OBJECT RELATIONAL LAB (3)

PLEASE NOTE THIS LAB ASSUMES YOU HAVE COMPLETED THE PREVIOUS Object Relational Lab

Make sure to read EACH section to the end before starting the exercises.

References or REFs

References (REF) can be used instead of foreign keys in many-to-one relationships. Note that the references point to object types not object tables.

```
CREATE TABLE employment_unscoped (
employee REF employee,
position REF my_ job);
```

In addition to referencing the type it is also possible to restrict the references to actual object tables by using "SCOPE IS". Scoped references are implemented more efficiently by Oracle and are processed faster. But scope can only be defined when creating a table, not when creating a type.

```
CREATE TABLE employment (
staffMember REF O_EMPLOYEE SCOPE IS employee_table,
position REF O_JOB SCOPE IS job_table);
```

The data to be inserted into tables with REFs comes from the corresponding object tables (i.e., employee_table and job_table). The function REF in the following statement provides the pointers to the objects in employee_table and job_table which are then inserted into employment like:

```
INSERT INTO employment
SELECT REF(e), REF(j)
FROM job_table j, employee_table e
WHERE e.emp_ID = 2
AND j.job_ID = 1;
```

Exercises:

1. CREATE an emp_table object table based on EMPLOYEE_T object type. E EMPLOYEE_T has attributes as follows name (string), ppsn(string), homeAddress(O_ADDRESS), emp_status(string), phone_number(string). Constraints required for the table are ppsn (pk), name and status are mandatory.

2. Insert 2 rows into the emp_table

3 Create a job_role_t object type that has the following attributes

```
JOBTITLE (string)
JOB_ID number
SALARY_AMOUNT(number)
```

The object table is job_post_table – all attributes are mandatory and job_id is the primary key

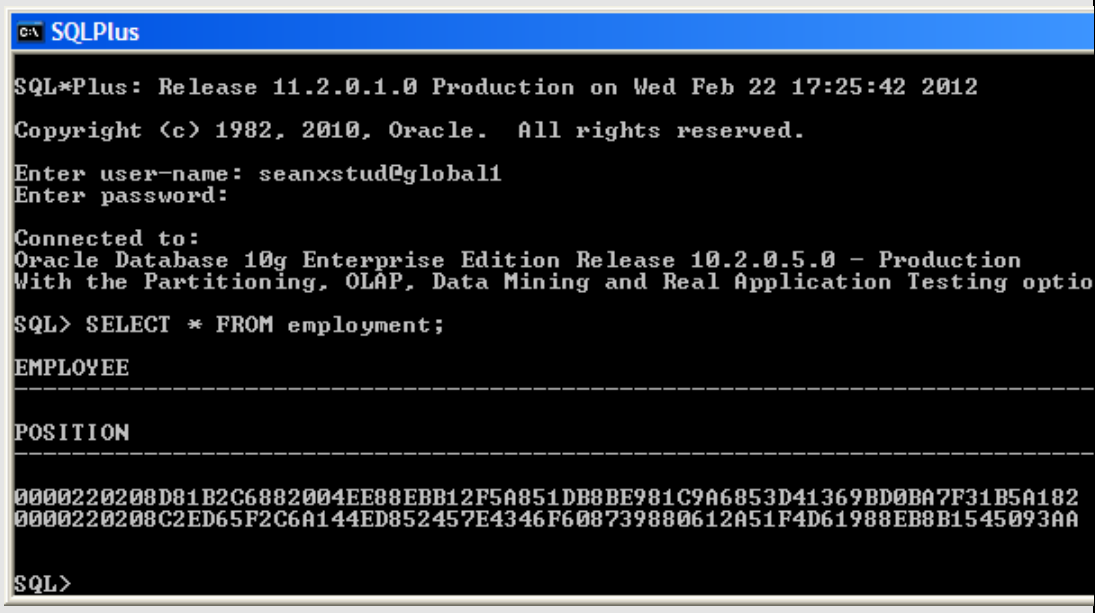
Insert 3 rows into the table.

4 Now create the scoped employment table called EMPLOYMENT

```
CREATE TABLE employment (
staffMember REF EMPLOYEE_T SCOPE IS emp_table,
position REF JOB_role_t SCOPE IS job_post_table);
```

Insert 3 rows into it. Make sure you the ppsn's and job_id's exist in your tables.

5 What does SELECT * FROM EMPLOYMENT show? The cloud database cannot display the REF value. This is what you would see in an on premise database i.e. the REF values



Print the complete information from table "employment" in two formats:

(i) output it in a way that it shows all types and all data. (E.g EMPLOYEE(NAME('Mary', NULL, 'Edwards'), ...) You can do this by dereferencing (DEREF) the two columns of employment. e.g. to DEREF an employee object as part of the SQL statement DEREF(e.employee) where e is the table alias.

Note: the cloud database only signifies they are objects or unsupported data types. It does not print the actual values. The Oracle APEX GUI not able to display the de-refed object due to the limitations of the interface.

(ii)Second, write a query that shows the following data for employees: name and job title.

6 (i) Using the employment table, print the names of all employees whose salary is larger than 20000.
(ii) Print the job titles of all employees whose home address is in Dublin (or another city of your choice!).
(Note that in contrast to relational databases you do not need a join to do this!).
Hint: you need to use dot notation e.g. `select e.staffMember.name`

7 Briefly evaluate the differences between the relational approach and the object relational approach that you have learned so far. Which of the approaches is easier to design, easier to maintain or easier to use? Is there any danger of anomalies or inconsistencies in the object-relational approach? Is normalisation relevant for the object-relational approach?

8 Using the Employment table only
(i) Write a query that prints the job title and salary for an employee Mary Miller for example. (amend to suit your data)

(ii) How many employees have a salary at most €100,000 and are web designers (amend to suit your data)?

(iii) What is the total salary bill for all employees employed as a Network Engineer (amend to suit your data)?

9 Let us try IS DANGLING predicate. First, delete an job record from the job_post_table that you referenced in exercise 4. Now use the IS DANGLING predicate on the employee_table as follows

```
SELECT e.staffMember.name
FROM employment e
WHERE e.position  IS DANGLING;
```

Note you will have to change the SQL query to suit your data. Try the IS NOT DANGLING predicate too!

--1.

```
CREATE OR REPLACE TYPE employee_t AS OBJECT
(name VARCHAR2(30),
 ppsn VARCHAR2(20),
 homeAddress  O_ADDRESS,
 status CHAR(2),
 phone_number  VARCHAR2(10)
 )
```

```
CREATE TABLE EMP_TABLE OF employee_t
(ppsn PRIMARY KEY,
 name NOT NULL,
 status NOT NULL
);
```

--2.

```
INSERT INTO EMP_TABLE VALUES ( employee_t('MARY MILLER',
                                          '3533533M',
                                          O_ADDRESS(O_STREET('MAIN
STREET',3, NULL),
                                                   'DUBLIN',
                                                   '24',
                                                   'DUBLIN',
                                                   'EIRE'

),
                                          'PT',
                                          '01-987569'
                                          )
                                );


 INSERT INTO EMP_TABLE VALUES ( employee_t('JOHN HAYES',
                                         '76455555M',
                                          O_ADDRESS(O_STREET('UPPER
STREET',233, NULL),
                                                   'DUBLIN',
                                                   '24',
                                                   'DUBLIN',
                                                   'EIRE'

),
                                          'FT',
                                          '01-234666'
                                          )
                                );
CREATE TABLE employment (
staffMember REF O_EMPLOYEE SCOPE IS employee_table,
position REF O_JOB SCOPE IS job_table);
```

```
--3
CREATE TYPE JOB_ROLE_T AS OBJECT (
JOBTITLE VARCHAR(20),
JOB_ID INTEGER,
SALARY_AMOUNT INTEGER
);

CREATE TABLE JOB_POST_TABLE OF JOB_ROLE_T
(JOB_ID PRIMARY KEY,
 JOBTITLE  NOT NULL,
 SALARY_AMOUNT NOT NULL
);

 INSERT INTO JOB_POST_TABLE VALUES (JOB_ROLE_T ('Web Designer',
                                          1,
                                          70000
                                                  )
                                    );

 INSERT INTO JOB_POST_TABLE VALUES (JOB_ROLE_T ('Network Engineer',
                                          2,
                                          75000
                                                  )
                                    );


 INSERT INTO JOB_POST_TABLE VALUES (JOB_ROLE_T ('Cloud Architect',
                                          3,
                                          90000
                                                  )
                                    );
--4
CREATE TABLE employment (
staffMember REF EMPLOYEE_T SCOPE IS emp_table,
position REF JOB_role_t SCOPE IS job_post_table);

INSERT INTO employment
SELECT REF(e), REF(j)
FROM job_post_table j, emp_table e
WHERE e.ppsn = '3533533M'
AND j.job_ID = 1;



INSERT INTO employment
SELECT REF(e), REF(j)
FROM job_post_table j, emp_table e
WHERE e.ppsn = '76455555M'
AND j.job_ID = 2;
```

--5.

```
SELECT * FROM EMPLOYMENT

-- (i)
select DEREF(e.staffMember), DEREF(e.position) from employment e;
```

```
      --(ii)
      select e.staffMember.name,e.position.jobtitle
      from employment e;
--6.
select e.staffMember.name,e.position.salary_amount
from employment e where e.position.salary_amount > 20000;

select DISTINCT(e.position.jobtitle) from employment e
where e.staffMember.homeaddress.city = 'DUBLIN';

--7.
```

See class notes
The traditional relational approach is based on highly normalised design. However, you are guaranteed of the integrity of the data in the database. Some points should be discussed

- focuses on tables with columns and rows and standard datatypes
- From an OOP perspective, to make the application objects persistent you must map your object to the the relational structures often to multiple tables- The impedance mismatch! ORMs like hibernate
- Discussion on normalisation and what is brings
- Transaction data being inserted may touch more than one table to be successful e.g. Master-Detail pattern Order and order items
- While SQL is a flexible declarative language, expensive joins are requiring to reconstitute the business entity in a GUI Application for example. E.g. Displaying and order.

With the Object relational approach, the idea is objects at the Application layer that need to be made persistent in the database are stored in their complex structures in one object table for example. Relational constraints are still supported:

- Reduces the impedance mismatch with OOP
- Most of the object-oriented concepts supported object types encapsulation, methods, overriding, inheritance etc. give some examples
- Complex object types supported which can stored in Object tables. Provide some examples with some discussion on object types, collections, and REFs
- Inserts are simpler in that one object table is touched; No joins required
- However, this can lead to duplicated data which can bring all the issues of unnormalised data e.g repeating groups. It a lot of circumstances this is not an issue as can capture a snapshot in time e.g. Shipping Address; OrderItems and their price
- Associations between objects are supported through REFs. No joins are required as pointers are used to traverse the associations.

Other insights considered

```
--8.
(i)
select e.position.jobtitle, e.position.salary_amount
from employment e
WHERE e.staffMember.pname.firstname ='MARY'
AND  e.staffMember.pname.lastname ='MILLER';

(ii)
(i)
```

```
select e.position.jobtitle, e.position.salary_amount
from employment e
WHERE e.staffMember.name ='MARY MILLER';

(ii)
SELECT COUNT(*)
from employment e
WHERE e.position.salary_amount <= 100000
AND e.position.jobtitle ='Web Designer';
(iii)
SELECT SUM(e.position.salary_amount)
from employment e
WHERE e.position.jobtitle ='Network Engineer';


--9.
DELETE FROM job_table
WHERE job_ID=2;

select e.staffMember.name
from employment e
WHERE e.position  IS DANGLING;
```