Institut Joan d'Àustria

M2.UF4
BASES DE DATOS OBJETO RELACIONALES

Eduard Lara

INDICE

1. CONCEPTO DE OBJETO Y CLASE

- Un objeto (informático) es un elemento que se puede relacionar con un objeto físico y abstracto del mundo real y sobre el cual se puede describir una serie de propiedades y acciones (métodos o funciones).
- Una clase es definición abstracta de objetos. Para crear un objeto es preciso que se haya definido previamente la clase del objeto.
- Se puede entender que una clase es la definición de un determinado elemento. Así, por ejemplo, la clase artículo tiene las propiedades: ancho, largo, alto, color, material... O sea, las características que queremos controlar o definir de un artículo.
- Cuando las características toman valores, decimos que tenemos un objeto. Por ejemplo, si defino que un artículo es de color marrón, tengo un objeto o una instancia de la clase objeto.

1. BASES DE DATOS OBJETO RELACIONAL

- Las bases de datos objeto-relacional o BDOR se basan en la creación y almacenamiento de objetos completos, pudiendo crearse a partir de otros ya existentes.
- Son una evolución del modelo relacional.
- Son híbridas: contienen la tecnología relacional y de orientación a objetos. Podemos usar el lenguaje SQL.
- Se pueden crear también a partir de la transformación de un modelo Entidad-Interrelación
- PostgreSQL y Oracle son bases de datos objeto-relacional
- Pueden suponer mejoras en los costes del desarrollo de una aplicación y pueden mejorar el rendimiento.

1. BASES DE DATOS OBJETO RELACIONAL

- El modelo objeto–relacional es una alternativa al modelo relacional
- Ventajas de las BD objeto-relacional:
    1. Mantenimiento:
        - Los cambios en un objeto se puede aplicar a todos los objetos que se reutilizan.
        - Los datos y su tratamiento están agrupados en la propia clase donde se define.
    2. Reutilización de código -> reutilización de objetos. Objetos pueden contener otros objetos.
    3. Coherencia de la aplicación. Ejemplo formato de un campo.
    4. Simplificación del modelo de bases de datos. Alguna relaciones 1..N se pueden eliminar.

1. DEFINICIÓN DE OBJETOS EN BDOR

- Idea básica: el usuario puede crear sus propios tipos de datos objeto.
- Para definir objetos en una BD objeto-relacional creamos **nuevos tipos de datos "complejos"** (son un conjunto de tipos de datos simples y/ otros tipos complejos):
- Un tipo de dato objeto es un paquete cohesionado de un tipo concreto de dato.
- Ofrece tres propiedades:
  - Encapsulación: ocultación de los detalles de implementación
  - Herencia: reutilización
  - Poliformismo: sobreescritura de métodos.

1. DEFINICIÓN DE OBJETOS EN BDOR

Encapsulacion

- Para el usuario la estructura interna del objeto no es necesaria conocerla
- La estructura interna de un objeto se denomina implementación
- La interfaz del objeto define tanto propiedades como métodos del objeto
- Al encapsular yo consigo:
  - La modificación de la implementación no afecta al usuario
  - Se simplifica el uso del objeto o, dicho de otra forma, disminuye la complejidad del sistema
  - Aumenta la integridad del sistema ya que los errores no se dispersan
  - Permite sustituir un objeto por otro con otra implementación si responde a la misma interfaz

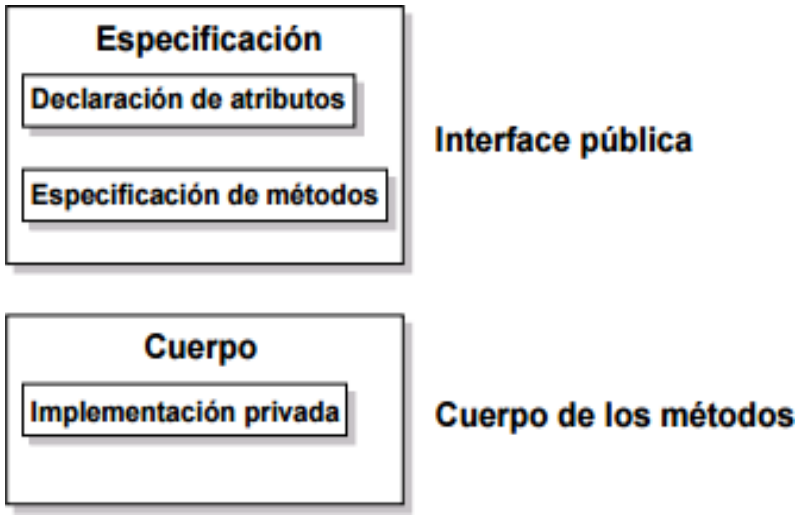
1. DEFINICIÓN DE OBJETOS EN BDOR

Herencia

- Mecanismo por el cual se utiliza la definición de una clase llamada padre para definir un clase llamada hijo
- Ejemplo: si defino la clase ANIMAL que tiene como propiedad tamaño, puedo definir la clase CABALLO como “hija” de ANIMAL y heredará su propiedad tamaño.
- Herencia múltiple: cuando una clase hereda de otras dos.

1. DEFINICIÓN DE OBJETOS EN BDOR

- Un tipo de dato-objeto es un tipo de dato que encapsula sus datos en una interfaz que define las funciones de acceso a esos datos.
- El tipo objeto o clase define tanto los atributos o propiedades como los métodos
- La estructura de un tipo de objeto consta de dos partes:
  - Especificación o “interface pública”: hace referencia a las propiedades y métodos accesibles
  - Cuerpo; donde está la implementación (no accesible) de los métodos definidos en la “interface pública”



1. DEFINICIÓN DE OBJETOS EN BDOR

- Unos ejemplos serían:

```
TipoObjeto: Modelo
        consumo: entero
        potencia: real
        cilindrada: real
        aceleración: real
Final Modelo
```

```
TipoObjeto: Vehículo
        Atributos
                marca:varchar
                modelo_vehículo:Modelo
                color: varchar
                ruedas: entero
        Métodos
                acelerar()
                frenar()
Final Vehículo
```

- En las especificaciones, las declaraciones son públicas y las implementaciones - el cuerpo del tipo Objeto - son privadas

1. DEFINICIÓN DE OBJETOS EN BDOR

- En una especificación de un tipo de objeto siempre debe constar por lo menos un atributo. Los métodos son opcionales.
- Los atributos se declaran indicando nombre y tipo.
- Los tipos básicos de un atributo pueden ser: lógicos, numéricos, alfanuméricos, fechas, horas y tipos objeto previamente definidos.

```
TipoObjeto: DirecciónPostal
        Atributos
                calle: varchar
                ciudad: varchar
                codigo: varchar
Final
```

```
TipoObjeto: Cliente
        Atributos
                numeroCli: entero
                nombreCli: varchar
                DirecciónPostal: tipo DirecciónPostal
        Métodos
                NumeroCliente() → entero
                DatosCliente() → varchar
```

2. CREACIÓN DE TIPOS EN POSTGRESQL

- Postgres permite la creación de tipos definidos por el usuario
- El nombre de estos tipos no puede coincidir con ninguno de los nombres de los tipos ya definidos por el sistema.
- Postgres soporta dos tipos de datos definidos por el usuario:
    - Tipo compuesto: será utilizado para la definición de columnas en tablas
    - Pseudo-tipo: será utilizado como tipo devuelto por una función definida por el usuario.

2. CREACIÓN DE TIPOS EN POSTGRESQL

- El tipo compuesto es un tipo de dato definido por el usuario. Solo puede ser utilizado en la base de datos que se esté utilizando
- Es, fundamentalmente, una lista de campos con sus tipos de datos.
- Se define utilizando una instrucción SQL, muy parecida a la definición de una tabla, con una diferencia, no se pueden indicar restricciones de ningún tipo:

```
CREATE TYPE <nombredelObjeto> AS (
-- definición de las propiedades del objeto.
)
```

- Para borrar un objeto utilizamos la instrucción

```
DROP TYPE nombredelObjeto [CASCADE]
```

- Con la opción CASCADE eliminamos el objeto aunque tenga dependencias.

2. CREACIÓN DE TIPOS EN POSTGRESQL

- Dispondrá de un identificador. En postgres a estos identificadores se les llama OID
- Dispondrá de atributos o propiedades. Modelizan la estructura del tipo. Pueden ser atributos básicos que se refieren a valores de tipo estándar o atributos definidos por el usuario.
- Dispondrá de unos métodos: funciones o procedimientos escritos en PL/PgSQL o escritos con cualquier otro lenguaje y almacenados externamente
- Todo objeto tendrá una especificación en la que se indica que interfaz de acceso dispone el objeto

2. CREACIÓN DE TIPOS EN POSTGRESQL

- Postgres puede generar identificadores de objeto de forma automática. Para ello debemos incluir la cláusula 'WITH OIDS' entre las restricciones de tabla.
- Con la cláusula 'WITH OIDS' se generan dos identificadores: uno de fila (OID) y otro de tabla (TABLEOID). Normalmente estos campos permanecerán ocultos cuando hagamos una consulta. Para visualizarlos debemos incluir el nombre de los campos explícitamente en la consulta.

```
SELECT OID, TABLEOID, * FROM <NombreTabla>;
```

- Estos campos nos serán muy útiles a la hora de identificar nuestros objetos.

2. CREACIÓN DE TIPOS EN POSTGRESQL

Ejemplo Creación del objeto objproducto

1) Tiene las siguientes propiedades:

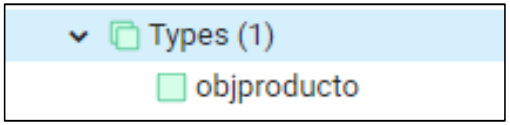
- código, nombre, familia, color, de tipo VARCHAR
- Altura, anchura, longitud, peso, precioCompra, precioVenta de tipo FLOAT
- Fecha creación de tipo DATE

2) Creamos el tipo con la instrucción CREATE TYPE

3) A continuación lo compilamos con el botón de play

Query Editor | Query History

```
1  CREATE TYPE objproducto AS (
2      codigo VARCHAR,
3      nombre VARCHAR,
4      familia VARCHAR,
5      color VARCHAR,
6      altura FLOAT,
7      anchura FLOAT,
8      longitud FLOAT,
9      peso FLOAT,
10     precioCompra FLOAT,
11     precioVenta FLOAT,
12     FechaCrea DATE
13 );
```



3. CREACIÓN DE TABLAS DE OBJETOS

- Una tabla de objetos es una tabla especial que permite almacenar objetos como si fuesen filas. Existen dos tipos de tablas:
  - **Las tablas con identidad de objeto** se utilizan como si de una tabla normal se tratara dónde cada una de sus filas es una fila del objeto. Cada campo de la fila corresponde a un atributo del objeto.
  - **Las tablas sin identidad de objeto** tienen filas que provienen de la definición del objeto y filas definidas con tipos normales de SQL (number, varchar,…)
- Se pueden agregar restricciones de columna a dichos campos-atributos y restricciones a la tabla.
- Sobre un objeto no se pueden realizar acciones de insert, update, etc. Para poder trabajar con objetos, es imprescindible crear una tabla.

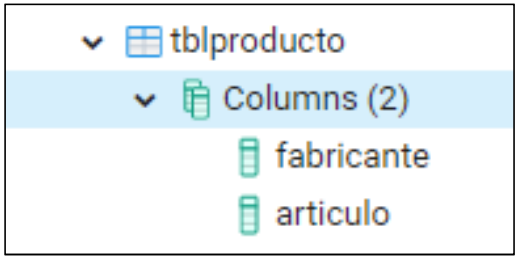
3. TABLA SIN IDENTIDAD DE OBJETOS

Creacion Tabla sin identidad de objeto

- Tienen filas que provienen de la definición del objeto y filas definidas con tipos normales de SQL (number, varchar,…)

- Ejemplo: creamos la tabla TBLProducto que almacena la clase objproducto y el fabricante del mismo.

```
CREATE TABLE tblproducto(
    fabricante varchar,
    articulo objproducto
);
```



3. TABLA SIN IDENTIDAD DE OBJETOS

Inserción de objetos

- En el siguiente ejemplo se observa como se insertan objetos en la tabla, el articulo lo añadimos con ROW delante:

```
INSERT INTO TBLPRODUCTO VALUES('LEROY MERLIN', ROW('p1','recambio cafetera','cafeteras','negro',
            25,8,5,250,25,45, TO_DATE('20201020','YYYYMMDD')));
INSERT INTO TBLPRODUCTO VALUES('LEROY MERLIN', ROW('p2','plato sopa','vajillas','blanco',
            4.3,15,5,158.2,26,10, TO_DATE('20161219','YYYYMMDD')));
INSERT INTO TBLPRODUCTO VALUES('IKEA', ROW('p3','tapa sarten','utensilios','transparente',
            30.2,25.2,5,100,5,15.99, TO_DATE('20171120','YYYYMMDD')));
```

3. TABLA SIN IDENTIDAD DE OBJETOS

Consulta de objetos

- Para acceder a los datos de objetos almacenados en tablas, utilizamos la sentencia SELECT. Para acceder a los datos del objeto utilizamos el nombre de la tabla y el nombre del objeto.

```
select * from tblproducto;
select fabricante, articulo from tblproducto;
select t.fabricante, t.articulo from tblproducto t;
```

Data Output | Explain | Messages | Notifications

	fabricante character varying	articulo objproducto
1	LEROY MERLIN	(p1,"recambio cafetera",cafeteras,negro,25,8,5,250,25,45,2020-10-20)
2	LEROY MERLIN	(p2,"plato sopa",vajillas,blanco,4.3,15,5,158.2,26,10,2016-12-19)
3	IKEA	(p3,"tapa sarten",utensilios,transparente,30.2,25.2,5,100,5,15.99,2017-11-20)

3. TABLA SIN IDENTIDAD DE OBJETOS

Consulta de objetos

- Para acceder a las propiedades de un objeto desde una tabla utilizamos la siguiente sintaxis:

  (nombretabla.objeto).propiedad

```
32  select (articulo).codigo, (articulo).nombre from tblproducto;
33  select (t.articulo).codigo, (t.articulo).nombre from tblproducto t;
34
```

Data Output | Explain | Messages | Notifications

	codigo character varying	nombre character varying
1	p1	recambio cafetera
2	p2	plato sopa
3	p3	tapa sarten

3. TABLA SIN IDENTIDAD DE OBJETOS

Consulta de objetos

- Podemos crear campos calculados a partir de las propiedades del objeto. Por ejemplo el volumen del articulo.

```
35  select (t.articulo).anchura, (t.articulo).longitud,
36         (t.articulo).anchura*(t.articulo).longitud as producto
37  from tblproducto t;
38
39
```

Data Output | Explain | Messages | Notifications

	anchura double precision	longitud double precision	producto double precision
1	8	5	40
2	15	5	75
3	25.2	5	126

3. TABLA SIN IDENTIDAD DE OBJETOS

Institut Joan d'Àustria

Modificación de datos

- La modificación de datos se realiza mediante la instrucción UPDATE. Para referenciar el campo que queremos modificar utilizamos el mismo mecanismo de las consultas.

Ejemplo

- Actualizamos a “Fab. propia”, el nombre del fabricante de aquellos artículos que actualmente son del fabricante ‘IKEA’ y reducimos en un 10% su precio de venta siempre y cuando éste no sea inferior o igual al precio de compra.

```
update tblproducto set fabricante='Fab.Propia',
                       articulo.precioVenta = (articulo).precioVenta * 0.9
where fabricante ='IKEA' AND
      (articulo).precioVenta*0.9>(articulo).precioCompra;
```

3. TABLA SIN IDENTIDAD DE OBJETOS

Modificación de objetos

Query Editor | Query History

```
1
2  select fabricante, (articulo).precioVenta, (articulo).precioVenta
3  from tblproducto
4
5
```

Data Output | Explain | Messages | Notifications

	fabricante character varying	precioventa double precision	precioventa double precision
1	LEROY MERLIN	45	45
2	LEROY MERLIN	10	10
3	IKEA	15.99	15.99

Query Editor | Query History

```
1  update tblproducto set fabricante='Fab.Propia',
2                         articulo.precioVenta = (articulo).precioVenta * 0.9
3  where fabricante ='IKEA' AND
4        (articulo).precioVenta*0.9>(articulo).precioCompra;
5
6  select fabricante, (t.articulo).precioVenta, (articulo).precioVenta
7  from tblproducto t
8
```

Data Output | Explain | Messages | Notifications

	fabricante character varying	precioventa double precision	precioventa double precision
1	LEROY MERLIN	45	45
2	LEROY MERLIN	10	10
3	Fab.Propia	14.391	14.391

3. TABLA SIN IDENTIDAD DE OBJETOS

Borrado de datos

- La modificación de datos se realiza mediante la instrucción DELETE. En la cláusula WHERE referenciamos los campos de los objetos de la misma forma que hemos hecho en las consultas y las modificaciones.

```
10  SELECT  * FROM TBLPRODUCTO;
11
```

Data Output | Explain | Messages | Notifications

	fabricante character varying	articulo objproducto
1	LEROY MERLIN	(p1,"recambio cafetera",cafeteras,negro,25,8,5,250,25,45,2020-10-20)
2	LEROY MERLIN	(p2,"plato sopa",vajillas,blanco,4.3,15,5,158.2,26,10,2016-12-19)
3	Fab.Propia	(p3,"tapa sarten",utensilios,transparente,30.2,25.2,5,100,5,14.391,2017-11-20)

```
12  DELETE FROM TBLPRODUCTO
13  WHERE (articulo).color='blanco';
14
15  SELECT  * FROM TBLPRODUCTO;
```

Data Output | Explain | Messages | Notifications

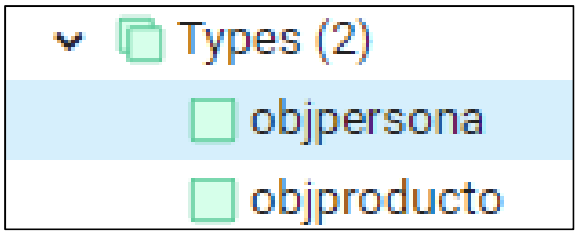
	fabricante character varying	articulo objproducto
1	LEROY MERLIN	(p1,"recambio cafetera",cafeteras,negro,25,8,5,250,25,45,2020-10-20)
2	Fab.Propia	(p3,"tapa sarten",utensilios,transparente,30.2,25.2,5,100,5,14.391,2017-11-20)

3. TABLA CON IDENTIDAD DE OBJETOS

Tabla con identidad de objeto

- Se utilizan como si de una tabla normal se tratara dónde cada una de sus filas es una fila del objeto.
- Ejemplo: Creamos un objeto persona

```
create type objpersona as(
    idpersona int,
    dni varchar,
    nombre varchar,
    apellidos varchar,
    fecha_nac date
)
```



3. TABLA CON IDENTIDAD DE OBJETOS

Tabla con identidad de objeto

- Para crear la tabla utilizamos la siguiente sintaxis:

CREATE TABLE <nombretabla> OF <tipoobjeto> (

campo PRIMARY KEY;
{ Column_Name WITH OPTIONS <restricciones de columna>
[...]<restricciones de tabla>} [, …])

)

- Ejemplo: Creamos una tabla de socios.

```
create table socios of objpersona(
    idpersona primary key
);
```

3. TABLA CON IDENTIDAD DE OBJETOS

Institut Joan d'Àustria

Tabla con identidad de objeto

- Las operaciones en estas tablas tienen la misma sintaxis que en las tablas relacionales.
- Cada instancia de objeto se almacena como un registro y por tanto accedemos de la misma forma que lo hacíamos en SQL.

```
30  INSERT INTO SOCIOS VALUES (1,'3455656','RAUL','GOMEZ','11/11/1975');
31  INSERT INTO SOCIOS VALUES (2,'4555676','FELIPE','ALVAREZ','10/10/1985');
32
33  select * from socios;
34
```

Data Output | Explain | Messages | Notifications

	idpersona [PK] integer	dni character varying	nombre character varying	apellidos character varying	fecha_nac date
1	1	3455656	RAUL	GOMEZ	1975-11-11
2	2	4555676	FELIPE	ALVAREZ	1985-10-10

3. TABLA CON IDENTIDAD DE OBJETOS

Ejemplo

```
CREATE TYPE Producto AS (
                Nombre  varchar(40),
                Id_Proveedor entero,
                Precio numeric);
```

```
CREATE TABLE ProductosTienda OF Producto (
                Precio WITH OPTIONS DEFAULT 1.0,
                Primary key (Nombre));
```

- La tabla ProductosTienda está asociada al tipo Producto.
- Si queremos eliminar el tipo Producto, no se podrá puesto que dicho tipo se usa en una tabla. Si queremos eliminar el tipo y la tabla asociada tendremos que utilizar la instrucción DROP TYPE … CASCADE;
- Estas tablas no contendrán campos del tipo SERIAL (autoincremento) ya que no pueden ser utilizados en la definición de tipos compuestos.

4. HERENCIA DE TABLAS

- En Postgres una tabla puede heredar atributos y métodos de otra.
- Se observa como la tabla estudiante o tabla hija, “hereda” atributos de persona o tabla padre.
- Cada registro de la tabla estudiante contendrá cinco campos: sus tres propios campos y los dos campos de la tabla persona.
- Es una forma de implementar las especializaciones y generalizaciones que vimos en el modelo ER.

```
CREATE TABLE tblpersona (
    Nombre    varchar(20),
    direccion varchar(100)
);
```

```
CREATE TABLE tblestudiante (
    carrera varchar(20),
    grupo char(5),
    grado integer
) INHERITS (tblpersona);
```

4. HERENCIA DE TABLAS

- Si insertamos en la tabla estudiante:

```
insert into tblestudiante values('edu lara','c/joandaustria','FP','DAM2',2);
```

- Debemos indicar los valores para los campos de la tabla persona. De hecho, si consultamos la tabla persona, observamos:

```
select * from tblestudiante;
```

a Output | Explain | Messages | Notifications

nombre character varying (20)	direccion character varying (100)	carrera character varying (20)	grupo character (5)	grado integer
edu lara	c/joandaustria	FP	DAM2	2

```
select * from tblpersona;
```

ta Output | Explain | Messages | Notifications

nombre [PK] character varying (20)	direccion character varying (100)
edu lara	c/joandaustria

- Es decir, ambas tablas están conceptual y lógicamente unidas. Así, si borramos un registro de una de las tablas, se borrará automáticamente la fila.

4. HERENCIA DE TABLAS

Teníamos (problema anterior) la tabla CLIENTE (Num_Cliente, Nombre_Cliente, Calle, Ciudad, Código_Postal, Tlf) y habíamos creado el tipo de datos compuesto CLIENTE. También habíamos creado la tabla de objetos CLIENTES.

- Ahora se pide almacenar es: CLIENTE_VIP (Descuento, PuntosAcomulados). Se pide:
  - Crear una tabla CLIENTES_VIP que hereda de la tabla CLIENTES
  - Insertar el siguiente registro en CLIENTES_VIP: Cliente Paco (110,'Paco','Tortosa 10','45098','908765678') con un descuento de 10% y dispone de 54 puntos acomulados.
  - ¿Qué ha sucedido en la tabla CLIENTES?

4. HERENCIA DE TABLAS

Definimos la tabla de objetos:

-- Queremos almacenar la siguiente información:

-- CLIENTE(Num_Cliente, Nombre_Cliente, Calle, Ciudad, Código_Postal, Tlf)

-- PEDIDO(Num_Pedido, Calle_envío, Ciudad_envío, CodigoPostal_envío, Num_Cliente)

-- DONDE {Num_Cliente} REFERENCIA A CLIENTE.

-- ISe pide:

-- Crear un tipo de datos compuesto para almacenar los datos de envío de un pedido (Calle_envío, Ciudad_envío y CodigoPostal_envío)

-- Crear un tipo de datos compuesto para almacenar todos los datos de un cliente

-- Crear una tabla de objetos que almacene objetos de tipo cliente indicando que el campo Num_Cliente será la clave primaria de la tabla

-- Crear la tabla PEDIDO. Utilizar el tipo de datos creado anteriormente para almacenar la dirección del pedido.

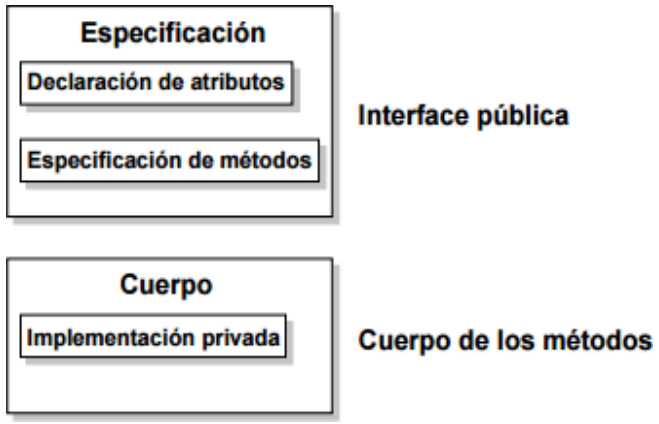
5. MÉTODOS

- Un método es una función o un procedimiento contenido en un objeto que permite realizar una operación sobre él por ejemplo un cálculo.
- Los métodos sirven para obtener información o realizar acciones sobre un objeto o su información.

- Ejemplos:
  - un método para validar los datos del objeto, antes de ser introducidos en la tabla
  - una función que devuelve el resultado de una operación, por ejemplo el volumen de un objeto de tipo objproducto

5. MÉTODOS

- Los métodos constan de especificación y de cuerpo.
- Para crear un método asociado a un objeto hay que declararlo en la definición o especificación del tipo y realizar su implementación en el cuerpo.
  - Especificación: nombre del método y listado de parámetros opcionales o no del método. Si devuelve un valor se indica de qué tipo es ese valor devuelto.
  - Cuerpo: contiene el código que se ejecuta al llamar al método.



5. MÉTODOS

- El nombre de un método no puede coincidir con el nombre del tipo Objeto ni con ninguno de sus atributos.
- Puede existir sobrecarga de métodos. Para ello, los dos (o los que sean) métodos deben de llamarse igual.
- Importante: En postgreSQL hay algunas limitaciones.
  - Para la sobrecarga de métodos
  - Para incluir métodos en las especificaciones

5. MÉTODOS

- En este ejemplo vemos como definimos cada una de las partes:
- Definimos y compilamos la “interfaz pública. Utilizamos la palabra “MEMBER” para definir un método “normal” de un objeto.

ORACLE

```
-- volver a crear el tipo
CREATE OR REPLACE TYPE  "OBJPRODUCTO" AS OBJECT
(
  codigo varchar2(20), nombre varchar2(60), familia varchar2(60), color varchar2(20),
  altura float, anchura float, longitud float, peso float, precioCompra float, precioVenta float,
  fechaCr date,
  MEMBER FUNCTION VOLUMEN RETURN NUMBER,
  MEMBER FUNCTION BENEFICIO RETURN NUMBER
);
```

POSTGRESQL

```
CREATE TYPE OBJPRODUCTO AS (
    codigo varchar(20),
    nombre varchar(60),
    familia varchar(60),
    color varchar(20),
    altura float,
    anchura float,
    longitud float,
    peso float,
    precioCompra float,
    precioVenta float,
    fechaCr date
);
```

5. MÉTODOS

- Ahora definimos el cuerpo del body **los métodos** con la implementación de las funciones o procedimientos.

ORACLE

```
-- crear el cuerpo del tipo
CREATE OR REPLACE TYPE BODY  "OBJPRODUCTO" AS
  MEMBER FUNCTION VOLUMEN RETURN NUMBER IS
      BEGIN
        RETURN altura*anchura*longitud;
      END;

  MEMBER FUNCTION BENEFICIO RETURN NUMBER IS
      BEGIN
        RETURN precioVenta-precioCompra;
      END;
  END;
/
```

POSTGRESQL

```
CREATE OR REPLACE FUNCTION BENEFICIO(i float,j float) RETURNS float AS
$$
    BEGIN
        RETURN i-j;
    END;
$$
LANGUAGE plpgsql;
```

```
CREATE or replace FUNCTION VOLUMEN(longitud float , anchura float , altura float) RETURNS float AS
$$
      BEGIN
        RETURN longitud*anchura*altura;
      END;
$$
LANGUAGE plpgsql;
```

5. MÉTODOS

Utilización de los métodos

- Ahora ya podemos ejecutar la función volumen y beneficio sobre un objeto de tipo “objproducto”.

```
-- Métodos
-- Ejecución de las funciones volumen y beneficio.
select t.articulo.nombre, t.articulo.volumen() as volumen , t.articulo.precioCompra, t.articulo.precioVenta,
t.articulo.beneficio() as beneficio
from tblproducto t;
```

ORACLE

	ARTICULO.NOMBRE	VOLUMEN	ARTICULO.PRECIOCOMPRA	ARTICULO.PRECIOVENTA	BENEFICIO
1	recambio cafetera	1000	25	45	20
2	plato sopa	322,5	6	10	4
3	tapa sartÃ©n	3805,2	5	15,99	10,99

```
132  select   BENEFICIO((t.articulo).precioVenta, (t.articulo).precioCompra) as beneficio,
133           VOLUMEN((articulo).longitud, (articulo).altura, (articulo).anchura) as volumen
134  from tblproducto t;
135
136
```

Data Output | Explain | Messages | Notifications

	beneficio double precision	volumen double precision
1	20	1000
2	-16	322.5
3	10.99	3805.2

POSTGRESQL

5. MÉTODOS

- Ejemplo con los socios.

ORACLE

```
create or replace type obj_persona as object(
   idpersona number,
   dni varchar2(9),
   nombre varchar2(15),
   apellidos varchar2(30),
   fecha_nac date,
   MEMBER FUNCTION edad RETURN NUMBER
);
/
```

```
-- crear el cuerpo del tipo
CREATE OR REPLACE TYPE BODY obj_persona AS
  MEMBER FUNCTION edad RETURN NUMBER IS
      BEGIN
        RETURN to_char(sysdate, 'YYYY')-to_char(fecha_nac, 'YYYY');
      END;
END;
/
```

POSTGRESQL

```
create type objpersona as (
   idpersona int,
   dni varchar(9),
   nombre varchar(15),
   apellidos varchar(30),
   fecha_nac date
);
```

```
CREATE OR REPLACE FUNCTION EDAD(fecha date) RETURNS int AS
$$
    BEGIN
        RETURN DATE_PART('year', current_date) - DATE_PART('year', fecha);
    END;
$$
LANGUAGE plpgsql;

select edad(to_date('10102010','DDMMYYYY'))
```

5. MÉTODOS

- Ahora ya podemos ejecutar la función edad sobre el objeto obj_persona

ORACLE

```
drop table socios;
create table socios of obj_persona(
  idpersona primary key
);

insert into socios values( 1, '123456789', 'raul', 'gomez', '19/12/1975')
insert into socios values( 2, '123456790', 'ana', 'mas', '10/01/1982');

select nombre, apellidos, fecha_nac, s.edad() as edad from socios s;
```

	NOMBRE	APELLIDOS	FECHA_NAC	EDAD
1	raul	gomez	19/12/75	41
2	ana	mas	10/01/82	34

POSTGRESQL

```
35 create table socios of objpersona(
36   idpersona primary key
37 );
38
39 insert into socios values( 1, '123456789', 'raul', 'gomez', '19/12/1975');
40 insert into socios values( 2, '123456790', 'ana', 'mas', '10/01/1982');
41
42 select nombre, apellidos, fecha_nac, edad(fecha_nac)
43 from socios;
44
```

Data Output | Explain | Messages | Notifications

	nombre character varying (15)	apellidos character varying (30)	fecha_nac date	edad integer
1	raul	gomez	1975-12-19	45
2	ana	mas	1982-01-10	38

6. COLECCIONES

- Las colecciones de elementos permiten implementar relaciones 1:N
- Tenemos dos sistemas:
  - ARRAYS
  - NESTED TABLES (TABLAS ANIDADAS)

- ARRAY es un tipo especial que permite definir una colección de objetos.
- A partir de un tipo de objeto ya existente o de un tipo básico, podemos definir un nuevo tipo que sea una colección de los mismos

6. COLECCIONES DE TIPO ARRAY

- Oracle y Postgres permiten almacenar un array de datos en una columna. Es decir, permite tablas multidimensionales
- Declaración:

```
CREATE TABLE prueba (
    id_prueba serial,
    campo_prueba integer [ ]
);
```

POSTGRESQL

- En este caso el campo_prueba albergará vectores de números enteros.
- Una sintaxis alternativa a la hora de definir el campo, más acorde con el estándar SQL es:

```
campo_prueba integer ARRAY
```

- Y si queremos indicar más de una dimensión:

```
campo_prueba integer ARRAY [ ] [ ]
```

6. COLECCIONES DE TIPO ARRAY

- Para insertar en la tabla anterior debemos utilizar las llaves y las comillas simples:

```
INSERT INTO prueba VALUES (DEFAULT, '{1,2,3,4,5}');
INSERT INTO prueba VALUES (DEFAULT, '{4,3,1}');
INSERT INTO prueba VALUES (DEFAULT, '{8,6,3,6}');
```

POSTGRESQL

- Para acceder, la segunda consulta nos devolverá el cuarto elemento de todos los arrays almacenados en la tabla prueba.

```
select * from prueba
```

a Output | Explain | Messages

id_prueba integer	campo_prueba integer[]
1	{1,2,3,4,5}
2	{4,3,1}
3	{8,6,3,6}

```
SELECT id_prueba, campo_prueba[4] FROM prueba;
```

ta Output | Explain | Messages | Notifications

id_prueba integer	campo_prueba integer
1	4
2	[null]
3	6

6. COLECCIONES DE TIPO ARRAY

- Postgres suministra multitud de operadores (aparte de los comunes) para trabajar con arrays:
  - 'Array1 @> Array2' → Devolverá cierto si el Array1 contiene al Array2
  - 'Array1 && Array2' → Devolverá cierto si ambos arrays disponen de elementos en común
  - 'Array1 || Array2' → Concatena arrays
  - …
- También suministra un buen número de funciones que trabajan con arrays:
  - 'array_append(Array, Elemento) ' → Para añadir elementos al final del array
  - 'array_ndims (Array)' → Devuelve número de dimensiones del array
  - 'array_prepend (Array, Elemento)' → Añade elemento al inicio del array

  …

6. COLECCIONES DE TIPO ARRAY

- Utilizamos la sentencia VARRAY para declarar esta colección de objetos. En los VARRAY definimos un número exacto de objetos
- Ejemplo: Queremos almacenar los datos del alumno con una lista de teléfonos. Definimos el objeto teléfono y el objeto lista de teléfonos.

ORACLE

```
-- Colecciones VARRAYS
CREATE OR REPLACE TYPE OBJ_TELEFONO AS OBJECT
(  contacto varchar2(15),
   numero_tel varchar2(15)
);

-- Colecciones VARRAYS
DROP TYPE LISTA_TEL force;
CREATE OR REPLACE TYPE LISTA_TEL AS VARRAY(5) OF OBJ_TELEFONO;
```

POSTGRESQL

```
CREATE TYPE OBJTELEFONO AS
(  contacto varchar(15),
   numero_tel varchar(15)
);

DROP TYPE LISTA_TELEFONOS CASCADE;
CREATE TYPE LISTA_TELEFONOS AS (
    telefonos objtelefono[5]
);
```

6. COLECCIONES DE TIPO ARRAY

- Declaramos un nuevo tipo OBJ_Alumno con los datos básicos del alumno y la lista de teléfonos.
- Declaramos también dos métodos:
  - GETTELF: Nos devolverá el número de teléfono.
  - GETCONTACTO: Nos devolverá el nombre del contacto y el número de telefono.

ORACLE

```
create or replace TYPE OBJ_ALUMNO AS OBJECT
(  numalumno NUMBER,
   nombre varchar2(15),
   apellidos varchar2(30),
   fecha_nac date,
   telfs LISTA_TEL,
   MEMBER FUNCTION GETTELF(NUM NUMBER) RETURN OBJ_TELEFONO,
   MEMBER FUNCTION GETCONTACTO (NUM NUMBER) RETURN VARCHAR2
);
```

POSTGRESQL

```
CREATE TYPE OBJALUMNO AS  (
    numalumno int,
    nombre varchar(15),
    apellidos varchar(30),
    fecha_nac date,
    telefonos lista_telefonos
);
```

6. COLECCIONES DE TIPO ARRAY

- Modificamos el obj_Alumno y creamos las dos funciones.

ORACLE

```
CREATE OR REPLACE TYPE BODY OBJ_ALUMNO AS
  MEMBER FUNCTION GETTELF(NUM NUMBER) RETURN OBJ_TELEFONO AS
  BEGIN
    IF NUM>telfs.count() THEN RETURN null;
    ELSE RETURN telfs(NUM);
    END IF;
   END GETTELF;
   MEMBER FUNCTION GETCONTACTO(NUM NUMBER) RETURN VARCHAR2 AS
    telefono OBJ_TELEFONO;
  BEGIN
    telefono := GETTELF(NUM);
    IF telefono IS NOT NULL THEN
      RETURN telefono.contacto || ' - ' || telefono.numero_tel;
    ELSE RETURN 'Sin datos';
    END IF;
   END GETCONTACTO;
END;
```

POSTGRESQL

```
CREATE OR REPLACE FUNCTION GETTELF(telefonos objtelefono[], num int) RETURNS OBJTELEFONO AS
$$
    BEGIN
        IF NUM > array_length(telefonos,1) THEN
            RETURN null;
        ELSE
            RETURN telefonos[NUM];
        END IF;
    END;
$$
LANGUAGE PLPGSQL;
```

```
CREATE OR REPLACE FUNCTION GETCONTACTO(telefonos objtelefono[], num int) RETURNS VARCHAR AS
$$
    DECLARE
        telefono OBJTELEFONO;
    BEGIN
        telefono := GETTELF(telefonos,NUM);
        IF telefono IS NOT NULL THEN
            RETURN telefono.contacto || ' - ' || telefono.numero_tel;
        ELSE
            RETURN 'Sin datos';
        END IF;
    END;
$$
LANGUAGE PLPGSQL;
```

6. COLECCIONES DE TIPO ARRAY

- Creamos la tabla tblalumnos con el nuevo tipo creado:

ORACLE

```
drop table tblalumnos;
CREATE TABLE tblalumnos of OBJ_ALUMNO(
  numalumno primary key
);
```

POSTGRESQL

```
CREATE TYPE OBJALUMNO AS  (
    numalumno int,
    nombre varchar(15),
    apellidos varchar(30),
    fecha_nac date,
    telefonos objtelefono ARRAY
);
CREATE TABLE tblalumnos of OBJALUMNO(
  numalumno primary key
);
```

- Insertamos datos de teléfono en la tabla.

```
INSERT INTO tblalumnos VALUES
(1, 'Manuel', 'Gómez Mas', '19/12/2000',
LISTA_TEL(OBJ_TELEFONO('Particular','937661212'),OBJ_TELEFONO('Móvil','637121212')));
INSERT INTO tblalumnos VALUES
(2, 'Laura', 'Paredes Roca', '15/03/2000',
LISTA_TEL(OBJ_TELEFONO('Particular','937953232'),OBJ_TELEFONO('Móvil','635131313'),
OBJ_TELEFONO('Abuelos','937953333')));
INSERT INTO tblalumnos VALUES
(3, 'Javier', 'Fuentes Ibañez', '09/01/1999',
LISTA_TEL(OBJ_TELEFONO('Móvil','676445566')));
```

```
INSERT INTO tblalumnos VALUES (1, 'Manuel', 'Gomez Mas', '19/12/2000',
    ARRAY[('ricky', '666666'),('fernan', '555555')]:: objtelefono[]);
INSERT INTO tblalumnos VALUES (2, 'Laura', 'Paredes Roca', '15/03/2000',
    ARRAY[('Juani','35131313'),('Puji','937953333')]::objtelefono[]);
INSERT INTO tblalumnos VALUES (3, 'Javier', 'Fuentes Ibañez', '09/01/1999',
    array[('Rafa','272445226'),('Jose','476141166'),('Marc','373444566')]::objtelefono[]);
```

6. COLECCIONES DE TIPO ARRAY

- Ejecutamos los métodos GETTELF y GETCONTACTO para obtener los datos especificados.

ORACLE

```
select a.nombre, a.GETTELF(2) from tblalumnos a;
select a.nombre,a.apellidos,a.GETCONTACTO(2) from tblalumnos a;
```

	NOMBRE	A.GETTELF(2)
1	Manuel	[SYSTEM.OBJ_TELEFONO]
2	Laura	[SYSTEM.OBJ_TELEFONO]
3	Javier	(null)

	NOMBRE	APELLIDOS	A.GETCONTACTO(2)
1	Manuel	Gómez Mas	Móvil - 637121212
2	Laura	Paredes Roca	Móvil - 635131313
3	Javier	Fuentes Ibañez	Sin datos

POSTGRESQL

```
select nombre, telefonos , GETTELF(telefonos,2), GETCONTACTO(telefonos,1)
from tblalumnos;
```

Output | Explain | Messages | Notifications

nombre character varying (15)	telefonos objtelefono[]	gettelf objtelefono	getcontacto character varying
Manuel	{"(ricky,666666)","...	(fernan,555555)	ricky - 666666
Laura	{"(Juani,3513131...	(Puji,937953333)	Juani - 35131313
Javier	{"(Rafa,27244522...	(Jose,47614116...	Rafa - 272445226

6. COLECCIONES DE TIPO ARRAY

- Crea una tabla que llamarás “listintelefono” con el campo ‘dirección’ (clave) y con el campo ‘teléfono’. Define este último campo como un array de enteros.
- Efectúa un par de inserciones con varios teléfonos por dirección.
- Muestra todos los primeros teléfonos (posiciones 1 de los array) de los registros que tengas
- Repite el ejercicio anterior pero define ahora el array de teléfonos como varchar (deberás investigar por Internet una notación alternativa a la explicada).
- Crea un tipo de dato compuesto: DIRECCION (Calle, Ciudad, CodigoPostal)
- Crea una tabla que llamarás “Listín direcciones” con el campo ‘Nombre_Cliente’ y con el campo ‘Dirección_Cliente’. Define este último campo como un array de direcciones.
- Realiza varias inserciones en la tabla anterior y consulta para ver dichos resultados.

7. NESTED TABLES

- La utilización de ARRAYs tiene una limitación: es preciso saber el nombre máximo de elementos que puede tener.
- Las NESTED TABLES no tienen esta limitación.
- Permiten representar una relación 1:N dentro de un objeto
- Ejemplo: creamos un tipo OBJADDRESS que se utiliza para almacenar direcciones

ORACLE

```
-- Colecciones NESTED TABLES
-- tipo que almacena direcciones
drop type OBJADDRESS force; -- obligamos a borrar el objeto
CREATE TYPE OBJADDRESS AS OBJECT (
   calle varchar2(20),
   cp varchar2(5),
   poblacion varchar2(20),
   provincia varchar2(20)
);
/
```

POSTGRESQL

```
drop type OBJADDRESS cascade;
CREATE TYPE OBJADDRESS AS (
    calle varchar(20),
    cp varchar(5),
    poblacion varchar(20),
    provincia varchar(20)
);
```

7. NESTED TABLES

- Queremos almacenar clientes. Un determinado cliente puede tener varias direcciones: relación 1 a N.
- Creamos un nuevo tipo que define una tabla de objetos dirección que nos permita asociar varias direcciones a un cliente.
- Se declara como AS TABLE OF (en lugar de AS OBJECT)

ORACLE

```
-- tipo que almacena un conjunto de direcciones
CREATE TYPE LISTADIRECCIONES AS TABLE OF OBJADDRESS;
/
```

POSTGRESQL

7. NESTED TABLES

- Creamos una tabla para almacenar clientes. De cada cliente se guardará su código, su nombre y sus direcciones (almacenadas en un objeto OBJADRESS).

ORACLE

```
-- Tabla que almacena clientes y sus direcciones
drop table tblclientes;
create table tblclientes (
    codigo number,
    nombre varchar(40),
    direccion LISTADIRECCIONES)
    NESTED TABLE direccion store as tblclidir;
```

```
-- insercion de clientes
insert into tblclientes values (1,'Muebles S.A', LISTADIRECCIONES(
    OBJADDRESS ('Calle Mayor 12','17300','Blanes','Girona'),
    OBJADDRESS ('Calle Balmes 1','17310','Lloret de Mar','Girona')));

insert into tblclientes values (2,'Interiorismo SL', LISTADIRECCIONES(
    OBJADDRESS ('Calle Iglesia 4','08370','Calella','Barcelona'),
    OBJADDRESS ('Avda Diagonal 23','08010','Barcelona','Barcelona')));
```

POSTGRESQL

```
create table tblclientes (
    codigo int,
    nombre varchar(40),
    direccion objaddress[]
);

insert into tblclientes values (1,'Muebles S.A',
    ARRAY[('Calle Mayor 12','17300','Blanes','Girona'),
          ('Calle Balmes 1','17310','Lloret de Mar','Girona')]::objaddress[]);
insert into tblclientes values (2,'Interiorismo SL',
    ARRAY[('Calle Iglesia 4','08370','Calella','Barcelona'),
          ('Avda Diagonal 23','08010','Barcelona','Barcelona')]::objaddress[]);
```

7. NESTED TABLES

- A diferencia de VARRAY, el uso de “nested tables” nos permite obtener los elementos en formato tabla. Es necesaria una función especial TABLE que convierte los datos a formato de tabla.

ORACLE

```
-- selects
select t1.codigo, t1.nombre, t2.*
from tblclientes t1, table (t1.direccion) t2;
```

	CODIGO	NOMBRE	CALLE	CP	POBLACION	PROVINCIA
1	1	Muebles S.A	Calle Mayor 12	17300	Blanes	Girona
2	1	Muebles S.A	Calle Balmes 1	17310	Lloret de Mar	Girona
3	2	Interiorismo SL	Calle Iglesia 4	08370	Calella	Barcelona
4	2	Interiorismo SL	Avda Diagonal 23	08010	Barcelona	Barcelona

POSTGRESQL

```
select codigo, nombre, direccion, direccion[2].provincia
from tblclientes;
```

Output | Explain | Messages | Notifications

codigo integer	nombre character varying (40)	direccion objaddress[]	provincia character varying (2
1	Muebles S.A	{"(\"Calle Mayor 12\",17300,Blanes,Giron...	Girona
2	Interiorismo SL	{"(\"Calle Iglesia 4\",08370,Calella,Barcel...	Barcelona

- Podemos tratar toda la información como si fuera una base de datos relacional no orientada a objetos.

```
select t1.codigo, t1.nombre, t2.*
from tblclientes t1, table (t1.direccion) t2
where t2.provincia = 'Barcelona';
```

CODIGO	NOMBRE	CALLE	CP	POBLACION	PROVINCIA
2	Interiorismo SL	Calle Iglesia 4	08370	Calella	Barcelona
2	Interiorismo SL	Avda Diagonal 23	08010	Barcelona	Barcelona

```
select codigo, nombre, direccion[1], direccion[2]
from tblclientes
where direccion[1].provincia = 'Barcelona' or direccion[2].provincia = 'Barcelona';
```

Output | Explain | Messages | Notifications

codigo integer	nombre character varying (40)	direccion objaddress	direccion objaddress
2	Interiorismo SL	("Calle Iglesia 4",08370,Calella,...	("Avda Diagonal 23",08010,Bar...

7. NESTED TABLES

- Para modificar valores de la “nested table” debemos convertirla en formato “tabla” (utilizando la función TABLE) para acceder directamente a sus campos.
- Modificamos una de las direcciones del cliente con código 2.
- El select del update solo puede devolver una fila (importante!!).

ORACLE

```
-- updates
update table (select t.direccion from tblclientes t where t.codigo = 2) t2
set t2.calle='Pio Baroja 12'
where t2.poblacion = 'Calella';
```

CODIGO	NOMBRE	CALLE	CP	POBLACION	PROVINCIA
1	Muebles S.A	Calle Mayor 12	17300	Blanes	Girona
1	Muebles S.A	Calle Balmes 1	17310	Lloret de Mar	Girona
2	Interiorismo SL	Pio Baroja 12	08370	Calella	Barcelona
2	Interiorismo SL	Avda Diagonal 23	08010	Barcelona	Barcelona

POSTGRESQL

```
update tblclientes
set direccion[1].calle='Diagonal 23'
where direccion[1].poblacion = 'Calella';
```

7. NESTED TABLES

- Para eliminar valores de la “nested table” debemos convertirla en formato “tabla” (utilizando la función TABLE).
- Eliminamos la dirección que tiene por población Barcelona del cliente con código 2.

ORACLE

```
-- delete
delete table (select t.direccion from tblclientes t where t.codigo = 2) t2
where t2.poblacion = 'Barcelona';
```

CODIGO	NOMBRE	CALLE	CP	POBLACION	PROVINCIA
1	Muebles S.A	Calle Mayor 12	17300	Blanes	Girona
1	Muebles S.A	Calle Balmes 1	17310	Lloret de Mar	Girona
2	Interiorismo SL	Pio Baroja 12	08370	Calella	Barcelona

POSTGRESQL

```
delete from tblclientes
where direccion[1].poblacion = 'Barcelona';

delete from tblclientes
where direccion[1].poblacion = 'Barcelona' or
      direccion[2].poblacion = 'Barcelona';
```

8. INTEGRIDAD REFERENCIAL

Trabajamos con los siguientes objetos:

- objaddress : almacena una dirección física.
- objpersona : almacena la información de una persona.

Y las siguientes tablas de objetos:

- tbldirecciones: almacena todos los objetos objaddress que asignaremos a personas.
- tblpersonas: almacena todos los objetos objpersona

Queremos limitar los valores de direcciones a los contenidos en la tabla de direcciones (tbldirecciones)

8. INTEGRIDAD REFERENCIAL

- Creamos e insertamos datos en la tabla de direcciones tbldirecciones.

ORACLE

```
-- INTEGRIDAD REFERENCIAL
-- Tabla de objetos dirección
drop type OBJADDRESS force; -- obligamos a borrar el objeto
CREATE TYPE OBJADDRESS AS OBJECT (
   calle varchar2(20),
   cp varchar2(5),
   poblacion varchar2(20),
   provincia varchar2(20)
);
/
-- creamos la tabla tbldirecciones del tipo objaddress
CREATE TABLE tbldirecciones OF objaddress;
```

```
insert into tbldirecciones values (OBJADDRESS ('Calle Mayor 12','17300','Blanes','Girona'));
insert into tbldirecciones values (OBJADDRESS ('Calle Balmes 1','17310','Lloret de Mar','Girona
insert into tbldirecciones values (OBJADDRESS ('Calle Iglesia 4','08370','Calella','Barcelona'
insert into tbldirecciones values ( OBJADDRESS ('Avda Diagonal 23','08010','Barcelona','Barcel
```

POSTGRESQL

```
drop type OBJADDRESS cascade;
CREATE TYPE OBJADDRESS AS (
   calle varchar(20),
   cp varchar(5),
   poblacion varchar(20),
   provincia varchar(20)
);
```

```
CREATE TABLE tbldirecciones (
    direccion objaddress primary key
);
```

```
insert into tbldirecciones
values (('Calle Mayor 12','17300','Blanes','Girona'));
insert into tbldirecciones
values (('Calle Balmes 1','17310','Lloret de Mar','Girona'));
insert into tbldirecciones
values (('Calle Iglesia 4','08370','Calella','Barcelona'));
insert into tbldirecciones
values (('Avda Diagonal 23','08010','Barcelona','Barcelona'));
```

8. INTEGRIDAD REFERENCIAL

- Creamos el tipo objpersona y la tabla de objetos correspondiente. Al definir el objeto con la palabra ref asociamos las direcciones
- Con la palabra “references” limitamos los valores posibles del atributo DOMICILIO a los varlores almacenados en la tabla de direcciones TBLDIRECCIONES.

ORACLE

```
-- tipo objpersona
drop type objpersona force;
create or replace type objpersona as object(
    dni varchar2(9),
    nombre varchar2(60),
    domicilio ref objaddress);
```

POSTGRESQL

```
drop type objpersona cascade;
create type objpersona as(
    dni varchar(9),
    nombre varchar(60),
    domicilio objaddress
);
```

- En el caso de no incluir esta restricción, sería posible eliminar direcciones asociadas a personas. Estos registros reciben el nombre de DANGLING REFERENCES.

```
-- tabla de objetos
create table tblpersona of objpersona (
  primary key (dni), domicilio references tbldirecciones);
```

```
drop table tblpersonas;
create table tblpersonas of objpersona (
    primary key (dni),
    domicilio references tbldirecciones(direccion)
);
```

8. INTEGRIDAD REFERENCIAL

Cláusula: REF y DEREF

- Insertamos registros en la tabla de objetos persona, tblpersonas.

ORACLE

```
-- inserción de registros en tblpersona
insert into tblpersona select '12345678A', 'Javier Saez', REF(p) from tbldirecciones p
where p.calle='Calle Mayor 12';
insert into tblpersona select '22223333B', 'Marta Alvarez', REF(p) from tbldirecciones p
where p.calle='Avda Diagonal 23';
```

POSTGRESQL

```
insert into tblpersonas values ('12345678A', 'Javier Saez',
       ('Calle Iglesia 4','08370','Calella','Barcelona'));
       --Correcta
insert into tblpersonas values ('34245671B', 'Pablo Iglesias',
       ('Calle Balmes 1','17310','Lloret de Mar','Girona'));
       --Correcta
insert into tblpersonas values ('34245671B', 'Pedro Sanchez',
       ('Calle Balmes 1','17310','Lloret de Mar','Girona'));
       --Incorrecta. Violacion primary key de tblpersonas
insert into tblpersonas values ('72245671B', 'Jose Abascal',
       ('Calle Balmes1','17310','Lloret de Mar','Girona'));
       --Incorrecta. Violacion foreign key de tbldirecciones
```

8. INTEGRIDAD REFERENCIAL

Cláusula: REF y DEREF

- Consultamos su contenido. Hemos de acceder a la información almacenada por el enlace, utilizando la instrucción DEREF y el campo al que queremos acceder.

ORACLE

```
select t.dni, t.nombre, deref(domicilio).calle as calle ,deref(domicilio).cp as cp,
deref(domicilio).poblacion as poblacion from tblpersona t;
```

	DNI	NOMBRE	CALLE	CP	POBLACION
1	12345678A	Javier Saez	Calle Mayor 12	17300	Blanes
2	22223333B	Marta Alvarez	Avda Diagonal 23	08010	Barcelona

POSTGRESQL

```
select dni, nombre, (domicilio).calle ,(domicilio).cp, (domicilio).poblacion
from tblpersonas;
```

ta Output | Explain | Messages | Notifications

dni [PK] character varying (9)	nombre character varying (60)	calle character varying (20)	cp character varying (5)	poblacion character varying (20)
12345678A	Javier Saez	Calle Iglesia 4	08370	Calella
34245671B	Pablo Iglesias	Calle Balmes 1	17310	Lloret de Mar

- Si intentamos borrar un registro de la tabla direcciones nos da un error de integridad referencial.

```
-- borrar registro, error de integridad.
delete from tbldirecciones where calle='Avda Diagonal 23';
```

```
Error report -
ORA-02292: s'ha violat la restricció d'integritat (SYSTEM.SYS_C0010271)
```

```
74  delete from tbldirecciones
75  where (direccion).calle='Calle Iglesia 4';
76  --Violacion de integridad referncial
77
```

Data Output | Explain | Messages | Notifications

```
ERROR:  update o delete en «tbldirecciones» viola la llave foránea «tblpersonas_domicilio_fkey» en la tabla «tblpersonas»
DETAIL:  La llave (direccion)=(("Calle Iglesia 4",08370,Calella,Barcelona)) todavía es referida desde la tabla
«tblpersonas».
SQL state: 23503
```

8. INTEGRIDAD REFERENCIAL

Trabajamos con los siguientes objetos:

- autores: almacena autor de un libro
- lectores: de una determinada biblioteca(socios)
- Libros: almacena el nombre y el autor de un libro
- Prestamos: almacena el prestamo de un libro a un lector en una biblioteca.

Y las siguientes tablas de objetos:

- tblautores
- Tbllectores
- Tbllibros
- tblprestamo

Queremos limitar los valores de tblprestamos a los contenidos en las tabla de libros y lectores.

8. INTEGRIDAD REFERENCIAL

- Creamos los diferentes objetos.

ORACLE

```
set serveroutput on;

drop table tblprestamo;
drop table tbllibros;
drop table tbllectores;
drop table tblautores;

create or replace type autores as object(codigoautor number, nombre varchar(50));
create or replace type lectores as object(codigolector number, nombre varchar(40), edad number);
create or replace type libros as object(codigolibro number, nombre varchar(40), autor ref autores);
create or replace type prestamo as object (id number, fechainicio date, fechadevolucion date,
libro ref libros, lector ref lectores);
```

POSTGRESQL

```
drop table tblautores;
drop table tbllectores;
drop table tblprestamo;
drop table tbllibros;
```

```
drop type objautor cascade;
drop type objlector cascade;
drop type objlibro cascade;
drop type objprestamo cascade;
```

```
create type objautor as ( codigoautor int, nombre varchar(50));
create type objlector as (codigolector int, nombre varchar(40), edad int);
create type objlibro as (codigolibro int, nombre varchar(40), autor objautor);
create type objprestamo as (id int, fechainicio date, fechadevolucion date, libro objlibro, lector objlector);
```

- Creamos el tipo libros al definir el objeto con la palabra ref asociamos los autores (limitamos a autores que existan). Lo mismo sucede al crear prestamos con libros y lectores (ref)

8. INTEGRIDAD REFERENCIAL

- Con la palabra “references” limitamos los valores posibles del atributo autor a los varores almacenados en la tabla de direcciones tblautores

ORACLE

```
-- Creamos la tabla tbllibros e insertamos datos (referencia al autor)

create table tbllibros of libros(codigolibro primary key, autor references tblautores);

insert into tbllibros select 1, 'Don Quijote de la Mancha',  ref (a) from  tblautores a
where a.codigoautor=1;
insert into tbllibros select 2, 'Hamlet',  ref (a) from  tblautores a
where a.codigoautor=2;
insert into tbllibros select 3, 'Macbeth',  ref (a) from  tblautores a
where a.codigoautor=2;
insert into tbllibros select 4, 'El nombre de la rosa',  ref (a) from  tblautores a
where a.codigoautor=3;
insert into tbllibros select 5, 'El pendulo de foucault',  ref (a) from  tblautores a
where a.codigoautor=3;
insert into tbllibros select 6, 'Los pilares de la tierra',  ref (a) from  tblautores a
where a.codigoautor=4;
insert into tbllibros select 7, 'Un mundo sin fin',  ref (a) from  tblautores a
where a.codigoautor=4;
```

POSTGRESQL

```
create table tbllibros (
    libro objlibro primary key,
    autor objautor references tblautores(autor)
);
```

```
insert into tbllibros values ((1, 'Don Quijote de la Mancha'), (1, 'Miguel de Cervantes'));
insert into tbllibros values ((2, 'Hamlet'),  (2, 'William Shakespeare'));
insert into tbllibros values ((3, 'Macbeth'), (2, 'William Shakespeare'));
insert into tbllibros values ((4, 'El nombre de la rosa'), (3, 'Umberto Eco'));
insert into tbllibros values ((5, 'El pendulo de foucault'),  (3, 'Umberto Eco'));
insert into tbllibros values ((6, 'Los pilares de la tierra'),  (4, 'Ken Follet'));
insert into tbllibros values ((7, 'Un mundo sin fin'),  (4, 'Ken Follet'));
```

8. INTEGRIDAD REFERENCIAL

- Consultamos su contenido. Para obtener la información hemos de acceder a la información almacenada por el enlace. En este caso utilizamos la instrucción DEREF y el campo al que queremos acceder.

ORACLE

```
-- algunos selects con libros
select * from tbllibros;
select codigolibro, nombre, deref(autor).nombre as autor from tbllibros;
select codigolibro, nombre, deref(autor).nombre as autor from tbllibros
where deref(autor).nombre='Umberto Eco';
```

	CODIGOLIBRO	NOMBRE	AUTOR
1	1	Don Quijote de la Mancha	Miguel de Cervantes
2	3	Macbeth	William Shakespeare
3	2	Hamlet	William Shakespeare
4	5	El pendulo de foucault	Umberto Eco
5	4	El nombre de la rosa	Umberto Eco
6	7	Un mundo sin fin	Ken Follet
7	6	Los pilares de la tierra	Ken Follet

POSTGRESQL

```
select (libro).codigolibro, (libro).nombre, (autor).nombre  from tbllibros
where (autor).nombre='Umberto Eco';
```

ta Output | Explain | Messages | Notifications

codigolibro integer	nombre character varying (40)	nombre character varying (50)
4	El nombre de la rosa	Umberto Eco
5	El pendulo de foucault	Umberto Eco

8. INTEGRIDAD REFERENCIAL

- Creamos la tabla tblprestamos e insertamos datos.

ORACLE

```
-- creamos la tabla tblprestamo

create table tblprestamo of prestamo (id primary key, libro references tbllibros,
lector references tbllectores);

insert into tblprestamo select 1, to_date('20-12-2016', 'dd-mm-yyyy'), to_date('27-12-2016', 'dd-mm-yyyy'),
ref(l),ref(e) from tbllibros l, tbllectores e
where l.codigolibro=7 and e.nombre='Joan';

insert into tblprestamo select 2, to_date('21-12-2016', 'dd-mm-yyyy'), to_date('28-12-2016', 'dd-mm-yyyy'),
ref(l),ref(e) from tbllibros l, tbllectores e
where l.codigolibro=1 and e.nombre='Marta';

insert into tblprestamo select 3, to_date('1-1-2017', 'dd-mm-yyyy'), to_date('7-1-2017', 'dd-mm-yyyy'),
ref(l),ref(e) from tbllibros l, tbllectores e
where l.codigolibro=3 and e.nombre='Miguel';

insert into tblprestamo select 4, to_date('1-1-2017', 'dd-mm-yyyy'), to_date('7-1-2017', 'dd-mm-yyyy'),
ref(l),ref(e) from tbllibros l, tbllectores e
where l.codigolibro=4 and e.nombre='Anna';
```

POSTGRESQL

```
drop table tblprestamos;
create table tblprestamos of objprestamo (
    id primary key,
    libro references tbllibros,
    lector references tbllectores
);
```

```
insert into tblprestamos values (1, to_date('20-12-2016', 'dd-mm-yyyy'),
    to_date('27-12-2016', 'dd-mm-yyyy'), (7, 'Un mundo sin fin'),(1, 'Joan', 47));
insert into tblprestamos values (2, to_date('21-12-2016', 'dd-mm-yyyy'),
    to_date('28-12-2016', 'dd-mm-yyyy'), (1, 'Don Quijote de la Mancha'),(2, 'Marta', 28 ));
insert into tblprestamos values (3, to_date('1-1-2017', 'dd-mm-yyyy'),
    to_date('7-1-2017', 'dd-mm-yyyy'), (3, 'Macbeth'),(3, 'Miguel', 18));
insert into tblprestamos values (4, to_date('1-1-2017', 'dd-mm-yyyy'),
    to_date('7-1-2017', 'dd-mm-yyyy'), (4, 'El nombre de la rosa'),(4, 'Anna', 23));
```

8. INTEGRIDAD REFERENCIAL

- Consultamos su contenido. Para obtener la información hemos de acceder a la información almacenada por el enlace. En este caso utilizamos la instrucción DEREF y el campo al que queremos acceder.

ORACLE

```
-- select
select * from tblprestamo;
select id,fechainicio,fechadevolucion, deref(libro).nombre as libro,
deref(lector).nombre as lector from tblprestamo;
-- accedemos a las propiedades de un objeto incluido en otro objeto.
select id,fechainicio,fechadevolucion, deref(libro).nombre as libro,
deref(libro).autor.nombre as autor, deref(lector).nombre as lector from tblprest
```

	ID	FECHAINICIO	FECHADEVOLUCION	LIBRO	AUTOR	LECTOR
1	1	20/12/16	27/12/16	Un mundo sin fin	Ken Follet	Joan
2	2	21/12/16	28/12/16	Don Quijote de la Mancha	Miguel de Cervantes	Marta
3	3	01/01/17	07/01/17	Macbeth	William Shakespeare	Miguel
4	4	01/01/17	07/01/17	El nombre de la rosa	Umberto Eco	Anna

POSTGRESQL

```
select * from tblprestamos;
select id, fechainicio, fechadevolucion, (libro).nombre,(lector).nombre
from tblprestamos;
```

a Output | Explain | Messages | Notifications

id [PK] integer	fechainicio date	fechadevolucion date	nombre character varying (40)	nombre character varying (40)
1	2016-12-20	2016-12-27	Un mundo sin fin	Joan
2	2016-12-21	2016-12-28	Don Quijote de la Mancha	Marta
3	2017-01-01	2017-01-07	Macbeth	Miguel
4	2017-01-01	2017-01-07	El nombre de la rosa	Anna

```
select p.id,p.fechainicio,p.fechadevolucion, (p.libro).nombre,(p.lector).nombre,(l.autor).nombre
from tblprestamos p inner join tbllibros l on p.libro=l.libro
```

a Output | Explain | Messages | Notifications

id [PK] integer	fechainicio date	fechadevolucion date	nombre character varying (40)	nombre character varying (40)	nombre character varying (50)
2	2016-12-21	2016-12-28	Don Quijote de la Mancha	Marta	Miguel de Cervantes
3	2017-01-01	2017-01-07	Macbeth	Miguel	William Shakespeare
4	2017-01-01	2017-01-07	El nombre de la rosa	Anna	Umberto Eco
1	2016-12-20	2016-12-27	Un mundo sin fin	Joan	Ken Follet

DOMINIOS EN POSTGRESQL

- Postgres permite definir dominios para un atributo dado.
- Un dominio consiste en un tipo, definido por el usuario o incluido en el sistema (integer, varchar, date, etc) más un conjunto de restricciones.
- Por tanto, a diferencia del tipo compuesto, un dominio Sí permite restricciones.

- Sintaxis:

```
CREATE DOMAIN <name> DATA_TYPE
        [COLLATE collation]
        [DEFAULT expresion]
        {NOT NULL | NULL | CHECK (expression)} [...]
```

- Veamos algún ejemplo:

DOMINIOS EN POSTGRESQL

- CREATE DOMAIN country_code char(2) NOT NULL; → Se crea el tipo country_code que solo puede contener dos caracteres y no puede ser nulo
- CREATE DOMAIN edad integer check (value >0 and value <100); → Se crea el tipo edad que solo permite valores enteros positivos menores que 100. Observar cómo se utiliza la palabra clave 'value' para referenciar el valor del campo.
- Dentro del check se puede incluir cualquier función del sistema o cualquier función definida por el usuario.
- Hay que recordar que la cláusula check se puede incluir en las restricciones de columna cuando estamos definiendo una tabla. La opción de crear un dominio es útil cuando pensamos utilizar dicho dominio en la creación de diversas tablas.

DOMINIOS EN POSTGRESQL

- Crea un dominio para el correo de un cliente. Debe permitir cuentas de correo acabadas en @gmail.com
- Crea una tabla que contenga como identificador el nombre y apellido del cliente y también almacene - en otro campo, lógicamente - el correo del cliente. Exige que se utilice los identificadores ocultos OID y TABLEOID
- Realiza alguna inserción en la tabla anterior comprobando que efectivamente el campo que debe almacenar las cuentas de correo actúa correctamente.
- Realiza alguna consulta en la tabla anterior pidiendo al sistema que muestre los identificadores ocultos.