

**RĪGAS TEHNISKĀ UNIVERSITĀTE**

**Datorzinātnes un informācijas tehnoloģijas fakultāte**

**Informācijas tehnoloģijas institūts**

Technology of Large Data Bases



Castillo Torres Miguel Eduardo

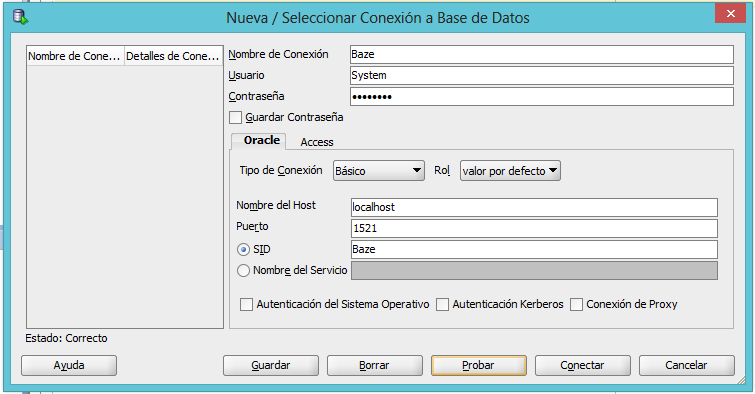
First practical work:

Relational-Object DB Structures

Semester fall 2015

# INTRODUCTION

Just a remainder: 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

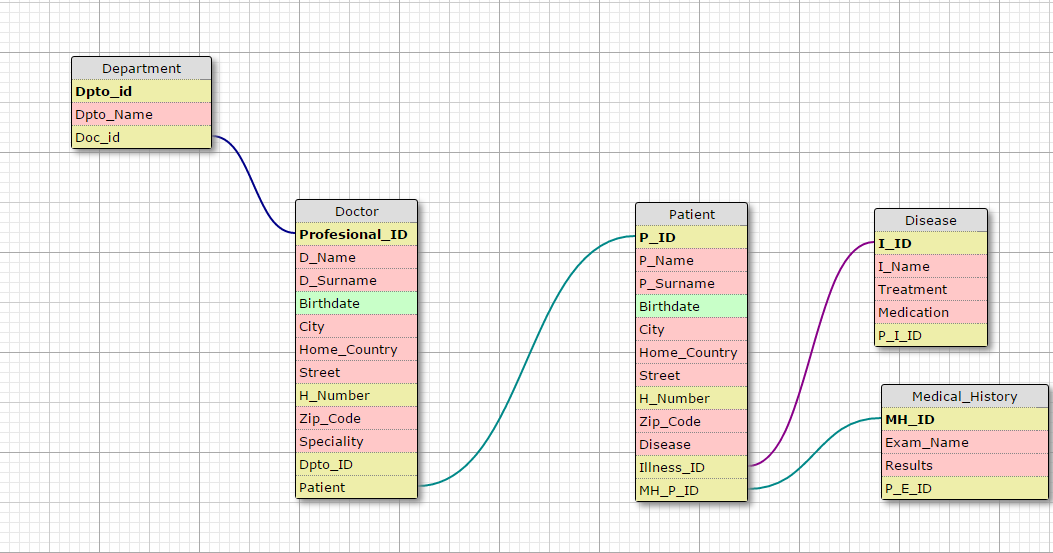
• Allows you to integrate data from different sources without custom programming

• 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 3](#_Toc433230841)

[Table with row type objects 6](#_Toc433230842)

[1 Creation 6](#_Toc433230843)

[2 Data Input 6](#_Toc433230844)

[3 Select 7](#_Toc433230845)

[4 Metadata 7](#_Toc433230846)

[Table with object column 7](#_Toc433230847)

[1 Creation 7](#_Toc433230848)

[2 Data input 7](#_Toc433230849)

[3 Select 7](#_Toc433230850)

[4 Metadata 7](#_Toc433230851)

[Table with object collection 7](#_Toc433230852)

[1 Creation 7](#_Toc433230853)

[2 Data input 7](#_Toc433230854)

[3 Select 7](#_Toc433230855)

[4 Metadata 7](#_Toc433230856)

[Object view 7](#_Toc433230857)

[Object references 7](#_Toc433230858)

[Use of special object functions in SQL 7](#_Toc433230859)

[1 Comparing collections 7](#_Toc433230860)

[2 Operator SUBMULTISET OF 7](#_Toc433230861)

[3 Operator MEMBER OF 7](#_Toc433230862)

[4 Operator IS A SET 7](#_Toc433230863)

[5 CARDINALITY 7](#_Toc433230864)

[Conclusion 7](#_Toc433230865)

[References 7](#_Toc433230866)

# 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.

## 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”.



## 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'));

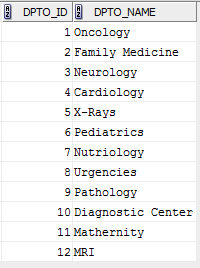
## Select

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.

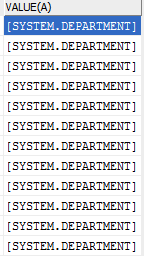


But, with a slight modification in the previous code, we will get a different result.

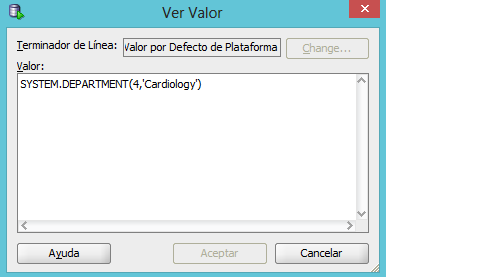
SELECT VALUE(A)

FROM Departments A;

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



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.



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';



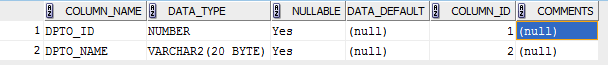
## Metadata

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.



# Table with object column

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

## 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.

## 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('Lativa', '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('Kazahstan', '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('Kazahstan', 'Pavldar', 'Udens', 15, '7689'), 12, 11);

INSERT INTO DOCTOR VALUES(12, PERSONAL\_DATA('Dina', 'Lol', (TO\_DATE('1995/11/08','yyyy/mm/dd'))), ADDRESS('Kazahstan', 'Pavldar', 'SnegBouleverd', 31, '7690'), 10, 12);

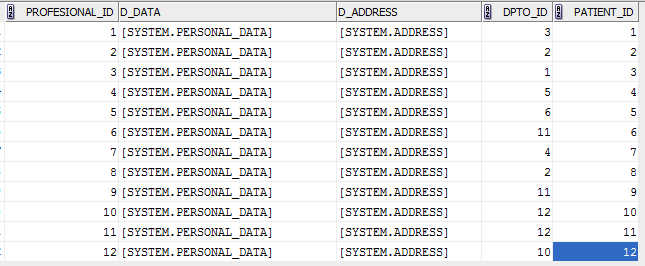
## Select

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

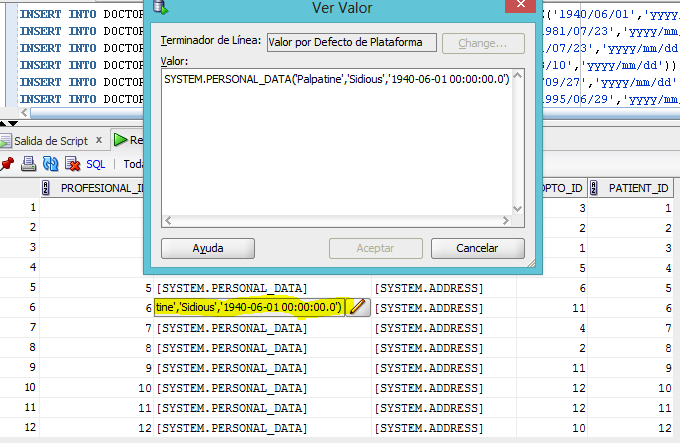
With the simple “SELECT ALL” we will get the complete table with the objects.

SELECT \*

FROM DOCTOR;



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



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;

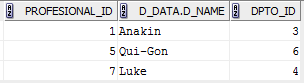


We get the names of the “Doctors” working in DPTO\_ID 12.

SELECT b.PROFESIONAL\_ID, B.D\_DATA.D\_NAME, B.DPTO\_ID

FROM DOCTOR B

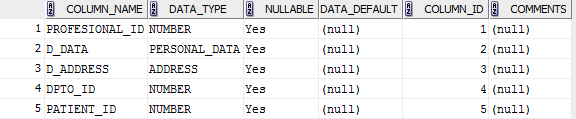
WHERE B.D\_ADDRESS.HOME\_COUNTRY = 'Mexico';



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

## Metadata

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



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.

## 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 codding 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.

## 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+')));

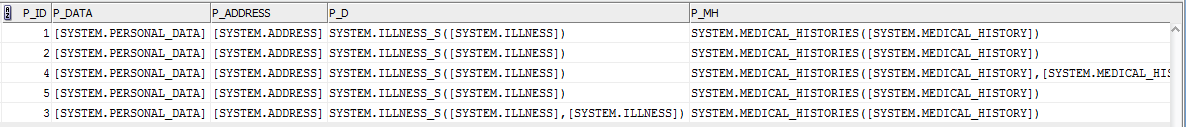
INSERT INTO PATIENT VALUES(4, PERSONAL\_DATA('Ando', 'Nagasaki', (TO\_DATE('1990/01/01','yyyy/mm/dd'))), ADDRESS('Japan','Osaka','Haghen', 4,'4532'), ILLNESS\_S(ILLNESS(5, 'Cancer','NON','NON')), MEDICAL\_HISTORIES(MEDICAL\_HISTORY(4, 'Blood', 'O+'), MEDICAL\_HISTORY(5, 'Cholesterol', '5')));

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')));

## Select

SELECT \*

FROM PATIENT;

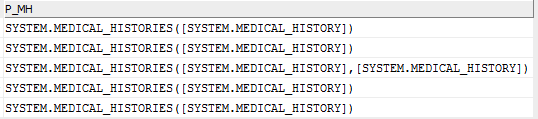


Our simply query show us the complete table collection, as we see the screen is not big enough to show the complete data.

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

SELECT C.P\_MH

FROM PATIENT C;



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

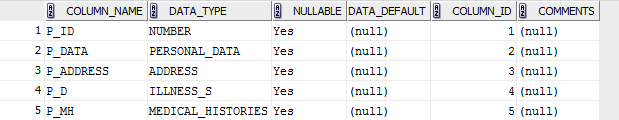
FROM PATIENT D, TABLE(D.P\_MH) E

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.

.

## Metadata



# 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 \*

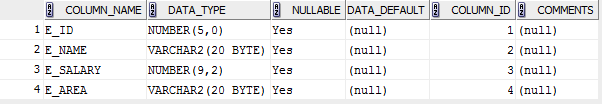
FROM EMPLOYEE\_VIEW;



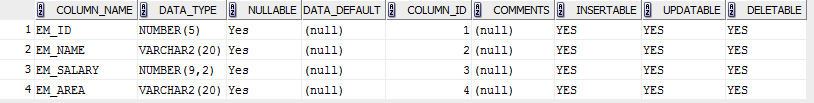
What I did was first create a table with some data of the employee, then create an object with similar attributes of the table. After that, the view is created and we can insert our data.

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.



And the metadata of the object view is.



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

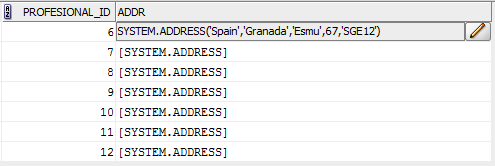
SELECT H.PROFESIONAL\_ID, ADDRESS(H.D\_ADDRESS.HOME\_COUNTRY, H.D\_ADDRESS.CITY, H.D\_ADDRESS.STREET, H.D\_ADDRESS.H\_NUMBER, H.D\_ADDRESS.ZIP\_CODE) AS DOC\_DPT\_ADDR

FROM DOCTOR H

WHERE H.PROFESIONAL\_ID > 5;

SELECT \*

FROM DEPT\_VIEW;



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:



# 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))

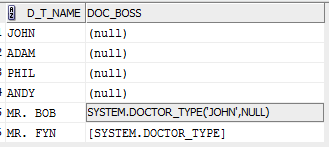
FROM DOCTOR\_OBJECT\_TABLE I

WHERE I.D\_T\_NAME = 'ANDY';

COLUMN D\_T\_NAME FORMAT A10

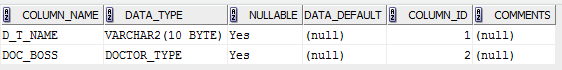
COLUMN DOC\_BOSS FORMAT A50

select \* from DOCTOR\_OBJECT\_TABLE J;



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

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



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 and ID and an attribute called E\_EMP which is type will make a reference to EMPLOYEE. Then create de 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;

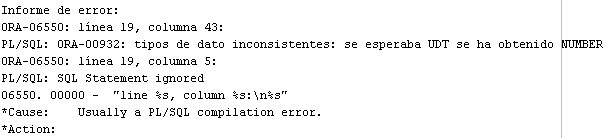
SELECT REF(K) INTO REFERENCE\_C

FROM EMPLOYEES K

WHERE K.EMP\_ID = 3;

INSERT INTO EMP\_KR VALUES(1, TYPE\_REF(100, REFERENCE\_A));

END;



# 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)

)

);

## 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.

## 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;

## Operator MEMBER OF

SELECT DISTINCT N.TEACHER\_NAME, Z\_T.ST\_GROUP

FROM LECTURE N,(

SELECT Z.STUDENT

FROM STUDENT Z

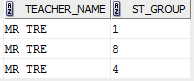
WHERE Z.STUDENT.ST\_GROUP= '4'

) Z,

TABLE(N.STUDENT) Z\_T

WHERE Z.STUDENT MEMBER OF N.STUDENT

GROUP BY N.TEACHER\_NAME, Z\_T.ST\_GROUP;



## 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;

## 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>