

# Exercise 2:

# Object-relational data model in Oracle

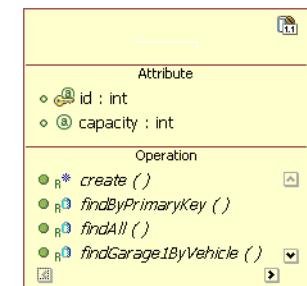
# Why object-relational model?

There are contexts (maps, images) where the use of relational model seems too restrictive due to the necessary complex representation.

With the object-relational approach, objects can be represented easily with structured data types, also associating methods to them.

AIR_HEART_RATE	STOKE_HEART_DATE	AIR_CHEST_VOLUME	STOKE_CHEST_VOLUME	AIR_BLOOD_OXYGEN	STOKE_BLOOD_OXYGEN
74.940	7/23/06	5419.000	3275.672	7771.705	11.11204
78.946	3/7/07	5779.150	4993.234	7771.705	18.18452
78.947	2/1/07	5500.395	3424.545	7771.705	15.25749
77.948	1/1/07	5500.395	3424.545	7771.705	15.25749
77.949	4/1/04/05	5760.239	3640.703	7703.215	13.24317
78.947	8/3/06/08	5779.150	3473.838	5953.1495	1844.22208
78.949	3/24/07	5500.395	3424.545	5412.115	269.8302
78.950	4/20/03	5770.476	4320.440	5009.84	402.87788
74.947	7/23/04	5779.150	3471.978	5248.292	779.3512
78.948	5/14/05	5300.985	3600.904	5198.439	713.3954
74.948	5/3/03/05	5238.972	3240.693	5603.800	325.84018
77.949	7/04/04	5250.430	4590.967	4010.919	1420.0084
78.949	1/1/06	5307.487	3472.530	6112.767	1664.1457
77.950	6/28/01	5203.254	1762.430	5008.364	224.93472
77.951	6/04/04	5305.3	4141.072	4400.817	1584.1202
72.940	5/04/02	5300.540	1931.212	5008.296	377.4432

AIR_HEART_RATE	STOKE_HEART_DATE	AIR_CHEST_VOLUME	STOKE_CHEST_VOLUME
75.948	2/1/07	5270.940	
75.949	2/1/07	5200.288	
75.950	4/20/04/05	5240.238	
75.951	3/24/07	5200.288	
75.952	3/24/07	5000.071	
76.948	4/3/03/05	5271.975	
74.947	7/23/04	5270.940	
74.948	5/3/03/05	5238.972	
74.949	5/03/05	5230.072	
75.949	5/03/05	5230.072	
77.949	7/04/04	5250.430	
78.949	1/1/06	5307.487	
78.950	6/28/01	5203.254	
77.951	6/04/04	5305.3	
77.952	6/04/04	5305.3	
72.940	5/04/02	5300.540	
75.948	2/1/05	5271.944	



# The abstract data types in Oracle ...

- user-defined data types;
- usually allow grouping columns or related fields as a **object**;
- the syntax is:

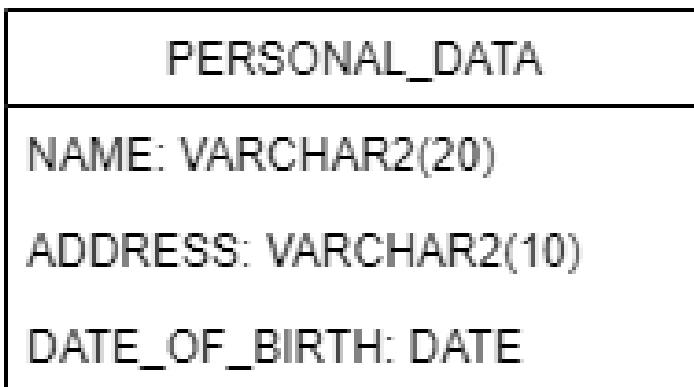
```
CREATE TYPE typeName AS OBJECT
(
column1 TypeCol1,
column2 TypeCol2,
.
.
.
columnK TypeColK
);
```

# ...The abstract data types in Oracle ...

For example, the columns NAME, ADDRESS, DATE\_OF\_BIRTH of the table CUSTOMER

NAME	ADDRESS	DATE_OF_BIRTH

can be grouped together to define the abstract data type PERSONAL\_DATA with the specification of its attributes



# ...The abstract data types in Oracle ...

```
CREATE TYPE typePersonalData AS OBJECT  
(NAME VARCHAR2(20),  
ADDRESS VARCHAR2(10),  
DATE_OF_BIRTH DATE);
```

and the CUSTOMER table may be amended as follows:

```
CREATE TABLE CUSTOMER  
(CF VARCHAR2 (20),  
VAT_NUMBER VARCHAR2 (20),  
CUSTOMERTYPE VARCHAR2 (20),  
PERSONAL_DATA typePersonalData);
```

# ...The abstract data types in Oracle ...

The abstract data types can be created or modified via Oracle graphics tool ([SQL Developer](#))

The screenshot shows the Oracle SQL Developer interface with several windows open:

- Connections** window: Shows various database objects like Functions, Operators, Queues, Triggers, and Types. The "Types" node is selected and highlighted with a red box.
- Code Editor** window: Displays the PL/SQL code for creating an abstract data type:
 

```
CREATE OR REPLACE TYPE TYPEPERSONALDATA AS OBJECT
(
  NAME VARCHAR2(20),
  ADDRESS VARCHAR2(10),
  DATE_OF_BIRTH DATE
);
```
- Connections** window: Shows the connection to the "YELLOWCOM" database, specifically the "CUSTOMER" table. The "PERSONAL\_DATA" column is selected and highlighted with a red box.
- Customer** window: Shows the table structure for "CUSTOMER". The "PERSONAL\_DATA" column is of type "TYPEPERSONALDATA".
- Script Output** window: Shows the successful execution of the create table command:
 

```
Table CUSTOMER created.
```

# ...The abstract data types in Oracle ...

Once you have created an abstract data type and a table defined also on the type created, any operation to be performed on that table will consider the new structuring of the table

`INSERT INTO CUSTOMER values ('SSN', 'VAT identification number',  
'Typecustomer', typePersonalData ('Name', 'Address', '10-DEC-95'))`

type constructor

table alias

dot notation to refer to the  
column in question

name of the abstract  
type column

`UPDATE CUSTOMER C SET C.PERSONAL_DATA.ADDRESS= 'address'  
where C.PERSONAL_DATA.ADDRESS= 'Address'`

The screenshot shows the Oracle SQL Developer interface. At the top, there are two tabs: 'TYPEPERSONALDATA' and 'script.sql'. The 'script.sql' tab is active, showing the following SQL code:

```
insert into CUSTOMER values('SSN', 'VAT_id_no', 'Typecustomer', typePersonalData ('Name', 'Address', '10-DEC-95'))
```

Below the code editor is a 'Script Output' window with the message: 'Task completed in 0.279 seconds'. Underneath, the output shows: '1 row inserted.'

The screenshot shows the Oracle SQL Developer interface. On the left, there is a tree view of database objects under 'Tables (Filtered)'. The 'CUSTOMER' table is selected. The main workspace contains the following SQL code:

```
UPDATE CUSTOMER C SET C.PERSONAL_DATA.ADDRESS= 'address'  
where C.PERSONAL_DATA.ADDRESS= 'address'
```

Below the code editor is a 'Script Output' window with the message: 'Task completed in 0.384 seconds'. The output shows: '0 rows updated.' followed by the note: 'because the where condition is not true'.

# ...The abstract data types in Oracle ...

In the statement of an abstract data type, we can also declare methods (only their signature) operating on the type attributes

PERSONAL_DATA
NAME: VARCHAR2(20)
ADDRESS: VARCHAR2(10)
DATE_OF_BIRTH: DATE
METHOD

The syntax is:

```
CREATE TYPE typeName AS OBJECT
(...  

columnK typeColumnK,  

...)
```

```
MEMBER FUNCTION  

methodName  

(typeParameter IN parameterName)  

RETURN returnType  

);
```

```
CREATE TYPE typeName AS OBJECT
(...  

columnK typeColumnK,  

...)
```

```
MEMBER PROCEDURES  

methodName  

(typeParameter IN / OUT parameterName)  

);
```

used parameter (IN) returned  
(OUT) by the method

# ...The abstract data types in Oracle ...

To declare the method implementation

```
CREATE TYPE BODY typeName IS
  MEMBER FUNCTION methodName (ParameterType parameterName)
  RETURN returnType IS
    BEGIN
      ....
    END;
END;
```

# ...The abstract data types in Oracle ...

Continuing the above example

*CREATE TYPE typePersonalData AS OBJECT*

*( NAME VARCHAR2(20),  
ADDRESS VARCHAR2(10),  
DATE\_OF\_BIRTH DATE, MEMBER FUNCTION ComputeAGE  
(DATE\_OF\_BIRTH IN DATE) RETURN NUMBER);*

*CREATE TYPE BODY typePersonalData IS  
MEMBER FUNCTION ComputeAGE (DATE\_OF\_BIRTH IN  
DATE) RETURN NUMBER IS*

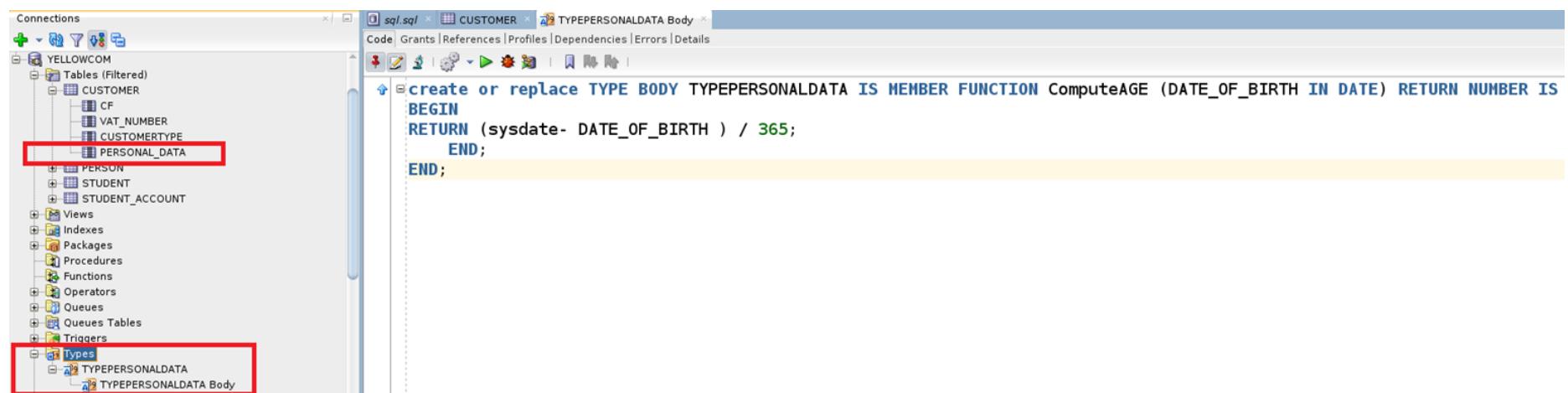
*BEGIN*

*RETURN (sysdate - DATE\_OF\_BIRTH) / 365;*

*END;*

*END;*

# ...The abstract data types in Oracle ...



# ...The abstract data types in Oracle

We can call the declared method:

```
SELECT C.personalData.ComputeAGE  
(C.personalData.DATE_OF_BIRTH) FROM CUSTOMER C
```

The screenshot shows the Oracle SQL Developer interface. On the left is the Object Navigator pane, which lists various database objects like CUSTOMER, PERSON, and Procedures. In the center is the SQL Worksheet pane where the following PL/SQL block is executed:

```
SELECT C.PERSONAL_DATA.ComputeAGE (C.PERSONAL_DATA.DATE_OF_BIRTH) FROM CUSTOMER C;
```

Below the worksheet is the Query Result pane, which displays the output of the query. The results are:

C.PERSONAL_DATA.COMPUTEAGE(C.PERSONAL_DATA.DATE_OF_BIRTH)
1 25.91373769660071029934043632673769660071
2 25.91099797057331303906646372399797057331

```
SELECT ROUND(C.personalData.ComputeAGE  
(C.personalData.DATE_OF_BIRTH), 2) FROM CUSTOMER C
```

→25.91  
→25.91

# VARRAY...

- Allow us to create aggregations (multivalued) of the same type in the same column objects;
- The syntax that lets us create the aggregation is:

```
CREATE TYPE nameVARRAY
AS VARRAY (size max) OF ObjectType
```

- The syntax for defining a column of aggregation type of objects is:

```
CREATE TABLE tableName
(...)
columnname nameVARRAY,
....)
```

# ... VARRAY...

For example, given the state of CUSTOMER table

Code	DoB	Address	Name	Tel. No	VAT No	Type

it is possible to add a VARRAY *arrayPHONE*

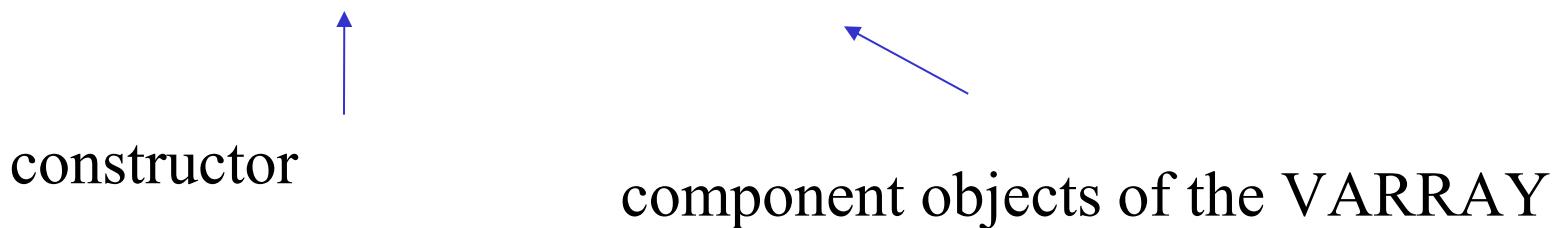
CREATE TYPE *arrayPHONE* AS VARRAY(10) OF NUMBER  
and recreate the table

```
CREATE TABLE CUSTOMER
(
...
telephoneNumber arrayPHONE;
...)
```

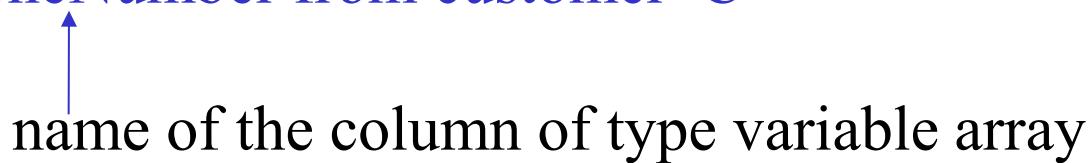
# ... VARRAY

Thus, any operation can take advantage of the new structure

```
insert into Customer ( 'StringFiscale', '31-Dec-05', 'stringInd',
'stringNome', arrayPHONE(080544,3280000), 1234, 'stringTipo')
```



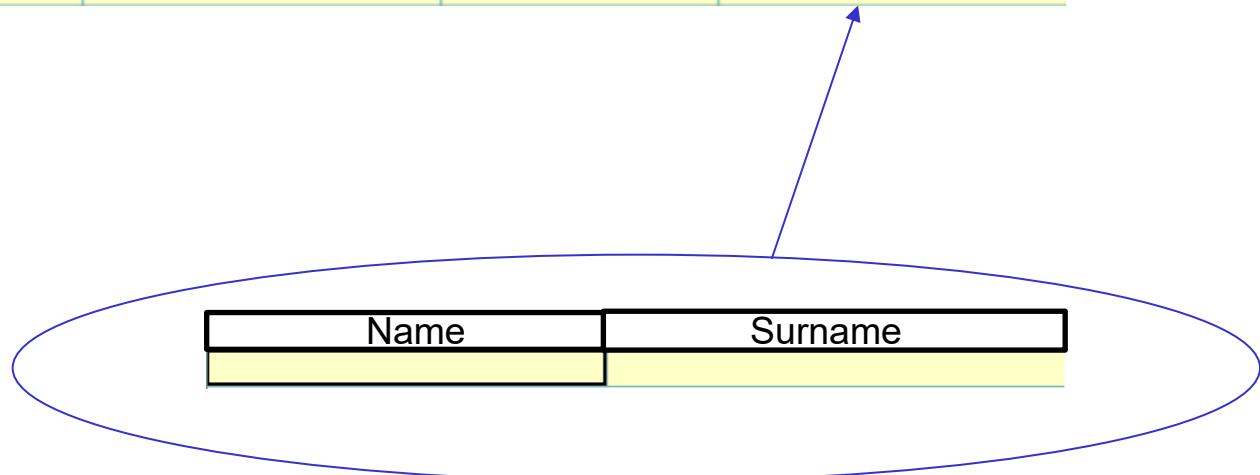
```
Select C.telephoneNumber from customer C
```



# Nested Tables...

- They make possible multiple relationships (1-N) within a table;
- A nested table is represented by a column in another table

Code	Total Cost	Date	No Participants	Event Name	Guest List



# ...Nested Tables...

- The syntax for creating a data type on which a nested table can be based is:

```
CREATE TYPE NESTED_TABLE_BASE  
AS TABLE OF type_nested_table_objects
```

- The syntax for creating a table that contains a column that is a nested table is

```
CREATE TABLE table  
(...  
 nomeNestedColumn NESTED_TABLE_BASE,  
 ...)  
NESTED TABLE nomeNestedColumn STORE AS nestedTableNT
```

- The type *type\_nested\_table\_objects* can also be an abstract type

# ...Nested Tables...

For example, given the type abstract *typeSpouse*

```
CREATE TYPE typeSpouse AS OBJECT  
(Name VARCHAR (50), weddingDate DATE)
```

We create the type of data on which to base the nested table

```
CREATE TYPE baseSpouse AS OF TABLE typeSpouse
```

below we create the table that will contain a column of type nested table

```
CREATE TABLE Customers1
```

```
( Name VARCHAR (50), date_of_birth DATE, place_of_birth  
VARCHAR (50), date_of_death DATE, place_of_death VARCHAR  
(50), listSpouses baseSpouse)
```

```
NESTED TABLE listSpouses STORE AS baseSpouse_TAB
```

# ...Nested Tables...

To insert customers, we will use the constructors of the new data types, in our example we have:

```
INSERT INTO VALUES Customers1  
( 'Paolo Rossi', '1-Jan-1962', 'Milano', null, null,  
baseSpouse ( ←  
typeSpouse ( 'Paola Bianchi', '12-May-1980'),  
typeSpouse ( 'Franca Verdi', '27-Jun-1997')  
)  
)
```

constructor of the  
abstract data type used  
by the base type

# ...Nested Tables.

To refer to the columns contained in the nested table, we must use the flattening operator (**TABLE**)

```
SELECT ES.Name FROM TABLE (SELECT listSpouses FROM Customers1
                           WHERE Birthdate < DATE '2006-06-01') ES;
```

provided that identifies which of nested table consider from the main table

```
INSERT INTO TABLE (select listSpouses from Customers1 where
                   name = 'Paolo Rossi') VALUES (typeSpouse ( 'Liliana Blue', '12 Oct 2002'))
```

If the query retrieves more than one instance, the DBMS raises an exception

*ORA-01427: single-row subquery returns more than one row*

```
SELECT ES.Name FROM Customers1 C,
       TABLE(C.listSpouses) ES
      WHERE C.Birthdate < DATE '2006-06-01';
```

In this case, the **TABLE(C.listSpouses)** takes the *listSpouses* collection of each customer C and expands it into rows.

This query performs a classical implicit join between the upper table and the nested one

# Using CAST and MULTISET

Remember that: if you try to directly enter the values of the columns of the nested table with an INSERT command in the main table, you get an error because the main table contains one column for each nested table.

To get the right result we have to use the operators **CAST** **MULTISET** in the form:

**CAST (MULTISET (<subquery>) AS <TipoNT>)**

# Exercises

Let us consider the case study information system of a mobile telephone operator.

- Create Abstract Types
- Create the corresponding tables
- Use nested tables.