



BASES DE DATOS 2

MEMORIA DE LA PRÁCTICA 1-2
CURSO 2017-2018

CARLOS MARAÑES NUENO 717788@CELES.UNIZAR.ES

NICOLÁS LERA LÓPEZ 721808@CELES.UNIZAR.ES



ÍNDICE

INTRODUCCIÓN	3
CONFIGURACIÓN DE LA MÁQUINA VIRTUAL	3
INSTALACIÓN Y ADMINISTRACIÓN BÁSICA DE LOS SGBD	3
COMENTARIOS ACERCA DE LAS LICENCIAS	9
DISEÑO CONCEPTUAL DE LA BASE DE DATOS	11
DISEÑO LÓGICO DE UNA BASE DE DATOS RELACIONAL	12
IMPLEMENTACIÓN CON EL MODELO RELACIONAL	12
DISEÑO LÓGICO DE UNA BASE DE DATOS OBJETO/RELACIONAL	14
IMPLEMENTACIÓN CON EL MODELO OBJETO/RELACIONAL	16
GENERACIÓN DE DATOS Y PRUEBAS	29
IMPLEMENTACIÓN CON DB4O	40
COMPARACIÓN DE LOS SGBD	54
DIFICULTADES ENCONTRADAS	55
ESFUERZOS INVERTIDOS	56
BIBLIOGRAFÍA	57

1. INTRODUCCIÓN

La siguiente práctica tiene como objetivo el diseño e implementación de una base de datos bancaria tanto en el modelo relacional, objeto-relacional y objeto puro en diferentes sistemas gestores de bases de datos. Además, se instalarán dichos gestores en la máquina virtual proporcionada y se detallarán todos los problemas surgidos y las correspondientes soluciones aplicadas.

2. CONFIGURACIÓN DE LA MÁQUINA VIRTUAL

La máquina virtual se gestiona desde el entorno VirtualBox¹. La máquina se instala de igual manera que se indica en Moodle excepto que hay que cambiar un parámetro. Normalmente da un error de ejecución al lanzarla. Ésto es debido a que VirtualBox genera errores con la configuración de los USBs. Para solucionar el problema, se selecciona la máquina, a continuación se pulsa el botón de configuración, después al apartado USB, se deshabilitan y se confirman los cambios. Una vez hecho esto la máquina puede lanzarse.

Para evitar las configuraciones del teclado dentro de la máquina virtual, para trabajar con ella se conecta vía ssh con el siguiente comando, si se está utilizando en un sistema operativo Linux:

```
ssh -p 2222 root@localhost
```

Pedirá la contraseña, que por defecto es *root*. Ahora ya se puede trabajar con la máquina. Para copiar ficheros a la misma se utiliza el comando *scp*. Por ejemplo, será utilizado para copiar algunos de los instaladores de los diferentes gestores.

3. INSTALACIÓN Y ADMINISTRACIÓN BÁSICA DE LOS SGBD

Antes de instalar ningún gestor, se ejecutan los siguientes comandos:

```
apt-get update  
apt-get upgrade  
apt-get install sudo
```

¹ VirtualBox: <https://www.virtualbox.org/>

El primero de ellos es para actualizar la lista de los paquetes disponibles, ya que la máquina se encuentra desactualizada. El segundo es para actualizar los paquetes anteriores. Si pide configurar el `localepurge`, se deja por defecto en el primero y en los dos restantes se selecciona la primera opción, que deja la versión que estaba:

keep the local version currently installed

El último de ellos es para instalar *sudo*², ya que se necesitarán privilegios para los usuarios en algunos de los gestores.

1. INSTALACIÓN MYSQL

En primer lugar se instala el SGBD:

apt-get install mysql-server mysql-client

Solicitará una clave para root, se pone *root* para evitar confusiones. Una vez instalado el sistema, ya se puede conectar a él:

mysql -u root -p

El siguiente paso es crear la base de datos, en este caso *banco*. Para ello se ejecuta el siguiente comando:

CREATE DATABASE banco;

Finalmente, ya se puede conectar a la base de datos creada con el comando:

CONNECT banco;

A partir de aquí ya se pueden ejecutar las sentencias SQL deseadas, ya sean para crear tablas o hacer consultas.

COMANDOS BÁSICOS

- Arrancar el gestor: *service mysql start*
- Parar el gestor: *service mysql stop*
- Crear una base de datos: *CREATE DATABASE <nombreBaseDeDatos>*
- Crear un usuario:
CREATE USER '<nombreUsuario>'@'localhost' IDENTIFIED BY '<contraseña>';
- Otorgar permisos al usuario. Con la base de datos ya creada, se le dan permisos al usuario para que sólo él (y el administrador) pueda trabajar en ella:

GRANT ALL PRIVILEGES ON <nombreBBDD>.<nombreTabla> TO '<nombreUsuario>'@'localhost';

FLUSH PRIVILEGES;

Por ejemplo, para que tenga privilegios en la base de datos creada anteriormente y pueda manipular todas las tablas:

GRANT ALL PRIVILEGES ON banco. TO 'pepito'@'localhost';*

Para crear una tabla de ejemplo:

² Sudo: <https://es.wikipedia.org/wiki/Sudo>

```
CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);
```

2. INSTALACIÓN ORACLE

Antes de ejecutar el instalador de Oracle, se han de resolver sus dependencias. Entre ellas se encuentra que se necesita un paquete llamado libaio1 y bc, por lo que se instala con el comando:

```
apt-get install bc  
apt-get install libaio1
```

Ahora se tiene que copiar el instalador de Oracle en la máquina virtual, para ello, se descarga en la misma con el siguiente comando:

```
wget  
https://oss.oracle.com/debian/dists/unstable/non-free/binary-i386/oracle-xe\_10.2.0.1-1.1\_i386.deb
```

Antes de ejecutar el instalador, hay que ajustar el tamaño de Swap tal y como se indica en Moodle.

Ahora, se ejecuta el instalador de Oracle con el comando (en el directorio en el que se encuentra)

```
dpkg -i oracle-xe_10.2.0.1-1.1_i386.deb
```

Una vez se haya instalado el gestor, se configura ejecutando el comando:

```
/etc/init.d/oracle-xe configure
```

Una vez hecho esto se configuran todos los ficheros tal y como está especificado en Moodle.

Para poder conectarse con la base de datos se ejecuta el script:

```
./etc/profile
```

Finalmente, hay que habilitar el servidor con el comando:

```
/etc/init.d/oracle-xe enable
```

COMANDOS BÁSICOS

Se accede a la ventana de comandos con sqlplus

- Arrancar el gestor: `/etc/init.d/oracle-xe start`
- Parar el gestor: `/etc/init.d/oracle-xe stop`

- Crear una base de datos: CREATE DATABASE <nombreBaseDeDatos>

En Oracle funciona de manera que cada usuario tiene asociada una base de datos. Por lo que para crear una base de datos, es necesario crear un usuario y darle todos los permisos correspondientes.

- Crear un usuario: CREATE USER <nombreUsuario> IDENTIFIED BY <contraseña>;
- Otorgar permisos al usuario. Con la base de datos ya creada, se le dan permisos al usuario para que sólo él (y el administrador) pueda trabajar en ella:

grant CREATE SESSION, ALTER SESSION, CREATE DATABASE LINK,
CREATE MATERIALIZED VIEW, CREATE PROCEDURE, CREATE PUBLIC SYNONYM,
CREATE ROLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE,
CREATE TRIGGER, CREATE TYPE, CREATE VIEW, UNLIMITED TABLESPACE
to <nombreUsuario>

- Para ejecutar un script SQL. Desde la consola de Oracle: @<nombreScript>

3. INSTALACIÓN POSTGRESQL

Para instalar PostgreSQL basta con ejecutar el siguiente comando:

```
apt-get install postgresql postgresql-contrib
```

Al instalarlo, se ha creado en el sistema un usuario llamado postgres, asociado al rol Postgres por defecto. Postgres utiliza un concepto llamado “roles”, en vez de usuarios y grupos. Si un rol existe, entonces hay un usuario en el sistema que tiene el mismo nombre del rol y se puede acceder con él al gestor. Ahora bien, se cambia al usuario postgres con el comando:

```
su postgres
```

Posteriormente, se accede a la consola Postgres con el comando *psql*.

COMANDOS BÁSICOS

- Arrancar el gestor: */etc/init.d/postgresql start*
- Parar el gestor: */etc/init.d/postgresql stop*
- Crear una base de datos. Se crea la base de datos para un usuario (rol) con el mismo nombre que el rol, para que por defecto tenga dicha base de datos:

```
createdb <nombreDB>
```

- Crear un usuario. Como se ha dicho anteriormente, Postgres utiliza un sistema de roles. Con la cuenta de postgres se puede ejecutar el comando:

```
createuser <nombreDelUsuario>
```

Si ese usuario no existe en el sistema operativo hay que crearlo, en Linux, desde la cuenta root:

```
adduser <nomreDelUsuario>
```

- Otorgar permisos al usuario. Con la base de datos ya creada, se le dan permisos al usuario para que sólo él (y el administrador) pueda trabajar en ella. Desde la terminal psql, el administrador (postgres), ejecuta la siguiente sentencia:
GRANT ALL PRIVILEGES ON DATABASE <nombreBBDD> TO <nombreUsuario>;

Para que el usuario se conecte a la base de datos ejecuta el comando desde su terminal:

```
psql -u <nombreUsuario> -W <nombreBBDD>
-W indica que se le pedirá la contraseña
```

Para crear una tabla de ejemplo:

```
CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);
```

- Para ejecutar un script SQL: `psql -f <nombreScript>`

4. INSTALACIÓN DB2

En primer lugar, se descarga el instalador desde la siguiente URL en la máquina donde se esté ejecutando Virtualbox:

<https://jazz.net/downloads/DB2/releases/10.1>

Una vez descargado se extrae el contenido y se copia en la máquina virtual vía scp con el comando:

```
scp -P 2222 -r expc root@localhost:/root
```

Antes de ejecutar el instalador, se ha de preparar el sistema. Para ello hay que solucionar, en primer lugar, la falta de *binutils*, que se instala con el comando:

```
apt-get install binutils
```

También hay que solucionar la falta de un archivo, que se soluciona creando el link simbólico correspondiente con el comando (previamente *apt-get install libaio1*):

```
ln -s /lib/i386-linux-gnu/libpam.so.0 /lib/libpam.so
```

Hecho esto, ya se puede ejecutar el instalador, dentro de la carpeta expc: `./db2_install`. Pedirá el directorio de instalación, que se deja por defecto, que es `/opt/ibm/db2`.

COMANDOS BÁSICOS

- Arrancar el gestor. Desde el poseedor de la instancia: *db2start*
- Parar el gestor. Desde el poseedor de la instancia: *db2stop*

- Crear una base de datos. Antes se tiene que definir el administrador, con el comando:

```
. /opt/ibm/db2/V10.1/instance/dascrt -u <nombreUsuario>
```

Donde el usuario es el administrador del servidor del gestor, que siguiendo la tabla que se comenta más adelante es dasusr1.

Una vez definido, se procede a crear una instancia con el comando:

```
. /opt/ibm/db2/V10.1/instance/db2icrt -a server -u <usuario> <nombreInstancia>
```

Donde usuario es el usuario delimitado (se puede omitir) que puede ejecutar funciones definidas por el usuario, por ejemplo:

```
. /opt/ibm/db2/V10.1/instance/db2icrt -a server -u db2fenc1 db2inst1
```

Ahora ya se puede trabajar en la base de datos, para ello se cambia al usuario poseedor, en este caso db2inst1, se accede a la nueva carpeta llamada sqllib que se ha creado en el home y se ejecuta el scrip db2profile. En este momento ya se poseen todos los comandos y se puede acceder a la instancia con el comando *db2*.

Para crear una base de datos se ejecuta el comando: *create db <nombreBBDD>*

Antes se ha tenido que arrancar el sistema db2.

Una vez creada la base de datos ya se puede conectar a ella y crear una tabla, por ejemplo:

```
db2 connect to prueba
```

Luego se ejecuta db2 para entrar a la terminal y se puede crear una tabla de ejemplo:

```
create table employee ( Empno smallint, Name varchar(30))
```

- Crear un usuario. Antes de nada hay que definir los usuarios y grupos en el sistema, según la siguiente tabla:

User	Example user name	Example group name
Instance owner	db2inst1	db2iadm1
Fenced user	db2fenc1	db2fadm1
DB2 administration server user	dasusr1	dasadm1

Se crean los grupos:

```
groupadd -g 999 db2iadm1
```

```
groupadd -g 998 db2fadm1
```

```
groupadd -g 997 dasadm1
```

Se crean los usuarios:

```
useradd -u 1004 -g db2iadm1 -m -d /home/db2inst1 db2inst1
```

```
useradd -u 1003 -g db2fadm1 -m -d /home/db2fenc1 db2fenc1
```

```
useradd -u 1002 -g dasadm1 -m -d /home/dasusr1 dasusr1
```

Se ponen las contraseñas (*contra*):

```
passwd db2inst1
```

```
passwd db2fenc1
```

```
passwd dasusr1
```

- Otorgar permisos al usuario. Como en principio se trabaja por instancias, cada usuario tiene las suyas propias y no puede acceder a las del resto.

- Ejecutar un script sql: `db2 -stv <nombreScript>`

4. COMENTARIOS ACERCA DE LAS LICENCIAS

1. MySQL

MySQL puede ser usado sin coste si la aplicación está desarrollada localmente y no tiene un uso comercial. Solo cuando el objetivo de esta aplicación es ser vendida a unos clientes aparece la cuestión de la licencia comercial.

MySQL puede ser usado gratuitamente en una página web. Si también se crea una aplicación PHP y se instala en un servidor web, no es necesario hacer tu código PHP open source en el sentido de GPL.

Finalmente, MySQL license puede ser usado gratuitamente para todo los proyectos que funcionan bajo GPL u otras licencias gratuitas similares.

Está estrictamente prohibido en el sentido de GPL cambiar, extender o vender nuevas versiones del código sin permitir que todo el mundo pueda utilizar tu código gratuitamente. Tampoco puedes desarrollar una nueva base de datos basada en MySQL. MySQL ofrece licencias de pago comerciales que permiten ciertas acciones no permitidas con la licencia GPL.

2. Oracle

Oracle permite utilizar este gestor para desarrollar, prototipar y lanzar aplicaciones para procesamiento de operaciones internas de datos. También permite distribuir los programas con sus aplicaciones y desarrollar licencias propias para nuestros programas siempre que cumplan con los términos de la licencia.

Tampoco está permitido nombrar ninguno de los programas/aplicaciones creados con este gestor con la palabra "ORACLE" formando parte del nombre o con otra derivación como "ORA" que pueda relacionarse con la empresa.

No está permitida la descompilación o ingeniería inversa sobre los programas ni permitir esto. Tampoco se pueden modificar los derechos del programa o modificar los resultados de los benchmark test sobre los programas en beneficio del creador.

No está permitido el uso o distribución de este gestor a cualquier persona, ya sea nacionalizado o residente, que esté bajo el control del gobierno de Cuba, Irán, Sudán, Libia,

Corea del Norte, Siria o cualquier otro país al que Estados Unidos haya prohibido la exportación.

A su vez, no se puede utilizar para ninguna acción que no permita la ley de EEUU, sobre todo nombrando el desarrollo de armas de destrucción masiva.

3. PostgreSQL

PostgreSQL da permiso para usar, copiar, modificar y distribuir su software y su documentación para cualquier propósito, sin cargo y sin ningún escrito de acuerdo necesario siempre que lo siguiente aparezca en todas las copias:

PostgreSQL Database Management System

(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2010, The PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

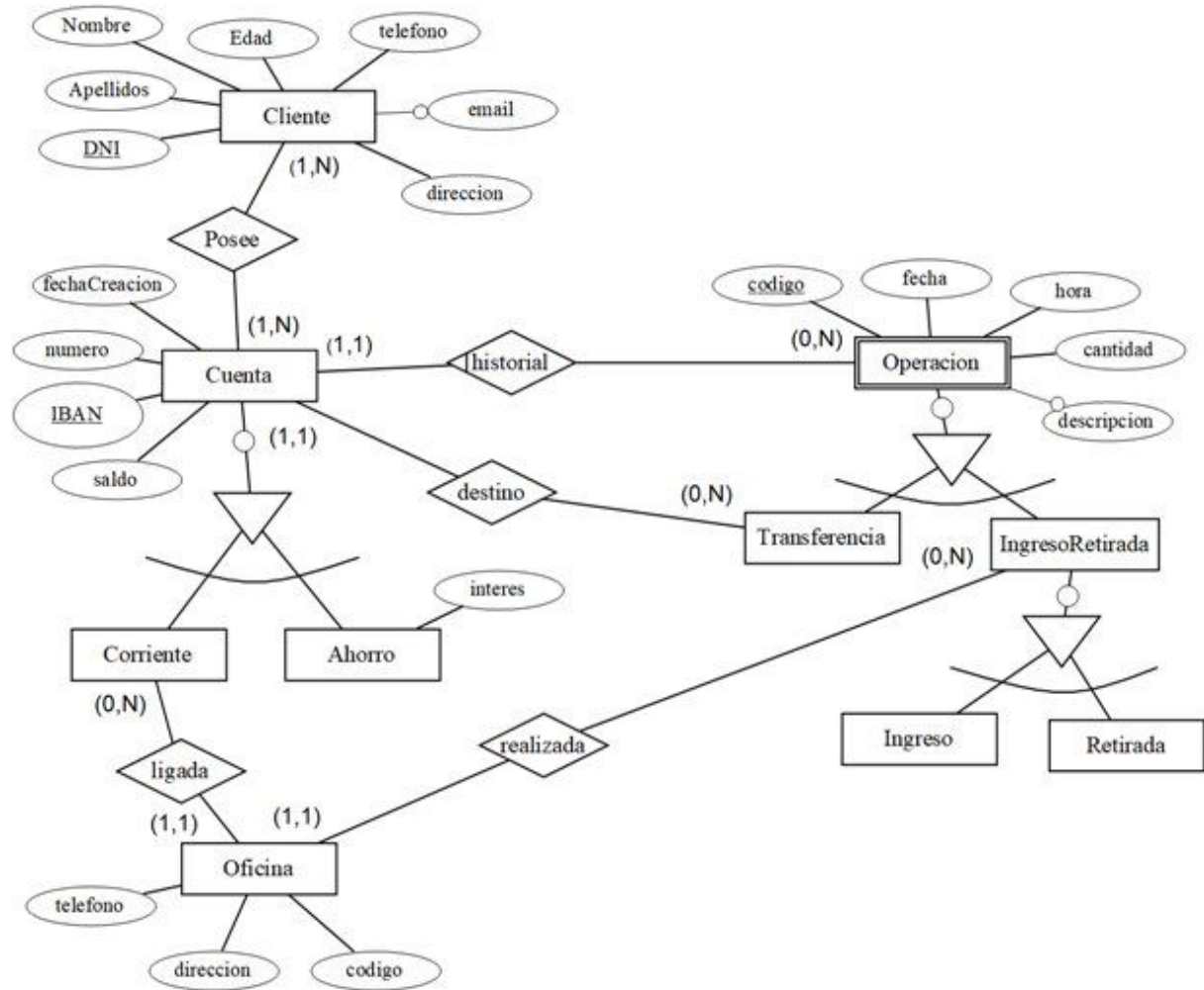
IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING
LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION,
EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS,
AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE,
SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

4. DB2 Express C

DB2 Express-C es la versión gratuita del sistema gestor de bases de datos de IBM. IBM permite con esta versión desarrollar, utilizar en la producción y distribuir el trabajo realizado con o sobre este sistema gestor de bases de datos.

5. DISEÑO CONCEPTUAL DE LA BASE DE DATOS



6. DISEÑO LÓGICO DE UNA BASE DE DATOS RELACIONAL

Oficina (codigo : cadena, direccion : cadena, telefono : natural)

Cliente (DNI : cadena, Nombre : cadena, Apellidos : cadena, Edad : natural, direccion : cadena, email : cadena, telefono : cadena)

Cuenta (IBAN : cadena, numero : natural, saldo : real, fechaCreacion : cadena, interes : real, código : cadena, tipo : cadena)

Posee (DNI : cadena, IBAN : cadena)

Operacion (codigo : cadena, fecha : cadena, hora : cadena, cantidad : natural, descripcion : cadena, IBAN : cadena, IIBANorig : cadena, oficina : cadena)

Por decisiones de diseño, se ha decidido que ambas especializaciones se resuelvan en una única tabla. Como inconveniente, se ha de garantizar la consistencia a través de disparadores. En el caso de Cuenta, si es corriente debe estar asociada a una oficina. En el caso de operación, si es de ingreso o retirada se debe asociar a una oficina y si es de transferencia debe tener una cuenta destino.

Para la relación posee, se debería garantizar, por medio de un disparador por ejemplo, que un cliente está asociado a una cuenta y viceversa, para traducir las cardinalidades mínimas.

Para eliminar los datos, aquellos que están relacionados se debe eliminar cuando se eliminan en un extremo, si es de tipo entidad débil como lo es operación.

7. IMPLEMENTACIÓN CON EL MODELO RELACIONAL

A continuación se incluye el código en SQL adaptado para Oracle que incluye la creación de las tablas:

```
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Operacion';
    EXCEPTION
        WHEN OTHERS THEN NULL;
END;
```

/

```
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Posee';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Cuenta';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Oficina';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Cliente';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

CREATE TABLE Cliente(
    DNI VARCHAR(10) NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    edad INTEGER NOT NULL,
    Direccion VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    Telefono VARCHAR(15) NOT NULL,
    PRIMARY KEY (DNI)
);

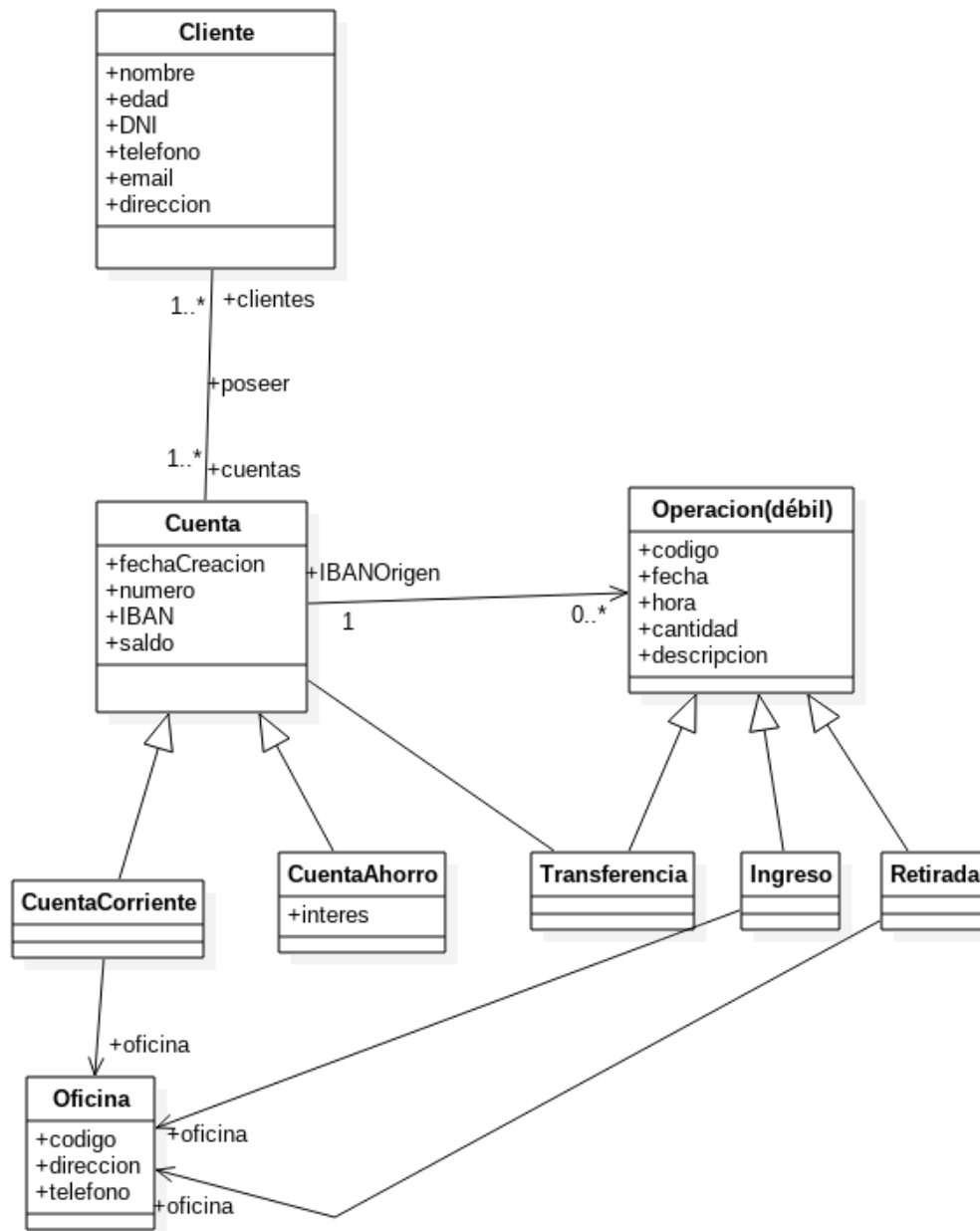
CREATE TABLE Oficina(
    codigo VARCHAR(15) NOT NULL,
    direccion VARCHAR(100) NOT NULL,
    telefono VARCHAR(15) NOT NULL, /* mejor varchar para especificar formato */
    PRIMARY KEY (codigo)
);

CREATE TABLE Cuenta(
    IBAN VARCHAR(150) NOT NULL,
    numero INTEGER NOT NULL,
    saldo DECIMAL(10,2) NOT NULL,
    fechaCreacion DATE NOT NULL,
    tipo VARCHAR(9) NOT NULL,
    interes DECIMAL(3,2),
    oficina VARCHAR(15),
```

```
PRIMARY KEY(IBAN),  
FOREIGN KEY(oficina) REFERENCES Oficina(codigo)  
);  
  
CREATE TABLE Posee(  
    DNI VARCHAR(10) NOT NULL,  
    IBAN VARCHAR(150) NOT NULL,  
    PRIMARY KEY (DNI,IBAN),  
    FOREIGN KEY (DNI) REFERENCES Cliente(DNI),  
    FOREIGN KEY (IBAN) REFERENCES Cuenta(IBAN)  
);  
  
CREATE TABLE Operacion(  
    codigo VARCHAR(150) NOT NULL,  
    tipo VARCHAR(20) NOT NULL,  
    fecha DATE NOT NULL,  
    hora VARCHAR(8) NOT NULL,  
    cantidad DECIMAL(10,2) NOT NULL,  
    descripcion VARCHAR(400),  
    IBANOrigen VARCHAR(150) NOT NULL,  
    IBANDestino VARCHAR(150),  
    oficina VARCHAR(15),  
    PRIMARY KEY (codigo, IBANOrigen),  
    FOREIGN KEY (IBANOrigen) REFERENCES Cuenta(IBAN),  
    FOREIGN KEY (IBANDestino) REFERENCES Cuenta(IBAN),  
    FOREIGN KEY (oficina) REFERENCES Oficina(codigo)  
);
```

8. DISEÑO LÓGICO DE UNA BASE DE DATOS OBJETO/RELACIONAL

El esquema Objeto/Relacional es el siguiente:



Se ha tomado la decisión de eliminar la entidad *poseer*, en el caso en el que el gestor soporte arrays de referencias. También se ha decidido crear entidades separadas para las diferentes cuentas y operaciones para utilizar la **herencia** que hace característico al modelo Objeto/Relacional. Dependiendo del gestor, la herencia será de tipos, tablas o ambos dependiendo del soporte que tengan.

Las relaciones 1:N se han hecho unidireccionales por sencillez, ya que por el contrario se ha de garantizar la consistencia con procedimientos o triggers.

También hacen falta triggers para comprobar que las relaciones N:M tengan cardinalidad mínima. En este caso, en la relación *poseer*.

9. IMPLEMENTACIÓN CON EL MODELO OBJETO/RELACIONAL

El esquema no se ha traducido directamente a las tablas correspondientes sino que se han tenido que llevar a cabo ligeras modificaciones para adaptarlas al SGBD que se está utilizando.

1. Oracle

Como Oracle soporta de forma nativa los arrays de referencias, se ha podido realizar la relación Cliente - Cuenta tal y como lo permite el estándar. Sin embargo, la consistencia se ha de comprobar mediante el uso de triggers, tanto como en la inserción como en el borrado. Ésto último se ha de exigir también en todo tipo de relaciones en el que las identidades tengan algún atributo de tipo referencia, ya que la versión de Oracle utilizada no permite el uso de claves ajenas a tipos de referencia. Para la relación Cliente-Cuenta se ha creado un procedimiento para la inserción de los datos de la relación como ejemplo. También se debería crear uno para la eliminación.

Para la creación de las tablas, en primer lugar se crea el tipo y después las tablas tipadas. Los tipos que dependen de otros y viceversa se han de declarar previamente para poder realizar las asociaciones. Posteriormente, éstos se modifican.

Oracle no permite jerarquía de tablas, al menos en la versión utilizada:

```
CREATE TABLE CuentaCorriente OF CuentaCorrienteUdt UNDER Cuenta(  
*  
ERROR at line 1:  
ORA-03001: unimplemented feature  
  
SQL> █
```

Se ha intentado realizar comprobar referencias pero la versión tampoco lo permite.

La especialización de Cuenta y de Operación se realiza a través de una sola tabla, pero al poseer jerarquía de tipos, la tabla se crea con el tipo padre y también permite almacenar a sus hijos.

```
/* ELIMINAR TABLAS SI EXISTIAN */
```

```
BEGIN  
    EXECUTE IMMEDIATE 'DROP TABLE Cliente FORCE';  
    EXCEPTION  
        WHEN OTHERS THEN NULL;  
END;  
/
```

```
BEGIN  
    EXECUTE IMMEDIATE 'DROP TABLE Cuenta FORCE';  
    EXCEPTION
```



```
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Oficina FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Operacion FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

/* ELIMINACION DE TIPOS SI EXISTIAN */

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE array_cuentas FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE array_clientes FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE TransferenciaUdt FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE IngresoUdt FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
    END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE RetiradaUdt FORCE';
    EXCEPTION
        WHEN OTHERS THEN NULL;
```

```
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE OperacionUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE CuentaCorrienteUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE CuentaAhorroUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE OficinaUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE ClienteUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TYPE CuentaUdt FORCE';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

/* BLOQUE DECLARACION TIPOS Y ARRAYS */

SET SERVEROUTPUT ON;
```

```
/* De Cliente <-> Cuenta */
CREATE TYPE ClienteUdt;
/
CREATE TYPE CuentaUdt;
/

/* VARRAYS QUE SE NECESITAN */
CREATE TYPE array_cuentas AS VARRAY(50) of REF CuentaUdt;
/

CREATE TYPE array_clientes AS VARRAY(20) of REF ClienteUdt;
/

/* De cuentaCorriente <-> Banco*/
CREATE TYPE CuentaCorrienteUdt;
/

CREATE TYPE CuentaAhorroUdt;
/

/* De Cuenta <-> Operacion */
CREATE TYPE OperacionUdt;
/

/* De Cuenta <-> Transferencia */

CREATE TYPE TransferenciaUdt;
/

/*----- DEFINICION DE TIPOS ----- */

CREATE TYPE ClienteUdt AS OBJECT (
    nombre    VARCHAR(100),
    apellido VARCHAR(100),
    DNI VARCHAR(10),
    edad INTEGER,
    direccion VARCHAR(100),
    email VARCHAR(100),
    telefono INTEGER,
    cuentas array_cuentas
)
NOT FINAL;
/

CREATE TYPE CuentaUdt AS OBJECT(
    IBAN VARCHAR(150),
    numero INTEGER,
    saldo DECIMAL(10,2),
    fechaCreacion DATE,
    clientes array_clientes
```

```
)  
NOT FINAL;  
/
```

```
CREATE TYPE OficinaUdt AS OBJECT(  
    codigo VARCHAR(10),  
    direccion VARCHAR(50),  
    telefono VARCHAR(10)  
)  
NOT FINAL;  
/
```

```
CREATE TYPE CuentaCorrienteUdt UNDER CuentaUdt(  
    oficina ref OficinaUdt  
)  
NOT FINAL;  
/
```

```
CREATE OR REPLACE TYPE CuentaAhorroUdt UNDER CuentaUdt(  
    interes DECIMAL(3,2)  
)  
NOT FINAL;  
/
```

```
CREATE TYPE OperacionUdt AS OBJECT(  
    codigo VARCHAR(150),  
    IBANOrigen VARCHAR(150),  
    fecha DATE,  
    hora VARCHAR(8),  
    cantidad DECIMAL(10,2),  
    descripcion VARCHAR(400),  
    cuentaOrigen ref CuentaUdt  
)  
NOT FINAL;  
/
```

```
CREATE TYPE TransferenciaUdt UNDER OperacionUdt(  
    IBANDestino ref CuentaUdt  
)  
NOT FINAL;  
/
```

```
CREATE TYPE IngresoUdt UNDER OperacionUdt(  
    oficina ref OficinaUdt  
)  
NOT FINAL;  
/
```

```
CREATE TYPE RetiradaUdt UNDER OperacionUdt(  
    oficina ref OficinaUdt  
)  
NOT FINAL;  
/
```

```
/* CREACION DE TABLAS */
```

```
CREATE TABLE Cliente OF ClienteUdt(  
    nombre NOT NULL,  
    apellido NOT NULL,  
    dni PRIMARY KEY,  
    edad NOT NULL,  
    telefono NOT NULL,  
    direccion NOT NULL,  
    cuentas NOT NULL  
) object id system generated;
```

```
CREATE TABLE Cuenta OF CuentaUdt(  
    IBAN PRIMARY KEY,  
    numero NOT NULL,  
    saldo DEFAULT 0,  
    fechaCreacion NOT NULL,  
    clientes NOT NULL  
) object id system generated;
```

```
CREATE TABLE Oficina OF OficinaUdt(  
    codigo PRIMARY KEY,  
    direccion NOT NULL,  
    telefono NOT NULL  
) object id system generated;
```

```
CREATE TABLE Operacion OF OperacionUdt(  
    codigo PRIMARY KEY,  
    IBANOrigen PRIMARY KEY  
    fecha NOT NULL,  
    hora NOT NULL,  
    cantidad NOT NULL,  
    cuentaOrigen NOT NULL,  
    FOREIGN KEY IBANOrigen REFERENCES Cuenta(IBAN)  
) object id system generated;
```

```
/* Procedimiento para la relacion cliente-cuenta */
```

```
CREATE OR REPLACE PROCEDURE relacionClienteCuenta(dniAsociar VARCHAR, ibanAsociar VARCHAR)  
IS  
    cuentas_aux array_cuentas := array_cuentas();  
    clientes_aux array_clientes := array_clientes();  
    cuenta_ref REF CuentaUdt;
```

```
    cliente_ref REF ClienteUdt;
BEGIN
-- Se hace primero con la tabla Cliente
EXECUTE IMMEDIATE
    'SELECT cuentas FROM Cliente WHERE DNI = :1 FOR UPDATE OF cuentas' INTO cuentas_aux
USING dniAsociar;

    SELECT ref(C) into cuenta_ref FROM Cuenta C WHERE C.IBAN=ibanAsociar;

    cuentas_aux.EXTEND(1);
    cuentas_aux(cuentas_aux.LAST) := cuenta_ref;

EXECUTE IMMEDIATE
    'UPDATE Cliente SET cuentas = :1 WHERE DNI = :2'
    USING cuentas_aux, dniAsociar;

-- Ahora con la tabla Cuenta

EXECUTE IMMEDIATE
    'SELECT clientes FROM Cuenta WHERE IBAN = :1 FOR UPDATE OF clientes' INTO clientes_aux
USING ibanAsociar;

    SELECT ref(C) into cliente_ref FROM Cliente C WHERE C.DNI=dniAsociar;

    clientes_aux.EXTEND(1);
    clientes_aux(clientes_aux.LAST) := cliente_ref;

EXECUTE IMMEDIATE
    'UPDATE Cuenta SET clientes = :1 WHERE IBAN = :2'
    USING clientes_aux, ibanAsociar;

END relacionClienteCuenta;
/
```

2. PostgreSQL

Postgre difiere bastante en el estándar. No soporta tipos de referencia ni arrays de los mismos. Es por ello que las relaciones se han realizado con claves ajenas. Además, no se realiza un tipo y posteriormente una tabla tipada del mismo, sino que se hace directamente la tabla. Soporta la jerarquía de tablas por lo que las especializaciones del modelo Objeto/Relacional se han implementado gracias a la cláusula INHERITS.

En cuanto a los arrays de referencia, los incluye en versiones posteriores o a través de un parche.

```
/* ELIMINACION DE TABLAS */

DROP TABLE IF EXISTS Transferencia CASCADE;
DROP TABLE IF EXISTS Ingreso CASCADE;
DROP TABLE IF EXISTS Retirada CASCADE;
DROP TABLE IF EXISTS Posee CASCADE;
DROP TABLE IF EXISTS Cliente CASCADE;
```

```
DROP TABLE IF EXISTS Oficina CASCADE;
DROP TABLE IF EXISTS CuentaCorriente CASCADE;
DROP TABLE IF EXISTS CuentaAhorro CASCADE;
DROP TABLE IF EXISTS Cuenta CASCADE;
DROP TABLE IF EXISTS Operacion CASCADE;

/* CREACION DE TABLAS */

CREATE TABLE Cliente(
    DNI VARCHAR(10) PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    edad INTEGER NOT NULL,
    direccion VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefono VARCHAR(20)
);

CREATE TABLE Oficina(
    codigo VARCHAR(10) PRIMARY KEY,
    direccion VARCHAR(50) NOT NULL,
    telefono VARCHAR(20) NOT NULL
);

CREATE TABLE Cuenta(
    IBAN VARCHAR(150) PRIMARY KEY,
    numero INTEGER NOT NULL,
    saldo DECIMAL(10,2) DEFAULT 0.0,
    fechaCreacion DATE NOT NULL
);

CREATE TABLE Posee(
    DNI VARCHAR(10) NOT NULL,
    IBAN VARCHAR(150) NOT NULL,
    PRIMARY KEY (DNI,IBAN),
    FOREIGN KEY (DNI) REFERENCES Cliente(DNI) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE CuentaCorriente(
    oficina VARCHAR(10) REFERENCES Oficina(codigo) NOT NULL
) INHERITS (Cuenta);

CREATE TABLE CuentaAhorro(
    interes DECIMAL(3,2) NOT NULL
) INHERITS (Cuenta);

CREATE TABLE Operacion(
    codigo VARCHAR(150) NOT NULL,
    IBANOrigen VARCHAR(150) NOT NULL,
    fecha DATE NOT NULL,
    hora VARCHAR(8) NOT NULL,
    cantidad DECIMAL(10,2) NOT NULL,
    descripcion VARCHAR(400),
```

```
PRIMARY KEY (codigo, IBANOrigen),  
FOREIGN KEY (IBANOrigen) REFERENCES Cuenta(IBAN)  
);  
  
CREATE TABLE Transferencia(  
    IBANDestino VARCHAR(150) NOT NULL  
) INHERITS (Operacion);  
  
CREATE TABLE Ingreso(  
    oficina VARCHAR(15),  
    FOREIGN KEY (oficina) REFERENCES Oficina(codigo)  
) INHERITS (Operacion);  
  
CREATE TABLE Retirada(  
    oficina VARCHAR(15), FOREIGN KEY (oficina) REFERENCES Oficina(codigo)  
) INHERITS(Operacion);
```

Al insertar datos de prueba se ha tenido que eliminar la restricción de clave ajena ya que Postgre no inserta las tuplas de las tablas hijas en el padre, por lo que al asignar una clave ajena en el padre ésta no se encuentra. La consistencia se debe garantizar a través del uso de triggers.

3. DB2

DB2 no soporta arrays de referencia, pero sí tipos de referencia, por lo que la relación de Cliente-Cuenta se ha detallado como una nueva entidad de tipos de referencia. Además, permite jerarquía tanto de tablas como de tipos.

Este gestor no permite realizar utilizar restricciones de claves ajenas sobre UDTs, por lo que habrá que usar triggers para verificarlas. Además, los oid's de las tablas se insertan de forma manual teniendo en cuenta en que no coincidan dentro de la jerarquía de tablas de una tabla.

En el código siguiente, un aspecto que ha dado problemas ha sido la eliminación de tablas y tipos. Principalmente, porque después de cada ';' en DB2 en un procedimiento hay que poner la indicación de comentario '--', porque si no lo toma como el final del procedimiento.

```
-- DROP DE TABLES
```

```
BEGIN  
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'  
BEGIN END;--  
EXECUTE IMMEDIATE 'DROP TABLE Posee';--  
END;
```

```
BEGIN  
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'  
BEGIN END;--  
EXECUTE IMMEDIATE 'DROP TABLE Retirada';--  
END;
```

```
BEGIN  
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
```



```
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Ingreso';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Transferencia';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Operacion';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE CuentaCorriente';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE CuentaAhorro';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Oficina';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Cuenta';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TABLE Cliente';--
END;

-- DROP TYPES

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE PoseeUdt';--
END;
```

```
BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE RetiradaUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE IngresoUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE TransferenciaUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE OperacionUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE CuentaCorrienteUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE CuentaAhorroUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE OficinaUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE CuentaUdt';--
END;

BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
BEGIN END;--
EXECUTE IMMEDIATE 'DROP TYPE ClienteUdt';--
END;
```

-- CREACION DE TIPOS

```
CREATE TYPE ClienteUdt AS(  
    nombre    VARCHAR(100),  
    apellido  VARCHAR(100),  
    DNI       VARCHAR(10),  
    edad      INTEGER,  
    direccion VARCHAR(100),  
    email     VARCHAR(100),  
    telefono  VARCHAR(20)  
) MODE DB2SQL;
```

```
CREATE TYPE CuentaUdt AS(  
    IBAN VARCHAR(150),  
    numero INTEGER,  
    saldo DECIMAL(10,2),  
    fechaCreacion DATE  
) MODE DB2SQL;
```

```
CREATE TYPE PoseeUdt AS(  
    cliente ref(ClienteUdt),  
    cuenta  ref(CuentaUdt)  
) MODE DB2SQL;
```

-- en operacion el ibanOrigen debe ser el var para que sea clave primaria

```
CREATE TYPE OperacionUdt AS(  
    codigo VARCHAR(150),  
    ibanOrigen VARCHAR(150),  
    fecha DATE,  
    hora VARCHAR(8),  
    cantidad DECIMAL(10,2),  
    descripcion VARCHAR(400)  
) MODE DB2SQL;
```

```
CREATE TYPE OficinaUdt AS(  
    codigo VARCHAR(10),  
    direccion VARCHAR(50),  
    telefono VARCHAR(20)  
) MODE DB2SQL;
```

```
CREATE TYPE CuentaCorrienteUdt UNDER CuentaUdt AS(  
    oficina ref(OficinaUdt)  
) MODE DB2SQL;
```

```
CREATE TYPE CuentaAhorroUdt UNDER CuentaUdt AS(  
    interes DECIMAL(3,2)  
) MODE DB2SQL;
```

```
CREATE TYPE TransferenciaUdt UNDER OperacionUdt AS(  
    cuentaDestino ref(CuentaUdt)
```

```
) MODE DB2SQL;

CREATE TYPE IngresoUdt UNDER OperacionUdt AS(
    oficina ref(OficinaUdt)
) MODE DB2SQL;

CREATE TYPE RetiradaUdt UNDER OperacionUdt AS(
    oficina ref(OficinaUdt)
) MODE DB2SQL;

-- Creacion de tablas

CREATE TABLE Cliente OF ClienteUdt(
    REF IS oid USER GENERATED,
    nombre WITH OPTIONS NOT NULL,
    apellido WITH OPTIONS NOT NULL,
    DNI WITH OPTIONS NOT NULL PRIMARY KEY,
    edad WITH OPTIONS NOT NULL,
    direccion WITH OPTIONS NOT NULL,
    telefono WITH OPTIONS NOT NULL
);

CREATE TABLE Cuenta OF CuentaUdt(
    REF IS oid USER GENERATED,
    IBAN WITH OPTIONS NOT NULL PRIMARY KEY,
    numero WITH OPTIONS NOT NULL,
    saldo WITH OPTIONS NOT NULL,
    fechaCreacion WITH OPTIONS NOT NULL
);

CREATE TABLE Posee OF PoseeUdt(
    REF IS oid USER GENERATED,
    cliente WITH OPTIONS SCOPE Cliente,
    cuenta WITH OPTIONS SCOPE Cuenta
);

CREATE TABLE Oficina OF OficinaUdt(
    REF IS oid USER GENERATED,
    codigo WITH OPTIONS NOT NULL PRIMARY KEY,
    direccion WITH OPTIONS NOT NULL,
    telefono WITH OPTIONS NOT NULL
);

CREATE TABLE Operacion OF OperacionUdt(
    REF IS oid USER GENERATED,
    codigo WITH OPTIONS NOT NULL,
    ibanOrigen WITH OPTIONS NOT NULL,
    fecha WITH OPTIONS NOT NULL,
    hora WITH OPTIONS NOT NULL,
    cantidad WITH OPTIONS NOT NULL,
    PRIMARY KEY (codigo, ibanOrigen)
```

```
);

CREATE TABLE CuentaCorriente OF CuentaCorrienteUdt UNDER Cuenta INHERIT SELECT PRIVILEGES(
    oficina WITH OPTIONS SCOPE Oficina
);

CREATE TABLE CuentaAhorro OF CuentaAhorroUdt UNDER Cuenta INHERIT SELECT PRIVILEGES(
    interes WITH OPTIONS NOT NULL
);

CREATE TABLE Transferencia OF TransferenciaUdt UNDER Operacion INHERIT SELECT PRIVILEGES(
    cuentaDestino WITH OPTIONS SCOPE Cuenta
);

CREATE TABLE Ingreso OF IngresoUdt UNDER Operacion INHERIT SELECT PRIVILEGES(
    oficina WITH OPTIONS SCOPE Oficina
);

CREATE TABLE Retirada OF RetiradaUdt UNDER Operacion INHERIT SELECT PRIVILEGES(
    oficina WITH OPTIONS SCOPE Oficina
);
```

10. GENERACIÓN DE DATOS Y PRUEBAS

En cuanto a la generación de los datos se han contemplado diferentes herramientas:

1. MOCKAROO³

De las que se han analizado, el equipo se ha decantado por generar los datos con esta herramienta. La principal ventaja es que es gratuita y permite generar datos para diferentes tablas que estén relacionadas teniendo en cuenta la clave ajena. Además, posee aproximadamente 150 tipos de datos que se pueden generar y además altamente personalizables, entre ellos nombres, calles, IBAN, etc. Para la generación de los datos para la base de datos analizada en concreto, se han podido crear listas personalizadas para los tipos y se ha hecho uso de funciones para las tablas que se han especializado. Los datos generados se pueden exportar tanto en CSV como en sentencias SQL, por lo que se ha escogido ésta última para la exportación de los datos.

Una característica a destacar es que permite la creación de proyectos y éstos se pueden compartir entre diferentes personas, para trabajar de manera concurrente.

La principal desventaja de esta herramienta es que sólo permite generar 1000 filas, aunque se puede corregir concatenando los ficheros. Tampoco permite generación de claves únicas, por lo que se tiene que procesar de manera externa.

³ Mockaroo: <https://www.mockaroo.com/>

Uno de los principales problemas al trabajar con esta herramienta es que es algo complicado generar las tablas que se relacionan entre ellas con claves ajenas ya que hay que descargar primeramente el fichero de la tabla padre y después subirlo. Al ser una herramienta relativamente nueva posee este tipo de fallos.

The screenshot shows the 'molokardoo' web interface. At the top, there's a navigation bar with 'SCHEMAS 5', 'DATASETS 3', 'SCENARIOS', 'APIS', and 'PROJECTS'. Below this, the 'Cliente' section is active, with a 'Save Changes' button. The main area is a form to define fields for a dataset. It has columns for 'Field Name', 'Type', and 'Options'. Fields defined include: DNI (EIN), nombre (First Name), apellido (Last Name), edad (Number, min: 18, max: 85, decimals: 0), direccion (Street Address), email (Email Address), and telefono (Phone, format: ###-###-####). At the bottom, there are settings for '# Rows: 1000', 'Format: CSV', 'Line Ending: Unix (LF)', and 'Include: header' (checked) and 'BOM' (unchecked).

2. GENEREDATA <https://www.generatedata.com/>

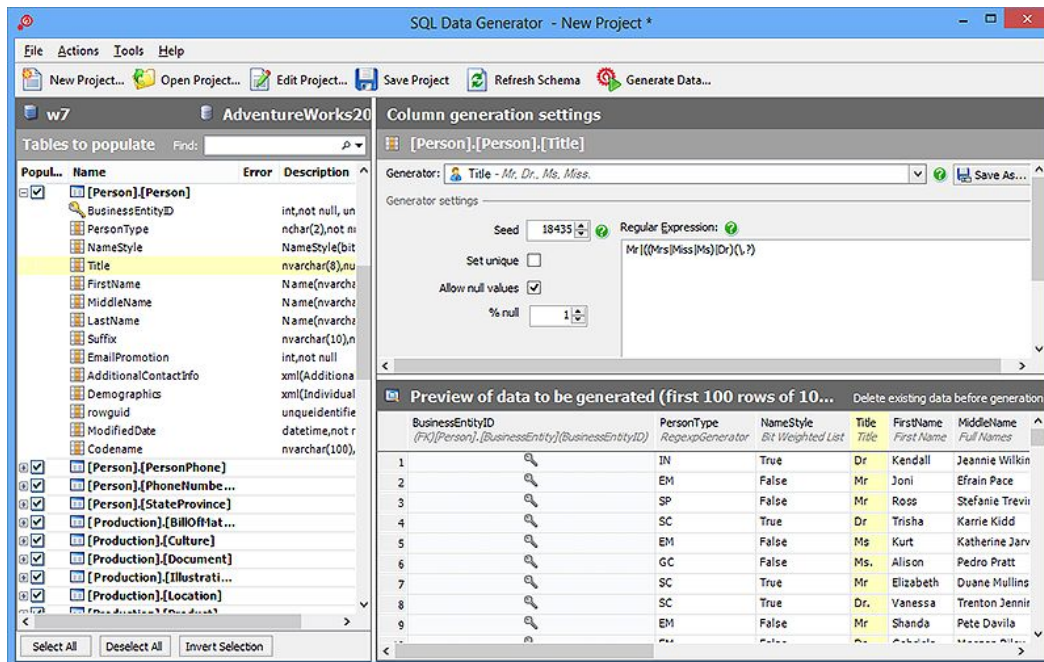
Herramienta similar a la anterior. Tiene muchos menos tipos de datos. Además, posee una interfaz más pobre. Al escoger entre esta herramienta y la anterior, se escoge la anterior ya que posee las mismas características e incluye nuevas.

The screenshot shows the 'generatedata.com' web interface. It has a top navigation bar with 'Generate', 'About', 'News', and 'Donate'. Below this, there's a 'Your data set name here...' field with a 'SAVE' button. The 'COUNTRY-SPECIFIC DATA' section has a dropdown for 'All countries'. The 'DATA SET' section is a table with columns: Order, Column Title, Data Type, Examples, Options, Help, and Del. It shows 4 rows, each with a 'Select Data Type' dropdown. Below the table, there's an 'Add 1 Row(s)' button. The 'EXPORT TYPES' section has tabs for CSV, Excel, HTML, JSON, LDIF, Programming Language, SQL, and XML. The 'Data format' section has radio buttons for '<table>', '', and '<dl>', and a checkbox for 'Use custom HTML format' with an 'edit' button.

3. SQL DATA GENERATOR

<https://www.red-gate.com/products/sql-development/sql-data-generator/>

Herramienta potente que permite generar datos para tablas relacionadas, no posee límite de datos a generar, puede trabajar con triggers, etc. Es capaz de integrarse con herramientas para poblar directamente la base de datos sin pasar por un fichero intermedio. Una ventaja frente a las anteriores herramientas es que permite guardar la semilla con la cual a generado los datos, para poder generar los mismos siempre que se desee. Ésto es de utilidad ya que en Mockaroo, por ejemplo, no se puede y hay que guardarse siempre el fichero si se quieren mantener los datos.

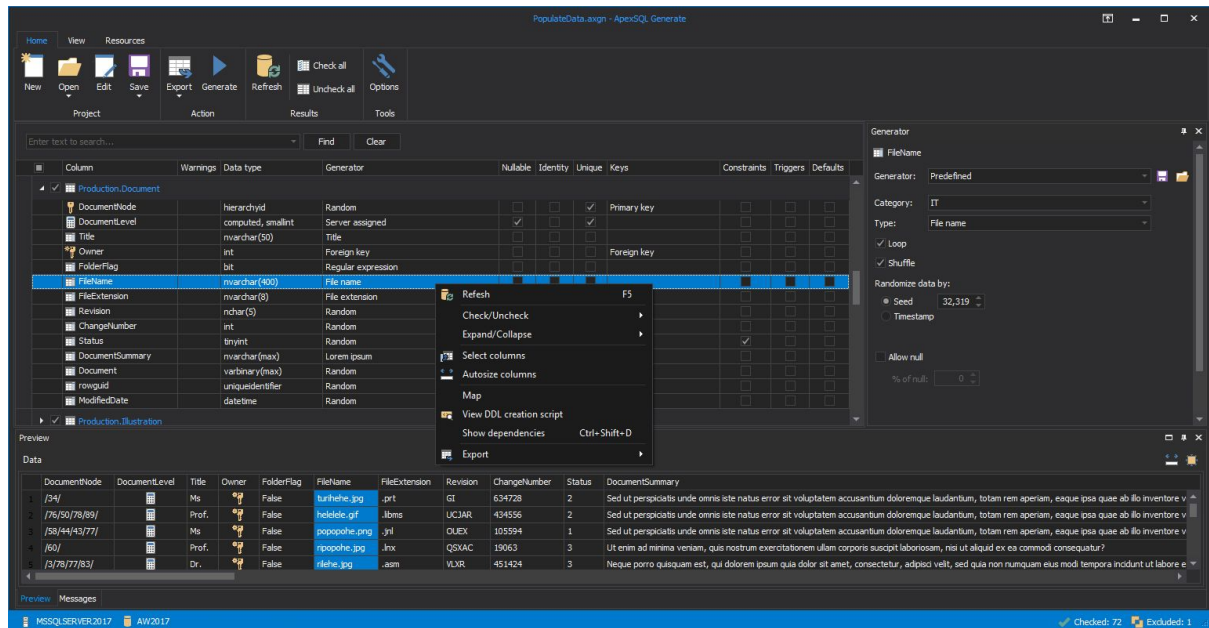


La desventaja de esta herramienta es que no es gratuita y tiene un precio de 325€.

4. APEXSQL GENERATE https://www.apexsql.com/sql_tools_generate.aspx

Herramienta muy similar a la anterior. Posee un gran catálogo de tipos para generar los datos que además simulan un comportamiento real de los datos. Está más enfocada a que la generación posea datos muy similares a los que se puedan encontrar en el mundo real. No posee límite para la generación de datos.

Este software posee una versión de prueba pero la completa cuesta entre 300\$ y 500\$ dependiendo de las características. En concreto, la versión de 500\$ incluye la generación de los datos de test mientras que la de 300\$ no.



1. Pruebas en Oracle Relacional

Los datos se han insertado a través de sentencias INSERT que directamente permiten ser exportadas por la herramienta de generación de datos. Las consultas probadas son las siguientes:

```
/*
Cantidad salida por transferencia de la primera cuenta que realizó una operación y tiene
cliente asociado.
*/
```

```
CREATE VIEW primeraCuenta AS
SELECT IBANOrigen FROM Operacion
WHERE fecha = (SELECT MIN(FECHA) FROM Operacion WHERE IBANORIGEN in (select iban from
Posee));
```

```
CREATE VIEW Clienteprim AS
SELECT Nombre, Apellido FROM Cliente
WHERE DNI IN (SELECT DNI FROM Posee WHERE IBAN IN (SELECT IBANORIGEN FROM primeraCuenta));
```

```
CREATE VIEW Ingre AS
SELECT SUM(cantidad) AS Ingresos FROM Operacion
WHERE IBANOrigen IN (SELECT IBANOrigen FROM primeraCuenta)
and tipo='transferencia';
```

```
SELECT C.Nombre, C.Apellido, I.Ingresos FROM Clienteprim C, Ingre I;
```

```
DROP VIEW primeraCuenta;
DROP VIEW Clienteprim;
DROP VIEW Ingre;
```

```
/*
Oficina de las 10 que menos cuentas tienen que registra más retiradas
*/
```



```

CREATE VIEW OficinaCuentas AS
SELECT Oficina, COUNT(IBAN) AS numCuentas FROM Cuenta
WHERE tipo = 'corriente'
GROUP BY Oficina;

CREATE VIEW topOficinas AS
SELECT Oficina, numCuentas FROM OficinaCuentas
ORDER BY numCuentas;

CREATE VIEW topDiezOficinas AS
SELECT Oficina, numCuentas FROM topOficinas
WHERE ROWNUM <=10;

CREATE VIEW topRetiradas AS
SELECT oficina, COUNT(codigo) AS numRetiradas FROM Operacion
WHERE tipo = 'retirada'
GROUP BY Oficina;

SELECT codigo, direccion FROM Oficina
WHERE (codigo IN (SELECT codigo FROM topOficinas)) AND
(codigo IN (SELECT codigo FROM topRetiradas WHERE numRetiradas = (SELECT Max(numRetiradas)
FROM topRetiradas))) AND ROWNUM <= 1;

DROP VIEW topRetiradas;
DROP VIEW topDiezOficinas;
DROP VIEW topOficinas;
DROP VIEW OficinaCuentas;

```

```

NOMBRE
-----
APELLIDO
-----
    INGRESOS
-----
Fredelia
Saulter
    11707.61

Robyn
McClintock
    11707.61

NOMBRE
-----
APELLIDO
-----
    INGRESOS
-----

```

```

CODIGO
-----
DIRECCION
-----
795
54916 Mayfield Drive

```

2. Pruebas en Oracle Objeto/Relacional

Debido a que hay que parsear los datos que genera la herramienta y adaptarlos para que encuentren los tipos de referencia, se han insertado algunos datos de prueba de forma manual y se han realizado algunas consultas para comprobar su integridad:

```
/* INSERCIÓN DE CLIENTES */
```

```
insert into Cliente (DNI, nombre, apellido, edad, direccion, email, telefono, cuentas)
values ('34-3480976', 'Conant', 'Lidgard', 24, '34 Lakewood Point', 'clidgard0@ox.ac.uk',
'4786467588', array_cuentas());
```

```
insert into Cliente (DNI, nombre, apellido, edad, direccion, email, telefono, cuentas)
values ('67-9251171', 'Wendye', 'Stuckey', 85, '464 Buhler Lane', 'wstuckey1@slate.com',
'9373548143', array_cuentas());
```

```
insert into Cliente (DNI, nombre, apellido, edad, direccion, email, telefono, cuentas)
values ('78-7995931', 'Annalise', 'Willcott', 53, '4 Ronald Regan Park', null,
'1984656758', array_cuentas());
```

```
insert into Cliente (DNI, nombre, apellido, edad, direccion, email, telefono, cuentas)
values ('92-5943193', 'Sanford', 'Leahey', 76, '94 La Follette Terrace',
'sleaheynv@phoca.cz', '1407316768', array_cuentas());
```

```
/* INSERCIÓN DE OFICINA */
```

```
insert into Oficina (codigo, direccion, telefono) values (795, '54916 Mayfield Drive',
'4655607686');
```

```
/* INSERCIÓN DE CUENTA */
```

```
-- CORRIENTE
```

```
insert into Cuenta VALUES (CuentaCorrienteUdt('DE33 7520 1499 2492 9685 27', 1060, 15392,
TO_DATE('1996-01-25', 'YYYY-MM-DD'), array_clientes(), (select ref(ofi) FROM Oficina ofi
WHERE ofi.codigo=795)));
```

```
insert into Cuenta VALUES (CuentaCorrienteUdt('FR77 2620 0309 16V3 KGNX B3PC N29', 1534,
8218, TO_DATE('2006-06-02', 'YYYY-MM-DD'), array_clientes(), (select ref(ofi) FROM
Oficina ofi WHERE ofi.codigo=795)));
```

```
-- AHORRO
```

```
insert into Cuenta VALUES (CuentaAhorroUdt('MD72 OY6E HSY7 0HZK AM0D PAB0', 572, 1590,
TO_DATE( '1974-02-28' , 'YYYY-MM-DD'), array_clientes(), 3.23));

insert into Cuenta VALUES (CuentaAhorroUdt('BR69 5012 8504 4863 3270 2788 435D T', 167,
9490, TO_DATE( '1991-09-25' , 'YYYY-MM-DD'), array_clientes(), 2.1));

/* INSERCIÓN DE OPERACIONES */

-- INGRESO

insert into Operacion VALUES (IngresoUdt(1, TO_DATE( '2014-05-06' , 'YYYY-MM-DD'), '2:24
PM', 2209.89, 'Activity, cheerleading', (select ref(cu) FROM Cuenta cu WHERE cu.IBAN='DE33
7520 1499 2492 9685 27'), (select ref(ofi) FROM Oficina ofi WHERE ofi.codigo=795)));

insert into Operacion VALUES (IngresoUdt(2, TO_DATE( '2014-05-03' , 'YYYY-MM-DD'), '1:24
PM', 302.89, 'Dinero para los estudios', (select ref(cu) FROM Cuenta cu WHERE cu.IBAN='BR69
5012 8504 4863 3270 2788 435D T'), (select ref(ofi) FROM Oficina ofi WHERE
ofi.codigo=795)));

-- RETIRADA

insert into Operacion VALUES (RetiradaUdt(3, 'DE33 7520 1499 2492 9685 27', TO_DATE(
'2003-05-06' , 'YYYY-MM-DD'), '6:40 PM', 100.99, 'Para un capricho', (select ref(cu) FROM
Cuenta cu WHERE cu.IBAN='DE33 7520 1499 2492 9685 27'), (select ref(ofi) FROM Oficina ofi
WHERE ofi.codigo=795)));

insert into Operacion VALUES (RetiradaUdt(4, 'FR77 2620 0309 16V3 KGNX B3PC N29', TO_DATE(
'2005-06-08' , 'YYYY-MM-DD'), '8:30 PM', 250.49, null, (select ref(cu) FROM Cuenta cu WHERE
cu.IBAN='FR77 2620 0309 16V3 KGNX B3PC N29'), (select ref(ofi) FROM Oficina ofi WHERE
ofi.codigo=795)));

-- TRANSFERENCIA

insert into Operacion VALUES (TransferenciaUdt(5, 'DE33 7520 1499 2492 9685 27', TO_DATE(
'2001-01-01' , 'YYYY-MM-DD'), '5:30 PM', 500.00, null, (select ref(cu) FROM Cuenta cu WHERE
cu.IBAN='FR77 2620 0309 16V3 KGNX B3PC N29'), (select ref(cu) FROM Cuenta cu WHERE
cu.IBAN='DE33 7520 1499 2492 9685 27')));

insert into Operacion VALUES (TransferenciaUdt(6, 'MD72 OY6E HSY7 0HZK AM0D PAB0', TO_DATE(
'2001-01-02' , 'YYYY-MM-DD'), '4:30 PM', 550.00, null, (select ref(cu) FROM Cuenta cu WHERE
cu.IBAN='FR77 2620 0309 16V3 KGNX B3PC N29'), (select ref(cu) FROM Cuenta cu WHERE
cu.IBAN='MD72 OY6E HSY7 0HZK AM0D PAB0')));

/* RELACIONES CLIENTE CUENTA PARA PROBAR EL PROCEDIMIENTO */

CALL relacionClienteCuenta('34-3480976', 'DE33 7520 1499 2492 9685 27');

CALL relacionClienteCuenta('34-3480976', 'MD72 OY6E HSY7 0HZK AM0D PAB0');

/* PRUEBAS */
```

```
-- Habilitamos la salida estandar
SET SERVEROUTPUT ON;

-- Mostramos el IBAN de las cuentas asociadas a 34-3480976

DECLARE
    iban_cuenta VARCHAR(150);
    cuenta_ref_var REF CuentaUdt;
    cuentas_ref_var array_cuentas;
BEGIN
    SELECT c.cuentas INTO cuentas_ref_var FROM Cliente c WHERE DNI = '34-3480976';

    FOR i IN cuentas_ref_var.FIRST .. cuentas_ref_var.LAST LOOP
        cuenta_ref_var := cuentas_ref_var(i);
        SELECT IBAN INTO iban_cuenta FROM Cuenta c WHERE ref(c) = cuenta_ref_var;
        DBMS_OUTPUT.PUT_LINE('IBAN: ' || iban_cuenta);
    END LOOP;
END;
/

-- Miramos si estan los de tipo operacion

SELECT TREAT(VALUE(o) AS RetiradaUdt)
FROM Operacion o;
```

```
IBAN: DE33 7520 1499 2492 9685 27
IBAN: MD72 OY6E HSY7 0HZK AM0D PAB0
IBAN: DE33 7520 1499 2492 9685 27
IBAN: MD72 OY6E HSY7 0HZK AM0D PAB0
IBAN: DE33 7520 1499 2492 9685 27
IBAN: MD72 OY6E HSY7 0HZK AM0D PAB0

PL/SQL procedure successfully completed.

TREAT(VALUE(O)ASRETIRADAUDT)(CODIGO, IBANORIGEN, FECHA, HORA, CANTIDAD, DESCRIPC
-----
RETIRADAUDT('3', 'DE33 7520 1499 2492 9685 27', '06-MAY-03', '6:40 PM', 100.99,
'Para un capricho', 000022020868C96BAFF4C9AAABE050007F0101108E68C96BAFF4C1AAABE0
50007F0101108E, 000022020868C96BAFF4C8AAABE050007F0101108E68C96BAFF4C2AAABE05000
7F0101108E)

RETIRADAUDT('4', 'FR77 2620 0309 16V3 KGNX B3PC N29', '08-JUN-05', '8:30 PM', 25
0.49, NULL, 000022020868C96BAFF4CAAAABE050007F0101108E68C96BAFF4C1AAABE050007F01
01108E, 000022020868C96BAFF4C8AAABE050007F0101108E68C96BAFF4C2AAABE050007F010110
8E)

TREAT(VALUE(O)ASRETIRADAUDT)(CODIGO, IBANORIGEN, FECHA, HORA, CANTIDAD, DESCRIPC
-----
```

3. Pruebas en PostgreSQL Objeto/Relacional

Como los datos se asemejan bastante al del modelo relacional, se han podido generar los datos con la herramienta Mockaroo. Se ha realizado la siguiente consulta de prueba y se comprueba que los datos han sido insertados correctamente en la base de datos:

```
/*
Oficina de las 10 que menos cuentas tienen que registra más retiradas
*/

CREATE VIEW OficinaCuentas AS
SELECT Oficina, COUNT(IBAN) AS numCuentas FROM ONLY CuentaCorriente
GROUP BY Oficina;

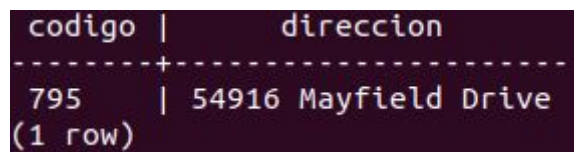
CREATE VIEW topOficinas AS
SELECT Oficina, numCuentas FROM OficinaCuentas
ORDER BY numCuentas;

CREATE VIEW topDiezOficinas AS
SELECT Oficina, numCuentas FROM topOficinas
limit 10;

CREATE VIEW topRetiradas AS
SELECT oficina, COUNT(codigo) AS numRetiradas FROM ONLY Retirada
GROUP BY Oficina;

SELECT codigo, direccion FROM Oficina
WHERE (codigo IN (SELECT codigo FROM topOficinas)) AND
(codigo IN (SELECT codigo FROM topRetiradas WHERE numRetiradas = (SELECT Max(numRetiradas)
FROM topRetiradas)))
limit 1;

DROP VIEW OficinaCuentas;
DROP VIEW topOficinas;
DROP VIEW topDiezOficinas;
DROP VIEW topRetiradas;
```



codigo	direccion
795	54916 Mayfield Drive

(1 row)

4. Pruebas en DB2 Objeto/Relacional

Debido a que hay que parsear los datos que genera la herramienta y adaptarlos para que encuentren los tipos de referencia, se han insertado algunos datos de prueba de forma manual y se han realizado algunas consultas para comprobar su integridad. Un aspecto interesante de DB2 es que permite acceder a los datos de tipo referencia como si de un puntero en cualquier lenguaje de programación se tratara y poder acceder a sus atributos sin tener que buscarlo antes en la tabla.

-- INSERCIÓN CLIENTES

```
insert into Cliente (oid, nombre, apellido, DNI, edad, direccion, telefono) VALUES
(ClienteUdt('1'), 'Conant', 'Lidgard', '34-3480976', 24, '34 Lakewood Point',
'4786467588');
```

```
insert into Cliente (oid, DNI, nombre, apellido, edad, direccion, email, telefono) values
(ClienteUdt('2'), '67-9251171', 'Wendye', 'Stuckey', 85, '464 Buhler Lane',
'wstuckey1@slate.com', '9373548143');
```

```
insert into Cliente (oid, DNI, nombre, apellido, edad, direccion, email, telefono) values
(ClienteUdt('3'),'78-7995931', 'Annalise', 'Willcott', 53, '4 Ronald Regan Park', null,
'1984656758');
```

```
insert into Cliente (oid, DNI, nombre, apellido, edad, direccion, email, telefono) values
(ClienteUdt('4'),'92-5943193', 'Sanford', 'Leahey', 76, '94 La Follette Terrace',
'sleaheynv@phoca.cz', '1407316768');
```

-- INSERCIÓN DE OFICINA

```
insert into Oficina (oid, codigo, direccion, telefono) values (OficinaUdt('1'), 795, '54916
Mayfield Drive', '4655607686');
```

-- INSERCIÓN CUENTA AHORRO

```
insert into CuentaAhorro (oid, IBAN, numero, saldo, fechaCreacion, interes) values
(CuentaAhorroUdt('1'),'MD72 0Y6E HSY7 0HZK AM0D PAB0', 572, 1590, '1974-02-28', 3.23);
```

```
insert into CuentaAhorro (oid, IBAN, numero, saldo, fechaCreacion, interes) values
(CuentaAhorroUdt('2'),'BR69 5012 8504 4863 3270 2788 435D T', 167, 9490, '1991-09-25',
2.1);
```

-- INSERCIÓN CUENTA CORRIENTE

```
insert into CuentaCorriente (oid, IBAN, numero, saldo, fechaCreacion, oficina) values
(CuentaCorrienteUdt('3'),'DE33 7520 1499 2492 9685 27', 1060, 15392, '1996-01-25', (select
oid FROM Oficina WHERE codigo=795));
```

```
insert into CuentaCorriente (oid, IBAN, numero, saldo, fechaCreacion, oficina) values
(CuentaCorrienteUdt('4'),'FR77 2620 0309 16V3 KGNX B3PC N29', 1534, 8218, '2006-06-02',
(select oid FROM Oficina WHERE codigo=795));
```

-- INSERCIÓN DE OPERACIONES

-- INGRESO

```
insert into Ingreso (oid, codigo, ibanOrigen, fecha, hora, cantidad, descripcion, oficina)
VALUES (IngresoUdt('1'), 1, 'DE33 7520 1499 2492 9685 27', '2014-05-06', '2:24 PM',
2209.89, 'Activity, cheerleading', (select oid FROM Oficina WHERE codigo=795));
```

```
insert into Ingreso (oid, codigo, ibanOrigen, fecha, hora, cantidad, descripcion, oficina)
VALUES (IngresoUdt('2'), 2, 'BR69 5012 8504 4863 3270 2788 435D T', '2014-05-03', '1:24
PM', 302.89, 'Dinero para los estudios', (select oid FROM Oficina WHERE codigo=795));
```

-- RETIRADA

```
insert into Retirada (oid, codigo, ibanOrigen, fecha, hora, cantidad, descripcion, oficina)
VALUES (RetiradaUdt('3'), 3, 'DE33 7520 1499 2492 9685 27', '2003-05-06', '6:40 PM',
100.99, 'Para un capricho', (select oid FROM Oficina WHERE codigo=795));
```



```
insert into Retirada (oid, codigo, ibanOrigen, fecha, hora, cantidad, oficina) VALUES
(RetiradaUdt('4'), 4, 'FR77 2620 0309 16V3 KGNX B3PC N29', '2001-01-01', '8:30 PM', 250.49,
(select oid FROM Oficina WHERE codigo=795));
```

```
-- TRANSFERENCIA
```

```
insert into Transferencia (oid, codigo, ibanOrigen, fecha, hora, cantidad, cuentaDestino)
VALUES (TransferenciaUdt('5'), 5, 'FR77 2620 0309 16V3 KGNX B3PC N29', '2001-02-02', '9:30
PM', 550.49, (select oid FROM Cuenta WHERE IBAN='DE33 7520 1499 2492 9685 27'));
```

```
insert into Transferencia (oid, codigo, ibanOrigen, fecha, hora, cantidad, cuentaDestino)
VALUES (TransferenciaUdt('6'), 6, 'FR77 2620 0309 16V3 KGNX B3PC N29', '2002-03-03', '10:30
PM', 5500.49, (select oid FROM Cuenta WHERE IBAN='MD72 OY6E HSY7 0HZK AM0D PAB0'));
```

```
-- POSEE
```

```
insert into Posee (oid, cliente, cuenta) VALUES (PoseeUdt('1'), (select oid FROM Cliente
WHERE DNI='34-3480976'), (select oid FROM Cuenta WHERE IBAN='MD72 OY6E HSY7 0HZK AM0D
PAB0'));
```

```
insert into Posee (oid, cliente, cuenta) VALUES (PoseeUdt('2'), (select oid FROM Cliente
WHERE DNI='34-3480976'), (select oid FROM Cuenta WHERE IBAN='BR69 5012 8504 4863 3270 2788
435D T'));
```

```
insert into Posee (oid, cliente, cuenta) VALUES (PoseeUdt('3'), (select oid FROM Cliente
WHERE DNI='34-3480976'), (select oid FROM Cuenta WHERE IBAN='FR77 2620 0309 16V3 KGNX B3PC
N29'));
```

```
----- PRUEBAS -----
```

```
SELECT DISTINCT T.codigo, T.ibanOrigen, T.cuentaDestino->saldo FROM Transferencia T, Cuenta
C WHERE T.cuentaDestino->fechaCreacion > '1991-09-25';
```

```
-- No puedes acceder a los campos de los subtipos a traves del padre
```

```
SELECT DISTINCT C.oficina->codigo FROM Posee P, CuentaCorriente C WHERE P.cuenta->saldo>8000 AND C.IBAN = P.cuenta->IBAN;
```

```
CODIGO
      IBANORIGEN
      SALDO
-----
5
      FR77 2620 0309 16V3 KGNX B3PC N29
      15392.00

1 record(s) selected.

SELECT DISTINCT C.oficina->codigo FROM Posee P, CuentaCorriente C WHERE P.cuenta->saldo>8000 AND C.IBAN = P.cuenta->IBAN
CODIGO
-----
795

1 record(s) selected.
```

11. IMPLEMENTACIÓN CON DB4O

El sistema gestor de bases de datos db4o trabaja guardando la información de los datos en un fichero que es tratado como un container en el código fuente. Escrito en java y c#, cada tabla del modelo lógico es representada como una clase en java. Este sistema gestor posee una licencia gratuita de tipo GPL.

Las relaciones están representadas como listas, estando en el objeto de la parte 1 en el caso de las relaciones 1:N y en ambos objetos en las relaciones N:N, como en el caso de las clases cliente y cuenta. La herencia es tratada como se trata la herencia de clases en java, extendiendo la clase padre.

Los principales problemas de este gestor son la volatilidad del fichero donde se almacena la información de los datos almacenados en la base de datos. Además, no fuerza la coherencia de las listas de un objeto con otros objetos del sistema. Por ejemplo, se podría eliminar un objeto del container sin que este fuera eliminado de la relación de otro objeto y pudiendo ser accedido desde él cuando ya no existe.

A continuación se expone el código de las clases en lenguaje java que representan cada una de las tablas de la base de datos:

```
public class Cliente {

    private String nombre;
    private String telefono;
    private String apellidos;
    private String DNI;
    private int edad;
    private String email;
    private String direccion;
    private List<Cuenta> cuentas;

    public Cliente(String name, String telefono, String apellidos, String DNI,int edad,
String direccion) {
        this.nombre=name;
        this.telefono=telefono;
        this.apellidos=apellidos;
        this.DNI=DNI;
        this.edad=edad;
        this.direccion=direccion;
        this.cuentas=new ArrayList();
    }

    public void setNombre(String name){ this.nombre=name;}
    public String getNombre(){return this.nombre;}

    public void setTelefono(String tfno){ this.telefono=tfno;}
    public String getTelefono(){return this.telefono;}
```



```

public void setApellidos(String apellidos){ this.apellidos=apellidos;}
public String getApellidos(){return this.apellidos;}

public void setDNI(String DNI){ this.DNI=DNI;}
public String getDNI(){return this.DNI;}

public void setEdad(int edad){ this.edad=edad;}
public int getEdad(){return this.edad;}

public void setDireccion(String direccion){ this.direccion=direccion;}
public String getDireccion(){return this.direccion;}

public void setEmail(String email){ this.email=email;}
public String getEmail(){return this.email;}

public List getCuentas(){ return this.cuentas; }
public void setCuentas( List l ){ this.cuentas=l; }

public void addCuenta(Cuenta c){this.cuentas.add(c);}

public String toString() {
    return nombre+" "+apellidos+" - "+DNI;
}
}

import java.util.*;

public class Cuenta
{
    protected String IBAN;
    protected int numero;
    protected double saldo;
    protected String fechaCreacion;
    protected List<Cliente> clientes;
    protected List<Operacion> operaciones;
    protected List<Transferencia> recibidas;

    public Cuenta(String IBAN, int numero, double saldo, String fechaCreacion)
    {
        this.IBAN=IBAN;
        this.numero=numero;
        this.saldo=saldo;
        this.fechaCreacion=fechaCreacion;
        this.clientes=new ArrayList();
        this.operaciones=new ArrayList();
        this.recibidas=new ArrayList();
    }

    public String getIBAN(){ return this.IBAN; }
    public void setIBAN( String IBAN ){ this.IBAN=IBAN; }

    public int getNumero(){ return this.numero; }

```

```

    public void setNumero( int numero ){ this.numero=numero; }

    public double getSaldo(){ return this.saldo; }
    public void setSaldo( double saldo ){ this.saldo=saldo; }

    public String getFechaCreacion(){ return this.fechaCreacion; }
    public void setFechaCreacion( String fecha ){ this.fechaCreacion=fecha; }

    public List getClientes(){ return this.clientes; }
    public void setClientes( List l ){ this.clientes=l; }

    public List getOperaciones(){ return this.operaciones; }
    public void setOperaciones( List l ){ this.operaciones=l; }

    public List getRecibidas(){ return this.recibidas; }
    public void setRecibidas( List l ){ this.recibidas=l; }

    public void addCliente( Cliente c ){ this.clientes.add(c); }

    public void addOperacion( Operacion o ){ this.operaciones.add(o); }

    public void addRecibida( Transferencia t ){ this.recibidas.add(t); }

    public String toString() {
        return IBAN+" -> "+saldo;
    }
}

public class Corriente extends Cuenta
{
    Oficina of;

    public Corriente(Oficina of, String IBAN, int numero, double saldo, String
fechaCreacion)
    {
        super(IBAN,numero,saldo,fechaCreacion);
        this.of=of;
    }

    public Oficina getCorriente(){ return this.of; }
    public void setCorriente ( Oficina of ){ this.of=of; }

    public String toString() {
        return super.IBAN+" -> "+super.saldo+" -> "+of.toString();
    }
}

public class Ahorro extends Cuenta
{
    double interes;

```

```

    public Ahorro(double interes, String IBAN, int numero, double saldo, String
fechaCreacion)
    {
        super(IBAN,numero,saldo,fechaCreacion);
        this.interes=interes;
    }

    public double getInteres(){ return this.interes; }
    public void setInteres( double interes ){ this.interes=interes; }

    public String toString() {
        return super.IBAN+" -> "+super.saldo+" -> "+interes+"%";
    }
}

```

```

public class Operacion
{
    private String codigo;
    private String fecha;
    private String hora;
    public double cantidad;
    private String descripcion;
    private Cuenta c;

    public Operacion(String codigo, String fecha, String hora, double cantidad, Cuenta
c)
    {
        this.codigo=codigo;
        this.fecha=fecha;
        this.hora=hora;
        this.cantidad = cantidad;
        this.c = c;
    }

    public String getCodigo(){ return this.codigo; }
    public void setCodigo( String codigo ){ this.codigo=codigo; }

    public String getFecha(){ return this.fecha; }
    public void setFecha( String fecha ){ this.fecha=fecha; }

    public String getHora(){ return this.hora; }
    public void setHora( String hora ){ this.hora=hora; }

    public double getCantidad(){ return this.cantidad; }
    public void setCantidad( double c ){ this.cantidad=c; }

    public String getDescripcion(){ return this.descripcion; }
    public void setDescripcion( String des ){ this.descripcion=des; }

    public Cuenta getCuentaOrigen(){ return this.c; }
    public void setCuentaOrigen( Cuenta c ){ this.c=c; }
}

```

```

        public String toString() {
            return codigo;
        }
    }

    public class IngresoRetirada extends Operacion
    {
        private Oficina of;

        public IngresoRetirada(String codigo, String fecha, String hora, double cantidad,
Cuenta c, Oficina of)
        {
            super(codigo, fecha, hora, cantidad, c);
            this.of=of;
        }

        public Oficina getOficina(){ return this.of; }
        public void setOficina( Oficina of ){ this.of=of; }

        public String toString() {
            return super.getCodigo()+" ingreso o retirada en " + of.toString();
        }
    }

    public class Transferencia extends Operacion
    {
        public Cuenta cuentaDest;

        public Transferencia(String codigo, String fecha, String hora, double cantidad,
Cuenta c, Cuenta cuentaDest)
        {
            super(codigo, fecha, hora, cantidad, c);
            this.cuentaDest=cuentaDest;
        }

        public Cuenta getCuentaDestino(){ return this.cuentaDest; }
        public void setCuentaDestino( Cuenta c ){ this.cuentaDest=c; }

        public String toString() {
            return super.getCodigo()+" transferencia de " +
super.getCuentaOrigen().toString() + " a " +cuentaDest.toString();
        }
    }

    import java.util.*;

    public class Oficina
    {

```

```
private String codigo;
private String direccion;
private String telefono;
private List<Corriente> corrientes;
private List<IngresoRetirada> ops;

public Oficina(String codigo, String direccion, String telefono)
{
    this.codigo=codigo;
    this.direccion=direccion;
    this.telefono=telefono;
    this.corrientes = new ArrayList();
    this.ops = new ArrayList();
}

public String getCodigo(){ return this.codigo; }
public void setCodigo( String codigo ){ this.codigo=codigo; }

public String getDireccion(){ return this.direccion; }
public void setDireccion( String direccion ){ this.direccion=direccion; }

public String getTelefono(){ return this.telefono; }
public void setTelefono( String telefono ){ this.telefono=telefono; }

public List getCorrientes(){ return this.corrientes; }
public void setCorrientes( List l ){ this.corrientes=l; }

public List getOps(){ return this.ops; }
public void setOps( List l ){ this.ops=l; }

public void addCorriente( Corriente c ){ this.corrientes.add(c); }

public void addOps( IngresoRetirada ir ){ this.ops.add(ir); }

public String toString() {
    return direccion;
}

}
```

5. Pruebas en DB4O

Para db4o se han ejecutado distintos tipos de inserciones, consultas y borrados de cada una de las distintas tablas y de la relación entre cuenta y cliente. Para ejecutar las pruebas se debe ejecutar el script Compile.sh y a continuación el script Example1.sh. El código de pruebas es el que se muestra a continuación:

```
/**
 *           Example           extracted           and           adapted           from
http://community.versant.com/documentation/reference/db4o-8.0/java/tutorial/docs/FirstGlanc
e.html
 * Date: March 15, 2014
 */

import java.io.*;
```

```
import com.db4o.*;

public class Example1 extends Util {

    final static String DB_FOLDER = "./DB-FILES";
    //final static String DB_FOLDER = System.getProperty("user.home");

    final static String DB_FILE = "Banco.db4o";

    final static String DB4OFILENAME = DB_FOLDER + "/" + DB_FILE;

    public static void main(String[] args) {
        new File(DB4OFILENAME).delete();
        accessDb4o();
        new File(DB4OFILENAME).delete();
        ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded
            .newConfiguration(), DB4OFILENAME);
        try {
            System.out.println("-----CLIENTE-----");
            storeFirstCliente(db);
            storeSecondCliente(db);
            retrieveAllClients(db);
            retrieveClientByName(db);
            retrieveClientByDNI(db);
            updateClient(db);
            deleteFirstClientByName(db);
            System.out.println("-----Cuenta-----");
            storeFirstCuenta(db);
            storeSecondCuenta(db);
            retrieveAllAccounts(db);
            retrieveAccountByIBAN(db);
            retrieveAccountByNumCuenta(db);
            updateAccount(db);
            deleteFirstAccountByIBAN(db);
            System.out.println("-----Relacion Cuenta-Cliente-----");
            storeFirstCliente(db);
            storeSecondCliente(db);
            storeFirstCuenta(db);
            anyadirClientesAFirstCuenta(db);
            anyadirCuentasAFirstCliente(db);
            retrieveClientesFromCuenta(db);
            retrieveCuentasFromCliente(db);
            System.out.println("-----Operacion-----");
            storeFirstCuenta(db);
            storeFirstOficina(db);
            storeOperacionTrasferencia(db);
            storeOperacionIngresoRetirada(db);
            retrieveAllOperations(db);
            retrieveOperationByCodigo(db);
            retrieveOperationBySaldo(db);
            retrieveOficinaDeOperacionRetirada(db);
            updateOperation(db);
            deleteFirstOperacionByCodigo(db);
            System.out.println("-----Oficina-----");
```

```

        storeFirstOficina(db);
        storeSecondOficina(db);
        anyadirCuentaAFirstOficina(db);
        retrieveCuentasFromOficina(db);
    } finally {
        db.close();
    }
}

public static void accessDb4o() {
    ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded
        .newConfiguration(), DB4OFILENAME);
    try {
        // do something with db4o
    } finally {
        db.close();
    }
}

public static void storeFirstCliente(ObjectContainer db) {
    Cliente client1 = new Cliente("Carlos", "612345678", "Maranyes Nuevo", "12345678A",
        25, "Calle de la cabra, Muel");
    db.store(client1);
    System.out.println("Stored cliente" + client1);
}

public static void storeSecondCliente(ObjectContainer db) {
    Cliente client2 = new Cliente("Nicolas", "612345678", "Lera Lopez", "87654321B",
        25, "Calle de la mala gente, Jaca");
    db.store(client2);
    System.out.println("Stored cliente" + client2);
}

public static void retrieveAllClientQBE(ObjectContainer db) {
    Cliente proto = new Cliente(null, null, null, null, 0, null);
    ObjectSet result = db.queryByExample(proto);
    listResult(result);
}

public static void retrieveAllClients(ObjectContainer db) {
    ObjectSet result = db.queryByExample(Cliente.class);
    listResult(result);
}

public static void retrieveClientByName(ObjectContainer db) {
    Cliente client1 = new Cliente("Nicolas", null, null, null, 0, null);
    ObjectSet result = db.queryByExample(client1);
    listResult(result);
}

public static void retrieveClientByDNI(ObjectContainer db) {
    Cliente client1 = new Cliente(null, null, null, "12345678A", 0, null);
    ObjectSet result = db.queryByExample(client1);
    listResult(result);
}

```

```
}

public static void updateClient(ObjectContainer db) {
    ObjectSet result = db
        .queryByExample(new Cliente("Nicolas", null, null,null, 0,null));
    Cliente found = (Cliente) result.next();
    found.setNombre("Anacardo");
    db.store(found);
    System.out.println("Ahora se llama" + found);
    retrieveAllClients(db);
}

public static void deleteFirstClientByName(ObjectContainer db) {
    ObjectSet result = db
        .queryByExample(new Cliente("Carlos", null, null,null, 0,null));
    Cliente found = (Cliente) result.next();
    db.delete(found);
    System.out.println("Deleted cliente " + found);
    retrieveAllClients(db);
}

/* AQUI EMPIEZAN LAS PRUEBAS DE LOS DOS TIPOS DE CUENTA */

public static void storeFirstCuenta(ObjectContainer db) {
    Cuenta Account1 = new Ahorro(0.05, "1234-5678-ABCD", 12345678, 20000.01,
"25/12/0");
    db.store(Account1);
    System.out.println("Stored Ahorro" + Account1);
}

public static void storeSecondCuenta(ObjectContainer db) {
    Oficina of = new Oficina("1234-SAN", "Calle Marujas ZGZ", "976123456");
    Cuenta Account2 = new Corriente(of, "5678-1234-EFGH", 56781234, 10000.15,
"1/1/2000");
    db.store(Account2);
    System.out.println("Stored Corriente" + Account2);
}

public static void retrieveAllAccountQBE(ObjectContainer db) {
    Cuenta proto = new Cuenta("1234-5678-ABCD", 0,0.0,null);
    ObjectSet result = db.queryByExample(proto);
    listResult(result);
}

public static void retrieveAllAccounts(ObjectContainer db) {
    ObjectSet result = db.queryByExample(Cuenta.class);
    listResult(result);
}

public static void retrieveAccountByIBAN(ObjectContainer db) {
    Cuenta Account1 = new Cuenta("5678-1234-EFGH", 0, 0.0,null);
    ObjectSet result = db.queryByExample(Account1);
    listResult(result);
}
```



```
}

public static void retrieveAccountByNumCuenta(ObjectContainer db) {
    Cuenta Account1 = new Cuenta(null,56781234 ,0,null);
    ObjectSet result = db.queryByExample(Account1);
    listResult(result);
}

public static void updateAccount(ObjectContainer db) {
    ObjectSet result = db
        .queryByExample(new Cuenta("5678-1234-EFGH", 0, 0.0,null));
    Cuenta found = (Cuenta) result.next();
    found.setSaldo(20000000.01);
    db.store(found);
    System.out.println("Ahora se llama anacardo" + found);
    retrieveAllAccounts(db);
}

public static void deleteFirstAccountByIBAN(ObjectContainer db) {
    ObjectSet result = db
        .queryByExample(new Cuenta("1234-5678-ABCD", 0, 0.0,null));
    Cuenta found = (Cuenta) result.next();
    db.delete(found);
    System.out.println("Deleted cuenta" + found);
    retrieveAllAccounts(db);
}

/* AQUI EMPIEZAN LAS PRUEBAS DE LA INTERRELACION CUENTA-cCLIENTE */
public static void anyadirClientesAFirstCuenta(ObjectContainer db){
    ObjectSet result = db
        .queryByExample(new Cuenta("1234-5678-ABCD", 0, 0.0,null));
    Cuenta found = (Cuenta) result.next();
    result = db
        .queryByExample(new Cliente("Nicolas", null, null,null, 0,null));
    Cliente c1 = (Cliente) result.next();
    found.addCliente(c1);
    result = db
        .queryByExample(new Cliente("Carlos", null, null,null, 0,null));
    Cliente c2 = (Cliente) result.next();
    found.addCliente(c2);
    db.store(found);
    System.out.println("Añadido " + c1 + " a " + found);
    System.out.println("Añadido " + c2 + " a " + found);
    retrieveAllAccounts(db);
}

public static void anyadirCuentasAFirstCliente(ObjectContainer db){
    ObjectSet result = db
        .queryByExample(new Cliente("Nicolas", null, null,null, 0,null));
    Cliente found = (Cliente) result.next();
    result = db
        .queryByExample(new Cuenta("1234-5678-ABCD", 0, 0.0,null));
    Cuenta c1 = (Cuenta) result.next();
    found.addCuenta(c1);
}
```

```
        result = db
            .queryByExample(new Cuenta("5678-1234-EFGH", 0, 0.0,null));
        Cuenta c2 = (Cuenta) result.next();
        found.addCuenta(c2);
        db.store(found);
        System.out.println("Añadido " + c1 + " a " + found);
        System.out.println("Añadido " + c2 + " a " + found);
        retrieveAllAccounts(db);
    }

    public static void retrieveClientesFromCuenta(ObjectContainer db) {
        Cuenta Account1 = new Cuenta("1234-5678-ABCD", 0, 0.0,null);
        ObjectSet result = db.queryByExample(Account1);
        Cuenta c = (Cuenta) result.next();
        System.out.println("Clientes de " + c.getIBAN());
        listResult(c.getClientes());
    }

    public static void retrieveCuentasFromCliente(ObjectContainer db) {
        Cliente Client1 = new Cliente("Nicolas", null, null,null, 0,null);
        ObjectSet result = db.queryByExample(Client1);
        Cliente c = (Cliente) result.next();
        System.out.println("Cuentas de " + c.getNombre() + " " + c.getApellidos());
        listResult(c.getCuentas());
    }

    /* AQUI EMPIEZAN LAS PRUEBAS DE LOS DOS TIPOS DE OPERACION */

    public static void storeOperacionIngresoRetirada(ObjectContainer db) {
        ObjectSet result = db
            .queryByExample( new Oficina("1234-SAN", null, null));
        Oficina of = (Oficina) result.next();
        Corriente c = new Corriente(null, "5678-1234-EFGH", 0, 0, null);
        Operacion op = new IngresoRetirada("abcde", "1/1/80", "00:01", 1050.05, c, of);
        db.store(op);
        System.out.println("Stored Operacion" + op);
    }

    public static void storeOperacionTrasferencia(ObjectContainer db) {
        ObjectSet result = db
            .queryByExample(new Cuenta("1234-5678-ABCD", 0, 0.0,null));
        Cuenta c = (Cuenta) result.next();
        result = db
            .queryByExample(new Cuenta("5678-1234-EFGH", 0, 0.0,null));
        Cuenta c2 = (Cuenta) result.next();
        Operacion op = new Transferencia("fghij", "2/1/80", "22:00", 500.50, c, c2);
        db.store(op);
        System.out.println("Stored Transferecia " + op + " entre " + c + " y " + c2);
    }

    public static void retrieveAllOperations(ObjectContainer db) {
        ObjectSet result = db.queryByExample(Operacion.class);
        listResult(result);
    }
```

```

    }

    public static void retrieveOficinaDeOperacionRetirada(ObjectContainer db) {
        IngresoRetirada proto = new IngresoRetirada("abcde",null, null, 0,null,null);
        ObjectSet result = db.queryByExample(proto);
        IngresoRetirada op = (IngresoRetirada) result.next();
        System.out.println("Oficina en la que se ha realizado la operacion abcde \n" +
op.getOficina());
    }

    public static void retrieveOperationByCodigo(ObjectContainer db) {
        Operacion op = new Operacion("abcde",null, null, 0,null);
        ObjectSet result = db.queryByExample(op);
        listResult(result);
    }

    public static void retrieveOperationBySaldo(ObjectContainer db) {
        Operacion Account1 = new Operacion(null,null, null, 1050.05,null);
        ObjectSet result = db.queryByExample(Account1);
        listResult(result);
    }

    public static void updateOperation(ObjectContainer db) {
        ObjectSet result = db
            .queryByExample(new Operacion("fghij",null, null, 0,null));
        Operacion found = (Operacion) result.next();
        found.setFecha("12/1/80");
        db.store(found);
        System.out.println("Ahora se data en 12/1/80" + found);
    }

    public static void deleteFirstOperacionByCodigo(ObjectContainer db) {
        ObjectSet result = db
            .queryByExample(new Operacion("fghij",null, null, 0,null));
        Operacion found = (Operacion) result.next();
        db.delete(found);
        System.out.println("Deleted operacion" + found);
    }

    /* AQUI EMPIEZAN LAS PRUEBAS DE OFICINA */

    public static void storeFirstOficina(ObjectContainer db) {
        Oficina of = new Oficina("1234-SAN", "Calle Marujas ZGZ", "976123456");
        db.store(of);
        System.out.println("Stored Oficina" + of);
    }

    public static void storeSecondOficina(ObjectContainer db) {
        Oficina of = new Oficina("5678-BBVA", "Avenida Perico Madrid", "902202122");
        db.store(of);
        System.out.println("Stored Oficina" + of);
    }

```

```
public static void anyadirCuentaAFirstOficina(ObjectContainer db){
    ObjectSet result = db
        .queryByExample(new Oficina("1234-SAN", null, null));
    Oficina found = (Oficina) result.next();
    result = db
        .queryByExample(new Corriente(null, "5678-1234-EFGH", 0, 0.0, null));
    Corriente c1 = (Corriente) result.next();
    found.addCorriente(c1);
    db.store(found);
    System.out.println("Añadido " + c1 + " a " + found);
}

public static void retrieveCuentasFromOficina(ObjectContainer db) {
    Oficina of = new Oficina("1234-SAN", null, null);
    ObjectSet result = db.queryByExample(of);
    of = (Oficina) result.next();
    System.out.println("Cuentas de " + of);
    listResult(of.getCorrientes());
}
}
```

}
Los resultados que se obtienen como salida de la ejecución del ejemplo 1 por terminal son los siguientes:

-----CLIENTE-----

Stored clienteCarlos Maranyes Nueno - 12345678A

Stored clienteNicolas Lera Lopez - 87654321B

2

Carlos Maranyes Nueno - 12345678A

Nicolas Lera Lopez - 87654321B

1

Nicolas Lera Lopez - 87654321B

1

Carlos Maranyes Nueno - 12345678A

Ahora se llamaAnacardo Lera Lopez - 87654321B

2

Carlos Maranyes Nueno - 12345678A

Anacardo Lera Lopez - 87654321B

Deleted cliente Carlos Maranyes Nueno - 12345678A

1

Anacardo Lera Lopez - 87654321B

-----Cuenta-----

Stored Ahorro1234-5678-ABCD -> 20000.01 -> 0.05%

Stored Corriente5678-1234-EFGH -> 10000.15 -> Calle Marujas ZGZ

2

1234-5678-ABCD -> 20000.01 -> 0.05%

5678-1234-EFGH -> 10000.15 -> Calle Marujas ZGZ

1

5678-1234-EFGH -> 10000.15 -> Calle Marujas ZGZ

1

5678-1234-EFGH -> 10000.15 -> Calle Marujas ZGZ

Ahora se llama anacardo5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ

2

1234-5678-ABCD -> 20000.01 -> 0.05%

5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ

Deleted cuenta1234-5678-ABCD -> 20000.01 -> 0.05%
1
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
-----Relacion Cuenta-Cliente-----
Stored clienteCarlos Maranyes Nuevo - 12345678A
Stored clienteNicolas Lera Lopez - 87654321B
Stored Ahorro1234-5678-ABCD -> 20000.01 -> 0.05%
AÃ±adido Nicolas Lera Lopez - 87654321B a 1234-5678-ABCD -> 20000.01 -> 0.05%
AÃ±adido Carlos Maranyes Nuevo - 12345678A a 1234-5678-ABCD -> 20000.01 -> 0.05%
2
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
1234-5678-ABCD -> 20000.01 -> 0.05%
AÃ±adido 1234-5678-ABCD -> 20000.01 -> 0.05% a Nicolas Lera Lopez - 87654321B
AÃ±adido 5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ a Nicolas Lera Lopez -
87654321B
2
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
1234-5678-ABCD -> 20000.01 -> 0.05%
Clientes de 1234-5678-ABCD
2
Nicolas Lera Lopez - 87654321B
Carlos Maranyes Nuevo - 12345678A
Cuentas de Nicolas Lera Lopez
2
1234-5678-ABCD -> 20000.01 -> 0.05%
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
-----Operacion-----
Stored Ahorro1234-5678-ABCD -> 20000.01 -> 0.05%
Stored OficinaCalle Marujas ZGZ
Stored Transferecia fghij transferencia de 1234-5678-ABCD -> 20000.01 -> 0.05% a
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ entre 1234-5678-ABCD -> 20000.01 ->
0.05% y 5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
Stored Operacionabcde ingreso o retirada en Calle Marujas ZGZ
2
fghij transferencia de 1234-5678-ABCD -> 20000.01 -> 0.05% a 5678-1234-EFGH ->
2.000000001E7 -> Calle Marujas ZGZ
abcde ingreso o retirada en Calle Marujas ZGZ
1
abcde ingreso o retirada en Calle Marujas ZGZ
1
abcde ingreso o retirada en Calle Marujas ZGZ
Oficina en la que se ha realizado la operacion abcde
Calle Marujas ZGZ
Ahora se data en 12/1/80fghij transferencia de 1234-5678-ABCD -> 20000.01 -> 0.05% a
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
Deleted operacionfghij transferencia de 1234-5678-ABCD -> 20000.01 -> 0.05% a
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ
-----Oficina-----
Stored OficinaCalle Marujas ZGZ
Stored OficinaAvenida Perico Madrid
AÃ±adido 5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ a Calle Marujas ZGZ
Cuentas de Calle Marujas ZGZ
1
5678-1234-EFGH -> 2.000000001E7 -> Calle Marujas ZGZ

12. COMPARACIÓN DE LOS SGBD

Aunque Oracle sea uno de los gestores más utilizados, en cuanto al modelado de bases de datos de tipo Objeto/Relacional quizás sea de utilidad plantearse el uso de DB2. Oracle no permite la herencia de tablas (al menos en la versión utilizada) mientras que DB2 permite jerarquía tanto de tipos como de tablas. Además, las referencias se asemejan más la de un lenguaje orientado a objetos en el sentido en el que puedes acceder directamente a la tupla apuntada por la referencia y a sus atributos. Sin embargo, no permite arrays de referencia mientras que Oracle permite el uso de tablas anidadas, por lo que en este aspecto le supera.

PostgreSQL se queda bastante por detrás en cuanto al modelo Objeto/Relacional, ya que lo único que permite del mismo es la jerarquía de tablas. Permite la creación de tipos pero no de tablas tipadas, por lo que éste gestor se reserva más al modelo relacional con algún aspecto de herencia.

Un problema que afecta a todos los gestores analizados es el hecho que para garantizar las consistencia de las referencias sea necesario crear un trigger o utilizar un campo adicional que sea la clave primaria de la entidad con la que se relacionan, por lo que la referencia sería un “atajo” para acceder al otro extremo de la relación.

13. DIFICULTADES ENCONTRADAS

Uno de los principales problemas a la hora del desarrollo de la práctica ha sido la instalación de DB2, debido a la falta de documentación para instalar la versión específica que se pide y los problemas de dependencias que tiene asociados.

Otro de los principales problemas ha sido crear las tablas de los gestores en Objeto/Relacional debido a que no hay ningún ejemplo en la web de la asignatura de cómo crearlas, por lo que se ha tenido que recurrir a material online y adaptarlo de alguna manera al caso que se pide, por lo que en este apartado se ha empleado una gran cantidad de horas. Además, lo que un gestor permite otro no y no se han encontrado este tipo de comparaciones en Internet.

14. ESFUERZOS INVERTIDOS

	Carlos Maraños (717788)	Nicolás Lera	Total Horas
Instalación de los diferentes gestores de BBDD	7h	3h	10h
Creación del modelo E/R	1h	2h	3h
Diseño del modelo lógico	1h	2h	3h
Creación de las tablas SQL	1h	2h	3h
Licencias	3h	4h	7h
Búsqueda de herramientas y población de la base de datos	6h	2h	8h
Consultas	3h	5 h	8h
Tablas Oracle Objeto - Relacional	9h	1h	10h
Pruebas Oracle Objeto - Relacional	2h	1h	3h
Tablas y pruebas PostgreSQL Objeto-Relacional	5h	2h	7h
Tablas y pruebas DB2 Objeto relacional	6h	1h	7h
Clases y pruebas de DB4O	2h	7h	9h
Documentación	11h	8h	19h
TOTAL horas	57h	40h	97h

15. BIBLIOGRAFÍA

Crear base de datos en Oracle - Último acceso: 1/04/2018

https://www.ibm.com/support/knowledgecenter/es/SS6KJL_8.6.0/FEB/in_oracle_creating_db.html

Crear usuarios en Oracle - Último acceso: 1/04/2018

https://www.ibm.com/support/knowledgecenter/es/SSFPJS_8.5.5/com.ibm.wbpm.imuc.sbp.doc/topics/db_typ_nd_lin_orcl.html

Tutorial PostgreSQL - Último acceso: 1/04/2018

<https://www.digitalocean.com/community/tutorials/como-instalar-y-utilizar-postgresql-en-ubuntu-16-04-es>

Crear usuarios en PostgreSQL - Último acceso: 1/04/2018

<https://www.nanotutoriales.com/como-crear-un-usuario-y-asignarle-permisos-en-postgresql>

Instalación DB2 - Último acceso: 1/04/2018

<http://angocadb2.blogspot.com.es/2012/12/instalacion-de-db2-97-en-ubuntu-1204.html>

Ejecutar script SQL en DB2 - Último acceso: 1/04/2018

<https://www.experts-exchange.com/questions/20932158/db2-running-sql-file.html>

Problema binutils DB2 - Último acceso: 1/04/2018

<http://www-01.ibm.com/support/docview.wss?uid=swg21984672>

Crear instancia DB2 - Último acceso: 1/04/2018

<http://lpetr.org/blog/archives/install-db2-while-sshing-to-a-linux-server-from-windows>
https://www.ibm.com/support/knowledgecenter/en/SSNE44_5.2.0/com.ibm.tpc_V52.doc/fqz0_t_installing_db2_on_aix_cli.html
https://www.ibm.com/support/knowledgecenter/es/SSEPGG_9.7.0/com.ibm.db2.luw.qb.server.doc/doc/c0011931.html

Crear tabla DB2 - Último acceso: 1/04/2018

<http://users.sdsc.edu/~jrowley/db2/howto.html>

Licencias DB2 - Último acceso: 1/04/2018

ftp://ftp.software.ibm.com/ps/products/db2/info/vr101/pdf/en_US/DB2AdminGettingStarted-db2xpe1010.pdf

Objeto/Relacional Oracle - Último acceso: 1/04/2018

https://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjbas.htm#i471260

Objeto/Relacional DB2 - Último acceso: 1/04/2018

<https://www.ibm.com/developerworks/data/library/techarticle/zeidenstein/0108zeidenstein-pdf.pdf>

Objeto/Relacional PostgreSQL - Último acceso: 1/04/2018

<http://www.postgresqltutorial.com/postgresql-user-defined-data-types/>

<https://blog.dbi-services.com/what-are-typed-tables-in-postgresql/>

<https://stackoverflow.com/questions/21573005/foreign-key-constraint-on-sub-column-of-composite-type-column-in-postgresql>

<https://dba.stackexchange.com/questions/37164/postgresql-get-single-attribute-of-udt-in-select-statement>

<https://www.postgresql.org/docs/9.5/static/ddl-inherit.html>

Jerarquía de tipos en Oracle - Último acceso: 2/04/2018

https://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjbas.htm#CIHEJHJJ

Problema Arrays de referencia en PostgreSQL - Último acceso: 2/04/2018

<https://stackoverflow.com/questions/34156695/postgres-creating-a-table-with-an-array-of-foreign-keys>

<https://www.postgresql.org/message-id/1343842863.5162.4.camel@greygoo.devise-it.lan>

Problema con ';' en DB2 - Último acceso: 2/04/2018

<https://www.ibm.com/developerworks/community/forums/html/topic?id=77777777-0000-0000-0000-000014141170>

Problema con arrays de referencia en DB2 - Último acceso: 2/04/2018

<https://stackoverflow.com/questions/6640990/db2-sql-array-in-attribute-defs-of-user-defined-type>

Problema claves ajenas en DB2 - Último acceso: 2/04/2018

<https://kb.informatica.com/howto/6/Pages/19/504692.aspx>