

# BBDD Objeto-Relacionales

## Contenidos

- » Las bases de datos Objeto - Relacionales
- » Los atributos multivaluados
- » Las colecciones
- » Tablas anidadas
- » Tipos de objetos
- » Los identificadores de objeto
- » Métodos en BBDD-OR
- » Herencia en BBDD-OR
- » DML con BBDD-OR

## Objetivos

- » Identificar las características de las bases de datos objeto relacionales
- » Crear tipos de datos objeto, sus atributos y métodos
- » Crear tablas de objetos y tablas de columnas tipo objeto
- » Crear tipos de datos colección
- » Realizar consultas
- » Modificar la información almacenada manteniendo la integridad y consistencia de los datos

*En este capítulo se explica cómo se gestiona la información almacenada en bases de datos objeto-relacionales, evaluando y utilizando las posibilidades que proporciona el sistema gestor.*

## 8.1. Las Bases de datos objetos-relacionales

A medida que el mundo de la informática va generando modelos del mundo real para crear aplicaciones, van apareciendo nuevas necesidades para poder dar solución a estos problemas. A lo largo del libro se ha estudiado que un modelo de datos cuenta con las siguientes características:

- Entidades
- Atributos en cada una de estas entidades
- Relaciones entre los atributos y entre las entidades
- Directrices o reglas entre las entidades y los atributos

Las BBDD pueden modelar todas estas características. En un modelo más complejo incluso se puede incluir acciones, operaciones o procedimientos entre entidades y atributos. Se puede decir que las Bases de Datos Objeto-Relacional (BBDDOR) son una extensión del modelo relacional que además, rompe ciertas reglas, tanto en el modelo relacional como en el paradigma de la orientación a objetos, pero seguro que aporta otra solución al modelado de información.

A lo largo de este capítulo se podrá observar con ejemplos la diferencia de un modelo entidad relación E-R convencional y un modelo de datos objeto relacional O-R.

La principal diferencia radica en la existencia de tipos. Estos tipos se pueden construir mediante sentencias DDL y serán la base del desarrollo de BBDD O-R. Por tanto, para definir el modelo objeto-relacional habría que añadir a las características básicas de un modelo E/R: entidades, atributos, relaciones y los *tipos*.

## 8.2. Los atributos multivaluados

Los atributos multivaluados, según lo estudiado, rompen directamente la primera formal normal. Con este tipo de atributo básicamente se expresa todo lo contrario a la 1FN. Se dice, que un atributo puede tener más de un valor, pero son del mismo tipo. Existen autores, como Ted Codd, que hablan de la primera forma normal solo como atomicidad para evitar que el valor no se pueda dividir conceptualmente, por tanto se enfoca más no en que se pueda tener varios valores del mismo tipo, sino que ese valor en sí sea indivisible conceptualmente.

En todo caso, este tipo de atributo amplia enormemente el modelo de cara a modelar en mundo real. Con los gestores BBDDO-R se puede lograr obtener atributos con múltiples valores.

### 8.2.1. Las colecciones

Al realizar el paso al modelo relacional, los atributos se convierten en columnas. Podemos representar los atributos multivaluados en la BBDD a través de las colecciones. Una colección es un grupo de elementos del mismo tipo. En Oracle se usan los tipos para crear colecciones.

Supóngase que se desea guardar los nombres de los hijos de los empleados. Para esto se puede usar una colección. La relación puede tener las siguientes ocurrencias:

ID	Nombre	Apellidos	Hijos
1	Francisco	Pérez	(luis,ursula)
2	Esperanza	Jiménez	(jose, carlos, pedro)

Lo primero sería crear la *colección*:

```
SQL> create type colec_hijos as varray(10) of varchar2(30);
2 /
```

Tipo creado.

En esta instrucción se ha definido un nuevo tipo de datos llamado *colec\_hijos* que tendrán como máximo grupos de 10 valores y además, que serán de máximo 30 caracteres. A continuación, se crea la tabla:

```
SQL> create table empleado
2 (id      number,
3  nombre   varchar2(30),
4  apellido varchar2(30),
5  hijos    colec_hijos)
6 /
```

Tabla creada.

Con esto se expresa que la columna de hijos será una colección de tipo *colec\_hijos* ya definida. Es posible insertar datos tal y como se hace en cualquier otra tabla con el comando INSERT, lo único a tener en cuenta es que hay que decirle que el tipo de la siguiente forma:

```
insert into empleado
values(1,'Francisco','Pérez',colec_hijos('luis','ursula'));
```

```
insert into empleado values  
(2,'Esperanza','Jiménez',colec_hijos('jose','carlos','pedro'));
```

Obsérvese que es necesario especificar el tipo *colec\_hijos* para que el SGBD realice las comprobaciones necesarias.

Para consultar, también se puede utilizar SELECT:

```
SQL> select *  
  2  from empleado;  
  
    ID NOMBRE      APELLIDO      HIJOS  
-----  
     1 Francisco   Pérez        COLEC_HIJOS('luis', 'ursula')  
     2 Esperanza   Jiménez     COLEC_HIJOS('jose', 'carlos', 'pedro')
```

De esta manera se ha introducido una restricción al mundo real: que un empleado no puede tener más de 10 hijos.

- 
- ◊ **Actividad 8.1:** Define un tipo tabla llamado Asignaturas. Después, crea una tabla llamada *ciclosFormativos* que pueda tener varias *asignaturas*, de tal manera que *asignaturas* sea una tabla anidada.
- 

### 8.2.2. Tablas anidadas

50 o 60 años atrás, existía la posibilidad de que 10 hijos fuese un parámetro corto en algunas ocasiones y fuese posible ampliarlo hasta 20 o incluso 50, pero siempre tendría una restricción. Existe la posibilidad de no limitar la cantidad de valores dentro del atributo multivalorado, definiendo lo que se denomina un **tipo tabla**, en la que solo se especifica el tipo de datos que se desea para ese atributo. Utilizando el mismo ejemplo se redefine el tipo hijos de la siguiente forma:

```
SQL> create type tabla_hijos as table of varchar2(30);  
  2 /  
  
Tipo creado.
```

Y al igual que con el ejemplo anterior, se crea la tabla basándola en el tipo tabla\_hijos:

```
SQL> create table empleado
  2  (id          number,
  3   nombre      varchar2(30),
  4   apellido    varchar2(30),
  5   hijos       tabla_hijos)
  6   nested table hijos store as t_HIJOS
  7 /
```

Tabla creada.

Ahora, *tabla\_hijos*, ya no es un tipo colección , es un *tipo tabla*. Los tipo tabla utilizan para su almacenamiento una *tabla anidada* o *nested table*. Dicho de otro modo, la columna *hijos* es del tipo *tabla\_hijos* almacenada sobre un tipo de segmento especial llamado tabla anidada. A diferencia de la colección, se ha definido esta tabla especial que permitirá almacenar en el atributo multivaluado tantos valores como sea necesario.

**¿Sabías que . . . ?** En Oracle, se puede consultar la tabla dba\_objects para examinar los objetos que se han creado en el SGBD. También se puede consultar mediante dba\_segments la estructura de almacenamiento que utiliza Oracle para almacenar el objeto:

```
SQL> select object_name, object_type, status
  2  from dba_objects
  3* where object_name like '%HIJOS%'
```

OBJECT_NAME	OBJECT_TYPE	STATUS
TABLA_HIJOS	TYPE	VALID
T_HIJOS	TABLE	VALID

```
SQL> select segment_name, segment_type
  2  from dba_segments
  3* where segment_name like '%HIJOS%'
```

SEGMENT_NAME	SEGMENT_TYPE
T_HIJOS	NESTED TABLE

Para consultar, insertar, borrar o actualizar información en tablas anidadas, se realiza igual que con las colecciones.

Con estos tipos se abren nuevas posibilidades al construir modelos de datos: es posible definir tipos de datos personalizados y muy completos, con comportamientos y características autónomas.

El modelo pasa de tener dos dimensiones (filas y columnas), a tres dimensiones (filas de tipos, filas y columnas). La cantidad de colecciones tipo y tablas tipo, dependerá únicamente de la destreza que se posea y de la habilidad para aplicar estas ventajosas características a cada uno de los modelos que representan el mundo que se desea modelar.

### 8.2.3. Tipos de Objeto

Aunque no es objetivo del capítulo explicar la teoría de la programación orientada a objetos (POO), se recordará que una clase es como el molde o un *patrón* para la fabricación de los objetos. Y visto desde el otro sentido, un objeto es una *instancia* de una clase.

En Oracle, se pueden crear *clases* a través de los tipos, más concretamente de los *tipos de objetos*, usando la sentencia *create type as object*. Gracias a este tipo objeto se puede agregar información a la base de datos anidando la información en única fila en tantos niveles como se deseé.

Para ilustrar el funcionamiento de esta estructura recurrimos a la siguiente relación:

CLIENTES ( Id, Nombre, Apellido, Dirección, Población, Provincia, Teléfono, Móvil)  
Se puede afirmar que todos sus atributos son atómicos y que cumple, al menos, la primera forma normal. Una instancia de esa relación cliente (Modelo Relacional), podría ser:

ID	Nombre	Apellido	Dirección	Población	Provincia	Teléfono	Móvil
1	Francisco	Pérez	Mayor,20	Madrid	Madrid	912823722	600000001

Suponiendo que este Cliente no cuenta con un móvil, sino que cuenta con 3 móviles, el móvil de su trabajo, su móvil particular de una compañía que le da una buena tarifa por la mañana, y otro móvil de una compañía que le da buenos descuentos en SMS. Se podría pensar en conservar el mismo modelo de datos de forma elegante, y no con una entidad cliente que contenga hasta 10 atributos llamados

movil1, movil2,... movil10 (no es muy inusual encontrarse con BBDD que tienen estas “soluciones”).

Ajustándose únicamente al modelo E-R la solución “óptima” sería crear una Relación que sea cliente-móvil, y que contenga el ID del cliente y su número de móvil. Por ejemplo:

ID	Móvil
1	600000001
1	600000002
1	600000003

Las BBDD orientadas a objeto brindan la posibilidad de crear un tipo de objeto de la siguiente forma:

```
SQL> create or replace type telefono as object
  2 (tipo varchar2(30),
  3 numero number)
  4 /
```

Se puede observar que cada objeto *telefono* tendrá dos atributos, el tipo de teléfono móvil y el número. A continuación, se crea una *tabla tipo* llamada *listin* basada en el *objeto tipo* para añadir la funcionalidad de múltiples valores:

```
SQL> create or replace type listin as table of telefono
  2 /
Tipo creado.
```

Finalmente se crea la tabla Clientes:

```
SQL> create table clientes
  2 (id number,
  3 nombre varchar2(10),
  4 apellido varchar2(10),
  5 direccion varchar2(30),
  6 poblacion varchar2(30),
  7 provincia varchar2(30),
  8 telefonos listin)
  9 nested table telefonos store as tel_tab
 10 /
```

Tabla creada.

A partir de aquí, se pueden ejecutar sentencias DML en la tabla clientes haciendo valer la deseada ampliación con relación al modelo de datos tradicional:

```
SQL> insert into clientes
  2 values
  3 (1,'Paco','Pérez','mayor 20','madrid','madrid',
  4 listin(telefono('fijo',911234567),
  5         telefono('movil personal',600000001),
  6         telefono('movil empresa',600000002)))
  7 /

1 fila creada.
SQL> commit;

Confirmación terminada.
```

Si se consulta la tabla se obtiene:

```
SQL> select * from cliente;
ID NOMBRE APELLIDO DIRECCION POBLACION PROVINCIA TELEFONOS(TIPO, NUMERO)
-- -----
1 Paco Pérez mayor 20 madrid madrid LISTIN(TELEFONO('fijo', 911234567),
                                             TELEFONO('movil personal', 600000001),
                                             TELEFONO('movil empresa', 600000002))
```

Se puede observar cómo se anida la información dentro de la propia fila. De esta manera, 1 cliente puede tener múltiples teléfonos y cada teléfono, varios atributos. Esto es la representación real de las *relaciones anidadas*.

### 8.3. Los identificadores de objeto

Oracle trabaja con OID (Object Identifiers). Un OID es un identificador para mejorar la búsqueda de la información. Cuando se crea una columna cuyo tipo es un tipo de objeto, en lugar de almacenarse todos los datos del objeto en la propia columna de la tabla, el SGBD solo almacena un identificador, que será la dirección de una fila en una tabla auxiliar que el SGBD para almacenar los datos del objeto. Es decir, internamente, el SGBD solo almacena un número de 16-Bytes que le sirve como referencia para ir a buscar la información de la tabla objeto si es necesario. Es posible ver estos OIDs con una pseudo-columna llamada *object\_id*. Para ver el valor del objeto, se puede utilizar *object\_value*. Se ilustra en el siguiente ejemplo:

```

SQL> create type coche as object
  2  (marca varchar(20),
  3   modelo varchar(20));
  4 /


Tipo creado.

SQL> create table vehiculos of coche;

Tabla creada.

SQL> insert into vehiculos values (coche('seat','ibiza'));

1 fila creada.

SQL> select object_id, object_value from vehiculos;

OBJECT_ID          OBJECT_VALUE(MARCA, MODELO)
-----
A5440089F85FFA30E040007F01000FDE COCHE('seat', 'ibiza')

```

La ventaja de usar OIDs es que la información del objeto está almacenada en otro sitio distinto de la propia fila, por tanto, en caso de no usar este objeto, el SGBD no tendrá que acceder a toda esta información reduciendo el tiempo de respuesta.

La desventaja es que en el momento de realizar consultas sobre las tablas, el SGBD debe realizar accesos a bloques extras de información, consumiendo más CPU.

## 8.4. Los métodos

Al igual que en la POO cada clase define el comportamiento de sus objetos a través de métodos, en las BBDD-OR también es posible crear métodos para los tipo objeto.

Se pueden definir *funciones* o procedimientos miembros, cuyas acciones modelan el comportamiento de un tipo de objeto. Estas funciones o procedimientos que pertenecen a un tipo de objeto se denominan *métodos*.

Para ilustrar como usar funciones miembro en tipos de objetos, se expone el siguiente ejemplo: si se desea modelar un objeto triángulo para almacenar sus ca-

racterísticas (la base y la altura), y almacenar en la BBDD cientos de triángulos pudiendo calcular el área de cada triángulo, se podría crear el siguiente esquema:

```
SQL> create or replace type tipo_triangulo as object
  2   (base  number,
  3    altura number,
  4    member function area return number)
  5  /
```

Tipo creado

/

Se puede observar que dentro del tipo de objeto *tipo\_triangulo* se ha definido la cabecera de una función llamada área (*método* o *member function* *area*).

A continuación se creará la definición del método área a través del cuerpo del tipo *create or replace type body*:

```
SQL> create or replace type body tipo_triangulo as
  2   member function area return number is
  3     a number;
  4   begin
  5     a := (base*altura)/2;
  6     return a;
  7   end;
  8 end;
 9 /
```

Cuerpo del tipo creado.

Una vez creado el tipo triángulo, se puede crear una tabla para almacenar triángulos:

```
SQL> create table triangulos
  2   (id number,
  3    triangulo tipo_triangulo);
Tabla creada.

SQL> desc triangulos
Nombre          Nullable?  Tipo
-----          -----
ID              NUMBER
TRIANGULO      TIPO_TRIANGULO
```

Y después, se insertan datos dentro la tabla, tal y como se ha expuesto previamente:

```
SQL> insert into triangulos values (1,tipo_triangulo(5,5));
1 fila creada.

SQL> insert into triangulos values (2,tipo_triangulo(10,10));
1 fila creada.

SQL> commit;
Confirmación terminada.

SQL> select * from triangulos;
   ID TRIANGULO(BASE, ALTURA)
-----
 1 TIPO_TRIANGULO(5, 5)
 2 TIPO_TRIANGULO(10, 10)
```

Finalmente, se puede declarar un pequeño bloque anónimo en PL-SQL para recorrer la tabla e invocar al método *area*:

```
SQL> declare
 2   t tipo_triangulo;
 3 begin
 4   for i in (select * from triangulos) loop
 5     t:=i.triangulo;
 6     dbms_output.put_line('El triangulo con id:'||i.id);
 7     dbms_output.put_line('Con base: '||t.base);
 8     dbms_output.put_line('Y altura: '||t.altura);
 9     dbms_output.put_line('tiene un area de: '||t.area);
10   end loop;
11 end;
12 /
```

El triangulo con id:1  
Con base: 5  
Y altura: 5  
tiene un area de: 12,5  
El triangulo con id:2  
Con base: 10  
Y altura: 10  
tiene un area de: 50

Procedimiento PL/SQL terminado correctamente.

Por cada iteración del bucle, se hace una referencia a un objeto *t*. Obsérvese que *t.base* y *t.altura* son los atributos del objeto *t*, mientras que *t.area* es la invocación al método.

◊ **Actividad 8.2:** Al igual que en POO, en BBDD es posible crear constructores. Consulta en la documentación de Oracle cómo crear constructores.

## 8.5. La herencia

Una de las grandes ventajas de la POO es la herencia, gracias a la cual se pueden crear superclases abstractas para después, crear subclases más específicas que hereden los atributos y métodos de las superclases.

En BBDDO-R es posible hacer algo parecido con los tipos de objetos: se pueden crear subtipos de objetos a partir de otros supertipos de objetos creados previamente.

Un ejemplo sencillo en el que se aprecia con facilidad qué es la herencia es una escuela, donde hay estudiantes, profesores y empleados. Todos ellos tienen características de una persona: un nombre, un número de identificación, un teléfono, etc. Por tanto se puede construir un supertipo de objeto llamado tipo\_persona que aglutine los atributos característicos de una persona.

```
SQL> create or replace type tipo_persona as object
  2  (id number,
  3   nombre    varchar2(10),
  4   telefonos listin)
  5  not final
  6  /
```

Tipo creado.

La creación del tipo lleva incluida la cláusula *not final* que indica que se pueden generar más tipos de objetos a partir de este. Dicho con nomenclatura POO, se pueden crear subclases a partir de la superclase, o dicho con un lenguaje más natural se pueden crear tipos de persona más específicos a partir del tipo genérico *persona*. A continuación se crea un tipo de persona específica, el estudiante:

```
SQL> create or replace type tipo_estudiante under tipo_persona
  2 (facultad varchar2(10),
  3  nota_media number)
  4 not final
  5 /
```

Tipo creado.

En este caso, la cláusula *under* especifica a qué supertipo pertenece el tipo estudiante. Nuevamente, se define como *not final* para poder generar clasificaciones más amplias.

Obsérvese que este tipo de construcciones pueden usarse también para las jerarquías en el modelo entidad relación. Ver sección 2.4.1.

Al igual que con cualquier otro objeto, se puede comprobar la estructura del tipo estudiante utilizando el comando describe, abreviado desc:

```
SQL> desc tipo_estudiante
tipo_estudiante extiende TJ.TIPO_PERSONA
tipo_estudiante NO es FINAL
Nombre          ¿Nulo?  Tipo
-----          -----
ID              NUMBER
NOMBRE          VARCHAR2(10)
TELEFONOS       LISTIN
FACULTAD        VARCHAR2(10)
NOTA_MEDIA      NUMBER
```

Incluso sería posible crear un tipo de subobjeto que tenga los atributos y métodos del tipo\_estudiante. Si además, se deseara crear un tipo para representar los estudiantes de intercambio, se podría definir:

```
SQL> create or replace type tipo_estudiante_intercambio
  2 under tipo_estudiante
  3 (pais varchar2(30),
  4  universidad varchar2(30))
  5 /
```

Tipo creado.

```
SQL> desc tipo_estudiante_intercambio
```

tipo_estudiante_intercambio extiende TJ.TIPO_ESTUDIANTE		
Nombre	¿Nulo?	Tipo
ID		NUMBER
NOMBRE		VARCHAR2(10)
TELEFONOS		LISTIN
FACULTAD		VARCHAR2(10)
NOTA_MEDIA		NUMBER
PAIS		VARCHAR2(30)
UNIVERSIDAD		VARCHAR2(30)

## 8.6. Operaciones DML

A lo largo del capítulo ya se ha mostrado en los ejemplos cómo utilizar la sentencia insert y select cuando se tratan tipos de objetos:

```

SQL> insert into empleado
  2 values (1,'Fernando','Moreno',tabla_hijos('Elena','Pablo'));

1 fila creada.

SQL> insert into empleado
  2 values (2,'David','Sanchez',tabla_hijos('Carmen','Candela'));

1 fila creada.

SQL> col hijos format a40
SQL> select * from empleado;

ID NOMBRE      APELLIDO HIJOS
----- -----
 1 Fernando    Moreno   TABLA_HIJOS('Elena', 'Pablo')
 2 David       Sanchez  TABLA_HIJOS('Carmen', 'Candela')

```

La única restricción es que no es posible operar sobre la tabla anidada:

```

SQL> select * from t_hijos;
select * from t_hijos
*
ORA-22812: no se puede hacer referencia a la tabla de
almacenamiento de una columna de tabla anidada

```

Para realizar actualizaciones se utiliza update:

```
SQL> update empleado
  2 set hijos=TABLA_HIJOS('Carmen','Candela','Cayetana')
  3 where id =1;

1 fila actualizada.
```

Y para realizar borrados se utiliza delete. Obsérvese cómo se puede utilizar un determinado método para la selección de los registros a borrar o eliminar:

```
SQL> select * from triangulos t where t.triangulo.area()>20;

      ID TRIANGULO(BASE, ALTURA)
-----
      2 TIPO_TRIANGULO(10, 10)

SQL> delete from triangulos t where t.triangulo.area()>20;

1 fila suprimida.
```

## 8.7. Las referencias

Una referencia es un puntero a un objeto creado a partir de su OID. Se utiliza principalmente para apuntar a un objeto que ya existe y no tener que duplicar la información. Por ejemplo, si se desea crear una tabla de mascotas en la que cada mascota tiene su veterinario, y esos veterinarios ya están almacenados en una tabla de objetos, se podría hacer uso de la palabra reservada REF para indicar que el veterinario ya existe y por tanto, solo se almacena una referencia a ese veterinario en la tabla de mascotas.

```
SQL> create type veterinario as object
  2 ( id integer,
  3   nombre varchar(100),
  4   direccion varchar(255));
  5 /
Tipo creado.

SQL> create type mascota as object
  2 (
```

```
nombre      varchar2(30),
descripcion varchar2(100),
precio      number,
porct_iva   number)
/

create or replace type tabla_articulos as
table of tipo_articulo;
/
```

5. Crea un tipo para la lista de la compra y otro para su detalle. La lista de la compra contendrá un identificador, fecha, cliente y una tabla de detalles de lista de la compra. A su vez, un detalle de lista de compra estará compuesto por un artículo y un campo que denote la cantidad pedida de ese artículo. Se deberá incluir en la definición una función miembro para calcular el total de la lista de la compra.

```
create or replace type tipo_lista_detalle as object (
    numero number,
    articulo tipo_articulo,
    cantidad number
)
/

create or replace type tab_lista_detalle as table of tipo_lista_detalle;
/

create or replace type tipo_lista_compra as object
(id number,
fecha DATE,
cli REF tipo_cliente,
Detalle tab_lista_detalle,
member function total return number
)
/
```

6. Crea ahora el cuerpo del tipo lista de la compra para definir el método total:

```
create or replace type body tipo_lista_compra as
member function total return number is
    i integer; tot number := 0;
begin
    for i in 1..Detalle.count loop
        tot := tot + (Detalle(i).cantidad * Detalle(i).articulo.precio ) *
            (1+(Detalle(i).articulo.porct_iva/100)); end loop;
    return tot;
end;
end;
/
```

7. Crea una tabla de clientes e inserta uno.

```
create table Clientes of tipo_cliente;
insert into Clientes values (1, 'Pedro', 'Suarez',
    TIPO_DIRECCION('Paseo del Museo 15', '28099'),
    TIPO_CONTACTO('917726525', 'psuarez@ono.com'), 0);
```

8. Crea una tabla para las listas de la compra e inserta una lista de la compra con un detalle de dos artículos para el cliente insertado anteriormente:

```
create table listas_de_compras of tipo_lista_compra
nested table Detalle store as tDetalle;

insert into listas_de_compras
select 1, current_date, ref(c), tab_lista_detalle(
    tipo_lista_detalle(1, tipo_articulo(1, 'Barra de pan', 'Tipo baguette', 1, 7), 4),
    tipo_lista_detalle(2, tipo_articulo(2, 'Lonchas de jamón', 'Iberico Bellota', 6, 7), 4)
) from Clientes c where c.id=1;
```

9. Muestra con una select los datos de la lista de la compra:

```
SQL> col cliente format a40
SQL> col detalle format a50
SQL> set linesize 132
SQL> select id, fecha, deref(cli) as cliente, detalle from listas_de_compras;
ID FECHA      CLIENTE(ID, NOMBRE, APELLIDO, DIRECCION) DETALLE(NUMERO, ARTICULO(ID_ART, NOMBRE, DESCRIPCION)
-- -----
1 13/06/11 TIPO_CLIENTE('1', 'Pedro', 'Suarez', TIP_TAB_LISTA_DETALLE(TIPO_LISTA_DETALLE(1, TIPO_ARTICULO('Barra de pan', 'Tipo baguette', 1, 7), 4),
    TIPO_DIRECCION('Paseo del Museo 15', 28099) ULO(1, 'Barra de pan', 'Tipo baguette', 1, 7), 4),
    TIPO_CONTACTO('917726525', 'psuarez@ono. TIPO_LISTA_DETALLE(2, TIPO_ARTICULO(2, 'Lonchas de com'), 0) jamón', 'Iberico Bellota', 6, 7), 4))
```

10. Construye una select para mostrar por pantalla el id de una lista de la compra y su total.

```
SQL> select id, c.total() from listas_de_compras c;
ID  C.TOTAL()
----- 
1      29,96
```



## 8.9. Prácticas Propuestas

---

### Práctica 8.2: Tipos básicos y herencia

Realiza los siguientes ejercicios:

1. Crea un tipo para las direcciones de empresas, contendrá el tipo de dirección y la dirección (número, calle y piso).
2. Crea un tipo colección para almacenar hasta 3 direcciones (fiscal, postal y administrativa).
3. Crea un supertipo empresa con los atributos CIF, Nombre y sus direcciones.
4. Crea un subtipo de empresa llamada Sociedad Anónima con los atributos número de accionistas, capital social y presupuesto.
5. Crea un subtipo de empresa llamada Sociedad Limitada que tenga una lista de socios, donde se especifique el nombre del socio y un porcentaje de posesión de la empresa.
6. Crea una tabla de empleados que tenga una referencia a una Empresa.



---

### Práctica 8.3: Comandos DML con objetos

Con los tipos de la práctica anterior realiza las siguientes operaciones:

1. Crea una tabla de empresas de tipo sociedades anonimas.
2. Inserta una sociedad anonima con dos direcciones.
3. Crea una tabla de empresas de tipo sociedades limitadas.
4. Inserta una sociedad limitada con dos direcciones y cuatro socios.
5. Inserta un empleado para cada una de las empresas insertadas.
6. Crea una consulta que muestre la empresa para la que trabajan los empleados utilizando la función DEREF.



---

## Práctica 8.4: Números complejos

Un número complejo  $a+bi$  está compuesto por una parte imaginaria ( $b$ ) y una parte real ( $a$ ). Se desea crear un tipo de objeto llamado tipo\_complejo con una serie de métodos para poder operar con ellos. Se pide:

1. Crea el tipo tipo\_complejo que contenga los atributos *imaginario* y *real*, y que defina la cabecera de los siguientes métodos:
  - Módulo: Retorna un valor numérico resultado de operar los valores imaginarios y reales con la fórmula  $\sqrt{a^2 + b^2}$ .
  - Valor: Retorna una cadena de caracteres con la representación del número complejo ( $a+bi$ ).
  - Conjugado: Retorna una cadena de caracteres con la representación del conjugado del número complejo ( $a-bi$ ).
  - Multiplicar(Número Complejo): Actualizará los atributos del número complejo resultado de multiplicar los propios atributos por otro número complejo que será pasado como parámetro:  $(a+bi)(c+di) = ac - bd + (ad + bc)i$
2. Crea el cuerpo del tipo desarrollando los métodos definidos en la cabecera del tipo.
3. Crea el tipo tipo\_tabla\_complejos para almacenar una serie de números complejos. Después, crea el tipo tipo\_lista\_complejos que almacene una tabla de complejos.
4. Crea una tabla de listas de números complejos e inserta un registro con 5 números.
5. Crea un bloque anónimo para multiplicar el primer número insertado por el segundo.
6. Crea un bloque anónimo que muestre el módulo, valor y conjugado de los números complejos insertados.
7. Crea una función que reciba una tabla de números complejos y devuelva un número complejo con la suma de todos ellos.
8. Crea un bloque anónimo que invoque a la función anterior y visualice la suma de los números complejos insertados.



## 8.10. Resumen

Los conceptos clave de este capítulo son los siguientes:

- Muchos SGBD permiten la creación de tipos personalizados, que son la base de las BBDD-OR.
- Los atributos multivaluados pueden almacenarse en colecciones (varray) de un número fijo de elementos o en tablas anidadas (nested tables) para almacenar un número variable de elementos .
- Los objetos definidos en Oracle se pueden consultar utilizando la tabla dba\_objects.
- Un tipo de objeto se crea con la sentencia DDL *create type tipo as object*.
- Es posible crear tipos de tabla de objetos con la sentencia DDL *create type tipo\_tabla as table of objeto*.
- Se puede anidar la creación de tipos en tantos niveles como se deseé.
- Un OID es un número de 16 bytes para mejorar la búsqueda de la información. Se puede consultar con la función object\_id.
- Un método es una función o un procedimiento miembro definido dentro de un tipo.
- Para invocar a un método se utiliza el operador punto (.) *objeto.metodo()*.
- Se pueden crear jerarquías de clases (*u objetos tipo*) definiendo tipos *NOT FINAL* .
- Para crear un subtipo de objetos se utiliza la cláusula *under supertipo*.
- Es posible compartir objetos mediante referencias a objetos utilizando la cláusula REF.
- Para ver la información de un objeto referenciado mediante REF se utiliza la función DEREF.

## 8.12. Comprueba tu aprendizaje

1. ¿Qué es una base de datos OR?
2. ¿Cuál es la principal diferencia con las bases de datos relacionales?
3. ¿Qué es un atributo multivaluado?
4. ¿Cómo encaja un atributo multivaluado en las bases de datos OR?
5. Enumera dos formas distintas de almacenar atributos multivaluados en una tabla.
6. Explica las diferencias entre usar una tabla anidada y una colección.
7. Escribe la sintaxis para crear:
  - Tipos
  - Tipos colecciones
  - Tipos Tabla
  - Tabla de objetos
8. ¿Qué es un tipo tabla? ¿Cómo se crean?
9. ¿Cómo se pueden consultar en Oracle qué tipos de objetos hay definidos?
10. ¿Cómo se puede consultar en Oracle las tablas anidadas creadas?
11. ¿Cómo se crea un método en un tipo?
12. Explica cómo se puede invocar a un método definido en un tipo.
13. Comenta cómo se aplica la herencia a las BBDD-OR.
14. ¿Cómo se puede insertar un registro en una tabla de objetos?
15. Define para qué sirven las siguientes funciones
  - object\_id
  - object\_value
  - DEREF
  - REF
16. Define para qué sirven las siguientes cláusulas
  - store as
  - under
  - final
  - not final
  - member
17. ¿Qué es una referencia? ¿Para qué usa?
18. ¿Cómo se puede insertar una referencia a un objeto en un tabla de objetos?