Ud 7 : Bases de Datos Objeto-Relacionales

IES Maestre de Cva.
Isabel Guerrero

Introducción

- Las **Bases de Datos Objeto-Relacionales (BDOR)** son una extensión de las bases de datos relacionales tradicionales a las que se les ha añadido conceptos del modelo orientado a objetos.
- Un **Sistema de Gestión de Base de Datos Objeto-Relacional (SGBDOR)** contiene características del modelo relacional y del orientado a objetos; es decir, es un sistema relacional que permite almacenar objetos en las tablas.

Pag. :2

Características

- Las características más importantes de los SGBDOR son las siguientes:
  - Soporte de **tipos de datos básicos y complejos**. El usuario puede crear sus propios tipos de datos.
  - Soporte para crear **métodos** para esos tipos de datos. Se pueden crear funciones miembro usando tipos de datos definidos por el usuario.
  - Gestión de tipos de datos complejos con un esfuerzo mínimo.
  - **Herencia**.
  - Se pueden almacenar **múltiples valores** en una columna de una misma fila.
  - **Relaciones (tablas) anidadas**.
  - Compatibilidad con las bases de datos relacionales tradicionales. Es decir, se pueden pasar las aplicaciones sobre bases de datos relacionales al nuevo modelo sin tener que rescribirlas
  - El inconveniente de las BDOR es que aumenta la complejidad del sistema, esto ocasiona un aumento del coste asociado.

Pag. :3

Objetos: Sintaxis para crear tipos

- Sintaxis de la orden **CREATE TYPE:**

```
CREATE or REPLACE TYPE nombreObjeto AS OBJECT (
  atributo TIPO,
  atributo TIPO,
  ...,
  MEMBER FUNCTION nombreFuncion RETURN Tipo,
  MEMBER FUNCTION nombreFuncion2(Var TIPO,…)
   RETURN Tipo,
  MEMBER PROCEDURE nombreProcedimiento,
  MEMBER PROCEDURE nombreFuncion2(Var TIPO,…)
);
```

Pag. :4

Objetos: Ejemplos

- Ejemplos de **CREATE TYPE:**

```
CREATE OR REPLACE TYPE DIRECCION AS OBJECT(
  CALLE        VARCHAR2(25),
  CIUDAD       VARCHAR2(20),
  CODIGO_POST  NUMBER(5)
);
/
CREATE OR REPLACE TYPE PERSONA AS OBJECT(
   CODIGO      NUMBER,
   NOMBRE      VARCHAR2(35),
   DIREC       DIRECCION,
   FECHA_NAC   DATE
);
```

Pag. :5

Objetos

- Una vez creados podemos usarlos para declarar e inicializar objetos como si se tratase de cualquier otro tipo predefinido, hemos de tener en cuenta que al declarar el objeto dentro de un bloque PL/SQL hay que inicializarlo.

Pag. :6

Inicialización de objetos

- **Crear o instanciar un objeto:** se realizará una **llamada a su método constructor**. Se puede hacer empleando la instrucción **NEW** seguido del nombre del tipo de objeto, como una llamada a una función en la que se indican como parámetros los valores que se desean asignar a los atributos inicialmente. En una asignación también puedes optar por hacer eso mismo omitiendo la palabra **NEW**.

```
variable_objeto := NEW Nombre_Tipo_Objeto
   (valor_atributo1, valor_atributo2, ...);
variable_objeto := Nombre_Tipo_Objeto
   (valor_atributo1, valor_atributo2, ...);
```

Pag. :7

Inicialización de objetos

- El siguiente ejemplo muestra la declaración y uso de los objetos creados anteriormente:

```
DECLARE
   DIR              DIRECCION:= DIRECCION(NULL,NULL,NULL);
   P                PERSONA:= NEW PERSONA(NULL,NULL,NULL,NULL);
   VDIRECCION1      DIRECCION:=NEW DIRECCION
                    ('VALENZUELA,1','CIUDAD REAL',13004);
   VDIRECCION2      DIRECCION;
   VDIR3            DIRECCION;

BEGIN
   DIR.CALLE :='LA MINA, 3';
   DIR.CIUDAD := 'GUADALAJARA';
   DIR.CODIGO_POST := 19001;
   P.CODIGO := 1;
   P.NOMBRE := 'JUAN';
   P.DIREC : = DIR;
   P.FECHA_NAC := '10/11/1988';
   VDIRECCION2:=DIRECCION('POSTAS,12',’ALMAGRO',13003);
   VDIR3.CALLE:=‘ALARCOS,21’; -- Daría error porque VDIR3 no está inicializado
END;
/
```

Pag. :8

Objetos: Eliminar tipo

- Para borrar un tipo usamos la orden **DROP TYPE** indicando a la derecha el nombre de tipo a borrar:

```
DROP TYPE nombre_tipo;
```

Pag. :9

Objetos: Sintaxis para crear cuerpo

- Una vez creado el tipo con la especificación de los métodos crearemos el cuerpo del nuevo tipo OBJECT mediante la instrucción **CREATE OR REPLACE TYPE BODY**:

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

Pag. :10

Objetos: Crear cuerpo

- Donde **<implementación de los métodos>** tiene el siguiente formato:

```
[STATIC|MEMBER|PROCEDURE NombreProc [(param1,parám2,...)]
IS
  Declaraciones;
BEGIN
  Instrucciones;
END;

[STATIC|MEMBER|CONSTRUCTOR] FUNCTION NombreFunc
    [(parametro1, parámetro2,...)) RETURN tipo_valor_retorno
IS
  Declaraciones;
BEGIN
  Instrucciones;
END;
```

Pag. :11

Objetos: Métodos

- Normalmente cuando creamos un objeto también creamos los métodos que definen el comportamiento del mismo y que permiten actuar sobre él.
- Los métodos son procedimientos y funciones que se especifican después de los atributos del objeto.

Pag. :12

Métodos: Member,Static, Constructor

- Pueden ser de varios tipos:
  - **MEMBER**: son los métodos que sirven para actuar con los objetos. Pueden ser procedimientos y funciones.
  - **STATIC**: son métodos estáticos independientes de las instancias del objeto. Pueden ser procedimientos y funciones.
  - **CONSTRUCTOR**: sirve para inicializar el objeto. Se trata de una función cuyos argumentos son los valores de los atributos del objeto y que devuelve el objeto inicializado.

Pag. :13

Métodos: Member

- Ejemplo de la declaración para el objeto dirección, lo hemos modificado incluyendo dos nuevos métodos:

```
CREATE OR REPLACE TYPE DIRECCION AS OBJECT
(
  CALLE             VARCHAR2(25),
  CIUDAD            VARCHAR2(20),
  CODIGO_POST       NUMBER(5),
  MEMBER FUNCTION GetCalle RETURN VARCHAR2,
  MEMBER FUNCTION GetDireccionCompleta
              RETURN VARCHAR2
);
```

Pag. :14

Métodos: Parámetro SELF

- Es un parámetro especial que puedes utilizar con los métodos MEMBER es el que recibe el nombre **SELF**. Este parámetro hace referencia a una instancia (objeto) del mismo tipo de objeto. Aunque no lo declares explícitamente, este parámetro siempre se declara automáticamente.
- Si se hace referencia al parámetro **SELF** dentro del cuerpo de un método, realmente se está haciendo referencia al objeto que ha invocado a dicho método. Por tanto, si utilizas **SELF.nombre_atributo** o **SELF.nombre_método**, estarás utilizando un atributo o un método del mismo objeto que ha llamado al método donde se encuentra utilizado el parámetro **SELF**.

Pag. :15

Método Constructor

- Cada tipo de objeto tiene un método constructor, es una **función con el mismo nombre que el tipo de objeto** y que se encarga de **inicializar los atributos y retornar una nueva instancia** de ese tipo de objeto.
- Oracle crea un **método constructor por defecto** para cada tipo de objeto declarado, cuyos parámetros formales coinciden en orden, nombres y tipos de datos con los atributos del tipo de objeto.
- Si deseas reemplazar el método constructor por defecto, debes utilizar la sentencia **CONSTRUCTOR FUNCTION** seguida del nombre del tipo de objeto en el que se encuentra (recuerda que los métodos constructores tienen el mismo nombre que el tipo de objeto). A continuación debes indicar los parámetros que sean necesarios de la manera habitual. Por último, debes indicar que el valor de retorno de la función es el propio objeto utilizando la cláusula **RETURN SELF AS RESULT**.

Pag. :16

Método Constructor

- Ejemplo: la declaración y el cuerpo de un método constructor para el tipo de objeto Usuario.

```
CREATE OR REPLACE TYPE USUARIO AS OBJECT (
   LOGIN       VARCHAR2(10),
   NOMBRE      VARCHAR2(30),
   F_INGRESO   DATE,
   CREDITO     NUMBER,
   CONSTRUCTOR FUNCTION USUARIO(LOGIN VARCHAR2, CREDITO NUMBER)
       RETURN SELF AS RESULT
);
CREATE OR REPLACE TYPE BODY USUARIO AS
    CONSTRUCTOR FUNCTION USUARIO(LOGIN VARCHAR2, CREDITO NUMBER)
         RETURN SELF AS RESULT
    IS
    BEGIN
         IF (CREDITO >= 0) THEN
            SELF.CREDITO := CREDITO;
         ELSE
            SELF.CREDITO := 0;
         END IF;
         RETURN;
    END;
END;
```

Pag. :17

Objetos: Ejemplo crear objeto DIRECCION

- Ejemplo:

```
CREATE OR REPLACE TYPE DIRECCION AS OBJECT
(
  CALLE         VARCHAR2(25),
  CIUDAD        VARCHAR2(20),
  CODIGO_POST   NUMBER(5),
  MEMBER FUNCTION GetCalle RETURN VARCHAR2,
  MEMBER FUNCTION GetDireccionCompleta
     RETURN VARCHAR2,
  MEMBER PROCEDURE SetCalle(CALLE VARCHAR2)
);
/
```

Pag. :18

Métodos: Ejemplo crear cuerpo del objeto DIRECCION

- La implementación del cuerpo del objeto DIRECCION es la siguiente:

```
CREATE OR REPLACE TYPE BODY DIRECCION AS
    MEMBER FUNCTION GetCalle RETURN VARCHAR2 IS
    BEGIN
      RETURN SELF.CALLE;
    END;
    MEMBER FUNCTION GetDireccionCompleta RETURN VARCHAR2 IS
    BEGIN
      RETURN SELF.CALLE||'---'||SELF.POBLACION;
    END;
    MEMBER PROCEDURE SetCalle(C VARCHAR2) IS
    BEGIN
      SELF.CALLE:=C;
    END;
END;
/
```

Pag. :19

Objetos: uso de los métodos

- El siguiente bloque PL/SQL muestra el uso del objeto DIRECCION, visualizará el nombre de la calle, al no definir constructor es necesario inicializar el objeto:

```
DECLARE
  DIR DIRECCION :=new
    DIRECCION('Postas,1','Ciudad Real',13001);
BEGIN
   DBMS_OUTPUT.PUT_LINE(DIR.GETCALLE());
   DBMS_OUTPUT.PUT_LINE(DIR.DIRECCIONCOMPLETA());
   DIR.SETCALLE('ALARCOS,21');
END;
/
```

Pag. :20

Objetos: Ejemplo crear tipo RECTANGULO

- El siguiente ejemplo define un tipo rectángulo con 3 atributos y un constructor que recibe 2 parámetros:

```
CREATE OR REPLACE TYPE RECTANGULO AS OBJECT
(
  BASE     NUMBER,
  ALTURA   NUMBER,
  AREA     NUMBER,
  CONSTRUCTOR FUNCTION
    RECTANGULO (BASE NUMBER,ALTURA NUMBER)
    RETURN SELF AS RESULT
);
/
```

Pag. :21

Objetos: Ejemplo Constructor RECTANGULO

- La implementación del método constructor del objeto RECTANGULO es la siguiente:

```
CREATE OR REPLACE TYPE BODY RECTANGULO AS
  CONSTRUCTOR FUNCTION RECTANGULO
  (BASE NUMBER, ALTURA NUMBER) RETURN SELF AS RESULT
  AS
  BEGIN
   SELF.BASE := BASE;
   SELF.ALTURA := ALTURA;
   SELF.AREA := BASE * ALTURA;
   RETURN;
  END;
END;
/
```

Pag. :22

Objetos: Uso del método constructor RECTANGULO

- El siguiente bloque PL/SQL muestra el uso del objeto RECTANGULO, se puede llamar al constructor usando los 3 atributos; pero es más robusto llamarlo usando 2 atributos de esta manera nos aseguramos que el atributo AREA tiene el valor inicial correcto. En este caso no es necesario inicializar los objetos R1 y R2 ya que se inicializan al llamar al constructor con NEW:

```
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;
END;
/
```

Pag. :23

Métodos: Sobrecarga

- Al igual que ocurre con los subprogramas empaquetados, los métodos pueden ser **sobrecargados**, es decir, puedes utilizar el mismo nombre para métodos diferentes siempre que sus parámetros formales sean diferentes (en cantidad o tipo de dato).
- Son ejemplos correctos:

```
MEMBER PROCEDURE setNombre(Nombre VARCHAR2)
MEMBER PROCEDURE setNombre(Nombre VARCHAR2, Apellidos VARCHAR2)
```

Pag. :24

Métodos: DROP TYPE BODY

- Para borrar el cuerpo de un tipo usamos la orden **DROP TYPE BODY** indicando a la derecha el nombre del tipo cuyo cuerpo deseamos borrar:

```
DROP TYPE BODY nombre_tipo;
```

Pag. :25

Herencia de tipos

- La herencia facilita la creación de objetos a partir de otros ya existentes e implica que un subtipo obtenga todo el comportamiento (métodos) y eventualmente los atributos de su supertipo.
- Los subtipos definen sus propios atributos y métodos y puede redefinir los métodos que heredan, esto se conoce como polimorfismo.

Pag. :26

Herencia de tipos

- Para indicar que un tipo de objeto es **heredado** de otro hay que usar la **palabra reservada UNDER** y además hay que tener en cuenta que el tipo de objeto del que **hereda** debe tener la **propiedad NOT FINAL**.
- Por defecto, los tipos de objeto se declaran como **FINAL**, es decir, que no se puede crear un tipo de objeto que herede de él.
- Igualmente si un **método** es **FINAL** los subtipos no pueden redefinirlo.
- La cláusula **OVERRIDING** se utiliza para redefinir el método.

Pag. :27

Herencia: Ejemplo

```
CREATE TYPE DatosGenarales
AS OBJECT(
       nombre VARCHAR2(20),
       apellidos VARCHAR2(30)
) NOT FINAL;
/

CREATE TYPE UsuarioPersona
UNDER DatosGenerales(
       login VARCHAR(30),
       f_ingreso DATE,
       credito NUMBER
);
/
```

```
DECLARE
  u1 UsuarioPersona;
BEGIN
  u1 := NEW UsuarioPersona
     ('nombre1', 'apellidos1',
     'user1', '01/01/2001',
      100);

dbms_output.put_line(u1.nombre);
END;
/
```

Pag. :28

Herencia de tipos: Ejemplos

- El siguiente ejemplo define un TIPO_PERSONA y a continuación el subtipo TIPO_ALUMNO:

```
-- Se define el tipo persona
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;
/
```

Pag. :29

Herencia de tipos: Ejemplos

- Cuerpo del tipo persona:

```
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;
```

Pag. :30

Herencia de tipos: Ejemplos

- Definimos el TIPO_ALUMNO:

```
CREATE OR REPLACE TYPE TIPO_ALUMNO UNDER TIPO_PERSONA(
  CURSO          VARCHAR2(10),
  NOTA_FINAL     NUMBER,
  MEMBER FUNCTION NOTA RETURN NUMBER,
  OVERRIDING MEMBER PROCEDURE VER_DATOS
);
/
```

Pag. :31

Herencia de tipos: Ejemplos

- El cuerpo de TIPO_ALUMNO:

```
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;
/
```

Pag. :32

Herencia de tipos: Ejemplos

- El siguiente bloque PL/SQL muestra un ejemplo de uso de los tipos definidos, al definir el objeto se inicializan todos los atributos ya que no se ha definido constructor para inicializar el objeto:

```
DECLARE
   A1 TIPO_ALUMNO := TIPO_ALUMNO(NULL,NULL,NULL,NULL,NULL1¡
   A2 TIPO_ALUMNO := TIPO_ALUMNO('871234533A','PEDRO','12/12/1996','SEGUNDO',7);
   NOM A1.NOMBRE%TYPE;
   DNI A1.DNI%TYPE;
   NOTAF A1.NOTA_FINAL%TYPE;
BEGIN
   A1.NOTA_FINAL:=8;
   A1.CURSO:='PRIMERO';
   A1.NOMBRE:='JUAN';
   A1.FEC_NAC:='20/10/1997';
   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;
```

Pag. :33

Herencia de tipos: Ejemplo

- A continuación se crea una tabla de TIPO_ALUMNO con el DNI como clave primaria, se insertan filas y se realiza alguna consulta:

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

INSERT INTO TALUMNOS VALUES
  ('871234533A‘,'PEDRO’,'12/12/1996‘, ‘SEGUNDO', 7);
INSERT INTO TALUMNOS VALUES ('809004534B‘,
  'MANUEL‘,'12/12/1997' ,'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;
```

Pag. :34

Métodos: MAP y ORDER

- En muchas ocasiones necesitamos comparar e incluso ordenar datos de tipos objetos. Para ello es necesario crear un método **MAP** u **ORDER**, debiéndose definir al menos uno de ellos por cada objeto que se quiere comparar:
- Los métodos **MAP** consisten en una función que devuelve un valor de tipo escalar (CHAR, VARCHAR2, NUMBER, DATE, ...) que será el que se **utilice en las comparaciones y ordenaciones** aplicando los criterios establecidos para este tipo de datos.
- **Sólo puede haber un método MAP u ORDER.**
- Un método **ORDER** utiliza 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. Este método devuelve un valor negativo si el parámetro de entrada es mayor que el atributo, un valor positivo si ocurre lo contrario y cero si ambos son iguales. Suelen. ser menos funcionales y eficientes, se utilizan cuando el criterio de comparación, es muy complejo como para Implementarlo con un método MAP.

Pag. :35

Método MAP: Ejemplo

- Ejemplo de creación del objeto con método **MAP**:

```
CREATE OR REPLACE TYPE USUARIO AS OBJECT (
   LOGIN VARCHAR2(30),
   NOMBRE VARCHAR2(30),
   APELLIDOS VARCHAR2(40),
   F_INGRESO DATE,
   CREDITO NUMBER,
   MAP MEMBER FUNCTION ORDENARUSUARIO RETURN VARCHAR2
);
```

Pag. :36

Método MAP: Ejemplo

- En el cuerpo del método se debe retornar el valor que se utilizará para realizar las comparaciones entre las instancias del tipo de objeto. Por ejemplo, si se quiere establecer que las comparaciones entre objetos del tipo Usuario se realice considerando el orden alfabético habitual de apellidos y nombre:

```
CREATE OR REPLACE TYPE BODY USUARIO AS
  MAP MEMBER FUNCTION ORDENARUSUARIO RETURN VARCHAR2 IS
  BEGIN
        RETURN (APELLIDOS || ' ' || NOMBRE);
  END ORDENARUSUARIO;
END;
```

Pag. :37

Método MAP: Ejemplo

- Ejemplo de uso de la función MAP:

```
DECLARE
  U1 USUARIO:=NEW
      USUARIO('USU1','USUARIO 1','APELLIDOS','01/01/2015',20);
  U2 USUARIO:=NEW
      USUARIO('USU2','USUARIO 2','APELLIDOS','01/12/2015',30);

BEGIN
  IF U1<U2 THEN
    DBMS_OUTPUT.PUT_LINE('U1 ES MENOR QUE U2');
  ELSIF U1=U2 THEN
      DBMS_OUTPUT.PUT_LINE('U1 ES IGUAL QUE U2');
  ELSE
    DBMS_OUTPUT.PUT_LINE('U1 ES MAYOR QUE U2');
  END IF;
END;
```

Pag. :38

Métodos MAP : Ejemplo

- La siguiente declaración indica que los objetos de tipo PERSONA se van a comparar por su atributo CODIGO:

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

Pag. :39

Métodos MAP: Ejemplo

```
CREATE OR REPLACE TYPE BODY PERSONA AS
  MAP MEMBER FUNCTION POR_CODIGO RETURN NUMBER
  IS
  BEGIN
   RETURN CODIGO;
  END;
END;
/
```

Pag. :40

Métodos MAP: Ejemplo

- El siguiente código PL/SQL compara dos objetos de tipo PERSONA, y visualiza 'OBJETOS IGUALES' ya que el atributo CODIGO tiene el mismo valor para los dos objetos:

```
DECLARE
   P1 PERSONA:= PERSONA(NULL,NULL,NULL,NULL);
   P2 PERSONA:= PERSONA (NULL,NULL,NULL,NULL) ;
BEGIN
   P1. CODIGO : = 1;
   P1.NOMBRE := 'JUAN';
   P2.CODIGO :=1;
   P2.NOMBRE :='MANUEL';
   IF P1=P2 THEN
    DBMS_OUTPUT.PUT_LINE( 'OBJETOS IGUALES');
   ELSE
    DBMS_OUTPUT.PUT_LINE('OBJETOS DISTINTOS');
   END IF;
END;
/
```

Pag. :41

Métodos: MAP y ORDER

- Es necesario un método MAP u ORDER para comparar objetos en PL/SQL.
- **Un tipo de objeto solo puede tener un método MAP o uno ORDER.**
- El lenguaje PL/SQL utiliza los métodos **MAP** para evaluar expresiones lógicas que resultan valores booleanos como **objeto1 > objeto2**, y para realizar las comparaciones implícitas en las cláusulas **DISTINCT**, **GROUP BY** y **ORDER BY**.

Pag. :42

Tablas de objetos

- Una vez definidos los objetos se pueden utilizar para definir nuevos tipos o para definir tablas de ese tipo de objetos.
- Una tabla de objetos es una tabla en la que cada fila se almacene un objeto de ese tipo.
- Se accede a los atributos de esos objetos como si se tratasen de columnas de la tabla.

Pag. :43

Tablas de objetos

- Ejemplo: crea la tabla ALUMNOS de tipo PERSONA con la columna CODIGO como clave primaria:

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

- Insertar fila en ALUMNOS:

```
INSERT INTO ALUMNOS(CODIGO, NOMBRE,DIREC,FECHA_NAC)
VALUES (1,'MARIA',DIRECCION('C/TOLEDO,10','CIUDAD REAL',
13003),'10/01/1980');
```

Pag. :44

Tablas de objetos: Insertar datos

- Insertar fila en ALUMNOS:

```
DECLARE
     DIR    DIRECCION;
     PER    PERSONA;
BEGIN
     INSERT INTO ALUMNOSS
        VALUES (1,'MARIA',
            DIRECCION('C/TOLEDO,10','CIUDAD REAL',13003),
            '10/01/1980');
     DIR:=NEW DIRECCION('C/POSTAS,10','TOMELLOSO',21000);
     INSERT INTO ALUMNOSS
            VALUES(2,'JOSE',DIR,'01/01/1980');
     DIR:= DIRECCION('C/ALARCOS,12','CIUDAD REAL',13001);
     PER:=NEW PERSONA(3,'PEPE',DIR,'12/02/1981');
     INSERT INTO ALUMNOSS VALUES(PER);
END;
```

Pag. :45

Tablas de objetos: Modificar datos

- Actualiza filas en ALUMNOS:

```
DECLARE
   DIR DIRECCION;
   PER PERSONA:= NEW PERSONA(3,'CARLOS',
          DIRECCION('C/TOLEDO,21','CIUDAD REAL',
          13004),'02/02/1990');
BEGIN
   UPDATE ALUMNOS A
       SET A.NOMBRE='JOSEFA'
       WHERE A.CODIGO=1;
   UPDATE ALUMNOS A
       SET A= PER
       WHERE CODIGO=3;
   UPDATE ALUMNOS A
       SET A= PERSONA(2,'MIGUEL',
          DIRECCION('C/TOLEDO,44','CIUDAD REAL',
          13004),'22/11/1993')
       WHERE CODIGO=2;

END;
```

Pag. :46

Tablas de objetos

- Ejemplo de consultas a la tabla : seleccionar aquellas filas de la localidad de GUADALAJARA:

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

- Seleccionar columnas individuales, para seleccionar columnas individuales de tipo OBJECT se necesita definir un **alias** de la tabla:

```
SELECT A.CODIGO,A.NOMBRE,A.DIREC.CALLE
  FROM ALUMNOS A;
```

Pag. :47

Tablas de objetos

- Para llamar a los métodos:

```
SELECT A.NOMBRE, A.DIREC.GETCALLE()
  FROM ALUMNOS A;
```

- Modificar aquellas filas de GUADALAJARA, convertimos la ciudad en minúsculas:

```
UPDATE ALUMNOS A
    SET A.DIREC.CIUDAD=LOWER(A.DIREC.CIUDAD)
    WHERE A.DIREC.CIUDAD='GUADALAJARA';
```

- Eliminamos aquellas filas cuya ciudad sea TOLEDO:

```
DELETE ALUMNOS A
    WHERE A.DIREC.CIUDAD='TOLEDO';
```

Pag. :48

Tablas de objetos

- Bloque PL/SQL que muestra el nombre y la calle de los alumnos:

```
BEGIN
    FOR I IN (SELECT * FROM ALUMNOS) LOOP
      DBMS_OUTPUT.PUT_LINE('NOMBRE:'||I.NOMBRE
       || 'CALLE:'||I.DIREC.GETCALLE()
       ||'CIUDAD: '||I.DIREC.CIUDAD);
    END LOOP;
END;
```

Pag. :49

Tablas de objetos: VALUE

- Cuando se quiera **hacer referencia a un objeto en lugar de alguno de sus atributos**, se puede utilizar la función **VALUE** junto con el nombre de la tabla de objetos o su alias, **dentro de una sentencia SELECT**.
- Ejemplo de uso de dicha función para hacer inserciones en otra tabla (Favoritos) del mismo tipo de objetos:

```
INSERT INTO Favoritos
     SELECT VALUE(u) FROM UsuariosObj u
      WHERE u.credito >= 100;
```

Pag. :50

Tablas de objetos: VALUE

- Esa misma función **VALUE** puedes utilizarla para hacer comparaciones de igualdad entre objetos, por ejemplo, si deseamos obtener datos de los usuarios que se encuentren en las tablas Favoritos y UsuariosObj.:

```
SELECT u.login
FROM UsuariosObj u JOIN Favoritos f
     ON VALUE(u)=VALUE(f);
```

Pag. :51

Tablas de objetos: VALUE

- En el siguiente ejemplo, se hace la comparación con una columna de tipo de objetos. En ese caso la referencia que se hace a la columna (g.unUsuario) permite obtener directamente un objeto, sin necesidad de utilizar la función **VALUE**.

```
SELECT g.dni
FROM Gente g JOIN Favoritos f
     ON g.unUsuario=VALUE(f);
```

Pag. :52

Tablas de objetos: VALUE

- Usando la **cláusula INTO se puede guardar en variables el objeto obtenido** en las consultas usando la función **VALUE**.

```
DECLARE
  u1 Usuario;
  u2 Usuario;
BEGIN
  SELECT VALUE(u) INTO u1
    FROM UsuariosObj u
    WHERE u.login = 'luitom64';
  dbms_output.put_line(u1.nombre);
  u2 := u1;
  dbms_output.put_line(u2.nombre);
END;
```

Pag. :53

Tipos colección

- Los tipos de colecciones en Oracle son:
  - **VARRAYS:** es una colección de elementos a la que se le establece una dimensión máxima que debe indicarse al declararla. Al tener una longitud fija, la eliminación de elementos no ahorra espacio en la memoria del ordenador.
  - **TABLAS ANIDADAS:** puede almacenar cualquier número de elementos. Tienen, por tanto, un tamaño dinámico, y no tienen que existir forzosamente valores para todas las posiciones de la colección.

Pag. :54

Tipos colección: VARRAYS con Objetos

- Tenemos declarado:

```
CREATE TYPE TELEFONO AS
    VARRAY(3) OF VARCHAR2(9);
CREATE OR REPLACE TYPE DIR AS OBJECT
 ( CALLE VARCHAR2(30),
   POBLACION VARCHAR2(30),
   TELEF TELEFONOS
 );
 CREATE OR REPLACE TYPE LISTADIRECCIONES
 AS VARRAY(5) OF DIR;
```

Pag. :55

Tipos colección: VARRAYS con Objetos

- Declaramos la siguiente tabla:

```
CREATE TABLE ALUMNOS(
   NOMBRE VARCHAR2(40),
   DIR    LISTADIRECCIONES);
```

Pag. :56

Tipos colección: VARRAYS con Objetos

```
DECLARE
    L LISTADIRECCIONES:=LISTADIRECCIONES();
    VNOMBRE VARCHAR2(50);
BEGIN
   L.EXTEND;
   L(1):=DIR('VALENZUELA,1','MIGUELTURRA',TELEFONOS(1,2,3));
   L.EXTEND;
   L(2):=DIR('POSTAS,1','CIUDAD REAL',TELEFONOS(4,5));
   INSERT INTO ALUMNOS VALUES('MARIO',L);
   SELECT NOMBRE,DIR INTO VNOMBRE,L FROM ALUMNOS WHERE NOMBRE='MARIO';
   FOR i IN 1..L.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE ('CALLE Nº '||I||' : '||L(i).CALLE);
        FOR J IN 1..L(I).TELEF.COUNT LOOP
          DBMS_OUTPUT.PUT_LINE ('TELEFONO Nº '||J||':'||L(i).TELEF(J));
        END LOOP;
   END LOOP;
END;
```

Pag. :57

Referencias: REF

- Mediante el operador **REF** asociado a un atributo se pueden definir referencias a otros objetos.
- Un atributo de este tipo almacena una referencia al objeto del tipo definido e implementa una relación de asociación entre los dos tipos de objetos.
- Una columna de tipo **REF** guarda un puntero a una fila de la otra tabla, contiene el OID (identificador del objeto fila) de dicha fila.

Pag. :58

Referencias: REF

- El siguiente ejemplo crea un tipo EMPLEADO_T donde uno de los atributos es una referencia a un objeto EMPLEADO_T, después se crea una tabla de objetos EMPLEADO_T:

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

Pag. :59

Referencias: REF

- Insertamos filas en la tabla, el segundo INSERT asigna al atributo JEFE la referencia al objeto con apellido GIL:

```
INSERT INTO EMPLEADO VALUES (EMPLEADO_T
  ('GIL', NULL));
INSERT INTO EMPLEADO
  SELECT EMPLEADO_T('ARROYO', REF(E))
  FROM EMPLEADO E
  WHERE E.NOMBRE = 'GIL';
```

Pag. :60

Referencias: DEREF

- Para acceder al objeto referido por un **REF** se utiliza el operador **DEREF**, en el ejemplo se visualiza el nombre del empleado y los datos del jefe de cada empleado:

```
DECLARE
  EMP        EMPLEADO_T;
  VNOMBRE    EMPLEADO.NOMBRE%TYPE;
BEGIN
  FOR C IN (SELECT NOMBRE, DEREF(E.JEFE) D FROM EMPLEADO E) LOOP
     DBMS_OUTPUT.PUT_LINE('Empleado : '||C.NOMBRE);
     IF C.D IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('NO TIENE JEFE');
     ELSE
        DBMS_OUTPUT.PUT_LINE('SU JEFE ES :  '||C.D.NOMBRE);
     END IF;
  END LOOP;
END;
```

- La siguiente consulta obtiene el identificador del objeto cuyo nombre es GIL:

```
SELECT REF(P) INTO EMP FROM EMPLEADO P WHERE NOMBRE='GIL';
```

Pag. :61

Referencias: DEREF

- Diferentes ejemplos de uso de REF y DEREF:

```
DECLARE
  REMP     REF EMPLEADO_T;
  EMP      EMPLEADO_T;
  VNOMBRE  EMPLEADO.NOMBRE%TYPE;
  VJEFE    EMPLEADO.NOMBRE%TYPE;
BEGIN
  SELECT REF(E) INTO REMP FROM EMPLEADO E WHERE E.NOMBRE='GIL';
  SELECT VALUE(E) INTO EMP FROM EMPLEADO E WHERE E.JEFE=REMP;
  DBMS_OUTPUT.PUT_LINE('JEFE DE GIL: '||EMP.NOMBRE);

  SELECT NOMBRE,DEREF(JEFE).NOMBRE INTO VNOMBRE,VJEFE
     FROM EMPLEADO  WHERE NOMBRE='ARROYO';
  DBMS_OUTPUT.PUT_LINE('EMPLEADO: '||VNOMBRE||'--JEFE: '||VJEFE);
END;
```

Pag. :62

Referencias :REF

- Ejemplo:

```
CREATE OR REPLACE TYPE Partida AS OBJECT (
     codigo INTEGER,
     nombre VARCHAR2(20),
     usuarioCreador REF Usuario
);
/

DECLARE
     u_ref REF Usuario;
     p1 Partida;
BEGIN
     SELECT REF(u) INTO u_ref
     FROM UsuariosObj u
     WHERE u.login = 'luitom64';
     p1 := NEW Partida(1, 'partida1', u_ref);
END;
/
```

Pag. :63