OBJECT RELATIONAL LAB (2)

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.

2.1 Information about object types

We have seen already there are data dictionary views available to us. The following select statements can be used to list user types and their attributes.

```
SELECT * FROM USER_TYPES;
SELECT * FROM USER_TYPE_METHODS;
SELECT * FROM USER_TYPE_ATTRS;
SELECT * FROM USER_OBJECTS;
SELECT * FROM USER_SOURCE;
```

2.2 Subtypes

Subtypes can be created under an existing type using "UNDER". But subtypes can only be created under types that are not "**FINAL**", i.e. not at the bottom of the type hierarchy. For example, to create a subtype "employee" under "person", the type "person" (from last week's exercises) must first be changed to "**NOT FINAL**". Because there is already an object table (PERSON_TABLE) with objects attached to "**O_PERSON**", the last word in the alter statement should be "CASCADE". That means that an alteration of "O_PERSON" also applies to "**PERSON_TABLE**" and its objects.

```
ALTER TYPE O_PERSON NOT FINAL CASCADE;
```

Next, a subtype "**O_EMPLOYEE**" can be generated under "**O_PERSON**". This subtype can itself be either NOT FINAL or FINAL. We will make it NOT FINAL for this example. A subtype inherits all columns from its supertype but can also have additional columns, which are declared within the brackets.

```
CREATE TYPE O_EMPLOYEE UNDER O_PERSON (
  empno    INTEGER,
  ppsn     VARCHAR2(10)
   ) NOT FINAL ;


CREATE TABLE EMPLOYEE_TABLE OF O_PERSON
```

Row objects or indeed column objects can be created which can store person objects and any of its subtypes. This is called substitutability. However, this can be turned off by using the clause NOT SUBSTITUTABLE AT ALL LEVELS for the object table of column object definition.

The "IS OF" clause can be used to check the type of objects. The following statement selects all contacts who are also a person, i.e., it selects all rows in employee_table. It assumes the EMPLOYEE_TABLE was created using the person object type.

SELECT e.pname.lastname FROM employee_table e WHERE value(e) IS OF (O_PERSON);

Exercise

1. Firstly, examine the objects below and make sure you have them created (**note:** you should have these already created from the last lab),
2. Now, create a type "O_EMPLOYEE" under "O_PERSON"
   - **i** First you need to alter your o_person type to allow inheritance
   - **ii** O_EMPLOYEE has an additional attributes "emp_ID", a "PPSN" and emp_status which is a string . Create it as NOT FINAL.
3. Create a corresponding CONTACTS_TABLE **object table** based on the super type O_PERSON. This table can hold objects of the supertype as well as any subtypes.
4. Insert 4 rows (2 of which are o_employee and 2 of which are o_person. Use the correct constructors.
5. Write a query that returns firstname, lastname of contacts that are employees ONLY (See Example above with IS OF). Extend the query to provide a count of the number of employees in the table

FOR REFERENCE PURPOSES
NOTE YOU CREATED THESE OBJECT TYPES IN THE LAST LAB
REFER TO YOUR DEFINITIONS AS you may have implemented them differently.

However, if you want to start a fresh use my definitions below. Change the definitions below with a prefix of OT_

```
CREATE OR REPLACE TYPE O_STREET AS OBJECT (
            SNAME VARCHAR2(30),
            SNUMBER NUMBER,
            FLATNUMBER VARCHAR2(5));


CREATE or REPLACE TYPE O_ADDRESS AS OBJECT (
            STREET_AND_NUMBER O_STREET,
            CITY VARCHAR2(30),
            POSTAL_CODE VARCHAR2(8)
            );
```

```
CREATE OR REPLACE TYPE O_PHONE AS OBJECT (
            HOMEPHONE          VARCHAR(15),
            BUSINESSPHONE      VARCHAR2(15),
            MOBILEPHONE        VARCHAR2(15));

CREATE OR REPLACE TYPE O_NAME AS OBJECT (
            FIRSTNAME          VARCHAR2(15),
            MIDDLE_INITIAL     VARCHAR2(2),
            LASTNAME           VARCHAR2(15));


CREATE OR REPLACE TYPE O_PERSON AS OBJECT (
            PNAME              O_NAME,
            PPHONE             O_PHONE,
            PADDRESS           O_ADDRESS);



--WANT TO START AFRESH
--Use these definitions. Please note all objects now start with OT_

CREATE OR REPLACE TYPE OT_STREET AS OBJECT (
            SNAME VARCHAR2(30),
            SNUMBER NUMBER,
            FLATNUMBER VARCHAR2(5));



CREATE or REPLACE TYPE OT_ADDRESS AS OBJECT (
            STREET_AND_NUMBER OT_STREET,
            CITY VARCHAR2(30),
            POSTAL_CODE VARCHAR2(8)
            );



CREATE OR REPLACE TYPE OT_PHONE AS OBJECT (
            HOMEPHONE          VARCHAR(15),
            BUSINESSPHONE      VARCHAR2(15),
            MOBILEPHONE        VARCHAR2(15));

CREATE OR REPLACE TYPE OT_NAME AS OBJECT (
            FIRSTNAME          VARCHAR2(15),
            MIDDLE_INITIAL     VARCHAR2(2),
            LASTNAME           VARCHAR2(15));


CREATE OR REPLACE TYPE OT_PERSON AS OBJECT (
            PNAME              OT_NAME,
            PPHONE             OT_PHONE,
            PADDRESS           OT_ADDRESS);
```

Exercise

6. Create a student supertype object type called Student_T with the following attributes: ID, Name, phone, course (choose appropriate DataTypes e.g. strings and integers). Make sure you can inherit from it!

7. Create the following subtypes (allow for future inheritance from these subtypes)
   - **i** PostGrad_T that has 2 attributes ResearchGrantAmount_T and researchArea
   - **ii** Undergrad that has two attributes academic year (e.g.1, 2,3) and status (e.g. FT or PT)

8. Create an object table called CollegeStudents based on Student_T

9. Insert 2 postgrad and 2 undergrad students into the table

10. Write a query that returns name of students that are postgrad. Amend the query to provide the total number of students in the table

2.3 Primary Keys and Constraints

Even though an object-relational database maintains object IDs for all objects (OIDs) (i.e., for types, row objects, column objects), it is still a good idea to use primary keys for some tables. The following statement shows the object IDs. Obviously they are too long and would be too difficult to remember to be used directly by users. You can view OIDs using the startement below but do not try accessing them beyond this!

```
SELECT SYS_NC_OID$ FROM CONTACTS_TABLE;
```

Object tables can be altered so that they have primary keys. Let us assume job_table is an object table and it contains an attribute job_ID

```
ALTER TABLE job_table
ADD (CONSTRAINT jobIDPK PRIMARY KEY (job_ID));
```

In this case "jobIDPK" is the name of the constraint whereas job_ID is the name of an actual attribute in job_table object table. If job_ID contains duplicates, then the alter statement produces an error.

Exercises

11. Alter the contacts_table (created above) that it has a primary key. The first_name, middle_initial and last_name (i.e. an object type you created last week – check your definition!) is the composite primary key for the contacts_table. Here is the ALTER command to add the Primary Key. Explain why the . notation is used.

```
ALTER TABLE contacts_table
ADD CONSTRAINT contactsPK PRIMARY KEY (PNAME.FIRSTNAME,
PNAME.MIDDLE_INITIAL, PNAME.LASTNAME);
```

12. Now alter the job_table (from last week's lab exercise) so that it job_id is the primary key.

13. Please also add NOT NULL constraints for "jobtitle" column in the job_table. Use ALTER TABLE commands to achieve this.

    [For reference purposes. You should have created from last lab:

```
CREATE TYPE O_JOB AS OBJECT (
JOBTITLE VARCHAR(20),
JOB_ID INTEGER,
SALARY_AMOUNT INTEGER,
YEARS_OF_EXPERIENCE INTEGER );

CREATE TABLE JOB_TABLE OF O_JOB; ]
```

14. Design 2 object types
    a. **TeamType** – whose attributes are team_name, year_established, league_name, and the home_stadium
    b. **PlayerType**– whose attributes are player_name, age, nationality, and sign_on_fee, playing_position

15. Create 1 **object** table called **Team** based on the **TeamType** object type you have just created. **Note** also team_name is the primary key of the Team table and the remaining attributes are mandatory.

16. Create a Relational Table called UEFAPlayer that has columns UEFA_ID (integer) UEFAPlayerStatus(varchar), playerDetails(playerType). Note: UEFA_ID (primary key,) player_name, age, nationality, and playing_position are all mandatory).

17. Insert 2 rows into each table.

18. Write a query that displays the team_name, year_established from Team for a particular team.

19. Write a query that displays UEFA_ID, player_name, age and playing_position from UEFAPlayer for a particular named player.