TBDA - Project 2

Human resources management system

Group members:
Juliane Marubayashi (up201800175)
Ricardo Carvalho (up201806791)
Rodrigo Reis (up201806534)

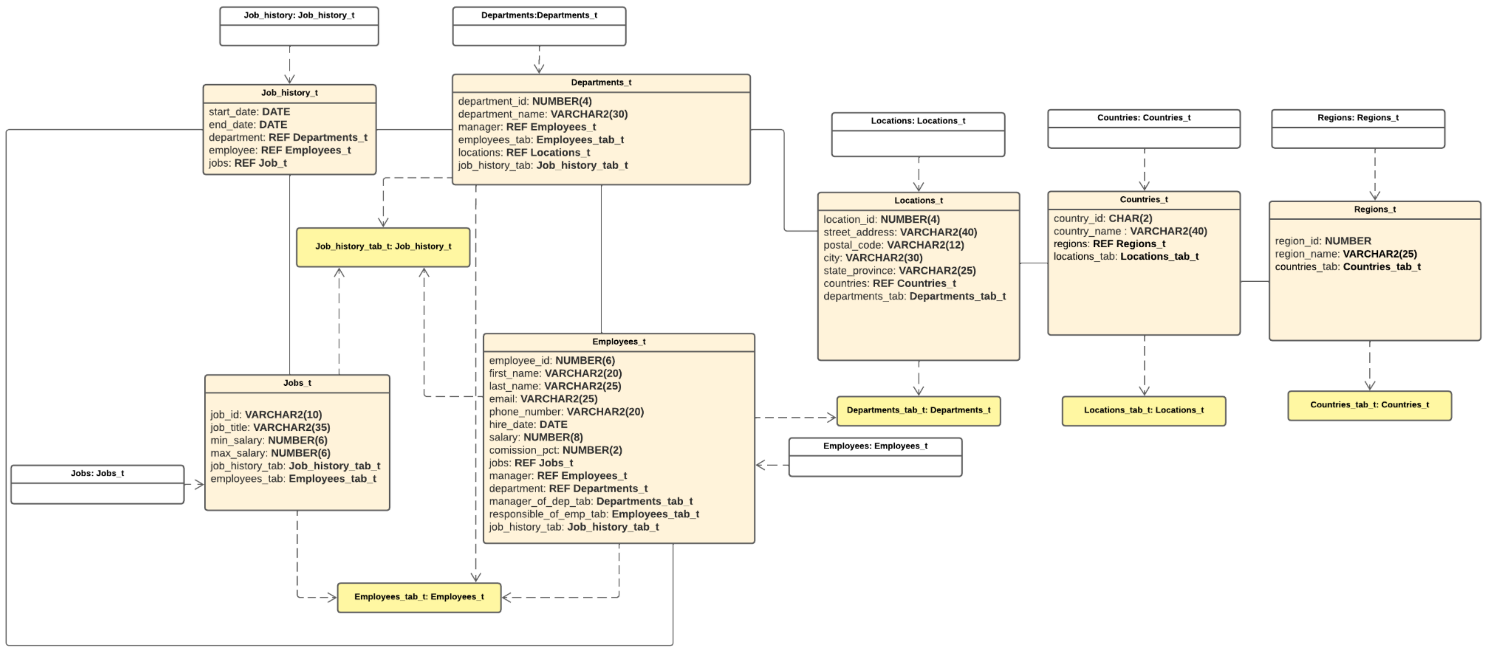
Faculdade de Engenharia do Porto
May, 13th of 2022, Porto

Index

Summary

This is a model relative to the HR system about the management of human resources in a multinational company. The goal of the project is to design an object-relational data model, populate it and execute some queries on top of the object relational database. To accomplish that goal, we used user defined types, nested tables and functions to potentialize to the maximum the use of an object relational database.

1) Relational Schema



Job_history: Job_history_t
Departments:Departments_t
Job_history_t
start_date: DATE
end_date: DATE
department: REF Departments_t
employee: REF Employees_t
jobs: REF Job_t
Departments_t
department_id: NUMBER(4)
department_name: VARCHAR2(30)
manager: REF Employees_t
employees_tab: Employees_tab_t
locations: REF Locations_t
job_history_tab: Job_history_tab_t
Job_history_tab_t: Job_history_t
Jobs_t
job_id: VARCHAR2(10)
job_title: VARCHAR2(35)
min_salary: NUMBER(6)
max_salary: NUMBER(6)
job_history_tab: Job_history_tab_t
employees_tab: Employees_tab_t
Jobs: Jobs_t
Employees_t
employee_id: NUMBER(6)
first_name: VARCHAR2(20)
last_name: VARCHAR2(25)
email: VARCHAR2(25)
phone_number: VARCHAR2(20)
hire_date: DATE
salary: NUMBER(8)
comission_pct: NUMBER(2)
jobs: REF Jobs_t
manager: REF Employees_t
department: REF Departments_t
manager_of_dep_tab: Departments_tab_t
responsible_of_emp_tab: Employees_tab_t
job_history_tab: Job_history_tab_t
Employees_tab_t: Employees_t
Locations: Locations_t
Countries: Countries_t
Regions: Regions_t
Locations_t
location_id: NUMBER(4)
street_address: VARCHAR2(40)
postal_code: VARCHAR2(12)
city: VARCHAR2(30)
state_province: VARCHAR2(25)
countries: REF Countries_t
departments_tab: Departments_tab_t
Countries_t
country_id: CHAR(2)
country_name : VARCHAR2(40)
regions: REF Regions_t
locations_tab: Locations_tab_t
Regions_t
region_id: NUMBER
region_name: VARCHAR2(25)
countries_tab: Countries_tab_t
Departments_tab_t: Departments_t
Locations_tab_t: Locations_t
Countries_tab_t: Countries_t
Employees: Employees_t

2) Creation Script

```
create or replace type regions_t;
create or replace type countries_t;
create or replace type locations_t;
create or replace type employees_t;
create or replace type jobs_t;
create or replace type job_history_t;
create or replace type departments_t;

create or replace type regions_t as object (
    region_id number,
    region_name varchar2(25)
);

create or replace type countries_t as object (
    country_id char(2),
    country_name varchar2(25),
    region REF regions_t
);

create or replace type locations_t as object (
    location_id number(4),
    street_addreess varchar2(40),
    postal_code varchar2(12),
    city varchar2(30),
    state_province varchar2(25),
    countries REF countries_t
);

create or replace type employees_t as object (
    employee_id NUMBER(6),
    first_name VARCHAR2(20),
    last_name VARCHAR2(25),
    email VARCHAR2(25),
    phone_number VARCHAR2(20),
    hire_date DATE,
    salary NUMBER(8),
    commission_pct NUMBER(2),
    jobs REF jobs_t,
    manager REF employees_t,
    department REF departments_t
);

create or replace type jobs_t as object (
    job_id VARCHAR2(10),
    job_title VARCHAR2(35),
    min_salary NUMBER(6),
    max_salary NUMBER(6)
);
```

```sql
create or replace type job_history_t as object (
    start_date DATE,
    end_date DATE,
    department REF departments_t,
    employee REF employees_t,
    jobs REF jobs_t
);


create or replace type departments_t as object (
    department_id NUMBER(4),
    department_name VARCHAR2(30),
    locations REF locations_t,
    manager REF employees_t
);


-- Creating nested tables.
create or replace type countries_tab_t as table of ref countries_t;
create or replace type locations_tab_t as table of ref locations_t;
create or replace type departments_tab_t as table of ref departments_t;
create or replace type job_history_tab_t as table of ref job_history_t;
create or replace type employees_tab_t as table of ref employees_t;

-- Add the nested tables.
alter type regions_t add attribute(countries_tab countries_tab_t) cascade;
alter type countries_t add attribute(locations_tab locations_tab_t) cascade;
alter type locations_t add attribute(departments_tab departments_tab_t) cascade;
alter type departments_t add attribute(job_history_tab job_history_tab_t,
employees_tab employees_tab_t) cascade;
alter type jobs_t add attribute(job_history_tab job_history_tab_t, employees_tab
employees_tab_t) cascade;
alter type employees_t add attribute(responsible_of_emp_tab employees_tab_t,
    manager_of_dep_tab departments_tab_t,
    job_history_tab job_history_tab_t) cascade;

-- Creating tables
create table regions of regions_t
    nested table countries_tab store as r_countries_nt;

create table countries of countries_t
    nested table locations_tab store as c_locations_nt;

create table locations of locations_t
    nested table departments_tab store as l_departments_nt;

create table departments of departments_t
    nested table job_history_tab store as d_job_history_nt
    nested table employees_tab store as employees_nt;
```

```
create table jobs of jobs_t
    nested table job_history_tab store as job_history_nt
    nested table employees_tab store as j_employees_nt;

create table employees of employees_t
    nested table responsible_of_emp_tab store as e_employees_nt
    nested table job_history_tab store as e_job_history_nt,
    nested table manager_of_dep_tab store as e_departments_nt;

create table job_history of job_history_t ;
```

3) Table Population

```
insert into regions (region_id, region_name)
select *
from gtd10.regions;

insert into countries (country_id, country_name, region)
select c.country_id, c.country_name, ref(r)
from gtd10.countries c
inner join regions r on r.region_id = c.region_id;

insert into locations (location_id, street_addreess, postal_code, city,
countries)
select l.location_id,  street_address, postal_code, city, ref(c)
from gtd10.locations l
inner join countries c on c.country_id = l.country_id ;

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 gtd10.jobs j;


insert into employees (employee_id, first_name, last_name, email,
    phone_number, hire_date, salary, commission_pct, jobs)
select e.employee_id, e.first_name, e.last_name, e.email,
    e.phone_number, e.hire_date, e.salary, e.commission_pct, ref(j)
from gtd10.employees e
inner join jobs j  on j.job_id = e.job_id;

insert into departments (department_id, department_name, manager, locations)
select d.department_id, d.department_name, ref(e), ref(l)
from gtd10.departments d
inner join employees e on e.employee_id = d.manager_id
inner join locations l on l.location_id = d.location_id;
```

```sql
update employees e
set e.department = (
    select ref(d)
    from departments d
    inner join gtd10.employees ge on ge.department_id = d.department_id
    where ge.employee_id = e.employee_id
);

insert into job_history (start_date, end_date, department, employee, jobs)
select jh.start_date, jh.end_date, ref(d), ref(e), ref(j)
from gtd10.job_history jh
inner join departments d on d.department_id = jh.department_id
inner join employees e on e.employee_id = jh.employee_id
inner join jobs j on j.job_id = jh.job_id;


-- Add nested tables
update regions r
set r.countries_tab = cast(multiset(
    select ref(c)
    from countries c
    where c.region.region_id = r.region_id
) as countries_tab_t);


update countries c
set c.locations_tab = cast(multiset(
    select ref(l)
    from locations l
    where l.countries.country_id = c.country_id
) as locations_tab_t);


update locations l
set l.departments_tab = cast(multiset(
    select ref(d)
    from departments d
    where d.locations.location_id = l.location_id
) as departments_tab_t);


update jobs j
set j.job_history_tab =
    cast(multiset(
        select ref(jh)
        from job_history jh
        where jh.jobs.job_id = j.job_id
    ) as job_history_tab_t),
j.employees_tab = cast(multiset(
        select ref(e)
        from employees e
        where e.jobs.job_id = j.job_id
    ) as employees_tab_t);


update departments d
set d.job_history_tab =
    cast(multiset(
        select ref(jh)
        from job_history jh
        where jh.department.department_id = d.department_id
    ) as job_history_tab_t),
d.employees_tab =
    cast(multiset(
        select ref(emp)
        from employees emp
        where emp.department.department_id = d.department_id
    ) as employees_tab_t);

update employees e
set e.job_history_tab =
    cast(multiset(
        select ref(jh)
        from job_history jh
        where jh.employee.employee_id = e.employee_id
    ) as job_history_tab_t),
e.responsible_of_emp_tab =
    cast(multiset(
        select ref(emp)
        from employees emp
        where emp.employee_id = e.manager.employee_id
    ) as employees_tab_t),
e.manager_of_dep_tab =
    cast(multiset(
        select ref(d)
        from departments d
        where d.manager.employee_id = e.employee_id
    ) as departments_tab_t);
```

4) Useful methods

Function : `total_employees`

This function is responsible for returning the total number of employees in the enterprise.

```
create or replace function total_employees return number is
    num_employees number;

begin
    select count(employee_id) into num_employees
    from employees;

    return num_employees;
end total_employees;
```

Function: `get_best_paid_employee`

This function returns the id of the best paid employee in a department.

```
create or replace function get_best_paid_employee(dep_id number) return number
is
    emp_id number;
begin
    select value(e).employee_id into emp_id
    from employees e
    where e.salary in (
        select max(value(e).salary)
        from departments d, table(d.employees_tab) e
        where d.department_id = dep_id
    ) and e.department.department_id = dep_id;
    return emp_id;
end;
```

Member function departments_t: `ever_worked`

This member function calculates how many employees are working **or** have worked in the department.

```
alter type departments_t add member function ever_worked return number cascade;
create or replace type body departments_t as

    member function ever_worked return number is
    num_ever_worked number;
    begin

        select count(e.employee_id) into num_ever_worked
        from employees e
        where employee_id in (
        select value(et).employee_id
        from departments d, table(d.employees_tab) et
        where d.department_id = self.department_id
    )
    or e.employee_id in (
        select value(jh).employee.employee_id
        from departments d, table(d.job_history_tab) jh
        where d.department_id = self.department_id
    );
        return num_ever_worked;
    end ever_worked;

end;
```

Function: `get_min_history_end_date`

Function responsible for retrieving the minimum history end date of a worker.

```
create or replace function get_min_history_end_date(employee_id number) return
date is
    end_date date;
begin
    select min(jh.end_date) as endDate into end_date
    from job_history jh
    where jh.employee.employee_id = employee_id;

    return end_date;
end get_min_history_end_date;
```

Function: `get_max_history_start_date`

Function responsible for retrieving the maximum history start date of a worker.

```
create or replace function get_max_history_start_date(employee_id number) return
date is
    start_date date;
```

```
begin
    select max(jh.start_date) into start_date
    from job_history jh
    where jh.employee.employee_id = employee_id;

    return start_date;
end get_max_history_start_date;
```

Function: `job_start_date`

Function responsible for retrieving the start date of the current job that an employee has. If the employee has records on the job history table, the start day is the day after the most recent end date of that employee, else it corresponds to the hire date.

```
alter type employees_t add member function job_start_date return date cascade;

create or replace type body employees_t as
    member function job_start_date return date is
        start_date date; end_date date;
    begin
        select max(value(jh).end_date+1) into end_date
        from table(self.job_history_tab) jh;

        if end_date != null then
            return end_date;
        else
            return self.hire_date;
        end if;
    end job_start_date;
end;
```

5) Questions

Question a)

Calculate the total number of employees that each department has got

Query

1) First approach (correct)

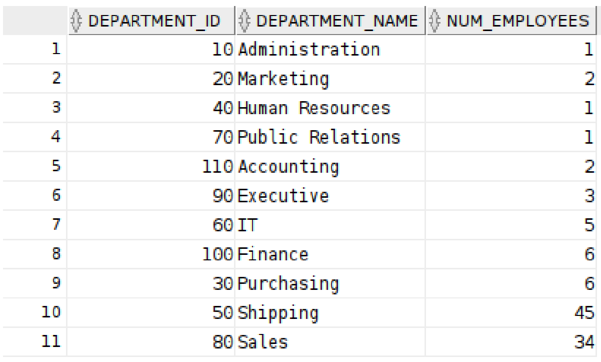
```
select department_id, department_name, cardinality(employees_tab) as
num_employees
from departments;
```

2) Second approach (incorrect)

```
select e.department.department_name, count(e.employee_id) as num_employees
from employees e
group by e.department;
```

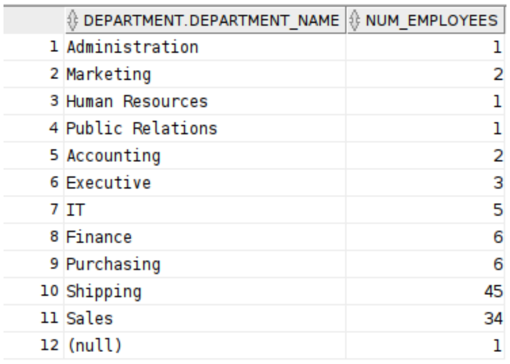
Result

1) First approach



	DEPARTMENT_ID	DEPARTMENT_NAME	NUM_EMPLOYEES
1	10	Administration	1
2	20	Marketing	2
3	40	Human Resources	1
4	70	Public Relations	1
5	110	Accounting	2
6	90	Executive	3
7	60	IT	5
8	100	Finance	6
9	30	Purchasing	6
10	50	Shipping	45
11	80	Sales	34

2) Second approach



	DEPARTMENT.DEPARTMENT_NAME	NUM_EMPLOYEES
1	Administration	1
2	Marketing	2
3	Human Resources	1
4	Public Relations	1
5	Accounting	2
6	Executive	3
7	IT	5
8	Finance	6
9	Purchasing	6
10	Shipping	45
11	Sales	34
12	(null)	1

Query comments

To answer this question, there are different ways to achieve the same result. We can have an approach starting from the departments table or we can have another approach using the employees table. In the first approach we use the departments table to reach the number of employees of that department, using the cardinality of the employees_tab. In the second approach we start from the employees table, grouping it by departments and using count to determine the number of employees. The first approach is more simple and efficient and can be more accurate if there's any employee that has no attributed department.

Question b)

In each department, how many employees are there for each job title?

Query

1) First approach (incorrect)

```
select unique(d.department_id) as department_id, value(e).jobs.job_id as jobs,
cardinality(value(e).jobs.employees_tab) as num_workers
from departments d, table(d.employees_tab) e;
```

2) Second approach (correct)

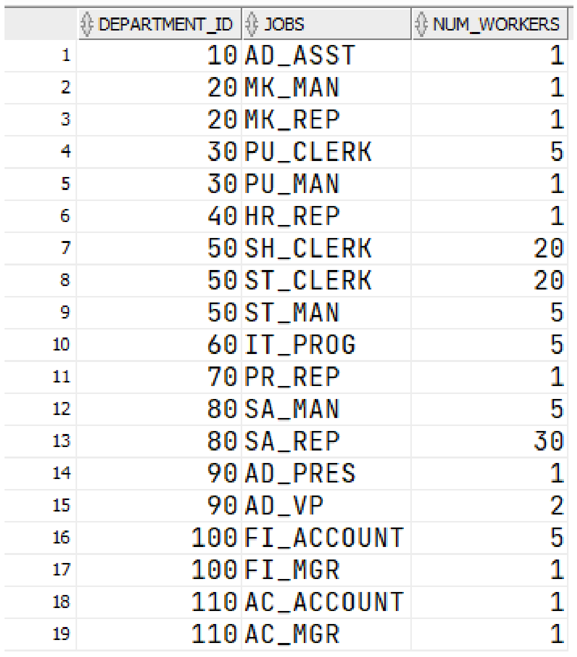
```
select unique(e.department.department_id) as department_id, e.jobs.job_id as
jobs, count(employee_id) as employee_cnt
from employees e
group by department, jobs;
```

3) Third approach (incorrect)

```
select unique(value(e).department.department_id) as department_id, j.job_id as
jobs, cardinality(j.employees_tab) as num_workers
from jobs j, table(j.employees_tab) e;
```

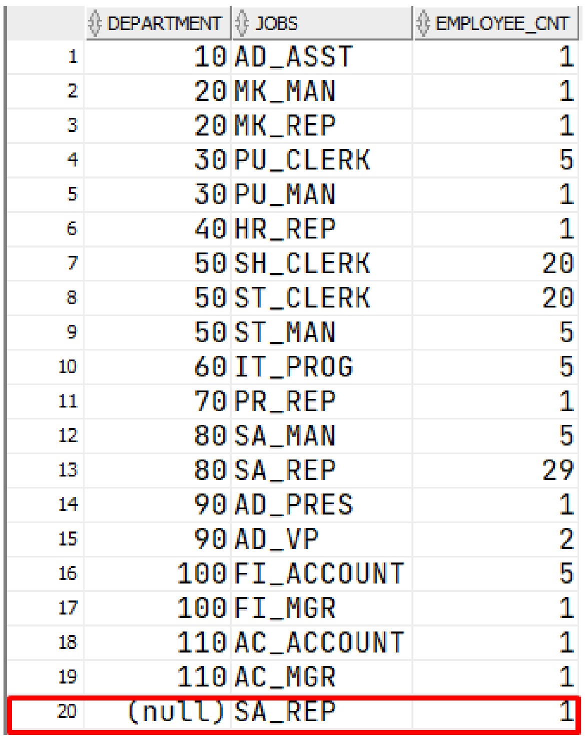
Result

1) First approach



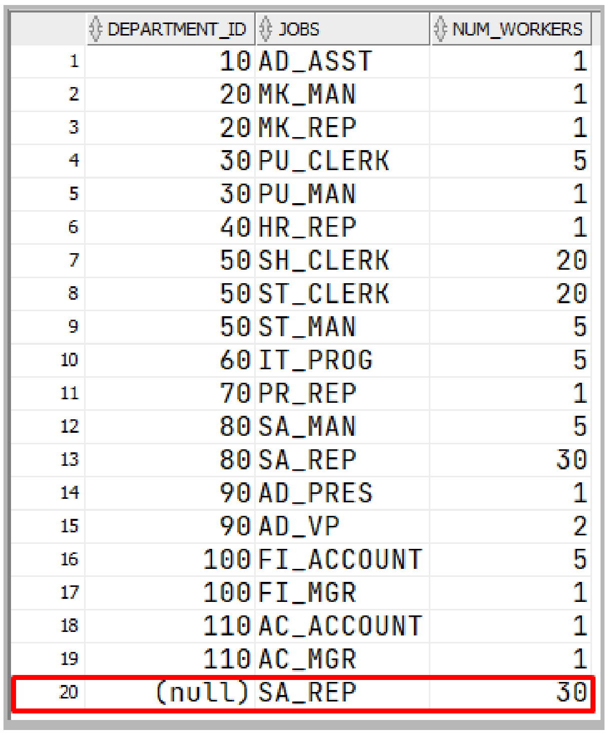
	DEPARTMENT_ID	JOBS	NUM_WORKERS
1	10	AD_ASST	1
2	20	MK_MAN	1
3	20	MK_REP	1
4	30	PU_CLERK	5
5	30	PU_MAN	1
6	40	HR_REP	1
7	50	SH_CLERK	20
8	50	ST_CLERK	20
9	50	ST_MAN	5
10	60	IT_PROG	5
11	70	PR_REP	1
12	80	SA_MAN	5
13	80	SA_REP	30
14	90	AD_PRES	1
15	90	AD_VP	2
16	100	FI_ACCOUNT	5
17	100	FI_MGR	1
18	110	AC_ACCOUNT	1
19	110	AC_MGR	1

2) Second approach



	DEPARTMENT	JOBS	EMPLOYEE_CNT
1	10	AD_ASST	1
2	20	MK_MAN	1
3	20	MK_REP	1
4	30	PU_CLERK	5
5	30	PU_MAN	1
6	40	HR_REP	1
7	50	SH_CLERK	20
8	50	ST_CLERK	20
9	50	ST_MAN	5
10	60	IT_PROG	5
11	70	PR_REP	1
12	80	SA_MAN	5
13	80	SA_REP	29
14	90	AD_PRES	1
15	90	AD_VP	2
16	100	FI_ACCOUNT	5
17	100	FI_MGR	1
18	110	AC_ACCOUNT	1
19	110	AC_MGR	1
20	(null)	SA_REP	1

3) Third approach



	DEPARTMENT_ID	JOBS	NUM_WORKERS
1	10	AD_ASST	1
2	20	MK_MAN	1
3	20	MK_REP	1
4	30	PU_CLERK	5
5	30	PU_MAN	1
6	40	HR_REP	1
7	50	SH_CLERK	20
8	50	ST_CLERK	20
9	50	ST_MAN	5
10	60	IT_PROG	5
11	70	PR_REP	1
12	80	SA_MAN	5
13	80	SA_REP	30
14	90	AD_PRES	1
15	90	AD_VP	2
16	100	FI_ACCOUNT	5
17	100	FI_MGR	1
18	110	AC_ACCOUNT	1
19	110	AC_MGR	1
20	(null)	SA_REP	30

Query comments

Although one might think that this question can be answered in multiple ways, there's just one approach that fits into what the question is expecting: the number of employees that work in a job and in a specific department. For this reason, three similar queries were created and in the following lines it will be explained the difference between those.

Before continuing, it's important that we understand the data: there's one employee that has **no department** attributed (i.e `employee_id = 178`), but has a job (i.e `job_id = SA_REP)`.

In the first query we're not able to identify what employees don't have a department, since we started by getting the employees of each department. However, by doing `cardinality(value(e).jobs.employees_tab) as num_workers,` we get the number of employees of each possible job that the department contains and **not necessarily** all the employees in a job belong to the same department. Thus, we're answering the question: the department is related to what jobs **and** each job has how many employees? However, since one employee belongs to just one department and, coincidentally, one job has just one department associated with it, the result is really close to the expected one.

In the second approach, the query starts in the employees table. This query suits what the question is asking for: the number of employees that work in a certain department and in a job.

The third query is analogous to the first one, but it becomes more clear that the results don't answer the question asked. In this query we are answering the question: the job is related to what departments **and** has how many employees? For this reason the last line is checking what is the department of the employee with id 176, which is null, and setting the number of employees to 30, since the number of employees in the SA_REP job is 30.

Question c)

Indicate the best paid employee in each department.

Query

1) First approach

```
create view maxSalariesByDepartment as select max(e.salary) as
maxSalary,value(d).department_name as departmentName
from employees e,table(e.manager_of_dep_tab) d
where value(d).department_name is not null
group by value(d).department_name;

select employee_id,e.department.department_name, maxSalary
from employees e
inner join maxSalariesByDepartment msbd on e.salary = msbd.maxSalary
and e.department.department_name = msbd.departmentName;
```

2) Second approach

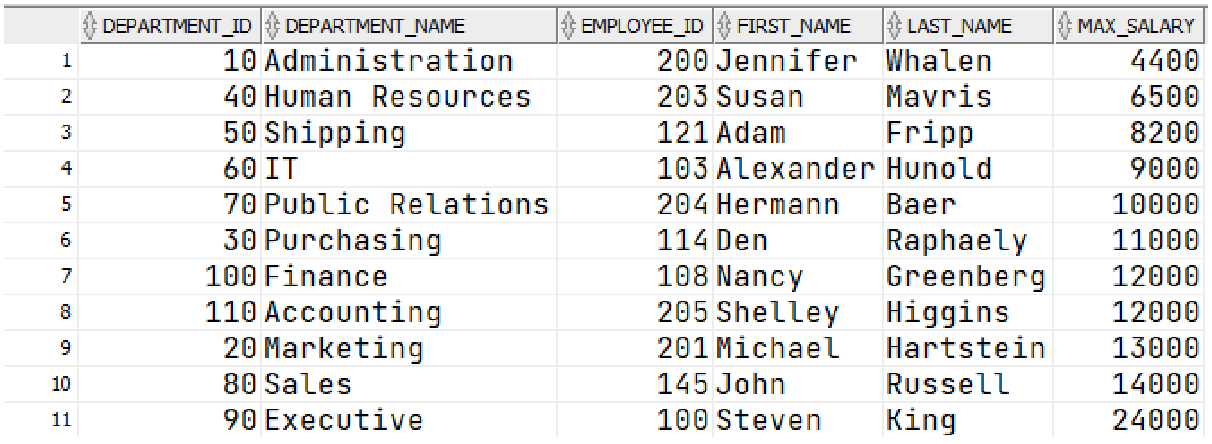
```
select value(d).department_id as department_id, value(d).department_name as
department_name, e.employee_id, e.first_name, e.last_name, max(e.salary)
max_salary
from employees e, table(e.manager_of_dep_tab) d
group by e.employee_id, e.first_name, e.last_name, value(d).department_name,
value(d).department_id;
```

3) Third approach

```
create or replace function get_best_paid_employee(dep_id number) return number
is
    emp_id number;
begin
    select value(e).employee_id into emp_id
    from employees e
    where e.salary in (
        select max(value(e).salary)
        from departments d, table(d.employees_tab) e
        where d.department_id = dep_id
    ) and e.department.department_id = dep_id;
    return emp_id;
end;
```

```
select department_id, department_name, get_best_paid_employee(department_id)
best_paid_employee,
    value(e).salary salary
from departments d, table(d.employees_tab) e
where value(e).employee_id = get_best_paid_employee(department_id)
order by salary;
```

Result



	DEPARTMENT_ID	DEPARTMENT_NAME	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	MAX_SALARY
1	10	Administration	200	Jennifer	Whalen	4400
2	40	Human Resources	203	Susan	Mavris	6500
3	50	Shipping	121	Adam	Fripp	8200
4	60	IT	103	Alexander	Hunold	9000
5	70	Public Relations	204	Hermann	Baer	10000
6	30	Purchasing	114	Den	Raphaely	11000
7	100	Finance	108	Nancy	Greenberg	12000
8	110	Accounting	205	Shelley	Higgins	12000
9	20	Marketing	201	Michael	Hartstein	13000
10	80	Sales	145	John	Russell	14000
11	90	Executive	100	Steven	King	24000

Query comments

In the first approach, we created a view that selects the maximum salary for each department. For that, we use the MAX function and we use the nested table manager_of_dep_tab to get the value of the department name. Then, we will search through every employee to find which employee has the salary equal to maximum salary, for the department where he works. For that, we use an inner join on the columns maxSalary and departmentName.

On the second approach, with a simple select and a group by, also using the MAX function for the salary attribute, we find the employee with the maximum salary for each department.

Although the first and second approaches use the `manager_of_dep_tab,` it still returns the correct result, since the manager of an employee, by our interpretation, is the department manager. However, this yet might not be true and for this reason the third approach was created: it uses the `get_best_paid_employee` function to return the best paid employee of a department and we use the `employees_tab` to display the salary.

Question d)

a) Check whether the job history has time gaps for each employee.
b) Sort the employees by job duration on the current day.

Query a)

```
create or replace function get_min_history_end_date(employee_id number) return
date is
    end_date date;
begin
    select min(jh.end_date) as endDate into end_date
    from job_history jh
    where jh.employee.employee_id = employee_id;

    return end_date;
end get_min_history_end_date;
```

```
create or replace function get_max_history_start_date(employee_id number) return
date is
    start_date date;
begin
    select max(jh.start_date) into start_date
    from job_history jh
    where jh.employee.employee_id = employee_id;

    return start_date;
end get_max_history_start_date;
```

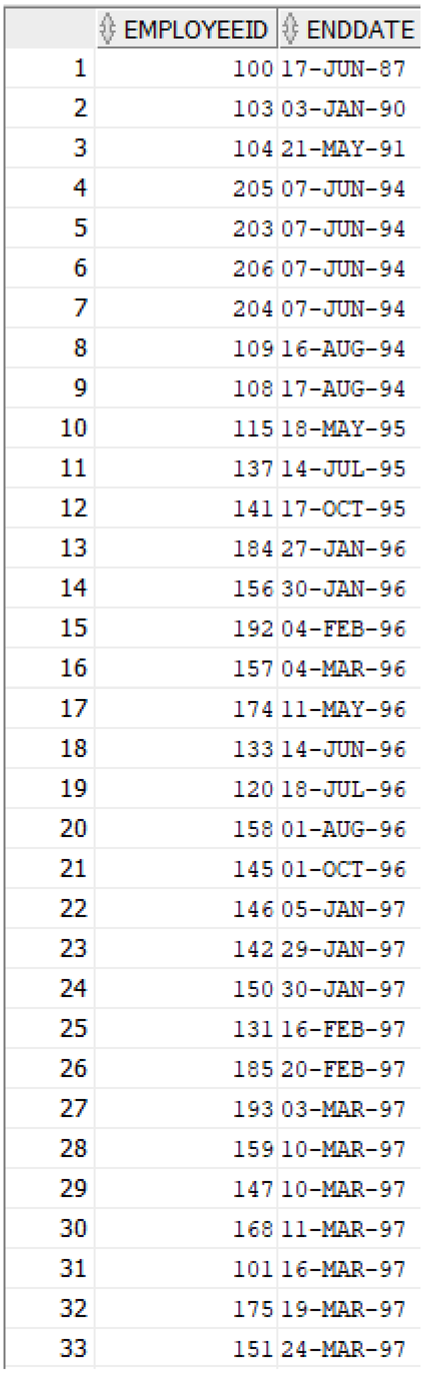
```
select employee_id, first_name, last_name
from employees e
where get_max_history_start_date(employee_id) -
      get_min_history_end_date(employee_id) > 1;
```

Query b)

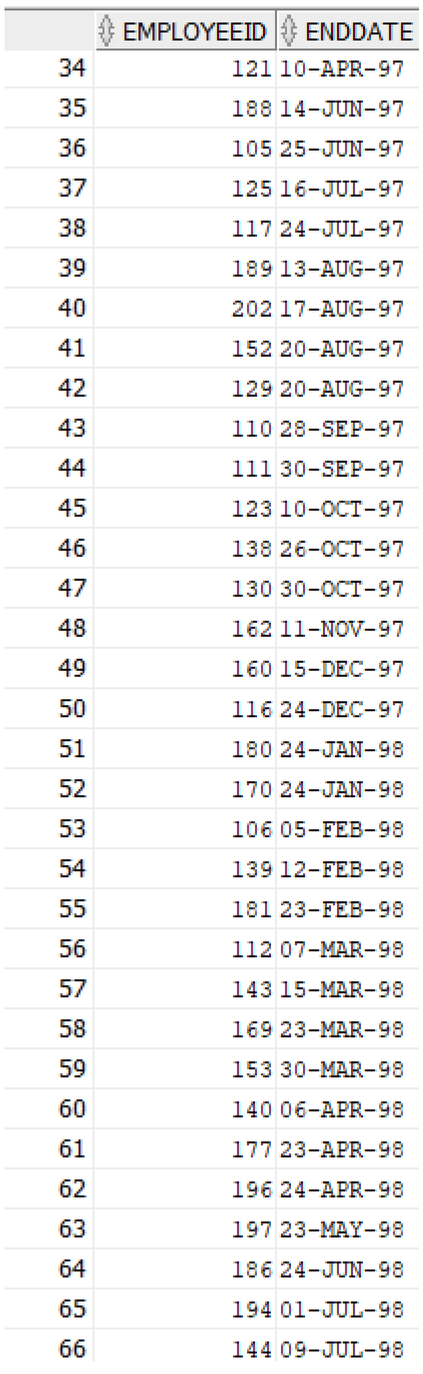
```
alter type employees_t add member function job_start_date return date cascade;

create or replace type body employees_t as
    member function job_start_date return date is
        start_date date; end_date date;
    begin
        select max(value(jh).end_date+1) into end_date
        from table(self.job_history_tab) jh;
        if end_date != null then
            return end_date;
        else
            return self.hire_date;
        end if;
    end job_start_date;
end;
```

```
select employee_id, first_name, last_name, e.job_start_date() as work_start_day
from employees e
order by months_between(current_date, work_start_day) desc;
```

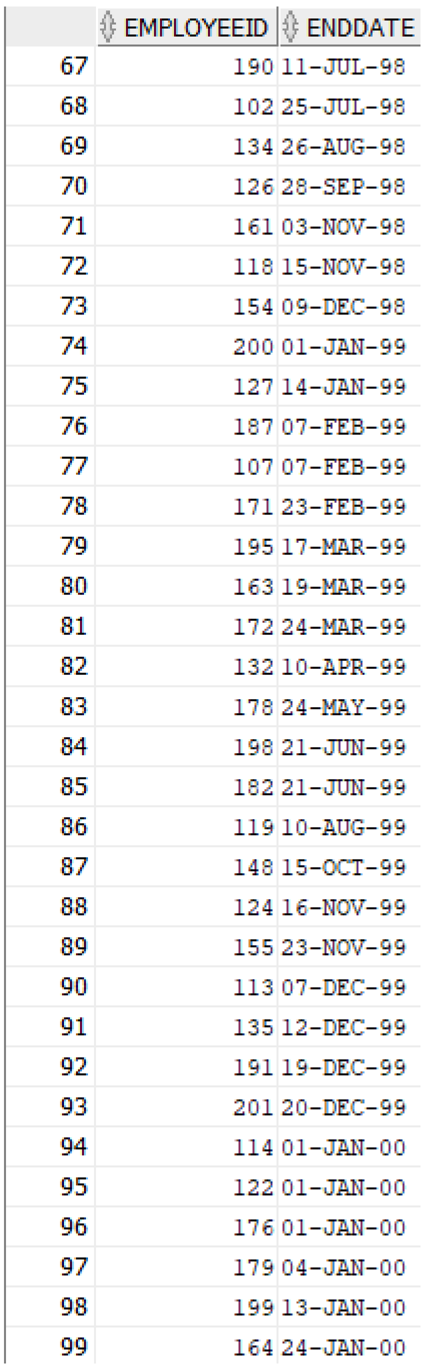
Result



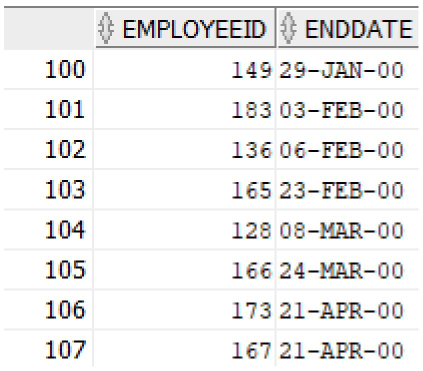
	EMPLOYEEID	ENDDATE
1	100	17-JUN-87
2	103	03-JAN-90
3	104	21-MAY-91
4	205	07-JUN-94
5	203	07-JUN-94
6	206	07-JUN-94
7	204	07-JUN-94
8	109	16-AUG-94
9	108	17-AUG-94
10	115	18-MAY-95
11	137	14-JUL-95
12	141	17-OCT-95
13	184	27-JAN-96
14	156	30-JAN-96
15	192	04-FEB-96
16	157	04-MAR-96
17	174	11-MAY-96
18	133	14-JUN-96
19	120	18-JUL-96
20	158	01-AUG-96
21	145	01-OCT-96
22	146	05-JAN-97
23	142	29-JAN-97
24	150	30-JAN-97
25	131	16-FEB-97
26	185	20-FEB-97
27	193	03-MAR-97
28	159	10-MAR-97
29	147	10-MAR-97
30	168	11-MAR-97
31	101	16-MAR-97
32	175	19-MAR-97
33	151	24-MAR-97



	EMPLOYEEID	ENDDATE
34	121	10-APR-97
35	188	14-JUN-97
36	105	25-JUN-97
37	125	16-JUL-97
38	117	24-JUL-97
39	189	13-AUG-97
40	202	17-AUG-97
41	152	20-AUG-97
42	129	20-AUG-97
43	110	28-SEP-97
44	111	30-SEP-97
45	123	10-OCT-97
46	138	26-OCT-97
47	130	30-OCT-97
48	162	11-NOV-97
49	160	15-DEC-97
50	116	24-DEC-97
51	180	24-JAN-98
52	170	24-JAN-98
53	106	05-FEB-98
54	139	12-FEB-98
55	181	23-FEB-98
56	112	07-MAR-98
57	143	15-MAR-98
58	169	23-MAR-98
59	153	30-MAR-98
60	140	06-APR-98
61	177	23-APR-98
62	196	24-APR-98
63	197	23-MAY-98
64	186	24-JUN-98
65	194	01-JUL-98
66	144	09-JUL-98



	EMPLOYEEID	ENDDATE
67	190	11-JUL-98
68	102	25-JUL-98
69	134	26-AUG-98
70	126	28-SEP-98
71	161	03-NOV-98
72	118	15-NOV-98
73	154	09-DEC-98
74	200	01-JAN-99
75	127	14-JAN-99
76	187	07-FEB-99
77	107	07-FEB-99
78	171	23-FEB-99
79	195	17-MAR-99
80	163	19-MAR-99
81	172	24-MAR-99
82	132	10-APR-99
83	178	24-MAY-99
84	198	21-JUN-99
85	182	21-JUN-99
86	119	10-AUG-99
87	148	15-OCT-99
88	124	16-NOV-99
89	155	23-NOV-99
90	113	07-DEC-99
91	135	12-DEC-99
92	191	19-DEC-99
93	201	20-DEC-99
94	114	01-JAN-00
95	122	01-JAN-00
96	176	01-JAN-00
97	179	04-JAN-00
98	199	13-JAN-00
99	164	24-JAN-00



	EMPLOYEEID	ENDDATE
100	149	29-JAN-00
101	183	03-FEB-00
102	136	06-FEB-00
103	165	23-FEB-00
104	128	08-MAR-00
105	166	24-MAR-00
106	173	21-APR-00
107	167	21-APR-00

Query comments

Regarding the first point, we first select the oldest end date that an employee has. Then, we select the most recent start date that the employee has, and we see if there is a difference of more than 1 day between those dates. If an employee only has 1 record on the table job_history, then it can't have time gaps.

For the second point, we first see if the employee has a record in the table job history. If it has, we select the oldest end date that he has (if he only has 1 record it doesn't matter, but if he has more it is important), and we see how many months have passed between he started the current job (he started the day after the oldest end date he has on job history table) and the current date. If the employee doesn't have records on the job history table, we will see how many months have passed between his hire date and the current date. Then, having the two attributes employeeId and monthsPassed, we order the table by monthsPassed, in descendent order.

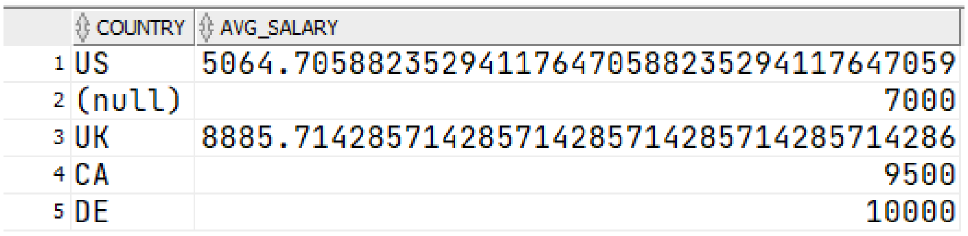
Question e)

Compare the average salary by country?

Query

```
select e.department.locations.countries.country_id as country, avg(salary) as
avg_salary
from employees e
group by e.department.locations.countries.country_id
order by avg_salary;
```

Result



	COUNTRY	AVG_SALARY
1	US	5064.705882352941176470588235294117647059
2	(null)	7000
3	UK	8885.714285714285714285714285714285714286
4	CA	9500
5	DE	10000

Query comments

As analyzed in query c, if we started by getting the information from the *countries* table, we would lose information by not considering that an employee might not have a country associated.

Question f)

What is the percentage of employees that work or have worked in a department?

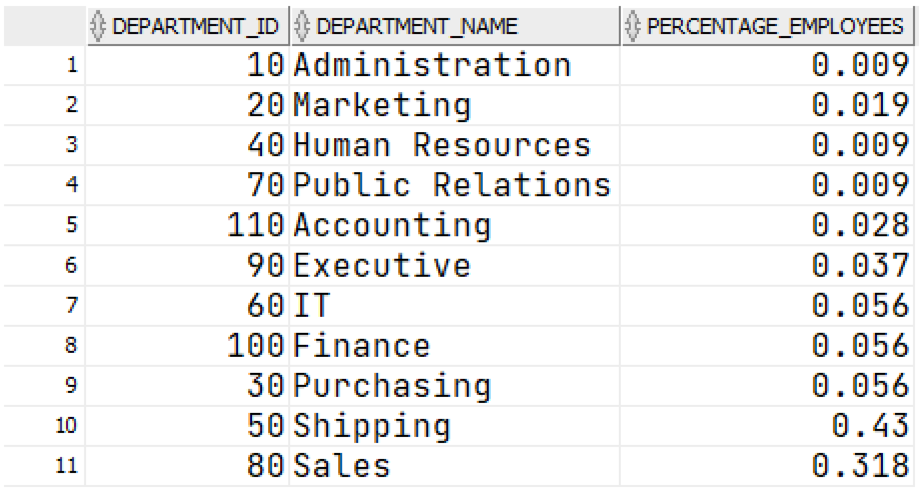
Query

```
create or replace function total_employees return number is
    num_employees number;

begin
    select count(employee_id) into num_employees
    from employees;

    return num_employees;
end total_employees;
```

```
alter type departments_t add member function ever_worked return number cascade;
create or replace type body departments_t as

    member function ever_worked return number is
    num_ever_worked number;
    begin

        select count(e.employee_id) into num_ever_worked
        from employees e
        where employee_id in (
        select value(et).employee_id
        from departments d, table(d.employees_tab) et
        where d.department_id = self.department_id
    )
    or e.employee_id in (
        select value(jh).employee.employee_id
        from departments d, table(d.job_history_tab) jh
        where d.department_id = self.department_id
    );
        return num_ever_worked;
    end ever_worked;

end;
```

```
select  d.department_id, d.department_name,
round(d.ever_worked()/total_employees(), 3) as percentage_employees
from departments d;
```

Result



	DEPARTMENT_ID	DEPARTMENT_NAME	PERCENTAGE_EMPLOYEES
1	10	Administration	0.009
2	20	Marketing	0.019
3	40	Human Resources	0.009
4	70	Public Relations	0.009
5	110	Accounting	0.028
6	90	Executive	0.037
7	60	IT	0.056
8	100	Finance	0.056
9	30	Purchasing	0.056
10	50	Shipping	0.43
11	80	Sales	0.318

Query comments

For this query some functions were created: the `ever_worked` member function and the `total_employees` function. The `ever_worked` function calculates how many employees have ever worked in the department and the `total_employees` calculates the total number of employees in the enterprise. With this question we wanted to show that, even in complex extended questions like this one, that involves many tables, the OR databases still didn't perform any kind of join.