Practical – 03

Title: - Abstract Data Types

Aim: - To implement various Abstract datatypes.

Lab Objectives: -

Students will understand various Abstract Data types as follows:

I. CLOB, BLOB
II. Varray
III. Nested Tables
IV. Abstract Data Type
V. Methods
VI. Inheritance

Description: -

Relational database management systems (RDBMSs) are the standard tool for managing business data.

They provide reliable access to huge amounts of data for millions of businesses around the world every day.

Oracle is an **object-relational** database management system (ORDBMS), which means that users can define additional kinds of data--specifying both the structure of the data and the ways of operating on it--and use these types within the relational model.

This approach adds value to the data stored in a database.

User-defined datatypes make it easier for application developers to work with complex data such as images, audio, and video.

Object types store structured business data in its natural form and allow applications to retrieve it that way.

For that reason, they work efficiently with applications developed using object-oriented programming techniques.

Applications

- computer-aided design,
- computer-aided software engineering
- multimedia and image databases, document/hypertext databases.

I. CLOB, BLOB, BFILE

- Large Objects (LOBs) are a set of data types that are designed to hold large amounts of data.
- A LOB can hold up to a maximum size ranging from 8 terabytes to 128 terabytes depending on how your database is configured.
- Storing data in LOBs enables you to access and manipulate the data efficiently in your application.
- The built-in LOB data types BLOB, CLOB and NCLOB (stored internally), and BFILE (stored externally), can store large and unstructured data such as text, images and spatial data up to 4 gigabytes in size.
  - **BLOB**
    - The BLOB data type stores binary large objects. BLOB can store up to 4 gigabytes of binary data.
  - **CLOB**
    - The CLOB data type stores character large objects. CLOB can store up to 4 gigabytes of character data.
  - **NCLOB**
    - The NCLOB data type stores character large objects in multibyte national character set. NCLOB can store up to 4 gigabytes of character data.
  - **BFILE**
    - The BFILE data type enables access to binary file LOBs that are stored in file systems outside the Oracle database. A BFILE column stores a locator, which serves as a pointer to a binary file on the server's file system. The maximum file size supported is 4 gigabytes

Example: CLOB

Creating Tables Containing CLOB Objects

```
SQL> CREATE TABLE MyTable (
  2    id          INTEGER PRIMARY KEY,
  3    clob_column CLOB NOT NULL
  4  );
```

Initialize CLOB column

```
INSERT INTO myTable(id, clob_column) VALUES (1, EMPTY_CLOB());

INSERT INTO myTable(id, clob_column) VALUES (101,to_clob('hello'));
```

Update CLOB Column

```
SQL> UPDATE myTable
  2  SET clob_column = 'AAAAA'
  3  WHERE id = 1;
```

Example: BLOB

Creating Tables Containing BLOB Objects

```
SQL> CREATE TABLE myTable (
```

```
  2    id           INTEGER PRIMARY KEY,
  3    blob_column BLOB NOT NULL
  4  );
```

Initialize CLOB column

```
SQL> INSERT INTO myTable(id, blob_column) VALUES (1, EMPTY_BLOB());
```

Update CLOB Column

```
SQL> UPDATE myTable
  2  SET blob_column = 'VVVVV'
  3  WHERE id = 1;
SET blob_column = 'VVVVV'
```

Example: BFILE

Creating Tables Containing BFILE Objects

```
SQL> CREATE TABLE myTable (
  2    id            INTEGER PRIMARY KEY,
  3    bfile_column BFILE NOT NULL
  4  );
```

Populating BFILE column

```
INSERT INTO myBFile VALUES (1,BFILENAME('BFILE_DIR','test.bmp'));
```

II. Variable-Sized Array (VARRAY)

- Items of type VARRAY are called *varrays*.
- They allow you to associate a single identifier with an entire collection.
- This association lets you manipulate the collection as a whole and reference individual elements easily.
- To reference an element, you use standard subscripting syntax
- A varray has a maximum size, which you must specify in its type definition.
- Its index has a fixed lower bound of 1 and an extensible upper bound.
- Thus, a varray can contain a varying number of elements, from zero (when empty) to the maximum specified in its type definition.
- **The basic Oracle syntax for the CREATE TYPE statement for a VARRAY type definition would be:**

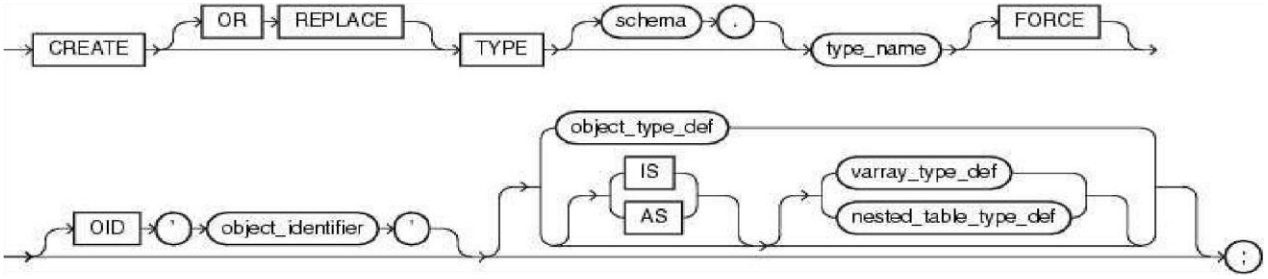
  CREATE OR REPLACE TYPE name-of-type IS VARRAY(nn) of type;

III. Nested Tables

- Within the database, nested tables can be considered one-column database tables.
- Oracle stores the rows of a nested table in no particular order.
- But, when you retrieve the nested table into a PL/SQL variable, the rows are given consecutive subscripts starting at 1.
- That gives you array-like access to individual rows.
- PL/SQL nested tables are like one-dimensional arrays.
- You can model multi-dimensional arrays by creating nested tables whose elements are also nested tables.
- **Syntax**
- **CREATE Or Replace TYPE type_name AS TABLE OF type;**

IV. Abstract Data Types

- **ADT** (Abstract DataType) is a user defined data type (also referred to as UDT's).
- Abstract Datatypes are data types that consist of one or more subtypes.
- Rather than being constrained to the standard Oracle data types of NUMBER, DATA, and VARCHAR2, abstract data types can more accurately describe your data.



Example:

Create type Address

```
CREATE OR REPLACE TYPE address
AS OBJECT
(
  street   char(20),
  city     char(20),
  state    char(2),
  zip      char(5)
);
```

Create a table called test_adt with the following columns, and describe the new test_adt table

4 1

```
CREATE TABLE
  test_adt
(
  first_name      char(20),
  last_name       char(20),
  full_address    address
);
```

Insert five (5) rows into your test_adt table.

```
INSERT INTO test_adt
VALUES ('Joe','Palooka',address('41 Cherise Ave.', 'Minot','ND','66654'));
```

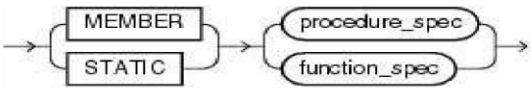
Show only the last_name, zip, and city columns

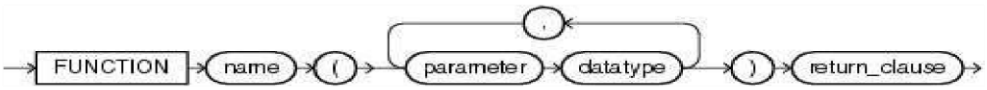
```
SELECT last_name,t.full_address.zip,t.full_address.city FROM test_adt t;
```

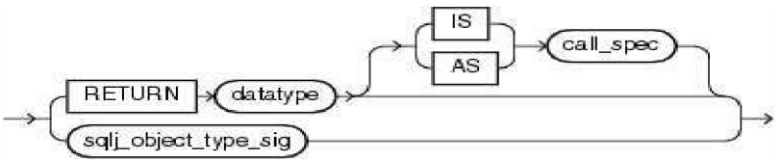
V. Methods/Member functions in Abstract Data Types

A function or procedure subprogram associated with the ADT that is referenced as an attribute. Typically, you invoke MEMBER methods in a selfish style, such as object_expression.method(). This class of method has an implicit first argument referenced as SELF in the method body, which represents the object on which the method was invoked.

Syntax







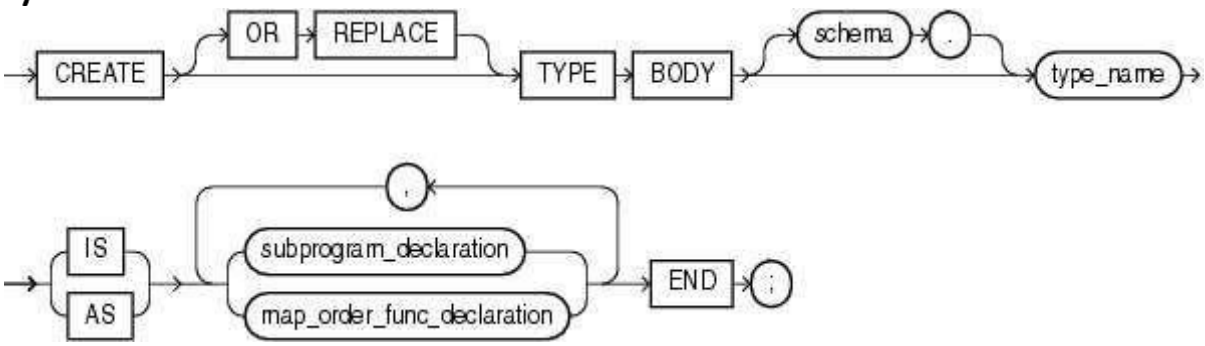
Example:

Create type employee_t with member function raise_sal()

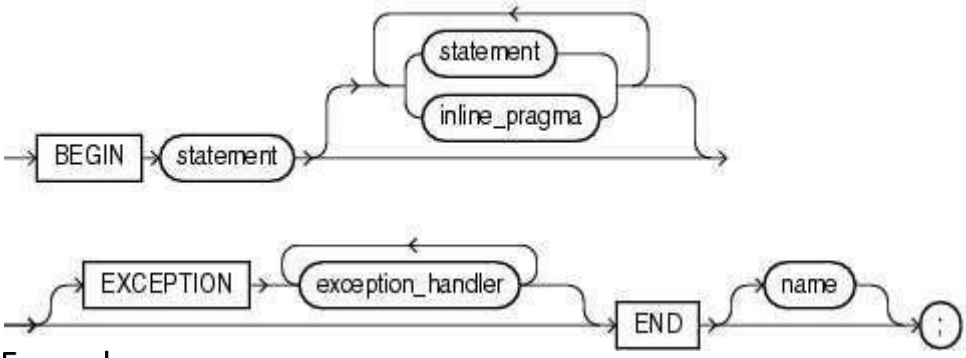
CREATE TYPE employee_t **AS OBJECT**
(name VARCHAR2(30), ssn VARCHAR2(11), salary NUMBER,
MEMBER FUNCTION raise_sal **RETURN NUMBER);**

Create Method

Syntax



Subprogram declaration



Example:

```
CREATE TYPE BODY employee_t AS
    MEMBER FUNCTION raise_sal RETURN NUMBER IS
     BEGIN
       RETURN salary * 2;
     END;
END;
```

Show only the raised salary of emplyoee

```
SELECT e.raise_sal()  from emp2 e;
```

VI. Inheritance

SQL object inheritance is based on a family tree of object types that forms a type hierarchy.

The type hierarchy consists of a parent object type, called a supertype, and one or more levels of child object types, called subtypes, which are derived from the parent.

Inheritance is the mechanism that connects subtypes in a hierarchy to their supertypes.

Subtypes automatically inherit the attributes and methods of their parent type.

Also, the inheritance link remains alive.

Subtypes automatically acquire any changes made to these attributes or methods in the parent: any attributes or methods updated in a supertype are updated in subtypes as well.

Subtypes can have new attributes and new methods that its parent supertype does not have

- **Creating a Parent or Supertype Object**
  - You can create a parent or supertype object using the CREATE TYPE statement.
- **Creating a Subtype Object**
  - A subtype inherits the attributes and methods of the supertype.
  - These are inherited:
    - All the attributes declared in or inherited by the supertype.
    - Any methods declared in or inherited by supertype.
  - Subtypes are created using the keyword UNDER as follows:

CREATE TYPE eng UNDER emp_typ

FINAL and NOT FINAL Types and Methods for Inheritance

Object types can be inheritable and methods can be overridden if they are so defined.

For an object type or method to be inheritable, the definition must specify that it is inheritable.

For both types and methods, the keywords FINAL or NOT FINAL are used are used to determine inheritability.

Object type: For an object type to be inheritable, thus allowing subtypes to be derived from it, the object definition must specify this. NOT FINAL means subtypes can be derived. FINAL, (default) means that no subtypes can be derived from it.

Method: The definition must indicate whether or not it can be overridden. NOT FINAL (default) means the method can be overridden. FINAL means that subtypes cannot override it by providing their own implementation.

Student Name :- Anish Ramakant Karlekar **Div**:-A **Roll No:-**40

Exercise :

Q1-Using object-oriented databases create the following types:

CODE :

a) AddrType (Pincode:number, Street:char, City:char, State:char)
b) BranchType (address: AddrType, phone1: integer, phone2: integer)
c) AuthorType (name:char, address: AddrType)
d) PublisherType (name:char, address:AddrType, branches:BranchType)
e) AuthorListType as varray, which is a reference to AuthorType

OUTPUT :

```
-- AddrType (Pincode:number, Street:char, City:char, State:char)--
CREATE TYPE AddrType AS OBJECT (
    Pincode NUMBER,
    Street VARCHAR2(100),
    City VARCHAR2(50),
    State VARCHAR2(50)
);
--) BranchType (address: AddrType, phone1: integer, phone2: integer)
CREATE TYPE BranchType AS OBJECT (
    address AddrType,
    phone1 INTEGER,
    phone2 INTEGER
);
--AuthorType (name:char, address: AddrType)
CREATE TYPE AuthorType AS OBJECT (
    name VARCHAR2(100),
    address AddrType
);
--PublisherType (name:char, address:AddrType, branches:BranchType)
CREATE TYPE PublisherType AS OBJECT (
    name VARCHAR2(100),
    address AddrType,
    branches BranchType
);
--AuthorListType as varray, which is a reference to AuthorType
CREATE TYPE AuthorListType AS VARRAY(100) OF REF AuthorType;
```

a) Branch of BranchType
b) Authors of AuthorType
c) Books (title:varchar,year:date,published_by ref Publishertype, auhtors AuthorListType)
d) Publishers of PublisherType

```
-- a) Branch Table
CREATE TABLE Branch OF BranchType;

-- b) Authors Table
CREATE TABLE Authors OF AuthorType (
    PRIMARY KEY (name)
);

-- c) Books Table
CREATE TABLE Books (
    title VARCHAR2(200),
    year DATE,
    published_by REF PublisherType,
    authors AuthorListType
);

-- d) Publishers Table
CREATE TABLE Publishers OF PublisherType (
    PRIMARY KEY (name)
);
```

```
Table created.
```

Insert records into the above tables and fire the following queries:

```
-- Inserting into Authors
INSERT INTO Authors VALUES (AuthorType('Harsh', AddrType(123456, 'Main Street', 'Vengurla', 'Maharashtra')));
INSERT INTO Authors VALUES (AuthorType('Ram', AddrType(654321, 'High Street', 'Kudal', 'Maharashtra')));

-- Inserting into Publishers
INSERT INTO Publishers VALUES (
    PublisherType('ABC',AddrType(123456, 'Pearson Street', 'Ratnagiri', 'Maharashtra'),
        BranchType(AddrType(654321, 'Branch Road', 'Ratnagiri', 'Maharashtra'), 1234567890, 9876543210) ) );

-- Inserting into Books
DECLARE
    author1 REF AuthorType;
    author2 REF AuthorType;
    publisher1 REF PublisherType;
BEGIN
    SELECT REF(a) INTO author1 FROM Authors a WHERE a.name = 'Harsh';
    SELECT REF(a) INTO author2 FROM Authors a WHERE a.name = 'Ram';
    SELECT REF(p) INTO publisher1 FROM Publishers p WHERE p.name = 'ABC';

    INSERT INTO Books VALUES ('ADBMS', DATE '2022-10-10', publisher1,AuthorListType(author1, author2));
    INSERT INTO Books VALUES ('ADBMS', DATE '2022-10-10', publisher1,AuthorListType(author1, author2));
END;
/
```

```
Statement processed.
```

a) **List all of the authors that have the same pin code as their publisher:**

```
76
77  SELECT a.name FROM Authors a, Publishers p WHERE a.address.Pincode = p.address.Pincode;
78
```

NAME
Harsh

b) List all books that have 2 or more authors:

```
78
79  SELECT b.title FROM Books b WHERE(SELECT COUNT(*)FROM TABLE(b.authors))>= 2;
80
```

TITLE
ADBMS
ADBMS

c) List the name of the publisher that has the most branches

```
81  SELECT p.name
82  FROM Publishers p
83  WHERE EXISTS (SELECT 1 FROM Branch b WHERE p.branches IS NOT NULL);
84
```

no data found

d) Name of authors who have not published a book

```
85  SELECT a.name FROM Authors a
86      WHERE NOT EXISTS (SELECT 1 FROM Books b
87      WHERE EXISTS(SELECT 1 FROM TABLE(b.authors)
88      WHERE COLUMN_VALUE = REF(a)));
89
```

no data found

e) List all authors who have published more than one book

```
--e
SELECT a.name
FROM Authors a
    JOIN Books b ON EXISTS(
    SELECT 1 FROM TABLE (b.authors)
    WHERE DEREF (COLUMN_VALUE).name =a.name
    )
    GROUP BY a.name
    HAVING COUNT(*) > 1;
```

NAME
Harsh
Ram

f) Name of authors who have published books with at least two different publishers4

```
SELECT a.name
FROM Authors a
    JOIN Books b ON EXISTS(
    SELECT 1 FROM TABLE (b.authors)
    WHERE DEREF (COLUMN_VALUE).name =a.name
    )
    GROUP BY a.name
    HAVING COUNT(DISTINCT b.published_by)>=2;
```

no data found

Q-2) Create Book_type by grouping the information Bookno, Title, and Author. Create table Purchase with Pid, book_details, date, amount. Insert five records in Purchase Table.

```
--Create Book_type by grouping the information Bookno, Title, and Author. Create table Purchase
    --with Pid, book_details, date, amount. Insert five records in Purchase Table.
CREATE TYPE Book_type AS OBJECT (
    Bookno NUMBER,
    Title VARCHAR2(200),
    Author VARCHAR2(100)
);

CREATE TABLE Purchase (
    Pid NUMBER PRIMARY KEY,
    book_details Book_type,
    purchase_date DATE,
    amount NUMBER
);

--insert values
INSERT INTO Purchase VALUES (
    1,
    Book_type(101, 'The Great Gatsby', 'F. Scott Fitzgerald'),
    TO_DATE('2023-10-01', 'YYYY-MM-DD'),
    500
);

INSERT INTO Purchase VALUES (
    2,
    Book_type(102, 'To Kill a Mockingbird', 'Harper Lee'),
    TO_DATE('2023-10-15', 'YYYY-MM-DD'),
    400
);

INSERT INTO Purchase VALUES (
    3,
    Book_type(103, '1984', 'George Orwell'),
    TO_DATE('2023-11-01', 'YYYY-MM-DD'),
    450
);

INSERT INTO Purchase VALUES (
    4,
    Book_type(104, 'Pride and Prejudice', 'Jane Austen'),
    TO_DATE('2023-11-10', 'YYYY-MM-DD'),
    350
);
```

```
1 row(s) inserted.
```

Q-3) Create a table customer with the attributes cust_no, cust_name, product and price. Create anADT name_type with the attribute fname, mname and lname to store the name details. Display the first name of the customer who purchased'Monitor'.

```
--Create a table customer with the attributes cust_no, cust_name, product and price.
   --Create an  ADT name_type with the attribute fname, mname and lname to store the name details.
   --Display the first name of the customer who purchased'Monitor'.

--Create name_type ADT
CREATE TYPE name_type AS OBJECT (
    fname VARCHAR2(50),
    mname VARCHAR2(50),
    lname VARCHAR2(50)
);
--Create Customer Table
CREATE TABLE Customer (
    cust_no NUMBER PRIMARY KEY,
    cust_name name_type,
    product VARCHAR2(100),
    price NUMBER
);
```

Insert record

```
--insert records
INSERT INTO Customer VALUES (
    1,
    name_type('Harsh', 'Arvind', 'Bagaytkar'),
    'Monitor',
    15000
);

INSERT INTO Customer VALUES (
    2,
    name_type('Tanvi ', 'Rajesh ', 'Humraskar'),
    'Keyboard',
    2000
);

INSERT INTO Customer VALUES (
    3,
    name_type('Harsh', 'R', 'Humraskar'),
    'Mouse',
    800
);

INSERT INTO Customer VALUES (
    4,
    name_type('Dharmesh', 'A.', 'Munankar'),
    'Monitor',
    18000
);
```

Query to Display the First Name of Customers Who Purchased "Monitor"

```
99
00  SELECT c.cust_name.fname AS FirstName
01  FROM Customer c
02  WHERE c.product = 'Monitor';
03
04
```

FIRSTNAME
Harsh
Dharmesh

Q-4) Create person type with attributes person_id, person_name and person_addr. Create a person_obj table of person type. Insert and display the details of the table.

```
111 ------------------------------------------------------------
112 --Q4
113 CREATE TYPE PersonType AS OBJECT (
114     person_id NUMBER,
115     person_name VARCHAR2(100),
116     person_addr VARCHAR2(200)
117 );
118
119 CREATE TABLE person_obj OF PersonType ;
120
121 INSERT INTO person_obj VALUES (PersonType(1, 'John ', ' Main Street, Ratnagiri, Maharashtra'));
122 INSERT INTO person_obj VALUES (PersonType(2, 'Roy', 'City Street, Sindhudurg, Maharashtra'));
123
124 SELECT * FROM person_obj;
125
```

PERSON_ID	PERSON_NAME	PERSON_ADDR
1	John	Main Street, Ratnagiri, Maharashtra
2	Roy	City Street, Sindhudurg, Maharashtra

Q-5) Create type rectangle with attributes length, breadth and member function rect_area that returns area of the rectangle. Create table shape of rectangle type & insert record into it. Display the length, breadth and area of rectangles.

```
127  CREATE TYPE Rectangle AS OBJECT (
128      length NUMBER,
129      breadth NUMBER,
130      MEMBER FUNCTION rect_area RETURN NUMBER
131  );
132
133  CREATE OR REPLACE TYPE BODY Rectangle AS
134      MEMBER FUNCTION rect_area RETURN NUMBER IS
135      BEGIN RETURN length * breadth;
136      END;
137  END;
138
139  CREATE TABLE shape OF Rectangle;
140
141  INSERT INTO shape VALUES (Rectangle(5, 10));
142  INSERT INTO shape VALUES (Rectangle(8, 12));
143
144  SELECT t.length AS length, t.breadth AS breadth,t.rect_area() As Area from shape t;
```

LENGTH	BREADTH	AREA
5	10	50
8	12	96

Q-6) Create type solid_type with attributes length, width, height and 2 member functions surface and volume that returns the surface area and volume of the shape respectively. Create table solid of solid_type, insert records into it and display surface and volume of the solid.

```
147  CREATE TYPE solid_type AS OBJECT (length NUMBER, width NUMBER,height NUMBER,
148      MEMBER FUNCTION surface RETURN NUMBER, MEMBER FUNCTION volume RETURN NUMBER);
149
150  CREATE OR REPLACE TYPE BODY solid_type AS
151      MEMBER FUNCTION surface RETURN NUMBER IS
152      BEGIN RETURN 2 * (length * width + width * height + length * height);
153      END;
154      MEMBER FUNCTION volume RETURN NUMBER IS
155      BEGIN RETURN length * width * height;
156      END;
157  END;
158
159  CREATE TABLE solid OF solid_type;
160
161  INSERT INTO solid VALUES (solid_type(5, 4, 3));
162
163  SELECT t.length ,t.width, t.height,
164      t.surface()As Surface_Area,
165      t.volume()As volume FROM solid t;
```

LENGTH	WIDTH	HEIGHT	SURFACE_AREA	VOLUME
5	4	3	94	60

Q-7) Create supertype person_typ with attributes id, name, phone_no and member function show that returns id and name of the person. Create table person of person_typ and insert records into it and display id and name of person using show function. Create subtype student_type of spertype person_typ with attributes dept_id and major. It has member function show that overrides member function of person_typ and returns the major of student. Create table student of student_type and insert record into it and display major of the student using show function.

```
--Q7
CREATE TYPE person_typ AS OBJECT (id NUMBER, name VARCHAR2(100), phone_no VARCHAR2(15),
    MEMBER FUNCTION show RETURN VARCHAR2
);

CREATE OR REPLACE TYPE BODY person_typ AS
    MEMBER FUNCTION show RETURN VARCHAR2 IS
    BEGIN RETURN 'ID: ' || id || ', Name: ' || name;
    END;
END;

CREATE TABLE person OF person_typ;

INSERT INTO person VALUES (person_typ(1, 'Kunal', '1234567890'));
INSERT INTO person VALUES (person_typ(2, 'Sujal', '9876543210'));

SELECT VALUE(p).show() AS Person_Details FROM person p;
```

PERSON_DETAILS
ID: 2, Name: Sujal
ID: 1, Name: Kunal

```
CREATE TABLE student OF student_type;
```

Table created.

```
INSERT INTO student VALUES (student_type(1, 'Alice Johnson', '1234509876', 101, 'Computer Science'));
INSERT INTO student VALUES (student_type(2, 'Bob Williams', '9876501234', 102, 'Mathematics'));
```

1 row(s) inserted.

1 row(s) inserted.

```
39  SELECT t.show() AS Person_Details
40  FROM person t;
41
```

PERSON_DETAILS
ID: 1, Name: John Doe
ID: 2, Name: Jane Smith

```
42  SELECT t.show() AS Student_Major
43  FROM student t;
44
```

STUDENT_MAJOR
Major: Computer Science
Major: Mathematics