

## ÍNDICE

<b>1. INTRODUCCIÓN .....</b>	<b>2</b>
▪ SGBD-OR .....	2
▪ CARACTERÍSTICAS .....	2
▪ MODELO RELACIONAL ANIDADO .....	3
▪ VISTAS ÚTILES .....	3
▪ PERMISOS NECESARIOS .....	3
▪ TIPOS DE DATOS .....	4
<b>2. TIPOS DE DATOS MULTIVALUADOS.....</b>	<b>4</b>
2.1. ARRAY (COLECCIÓN) .....	4
▪ DEFINICIÓN .....	4
▪ CREACIÓN Y BORRADO .....	5
▪ MANIPULACIÓN Y CONSULTA .....	6
ACTIVIDAD. MANIPULACIÓN DE ARRAYS .....	7
2.2. TABLA ANIDADA (NESTED TABLE) .....	7
▪ DEFINICIÓN .....	7
▪ CREACIÓN Y BORRADO .....	8
▪ MANIPULACIÓN Y CONSULTA .....	9
ACTIVIDAD. MANIPULACIÓN DE TABLAS ANIDADAS .....	10
<b>3. ORIENTACIÓN A OBJETOS.....</b>	<b>11</b>
3.1. TIPO OBJETO.....	11
▪ DEFINICIÓN .....	11
▪ CREACIÓN Y BORRADO .....	11
▪ USO DEL TIPO OBJETO EN UN CAMPO DE UNA TABLA QUE SOLO TENGA ESE CAMPO .....	12
ACTIVIDAD. TABLA CON UN ÚNICO CAMPO DE TIPO OBJETO .....	13
▪ USO DEL TIPO OBJETO EN UN CAMPO DE UNA TABLA .....	14
ACTIVIDAD. TABLA CON CAMPO DE TIPO OBJETO .....	15
▪ USO DEL TIPO OBJETO EN UNA TABLA ANIDADA .....	16
ACTIVIDAD. TABLAS CON OBJETOS.....	18
3.2. MÉTODOS.....	19
3.3. CONSTRUCTOR .....	20
ACTIVIDAD. MÉTODOS .....	21
3.4. HERENCIA.....	22
ACTIVIDAD. HERENCIA .....	23
<b>4. ANEXO: RESUMEN DE TIPOS DE DATOS.....</b>	<b>24</b>

En este documento, a no ser que se indique lo contrario, se reflejan los comandos SQL referidos al Sistema Gestos de Base de Datos (SGBD) Oracle Database.

## 1. Introducción

### ▪ SGBD-OR

En los años noventa irrumpieron los lenguajes de programación orientados a objetos frente a la programación estructurada. Esto hizo que aparecieran las bases de datos orientadas a objetos, que organizaron los datos de forma adecuada para poder ser manipulada con facilidad por estos nuevos lenguajes de programación.

Los Sistemas Gestores de Bases de Datos Orientados a Objetos puros no son demasiados. Sin embargo, los desarrolladores de los SGBD Relacionales consideraron adecuada la incorporación de ciertos conceptos de orientación a objetos en sus sistemas dando lugar así a los **SGBD Objeto-Relacionales** (SGBDOR).

Algunos SGBDOR:

- Oracle
- Microsoft SQL Server
- IMB DB2
- PostgreSQL
- SAP
- IBM Informix

A pesar de que los SGBDOR han ampliado su repertorio de objetos, el estándar SQL:1999 ya incluía sus elementos básicos, en los que nos vamos a centrar.

La principal diferencia entre las BBDD-R y las BBDD-OR radica en la existencia de **tipos de datos**. Estos tipos se pueden construir mediante sentencias DDL y serán la base del desarrollo de BBDD-OR.

### ▪ Características

- ▶ **Facilita el manejo de datos desde los lenguajes de programación orientados a objetos.**

Es posible recuperar datos de la base de datos para instanciar objetos de programación, y recorrerlos mediante los atributos, métodos y funciones apropiados.

- ▶ **Incorpora campos multivaluados**

Se pueden definir tipos de datos complejos (colecciones, tablas anidadas y objetos), con severas consecuencias para el modelo relacional subyacente puesto que **estos tipos de datos son considerados multivaluados y por tanto contradicen frontalmente la primera forma normal (1FN)**, al no garantizar la atomicidad de los atributos.

**Ejemplo de campo multivaluado.** Se tiene la tabla libros con varios valores en el campo “temas” (podría implementarse mediante una colección o tabla anidada):

IDLibro	Título	Temas
14	El Quijote	Novela, Literatura clásica española
17	Romeo y Julieta	Teatro, Literatura clásica inglesa

Los tipos de datos pueden representar una estructura, las tablas **dejan de ser bidimensionales y pueden incluir nuevas dimensiones** de profundidad de la información.

**Ejemplo de tipo complejo.** Se tiene la tabla libros con el campo editorial como un campo complejo:

IDLibro	Título	Editorial
14	El Quijote	{ Nombre: Planeta País: España }
17	Romeo y Julieta	{ Nombre: Sterling País: Reino Unido }

Incluso podría haber varias instancias:

IDLibro	Título	Editorial	
14	El Quijote	{ nombre: Planeta país: España }	{ nombre: Austral país: España }
17	Romeo y Julieta	{ Nombre: Sterling País: Reino Unido }	

#### ▪ Modelo relacional anidado

Dado que el modelo objeto-relacional permite que los valores de un campo sean registros de otra tabla (que, a su vez, podría contar con registros de una tercera tabla como valores de sus propios campos), esta forma de almacenar la información se conoce como **modelo relacional anidado**. Gracias a esta característica se pueden eliminar tablas relacionadas del modelo relacional clásico y simplificar las sentencias SELECT de consulta, mejorando la vinculación con un lenguaje de programación orientado a objetos.

#### ▪ Vistas útiles

En Oracle, se puede consultar la vista **user\_types** para examinar los tipos que se han creado en el SGBD.

```
select* from USER_TYPES;
```

#### ▪ Permisos necesarios

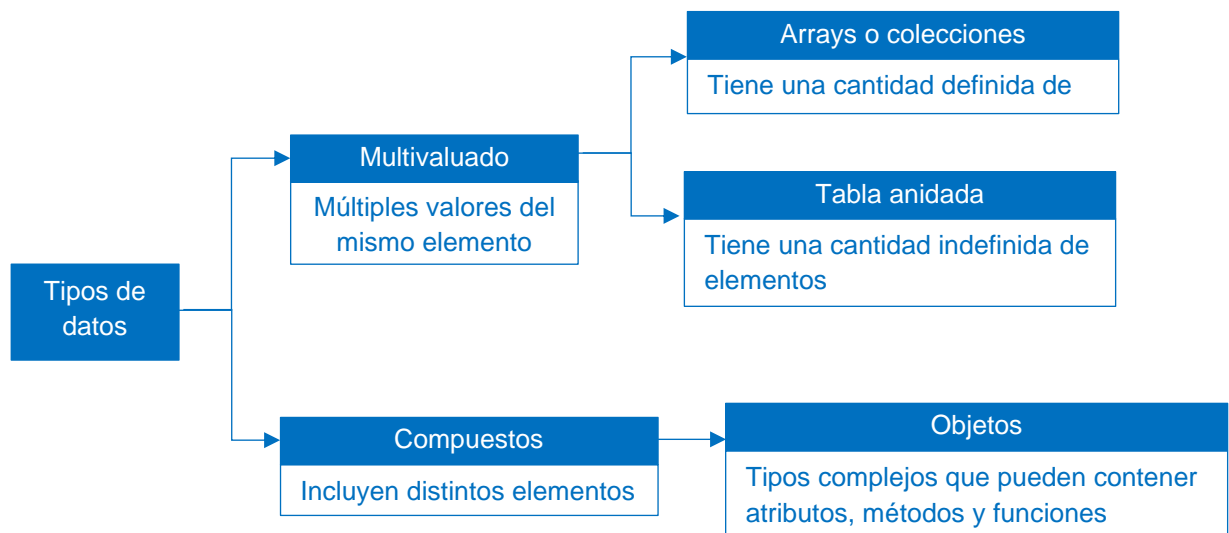
Para definir un tipo de dato en Oracle es necesario tener permiso de creación de tipos (create any type), además de todos los permisos necesarios para manipularlo:

```
GRANT CREATE ANY TYPE,
      ALTER ANY TYPE,
      DROP ANY TYPE,
      EXECUTE ANY TYPE,
      UNDER ANY TYPE
TO usuario;
```

Permite usar y referenciar tipos

Permite crear subtipos

- Tipos de Datos



## 2. Tipos de datos multivaluados

### 2.1. Array (colección)

- Definición

Un **array** o colección es una lista de elementos a los que se puede acceder mediante su posición. Esta lista es de **tamaño fijo** (tiene un número determinado de elementos).

Por tanto, si un campo es de tipo array permitirá asignar varios valores: **es multivaluado**.

TABLE

Campo1	Campo2	Campo3
...	...	...
...	...	...

Campo declarado con un tipo Array:  
sus datos serán listas de valores

### ▪ Creación y borrado

La sintaxis para la creación de un tipo array indicándole el número de elementos que contendrá:

```
CREATE [OR REPLACE] TYPE nombreTipo
AS VARRAY(numElementos) OF tipoDato;
/
```

Número máximo de elementos que contendrá el array

Tipo de cada elemento

La sintaxis para la creación de una tabla con un campo tipo array es:

```
CREATE TABLE nombreTabla(
    ...
    campo tipoArray,
    ...);
```

Nota: a un campo de tipo array no se le pueden aplicar restricciones como ser clave primaria.

**Ejemplo.** Sentencia de creación de una tabla con un campo tipo array:

- 1) Creación del tipo “tipotemasArray” con 10 elementos de tipo varchar2(35) cada uno:

```
create or replace type tipotemasArray
as varray(10) of varchar2(35);
/
```

- 2) Creación de la tabla haciendo uso del nuevo tipo:

```
create table libro1(
    idlibro number(2) primary key,
    titulo varchar2(40) not null,
    temas tipotemasArray
);
```

“temas” puede albergar hasta 10 valores varchar2(35)

Para el borrado de un tipo array:

```
DROP TYPE nombreTipo;
```

### ▪ Manipulación y consulta

Para consultar, insertar, borrar o actualizar información en tablas con campos de tipo colección es necesario utilizar el constructor del tipo:

`nombreTipo(valor1,valor2,...)`

**Ejemplo.** Sentencias de inserción y actualización:

```
insert into libro1
values (17,'Romeo y Julieta',tipotemasArray('teatro','literatura clásica
inglesa'));
```

↑  
constructor

```
update libro1
set temas=tipotemasArray('Teatro' , 'literatura clásica anglosajona')
where idlibro =17;
```

**Nota:** no es posible trabajar desde SQL con valores individuales del Array

**Ejemplo.** Sentencia de consulta en una tabla con un campo tipo colección.

```
select * from libro1;
```

IDLIBRO	TITULO	TEMAS
17	Romeo y Julieta	ALUMNO.TIPOTEMASARRAY('Teatro', 'literatura clásica anglosajona')

Es importante destacar que **desde una consulta SQL, los valores de un array (varray) solamente pueden ser accedidos y recuperados en un bloque**. Es decir, no se puede acceder a los elementos de un array individualmente.

Sin embargo, desde un programa PL/SQL sí que es posible definir un bucle que itere sobre los elementos de un array. Para ello:

<code>campo(n)</code>	Acceso a la posición n de un campo de tipo VARRAY
<code>campo.first</code>	Acceso a la primera posición de un campo de tipo VARRAY
<code>campo.last</code>	Acceso a la última posición de un campo de tipo VARRAY

### Actividad. Manipulación de arrays

- a) Con el usuario alumno crea un tipo llamado `colec_hijos` de tipo **array** que pueda albergar el nombre de al menos 10 personas. Recuerda que lo primero que debes hacer es dar permisos al usuario sobre los tipos.
- b) Crea una tabla llamada `EMPLE_HIJOS1` donde podamos albergar los hijos de cada empleado. Para ello la tabla contará con los campos `id_persona`, `nombre`, `apellido1`, `apellido2` e `hijos`.
- c) Inserta los siguientes datos.

ID_Persona	Nombre	Apellido1	Apellido2	Hijos
1	Francisco	Romero	Torres	(Luisa, Alba)
2	Julieta	Jiménez	López	(Carlos, José, Pedro)

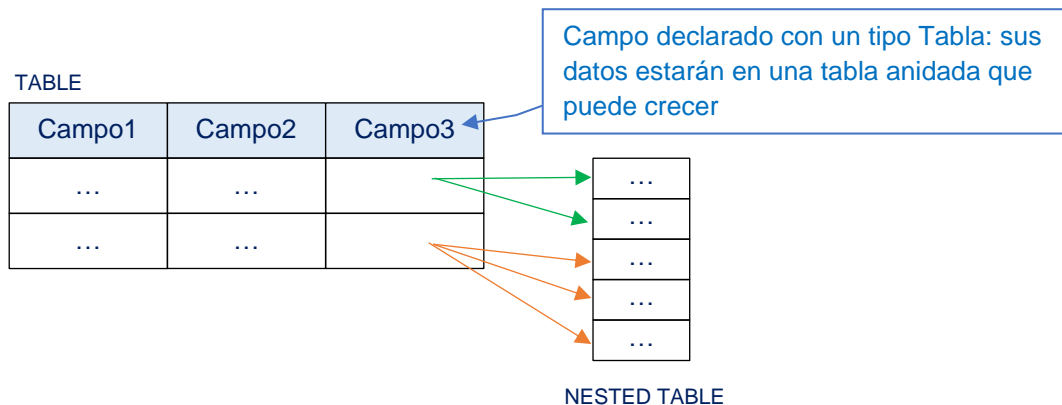
- d) Obtén todos los datos de la tabla empleados (`select`).
- e) Analiza el programa `bloque1.sql` y ejecútalo.

## 2.2. Tabla anidada (nested table)

### ▪ Definición

Una **tabla anidada** es un tipo especial de tabla que permite introducir una lista de elementos a los que se puede acceder mediante su posición. Esta lista es de **tamaño variable** (se puede insertar un número indefinido de registros en la tabla anidada). El tamaño de cada elemento sí es fijo.

Por tanto, si un campo es de tipo tabla anidada asignar varios valores: **es multivaluado**.



### ▪ Creación y borrado

La sintaxis para la creación de un tipo tabla:

```
CREATE [OR REPLACE] TYPE nombreTipo
AS TABLE OF tipoDato;
/
```

**Ejemplo.** Creación de un tipo tabla anidada. Este tipo contendrá una lista ilimitada de elementos, cada uno de ellos será un varchar2(30):

```
create or replace type tipotemasTabla
as table of varchar2(35);
/
```

Cuando se utilice este tipo de datos en un campo de una tabla, dado que a priori no se conoce el espacio que ocuparán (es una lista ilimitada de elementos), **será necesario definir una “tabla almacén”** en la que se almacenarán los datos de ese campo. Esta “tabla almacén” es una tabla especial denominada **tabla anidada** utilizada para este fin, no es necesario crearla como una tabla usual con CREATE TABLE.

La sintaxis para la creación de una tabla con un campo de tipo tabla es:

```
CREATE TABLE nombreTabla(
...
campo nombreTipo,
...
) NESTED TABLE campo STORE AS nombreTablaAlmacen;
```

NESTED TABLE (tabla anidada) asocia “campo” (declarado como tipo tabla) a la “tabla almacén” en la que se guardarán los datos de ese campo



Nota: vamos a trabajar con un solo campo de tipo tabla

**Ejemplo.** El campo “temas” dentro de la tabla LIBRO puede albergar un numero indefinido de elementos porque está declarado como TABLE de VARCHAR(30):

```
create table libro2(  
    idlibro number(2) primary key,  
    titulo varchar2 (40) not null,  
    temas tipotemasTabla  
)nested table temas store as t_temas;
```

Para el borrado de un tipo tabla anidada:

```
DROP TYPE nombreTipo;
```


#### ▪ Manipulación y consulta

Para consultar, insertar, borrar o actualizar información en tablas con campos de tipo tabla anidada, al igual que con los arrays, es necesario utilizar el constructor del tipo:

```
nombreTipo(valor1,valor2,...)
```

**Ejemplo.** Sentencias de inserción y actualización en una tabla con un campo tipo tabla:

```
insert into libro2  
values (17, 'romeo y julietta',tipotemasTabla('teatro', 'literatura clásica  
inglesa'));
```



```
update libro2  
set temas=tipotemasTabla('Teatro' , 'literatura clásica anglosajona' )  
where idlibro =17;
```

**Nota: no es posible trabajar desde SQL con valores individuales de la tabla anidada**

**Ejemplo.** Sentencia de consulta en una tabla con un campo tipo tabla.

```
select * from libro2;
```

IDLIBRO	TITULO	TEMAS
17	romeo y julieta	ALUMNO.TIPOPOTEMASTABLA('Teatro', 'literatura clásica anglosajona')

Es importante destacar que **desde una consulta SQL, los valores de una tabla anidada solamente pueden ser accedidos y recuperados en un bloque**. Es decir, no se puede acceder a los elementos de un array individualmente.

Sin embargo, desde un programa PL/SQL sí que es posible definir un bucle que itere sobre los elementos de una tabla anidada. Para ello se puede seguir el mismo procedimiento que en el caso de los arrays.

### Actividad. Manipulación de tablas anidadas

- Crea un tipo llamado tabla\_hijos de tipo **tabla anidada** que pueda albergar el nombre de un número indeterminado de personas.
- Crea una tabla llamada EMPLE\_HIJOS2 donde podamos albergar los hijos de cada empleado. Para ello la tabla contará con los campos id\_persona, nombre, apellido1, apellido2 e hijos.
- Inserta los siguientes datos.

ID_Persona	Nombre	Apellido1	Apellido2	Hijos
1	Francisco	Romero	Torres	(Luisa, Alba)
2	Julieta	Jiménez	López	(Carlos, José, Pedro)

- Obtén todos los datos de la tabla Emple\_Hijos2.
- Analiza el programa bloque2.sql y ejecútalo.

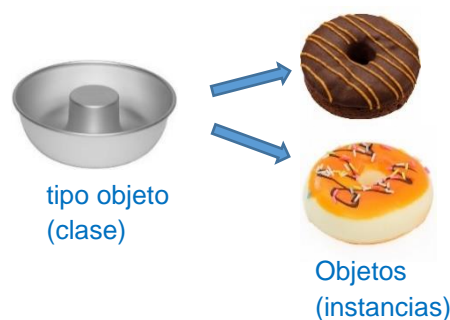
### 3. Orientación a objetos

#### 3.1. Tipo Objeto

##### ▪ Definición

Recordemos que en programación orientada a objetos (POO), una clase es como el molde o un patrón para la fabricación de los objetos. Por tanto, 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 una única fila en tantos niveles como se desee.



##### ▪ Creación y borrado

Un tipo de objeto puede contener atributos, procedimientos y funciones. La sintaxis básica para la creación de un tipo objeto es:

```
CREATE [OR REPLACE] TYPE nombreTipoObj AS OBJECT(
    NombreCampo tipoCampo,
    ...
    [MEMBER FUNCTION nombreFuncion return tipoDato,]
    ...
    [MEMBER PROCEDURE nombreProcedimiento,]
    ...);
/
```



Tipo Objeto

```
DESCRIBE nombreTipoObj; --ver la estructura del objeto
```

Para instanciar un objeto, se emplea el constructor:

```
nombreTipoObj(valor1,valor2,...)
```



Objeto  
(instancia)

Para el borrado de un tipo objeto:

```
DROP TYPE nombreTipo;
```

**Ejemplo.** Creación del tipo de objeto “tipo\_telefono”, que contiene dos atributos que describirán a los objetos que se instancien:

```
create or replace type tipo_telefono as object (
  tipo varchar2(30), --personal, empresa, casa, ...
  numero number(9)
);
/
```

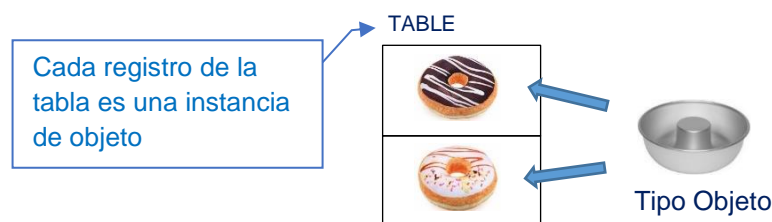
Observa que un objeto no tiene clave primaria.

A continuación se exponen tres casos de uso de un objeto en tablas:

- ▶ Uso del tipo objeto en un campo de una tabla que solo tenga ese campo
- ▶ Uso del tipo objeto en un campo de una tabla
- ▶ Uso del tipo objeto en una tabla anidada

#### ▪ Uso del tipo objeto en un campo de una tabla que solo tenga ese campo

Es posible definir una tabla que **solo** contenga un campo de tipo objeto. Este es un caso simplificado en el que **se manipulan los datos de la tabla como si los atributos del tipo de objeto fueran directamente atributos de la tabla**.



```
CREATE TABLE nombreTabla OF tipoObjeto;
```

Nota: no hay clave primaria

**Ejemplo.** Se tiene una tabla vehículos compuesta por elementos de tipo objeto “coche”:

1) Declaración del tipo objeto:

```
create type tipo_telefono as object(
  tipo varchar2(30),
  numero number(9));
/
```

2) Creación de la tabla:

```
create table telefonos of tipo_telefono;
```

## 3) Sentencias de inserción, consulta, actualización y borrado:

`insert into telefonos values (`  
  
`tipo_telefono('oficina',916002000));`

constructor

Aparecen los atributos del tipo objeto  
como si fuesen atributos de la tabla

`select * from telefonos;`

TIPO	NUMERO
oficina	916002000

`select`   
  
`tipo` `from telefonos;`

Se accede directamente al atributo del objeto

`update telefonos set`   
  
`numero=916002033 where`   
  
`tipo='oficina';`

Se accede directamente al atributo del objeto

`delete from telefonos where`   
  
`tipo = 'oficina';`

Se accede directamente al atributo del objeto

**Actividad. Tabla con un único campo de tipo objeto**

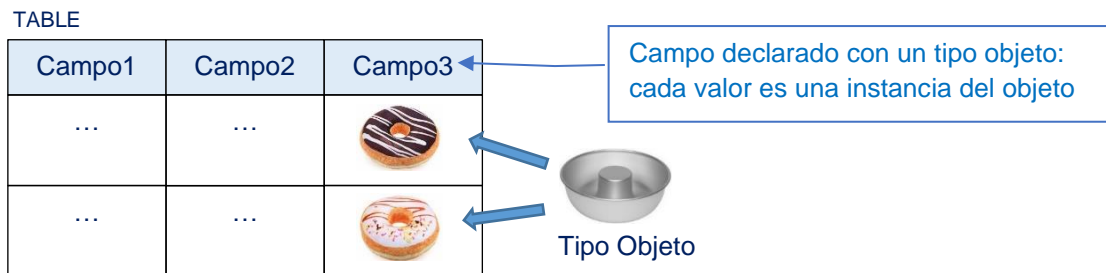
- Crea un **tipo objeto** llamado **cuenta** que contenga los campos:
  - NumeroCuenta number(6)
  - Titular varchar2(30)
  - Saldo number(30)
- Crea una tabla llamada SUCURSAL que albergará las diferentes cuentas de los clientes. Tendrá solo un campo de tipo "cuenta".
- Inserta los siguientes datos.

NumeroCuenta	Titular	Saldo
000001	Paco Romero	1000000
000002	María Jiménez	25000

- Obtén todos los datos de la tabla Sucursal.
- Borra la cuenta 000002

### ▪ Uso del tipo objeto en un campo de una tabla

Es posible definir un campo de una tabla como un objeto. En este caso cada registro contendrá un solo objeto dentro de ese campo.



Para manipular los datos de tipo objeto de esta tabla es necesario emplear un alias de tabla y acceder al dato:

`aliasTabla.campo.atributoObjeto`

**Ejemplo.** Supongamos una tabla CLIENTES donde cada cliente tiene un teléfono del que deberemos saber el tipo y el número.

- 1) Declaración del tipo objeto:

```
create or replace type tipo_telefono as object
(tipo varchar2(30),
numero number(9));
/
```

- 2) Creación de una tabla donde se hace uso de este nuevo tipo objeto:

```
create table clientes1(
idcliente number (2) primary key,
nombre varchar2 (30) not null,
apellidos varchar2 (30),
direccion varchar2 (30),
poblacion varchar2 (30),
provincia varchar2 (30),
telefono tipo_telefono
);
```

- 3) Sentencias de manipulación y consulta en una tabla con un campo de tipo objeto:

```
insert into clientes1 values(1, 'Pablo', 'Pérez', 'C/Mayor n.20',
'Madrid', 'Madrid', tipo_telefono('Fijo',911234567) );
```

```
select * from clientes1;
```

IDCLIENTE	NOMBRE	APELLIDOS	DIRECCION	POBLACION	PROVINCIA	TELEFONO
1	Pablo	Pérez	C/Mayor n.20	Madrid	Madrid	[ALUMNO.TIPO_TELEFONO]

```
delete from clientes1 where idcliente =1;
```

aliasTabla.campo.atributoObjeto

```
select C.telefono.tipo from clientes1 C where idcliente= 1;
```

aliasTabla.campo.atributoObjeto

```
update clientes1 C set C.telefono.numero=915000000 where idcliente= 1;
```

aliasTabla.campo.atributoObjeto

```
delete from clientes1 C where C.telefono.tipo='Fijo';
```

### Actividad. Tabla con campo de tipo objeto

Para registrar las funciones en un teatro se crea una tabla orientada a objetos

a) Crea un **tipo de objeto** llamado **precio\_funcion** que contenga los campos:

- “Infantil” number(5,2)
- “General” number(5,2)
- “Reducida” number(5,2)

b) Crea una tabla llamada FUNCIONES que contendrá las funciones actuales en el teatro. Los campos serán:

- Obra: varchar2(35)
- Hora: varchar2(5). El formato debe ser: dos caracteres, dos puntos (:) y dos caracteres (por ejemplo “20:00”)
- El campo “precio” debe ser de tipo “precio\_funcion”.

La clave primaria será “obra” y “hora”.

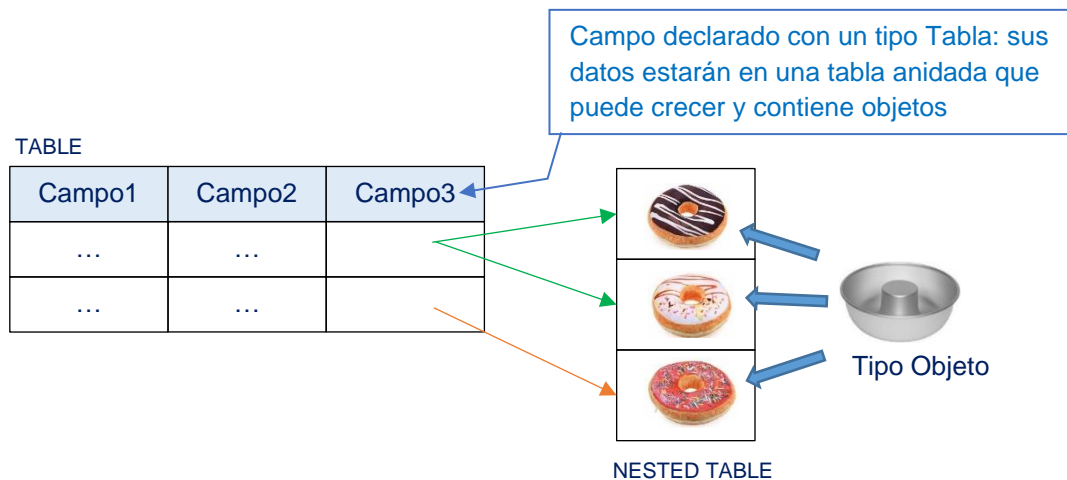
c) Inserta los siguientes datos.

Obra	Hora	Precio
El Verdugo	19:00	Infantil: 14.50 €
		General: 22 €
		Reducida: 18 €
El Verdugo	21:00	Infantil: 15.50 €
		General: 24 €
		Reducida: 20 €

d) Modifica el precio de la entrada Reducida de la obra “El Verdugo” a las 21:00 para que tenga el valor 18 €. Visualiza que el cambio se haya realizado correctamente (select)

### ▪ Uso del tipo objeto en una tabla anidada

Es posible definir un campo de una tabla como una tabla anidada en la que cada elemento de esa tabla anidada sea un objeto.



Pasos:

2. Creación del tipo Objeto
3. Creación de tabla anidada donde guardaremos el tipo objeto (tipo tabla)
4. Creación de la tabla final.

**Ejemplo.** Supongamos una tabla CLIENTES donde cada cliente puede tener más de un teléfono del que deberemos conocer el tipo de teléfono y el número.

- 1) Declaración del tipo objeto:

```
create or replace type tipo_telefono as object
(tipo varchar2(30),
numero number);
```

- 2) Declaración del tipo tabla anidada:

```
create or replace type listin as table of tipo_telefono;
```

- 3) Creación de una tabla donde se hace uso de este nuevo tipo objeto:

```
create table clientes2(
idcliente number (2) primary key,
nombre varchar2 (30) not null,
apellidos varchar2 (30),
direccion varchar2 (30),
poblacion varchar2 (30),
provincia varchar2 (30),
telefonos listin)
nested table telefonos store as t_telefonos;
```



- 4) Sentencias de manipulación y consulta en una tabla con un campo tipo tabla de objetos:

```
insert into clientes2 values(1, 'Pablo', 'Pérez', 'mayor 20', 'madrid',
    'madrid', listin( tipo_telefono('fijo', 911234567),
                    tipo_telefono('móvil personal', 600000001),
                    tipo_telefono('móvil empresa', 600000002))
);
```

```
select * from clientes;
```

Haciendo doble click y pulsando el lápiz es posible ver los valores de tabla anidada

OMBRE	APELLIDOS	DIRECCION	POBLACION	PROVINCIA	TELEFONOS
o	Pérez	Mayor 20	Madrid	Madrid	ALUMNO.LISTIN(ALUMNO.TELEFONO('fijo', 911234567), ALUMNO.TELEFONO('móvil

ALUMNO.LISTIN
ALUMNO.TELEFONO('fijo', 911234567)
ALUMNO.TELEFONO('móvil personal', 600000001)
ALUMNO.TELEFONO('móvil empresa', 600000002)

```
delete from clientes where idcliente =1;
```

**Nota:** hay que recordar que no es posible trabajar desde SQL con valores individuales de la tabla anidada:

```
update clientes2 set telefonos=
    listin( tipo_telefono('fijo',911234567),
            tipo_telefono('móvil personal', 600000001),
            tipo_telefono('móvil empresa', 600000055))
where idcliente=1;
```

### Actividad. Tablas con Objetos

Se va a utilizar una tabla CLIENTES\_TIENDA con los datos de los clientes de una tienda. Cada uno puede tener varias tarjetas para realizar pagos, que se declararán como objetos.

- a) Crea un **tipo objeto** llamado **tarjetas** que contenga los campos:
  - Clase varchar2(20)
  - Numero number (16).
- b) Crea un tipo tabla anidada llamado **cartera** que albergará las diferentes tarjetas de los clientes.
- c) Crea una tabla llamada CLIENTES\_TIENDA donde podamos albergar los datos de los clientes de una tienda y las tarjetas con las que pueden realizar sus pagos. Para ello la tabla contará con los campos idCliente, nombre, apellidos, direccion, tarjetas. (Elige los tipos adecuados para cada campo "tarjetas" es de tipo cartera).
- d) Inserta los siguientes datos.

IDCliente	Nombre	Apellido	Dirección	Tarjetas
1	Paco	Romero	Real 10	(Visa Débito, 0000000000) (Mastercard, 0000000001)
2	María	Jiménez	Gran Vía 1	(Visa Oro, 0000000002) (American E, 0000000003) (Visa Crédito, 0000000004)

- e) Obtén todos los datos de la tabla y visualiza los datos de forma completa.

Ver [resumen tipos de datos](#)

### 3.2. Métodos

Hasta ahora solo hemos utilizado los tipo objeto para definir tipos de datos compuestos pero, 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 miembros que pertenecen a un tipo de objeto se denominan métodos.**

Los tipos de objeto que contienen métodos, **deben declarar un BODY** que contendrá el código de todos los métodos.

```
CREATE [OR REPLACE] TYPE BODY nombreTipoObj AS|IS
MEMBER PROCEDURE nombreProcedimiento
    Código del procedimiento
MEMBER FUNCTION nombreFunción RETURN tipoDato
    Código de la función
...
END;
```

Bloque PL/SQL: declaraciones, instrucciones y excepciones:

```
AS|IS
    [...]
BEGIN
    ...
[EXCEPTION
    ...]
END;
```

Cuando un tipo de objeto no tiene métodos, no es necesario declarar su BODY.

**Ejemplo:** Se desea modelar un objeto triángulo y almacenar en la BBDD cientos de triángulos pudiendo calcular el área de cada triángulo:

```
create or replace type tipo_triangulo as object(
    base number(5),
    altura number(5),
    member function area return real);
/
```

Se ha definido la cabecera de una función llamada "area" (método)

Creación del cuerpo del tipo tipo\_triangulo:

```
create or replace type body tipo_triangulo as
member function area return real
is
    a number;
begin
    a:= base*altura/2;
    return a;
end;
end;
/
```

Creación de una tabla que almacene todos los triángulos. Contendrá dos campos, el idTriangulo y el objeto triángulo:

```
create table triangulos
(idtriangulo number,
 triangulo tipo_triangulo);
```

Ya hemos estudiado como se manipulan datos en una tabla que incluye un objeto en una de sus columnas. Se hará a través del constructor como se muestra a continuación.

### 3.3. Constructor

Un tipo especial de métodos serán los **constructores** que serán funciones miembros del tipo de objeto y **se crearán de forma automática**. Para referirse a los atributos miembros se utilizará la palabra **SELF** (equivalente a this en JAVA).

Estos constructores tienen el mismo nombre que el tipo objeto, y devolverán el objeto creado.

Hay que tener en cuenta que cuando se insertan datos (INSERT INTO), realmente se está llamando al **constructor** por defecto.

**Ejemplos.** Se retoma la tabla TRIÁNGULOS del ejemplo anterior.

- Insertar datos:

```
insert into triangulos values (1, tipo_triangulo(5,5));
insert into triangulos values (2, tipo_triangulo(10,10));
```

constructor  
↑

- Consultar un atributo del objeto:

```
select t.triangulo.base from triangulos t;
```

aliasTabla.campoTabla.atributoObjeto  
↑

- Llamada a un método del objeto (en este caso es una función, por lo que debe llevar paréntesis):

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

aliasTabla.campoTabla.métodoObjeto  
↑

- Borrar datos:

```
delete from triangulos where idtriangulo = 1;
```

- Bloque PL/SQL para recorrer la tabla de triángulos:

```

declare
    t tipo_triangulo;
begin
    for i in (select * from triangulos) loop
        t:=i.triangulo; --t contiene el objeto triángulo
        dbms_output.put_line('El triángulo con id: '||i.idtriangulo);
        dbms_output.put_line('Con base: '||t.base);
        dbms_output.put_line('Y altura: '||t.altura);
        dbms_output.put_line('Tiene un área de: '||t.area);
        dbms_output.put_line('-----');
    end loop;
end;
/

```

t.base equivale a i.triangulo.base  
t.altura equivale a i.triangulo.altura  
t.area equivale a i.triangulo.area

### Actividad. Métodos

- a) Crea un tipo objeto llamado tipo\_ciudadano que contenga los siguientes atributos y métodos:

Atributos	Métodos
<ul style="list-style-type: none"> <li>▪ Nif: char(9)</li> <li>▪ Nombre: varchar2(30)</li> <li>▪ Apellidos: varchar2(60)</li> <li>▪ Email varchar2(120)</li> </ul>	<ul style="list-style-type: none"> <li>▪ Función “nombrecompleto”, devuelve un varchar2</li> </ul>

- b) Crea el cuerpo de la función “nombrecompleto”: devuelve el nombre completo del ciudadano (nombre+apellidos).

- c) Crea una tabla llamada CIUDADANOS con los siguientes campos:

- Idciudadano number(2). Es la clave primaria.
- Ciudadano objeto de tipo “tipo\_ciudadano”. No puede ser nulo.
- Ciudad varchar2(60)

- d) Inserta en la tabla los siguientes datos:

idciudadano	Ciudadano	Ciudad
1	NIF: 11111111N Nombre: Paco Apellidos: Pérez Agudo Email: pp@mail.com	Coslada
2	NIF: 22222222N Nombre: María Apellidos: Mesa Martí Email: mm@mail.com	San Fernando de Henares

- e) Realiza un SELECT a la tabla CIUDADANOS de forma que el resultado sea el siguiente. Ten en cuenta que para la tercera columna se debe llamar al método del objeto:

IDCIUDADANO	NIF	NOMBRE COMPLETO	CIUDAD
1	11111111N	Paco Pérez Agudo	Coslada
2	22222222N	María Mesa Martí	San Fernando de Henares

### 3.4. 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 super tipos de objetos creados previamente.**

- **Supertipo de objeto:** se declaran con la cláusula **NOT FINAL**.

```
CREATE [OR REPLACE] TYPE nombreSuperTipo AS OBJECT(
    ...
) NOT FINAL;
```

- **Subtipo de objeto:** hay que indicar de qué supertipo descende con la cláusula **UNDER**.

```
CREATE [OR REPLACE] TYPE nombreSubTipo
UNDER nombreSuperTipo (
    ...
);
```

No se indica AS OBJECT porque se sobreentiende

Hereda de nombreSuperTipo

**Ejemplo:** En una universidad hay distintos edificios (rectorado, escuelas, facultades, oficinas, residencias...). Todos ellos tienen características comunes de un edificio (dirección, metros útiles, aforo, año de construcción...). Por tanto, se puede construir un supertipo de objeto llamado tipo\_edificio que aglutine los atributos característicos de un edificio.

```
create or replace type tipo_edificio as object
(dirección varchar2(50),
 año_construccion number(4)) not final;
/
```

```
create or replace type tipo_escuela
under tipo_edificio
(grado_impartido varchar2(30),
 numero_aulas number);
/
```

Creación de un tipo de edificio específico, la escuela universitaria.

DESCRIBE tipo\_escuela;

Nombre	¿Nulo?	Tipo
DIRECCIÓN		VARCHAR2(50)
AÑO_CONSTRUCCION		NUMBER(4)
GRADO_IMPARTIDO		VARCHAR2(30)
NUMERO_AULAS		NUMBER

```
create or replace type tipo_residencia
under tipo_edificio
(tipo_comedor varchar2(30),
 num_dormitorios number(3));
/
```

Creación de un tipo de edificio específico, la residencia universitaria.

DESCRIBE tipo\_residencia;

Nombre	¿Nulo?	Tipo
DIRECCIÓN		VARCHAR2(50)
AÑO_CONSTRUCCION		NUMBER(4)
TIPO_COMEDOR		VARCHAR2(30)
NUM_DORMITORIOS		NUMBER(3)

## Actividad. Herencia

Para gestionar el personal de los institutos de una comunidad se va a utilizar una base de datos orientada a objetos.

a) Crea un tipo de objeto llamado `tipo_persona` que contenga los siguientes campos o atributos

- Nif: char(9)
- Nombre: varchar2(30)
- Apellidos: varchar2(60)
- Email varchar2(120)

Este tipo de objeto **deberá permitir crear subtipos** que desciendan de él.

b) Crea un tipo de objeto llamado `tipo_trabajador` que descienda de `tipo_persona` y que contenga los siguientes campos o atributos:

- Instituto: varchar2(30)
- puesto: varchar2(10)

Este tipo de objeto **deberá permitir crear subtipos** que desciendan de él.

c) Crea un tipo de objeto llamado `tipo_profesor` que descienda de `tipo_trabajador` y que contenga los siguientes campos o atributos:

- cuerpo varchar2(30)
- titulación varchar2(30)

d) Crea dos tablas:

CONSERJES de objetos de tipo `tipo_trabajador` (solo tendrá una columna)


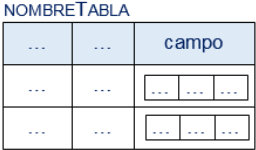

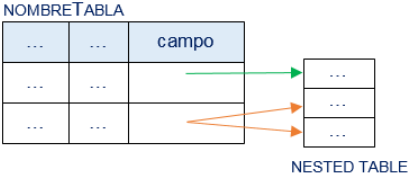



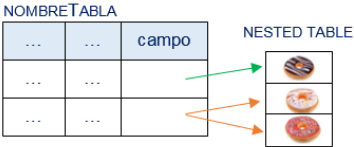
PROFESORES de objetos de tipo `tipo_profesor` (solo tendrá una columna)

Una vez creadas, haz un DESCRIBE de las tablas.

e) Inserta los siguientes datos:

Profesora	Conserje
NIF: 00000000A Nombre: Mar Apellidos: Borondo Jai Email: mm@iesluisbraille.es Instituto: IES Luis Braille Puesto: fijo Cuerpo: Informática Titulación: Ingeniería informática	NIF: 33333333A Nombre: Luis Apellidos: Rivera Moya Email: rm@iesluisbraille.es Instituto: IES Luis Braille Puesto: temporal

## 4. Anexo: Resumen de tipos de datos

CREACIÓN DEL TIPO DE DATOS	UTILIZACIÓN EN TABLA	MANIPULACIÓN DE DATOS
<p><b>Array o colección:</b> lista de tamaño fijo</p>  <pre>create [or replace] type tipoArray as varray(numElementos) of tipoDato; /</pre>	<pre>create table miTabla( ... campo tipoArray);  insert into miTabla values (... , tipoArray(valor1,valor2,...));</pre> 	<p>No es posible trabajar desde SQL con valores individuales del Array</p> <pre>update miTabla set campo= tipoArray(valor1,valor2,...) where ...;</pre>
<p><b>Tabla anidada (nested table):</b> lista de tamaño indefinido</p>  <pre>create [or replace] type tipoTabla as table of tipoDato; /</pre>	<pre>create table miTabla( ..., campo tipoTabla )nested table campo store as t_almacen;  insert into miTabla values (... , tipoTabla(valor1,valor2,...));</pre> 	<p>No es posible trabajar desde SQL con valores individuales de la tabla anidada</p> <pre>update miTabla set campo= tipoTabla(valor1,valor2,...) where ...;</pre>
<p><b>Objeto</b></p>  <p>(Contiene atributos, métodos y funciones)</p> <pre>create [or replace] type tipoObj as object( campoObj ..., ... ); /</pre>	<p>Tabla con un solo campo de tipo objeto</p> <pre>create table miTabla of tipoObj;  insert into miTabla values (tipoObj(valor1,valor2,...));</pre> 	<p>Se manipulan los datos de la tabla como si los atributos del tipo de objeto fueran directamente atributos de la tabla.</p> <pre>update miTabla set campoObj=valor where ...;</pre>
	<p>Tabla con un campo de tipo objeto</p> <pre>create table miTabla( ..., campo tipoObj );  insert into miTabla values (... , tipoObj(valor1,valor2,...));</pre> 	<p>Para manipular los datos de esta tabla es necesario emplear un alias de tabla y acceder al dato: "aliasTabla.campo.atributoObjeto"</p> <pre>update miTabla A set A.campo.campoObj=valor where ...;</pre>
	<p>Tabla con tabla anidada de objetos</p> <pre>create or replace type tipoTabla as table of tipoObj; create table miTabla( ..., campo tipoTabla)nested table campo store as t_almacen;  insert into miTabla values (... , tipoTabla( tipoObj(valor1,valor2,...), tipoObj(valor1,valor2,...),...));</pre> 	<p>No es posible trabajar desde SQL con valores individuales de la tabla anidada</p> <pre>update miTabla set campo= tipoTabla(tipo_Obj(v1,v2,...), tipo_Obj(v1,v2,...), ...) where ...;</pre>