2.1 Information about object types

As you will have noticed by now, there is no information about types in the user catalog ("select * from cat"). The following select statements can be used to list user types and their attributes.

```
select * from user_types;
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. 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 "person", the last word in the alter statement should be "CASCADE". That means that an alteration of "person" also applies to "person_table" and its objects.

```
ALTER TYPE person NOT FINAL CASCADE;
```

If "describe person" now produces an error message, you need to logout of SQLPLUS and login again. This is so that Oracle can update the altered type and its objects.

Next, a subtype "employee" can be generated under "person". This subtype can itself be either NOT FINAL or FINAL. A subtype inherits all columns from its supertype but can also have additional columns, which are declared within the brackets.

```
CREATE TYPE employee UNDER person () NOT FINAL ;
/
```

The "IS OF" clause can be used to check the type of objects. The following statement selects all employees who are also a person, i.e., it selects all rows in employee_table.

```
SELECT value(p) FROM employee_table p WHERE value(p) IS OF (person);
```

Exercise

- 11. Create a type "employee" under "person" that has an additional attribute "emp_ID" of type INT. Create it as NOT FINAL.
- 12. Create a corresponding employee_table and insert 3 rows into it.

2.3 Primary Keys

Even though an object-relational database maintains object IDs for all objects (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.

```
select SYS_NC_OID$ from person_table;
```

Object tables can be altered so that they have primary keys:

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

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

Exercises

- 13. Alter both the employee_table and the job_table (from exercise 4) so that they have primary keys.

2.4 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 (
    employee REF employee,
    position REF 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 (
    employee REF employee SCOPE IS employee_table,
    position REF 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.

```
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:

- 14. Create the employment table as described above and insert 3 rows into it. (The insert statement above will probably not work for your database. You need to use the same attribute names that you used in your tables from exercises 4 and 10.)

- 15. What does "Select * from employment" show? Print the complete information from table "employment" in two formats:
  First, print 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.
  Second, print it in a way so that it only shows the data, not the types.

- 16. Using the employment table, print the names of all employees whose salary is larger than 20000. Print the jobtitles of all employees who live in Edinburgh. (Note that in contrast to relational databases you do not need a join to do this!)

- 17. Sit back for a moment and 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?

- 18. Create an object type "project" with columns: "project_id", "project_leader", "project_title" where "project_leader" is a REF to person. Create a corresponding object table "project_table". Alter the table so that project_id is a primary key. Insert 3 rows into "project_table".

- 19. Create a table "project_membership" with two columns "project_id" and "member" so that the project IDs are references to projects but can only be filled with data from project_table. The members are references to person and can only be filled with data from employee_table. Insert 3 rows into this table. Select all rows from this table using dereferencing of the two columns.