



RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas
tehnoloģijas fakultāte

Informācijas tehnoloģijas institūts

Technology of Large Data Bases



ORACLE

Castillo Torres Miguel Eduardo

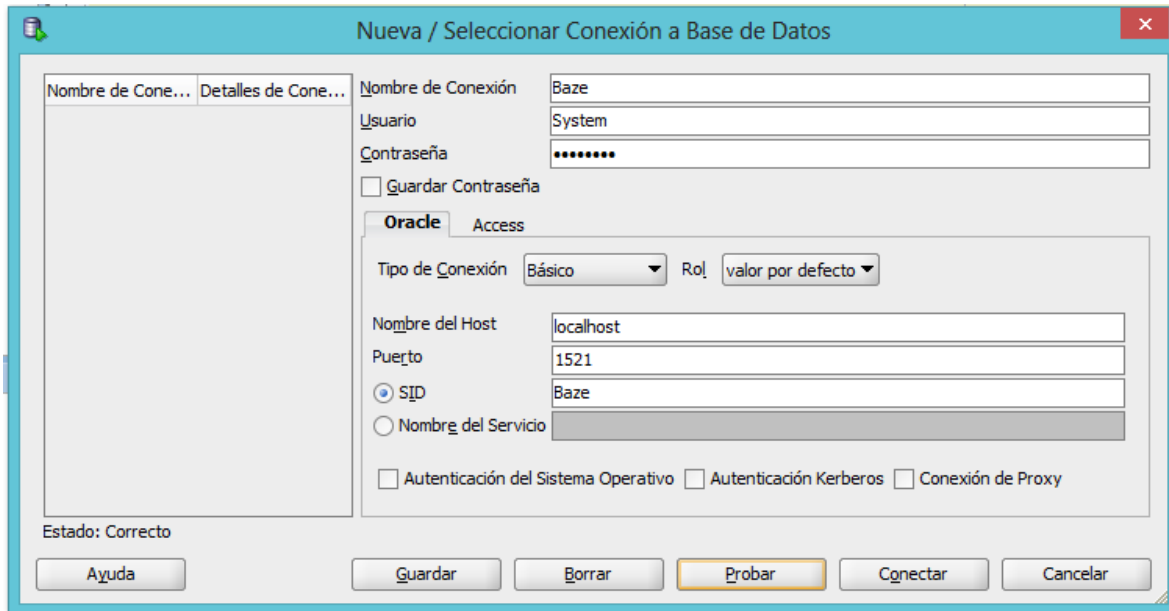
First practical work:

Relational-Object DB Structures

Semester fall 2015

INTRODUCTION

Just a reminder: Before installing SQL Developer, make sure you have already installed ORACLE 10 or later and JDK. Once you have done that, you may proceed to create a new connection, just remember your user name, password and SID that you write at the beginning of the installation.

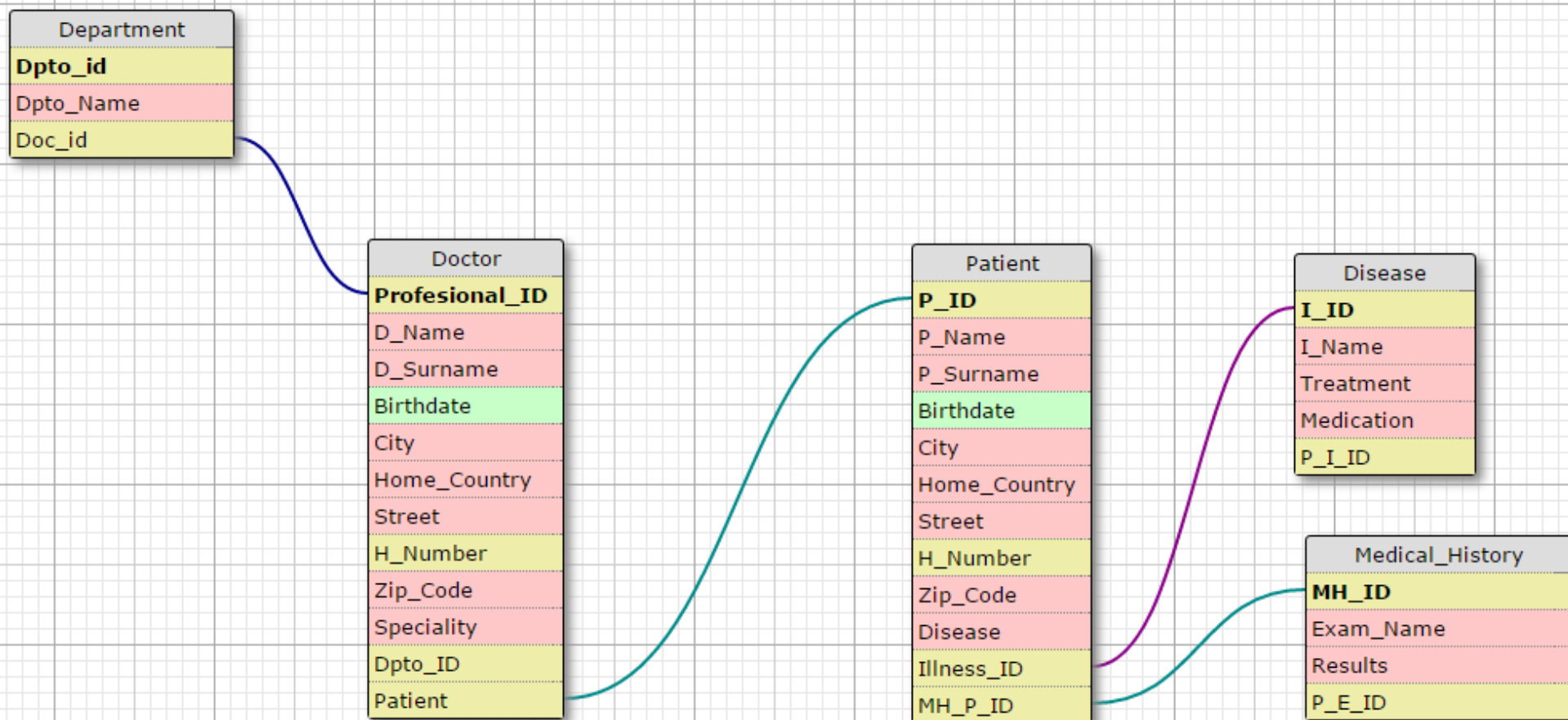


So the question, why using Oracle?

- Designed to represent knowledge in a distributed world
- A method to decompose knowledge into small pieces, with rules about the semantics of those pieces
- RDF data is self-describing; it “means” something
- Allows you to model and integrate DBMS schemas
- Allows you to integrate data from different sources without custom programming
- Supports decentralized data management
- Infer implicit relationships across data

Possible answers to motivate ourselves to use Oracle.

In the next pages, as a little practical work to reinforce the first topics of the course, I will create a new object-relational database based in the next relational database schema.



Contents

INTRODUCTION	2
Table with row type objects	5
1 Creation	5
2 Data Input.....	5
3 Select	6
4 Metadata	8
Table with object column	9
1 Creation	9
2 Data input.....	9
3 Select	10
4 Metadata	12
Table with object collection	13
1 Creation	13
2 Data input.....	14
3 Select	14
4 Metadata	15
Object view.....	17
Object references.....	20
Use of special object functions in SQL	24
1 Comparing collections	25
2 Operator SUBMULTISET OF	25
3 Operator MEMBER OF.....	26
4 Operator IS A SET	27
5 CARDINALITY	28
Conclusion	29
References.....	30

Table with row type objects

As we saw in the introduction, we need to convert the relational database into a Object-Relational Database. So, our first approach will be converting the table “Department” into a row type table.

1 Creation

If we write the following code, we will create our first row type table.

```
CREATE TYPE Department AS OBJECT(  
    DPTO_ID NUMBER,  
    DPTO_NAME VARCHAR2(20),  
    DOC_ID NUMBER  
);
```

```
CREATE TABLE Departments OF Department;
```

In the first part, we define our object “Department” which will include the ID of the hospital department, the name of the department and the correspondent ID of the doctor(s) which work in the department.

In the second part, what we do is create a table called “Departments” which is made of row type objects called “Department”.

```
TYPE Department compilado  
table DEPARTMENTS creado.
```

2 Data Input

The next step is to fill the new table with data, we do it as follow.

```
INSERT INTO Departments values(Department(1, 'Oncology'));  
INSERT INTO Departments values(Department(2, 'Family Medicine'));  
INSERT INTO Departments values(Department(3, 'Neurology'));  
INSERT INTO Departments values(Department(4, 'Cardiology'));  
INSERT INTO Departments values(Department(5, 'X-Rays'));  
INSERT INTO Departments values(Department(6, 'Pediatrics'));  
INSERT INTO Departments values(Department(7, 'Nutriology'));  
INSERT INTO Departments values(Department(8, 'Urgencies'));  
INSERT INTO Departments values(Department(9, 'Pathology'));
```

```
INSERT INTO Departments values(Department(10, 'Diagnostic Center'));
INSERT INTO Departments values(Department(11, 'Mathernity'));
INSERT INTO Departments values(Department(12, 'MRI'));
```

3 Select

And, to see what we have inserted, we use a simply command of “SELECT * FROM”.

```
SELECT *
FROM departments;
```

What we see in the screen is an ordinary relational database.

DPTO_ID	DPTO_NAME
1	Oncology
2	Family Medicine
3	Neurology
4	Cardiology
5	X-Rays
6	Pediatrics
7	Nutriology
8	Urgencies
9	Pathology
10	Diagnostic Center
11	Mathernity
12	MRI

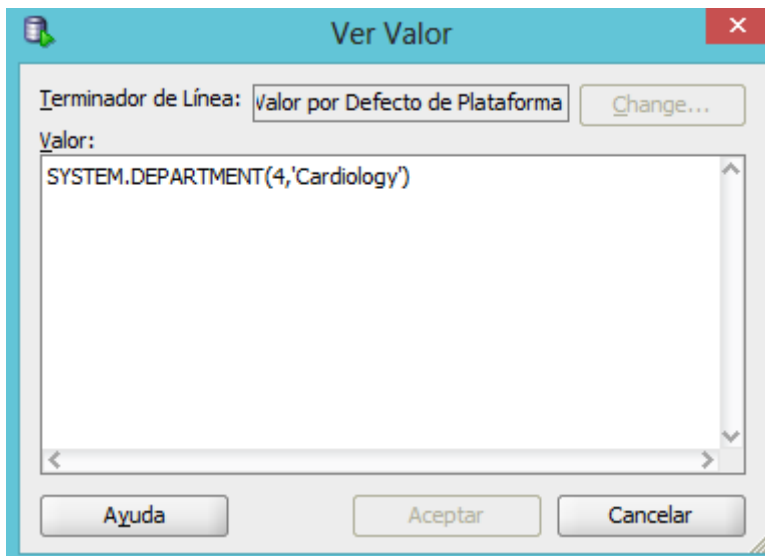
But, with a slight modification in the previous code, we will get a different result.

```
SELECT VALUE(A)
FROM Departments A;
```

This code, shows in the screen the row object type table:

VALUE(A)
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]
[SYSTEM.DEPARTMENT]

We only see the “VALUE(A)”, whereas a double click over the value we want to see will reveal the information.




So, the value is inside this “SYSTEM.DEPARTMENT” type.

Finally, if we want to select exact data, we use the following sentence.

```
SELECT value(A)
FROM Departments A
WHERE value(A).DPTO_NAME = 'Mathernity';
```

Which will show us the following.

VALUE(A)
SYSTEM.DEPARTMENT(11,'Mathernity') 

Again, we have two ways of asking to the computer to throw us a result, the previous one show the result as an object, and the next one will show it as a relational table.

```
SELECT *
FROM Departments
WHERE Departments.DPTO_NAME = 'Mathernity';
```

DPTO_ID	DPTO_NAME
11	Mathernity

4 Metadata

First, we have to define what Metadata is: Metadata is "data about data". Two types of metadata exist: structural metadata and descriptive metadata.

The main purpose of metadata is to facilitate in the discovery of relevant information, more often classified as resource discovery. Metadata also helps organize electronic resources, provide digital identification, and helps support archiving and preservation of the resource. Metadata assists in resource discovery by "allowing resources to be found by relevant criteria, identifying resources, bringing similar resources together, distinguishing dissimilar resources, and giving location information."

So, now that we know what Metadata is, we can look for this data about our database.

In SQL Developer, what we do is to go to the left panel, and in CONNECTION>TABLES>Departments and double-click over it, so we can see the metadata of our DB.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	DPTO_ID	NUMBER	Yes	(null)	1	(null)
2	DPTO_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)

Table with object column

The second modification in our initial schema is to convert the table “Doctor” into an object column type.

1 Creation

```
CREATE TYPE PERSONAL_DATA AS OBJECT(  
    D_NAME VARCHAR2(30),  
    D_SURNAME VARCHAR2(30),  
    BIRTHDAY DATE  
);  
  
CREATE TYPE ADDRESS AS OBJECT(  
    HOME_COUNTRY VARCHAR2(20),  
    CITY VARCHAR2(20),  
    STREET VARCHAR(20),  
    H_NUMBER NUMBER,  
    ZIP_CODE VARCHAR2(10)  
);  
  
CREATE TABLE DOCTOR(  
    PROFESIONAL_ID NUMBER,  
    D_DATA PERSONAL_DATA,  
    D_ADDRESS ADDRESS,  
    DPTO_ID NUMBER,  
    PATIENT_ID NUMBER  
);
```

The previous code is divided in two principal parts. The first part is about making objects: Personal data and address; and the second part is about creating the table. As we read in the code, the objects are a new type of data, so now we can fill, for example, the field “D_DATA” with not only a simple string of characters (VARCHAR) but with a group (Objects) of types which will define all the field.

5 Data input

```

INSERT INTO DOCTOR VALUES(1, PERSONAL_DATA('Anakin', 'Skywalker',
(TO_DATE('1984/04/10','yyyy/mm/dd'))), ADDRESS('Mexico', 'Mexico', 'One', 20,
'1234'), 3, 1);

INSERT INTO DOCTOR VALUES(2, PERSONAL_DATA('Obi-Wan', 'Kenobi',
(TO_DATE('1988/04/25','yyyy/mm/dd'))), ADDRESS('USA', 'Philadelphia', 'Two', 15,
'OWK12'), 2, 2);

INSERT INTO DOCTOR VALUES(3, PERSONAL_DATA('Yoda', 'Ignari',
(TO_DATE('1975/02/21','yyyy/mm/dd'))), ADDRESS('Canada', 'Toronto', 'Ari', 45,
'89A'), 1, 3);

INSERT INTO DOCTOR VALUES(4, PERSONAL_DATA('Mace', 'Windu',
(TO_DATE('1980/01/20','yyyy/mm/dd'))), ADDRESS('USA', 'California', 'But', 2,
'12540'), 5, 4);

INSERT INTO DOCTOR VALUES(5, PERSONAL_DATA('Qui-Gon', 'Jinn',
(TO_DATE('1969/12/01','yyyy/mm/dd'))), ADDRESS('Mexico', 'Guadalajara',
'Manzana', 9, '1254'), 6, 5);

INSERT INTO DOCTOR VALUES(6, PERSONAL_DATA('Palpatine', 'Sidious',
(TO_DATE('1940/06/01','yyyy/mm/dd'))), ADDRESS('Spain', 'Granada', 'Esmu', 67,
'SGE12'), 11, 6);

INSERT INTO DOCTOR VALUES(7, PERSONAL_DATA('Luke', 'Skywalker',
(TO_DATE('1981/07/23','yyyy/mm/dd'))), ADDRESS('Mexico', 'Mexico', 'Pera', 12,
'6789'), 4, 7);

INSERT INTO DOCTOR VALUES(8, PERSONAL_DATA('Leia', 'Organa',
(TO_DATE('1981/07/23','yyyy/mm/dd'))), ADDRESS('Latvia', 'Riga', 'Z.A.Meirovica
Blvd', 1, 'LV-1543'), 2, 8);

INSERT INTO DOCTOR VALUES(9, PERSONAL_DATA('Han', 'Solo',
(TO_DATE('1961/08/10','yyyy/mm/dd'))), ADDRESS('Latvija', 'Riga', 'Azenes', 8, 'LV-
1455'), 11, 9);

INSERT INTO DOCTOR VALUES(10, PERSONAL_DATA('Maria', 'DZY',
(TO_DATE('1995/09/27','yyyy/mm/dd'))), ADDRESS('Kazakhstan', 'Pavldar', 'Odin St.',
12, '7689'), 12, 10);

INSERT INTO DOCTOR VALUES(11, PERSONAL_DATA('Dinara', 'BlaBla',
(TO_DATE('1995/06/29','yyyy/mm/dd'))), ADDRESS('Kazakhstan', 'Pavldar', 'Udens',
15, '7689'), 12, 11);

INSERT INTO DOCTOR VALUES(12, PERSONAL_DATA('Dina', 'Lol',
(TO_DATE('1995/11/08','yyyy/mm/dd'))), ADDRESS('Kazakhstan', 'Pavldar',
'SnegBoulevard', 31, '7690'), 10, 12);

```

6 Select

Again, there are many ways for using the “SELECT” command in SQL, so I am going to list some of them.

With the simple “SELECT ALL” we will get the complete table with the objects.

```
SELECT *
FROM DOCTOR;
```

PROFESIONAL_ID	D_DATA	D_ADDRESS	DPTO_ID	PATIENT_ID
1	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	3	1
2	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	2	2
3	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	1	3
4	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	5	4
5	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	6	5
6	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	11	6
7	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	4	7
8	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	2	8
9	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	11	9
10	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	12	10
11	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	12	11
12	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	10	12

If we want to see the information of the objects, with a double-click action the computer shows the hidden data.

PROFESIONAL_ID	D_DATA	D_ADDRESS	DPTO_ID	PATIENT_ID
1	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	3	1
2	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	2	2
3	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	1	3
4	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	5	4
5	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	6	5
6	['Palpatine','Sidious','1940-06-01 00:00:00.0']	[SYSTEM.ADDRESS]	11	6
7	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	4	7
8	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	2	8
9	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	11	9
10	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	12	10
11	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	12	11
12	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	10	12

With a conditional select we can choose only some data, as the following examples.

```
SELECT B.D_DATA.D_NAME
```

```
FROM DOCTOR B
WHERE DPTO_ID = 12;
```

D_DATA.D_NAME
Maria
Dinara

We get the names of the “Doctors” working in DPTO_ID 12.

```
SELECT b.PROFESIONAL_ID, B.D_DATA.D_NAME, B.DPTO_ID
FROM DOCTOR B
WHERE B.D_ADDRESS.HOME_COUNTRY = 'Mexico';
```

PROFESIONAL_ID	D_DATA.D_NAME	DPTO_ID
1	Anakin	3
5	Qui-Gon	6
7	Luke	4

In this example what we see are the ID’s, names and departments of the doctors which live in Mexico.

7 Metadata

What we observe in the metadata table of SQL Developer is.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PROFESIONAL_ID	NUMBER	Yes	(null)	1	(null)
2	D_DATA	PERSONAL_DATA	Yes	(null)	2	(null)
3	D_ADDRESS	ADDRESS	Yes	(null)	3	(null)
4	DPTO_ID	NUMBER	Yes	(null)	4	(null)
5	PATIENT_ID	NUMBER	Yes	(null)	5	(null)

There we can notice the objects: “PERSONAL_DATA” and “ADDRESS”.

Table with object collection

If we remember the original schema, we can say that there are three tables left, so, what are we going to do with them?, we are going to make one object collection table. This, in order to complete our task and to manage easier big data in our database.

1 Creation

The creation of an object collection implies a complex coding, but not difficult as we will see next.

```
CREATE TYPE ILLNESS AS OBJECT(  
    I_ID NUMBER,  
    I_NAME varchar2(20),  
    TREATMENT VARCHAR2(50),  
    MEDICATION VARCHAR2(30)  
);  
  
CREATE TYPE ILLNESS_S AS table of ILLNESS;  
  
CREATE TYPE MEDICAL_HISTORY AS OBJECT(  
    MH_ID NUMBER,  
    EXAM_NAME VARCHAR2(20),  
    RESULTS VARCHAR2(50)  
);  
  
CREATE TYPE MEDICAL_HISTORIES AS TABLE OF MEDICAL_HISTORY;  
  
CREATE TABLE PATIENT(  
    P_ID NUMBER,  
    P_DATA PERSONAL_DATA,  
    P_ADDRESS ADDRESS,  
    P_D ILLNESS_S,  
    P_MH MEDICAL_HISTORIES  
)  
nested table P_D store as PATIENT_ILLNES,
```

```
nested table P_MH store as PATIENT_M_H;
```

Here what we are doing is coding in three steps:

Firstly, we create the object “ILLNES” with their each attributes; Secondly, we create a new type from “ILLNESS” as a table.

The same thing we do with the object “MEDICAL_HISTORY”

Finally, we create our table as we have been doing before, but with a little difference: we add at the end the “nested table” command, so we can nested the tables we have made before: ILLNES_S and MEDICAL_HISTORIES.

8 Data input

The data insertion is now more ambitious than in previous tables because of the big data I am inserting, but with the needed caution we can manage to insert all data.

```
INSERT INTO PATIENT VALUES(1, PERSONAL_DATA('Peter', 'Petrelli',
(TO_DATE('1990/01/01','yyyy/mm/dd'))),
ADDRESS('USA','Manhattan','One',1,'9043'), ILLNESS_S(ILLNESS(1, 'Brain
Pathology','Psychiatric','NON')), MEDICAL_HISTORIES(MEDICAL_HISTORY(1, 'NON',
'0')));

INSERT INTO PATIENT VALUES(2, PERSONAL_DATA('Hiro', 'Nakamura',
(TO_DATE('1990/01/01','yyyy/mm/dd'))),
ADDRESS('Japan','Tokyo','Hirashi',43,'JP09'), ILLNESS_S(ILLNESS(2,
' Cancer','QuimicTherapy','NON')), MEDICAL_HISTORIES(MEDICAL_HISTORY(2,
'BLOOD', 'AB+')));

INSERT INTO PATIENT VALUES(3, PERSONAL_DATA('Nathan', 'Petrelli',
(TO_DATE('1990/01/01','yyyy/mm/dd'))), ADDRESS('USA','New-York','Big
Apple',1,'USY78'), ILLNESS_S(ILLNESS(3, 'Blood Problems','NON','NON'), ILLNESS(4,
'Hits','NON','NON')), MEDICAL_HISTORIES(MEDICAL_HISTORY(3, 'BLOOD', 'O+')));

INSERT INTO PATIENT VALUES(4, PERSONAL_DATA('Ando', 'Nagasaki',
(TO_DATE('1990/01/01','yyyy/mm/dd'))), ADDRESS('Japan','Osaka','Haghen',
4,'4532'), ILLNESS_S(ILLNESS(5, 'Cancer','NON','NON')),
MEDICAL_HISTORIES(MEDICAL_HISTORY(4, 'Blood', 'O+'), MEDICAL_HISTORY(5,
'Cholesterol', '5')));

INSERT INTO PATIENT VALUES(5, PERSONAL_DATA('Lea', 'Quinto',
(TO_DATE('1990/01/01','yyyy/mm/dd'))), ADDRESS('Italy','Florence','Ten',1,'I-90'),
ILLNESS_S(ILLNESS(6, 'Baby Problems','NON','XZ')),
MEDICAL_HISTORIES(MEDICAL_HISTORY(6, 'NON', 'NON')));
```

9 Select

```
SELECT *
FROM PATIENT;
```

P_ID	P_DATA	P_ADDRESS	P_D	P_MH
1	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	SYSTEM.ILLNESS_S([SYSTEM.ILLNESS])	SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
2	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	SYSTEM.ILLNESS_S([SYSTEM.ILLNESS])	SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
4	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	SYSTEM.ILLNESS_S([SYSTEM.ILLNESS])	SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY], [SYSTEM.MEDICAL_HI
5	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	SYSTEM.ILLNESS_S([SYSTEM.ILLNESS])	SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
3	[SYSTEM.PERSONAL_DATA]	[SYSTEM.ADDRESS]	SYSTEM.ILLNESS_S([SYSTEM.ILLNESS], [SYSTEM.ILLNESS])	SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])

Our simply query show us the complete table collection, as we see the screen is not big enough to show the complete data.

We can try to go further in our collection, with the next sentence

```
SELECT C.P_MH
FROM PATIENT C;
```

P_MH
SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY], [SYSTEM.MEDICAL_HISTORY])
SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])
SYSTEM.MEDICAL_HISTORIES([SYSTEM.MEDICAL_HISTORY])

This show us the MEDICAL_HISTORIES' collections, but we cannot enter to the data with a simple SELECT command because an object collection table is not order as a relational table, so we need to use a function that “converts” this collection of objects in a table in order to achieve the data. The function we are going to use is TABLE().






For example, we want to see name of the persons and where they live, which are type of blood ‘O+’, so we do the following:

```
SELECT D.P_DATA.D_NAME, D.P_ADDRESS.HOME_COUNTRY
FROM PATIENT D, TABLE(D.P_MH) E
WHERE E.RESULTS = 'O+';
```

We achieve the name and country as we have done before, but to achieve the “RESULTS” in the object MEDICAL_HISTORIES, we need to convert that collection into a table with the function TABLE(). Then we can get the next result, that is what we were looking.

P_DATA.D_NAME	P_ADDRESS.HOME_COUNTRY
Ando	Japan
Nathan	USA

10 Metadata

	 COLUMN_NAME	 DATA_TYPE	 NULLABLE	DATA_DEFAULT	 COLUMN_ID	 COMMENTS
1	P_ID	NUMBER	Yes	(null)	1	(null)
2	P_DATA	PERSONAL_DATA	Yes	(null)	2	(null)
3	P_ADDRESS	ADDRESS	Yes	(null)	3	(null)
4	P_D	ILLNESS_S	Yes	(null)	4	(null)
5	P_MH	MEDICAL_HISTORIES	Yes	(null)	5	(null)

Object view

Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data - of either built-in or user-defined types - stored in the columns of relational or object tables in the database.

Object views provide the ability to offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that doesn't have attributes containing sensitive data and doesn't have a deletion method.

Object views allow the use of relational data in object-oriented applications. They let users

Try object-oriented programming techniques without converting existing tables.

Convert data gradually and transparently from relational tables to object-relational tables.

Use legacy RDBMS data with existing object-oriented applications.

The procedure for defining an object view is:

1. Define an object type, where each attribute of the type corresponds to an existing column in a relational table.
2. Write a query that specifies how to extract the data from the relational table. Specify the columns in the same order as the attributes in the object type.
3. Specify a unique value, based on attributes of the underlying data, to serve as an object identifier, enabling you to create pointers (REFs) to the objects in the view.

So, to continue with our database example in a hospital, we want to add a new table, this one is a relational one.

```
CREATE TABLE EMPLOYEE_TABLE (  
    E_ID    NUMBER (5),  
    E_NAME  VARCHAR2 (20),  
    E_SALARY NUMBER (9,2),  
    E_AREA  VARCHAR2 (20)  
);  
  
CREATE TYPE EMPLOYEE_T AS OBJECT (  
    EM_ID    NUMBER (5),  
    EM_NAME  VARCHAR2 (20),  
    EM_SALARY NUMBER (9,2),  
    EM_AREA  VARCHAR2 (20)  
);
```

```
CREATE VIEW EMPLOYEE_VIEW OF EMPLOYEE_T
WITH OBJECT IDENTIFIER (EM_ID) AS
SELECT F.E_ID, F.E_NAME, F.E_SALARY, F.E_AREA
FROM EMPLOYEE_TABLE F
WHERE E_AREA = 'SOCIAL SERVICE';
```

```
insert into EMPLOYEE_TABLE values(1,'John',1000.00,'DOCTOR');
insert into EMPLOYEE_TABLE values(2,'Robert',900.00,'OFFICE');
insert into EMPLOYEE_TABLE values(3,'James',2000.00,'CONSULTANT');
insert into EMPLOYEE_TABLE values(4,'Gisela',9000.00,'SOCIAL SERVICE');
insert into EMPLOYEE_TABLE values(5,'Martina',2000.00,'SOCIAL SERVICE');
```

```
SELECT *
FROM EMPLOYEE_VIEW;
```

EM_ID	EM_NAME	EM_SALARY	EM_AREA
4	Gisela	9000	SOCIAL SERVICE
5	Martina	2000	SOCIAL SERVICE

What I did was first create a table with some data of the employee, then create an object with similar attributes of the table. After that, the view is created and we can insert our data.

With a select of the view we received what we wanted before introducing our data. In other words, an object view could work when we are repeating same queries again and again and again, because you only need to select the object view and run it.

The metadata of the table is the following.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	E_ID	NUMBER(5,0)	Yes	(null)	1	(null)
2	E_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	E_SALARY	NUMBER(9,2)	Yes	(null)	3	(null)
4	E_AREA	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)

And the metadata of the object view is.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	EM_ID	NUMBER(5)	Yes	(null)	1	(null)	YES	YES	YES
2	EM_NAME	VARCHAR2(20)	Yes	(null)	2	(null)	YES	YES	YES
3	EM_SALARY	NUMBER(9,2)	Yes	(null)	3	(null)	YES	YES	YES
4	EM_AREA	VARCHAR2(20)	Yes	(null)	4	(null)	YES	YES	YES

What we see then, is that the object view is a row type table created from the EMPLOYEE_T type.

But, for academical purpose lets make an object view from a nested table. Using again our database, we will create an Object View from the table (column type) Doctor.

```
CREATE TYPE DOCTORS_ADDRESS AS OBJECT(
  PROFESIONAL_ID NUMBER,
  ADDR ADDRESS
);

CREATE VIEW DEPT_VIEW OF DOCTORS_ADDRESS WITH OBJECT IDENTIFIER
(PROFESIONAL_ID) AS

  SELECT      H.PROFESIONAL_ID,      ADDRESS(H.D_ADDRESS.HOME_COUNTRY,
H.D_ADDRESS.CITY,      H.D_ADDRESS.STREET,      H.D_ADDRESS.H_NUMBER,
H.D_ADDRESS.ZIP_CODE) AS DOC_DPT_ADDR

  FROM DOCTOR H

  WHERE H.PROFESIONAL_ID > 5;

SELECT *
FROM DEPT_VIEW;
```

PROFESIONAL_ID	ADDR
6	SYSTEM.ADDRESS('Spain','Granada','Esmu',67,'SGE12')
7	[SYSTEM.ADDRESS]
8	[SYSTEM.ADDRESS]
9	[SYSTEM.ADDRESS]
10	[SYSTEM.ADDRESS]
11	[SYSTEM.ADDRESS]
12	[SYSTEM.ADDRESS]

As you see, what I did here was an object view in which I grouped the addresses of the doctors with ID greater than 5.

And their corresponding metadata is:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	PROFESIONAL_ID	NUMBER	Yes	(null)	1 (null)		YES	YES	YES
2	ADDR	ADDRESS ()	Yes	(null)	2 (null)		YES	YES	YES

Object references

A REF is a logical pointer or reference to a row object that you can construct from an object identifier (OID). You can use the REF to obtain, examine, or update the object. You can change a REF so that it points to a different object of the same object type hierarchy or assign it a null value. REFs are Oracle Database built-in data types. REFs and collections of REFs model associations among objects, particularly many-to-one relationships, thus reducing the need for foreign keys. REFs provide an easy mechanism for navigating between objects.

```
CREATE TYPE DOCTOR_TYPE AS OBJECT(  
    D_T_NAME VARCHAR2(10),  
    DOC_BOSS REF DOCTOR_TYPE  
);  
  
CREATE TABLE DOCTOR_OBJECT_TABLE OF DOCTOR_TYPE;  
  
INSERT INTO DOCTOR_OBJECT_TABLE VALUES(DOCTOR_TYPE('JOHN',NULL));  
INSERT INTO DOCTOR_OBJECT_TABLE VALUES(DOCTOR_TYPE('ADAM',NULL));  
INSERT INTO DOCTOR_OBJECT_TABLE VALUES(DOCTOR_TYPE('PHIL',NULL));  
INSERT INTO DOCTOR_OBJECT_TABLE VALUES(DOCTOR_TYPE('ANDY',NULL));  
  
INSERT INTO DOCTOR_OBJECT_TABLE  
    SELECT DOCTOR_TYPE ('MR. BOB', REF(I))  
    FROM DOCTOR_OBJECT_TABLE I  
    WHERE I.D_T_NAME = 'JOHN' ;  
  
INSERT INTO DOCTOR_OBJECT_TABLE  
    SELECT DOCTOR_TYPE ('MR. FYN', REF(I))  
    FROM DOCTOR_OBJECT_TABLE I  
    WHERE I.D_T_NAME = 'ANDY';  
  
COLUMN D_T_NAME FORMAT A10  
COLUMN DOC_BOSS FORMAT A50  
select * from DOCTOR_OBJECT_TABLE J;
```

D_T_NAME	DOC_BOSS
JOHN	(null)
ADAM	(null)
PHIL	(null)
ANDY	(null)
MR. BOB	SYSTEM.DOCTOR_TYPE('JOHN',NULL)
MR. FYN	[SYSTEM.DOCTOR_TYPE]

What we see is that MR. BOB is the boss of Doctor JOHN.

In metadata, what we will see is the next table.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
D_T_NAME	VARCHAR2(10 BYTE)	Yes	(null)	1	(null)
DOC_BOSS	DOCTOR_TYPE	Yes	(null)	2	(null)

DOC_BOSS is made from DOCTOR_TYPE.

Another example for REF, first we are going to create a type: EMPLOYEE. Secondly we create a table EMPLOYEES of EMPLOYEE. Thirdly we will insert some data to the row type table.

After doing this steps we have made along the practical work, is time to create a new object (E_REF) which will contain an ID and an attribute called E_EMP which is type will make a reference to EMPLOYEE. Then create the type as a table of E_REF. Finally, a table with a nested table, so our final table will be in fact an object collection.

```
CREATE TYPE EMPLOYEE AS OBJECT(
```

```
    EMP_ID NUMBER,
```

```
    EMP_NAME VARCHAR2(20),
```

```
    EMP_SURNAME VARCHAR2(20),
```

```
    EMP_AREA VARCHAR2(20),
```

```
    YEARS_WORKING NUMBER
```

```
);
```

```
CREATE TABLE EMPLOYEES OF EMPLOYEE;
```

```
INSERT INTO EMPLOYEES VALUES(1, 'Steven', 'Wolkiwa', 'Office', 5);
```

```
INSERT INTO EMPLOYEES VALUES(2, 'Viktorija', 'Pasova', 'Engineer', 1);
```

```
INSERT INTO EMPLOYEES VALUES(3, 'Anna', 'Poklonskaya', 'Engineer', 1);
```

```
INSERT INTO EMPLOYEES VALUES(4, 'Elsa', 'Torres', 'Office', 7);
```

```
INSERT INTO EMPLOYEES VALUES(5, 'Hans', 'Spiet', 'Ambulance', 3);
```

```

INSERT INTO EMPLOYEES VALUES(6, 'Krsitoff', 'Spiet', 'Library', 2);
INSERT INTO EMPLOYEES VALUES(7, 'Peter', 'Seles', 'Ambulance', 9);

CREATE TYPE E_REF AS OBJECT(
    E_ID NUMBER,
    E_EMP REF EMPLOYEE
);

CREATE TYPE TYPE_REF AS TABLE OF E_REF;

CREATE TABLE EMP_KR(
    EMP_ID2 NUMBER PRIMARY KEY,
    KR_REF TYPE_REF)
    NESTED TABLE KR_REF STORE AS XSTORE;

```

As far so good, but then we want to use the references to edit, add or insert new data, however an error occurred and I do not know how to solve it as I have revised plenty of times.

```

DECLARE
    REFERENCE_A REF EMPLOYEE;
    REFERENCE_B REF EMPLOYEE;
    REFERENCE_C REF EMPLOYEE;

BEGIN
    SELECT REF(K) INTO REFERENCE_A
    FROM EMPLOYEES K
    WHERE K.EMP_ID = 1;

    SELECT REF(K) INTO REFERENCE_B
    FROM EMPLOYEES K
    WHERE K.EMP_ID = 2;

    SELECT REF(K) INTO REFERENCE_C
    FROM EMPLOYEES K
    WHERE K.EMP_ID = 3;

```

```
INSERT INTO EMP_KR VALUES(1, TYPE_REF(100, REFERENCE_A));  
END;
```

Informe de error:

ORA-06550: línea 19, columna 43:

PL/SQL: ORA-00932: tipos de dato inconsistentes: se esperaba UDT se ha obtenido NUMBER

ORA-06550: línea 19, columna 5:

PL/SQL: SQL Statement ignored

ORA-06550. 00000 - "line %s, column %s:\n%s"

*Cause: Usually a PL/SQL compilation error.

*Action:

Use of special object functions in SQL

For the following examples, I will create a new table (Object collection).

```
CREATE TYPE STUDENT_INFO AS OBJECT(  
    STUDENT_ID NUMBER,  
    GRADE NUMBER,  
    ST_GROUP VARCHAR2(5),  
    ST_FACULTY VARCHAR(20)  
);  
  
CREATE TYPE STUDENTS_INFO AS TABLE OF STUDENT_INFO;  
  
CREATE TABLE STUDENT(  
    S_ID NUMBER PRIMARY KEY,  
    STUDENT STUDENT_INFO  
);  
  
INSERT INTO STUDENT VALUES(1, STUDENT_INFO(1, 10, '8', 'Computer Science'));  
INSERT INTO STUDENT VALUES(2, STUDENT_INFO(2, 9, '4', 'Computer Science'));  
INSERT INTO STUDENT VALUES(3, STUDENT_INFO(3, 10, '8', 'Computer Science'));  
INSERT INTO STUDENT VALUES(4, STUDENT_INFO(4, 9, '4', 'Computer Science'));  
INSERT INTO STUDENT VALUES(5, STUDENT_INFO(5, 7, '1', 'Bussines Admin'));  
INSERT INTO STUDENT VALUES(6, STUDENT_INFO(6, 9, '4', 'Computer Science'));  
INSERT INTO STUDENT VALUES(7, STUDENT_INFO(7, 8, '1', 'Bussines Admin'));  
INSERT INTO STUDENT VALUES(8, STUDENT_INFO(8, 10, '4', 'Computer Science'));  
  
CREATE TABLE LECTURE(  
    LEC_ID NUMBER PRIMARY KEY,  
    TEACHER_NAME VARCHAR2(20),  
    STUDENT STUDENTS_INFO  
) NESTED TABLE STUDENT STORE AS STUDENT_NESTED_TABLE;  
  
INSERT INTO LECTURE VALUES(1, 'MR ONE', STUDENTS_INFO(
```



```

(SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 1),
(SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 3 )
)
);

INSERT INTO LECTURE VALUES(2, 'MR DIVI', STUDENTS_INFO(
  SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 3
)
);

INSERT INTO LECTURE VALUES(3, 'MR TRE', STUDENTS_INFO(
  (SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 1),
  (SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 2),
  (SELECT Z.STUDENT FROM STUDENT Z WHERE S_ID = 5)
)
);

```

1 Comparing collections

You can check whether a collection is null, and whether two collections are the same. Comparisons such as greater than, less than, and so on are not allowed.

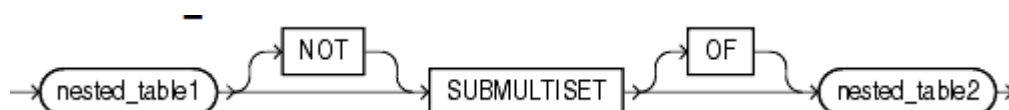
This restriction also applies to implicit comparisons. For example, collections cannot appear in a DISTINCT, GROUP BY, or ORDER BY list.

If you want to do such comparison operations, you must define your own notion of what it means for collections to be greater than, less than, and so on, and write one or more functions to examine the collections and their elements and return a true or false value.

You can apply set operators (CARDINALITY, MEMBER OF, IS A SET, IS EMPTY) to check certain conditions within a nested table or between two nested tables.

1.1 Operator SUBMULTISET OF

The SUBMULTISET condition tests whether a specified nested table is a submultiset of another specified nested table. The operator returns a boolean value: TRUE if nested_table1 is a submultiset of nested_table2.



- The OF keyword is optional and does not change the behavior of the operator.
- The NOT keyword reverses the boolean output: Oracle returns FALSE if nested_table1 is a subset of nested_table2.
- The element types of the nested table must be comparable. Please refer to "Comparison Conditions " for information on the comparability of nonscalar types.

```

SELECT
  I2.ST_GROUP, T2_Z.ST_GROUP,
  I1.ST_GROUP, T1_Z.ST_GROUP
FROM
  LECTURE I1, LECTURE I2,
  TABLE(I1.STUDENT) T1_Z,
  TABLE(I2.STUDENT) T2_Z
WHERE
  I1.STUDENT SUBMULTISET OF I2.STUDENT AND NOT(I1.LEC_ID = I2.LEC_ID)
GROUP BY
  I2.ST_GROUP, T2_Z.ST_GROUP,
  I1.ST_GROUP, T1_Z.ST_GROUP;

```

12 Operator *MEMBER OF*

```

SELECT DISTINCT N.TEACHER_NAME, Z_T.ST_GROUP
FROM LECTURE N,(
  SELECT Z.STUDENT
  FROM STUDENT Z
  WHERE Z.STUDENT.ST_GROUP= '4'
) Z,
TABLE(N.STUDENT) Z_T
WHERE Z.STUDENT MEMBER OF N.STUDENT
GROUP BY N.TEACHER_NAME, Z_T.ST_GROUP;

```

TEACHER_NAME	ST_GROUP
MR TRE	1
MR TRE	8
MR TRE	4

13 Operator IS A SET

You can test certain properties of a nested table, or compare two nested tables, using ANSI-standard set operations:

```

DECLARE
    TYPE nested_typ IS TABLE OF NUMBER;
    nt1 nested_typ := nested_typ(1,2,3);
    nt2 nested_typ := nested_typ(3,2,1);
    nt3 nested_typ := nested_typ(2,3,1,3);
    nt4 nested_typ := nested_typ(1,2,4);
    answer BOOLEAN;
    howmany NUMBER;

    PROCEDURE testify(truth BOOLEAN DEFAULT NULL, quantity NUMBER DEFAULT
NULL) IS
    BEGIN
        IF truth IS NOT NULL THEN
            dbms_output.put_line(CASE truth WHEN TRUE THEN 'True' WHEN FALSE THEN 'False'
END);
        END IF;
        IF quantity IS NOT NULL THEN
            dbms_output.put_line(quantity);
        END IF;
    END;
BEGIN
    answer := nt1 IN (nt2,nt3,nt4); -- true, nt1 matches nt2
    testify(truth => answer);
    answer := nt1 SUBMULTISET OF nt3; -- true, all elements match
    testify(truth => answer);
    answer := nt1 NOT SUBMULTISET OF nt4; -- also true
    testify(truth => answer);

```

```

howmany := CARDINALITY(nt3); -- number of elements in nt3
testify(quantity =>howmany);
howmany := CARDINALITY(SET(nt3)); -- number of distinct elements
testify(quantity =>howmany);

answer := 4 MEMBER OF nt1; -- false, no element matches
testify(truth => answer);
answer := nt3 IS A SET; -- false, nt3 has duplicates
testify(truth => answer);
answer := nt3 IS NOT A SET; -- true, nt3 has duplicates
testify(truth => answer);
answer := nt1 IS EMPTY; -- false, nt1 has some members
testify(truth => answer);
END;

```

14 CARDINALITY

CARDINALITY returns the number of elements in a nested table. The return type is NUMBER. If the nested table is empty, or is a null collection, then CARDINALITY returns NULL.

Conclusion

In this first practical work of the course, was viewed object-relational database principle to form a library database. The paper was made both object tables and relational tables with object columns and tables with a collection of objects.

Although I have worked previously with Microsoft SQL Server, working with Oracle have represented a very interesting challenge. SQL language could be the same in many ways, but the Graphic User Interface is different and it took me a bit long to get used to it.

The very problem at the beginning was the impossibility to connect with the RTU Server, but it was not a limitation to achieve this practical work, as I could make up my own examples to understand the different topics. During the programming occurred some problems with the coding that were, after a serious of reading, achieved.

In the end it must be said that this work was sufficiently difficult, because it required a lot of new concepts to use, which before they were not covered by the concept of databases.

References

- 1 https://docs.oracle.com/cd/A58617_01/server.804/a58227/ch_objvw.htm
- 2 http://download.oracle.com/otndocs/tech/semantic_web/pdf/oow2007_semantics_techtalk_k.pdf
- 3 https://docs.oracle.com/cd/B12037_01/server.101/b10759/conditions019.htm
- 4 https://docs.oracle.com/cd/B10500_01/appdev.920/a96624/10_objs.htm
- 5 https://docs.oracle.com/cd/B13789_01/appdev.101/b10807/05_colls.htm