Tipos de Objeto en PL/SQL

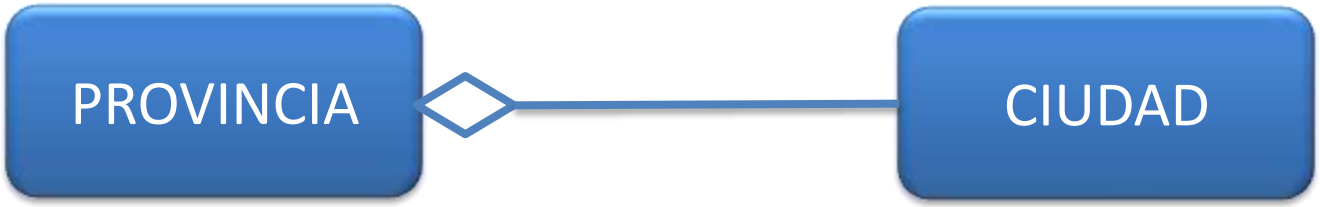
Índice

1. Introducción
2. Estructura de un Objeto
    1. Métodos
3. Gestión de Objetos
    1. Notación Punto
    2. OIDs
4. Tipos Referencia
    1. Operador VALUE
5. Tipos Colección

Introducción

- Una BDOR soporta las dos tecnologías: relacional y objeto-relacional
- Aportaciones OR
  - UDTs
    - Atributos / Métodos …
  - Tipos Complejos en una columna
  - Referencias
- Oracle incorpora estas mejoras desde la versión 8*i*
- Las estructuras de almacenamiento siguen siendo tablas, pero se pueden usar mecanismos de orientación al objeto para la gestión de datos
- Cada objeto tiene un tipo, se almacena en una fila de una tabla y tiene un identificador que permite referenciarlo desde otros objetos (otras filas de otras tablas)

- Los objetos permiten modelar mejor las relaciones “parte-todo”



Modelo Orientado a Objetos

Modelo Relacional

PK

PROVINCIA	ID
MADRID	MAD
BARCELONA	BCN
VALENCIA	VLC

PROVINCIAS

FK

ID_PROV	CIUDAD
MAD	MÓSTOLES
BCN	ALCORCÓN
VLC	GANDÍA

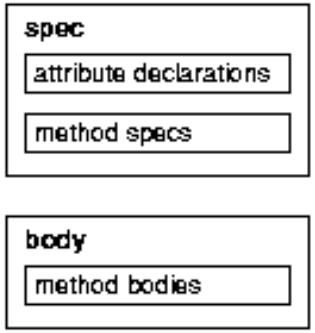
PROVINCIAS_CIUDADES

FK

CIUDAD
MÓSTOLES
ALCORCÓN
GANDÍA

CIUDADES

Estructura de un Tipo de Objeto




```
CREATE [OR REPLACE] TYPE Complejo AS OBJECT (
parte_r   REAL, -- attributo
parte_e   REAL,
MEMBER FUNCTION suma (x Complejo) RETURN Complejo, -- metodo
MEMBER FUNCTION resta (x Complejo) RETURN Complejo,
```

```
CREATE [OR REPLACE] TYPE BODY Complejo AS
   MEMBER FUNCTION suma (x Complejo) RETURN Complejo IS
   BEGIN
      RETURN Complejo(parte_r + x.parte_r, parte_e + x.parte_e);
   END plus;

   MEMBER FUNCTION resta (x Complejo) RETURN Complejo IS
   BEGIN
      RETURN Complejo(parte_r - x.parte_r, parte_e - x.parte_e);
   END less;
END;
```

- Los parámetros no deben restringirse en tamaño
- Se invocan igual que se recupera un campo
  - Especificando los parámetros si fuera el caso
- Definir un tipo no implica reservar espacio para objetos del tipo

- Las tablas tipadas son:
  - Tablas que almacenan objetos del tipo sobre el que han sido definidas
    - Cada fila almacena un objeto
  - También podemos verlas como
    - Una tabla con una única columna del tipo objeto
    - Una tabla con tantas columnas como el tipo objeto

```
CREATE OR REPLACE TYPE Tipo_Coche AS OBJECT (
      Marca          VARCHAR2(25),
      Modelo         VARCHAR2(25),
      Matricula      VARCHAR2(9))
/

CREATE TABLE COCHES OF Tipo_Coche;
```

Tablas de Objetos

- Almacenan un objeto en cada fila y permiten acceder a los campos del objeto cómo si fueran columnas de la tabla
- Las restricciones se definen sobre la tabla y se aplicarán SÓLO sobre aquellos objetos almacenados en dicha tabla

```
CREATE OR REPLACE TYPE Tipo_Coche AS OBJECT (
      Marca            VARCHAR2(25),
      Modelo           VARCHAR2(25),
      Matricula        VARCHAR2(9))
/

CREATE TABLE COCHES OF Tipo_Coche;

ALTER TABLE COCHES ADD
                  CONSTRAINT PK_COCHE PRIMARY KEY (Marca, Modelo);
```

Índice

1. Introducción
2. Estructura de un Objeto
    1. Métodos
3. Gestión de Objetos
    1. Notación Punto
    2. OIDs
4. Tipos Referencia
    1. Operador VALUE
5. Tipos Colección

Métodos

- Implementan el comportamiento de los objetos del tipo
  - 3 tipos → MEMBER, STATIC, CONSTRUCTOR

```
CREATE TYPE Tipo_Cubo AS OBJECT (
                largo           INTEGER,
                ancho           INTEGER,
                alto            INTEGER,
                MEMBER FUNCTION superficie RETURN INTEGER,
                MEMBER FUNCTION volumen RETURN INTEGER,
                MEMBER PROCEDURE mostrar ());
/
CREATE TYPE BODY Tipo_Cubo AS
                MEMBER FUNCTION volume RETURN INTEGER IS
                BEGIN
                                RETURN largo * ancho * alto;
                                -- RETURN SELF.largo * SELF.ancho * SELF.alto;
                END;
                MEMBER FUNCTION superficie RETURN INTEGER IS
                BEGIN
                                RETURN 2 * (largo * ancho + largo * alto + ancho * alto);
                END;
                MEMBER PROCEDURE mostrar ()
                IS
                BEGIN
                                DBMS_OUTPUT.PUT_LINE('Largo: ' || largo || ' - ' || 'Ancho: ' || ancho || ' - ' || 'Alto: ' || alto);
                                DBMS_OUTPUT.PUT_LINE('Volumen: ' || volumen || ' - ' || 'Superficie: ' || superficie);
                END;
END;
/
```

La variable SELF permite referirse al objeto sobre el que se invocó la función/procedimiento

Métodos MEMBER

```
CREATE TYPE Tipo_Cubo AS OBJECT (…);
/

CREATE TYPE BODY Tipo_Cubo AS …
END;
/

CREATE TABLE Cubos of Tipo_Cubo;
INSERT INTO Cubos VALUES(Tipo_Cubo (10, 10, 10));
INSERT INTO Cubos VALUES(Tipo_Cubo (3, 4, 5));
SELECT * FROM Cubos;
SELECT c.volumen(), c.superficie () FROM cubos c
WHERE c.largo = 10;

DECLARE
        mi_cubo      Tipo_Cubo;
BEGIN

  SELECT VALUE(c) INTO mi_cubo FROM Cubos c
  WHERE c.largo = 10;
  mi_cubo.mostrar ();
END;
/
DROP TABLE Cubos;
DROP TYPE tipo_Cubo FORCE;
```

Métodos

- Métodos STATIC: operaciones globales, que no son de los objetos, si no del tipo

```
CREATE TYPE Tipo_Cubo AS OBJECT(
 ….,
STATIC PROCEDURE nuevoCubo (
  v_largo      INTEGER,
  v_ancho      INTEGER,
  v_alto       INTEGER));
/
CREATE TYPE BODY Tipo_Cubo AS
STATIC PROCEDURE nuevoCubo (
                v_largo         INTEGER,
                v_ancho         INTEGER,
                v_alto          INTEGER)
IS
   sqlstmt VARCHAR2(100);
BEGIN
   sqlstmt := 'INSERT INTO Cubos '||schname||'.'||tabname|| '
              VALUES (Tipo_Cubo(largo, ancho, alto))';
   EXECUTE IMMEDIATE sqlstmt;
END;
/

BEGIN
    Tipo_Cubo.nuevoCubo(1, 1, 1);
END;
/
```

- Cada vez que se crea un tipo de objeto, Oracle crea automáticamente un método constructor
  - Identificado por el mismo nombre del tipo
  - Es una función que devuelve una nueva instancia del tipo definido por el usuario y establece los valores de sus atributos.
  - Recibe como parámetros los **atributos** del tipo
  - Debe ser **siempre** explícitamente invocado cada vez que se desee crear un objeto del tipo

Métodos Constructor:

```
CREATE OR REPLACE TYPE Tipo_Coche AS OBJECT (
      Marca          VARCHAR2(25),
      Modelo         VARCHAR2(25),
      Matricula      VARCHAR2(9))
/

CREATE OR REPLACE TYPE Tipo_Persona AS OBJECT (
      Nombre         VARCHAR2(25),
      Coche          Tipo_Coche)
/

CREATE TABLE PERSONAS OF Tipo_Persona;

INSERT INTO PERSONAS VALUES ('Ramón Ramirez',
                             Tipo_Coche('CITROEN', '2-CV', 'M-9999999'));
```

Llamada al constructor del tipo para crear el objeto embebido

Índice

1. Introducción
2. Estructura de un Objeto
    1. Métodos
3. Gestión de Objetos
    1. Notación Punto
    2. OIDs
4. Tipos Referencia
    1. Operador VALUE
5. Tipos Colección

- Podemos usar un tipo de objeto igual que cualquier otro tipo de dato

```
DECLARE
   r Tipo_Persona; -- r toma valor NULL
BEGIN
   IF r IS NULL THEN ... – esta comparación devuelve TRUE
   IF r > (2/3) … -- pero esta comparación TAMBIÉN devuelve NULL
   r := Tipo_Persona('Ramón Ramírez', NULL);
```

- Una buena práctica es inicializar los objetos al declararlos

Acceso a los atributos de un objeto

- Para acceder a las propiedades de un objeto se utiliza la notación punto (.)

```
CREATE TYPE Tipo_Coche AS OBJECT (
      Marca          VARCHAR2(25),
      Modelo         VARCHAR2(25),
      Matricula      VARCHAR2(9));
/

CREATE TYPE Tipo_Persona AS OBJECT (
      Nombre         VARCHAR2(25),
      Coche          Tipo_Coche);
/
```

```
DECLARE
  v_persona Tipo_Persona;
BEGIN
  v_persona.coche.marca := 'CITROEN';
  -- error ACCESS_INTO_NULL
  v_persona := Tipo_Persona(NULL,…);
  v_persona.coche.marca := 'CITROEN';
 …
END;
```

Variables de Correlación

- Son un tipo especial de variable PL/SQL
- En general, las *correlation variables* se identifican con los alias de tabla
  - Son opcionales, podemos omitirlas

```
SELECT E.Nombre FROM Empleado E;
SELECT Empleado.Nombre FROM Empleado;
```

- Pero cuando trabajamos con UDTs es OBLIGATORIO utilizarlas para acceder a los campos del UDT

```
CREATE TYPE Tipo_Coche AS OBJECT (
    Marca           VARCHAR2(25),
    Modelo          VARCHAR2(25))
/
CREATE TABLE Personas (
    Nombre          VARCHAR2(25),
    Coche           Tipo_Coche);
```

```
SELECT coche FROM Personas;
SELECT Personas.coche FROM Personas;
SELECT BD_00.Personas.coche FROM Personas;
```

```
COCHE(MARCA, MODELO)
--------------------------------------------------------
TIPO_COCHE('Pontiac', 'Firebird')
```

Si tratamos de recuperar el campo de tipo UDT, los datos se devuelven acompañados del constructor del tipo.

Variables de Correlación (II)

- Igualmente, para acceder a los campos del UDT por separado hay que utilizar las *correlation variables*

```
SQL> SELECT Coche.Marca
FROM Personas;
SELECT Coche.Marca FROM Personas
       *
ERROR en línea 1:
ORA-00904: "COCHE"."MARCA":
identificador no válido
```

```
SQL> SELECT Personas.Coche.Marca
FROM Personas;
SELECT Personas.Coche.Marca FROM
Personas
       *
ERROR en línea 1:
ORA-00904: “Personas”."COCHE"."MARCA":
identificador no válido
```

```
SQL> SELECT P.Coche.Marca FROM Personas P;

COCHE.MARCA
------------------------
Pontiac
```

Variables de Correlación (III)

- Las *correlation variables* permiten resolver ambigüedades

```
SQL> SELECT Cliente.Coche.Marca;
```

  - Cliente es un nombre de usuario, Coche es una tabla y Marca una columna
  - Cliente es un nombre de tabla, Coche de columna y Marca un campo de esa columna

- El uso de la variable resuelve el problema

```
SQL> SELECT C.Coche.Marca;
```

OID – Object IDentifier

- Cada fila de una tabla tipada (OR) tendrá un identificador del objeto fila → OID
- Para guardar esos identificadores Oracle utiliza un tipo REF

```
SQL> SELECT REF(P) FROM PERSONA P WHERE P.APELLIDO = 'SÁNCHEZ'

REF(P)
--------------------------------------------------------------------------------
0000280209726911892BAD4CB7BE7824E07E2B2C7ECA4E2D0291C2415CA1DD7B
B75494F1D601C003850000
```

La función REF devuelve el OID del objeto seleccionado

Índice

1. Introducción
2. Estructura de un Objeto
    1. Métodos
3. Gestión de Objetos
    1. Notación Punto
    2. OIDs
4. Tipos Referencia
    1. Operador VALUE
5. Tipos Colección

Tipos Referencia

- Cada fila (objeto fila) podrá ser referenciada como un objeto a través de su OID

```
CREATE OR REPLACE TYPE Tipo_Persona AS OBJECT (
      Nombre     VARCHAR2(25),
      Coche      Tipo_Coche)
/

CREATE OR REPLACE TYPE Tipo_Empresa AS OBJECT (
      Nombre     VARCHAR2(25),
      NIF        VARCHAR2(25),
      Director   REF Tipo_Persona)
/
CREATE TABLE PERSONAS OF Tipo_Persona;
CREATE TABLE EMPRESAS OF Tipo_Empresa;

SQL> DESC EMPRESAS
 Nombre                                    ¿Nulo?   Tipo
 ----------------------------------------- -------- ------------------------------
 NOMBRE                                             VARCHAR2(25)
 NIF                                                VARCHAR2(25)
 DIRECTOR                                           TIPO_PERSONA()


SQL> SELECT * FROM EMPRESAS;
Nombre                    NIF          DIRECTOR
------------------------- ------------ ----------------
'Juan Nadie'              '000001'     00002802097269118928AD4CB7BE7824E07E ....
```

Tipos Referencia

- Para obtener una referencia a un objeto utilizamos el operador REF

```
DECLARE
   persona_ref REF Persona;
BEGIN
   SELECT REF(p) INTO persona_ref FROM Personas p
   WHERE p.nombre = 'Pepe Pérez';
END;
/
```

- Una columna de tipo REF guarda un puntero a una fila de la otra tabla
  - Contiene el OID de dicha fila

```
INSERT INTO EMPRESAS VALUES ('ACME', '666',
            (SELECT REF(P) FROM PERSONAS P WHERE P.NOMBRE LIKE '%Ramirez'));

SELECT * FROM EMPRESAS;
NOMBRE    NIF  DIRECTOR
----------  ----  ----------------------------------------------------------------------------------------------------
ACME      666  0000220208EB2D8E93BE39430EB8D325E255E81F5E2F8521EDFD3F4
```

Tipos Referencia

- Las referencias no se pueden navegar en PL/SQL

```
DECLARE
  p_ref REF Persona;
  telefono VARCHAR2(15);
BEGIN
  telefono := p_ref.telefono; -- no permitido
……
```

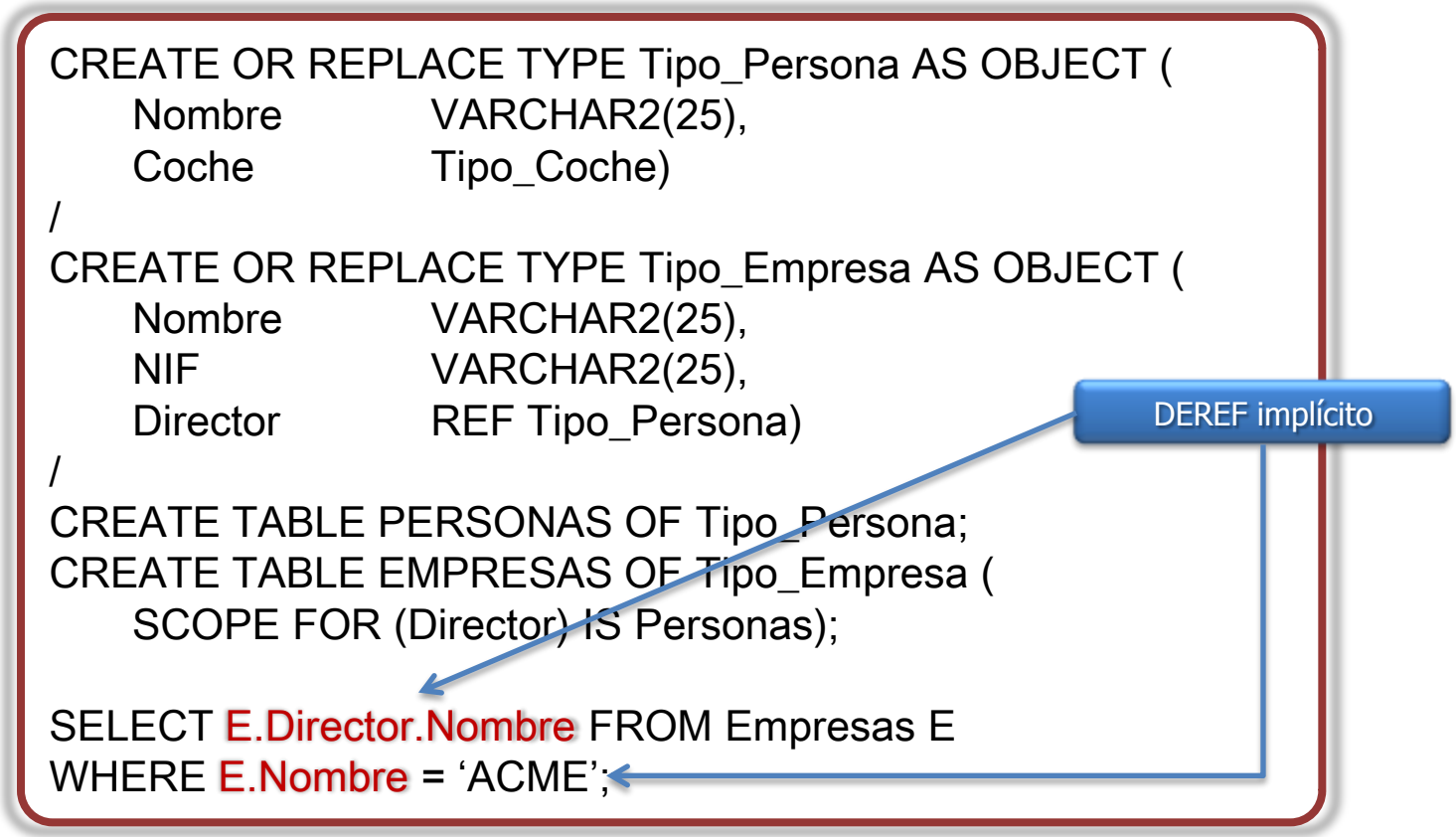
- En su lugar hay que usar la función DEREF (o el paquete UTL_REF), que permite obtener el objeto referenciado

```
DECLARE
  p1 Persona;
  p_ref REF Persona;
  nombre VARCHAR2(15);
BEGIN ...
  -- Si p_ref tiene una referencia válida a una fila de una tabla
  SELECT DEREF(p_ref) INTO p1 FROM dual;
  nombre :=    p1.nombre;
  …..
```

```
SQL> SELECT DEREF(E.Director) FROM Empresas E;

DEREF(E.DIRECTOR)(NOMBRE, COCHE(MARCA, MODELO, MATRICULA))
--------------------------------------------------------------------------------------------------------------
TIPO_PERSONA('Ramón Ramirez', TIPO_COCHE('CITROEN', '2-CV', 'M-9999999'))
```

- En cambio si se pueden navegar en SQL

```
CREATE OR REPLACE TYPE Tipo_Persona AS OBJECT (
      Nombre          VARCHAR2(25),
      Coche           Tipo_Coche)
/
CREATE OR REPLACE TYPE Tipo_Empresa AS OBJECT (
      Nombre          VARCHAR2(25),
      NIF             VARCHAR2(25),
      Director        REF Tipo_Persona)
/
CREATE TABLE PERSONAS OF Tipo_Persona;
CREATE TABLE EMPRESAS OF Tipo_Empresa (
      SCOPE FOR (Director) IS Personas);

SELECT E.Director.Nombre FROM Empresas E
WHERE E.Nombre = 'ACME';
```



Tipos Referencia

- Podemos restringir el conjunto de objetos a los que apuntará la REF a los contenidos en una única tabla

```
CREATE OR REPLACE TYPE Tipo_Persona AS OBJECT (
      Nombre          VARCHAR2(25),
      Coche           Tipo_Coche)
/
CREATE OR REPLACE TYPE Tipo_Empresa AS OBJECT (
      Nombre          VARCHAR2(25),
      NIF             VARCHAR2(25),
      Director        REF Tipo_Persona)
/
CREATE TABLE PERSONAS OF Tipo_Persona;
CREATE TABLE EMPRESAS OF Tipo_Empresa (
      SCOPE FOR (Director) IS Personas);
```

- De esta forma el almacenamiento ocupará menos espacio y el acceso será más eficiente

Operador VALUE

- Para obtener el objeto almacenado en una fila (y no sólo el valor de los campos de dicho objeto) se necesita la función VALUE

```
SELECT * FROM Personas;

NOMBRE
------------------
COCHE(MARCA, MODELO, MATRICULA)
--------------------------------------------------------------------------------
Ramón Ramirez
TIPO_COCHE('CITROEN', '2-CV', 'M-9999999')
```

La consulta sólo devuelve el valor de los campos del objeto de la clase Tipo_Persona

```
SQL> SELECT VALUE(P) FROM PERSONAS P;

VALUE(P)(NOMBRE, COCHE(MARCA, MODELO, MATRICULA))
------------------------------------------------------------------
TIPO_PERSONA('Ramón Ramirez', TIPO_COCHE('CITROEN', '2-CV', 'M-9999999'))
```

La consulta devuelve el objeto de la clase Tipo_Persona

- Al recuperar el objeto completo, de pueden realizar operaciones con él: modificarlo, insertarlo …

```
SQL> SET SERVEROUTPUT ON
DECLARE
  v_persona     Tipo_Persona;
BEGIN
  SELECT VALUE(P) INTO v_persona FROM PERSONAS P WHERE NOMBRE LIKE '%Ram%';
  DBMS_OUTPUT.PUT_LINE(v_persona.nombre);
  DBMS_OUTPUT.PUT_LINE(v_persona.coche.marca);
  DBMS_OUTPUT.PUT_LINE(v_persona.coche.modelo);
END;
/
Ramón Ramirez
CITROEN
2-CV

DECLARE
       v_persona     Tipo_Persona;
BEGIN
     v_persona := Tipo_Persona('Mamen Tido', Tipo_Coche('SEAT', '600', 'B-8888888'));
     INSERT INTO Personas VALUES (v_persona);
END;
/
```

Como hemos creado el objeto antes, no necesitamos invocar el constructor del tipo

Forward Type *Definitions*

- A la hora de crear un tipo sólo podemos referirnos a otros tipos que ya estén creados
  - Esto representa un problema en caso de referencias circulares
- Para resolverlo se utilizan las *Forward Type Definitions*
  - Declarar A, crear B y crear A
  - Recompilar A

```
CREATE TYPE Empleado AS OBJECT (
  nombre VARCHAR2(20),
  dept REF Departmento,
  ... );

CREATE TYPE Departamento AS OBJECT (
  number INTEGER,
  jefe REF Empleado,
  ... );
```

```
CREATE TYPE Empleado; -- tipo incompleto

CREATE TYPE Departamento AS OBJECT (
  number INTEGER,
  jefe REF Empleado,
  ... );

CREATE TYPE Empleado AS OBJECT (
  nombre VARCHAR2(20),
  dept REF Departmento,
  ... );
```

Índice

1. Introducción
2. Estructura de un Objeto
    1. Métodos
3. Gestión de Objetos
    1. Notación Punto
    2. OIDs
4. Tipos Referencia
    1. Operador VALUE
5. Tipos Colección

Tipos Colección

- Oracle soporta dos tipos
  - VARRAYs: colección ordenada de elementos de tamaño fijo
  - NESTED TABLES: colección no ordenada y de tamaño variable
- Como el tipo objeto, incorpora constructores por defecto que hay que utilizar para crear objetos colección
- Sus parámetros serán los elementos de la colección

Creación de Tipos Colección

```
CREATE OR REPLACE TYPE Tipo_Persona AS OBJECT (
      Nombre                VARCHAR2(25),
      DNI                   VARCHAR2(10))
/

CREATE OR REPLACE TYPE Tipo_Personas AS
              VARRAY(15) OF Tipo_Persona
/

CREATE OR REPLACE TYPE Tipo_Empresa AS OBJECT (
      Nombre                VARCHAR2(25),
      NIF                   VARCHAR2(25),
      Empleados             Tipo_Personas)
/
CREATE TABLE EMPRESAS OF Tipo_Empresa ();
```

```
CREATE OR REPLACE TYPE Tipo_Personas AS
              TABLE OF Tipo_Persona
/
```

```
CREATE TABLE Empresas OF Tipo_Empresa
  NESTED TABLE Empleados STORE AS Lista_Emp;
/
```

Constructor del tipo colección

```
INSERT INTO EMPRESA VALUES ('ACME', '00000000X',
              Tipo_Personas(Tipo_Persona('Pepe Pèrez', '999999999M')));
```

Constructor del tipo objecto

Tipos Colección: consulta

Consulta de tipos colección

- La consulta *estándar* recupera los datos anidados

```
SELECT E.Empleados FROM  Empresa(E) WHERE E.Nombre = 'ACME'

Empleados(Nombre, DNI)
-----------------------------------------------------------------------------
Tipo_Personas(Tipo_Persona('Pepe Pèrez', '999999999M')
```

Devuelve los datos anidados

- Mientras que la expresión TABLE permite descomponerlos

```
SELECT Emp.* FROM Empresa E, TABLE(E.Empleados) Emp
WHERE E.Nombre = 'ACME'

Nombre DNI
---------  ------------------------
'Pepe Pèrez', '999999999M'
```

Esta sería la forma *intutiitva* de realizar la consulta, pero al ejecutarla obtendríamos un error

```
SELECT E.Emp.* FROM Empresa E
WHERE E.Nombre = 'ACME'
```

Tipos Colección: operaciones DML

- Operaciones que actúan sobre toda la colección o sobre elementos aislados
  - Las segundas utilizan el operador TABLE
  - No se soportan actualizaciones individuales en VARRAYs

Tipos Colección: operaciones DML

Ejemplos

```
-- INSERCIÓN
INSERT INTO TABLE(SELECT E.Empleados FROM  Empresa(E)
                  WHERE E.Nombre = 'ACME')
VALUES ('Mamen Tido', '12345678A')

--ACTUALIZACIÓN
UPDATE TABLE (SELECT E.Empleados FROM  Empresa(E)
                  WHERE E.Nombre = 'ACME')  E
SET VALUE(E) = ('Mamen Tido Mucho', '12345678A')
WHERE E.NOMBRE = 'Mamen Tido';

-- BORRADO
DELETE FROM TABLE(SELECT E.Empleados FROM  Empresa(E)
                  WHERE E.Nombre = 'ACME')  E
WHERE E.NOMBRE = 'Mamen Tido';
```

Tipos Colección: operadores

- Obtener información sobre la colección
  - COUNT devuelve el número de filas.
  - EXISTS devuelve TRUE si la fila existe.
  - FIRST/LAST devuelve el índice de la primera y última fila.
  - NEXT/PRIOR devuelve la fila anterior o posterior a la actual.
  - LIMIT informa del número máximo de elementos que puede contener .
- Modificar los elementos de la colección
  - DELETE borra uno o más elementos usando su índice.
  - EXTEND añade nuevas filas.
  - TRIM elimina filas.

Tipos Colección: PL/SQL

- Ejemplo

```
CREATE TYPE Tipo_Nombres_Dep IS VARRAY(7) OF VARCHAR2(30);
/
CREATE TABLE Departamentos (
                region                  VARCHAR2(25),
                nombres_dep             Tipo_Nombres_Dep);

BEGIN
  INSERT INTO Departamentos VALUES ('Europe', Tipo_Nombres_Dep('Shipping','Sales','Finance'));
  INSERT INTO Departamentos VALUES('Americas', Tipo_Nombres_Dep('Sales','Finance','Shipping'));
  INSERT INTO Departamentos VALUES('Asia', Tipo_Nombres_Dep('Finance','Payroll','Shipping','Sales'));
COMMIT;
END;
/

DECLARE
  v_nombres   Tipo_Nombres_Dep := Tipo_Nombres_Dep('Benefits', 'Advertising', '
              Contracting', 'Executive', 'Marketing');
  v_nombres2 Tipo_Nombres_Dep;
BEGIN
  UPDATE Departamentos SET nombres_dep = v_nombres WHERE region = 'Europe';
  COMMIT;
  SELECT nombres_dep INTO v_nombres2 FROM Departamentos WHERE region = 'Europe';
  FOR i IN v_nombres2.FIRST .. V_nombres2.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE('Departamentos = ' || v_nombres2(i));
  END LOOP;
END;
/
```

Recorre la colección de nombres de departamento

Tipos Colección: cursores y colecciones

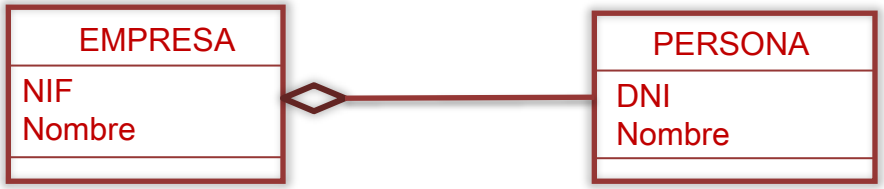
Ejemplo

```
CREATE TYPE Tipo_Nombres_Dep IS VARRAY(7) OF VARCHAR2(30);
/
CREATE TABLE Departamentos (
                region                  VARCHAR2(25),
                nombres_dep             Tipo_Nombres_Dep);

BEGIN
  INSERT INTO Departamentos VALUES ('Europe', Tipo_Nombres_Dep('Shipping','Sales','Finance'));
COMMIT;
END;
/

DECLARE
  CURSOR c_depts IS SELECT * FROM Departamentos;
  v_region      VARCHAR2(25);
  v_nombres   Tipo_Nombres_Dep;
BEGIN
  OPEN c_depts;
  LOOP
    FETCH c_depts INTO v_region, v_nombres;
    EXIT WHEN c_depts%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('REGION: ' || v_region);
    FOR i IN v_nombres2.FIRST .. V_nombres2.LAST
    LOOP
      DBMS_OUTPUT.PUT_LINE(' - Departamento = ' || '(' || i || ') → ' || v_nombres2(i));
    END LOOP;
  END LOOP;
END;
/
```

Carga una fila
(región + lista de nombres de departamento)
en las dos variables utilizadas

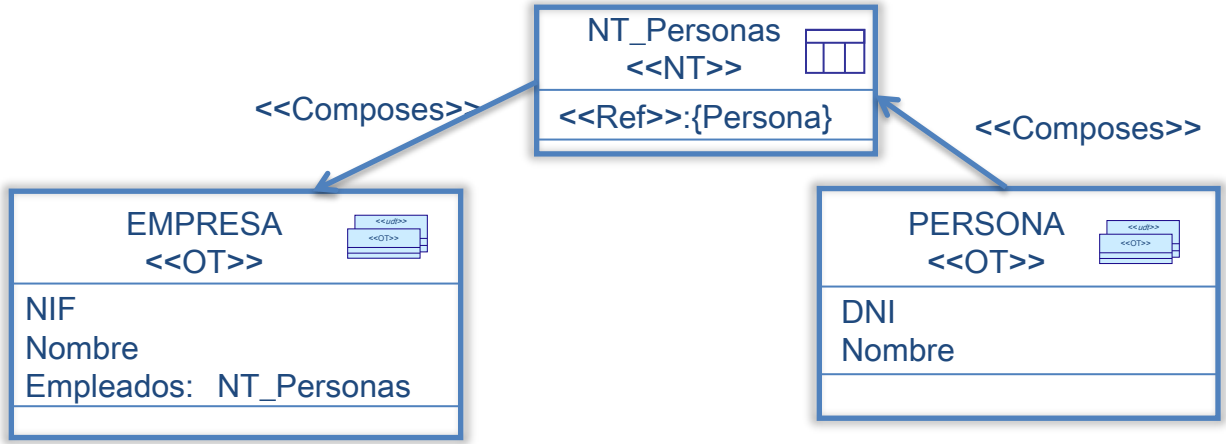
Tipos Colección: colecciones de tipos REF

- La consulta a colecciones de tipos REF crea algunos problemas



Modelo Conceptual

Modelo Objeto-Relacional (ORACLE)



Tipos Colección

```
CREATE TYPE Tipo_Persona AS OBJECT (
  DNI            VARCHAR2(30),
  Nombre         VARCHAR2(30))
/

CREATE TYPE Tipo_Personas AS TABLE OF REF Tipo_Persona
/

CREATE TYPE Tipo_Persona AS OBJECT (
  NIF            VARCHAR2(30),
  Nombre          VARCHAR2(30),
  Empleados      Tipo_Personas)
/

CREATE TABLE Personas OF Tipo_Persona;

CREATE TABLE Empresas OF Tipo_Empresa
  NESTED TABLE Empleados STORE AS Lista_Emp;
```

```
INSERT INTO Personas VALUES ('0000001', 'Mamen Tido');
INSERT INTO Personas VALUES ('0000002', 'Francisco Rupto');

DECLARE
  v_p1           REF Tipo_Persona;
  v_p2           REF Tipo_Persona;
BEGIN
  SELECT REF(P) INTO v_p1 FROM Personas P
    WHERE P.DNI = '0000001';
  SELECT REF(P) INTO v_p2 FROM Personas P
    WHERE P.DNI = '0000002';
  INSERT INTO Empresas E VALUES ('12345678A', 'ACME',
    Tipo_Personas(vp1, vp2));
END;
/
```

Una alternativa pasa por crear un tipo intermedio que proporcione el nombre columna ...

```
SELECT E.Nombre, EMP.Nombre FROM Empresas E,
TABLE(E.Empleados) EMP;
```

Esta consulta eleva un error, el problema es que el resultado es un OID sin más. Necesitamos un nombre de columna ...

```
SELECT E.Nombre, EMP.COLUMN_VALUE.Nombre
FROM Empresas E, TABLE(E.Empleados) EMP;
```

```
CREATE TYPE Tipo_REF_Persona AS OBJECT (
  Persona        REF Tipo_Persona)
/

CREATE TYPE Tipo_Personas AS TABLE OF Tipo_REF_Persona
/

SELECT E.Nombre, EMP.Persona.Nombre FROM Empresas E,
TABLE(E.Empleados) EMP;
```

Tipos de Objeto en PL/SQL

Guía Completa

Oracle® Database Object-Relational Developer's Guide

11g Release 1 (11.1)

URL: http://www.filibeto.org/sun/lib/nonsun/oracle/11.1.0.6.0/B28359_01/appdev.111/b28371/adobjint.htm

Resolución de Nombres: uso de alias

- SQL permite
- *Capture*
  - Cuando una declaración de tipo de un ámbito diferente impide que el compilador resuelva correctamente un nombre o referencia
  - Para evitar estos errores, Oracle define una serie de reglas, entre otras:
    - ✓ Especificar una alias para cualquier tabla involucrada en una sentencia DML
    - ✓ Preceder cualquier nombre de columna con el alias dado a la tabla correspondiente
    - Evitar el uso de alias que coincidan con el nombre del esquema

Resolución de Nombres: uso de alias

```
CREATE TYPE Tipo1 AS OBJECT (
            a NUMBER);
CREATE TABLE Tab1 (
            tab2 Tipo1);
CREATE TABLE Tab2 (
            x NUMBER);
SELECT * FROM Tab1 T1  -- alias con el mismo nombre que el esquema
  WHERE EXISTS (SELECT * FROM T1.Tab2 WHERE x = T1.tab2.a);
-- T1.tab2.a se resuelve apuntando al atributo a de la columna tab2 de la tabla Tab1
```

```
CREATE TYPE Tipo1 AS OBJECT (
            a NUMBER);
CREATE TABLE Tab1 (
            tab2 Tipo1);
CREATE TABLE Tab2 (
            x NUMBER,
            a NUMBER);
SELECT * FROM Tabla1 T1  -- alias con el mismo nombre que el esquema
  WHERE EXISTS (SELECT * FROM T1.Tabla2 WHERE x = T1.tab2.a);
-- Ahora T1.tab2.a se resuelve apuntando al atributo a de la la tabla Tab2
```

Resolución de Nombres: uso de alias

```
SELECT * FROM Tabla1 tb1
WHERE EXISTS (SELECT * FROM T1.Tabla2 tb2
                        WHERE tb2.x = tb1.tab2.a);
-- Evitamos ambigüedades siguiendo las reglas de nombrado
```

Bibliografía Complementaria

- http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96624/10_objs.htm
- http://www.stanford.edu/dept/itss/docs/oracle/10g/appdev.101/b10807/10_objs.htm
- http://www.filibeto.org/sun/lib/nonsun/oracle/11.1.0.6.0/B28359_01/appdev.111/b28371/adobjplsql.htm