2º Trabalho Prático

Object Relational

Mestrado em Engenharia Informática e Computação

Tecnologias de Base de Dados

Grupo I:

Inês Oliveira - up202103343

Hugo Gomes - up202004343

2º Semestre

Ano Letivo 2023/2024

Meter índice (não sei meter)

Introdução

No âmbito da unidade curricular de Tecnologias de Base de Dados, integrada no mestrado em Engenharia Informática e Computação, foi-nos proposta a resolução do segundo trabalho prático com o intuito de explorar as possibilidades oferecidas pelo esquema objeto-relacional em comparação com o esquema relacional tradicional. Este trabalho envolve a criação de uma base de dados ilustrativa que utiliza tipos definidos pelo usuário, objetos que combinam estruturas de dados com funções para manipulá-los, herança e tabelas, referências de objetos e métodos de comparação e ordenação.

Este trabalho coloca em prática a teoria adquirida em aula e prepara-nos para enfrentar desafios no mundo da engenharia de software, onde a utilização de esquemas objeto-relacionais e a otimização de bases de dados desempenham um papel crucial na construção de sistemas robustos, eficientes e escaláveis.

Data

O estudo foca-se na modelagem de um sistema de Recursos Humanos para a gestão de uma empresa multinacional. A empresa está organizada em departamentos, cada um sediado em uma localidade pertencente a um país, dentro de uma região do mundo. Cada departamento possui um gerente, que é um funcionário. Os funcionários têm dados pessoais, data de contratação, são atribuídos a um departamento, possuem um gerente direto e executam um cargo com salário e percentagem de comissão determinados. Cada cargo está associado a uma faixa salarial (mínima e máxima). Há também um registo histórico de cargos para funcionários que já ocuparam mais de um cargo, contendo datas de início e fim, o cargo e o departamento onde foi desempenhado. O cargo

atual está em vigor desde o dia seguinte ao término do último registo histórico ou desde a data de contratação, se não houver registo na história para esse funcionário.

```
DROP TABLE Regions_Orig;
CREATE TABLE Regions_Orig AS
SELECT * FROM GTD10.Regions;

DROP TABLE Countries_Orig;
CREATE TABLE Countries_Orig AS
SELECT * FROM GTD10.Countries;

DROP TABLE Locations_Orig;
CREATE TABLE Locations_Orig AS
SELECT * FROM GTD10.Locations;

DROP TABLE Departments_Orig;
CREATE TABLE Departments_Orig AS
SELECT * FROM GTD10.Departments;

DROP TABLE Jobs_Orig;
CREATE TABLE Jobs_Orig AS
SELECT * FROM GTD10.Jobs;

DROP TABLE Job_History_Orig;
CREATE TABLE Job_History_Orig AS
SELECT * FROM GTD10.Job_History;

DROP TABLE Employees_Orig;
CREATE TABLE Employees_Orig AS
SELECT * FROM GTD10.Employees;

-----------------------------------
---- Dropping Tables and Types ----
-----------------------------------

-- Drop Tables
DROP TABLE Job_History FORCE;
DROP TABLE Employees FORCE;
DROP TABLE Jobs FORCE;
DROP TABLE Departments FORCE;
DROP TABLE Locations FORCE;
DROP TABLE Countries FORCE;
DROP TABLE Regions FORCE;

-- Drop Types
DROP TYPE Job_History_t FORCE;
DROP TYPE Employee_t FORCE;
DROP TYPE Job_t FORCE;
DROP TYPE Department_t FORCE;
DROP TYPE Location_t FORCE;
DROP TYPE Country_t FORCE;
DROP TYPE Region_t FORCE;
```

```
DROP TYPE Countries_ref_table_t force;
DROP TYPE Departments_ref_table_t force;
DROP TYPE Employees_ref_table_t force;
DROP TYPE Job_History_ref_table_t force;
DROP TYPE Locations_ref_table_t force;


-----------------------------------
--------- Declaring Types ---------
-----------------------------------


CREATE TYPE Employee_t;

--Region Type
CREATE TYPE Region_t AS OBJECT(
    region_id INT,
    region_name VARCHAR(100)
)NOT FINAL;

--Country Type
CREATE TYPE Country_t AS OBJECT(
    country_id VARCHAR(2),
    country_name VARCHAR(100),
    REGION REF Region_t
)NOT FINAL;

--Location Type
CREATE TYPE Location_t AS OBJECT(
    location_id INT,
    street_address VARCHAR(255),
    postal_code VARCHAR(20),
    city VARCHAR(100),
    state_province VARCHAR(100),
    COUNTRY REF Country_t
)NOT FINAL;

--Department Type
CREATE TYPE Department_t AS OBJECT(
    department_id INT,
    department_name VARCHAR(100),
    MANAGER REF Employee_t,
    LOCATION REF Location_t
)NOT FINAL;

--Job Type
CREATE TYPE Job_t AS OBJECT(
    job_id VARCHAR(20),
    job_title VARCHAR(100),
    min_salary NUMBER,
    max_salary NUMBER
)NOT FINAL;
```

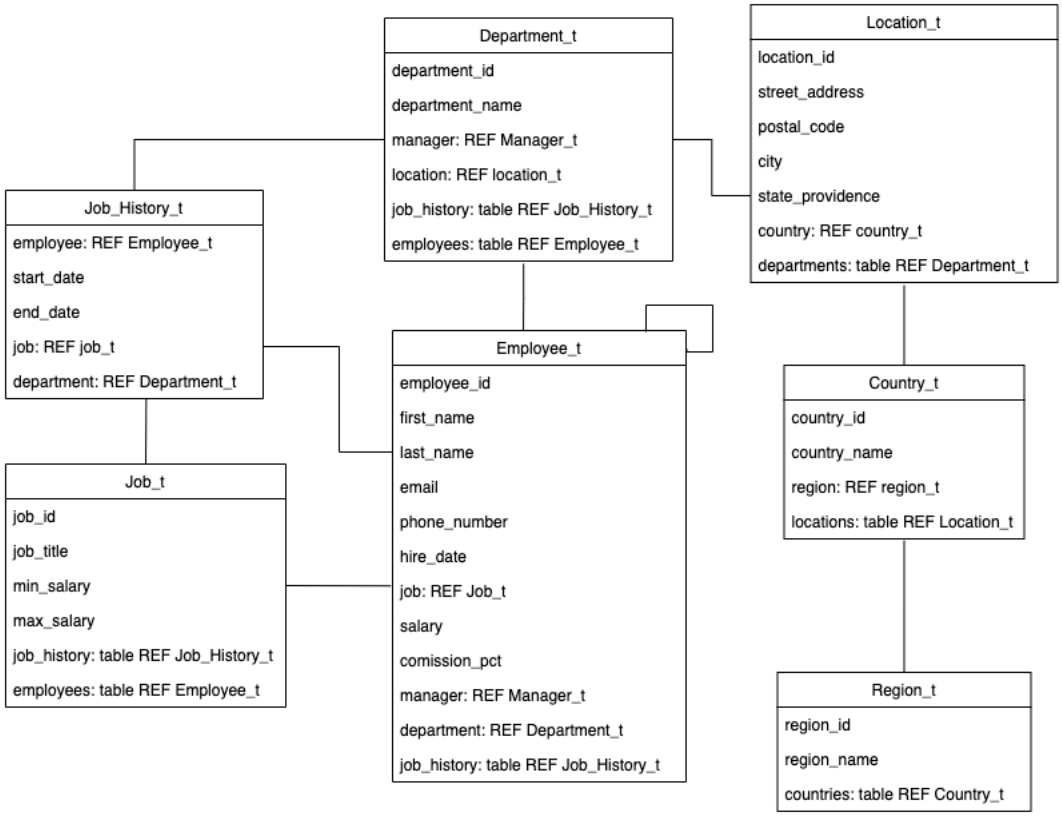
```sql
--Employee Type
CREATE TYPE Employee_t AS OBJECT (
    employee_id INT,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    hire_date DATE,
    JOB REF Job_t,
    salary NUMBER,
    commission_pct NUMBER,
    MANAGER REF Employee_t,
    DEPARTMENT REF Department_t
)NOT FINAL;


--Job History Type
CREATE TYPE Job_History_t AS OBJECT(
    EMPLOYEE REF Employee_t,
    start_date DATE,
    end_date DATE,
    JOB REF Job_t,
    DEPARTMENT REF Department_t
)NOT FINAL;

-------------------------------------
---- Declaring Reference Tables ----
-------------------------------------

--Countries Reference Table
CREATE TYPE Countries_ref_table_t AS TABLE OF REF Country_t;

--Locations Reference Table
CREATE TYPE Locations_ref_table_t AS TABLE OF REF Location_t;

--Departments Reference Table
CREATE TYPE Departments_ref_table_t AS TABLE OF REF Department_t;

--Employees Reference Table
CREATE TYPE Employees_ref_table_t AS TABLE OF REF Employee_t;

--Job History Reference Table
CREATE TYPE Job_History_ref_table_t AS TABLE OF REF Job_History_t;


-------------------------------------
--- Adding Nested Tables to Types ---
-------------------------------------

-- Countries to Region
ALTER TYPE Region_t
```

```
ADD ATTRIBUTE (Countries Countries_ref_table_t) CASCADE;

-- Locations to Country
ALTER TYPE Country_t
ADD ATTRIBUTE (Locations Locations_ref_table_t) CASCADE;

-- Departments to Location
ALTER TYPE Location_t
ADD ATTRIBUTE (Departments Departments_ref_table_t) CASCADE;

-- Job_History to Department
ALTER TYPE Department_t
ADD ATTRIBUTE (Job_History Job_History_ref_table_t) CASCADE;

--Employees to Department
ALTER TYPE Department_t
ADD ATTRIBUTE (Employees Employees_ref_table_t) CASCADE;

-- Job_History to Job
ALTER TYPE Job_t
ADD ATTRIBUTE (Job_History Job_History_ref_table_t) CASCADE;

-- Employees to Job
ALTER TYPE Job_t
ADD ATTRIBUTE (Employees Employees_ref_table_t) CASCADE;

-- Job_History to Employee
ALTER TYPE Employee_t
ADD ATTRIBUTE (Job_History Job_History_ref_table_t) CASCADE;

-- Departments to Employee
ALTER TYPE Employee_t
ADD ATTRIBUTE (Departments Departments_ref_table_t) CASCADE;

-- Employees to Employee
ALTER TYPE Employee_t
ADD ATTRIBUTE (Employees Employees_ref_table_t) CASCADE;




---------------------------------
-------- Creating Tables --------
---------------------------------

-- Regions Table
CREATE TABLE Regions of Region_t
    NESTED TABLE Countries STORE AS Countries_Region;

-- Countries Table
CREATE TABLE Countries of Country_t
    NESTED TABLE Locations STORE AS Locations_Country;
```

```
-- Locations Table
CREATE TABLE Locations of Location_t
    NESTED TABLE Departments STORE AS Departments_Location;

-- Departments Table
CREATE TABLE Departments of Department_t
    NESTED TABLE Job_History STORE AS Job_History_Department,
    NESTED TABLE Employees STORE AS Employees_Department;

-- Jobs Table
CREATE TABLE Jobs of Job_t
    NESTED TABLE Job_History STORE AS Job_History_Job,
    NESTED TABLE Employees STORE AS Employees_Job;

-- Employees Table
CREATE TABLE Employees of Employee_t
    NESTED TABLE Job_History STORE AS Job_History_Employee,
    NESTED TABLE Departments STORE AS Departments_Employee,
    NESTED TABLE Employees STORE AS Employees_Manager;

-- Job_History Table
CREATE TABLE Job_History of Job_History_t;
```

Desenvolvimento do projeto

1. Modelo Relacional



O diagrama apresentado representa um modelo de dados objeto-relacional para um sistema de Recursos Humanos de uma empresa multinacional. Este modelo é composto por várias tabelas interconectadas que refletem a estrutura organizacional e as relações entre os diferentes elementos da empresa.

No centro do modelo, temos a tabela Employee_t, que armazena informações detalhadas sobre os funcionários, incluindo dados pessoais, data de contratação, cargo atual, salário, percentual de comissão, gerente direto, departamento ao qual pertence, e um histórico de cargos. A tabela Job_t descreve os cargos disponíveis, incluindo títulos e faixas salariais. O histórico de cargos dos funcionários é mantido na tabela Job_History_t, que regista as mudanças de cargo e departamento ao longo do tempo.

Os departamentos são definidos na tabela Department_t, que inclui informações sobre o nome do departamento, gerente, localização, listas de funcionários e históricos de cargos. As localizações estão na tabela Location_t, que inclui detalhes como endereço, cidade, código postal, estado e país. Os países e regiões são armazenados nas tabelas Country_t e Region_t, respetivamente, com cada país associado a uma região.

2. Preencher o objeto-relacional

```
--Populating Regions Table
DELETE FROM Regions;
INSERT INTO Regions (region_id, region_name)
select r.region_id, r.region_name
from Regions_Orig r;

-- Populating Countries Table
DELETE FROM Countries;
INSERT INTO Countries (country_id, country_name, REGION)
SELECT c.country_id, c.country_name, REF(r)
FROM Countries_Orig c
JOIN Regions r ON c.region_id = r.region_id;

-- Populating Locations Table
DELETE FROM Locations;
INSERT INTO Locations (location_id, street_address, postal_code, city,
state_province, COUNTRY)
SELECT l.location_id, l.street_address, l.postal_code, l.city,
l.state_province, REF(c)
FROM Locations_Orig l
JOIN Countries c ON l.country_id = c.country_id;

-- Populating Departments Table
DELETE FROM Departments;
INSERT INTO Departments (department_id, department_name, MANAGER, LOCATION)
SELECT d.department_id,
       d.department_name,
       (SELECT REF(e) FROM Employees e WHERE e.employee_id = d.manager_id),
       (SELECT REF(l) FROM Locations l WHERE l.location_id = d.location_id)
FROM Departments_Orig d;

--Populating Jobs Table
DELETE FROM Jobs;
INSERT INTO Jobs (job_id, job_title, min_salary, max_salary)
SELECT j.job_id, j.job_title, j.min_salary, j.max_salary
FROM Jobs_Orig j;

-- Populating Employees Table
DELETE FROM Employees;
INSERT INTO Employees (employee_id, first_name, last_name, email,
phone_number, hire_date, JOB, salary, commission_pct, MANAGER, DEPARTMENT)
SELECT e.employee_id,
       e.first_name,
       e.last_name,
       e.email,
       e.phone_number,
```

```
       e.hire_date,
       (SELECT REF(J) FROM Jobs J WHERE J.job_id = e.job_id),
       e.salary,
       e.commission_pct,
       NULL,
       (SELECT REF(D) FROM Departments D WHERE D.department_id =
e.department_id)
FROM Employees_Orig e;

UPDATE Employees e
set e.Manager = (Select ref(Manager) From Employees Manager Where
Manager.Employee_Id = (Select manager_id from Employees_Orig Where
Employees_Orig.Employee_id = e.Employee_id));

-- Populating Job_History Table
delete from Job_History;
insert into Job_History (EMPLOYEE,START_DATE,END_DATE,JOB,DEPARTMENT)
select (select ref(e) From Employees e Where e.employee_id = jh.employee_id),
jh.START_DATE, jh.END_DATE, (select ref(j) From Jobs j where j.job_id =
jh.job_id), (select ref(d) From Departments d where
d.department_id=jh.department_id)
from Job_History_Orig jh;

--------------------------------
------- Nested Migration -------
--------------------------------

-- Populate Job_History nested table in Jobs
update Jobs j
set j.Job_History =
cast(multiset(select ref(jh) from Job_History jh where j.job_id =
jh.job.job_id) as Job_History_ref_table_t);

-- Populate Employees nested table in Jobs
update Jobs j
set j.Employees =
cast(multiset(select ref(e) from Employees e where j.job_id = e.job.job_id) as
Employees_ref_table_t);

-- Populate Employees nested table in Employees
update Employees e
set e.Employees =
cast(multiset(select ref(e2) from Employees e2 where e.employee_id =
e2.manager.employee_id) as Employees_ref_table_t);

--Populate Departments nested table in Employees
update Employees e
set e.Departments =
cast(multiset(select ref(d) from Departments d where e.employee_id =
d.manager.employee_id) as Departments_ref_table_t);

--Populate Job_History nested table in Departments
```

```
update Departments d
set d.Job_History =
cast(multiset(select ref(jh) from Job_History jh where d.department_id =
jh.department.department_id) as Job_History_ref_table_t);

--Populate Employees nested table in Departments
update Departments d
set d.Employees =
cast(multiset(select ref(e) from Employees e where d.department_id =
e.department.department_id) as Employees_ref_table_t);

--Populate Departments nested table in Locations
update Locations l
set l.Departments =
cast(multiset(select ref(d) from Departments d where l.location_id =
d.location.location_id) as Departments_ref_table_t);

--Populate Locations nested table in Countries
update Countries c
set c.Locations =
cast(multiset(select ref(l) from Locations l where c.country_id =
l.country.country_id) as Locations_ref_table_t);

--Populate Countries nested table in Regions
update Regions r
set r.Countries =
cast(multiset(select ref(c) from Countries c where r.region_id =
c.region.region_id) as Countries_ref_table_t);
```

3. Funções

3.1. Função GetMaxSalary (Employee_t)

Esta função devolve o salário mais alto dentro de um departamento especificado. Assim, é feita uma consulta à tabela de Empregados, selecionando o salário máximo onde o departamento do empregado corresponde à referência fornecida.

```
ALTER TYPE Employee_t
    ADD MEMBER FUNCTION GetMaxSalary(d REF Department_t) RETURN FLOAT CASCADE;

--function
MEMBER FUNCTION GetMaxSalary(d REF Department_t) RETURN FLOAT IS
        max_salary FLOAT;
    BEGIN
        SELECT MAX(e.salary) INTO max_salary FROM Employees e WHERE
e.department = d;
        RETURN max_salary;
    END GetMaxSalary;
```

3.2. Função GetNumberOfEmployees (Employee_t)

Esta função conta o número total de empregados dentro de um dado departamento. Realiza uma contagem na tabela de Empregados onde o departamento do empregado corresponde à referência fornecida.

```
ALTER TYPE Employee_t
    ADD MEMBER FUNCTION GetNumberOfEmployees(d REF Department_t) RETURN INT
CASCADE;

--function
MEMBER FUNCTION GetNumberOfEmployees(d REF Department_t) RETURN INT IS
        num_employees INT;
    BEGIN
        SELECT COUNT(*) INTO num_employees FROM Employees e WHERE e.department
= d;
        RETURN num_employees;
    END GetNumberOfEmployees;
```

3.3. Função GetAverageSalary (Employee_t)

Esta função calcula o salário médio dos empregados dentro de um departamento específico. É calculada a média dos salários da tabela de Empregados onde o departamento do empregado corresponde à referência fornecida.

```
ALTER TYPE Employee_t
    ADD MEMBER FUNCTION GetAverageSalary(d REF Department_t) RETURN NUMBER
CASCADE;

--function
MEMBER FUNCTION GetAverageSalary(d REF Department_t) RETURN NUMBER IS
        avg_salary NUMBER;
    BEGIN
        SELECT AVG(salary) INTO avg_salary FROM Employees e WHERE e.department
= d;
        RETURN avg_salary;
    END GetAverageSalary;
```

3.4. Função GetAverageSalary com Filtro de Função (Employee_t)

Esta função calcula o salário médio dos empregados dentro de um departamento e função específicos. Filtra os empregados na tabela de Empregados por ambas as referências de departamento e função antes de fazer a média dos salários.

```
ALTER TYPE Employee_t
    ADD MEMBER FUNCTION GetAverageSalary(d REF Department_t, j REF Job_t) RETURN
NUMBER CASCADE;
```

```
--function
MEMBER FUNCTION GetAverageSalary(d REF Department_t, j REF Job_t) RETURN
NUMBER IS
        avg_salary NUMBER;
    BEGIN
        SELECT AVG(e.salary) INTO avg_salary FROM Employees e WHERE
e.department = d AND e.job = j;
        RETURN avg_salary;
    END GetAverageSalary;
```

3.5 Função GetNumberOfEmployees com Filtro de Função (Employee_t)

Esta função conta o número de empregados dentro de um departamento e função especificados. Realiza uma contagem na tabela de Empregados onde tanto o departamento quanto a função do empregado correspondem às referências fornecidas.

```
ALTER TYPE Employee_t
    ADD MEMBER FUNCTION GetNumberOfEmployees(d REF Department_t, j REF Job_t)
RETURN INT CASCADE;

--function
MEMBER FUNCTION GetNumberOfEmployees(d REF Department_t, j REF Job_t) RETURN
INT IS
        num_employees INT;
    BEGIN
        SELECT COUNT(*) INTO num_employees FROM Employees e WHERE e.department
= d AND e.job = j;
        RETURN num_employees;
    END GetNumberOfEmployees;
```

3.6 Função GetNumberOfEmployees (Country_t)

Esta função calcula o número total de empregados num país. Realiza uma consulta que navega por países, localizações, departamentos e empregados, contando todos os empregados que correspondem à referência do país fornecida.

```
ALTER TYPE Country_t
     ADD MEMBER FUNCTION GetNumberOfEmployees(c REF Country_t) RETURN INT
CASCADE;

--function
MEMBER FUNCTION GetNumberOfEmployees(c REF Country_t) RETURN INT IS
        num_employees INT;
    BEGIN
```

```
        SELECT COUNT(*) INTO num_employees FROM Countries c1,
table(c1.Locations) l, table(value(l).departments) d,
table(value(d).Employees) e
        WHERE c = ref(c1);
        RETURN num_employees;
    END GetNumberOfEmployees;
```

3.7 Função GetAverageSalary (Country_t)

Esta função calcula o salário médio dos empregados em todos os departamentos e localizações dentro de um país. Realiza uma consulta que navega por países, localizações, departamentos e empregados para calcular o salário médio.

```
ALTER TYPE Country_t
    ADD MEMBER FUNCTION GetAverageSalary(c REF Country_t) RETURN NUMBER CASCADE;

--function
MEMBER FUNCTION GetAverageSalary(c REF Country_t) RETURN NUMBER IS
        avg_salary NUMBER;
    BEGIN
        SELECT nvl(AVG(value(e).salary),0) INTO avg_salary FROM Countries c1,
table(c1.Locations) l, table(value(l).departments) d,
table(value(d).Employees) e
        WHERE c = ref(c1);
        RETURN avg_salary;
    END GetAverageSalary;
```

4. Queries

4.1. Calcular o Número Total de Empregados por Departamento

```
-- Query 1
SELECT d.department_name, e.GetNumberOfEmployees(REF(d)) AS total_employees
FROM Departments d, Employees e
WHERE e.department = REF(d);
```

Esta consulta utiliza uma função de membro *GetNumberOfEmployees* definida no tipo *Employee_t*. Retorna o número total de empregados para cada departamento. A função é chamada passando uma referência ao departamento (REF(d)) que é associado a cada empregado. O resultado inclui o nome do departamento e o número total de empregados.

	DEPARTMENT_NAME	TOTAL_EMPLOYEES
1	Purchasing	6
2	Shipping	45
3	Accounting	2
4	IT	5
5	Finance	6
6	Marketing	2
7	Administration	1
8	Executive	3
9	Public Relations	1
10	Sales	34
11	Human Resources	1

4.2. Número de empregados por título de emprego em cada departamento

```
-- Query 2
SELECT DISTINCT e.department.department_name as department_name,
e.job.job_title as Job_Title, e.GetNumberOfEmployees(e.department,e.job) as
NumberEmployees
FROM Employees e;
```

Esta consulta lista cada departamento junto com os títulos de trabalho dentro desses departamentos e quantos empregados existem para cada título de emprego. Usa a função *GetNumberOfEmployees* com dois parâmetros (departamento e trabalho) para o calculo.

	DEPARTMENT_NA...	JOB_TITLE	NUMBEREMPLOYEES
1	Accounting	Accounting Manager	1
2	Accounting	Public Accountant	1
3	Administration	Administration Assistant	1
4	Executive	Administration Vice President	2
5	Executive	President	1
6	Finance	Accountant	5
7	Finance	Finance Manager	1
8	Human Resources	Human Resources Representative	1
9	IT	Programmer	5
10	Marketing	Marketing Representative	1
11	Marketing	Marketing Manager	1
12	Public Relations	Public Relations Representative	1
13	Purchasing	Purchasing Manager	1
14	Purchasing	Purchasing Clerk	5
15	Sales	Sales Representative	29
16	Sales	Sales Manager	5
17	Shipping	Stock Clerk	20
18	Shipping	Stock Manager	5
19	Shipping	Shipping Clerk	20
20	(null)	Sales Representative	0

4.3. Indicar o emprego mais bem pago de cada departamento

```sql
-- Query 3
SELECT e.employee_id, e.first_name, e.last_name, e.department.department_name
as Department, e.salary
FROM Employees e
WHERE e.salary = e.GetMaxSalary(e.department);
```

Esta consulta identifica o empregado mais bem pago em cada departamento utilizando a função *GetMaxSalary*, que retorna o maior salário encontrado em um departamento específico. Os resultados incluem a identificação do empregado, nome, sobrenome, departamento e salário.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY
1	201	Michael	Hartstein	Marketing	13000
2	203	Susan	Mavris	Human Resources	6500
3	204	Hermann	Baer	Public Relations	10000
4	205	Shelley	Higgins	Accounting	12000
5	100	Steven	King	Executive	24000
6	108	Nancy	Greenberg	Finance	12000
7	121	Adam	Fripp	Shipping	8200
8	145	John	Russell	Sales	14000
9	200	Jennifer	Whalen	Administration	4400
10	103	Alexander	Hunold	IT	9000
11	114	Den	Raphaely	Purchasing	11000

4.4. Verificar se existem lacunas temporais no histórico de emprego

```sql
-- Query 4
SELECT R.employee_id
FROM (
    SELECT E.employee_id AS employee_id
    FROM employees E
    JOIN job_history JH1 ON JH1.employee = REF(E)
    JOIN job_history JH2 ON JH2.employee = REF(E)
                        AND JH2.start_date = JH1.end_date + INTERVAL '1' DAY
    UNION
    SELECT employee_id
    FROM job_history JH
    JOIN employees E ON JH.employee = REF(E)
    GROUP BY employee_id
    HAVING COUNT(*) = 1
) R;
```

Esta consulta visa identificar lacunas no histórico de emprego de cada empregado. A consulta junta a tabela de histórico de emprego consigo mesma para encontrar registos consecutivos onde a data de início de um emprego segue diretamente o dia após o término de outro. Além disso, procura empregados com apenas uma entrada de histórico, indicando que pode haver lacunas não registadas.

	EMPLOYEE_ID
1	101
2	102
3	114
4	122
5	176
6	201

4.5. Comparar o salário médio por país

```
-- Query 5
SELECT c.country_id, c.country_name, round(c.GetAverageSalary(REF(C)))
FROM Countries c;
```

Esta consulta calcula o salário médio por país utilizando a função *GetAverageSalary* do tipo *Country_t*.

	COUNTRY_ID	COUNTRY_NAME	ROUND(C.GETAVERAGESALARY(REF(C)))
1	AR	Argentina	0
2	AU	Australia	0
3	BE	Belgium	0
4	BR	Brazil	0
5	CA	Canada	9500
6	CH	Switzerland	0
7	CN	China	0
8	DE	Germany	10000
9	DK	Denmark	0
10	EG	Egypt	0
11	FR	France	0
12	HK	HongKong	0
13	IL	Israel	0
14	IN	India	0
15	IT	Italy	0
16	JP	Japan	0
17	KW	Kuwait	0
18	MX	Mexico	0
19	NG	Nigeria	0
20	NL	Netherlands	0
21	SG	Singapore	0
22	UK	United Kingdom	8886
23	US	United States of America	5065
24	ZM	Zambia	0
25	ZW	Zimbabwe	0

4.6. Consulta utilizando extensões OR para calcular o salário médio por departamento em Seattle

```
-- Query 6
SELECT DISTINCT l.city,
       value(d).department_name,
       round(e.getAverageSalary(d.column_value),2) as Average_Salary
FROM Locations l,
```

```
table (l.departments) D,
     Employees e
WHERE l.city = 'Seattle' and
e.department = D.column_value
ORDER BY l.city, value(D).department_name;
```

Esta consulta exemplifica o uso de funções e extensões OR para calcular o salário médio em departamentos específicos dentro da cidade de Seattle. O resultado apresenta o nome de cada departamento em Seattle e o respetivo salário médio.

	CITY	VALUE(D).DEPARTMENT_NAME	AVERAGE_SALARY
1	Seattle	Accounting	10150
2	Seattle	Administration	4400
3	Seattle	Executive	19333,33
4	Seattle	Finance	8600
5	Seattle	Purchasing	4150

Conclusão

Neste projeto, exploramos as capacidades das funções membro e de consultas complexas num sistema de base de dados orientado a objetos. Investigamos a implementação e a aplicação de funções específicas que manipulam e extraem dados relacionados a empregados e departamentos, abordando de forma detalhada como cada função e consulta contribui para o entendimento operacional e estratégico da organização.

Uma conquista notável foi o aprimoramento da nossa competência em formular consultas que não apenas respondem a perguntas de negócios específicas, mas também maximizam a eficiência da interação com a base de dados. Desenvolvemos a habilidade de calcular dinamicamente métricas essenciais, como salários máximos, médios e a contagem de empregados em várias dimensões organizacionais, o que nos permitiu extrair mais facilmente valores sobre a estrutura da empresa.

Além disso, exploramos as funcionalidades OR e demonstramos o quão úteis elas podem ser no manuseio de bases de dados complexas. Essas funcionalidades permitem uma modelagem e uma consulta de dados mais intuitiva e alinhada com os objetos do mundo real, facilitando a implementação de lógicas de negócios complexas e a integração de dados estruturados de maneira eficiente.