**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\_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. 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 ;  
      /

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 contacts\_table.

      SELECT value(p) FROM employee\_table p WHERE value(p) IS OF (person);

|  |
| --- |
| **Exercise**   1. Firstly, examine and create the objects below, Now, create a type "employee" under "individual" that has an additional attribute "emp\_ID" of type INT. Create it as NOT FINAL. First you need to alter your individual type to allow inheritance:   ALTER TYPE INDIVIDUAL NOT FINAL CASCADE;   1. Create a corresponding CONTACTS\_TABLE and insert 2 rows( 2 of which are “employee” and 2 of which are “individual”) into it. 2. Write a query that returns contacts that are employees ONLY. |

Exercise 1:

CREATE or replace TYPE ADDRESS\_DETAILS AS OBJECT (

STREeT varchar2(50),

CITY VARCHAR2(30),

POSTAL\_CODE VARCHAR2(8),

PROVINCE VARCHAR2(30),

COUNTRY VARCHAR2(30));

/

CREATE OR REPLACE TYPE CONTACT\_NUMBERS AS OBJECT (

HOMEPHONE VARCHAR(15),

BUSINESSPHONE VARCHAR2(15),

MOBILEPHONE VARCHAR2(15));

/

CREATE OR REPLACE TYPE NAME\_DETAILS AS OBJECT (

FIRSTNAME VARCHAR2(15),

MIDDLE\_INITIAL VARCHAR2(6),

LASTNAME VARCHAR2(15));

/

CREATE OR REPLACE TYPE INDIVIDUAL AS OBJECT (

INDIV\_NAME NAME\_DETAILS,

PPHONE CONTACT\_NUMBERS,

PADDRESS ADDRESS\_DETAILS);

/

ALTER TYPE INDIVIDUAL NOT FINAL CASCADE;

CREATE TYPE employee UNDER individual

(emp\_id INT) NOT FINAL;

Exercise 2: (

create table CONTACTS\_table of individual;

INSERT INTO CONTACTS\_TABLE VALUES (

INDIVIDUAL( NAME\_DETAILS('JOHN', 'R', 'SMITH'),

CONTACT\_NUMBERS('123-4567', NULL,'73746-56'),

ADDRESS\_DETAILS('1 MARY ST','DUBLIN','11A', 'LEINSTER','IRELAND'))

);

/

INSERT INTO CONTACTS\_TABLE VALUES (

INDIVIDUAL( NAME\_DETAILS('MARY', 'R', 'MILLER'),

CONTACT\_NUMBERS('123-4567', '2334-83838','73746-56'),

ADDRESS\_DETAILS('10 GRAFTON STREET','DUBLIN','81A', 'LEINSTER','IRELAND'))

);

/

INSERT INTO CONTACTS\_TABLE VALUES (

EMPLOYEE( NAME\_DETAILS('MARY', 'S', 'MILLER'),

CONTACT\_NUMBERS('354-5643', '453-5746','73346-56'),

ADDRESS\_DETAILS('344 GRAFTON ST.','DUBLIN', 'SD2','LEINSTER','IRELAND'),89898)

);

/

INSERT INTO CONTACTS\_TABLE VALUES (

EMPLOYEE( NAME\_DETAILS('PAUL', 'B', 'BYRNE'),

CONTACT\_NUMBERS('354-564333', '453-53636','89346-56'),

ADDRESS\_DETAILS('844 VERNON ST.','DUBLIN', '34D2','LEINSTER','IRELAND'),898989)

);

/

Exercise 3:

SELECT value(p) FROM CONTACTS\_table p WHERE value(p) IS OF (EMPLOYEE);

**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**   1. Alter both the contacts\_table and the job\_table (from last week lab exercise) so that they have primary keys. The first\_name, middle\_initial and last\_name (i.e. check the object you created last week – check your definition ) is the composite primary key for the contacts\_table. If you have time also add NOT NULL constraints for “jobtitle” column in the job\_table |

Exercise 4:

ALTER TABLE job\_table

ADD (CONSTRAINT jobID PRIMARY KEY (job\_ID));

ALTER TABLE contacts\_table

ADD CONSTRAINT contactsPK PRIMARY KEY (INDIV\_NAME.FIRSTNAME, INDIV\_NAME.MIDDLE\_INITIAL, INDIV\_NAME.LASTNAME);

ALTER TABLE job\_table

MODIFY(jobtitle NOT NULL);

Or

ALTER TABLE job\_table ADD CONSTRAINT job\_table\_cons1

CHECK(jobtitle IS NOT NULL);

**2.4 References or REFs**

References (REF) can be used instead of foreign keys in many-to-one relationships. Note that the (UNSCOPED) 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:**   1. CREATE an employee\_table and insert rows as follows   (**note** please refer to your definition of employee and change THE insert statement accordingly)  CREATE TABLE EMPLOYEE\_TABLE OF employee;  INSERT INTO EMPLOYEE\_TABLE VALUES (  EMPLOYEE( NAME\_DETAILS('PAUL', 'B', 'BYRNE'),  CONTACT\_NUMBERS('354-56433', '3453-5746','373346-56'),  ADDRESS\_DETAILS('123 VERNON ST.','DUBLIN','S212', NULL,'IRELAND'),1));  /  INSERT INTO EMPLOYEE\_TABLE VALUES (  EMPLOYEE( NAME\_DETAILS('MARY', 'S', 'MILLER'),  CONTACT\_NUMBERS('354-5643', '453-5746','73346-56'),  ADDRESS\_DETAILS('22 GRAFTON ST.','DUBLIN','A212', NULL,'IRELAND'),2)  );  Commit;  6. Now create the employment table as described above and insert 3 rows into it.  7 .What does SELECT \* FROM EMPLOYMENT show? 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. (ii)Second, write a query that shows the following data for employees: firstname middle\_initial, lastname, mobile number home phone number and business phone number and job title not the types.  8. Using the employment table, print the names of all employees whose salary is larger  than 20000. What are the unique job titles of all employees who live in Dublin. (Note that in  contrast to relational databases you do not need a join to do this!)  9. 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?  10 . Let us try IS DANGLING predicate. First, delete a job from the employee\_table, one that you referenced in employment\_table in exercise 6. Now use the IS DANGLING predicate on the employment\_table to find rows that are point to a job object that now does not exist.  Note you will have to change the SQL query to suit your data. Try the IS NOT  DANGLING predicate. |

Exercise 5

See above for CREATE statement and INSERT statements. **Note**: You may have to make edits to your INSERT statement depending on your object type definitions

Exercise 6:

CREATE TABLE employment (

employee REF employee SCOPE IS employee\_table,

position REF job SCOPE IS job\_table);

INSERT INTO employment

SELECT REF(e), REF(j)

FROM job\_table j, employee\_table e

WHERE e.emp\_ID = 1

AND j.job\_ID = 1;

INSERT INTO employment

SELECT REF(e), REF(j)

FROM job\_table j, employee\_table e

WHERE e.emp\_ID = 2

AND j.job\_ID = 3;

INSERT INTO employment

SELECT REF(e), REF(j)

FROM job\_table j, employee\_table e

WHERE e.emp\_ID = 1

AND j.job\_ID = 2;

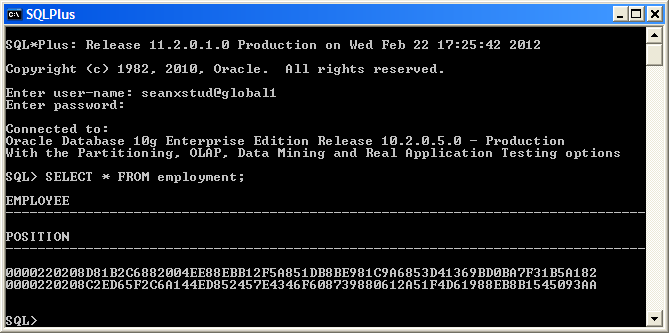
Commit;

---

Exercise 7:

SELECT \* FROM employment.

Note: SQL DEVELOPER java tool dereferences on the fly (behind the scenes). However, if you use a SQLPLUS or accessing the dbms from a programming API dereferencing does not occur and you are left with pointers to the objects in the job\_table and employee\_table. For example, here is some sample data where REF values are displayed



(i)select DEREF(e.employee), DEREF(e.position) from employment e;

(ii) select e.employee.indiv\_name.firstname, e.employee.indiv\_name.middle\_initial, e.employee.indiv\_name.lastname,

e.employee.pphone.homephone, e.employee.pphone.businessphone,

e.employee.pphone.mobilephone,

e.position.jobtitle

from employment e;

---

Exercise 8:

select p.employee.indiv\_name.firstname, p.employee.indiv\_name.middle\_initial, p.employee.indiv\_name.lastname

from employment p where p.position.salary\_amount > 20000;

select DISTINCT(e.position.jobtitle) from employment e

where e.employee.paddress.city = 'DUBLIN';

---

Exercise 10

DELETE FROM job\_table

WHERE job\_ID=2;

select e.employee.indiv\_name.firstname, e.employee.indiv\_name.middle\_initial, e.employee.indiv\_name.lastname

from employment e

WHERE e.position IS DANGLING;