2º Trabalho Prático

***Object Relational***

Shape

Description automatically generated with medium confidence

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**](#_heading=h.gjdgxs) **3**

[**Data**](#_heading=h.30j0zll) **3**

[**Desenvolvimento do projeto**](#_heading=) **13**

[1. Modelo Relacional](#_heading=) 13

[2. Preencher o objeto-relacional](#_heading=) 14

[3. Funções](#_heading=) 16

[3.1. Função GetMaxSalary (Employee\_t)](#_heading=) 16

[3.2. Função GetNumberOfEmployees (Employee\_t)](#_heading=) 17

[3.3. Função GetAverageSalary (Employee\_t)](#_heading=) 17

[3.4. Função GetAverageSalary com Filtro de Função (Employee\_t)](#_heading=) 18

[3.5 Função GetNumberOfEmployees com Filtro de Função (Employee\_t)](#_heading=) 18

[3.6 Função GetNumberOfEmployees (Country\_t)](#_heading=) 19

[3.7 Função GetAverageSalary (Country\_t)](#_heading=) 19

[4. Queries](#_heading=) 20

[4.1. Calcular o Número Total de Empregados por Departamento](#_heading=) 20

[4.2. Número de empregados por título de emprego em cada departamento](#_heading=) 20

[4.3. Indicar o emprego mais bem pago de cada departamento](#_heading=) 21

[4.4. Verificar se existem lacunas temporais no histórico de emprego](#_heading=) 21

[4.5. Comparar o salário médio por país](#_heading=) 22

[4.6. Consulta utilizando extensões OR para calcular o salário médio por departamento em Seattle](#_heading=) 23

[**Conclusão**](#_heading=h.3znysh7) **23**

# 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;

--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

## Modelo RelacionalA diagram of a company Description automatically generated

*Figura SEQ Figura \\* ARABIC 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.

## 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);

## 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;

## Queries

### Calcular o Número Total de Empregados por Departamento

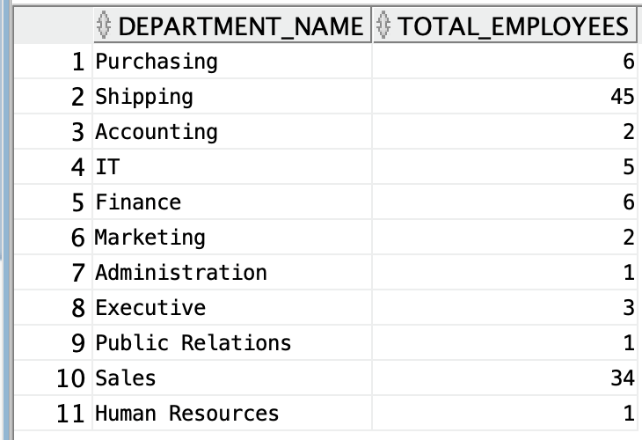
-- Query 1

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.



### 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.



### Indicar o emprego mais bem pago de cada departamento

-- 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.



### Verificar se existem lacunas temporais no histórico de emprego

-- 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

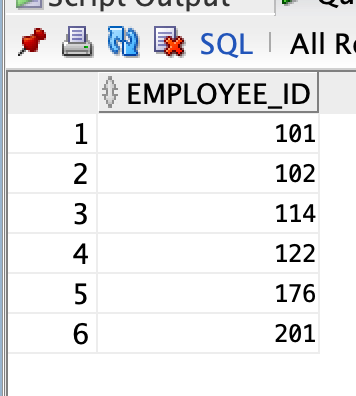
    JOIN employees E ON JH.employee = REF(E)

    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.



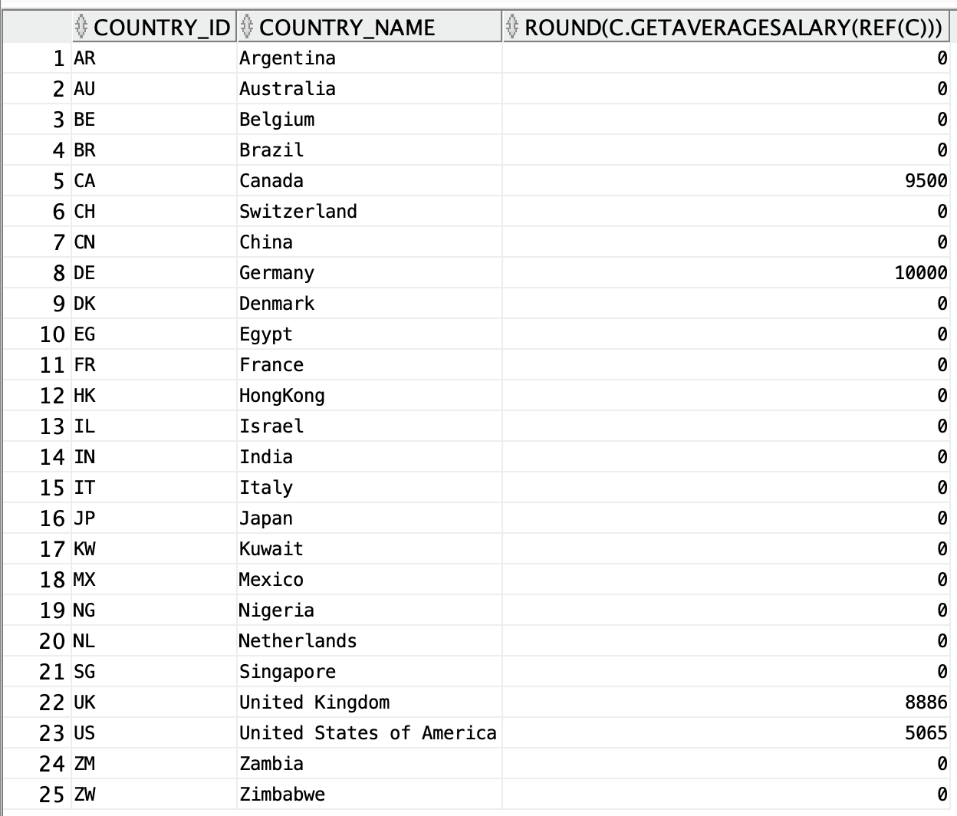
### Comparar o salário médio por país

-- 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*.



### 

### 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,

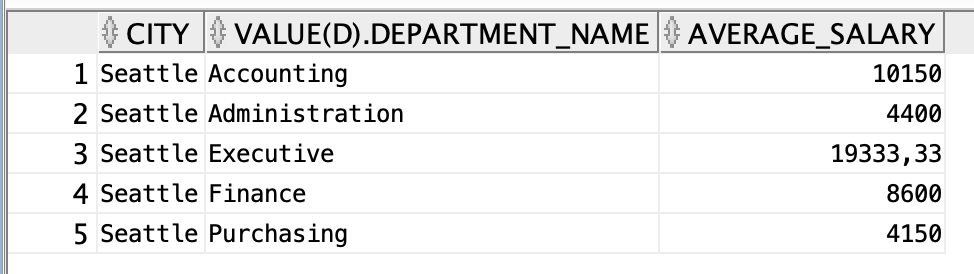
     Employees e

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.



# 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.