Bases de Datos Objeto Relaciones

1

Introducción

- Los modelos de bases de datos relacionales han tenido que adaptarse a la orientación a objetos, por dos aspectos:
  - La utilización y proliferación de lenguajes de programación orientados a objetos como C++, C# o Java.
  - La aparición de aplicaciones más sofisticadas con unos requerimientos de almacenamiento distintos, con necesidad de nuevos tipos de datos.
- Las Bases de Datos Objeto-Relacionales (BDOR) son una extensión de las bases de datos relacionales a las que se les ha añadido conceptos del modelo orientado a objetos.

2

Características

- Creación de Nuevos tipos de datos.
- Soporte para crear Métodos para los tipos de datos.
- Creación de funciones miembro, usando tipos de datos definidos por el usuario permitiendo flexibilidad y seguridad.
- Permite almacenar datos complejos sin necesidad de forzar los tipos de datos tradicionales.
- Compatible con las Base de datos relacionales (se puede pasar de relacionales a objeto relacional).
- Aunque las estructuras de datos son tablas, pero permite utilizar los mecanismos de orientación a objetos para definir y acceder a los datos.
- Una columna de tabla puede ser construida a partir de un tipo de dato definido (multivaluado).
- Tablas anidadas.
- Existen relaciones de asociación y de agregación.

3

Tipos de datos definidos por el usuario

- Es una estructura y un comportamiento común para un conjunto de datos de las aplicaciones.
- Categorías:
  - Tipos de objetos (***Object type***)
  - Tipos para colecciones (***Collection type***)

4

ORACLE DATABASE

Oracle® Database (commonly referred to as Oracle RDBMS or simply as Oracle) is an object-relational database management system produced and marketed by Oracle Corporation.

5

Tipos de objetos (I)

- Representa una entidad del mundo real y se compone de:
  - Nombre
  - Atributos: pueden ser de tipo de dato básico o de un tipo de usuario.
  - Métodos: en PL/SQL.

6

Tipos de objetos (II)

- Sintaxis:

  CREATE [OR REPLACE] **TYPE** nombre_objeto AS OBJECT (

  Atributo tipos_dato, …,

  [MEMBER FUNCTION nombre_método RETURN tipo_dato])

- Para borrar un tipo:

  DROP TYPE nombre_tipo;

7

Tipos de objetos (III)

- Ejemplo:

```
CREATE OR REPLACE TYPE DIRECCION AS OBJECT (
CALLE  VARCHAR2(25),
CIUDAD  VARCHAR2(29),
COD_POSTAL  NUMBER(5) );
/
CREATE OR REPLACE TYPE PERSONA AS OBJECT (
CODIGO NUMBER,
NOMBRE  VARCHAR2(35),
DIREC   DIRECCION,
FECHA_NAC DATE );
/
```

8

Tipos de objetos (IV)

- Al declarar objetos de los tipos anteriores en un bloque de código PL/SQL, hay que inicializarlos. Ejemplo de utilización:

```
DECLARE
        DIR DIRECCION:= DIRECCION(NULL, NULL, NULL);
        P PERSONA:= PERSONA(NULL, NULL, NULL, NULL);
        DIR2 DIRECCION;          -- se iniciará con NEW
        P2 PERSONA;              -- se iniciará con NEW
BEGIN
        DIR.CALLE:= 'Toro, 18';
        DIR.CIUDAD:= 'Salamanca';
        DIR.COD_POSTAL:= 37002;
        P.CODIGO:= 1;
        P.NOMBRE:= 'Elisa Gil Ramos';
        P.DIREC:= DIR;
        P.FECHA_NAC:= '10/10/1980';
        DBMS_OUTPUT.PUT_LINE('NOMBRE: ' || P.NOMBRE || ' * CALLE:' || P.DIREC.CALLE);
        …
        DIR2:= NEW DIRECCION('C/ Luna 23', 'Sevilla', 64500);
        P2:= NEW PERSONA(2, 'Ana Martín Esteban', DIR2, SYSDATE);
END;
/
```


Tipos de objetos (V)

- **Métodos:**
- Son los encargados de definir el comportamiento de los objetos y permiten actuar sobre ellos.
- Son procedimientos y funciones que se especifican después de los atributos y pueden ser de varios tipos:
  - **MEMBER**: son métodos que sirven para actuar con los objetos. Pueden ser procedimientos y funciones.
  - **STATIC**: son métodos independientes de las instancias de los objetos. Pueden ser procedimientos y funciones.
  - **CONSTRUCTOR**: son un tipo especial de métodos que sirven para inicializar los objetos. Son funciones que actúan sobre los valores de los atributos del objeto y devuelven el objeto inicializado. Llevan en la cláusula RETURN la expresión RETURN SELF AS RESULT.

Tipos de objetos (VI)

- Una nueva versión del tipo DIRECCION pero ahora con métodos y constructor:

```
CREATE OR REPLACE TYPE DIRECCION AS OBJECT (
CALLE  VARCHAR2(25),
CIUDAD  VARCHAR2(29),
COD_POSTAL  NUMBER(5),

MEMBER PROCEDURE SET_CALLE(C  VARCHAR2),
MEMBER FUNCTION GET_CALLE RETURN   VARCHAR2,

CONSTRUCTOR FUNCTION DIRECCION (C VARCHAR2, CI
VARCHAR2, CP NUMBER) RETURN SELF AS RESULT );
/
```


Tipos de objetos (VII)

- Una vez creado el tipo es necesario crear el cuerpo del nuevo tipo mediante la instrucción CREATE OR REPLACE TYPE BODY, con la sintaxis:

```
CREATE OR REPLACE TYPE BODY nombre_del_tipo AS
        <implementación de los métodos>
END;
```

- Donde <implementación de los métodos> puede ser:

```
[STATIC | MEMBER] PROCEDURE nombreProc [( param1, param2, …)]
IS
        Declaraciones;
BEGIN
        Instrucciones;
END;

[STATIC | MEMBER | CONSTRUCTOR] FUNCTION nombreFunc [( param1,
param2, …)] RETURN tipo_retorno
IS
        Declaraciones;
BEGIN
        Instrucciones;
END;
```

Tipos de objetos (VIII)

- Ejemplo para el tipo DIRECCION:

```
CREATE OR REPLACE TYPE BODY DIRECCION AS
    MEMBER PROCEDURE SET_CALLE ( C VARCHAR2) IS
    BEGIN
            CALLE:= C;
    END;

    MEMBER FUNCTION GET_CALLE RETURN VARCHAR2 IS
    BEGIN
            RETURN CALLE;
    END;

    CONSTRUCTOR FUNCTION DIRECCION ( C VARCHAR2, CI VARCHAR2, CP NUMBER)
    RETURN SELF AS RESULT IS
    BEGIN
            SELF.CALLE:= C;
            SELF.CIUDAD:= CI;
            SELF.COD_POSTAL:= CP;
            RETURN;
    END;
END;
```


Tipos de objetos (IX)

- Otro ejemplo:

```
CREATE OR REPLACE TYPE RECTANGULO AS OBJECT
(
  BASE   NUMBER,
 ALTURA NUMBER,
 AREA   NUMBER,
 STATIC PROCEDURE PROC1 (ANCHO INTEGER,ALTO INTEGER),
 MEMBER PROCEDURE PROC2 (ANCHO INTEGER,ALTO INTEGER),
 CONSTRUCTOR FUNCTION RECTANGULO (BASE NUMBER,ALTURA
   NUMBER)
                RETURN SELF AS RESULT
);
/


CREATE TABLE TABLAREC (VALOR INTEGER);
/
```

Tipos de objetos (X)

- Otro ejemplo:

```
CREATE OR REPLACE TYPE BODY RECTANGULO AS
 CONSTRUCTOR FUNCTION RECTANGULO (BASE NUMBER, ALTURA NUMBER)    RETURN SELF AS RESULT IS
 BEGIN
  SELF.BASE := BASE;
  SELF.ALTURA := ALTURA;
  SELF.AREA := BASE * ALTURA;
  RETURN;
 END;
 STATIC PROCEDURE PROC1 (ANCHO INTEGER, ALTO INTEGER) IS
 BEGIN
  INSERT INTO TABLAREC VALUES(ANCHO*ALTO);
  --ALTURA := ALTO; --ERROR NO SE PUEDE ACCEDER A LOS ATRIBUTOS DEL TIPO
  DBMS_OUTPUT.PUT_LINE('FILA INSERTADA');
  COMMIT;
 END;
MEMBER PROCEDURE PROC2 (ANCHO INTEGER, ALTO INTEGER) IS
 BEGIN
  SELF.ALTURA := ALTO; --SE PUEDE ACCEDER A LOS ATRIBUTOS DEL TIPO
  SELF.BASE := ANCHO;
  AREA := ALTURA*BASE;
  INSERT INTO TABLAREC VALUES(AREA);
  DBMS_OUTPUT.PUT_LINE('FILA INSERTADA');
  COMMIT;
 END;
END;
/
```


Tipos de objetos (XI)

- Y se utilizaría:

```
DECLARE
 R1 RECTANGULO;
 R2 RECTANGULO;
 R3 RECTANGULO := RECTANGULO(NULL, NULL, NULL);
BEGIN
 R1 := NEW RECTANGULO(10, 20, 200);
 DBMS_OUTPUT.PUT_LINE('AREA R1: '||R1.AREA);

 R2 := NEW RECTANGULO(10,20);
 DBMS_OUTPUT.PUT_LINE('AREA R2: '||R2.AREA);

 R3.BASE := 5;
 R3.ALTURA := 15;
 R3.AREA := R3.BASE * R3.ALTURA;
 DBMS_OUTPUT.PUT_LINE('AREA R3: '||R3.AREA);

 --USO DE LOS MÉTODOS DEL TIPO  RECTANGULO
 RECTANGULO.PROC1(10, 20);   --LLAMADA AL MÉTODO STATIC
 --RECTANGULO.PROC2(20, 30); --ERROR, LLAMADA AL MÉTODO MEMBER
 --R1.PROC1(5, 6);          --ERROR, LLAMADA AL MÉTODO STATIC
 R1.PROC2(5, 10);          --LLAMADA AL MÉTODO MEMBER
END;
/
```

Tipos de objetos (XII)

- Los métodos **MAP** y **ORDER** se utilizan para poder ordenar colecciones de elementos OBJECT, al menos se debe definir uno de ellos para cada tipo de objeto que se quiera comparar.
- Los métodos **MAP** son funciones que devuelven un valor de tipo escalar (CHAR, VARCHAR2, NUMBER, DATE, …) que será el que se utilice en las comparaciones y ordenaciones.
- Los métodos **ORDER** utilizan los atributos del objeto sobre el que se ejecuta para realizar un cálculo y compararlo con otro objeto del mismo tipo que toma como argumento de entrada. El método devuelve un valor negativo, cero o positivo, dependiendo de si el primer elemento es mayor, igual o menor que el segundo. Suelen ser menos funcionales y eficientes, se utilizan cuando el criterio de comparación es complejo como para implementarlo con un MAP.

17

Tipos de objetos (XIII)

- Ejemplo con MAP para el caso PERSONA:

```
CREATE OR REPLACE TYPE PERSONA AS OBJECT (
   CODIGO NUMBER,
   NOMBRE VARCHAR2(35),
   DIREC  DIRECCION,
   FECHA_NAC DATE,
   MAP MEMBER FUNCTION POR_CODIGO RETURN NUMBER );
/
CREATE OR REPLACE TYPE BODY PERSONA AS
   MAP MEMBER FUNCTION POR_CODIGO RETURN NUMBER IS
   BEGIN
         RETURN CODIGO;
   END;
END;
/
```

18

Tipos de objetos (XIV)

- El siguiente código compara dos objetos de tipo PERSONA y visualizará que son iguales ya que tienen el mismo valor de CODIGO:

```
DECLARE
   P1 PERSONA:= PERSONA(NULL, NULL, NULL,NULL);
   P2 PERSONA:= PERSONA(NULL, NULL, NULL,NULL);
BEGIN
   P1.CODIGO:= 1;
   P1.NOMBRE:= 'Elisa Gil Ramos';
   P2.CODIGO:= 1;
   P2.NOMBRE:= 'Pedro Alonso Conde';
   IF P1=P2 THEN
         DBMS_OUTPUT.PUT_LINE('Objetos iguales');
   ELSE
         DBMS_OUTPUT.PUT_LINE('Objetos distintos');
   END IF;
END;
/
```

19

Tipos de objetos (XV)

- Ejemplo con MAP para el caso RECTANGULO:

```
CREATE TYPE RECTANGULO AS OBJECT (
  ancho NUMBER,
  alto NUMBER,
  MAP MEMBER FUNCTION area RETURN NUMBER);
/
CREATE TYPE BODY RECTANGULO AS
MAP MEMBER FUNCTION area RETURN NUMBER IS
BEGIN
   RETURN ancho * alto;
    END area;
END; /
```

20

Tipos de objetos (XVI)

- Ejemplo con ORDER:

```
CREATE TYPE LOCALIZACION AS OBJECT (
 num_edificio NUMBER,
 ciudad VARCHAR2(40),
 ORDER MEMBER FUNCTION igualar (localiz LOCALIZACION) RETURN INTEGER );
/

CREATE TYPE BODY LOCALIZACION AS
ORDER MEMBER FUNCTION igualar (localiz LOCALIZACION) RETURN INTEGER IS
BEGIN
    IF num_edificio < localiz.num_edificio
    THEN RETURN -1; -- vale cualquier número negativo
    ELSIF num_edificio > num_edificio
    THEN RETURN 1; -- cualquier número positivo
    ELSE RETURN 0;
    END IF;
END;
END;/
```

21

Tablas de Objetos (I)

- Una vez definidos los objetos, se pueden utilizar para definir nuevos tipos, para definir columnas de tablas de ese tipo o para definir tablas que almacenan objetos.
- Las tablas de objetos definen un objeto en cada fila. Los atributos se comportarían como las columnas de la tabla.
- Ejemplo:

```
CREATE TABLA ALUMNOS OF PERSONA (
        CODIGO PRIMARY KEY);

INSERT INTO ALUMNOS VALUES(1, 'Elisa Gil Ramos',
DIRECCION('Toro, 18', 'Salamanca',37002), '10/10/1980');

INSERT INTO ALUMNOS (CODIGO, NOMBRE, DIREC, FECHA_NAC)
VALUES(2, 'Pedro Gel Mos',
DIRECCION('Toros, 8', 'Soria',27002), '1/12/1970');
```

22

Tablas de Objetos (II)

- Otra forma de insertar mediante un bloque PL/SQL:

```
DECLARE
  DIR DIRECCION:= DIRECCION('Toro, 18', 'Salamanca',37002);
  PER PERSONA:= PERSONA(1, 'Elisa Gil Ramos', DIR, '10/10/1980');

BEGIN
  INSERT INTO ALUMNOS VALUES(PER);
  COMMIT;
END;
/
```


Tablas de Objetos (III)

- Consultas sobre este tipo de tablas. Ejemplos:

```
SELECT * FROM ALUMNOS A WHERE A.DIREC.CIUDAD= 'Salamanca';

SELECT CODIGO, A.DIREC FROM ALUMNOS A;

SELECT NOMBRE, A.DIREC.GET_CALLE() FROM ALUMNOS A;

UPDATE ALUMNOS A SET A.DIREC.CIUDAD=LOWER(A.DIREC.CIUDAD) WHERE
A.DIREC.CIUDAD='SALAMANCA';
DELETE ALUMNOS A WHERE A.DIREC.CIUDAD='salamanca';

DECLARE
        CURSOR C1 IS SELECT * FROM ALUMNOS;
BEGIN
        FOR I IN C1 LOOP
                DBMS_OUTPUT.PUT_LINE(I.NOMBRE || ' – Calle: ' || I.DIREC.CALLE);
        END LOOP;
END;
/
```

Tipos para Colecciones

- Se pueden almacenar colecciones de elementos en una única columna. Pueden ser un array u otra tabla.
- En el caso de Oracle se dispone de los VARRAYS, similares a los arrays de C, que permiten almacenar un conjunto de elementos del mismo tipo, y disponen de un índice asociado.

- Para crear un VARRAY:

```
CREATE TYPE TELEFONO AS VARRAY(3) OF VARCHAR2(9);
```

- Para crear una tabla con una columna de tipo VARRAY:

```
CREATE TABLE AGENDA (
   NOMBRE VARCHAR2(),
   TELEF TELEFONO );
```

25

Tipos para Colecciones. VARRAY (I)

- Para insertar filas:

```
INSERT INTO AGENDA VALUES ('Antonio',
TELEFONO('656060111','923181111','656060222'));

INSERT INTO AGENDA (NOMBRE, TELEF) VALUES (
'Marta', TELEFONO('646012345'));
```

- En las consultas no se puede poner condiciones sobre los elementos del VARRAY, y además los elementos del VARRAY sólo pueden ser accedidos y recuperados como bloque, no a los elementos individuales (pero sí desde un programa PL/SQL):

```
SELECT TELEF FROM AGENDA;
SELECT A.TELEF FROM AGENDA A;
UPDATE AGENDA SET TELEF=TELEFONO('646200855','657776666')
WHERE NOMBRE='Marta';
```

26

Tipos para Colecciones. VARRAY (II)

- Recorrer elementos de un VARRAY desde un programa:

```
DECLARE
        CURSOR C1 IS SELECT * FROM AGENDA;
        CAD VARCHAR2(50);
BEGIN
        FOR I IN C1 LOOP
                DBMS_OUTPUT.PUT_LINE(I.NOMBRE || ', Teléfonos: ' ||
I.TELEF.COUNT);
                CAD:='*';
                FOR J IN 1..I.TELEF.COUNT LOOP
                        CAD:= CAD || I.TELEF(J) || '*';
                END LOOP;
                DBMS_OUTPUT.PUT_LINE(CAD);
        END LOOP;
END;
/
```


Tipos para Colecciones. Métodos (I)

- Para obtener información de las colecciones se dispone de los métodos:
  - **COUNT**. Devuelve el número de elementos de la colección.
  - **EXISTS**. Devuelve TRUE si la fila existe.
  - **FIRST/LAST**. Devuelve el índice del primer y último elemento de la colección.
  - **NEXT/PRIOR**. Devuelve el elemento próximo o anterior al actual.
  - **LIMIT**. Número máximo de elementos que puede tener la colección.
- Para modificar los elementos de la colección:
  - **DELETE**. Elimina todos los elementos de la colección.
  - **EXTEND**. Añade un elemento nulo a la colección.
  - **EXTEND(n)**. Añade n elementos nulos.
  - **TRIM**. Elimina el último elemento de la colección.
  - **TRIM(n)**. Elimina n elementos del final de la colección.

Tipos para Colecciones. Métodos (II)

```
DECLARE
  TEL TELEFONO := TELEFONO(NULL, NULL, NULL);
BEGIN
  SELECT TELEF INTO  TEL FROM AGENDA WHERE NOMBRE = 'MARTA';

  --Visualizar Datos
  DBMS_OUTPUT.PUT_LINE('Nº DE TELÉFONOS ACTUALES:  ' || TEL.COUNT);
  DBMS_OUTPUT.PUT_LINE('ÍNDICE DEL PRIMER ELEMENTO: ' || TEL.FIRST);
  DBMS_OUTPUT.PUT_LINE('ÍNDICE DEL ÚLTIMO ELEMENTO: ' || TEL.LAST);
  DBMS_OUTPUT.PUT_LINE('MÁXIMO Nº DE TLFS PERMITIDO:' || TEL.LIMIT);

  --Añade un número de teléfono a MARTA
  TEL.EXTEND;
  TEL(TEL.COUNT):= '123000000';
  UPDATE AGENDA A SET A.TELEF = TEL WHERE NOMBRE = 'MARTA';

  --Elimina un teléfono
  SELECT TELEF INTO TEL FROM AGENDA WHERE NOMBRE = 'MANUEL';
  TEL.TRIM;   --Elimina el último elemento del array
  TEL.DELETE; --Elimina todos los elementos
  UPDATE AGENDA A SET A.TELEF = TEL WHERE NOMBRE = 'MANUEL';
END;
```


Tipos para Colecciones. Tablas anidadas (I)

- Las tablas anidadas están formadas por un conjunto de elementos todos del mismo tipo. La tabla anidada está contenida en una columna y el tipo de esta columna tiene que ser un objeto previamente definido. No se especifica el tamaño máximo de una tabla anidada.
- Para crear una tabla anidada:

```
CREATE TYPE TABLA_ANIDADA AS TABLE OF DIRECCION;
```

- Para definir una columna cuyos elementos son tablas:

```
CREATE TABLE EJ_TABLA_ANIDADA (
   ID  NUMBER(2),
   APELLIDOS  VARCHAR2(35),
   DIREC  TABLA_ANIDADA )
   NESTED TABLE DIREC STORE AS DIREC_ANIDADA;
```

Tipos para Colecciones. Tablas anidadas (II)

- La cláusula NESTED TABLE identifica el nombre de la columna que contendrá la tabla anidada.
- La cláusula STORE AS especifica el nombre de la tabla (DIREC_ANIDADA) en la que se van a almacenar las direcciones que se representan en el atributo DIREC de cualquier objeto de la tabla EJ_TABLA_ANIDADA.

Tipos para Colecciones. Tablas anidadas (III)

- Ejemplos:

```
INSERT INTO EJ_TABLA_ANIDADA VALUES(1, 'Gil Ramos', TABLA_ANIDADA(
DIRECCION('Toro 18', 'Salamanca', 37002),
DIRECCION('Sol 22', 'Salamanca', 37001),
DIRECCION('Jamaica 123', 'Burgos', 56001));

INSERT INTO EJ_TABLA_ANIDADA VALUES(2, 'Miño Ríos', TABLA_ANIDADA(
DIRECCION('Sol 18', 'Madrid', 28002),
DIRECCION('Callao 222', 'Madrid', 28002));
```

Tipos para Colecciones. Tablas anidadas (IV)

- Ejemplos de consultas:

```
SELECT ID, APELLIDOS, CURSOR(SELECT TT.CALLE FROM
TABLE(T.DIREC) TT) FROM EJ_TABLA_ANIDADA T;
```

- Se pueden realizar subconsultas mediante la cláusula **THE**:

```
SELECT CALLE FROM THE (SELECT T.DIREC FROM
EJ_TABLA_ANIDADA T WHERE ID=1) WHERE
CIUDAD='Salamanca';
```

- Se puede insertar una dirección al final de la tabla anidada:

```
INSERT INTO TABLE (SELECT DIREC FROM
EJ_TABLA_ANIDADA WHERE ID=1) VALUES (DIRECCION
('Los Robles 44', 'Soria', 23009));
```


Referencias (I)

- Mediante el operador REF asociado a un atributo se pueden definir referencias a otros objetos. De esta forma se implementa una relación de asociación entre los dos tipos de objetos.
- Una columna de tipo REF contendrá un puntero a una fila de la otra tabla, contiene su OID (identificador del objeto fila).

```
CREATE TYPE EMPLEADO_T AS OBJECT (
   NOMBRE VARCHAR2(30),
   JEFE REF EMPLEADO_T);
   /

CREATE TABLE EMPLEADO OF EMPLEADO_T;
INSERT INTO EMPLEADO VALUES (EMPLEADO_T ('Gil', NULL));
INSERT INTO EMPLEADO
    SELECT EMPLEADO_T ('Alonso', REF(E)) FROM EMPLEADO E WHERE
    E.NOMBRE='Gil';
```

Referencias (II)

- Para acceder al objeto referido por un REF se utiliza el operador DEREF.
- En el siguiente ejemplo se visualiza el nombre del empleado y los datos de su jefe:

```
SELECT NOMBRE,  DEREF(JEFE) FROM EMPLEADO;
```

- Si queremos el nombre del empleado y el nombre de su jefe:

```
SELECT NOMBRE,  DEREF(JEFE).NOMBRE FROM EMPLEADO;
```

- En la siguiente consulta se obtiene el identificador del objeto cuyo nombre es 'Gil':

```
SELECT REF(P) FROM EMPLEADO P WHERE NOMBRE='Gil';
```

35

Herencia de tipos (I)

- Un subtipo obtiene el comportamiento (métodos) y los atributos de su supertipo.
- Los subtipos definen sus propios atributos y métodos y pueden redefinir los que heredan.

```
CREATE OR REPLACE TYPE TIPO_PERSONA AS OBJECT(
        DNI VARCHAR2(10),
        NOMBRE VARCHAR2(25),
        FEC_NAC DATE,
        MEMBER FUNCTION EDAD RETURN NUMBER,
        FINAL MEMBER FUNCTION GET_DNI RETURN VARCHAR2,
        MEMBER FUNCTION GET_NOMBRE RETURN VARCHAR2,
        MEMBER PROCEDURE VER_DATOS
) NOT FINAL;
/
```

36

Herencia de tipos (II)

```
CREATE OR REPLACE TYPE BODY TIPO_PERSONA AS
  MEMBER FUNCTION EDAD RETURN NUMBER IS
    ED NUMBER;
  BEGIN
    ED:= TO_CHAR(SYSDATE,'YYYY') – TO_CHAR(FEC_NAC,'YYYY');
    RETURN ED;
  END;
  FINAL MEMBER FUNCTION GET_DNI RETURN VARCHAR2 IS
  BEGIN
    RETURN DNI;
  END;
  MEMBER FUNCTION GET_NOMBRE RETURN VARCHAR2 IS
  BEGIN
    RETURN NOMBRE;
  END;
  MEMBER PROCEDURE  VER_DATOS ) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(DNI || '*' || NOMBRE || '*' || EDAD());
  END;
END;
/
```

37

Herencia de tipos (III)

```
CREATE OR REPLACE  TYPE  TIPO_ALUMNO UNDER TIPO_PERSONA (
        CURSO VARCHAR2(),
        NOTA_FINAL NUMBER,
        MEMBER FUNCTION NOTA RETURN NUMBER,
        OVERRIDING MEMBER PROCEDURE VER_DATOS);
/
CREATE OR REPLACE  TYPE  BODY TIPO_ALUMNO AS
        MEMBER FUNCTION NOTA RETURN NUMBER IS
        BEGIN
                RETURN NOTA_FINAL;
        END;
        OVERRIDING MEMBER PROCEDURE VER_DATOS IS
        BEGIN
                DBMS_OUTPUT.PUT_LINE(CURSO || '*' || NOTA_FINAL);
        END;
END;
/
```

38

Herencia de tipos (IV)

- Con la cláusula NOT FINAL, se indica que se pueden derivar subtipos de ese tipo. De no especificarse, se considera que es FINAL.
- Si un método es FINAL, los subtipos no pueden redefinirlo.
- La cláusula OVERRIDING, se utiliza para redefinir el método.
- Ejemplo de utilización de los tipos anteriores:

Herencia de tipos (V)

```
DECLARE
        A1 TIPO_ALUMNO:= TIPO_ALUMNO(NULL, NULL, NULL, NULL, NULL);
        A2 TIPO_ALUMNO:= TIPO_ALUMNO('14666588M', 'Pedro', '23/02/1976', 'Segundo', 3);
        NOM A1.NOMBRE%TYPE;
        DNI A1.DNI%TYPE;
        NOTAF A1.NOTA_FINAL%TYPE;
BEGIN
        A1.NOTA_FINAL:=7;
        A1.CURSO:= 'Primero';
        A1.NOMBRE:= 'Juan';
        A1.FEC_NAC:='20/01/1999';
        A1.VER_DATOS;
        NOM:= A2.GET_NOMBRE();
        DNI:= A2.GET_DNI();
        NOTAF:= A2.NOTA();
        A2.VER_DATOS;
        DBMS_OUTPUT.PUT_LINE(A1.EDAD());
        DBMS_OUTPUT.PUT_LINE(A2.EDAD());
END;
```

Herencia de tipos (VI)

```
CREATE TABLE TALUMNOS OF TIPO_ALUMNO (DNI PRIMARY KEY);

INSERT INTO TALUMNOS VALUES ('85666999B', 'Ana', 10/10/1987', 'Segundo', 7);
INSERT INTO TALUMNOS VALUES ('8444449C', 'Andrés', 10/11/1992', 'Tercero', 8);

SELECT * FROM TALUMNOS;
SELECT DNI, NOMBRE, CURSO, NOTA_FINAL FROM TALUMNOS;
SELECT P.GET_DNI(), P.GET_NOMBRE(), P.EDAD(), P.NOTA() FROM TALUMNOS P;
```

41