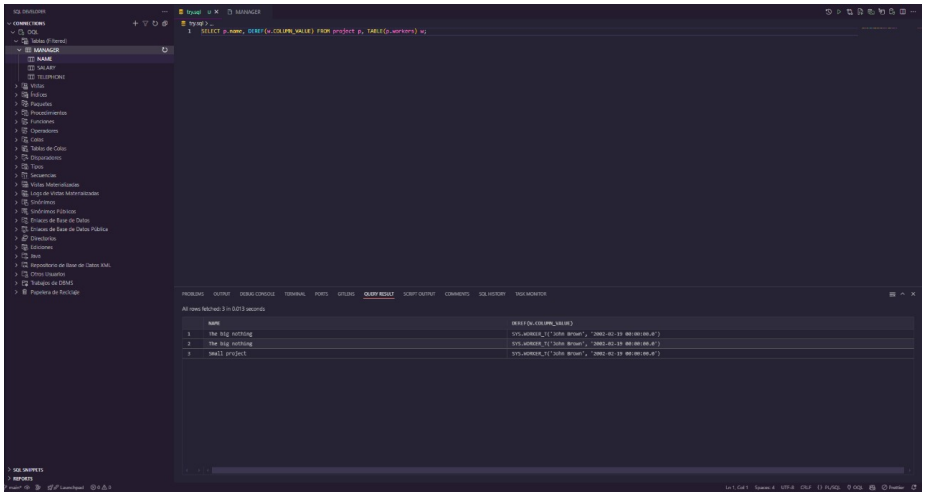

Figure 19: Select query

Summary of Results

After overcoming the initial challenges of setting up the environment and configuring the Oracle extension in VSCode, we successfully executed the OQL scripts. The object types, tables, and relationships were created as expected, and the database schema was validated through a series of queries. The successful com-

pilation and execution of the OQL scripts, as shown in the figures, demonstrate the effectiveness of OQL in managing object-relational databases. This exercise highlights the potential of OQL for modern, object-oriented applications, despite the initial difficulties encountered due to limited documentation.

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

To implement the object-relational database system on an **any Linux** system, we used **Docker** to containerize the Oracle Database Express Edition (XE). This approach simplifies the setup process and ensures consistency across different environments. Below are the steps to run the system and execute the OQL scripts:

1. Start the Oracle XE Container

First, start the Oracle XE container using the following command:

```
sudo docker start oracle-xe
```

If the container does not exist, create it using:

```
sudo docker run -d \
  -p 1521:1521 \
  --name oracle-xe \
  -e ORACLE_PASSWORD=<Password> \
  -v /home/erick/SQLSexyQueries/oql:/sql \
  gvenzl/oracle-xe
```

- Replace `<Password>` with your desired password for the Oracle database.
- The `-v` flag mounts the local directory `/home/erick/SQLSexyQueries/oql` to the `/sql` directory inside the container, allowing you to access your SQL scripts.

2. Connect to the Oracle Database

Once the container is running, connect to the Oracle database using `sqlplus`:

```
sqlplus sys/<Password>@//localhost:1521/XE as sysdba
```

- Replace `<Password>` with the password you set during container creation.

3. Execute the OQL Script

To execute the OQL script (`t2.sql`) stored in the mounted directory, use the following command:

```
sudo docker exec -i oracle-xe sqlplus sys/<Password> as sysdba @/sql/t2.sql
```

- This command runs the `t2.sql` script inside the Oracle XE container.

4. OQL Script Overview

The OQL script performs the following steps:

1. **Clean Up Existing Objects**: Drops any existing tables or types to avoid conflicts.
2. **Create Object Types**: Defines object types such as `employee_t`, `worker_t`, and `manager_t` with attributes and methods.
3. **Create Tables**: Creates tables with nested collections (e.g., `telephone_t` as a nested table of phone numbers).
4. **Insert Sample Data**: Populates the tables with sample data to demonstrate the functionality of the object-relational schema.
5. **Query Data**: Executes queries to retrieve and validate the data, ensuring the schema works as expected.

5. Example Queries

The script includes queries to:

- Retrieve all workers:

```
SELECT * FROM worker;
```

- Retrieve all managers:

```
SELECT * FROM manager;
```

- Retrieve all projects with their associated workers and managers:

```
SELECT p.name, DEREF(p.managedBy).name, DEREF(w.COLUMN_VALUE).name
FROM project p, TABLE(p.workers) w;
```

6. Benefits of Using Docker

- **Isolation**: Docker ensures that the Oracle database runs in an isolated environment, avoiding conflicts with other software on the system.
- **Portability**: The containerized setup can be easily replicated on other systems, ensuring consistency across development and production environments.
- **Ease of Setup**: Docker simplifies the installation and configuration of Oracle Database XE, especially on Linux distributions like Arch Linux where manual installation can be complex.

By following these steps, you can successfully run the object-relational database system on any Linux using Docker and Oracle XE, and execute the OQL scripts to validate the schema.

PROVE

The following SQL/OQL code was used to implement the object-relational database system:

```
-- Step 1: Clean up existing objects (if they exist)
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE project CASCADE CONSTRAINTS';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -942 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE manager CASCADE CONSTRAINTS';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -942 THEN
      RAISE;
    END IF;
END;
/
```

```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE worker CASCADE CONSTRAINTS';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -942 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE project_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE workers_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE manager_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE telephone_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/
BEGIN
```

```
  EXECUTE IMMEDIATE 'DROP TYPE worker_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE employee_t FORCE';
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE != -4043 THEN
      RAISE;
    END IF;
END;
/

-- Step 2: Create object types
CREATE TYPE employee_t AS OBJECT (name varchar2(100)) NOT FINAL;
/
CREATE TYPE worker_t UNDER employee_t (contractDate date);
/
CREATE TYPE telephone_t AS TABLE OF varchar2(30);
/
CREATE TYPE manager_t UNDER employee_t (salary integer, telephone telephone_t);
/
CREATE TYPE workers_t AS TABLE OF REF worker_t;
/
CREATE TYPE project_t AS OBJECT (
  name varchar2(100),
  managedBy REF manager_t,
  workers workers_t
);
/

-- Step 3: Create tables
CREATE TABLE worker OF worker_t;
CREATE TABLE manager OF manager_t NESTED TABLE telephone STORE AS telephone_nt;
CREATE TABLE project OF project_t NESTED TABLE workers STORE AS workers_nt;

-- Step 4: Add constraints
ALTER TABLE project ADD (SCOPE FOR (managedBy) IS manager);
ALTER TABLE workers_nt ADD (SCOPE FOR (COLUMN_VALUE) IS worker);

-- Step 5: Insert sample data
```

```
-- Insert workers
INSERT INTO worker VALUES ('John Brown', TO_DATE('19-FEB-2002', 'DD-MON-YYYY'));
INSERT INTO worker VALUES (worker_t('Jim Smith', TO_DATE('11-FEB-2002', 'DD-MON-YYYY')));

-- Insert manager
INSERT INTO manager VALUES (
  manager_t('Big Boss', 60000, telephone_t('324-32', '321-32'))
);

-- Insert project with no workers
INSERT INTO project VALUES (
  'The big nothing',
  (SELECT REF(m) FROM manager m WHERE m.name='Big Boss'),
  workers_t()
);

-- Assign workers to the project
INSERT INTO TABLE (
  SELECT workers FROM project WHERE name='The big nothing'
) (
  SELECT REF(w) FROM worker w WHERE w.name='John Brown' OR w.name='Jim Smith'
);

-- Insert another project with one worker
INSERT INTO project VALUES (
  project_t(
    'Small project',
    (SELECT REF(m) FROM manager m WHERE m.name='Big Boss'),
    workers_t((SELECT REF(w) FROM worker w WHERE w.name='John Brown'))
  )
);

-- Step 6: Verify data
-- Query all workers
SELECT * FROM worker;
-- Query all managers
SELECT * FROM manager;
-- Query all projects
SELECT p.name, DEREF(p.managedBy).name, DEREF(w.COLUMN_VALUE).name
FROM project p, TABLE(p.workers) w;
```

This implementation demonstrates the key features of object-relational databases using OQL:

1. Object type hierarchy with `employee_t` as the base type and `worker_t` and `manager_t` as derived types

2. Nested collections using `telephone_t` for manager phone numbers
3. Object references with `REF manager_t` and `workers_t`
4. Complex queries using `DEREF` to access referenced objects

The successful execution of these queries, as shown in the previous figures, validates the correct implementation of our object-relational schema.

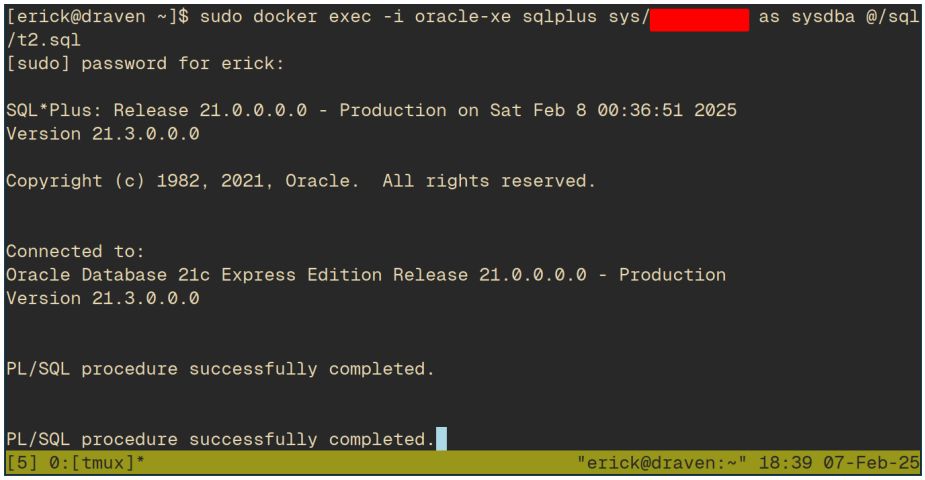

Figure 20: Successfull compilation

Figure 21: Successfull compilation

Figure 22: Successfull compilation

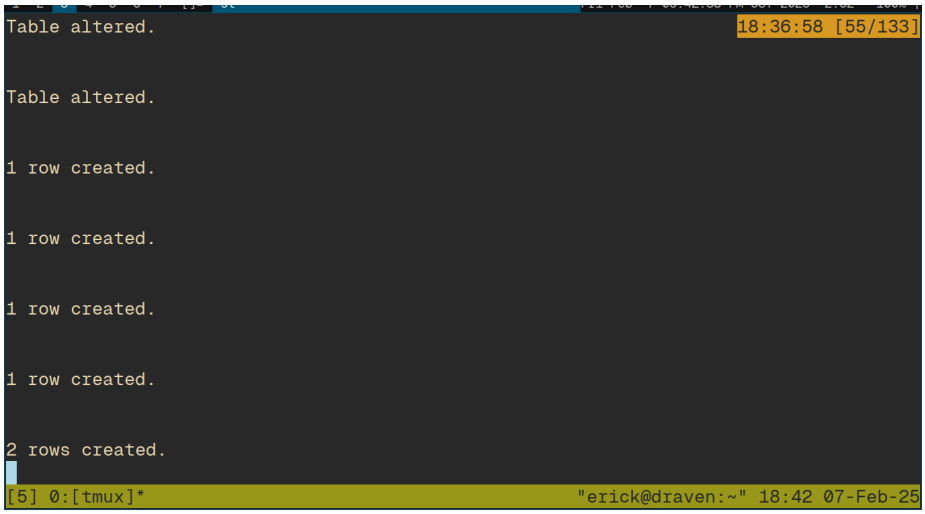

Figure 23: Successfull compilation

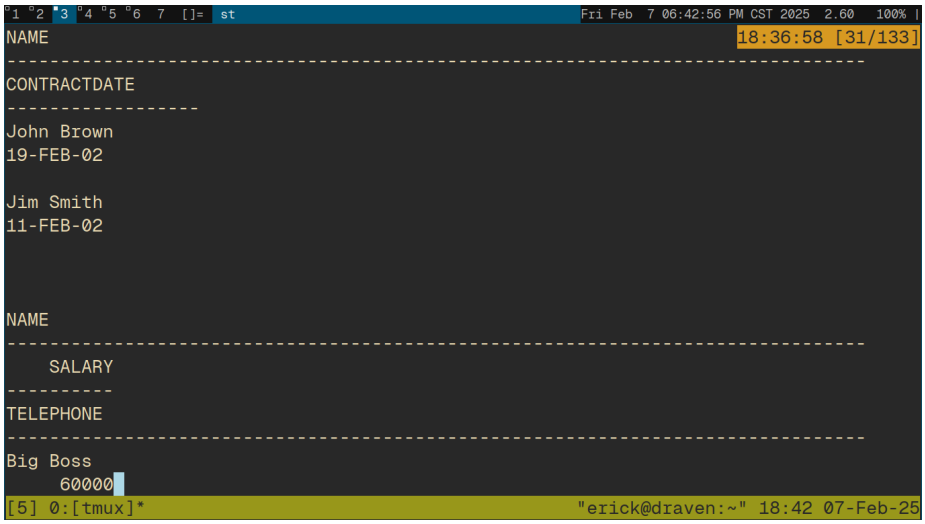

Figure 24: Successfull compilation

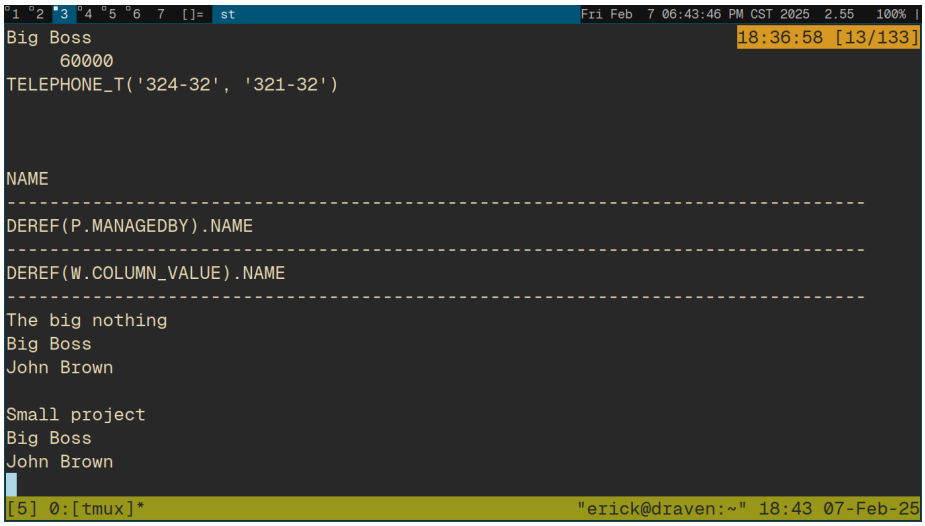

Figure 25: The queries

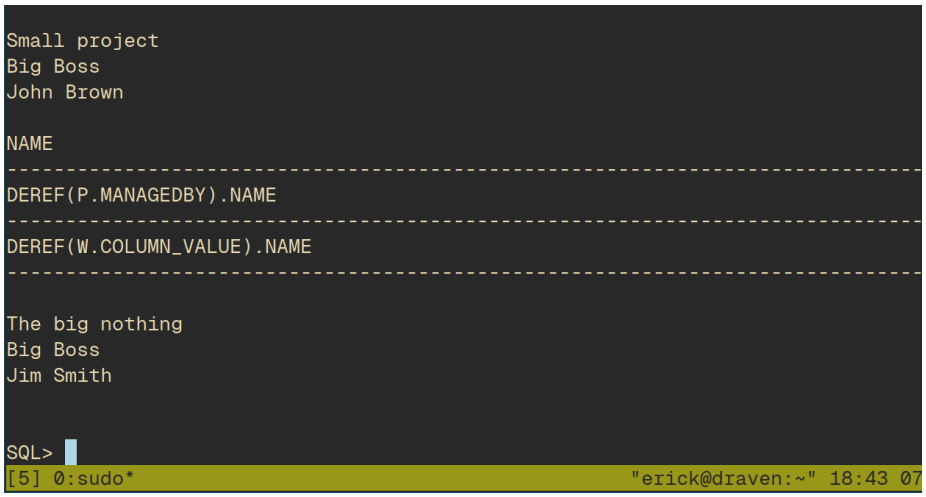

Figure 26: The queries

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