

# 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.

VEHICLE_ID	VEHICLE_TYPE	VEHICLE_YEAR	VEHICLE_PRICE	VEHICLE_CAPACITY	VEHICLE_COLOR	VEHICLE_STATUS
74.1230	1	2008	15000	5	Red	Active
74.1231	1	2009	16000	5	Blue	Active
74.1232	1	2010	17000	5	Green	Active
74.1233	1	2011	18000	5	Yellow	Active
74.1234	1	2012	19000	5	Black	Active
74.1235	1	2013	20000	5	White	Active
74.1236	1	2014	21000	5	Grey	Active
74.1237	1	2015	22000	5	Brown	Active
74.1238	1	2016	23000	5	Pink	Active
74.1239	1	2017	24000	5	Orange	Active
74.1240	1	2018	25000	5	Purple	Active
74.1241	1	2019	26000	5	Gold	Active
74.1242	1	2020	27000	5	Silver	Active
74.1243	1	2021	28000	5	Platinum	Active
74.1244	1	2022	29000	5	Diamond	Active
74.1245	1	2023	30000	5	Emerald	Active
74.1246	1	2024	31000	5	Sapphire	Active
74.1247	1	2025	32000	5	Amethyst	Active
74.1248	1	2026	33000	5	Garnet	Active
74.1249	1	2027	34000	5	Peridot	Active
74.1250	1	2028	35000	5	Spinel	Active
74.1251	1	2029	36000	5	Topaz	Active
74.1252	1	2030	37000	5	Quartz	Active
74.1253	1	2031	38000	5	Jade	Active
74.1254	1	2032	39000	5	Obsidian	Active
74.1255	1	2033	40000	5	Malachite	Active
74.1256	1	2034	41000	5	Onyx	Active
74.1257	1	2035	42000	5	Opal	Active
74.1258	1	2036	43000	5	Turquoise	Active
74.1259	1	2037	44000	5	Flint	Active
74.1260	1	2038	45000	5	Amber	Active
74.1261	1	2039	46000	5	Jet	Active
74.1262	1	2040	47000	5	Shale	Active
74.1263	1	2041	48000	5	Slate	Active
74.1264	1	2042	49000	5	Schist	Active
74.1265	1	2043	50000	5	Gneiss	Active
74.1266	1	2044	51000	5	Granite	Active
74.1267	1	2045	52000	5	Basalt	Active
74.1268	1	2046	53000	5	Andesite	Active
74.1269	1	2047	54000	5	Diorite	Active
74.1270	1	2048	55000	5	Gabbro	Active
74.1271	1	2049	56000	5	Peridotite	Active
74.1272	1	2050	57000	5	Eclogite	Active
74.1273	1	2051	58000	5	Marble	Active
74.1274	1	2052	59000	5	Serpentine	Active
74.1275	1	2053	60000	5	Talc	Active
74.1276	1	2054	61000	5	Pyroxene	Active
74.1277	1	2055	62000	5	Quartzite	Active
74.1278	1	2056	63000	5	Schist	Active
74.1279	1	2057	64000	5	Gneiss	Active
74.1280	1	2058	65000	5	Granite	Active
74.1281	1	2059	66000	5	Basalt	Active
74.1282	1	2060	67000	5	Andesite	Active
74.1283	1	2061	68000	5	Diorite	Active
74.1284	1	2062	69000	5	Gabbro	Active
74.1285	1	2063	70000	5	Peridotite	Active
74.1286	1	2064	71000	5	Eclogite	Active
74.1287	1	2065	72000	5	Marble	Active
74.1288	1	2066	73000	5	Serpentine	Active
74.1289	1	2067	74000	5	Talc	Active
74.1290	1	2068	75000	5	Pyroxene	Active
74.1291	1	2069	76000	5	Quartzite	Active
74.1292	1	2070	77000	5	Schist	Active
74.1293	1	2071	78000	5	Gneiss	Active
74.1294	1	2072	79000	5	Granite	Active
74.1295	1	2073	80000	5	Basalt	Active
74.1296	1	2074	81000	5	Andesite	Active
74.1297	1	2075	82000	5	Diorite	Active
74.1298	1	2076	83000	5	Gabbro	Active
74.1299	1	2077	84000	5	Peridotite	Active
74.1300	1	2078	85000	5	Eclogite	Active
74.1301	1	2079	86000	5	Marble	Active
74.1302	1	2080	87000	5	Serpentine	Active
74.1303	1	2081	88000	5	Talc	Active
74.1304	1	2082	89000	5	Pyroxene	Active
74.1305	1	2083	90000	5	Quartzite	Active
74.1306	1	2084	91000	5	Schist	Active
74.1307	1	2085	92000	5	Gneiss	Active
74.1308	1	2086	93000	5	Granite	Active
74.1309	1	2087	94000	5	Basalt	Active
74.1310	1	2088	95000	5	Andesite	Active
74.1311	1	2089	96000	5	Diorite	Active
74.1312	1	2090	97000	5	Gabbro	Active
74.1313	1	2091	98000	5	Peridotite	Active
74.1314	1	2092	99000	5	Eclogite	Active
74.1315	1	2093	100000	5	Marble	Active
74.1316	1	2094	101000	5	Serpentine	Active
74.1317	1	2095	102000	5	Talc	Active
74.1318	1	2096	103000	5	Pyroxene	Active
74.1319	1	2097	104000	5	Quartzite	Active
74.1320	1	2098	105000	5	Schist	Active
74.1321	1	2099	106000	5	Gneiss	Active
74.1322	1	2100	107000	5	Granite	Active
74.1323	1	2101	108000	5	Basalt	Active
74.1324	1	2102	109000	5	Andesite	Active
74.1325	1	2103	110000	5	Diorite	Active
74.1326	1	2104	111000	5	Gabbro	Active
74.1327	1	2105	112000	5	Peridotite	Active
74.1328	1	2106	113000	5	Eclogite	Active
74.1329	1	2107	114000	5	Marble	Active
74.1330	1	2108	115000	5	Serpentine	Active
74.1331	1	2109	116000	5	Talc	Active
74.1332	1	2110	117000	5	Pyroxene	Active
74.1333	1	2111	118000	5	Quartzite	Active
74.1334	1	2112	119000	5	Schist	Active
74.1335	1	2113	120000	5	Gneiss	Active
74.1336	1	2114	121000	5	Granite	Active
74.1337	1	2115	122000	5	Basalt	Active
74.1338	1	2116	123000	5	Andesite	Active
74.1339	1	2117	124000	5	Diorite	Active
74.1340	1	2118	125000	5	Gabbro	Active
74.1341	1	2119	126000	5	Peridotite	Active
74.1342	1	2120	127000	5	Eclogite	Active
74.1343	1	2121	128000	5	Marble	Active
74.1344	1	2122	129000	5	Serpentine	Active
74.1345	1	2123	130000	5	Talc	Active
74.1346	1	2124	131000	5	Pyroxene	Active
74.1347	1	2125	132000	5	Quartzite	Active
74.1348	1	2126	133000	5	Schist	Active
74.1349	1	2127	134000	5	Gneiss	Active
74.1350	1	2128	135000	5	Granite	Active
74.1351	1	2129	136000	5	Basalt	Active
74.1352	1	2130	137000	5	Andesite	Active
74.1353	1	2131	138000	5	Diorite	Active
74.1354	1	2132	139000	5	Gabbro	Active
74.1355	1	2133	140000	5	Peridotite	Active
74.1356	1	2134	141000	5	Eclogite	Active
74.1357	1	2135	142000	5	Marble	Active
74.1358	1	2136	143000	5	Serpentine	Active
74.1359	1	2137	144000	5	Talc	Active
74.1360	1	2138	145000	5	Pyroxene	Active
74.1361	1	2139	146000	5	Quartzite	Active
74.1362	1	2140	147000	5	Schist	Active
74.1363	1	2141	148000	5	Gneiss	Active
74.1364	1	2142	149000	5	Granite	Active
74.1365	1	2143	150000	5	Basalt	Active
74.1366	1	2144	151000	5	Andesite	Active
74.1367	1	2145	152000	5	Diorite	Active
74.1368	1	2146	153000	5	Gabbro	Active
74.1369	1	2147	154000	5	Peridotite	Active
74.1370	1	2148	155000	5	Eclogite	Active
74.1371	1	2149	156000	5	Marble	Active
74.1372	1	2150	157000	5	Serpentine	Active
74.1373	1	2151	158000	5	Talc	Active
74.1374	1	2152	159000	5	Pyroxene	Active
74.1375	1	2153	160000	5	Quartzite	Active
74.1376	1	2154	161000	5	Schist	Active
74.1377	1	2155	162000	5	Gneiss	Active
74.1378	1	2156	163000	5	Granite	Active
74.1379	1	2157	164000	5	Basalt	Active
74.1380	1	2158	165000	5	Andesite	Active
74.1381	1	2159	166000	5	Diorite	Active
74.1382	1	2160	167000	5	Gabbro	Active
74.1383	1	2161	168000	5	Peridotite	Active
74.1384	1	2162	169000	5	Eclogite	Active
74.1385	1	2163	170000	5	Marble	Active
74.1386	1	2164	171000	5	Serpentine	Active
74.1387	1	2165	172000	5	Talc	Active
74.1388	1	2166	173000	5	Pyroxene	Active
74.1389	1	2167	174000	5	Quartzite	Active
74.1390	1	2168	175000	5	Schist	Active
74.1391	1	2169	176000	5	Gneiss	Active
74.1392	1	2170	177000	5	Granite	Active
74.1393	1	2171	178000	5	Basalt	Active
74.1394	1	2172	179000	5	Andesite	Active
74.1395	1	2173	180000	5	Diorite	Active
74.1396	1	2174	181000	5	Gabbro	Active
74.1397	1	2175	182000	5	Peridotite	Active
74.1398	1	2176	183000	5	Eclogite	Active
74.1399	1	2177	184000	5	Marble	Active
74.1400	1	2178	185000	5	Serpentine	Active
74.1401	1	2179	186000	5	Talc	Active
74.1402	1	2180	187000	5	Pyroxene	Active
74.1403	1	2181	188000	5	Quartzite	Active
74.1404	1	2182	189000	5	Schist	Active
74.1405	1	2183	190000	5	Gneiss	Active
74.1406	1	2184	191000	5	Granite	Active
74.1407	1	2185	192000	5	Basalt	Active
74.1408	1	2186	193000	5	Andesite	Active
74.1409	1	2187	194000	5	Diorite	Active
74.1410	1	2188	195000	5	Gabbro	Active
74.1411	1	2189	196000	5	Peridotite	Active
74.1412	1	2190	197000	5	Eclogite	Active
74.1413	1	2191	198000	5	Marble	Active
74.1414	1	2192	199000	5	Serpentine	Active
74.1415	1	2193	200000	5	Talc	Active
74.1416	1	2194	201000	5	Pyroxene	Active
74.1417	1	2195	202000	5	Quartzite	Active
74.1418	1	2196	203000	5	Schist	Active
74.1419	1	2197	204000	5	Gneiss	Active
74.1420	1	2198	205000	5	Granite	Active
74.1421	1	2199	206000	5	Basalt	Active
74.1422	1	2200	207000	5	Andesite	Active
74.1423	1	2201	208000	5	Diorite	Active
74.1424	1	2202	209000	5	Gabbro	Active
74.1425	1	2203	210000	5</		

# 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

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

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

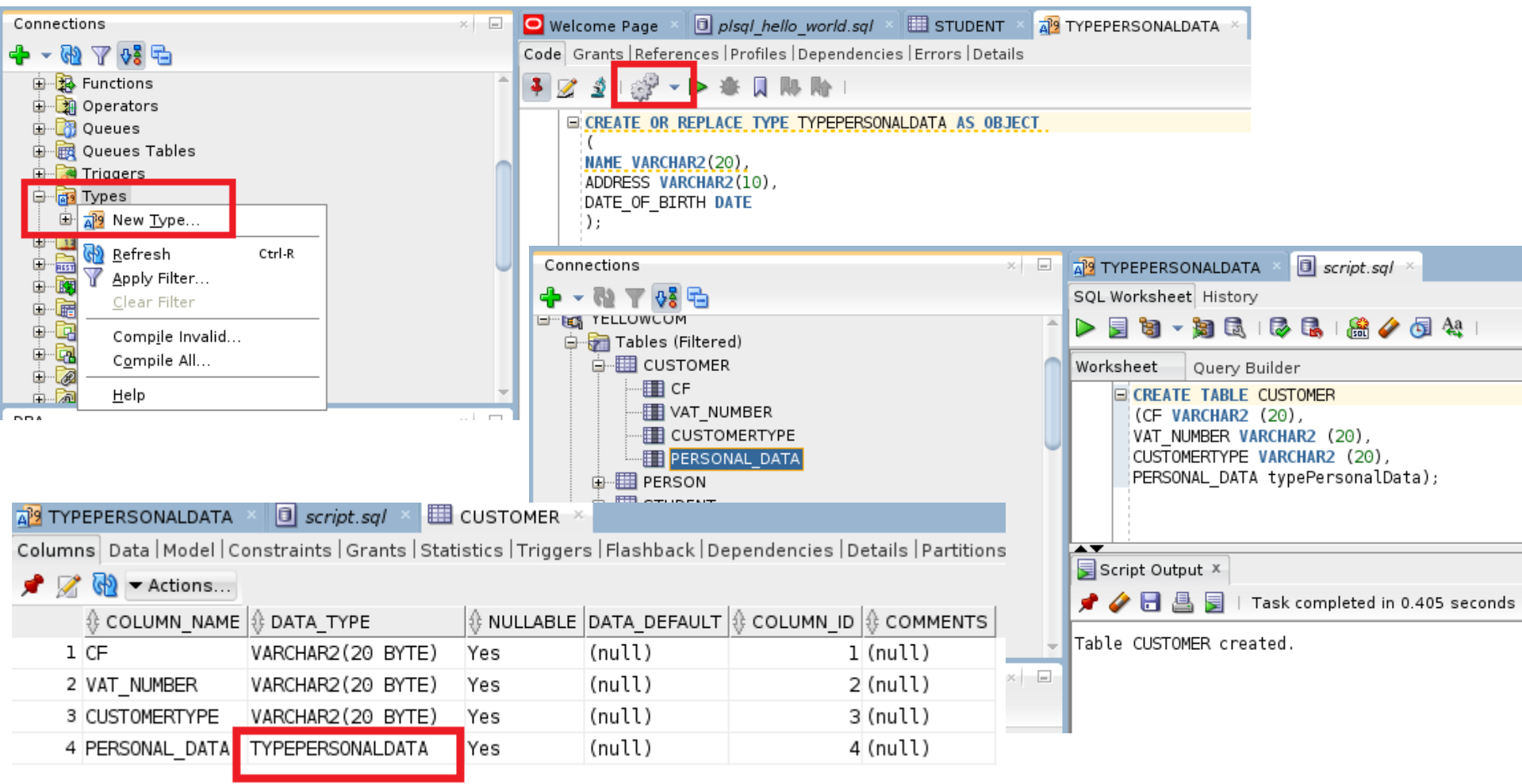
```
CREATE TYPE typePersonalData AS OBJECT  
(NAME VARCHAR2(20),  
ADDRESS VARCHAR2(10),  
DATE_OF_BIRTH DATE);
```

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 illustrates the process of creating and using an abstract data type (ADT) in Oracle SQL Developer. The main window shows the 'CREATE OR REPLACE TYPE TYPEPERSONALDATA AS OBJECT' statement with attributes: NAME VARCHAR2(20), ADDRESS VARCHAR2(10), and DATE\_OF\_BIRTH DATE. The 'Types' folder in the left pane is highlighted, and the 'New Type...' option is selected. The 'Connections' pane shows the 'CUSTOMER' table under the 'PERSONAL\_DATA' schema. The 'Columns' pane displays the table structure, with the 'PERSONAL\_DATA' column type highlighted. The 'Script Output' pane shows the successful execution of the script, confirming the creation of the 'CUSTOMER' table.

**CREATE OR REPLACE TYPE TYPEPERSONALDATA AS OBJECT**

```

(
  NAME VARCHAR2(20),
  ADDRESS VARCHAR2(10),
  DATE_OF_BIRTH DATE
);

```

**Columns** | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CF	VARCHAR2(20 BYTE)	Yes	(null)	1 (null)	
2	VAT_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	2 (null)	
3	CUSTOMERTYPE	VARCHAR2(20 BYTE)	Yes	(null)	3 (null)	
4	PERSONAL_DATA	TYPEPERSONALDATA	Yes	(null)	4 (null)	

**Script Output** | Task completed in 0.405 seconds

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

dot notation to refer to the  
column in question

table alias

name of the abstract  
type column

UPDATE CUSTOMER C SET C.PERSONAL\_DATA.ADDRESS= 'address'  
where C.PERSONAL\_DATA.ADDRESS= 'Address'

The screenshot shows a SQL Worksheet interface with a tab labeled 'script.sql'. The 'Worksheet' tab is active, displaying the following SQL statement:

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

Below the SQL statement, the 'Script Output' pane shows the result: '1 row inserted.' The status bar indicates 'Task completed in 0.279 seconds'.

The screenshot shows a SQL Worksheet interface with a tab labeled 'script.sql'. The 'Worksheet' tab is active, displaying the following SQL statement:

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

Below the SQL statement, the 'Script Output' pane shows the result: '0 rows updated.' The status bar indicates 'Task completed in 0.384 seconds'. A red box highlights the text '0 rows updated.' and a red note explains: '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:

<p>CREATE TYPE <i>typeName</i> AS OBJECT</p> <p>(...</p> <p><i>columnK</i> typeColumnK,</p> <p>...</p> <p>MEMBER FUNCTION</p> <p><i>methodName</i></p> <p>(<i>typeParameter</i> IN <i>parameterName</i>)</p> <p>RETURN returnType</p> <p>);</p>	<p>CREATE TYPE <i>typeName</i> AS OBJECT</p> <p>(...</p> <p><i>columnK</i> typeColumnK,</p> <p>...</p> <p>MEMBER PROCEDURES</p> <p><i>methodName</i></p> <p>(<i>typeParameter</i> IN / OUT <i>parameterName</i>)</p> <p>);</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


 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
```

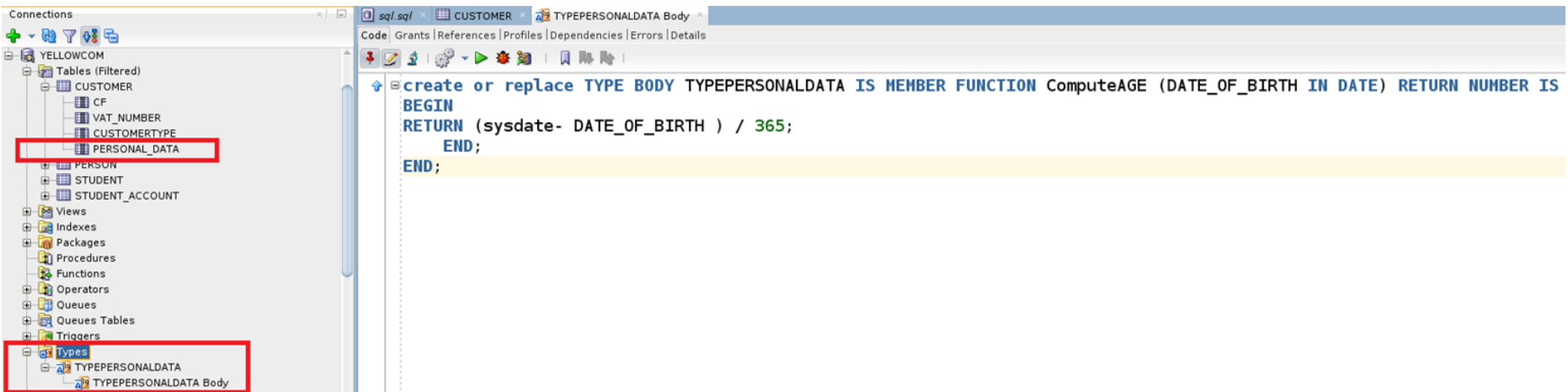
```
BEGIN
```

```
RETURN (sysdate - DATE_OF_BIRTH) / 365;
```

```
END;
```

```
END;
```

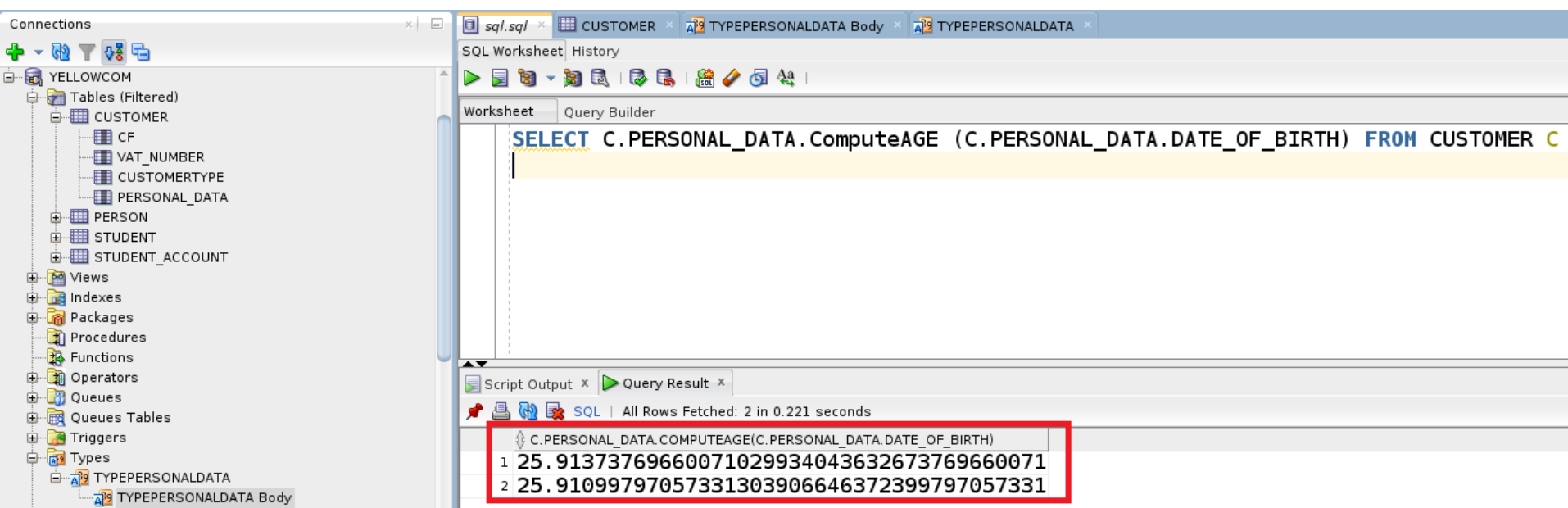
# ...The abstract data types in Oracle ...



# ...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, the 'Connections' pane shows a connection to 'YELLOWCOM'. The 'Tables (Filtered)' pane shows a tree structure with 'CUSTOMER' expanded, containing 'CF', 'VAT\_NUMBER', 'CUSTOMERTYPE', and 'PERSONAL\_DATA'. The 'Worksheet' pane shows the query: `SELECT C.PERSONAL_DATA.ComputeAGE (C.PERSONAL_DATA.DATE_OF_BIRTH) FROM CUSTOMER C`. The 'Query Result' pane shows the results of the query, with two rows highlighted in red:

	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 name VARRAY  
AS VARRAY (size max) OF ObjectType
```

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

```
CREATE TABLE tableName  
(...  
columnname name VARRAY,  
....)
```

# ... 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')
```

constructor

component objects of the VARRAY

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

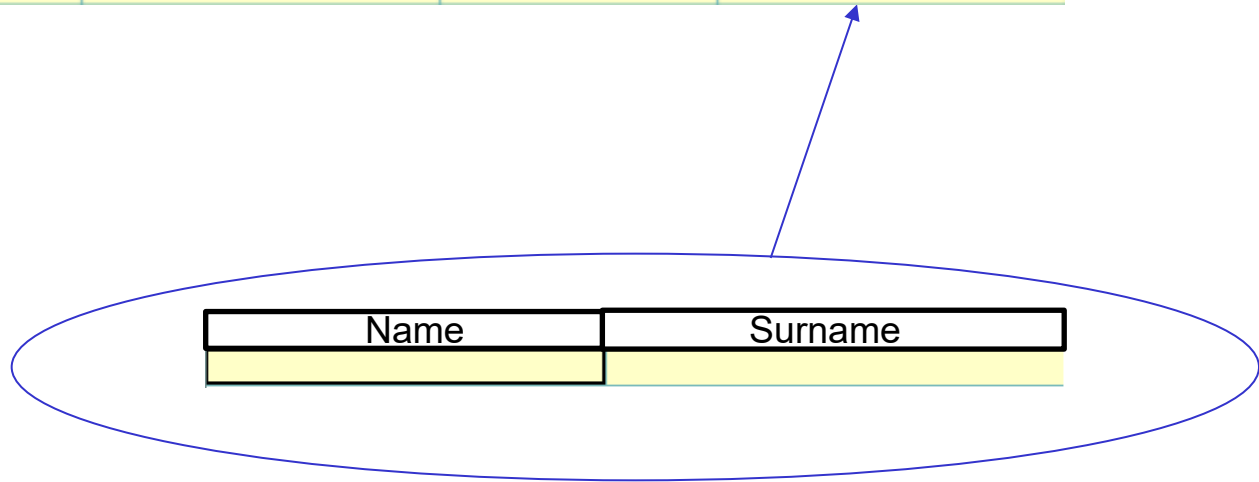
name of the column of type variable array



# 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 MULTISSET

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 MULTISSET in the form:

**CAST (MULTISSET (<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.