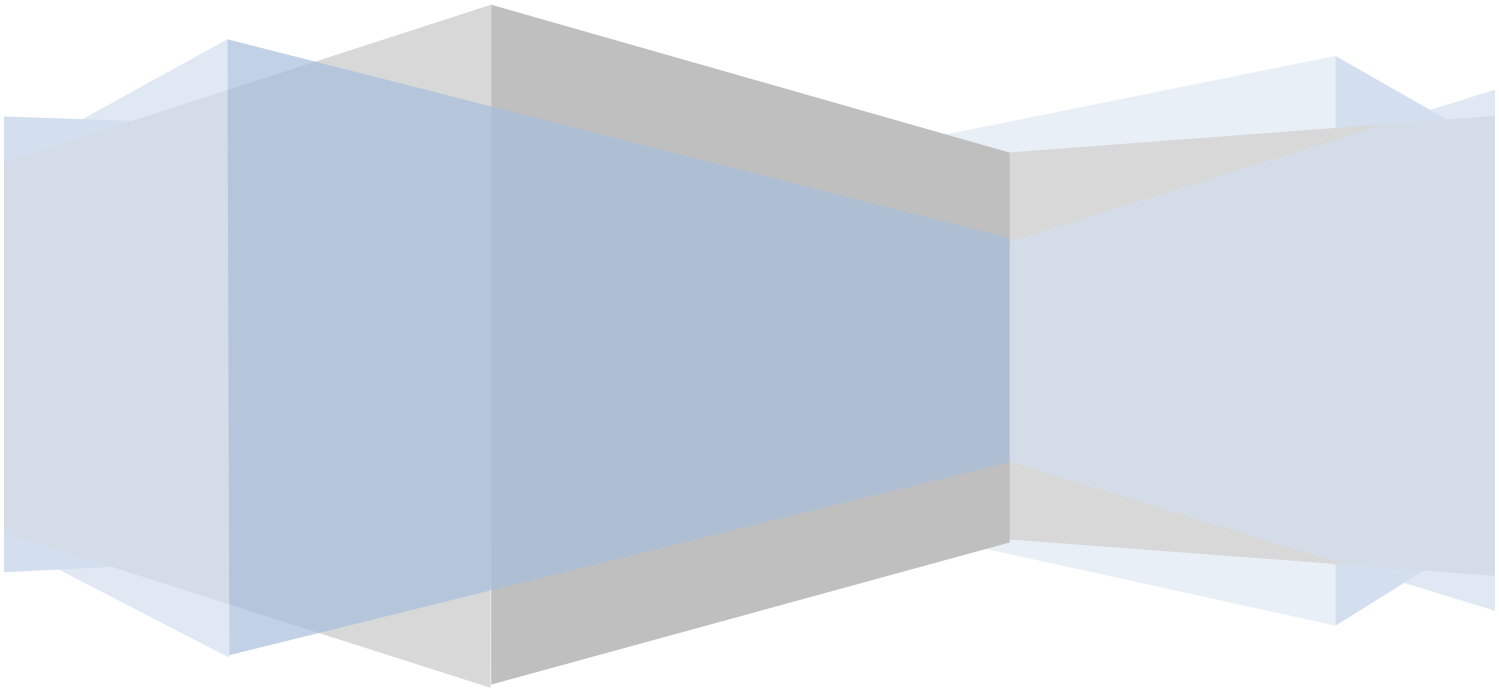
Universidad de Zaragoza

Práctica 1+2:

Instalación de Sistemas Gestores de Bases de Datos y Diseño de Bases de Datos con Características de Orientación a Objetos

Adrian Casans (590114) Sergio Pedrero (627669) Diego Sánchez (628279) Cristian Simón (611487)



ÍNDICE

Esfuerzos invertidos:

Tabla que recoge un desglose con las tareas y esfuerzos realizados por cada uno de los miembros del grupo junto con el tiempo invertido:

Tarea	Encargado	Horas
Instalar gestores	Sergio	3
Diseño conceptual: Entidad/Relación	Adrián/Cristian	3
Diseño lógico relacional	Adrián	1
Implementar diseño lógico relacional (Oracle)	Sergio	4
Diseño lógico objeto/relacional	Diego/Cristian	4/0,25
Implementar diseño lógico objeto/relacional	Diego	13
Introducir datos modelo relacional/objetos	Cristian	2
Pruebas/consultas modelo relacional/objetos	Sergio	4
Información acerca de las licencias	Cristian	1
Comparativa de los SGBD	Cristian	0,3
Implementación db4o	Adrián/Cristian	6/0,3
Memoria	Cristian	3,5

Objetivos:

El objetivo del desarrollo de la practica consistirá en el trabajo en una maquina virtual con los diversos sistemas gestores de bases de datos (SGBD) mediante el diseño y la implementación de una base de datos bancaria.

Esto nos permitirá observar las diferencias y aspectos relevantes de cada gestor, así como conocer las diferentes estrategias y herramientas para la generación e inserción de datos.

Configuración de la maquina virtual:

Se va a trabajar sobre una maquina virtual ***Turnkey Linux*** de 32 bits basada en una distribución Debian sobre la cual se instalaran los determinados gestores de bases de datos que se usaran a lo largo de la práctica.

Esta se virtualizará sobre un software ***Virtual Box*** y en nuestro caso se ha utilizado la imagen que se proporciona en el servidor Hendrix.

Una vez que lanzamos la máquina virtual debemos configurar algunos aspectos básicos como son :

1) **Configurar idioma de teclado**

   Por defecto la distribución del teclado de la maquina virtual no esta en castellano por lo que lanzando los comandos del siguiente link conseguimos cambiarla :

   https://moodle2.unizar.es/add/mod/page/view.php?id=560956

   Hay que tener en cuenta también la configuración del terminal desde el que se accede. Se recomienda acceder por ssh desde un terminal remoto a hacerlo desde el propio gestor de ventanas de VirtualBox ya que de esta forma ganamos versatilidad al poder usar las funciones de copy+paste,etc.

2) **Configuración de la red en las maquinas virtuales**

   Se necesita configurar la red como NAT con mapeo de puertos.
   Siguiendo las siguientes instrucciones podemo configurar dicho aspecto :

   https://moodle2.unizar.es/add/mod/page/view.php?id=560957

   En las instrucciones mencionadas anteriormente se incluye un ejemplo de cómo mapear los puertos entre la maquina virtual y el sistema anfitrión, en este caso se explica como seria el del servicio de SSH para poder así conectarse de forma remota desde el terminal anfitrión y ganar las opciones mencionadas en el apartado anterior.

Instalación y administración básica de los SGBD:

- **MySQL:**

Para la instalación de MySQL en nuestra maquina virtual son necesarios dos paquetes: *mysql-server* y *mysql-client*. Para su instalación debemos ejecutar desde la maquina virtual los comandos:

apt-get update

apt-get install mysql-server mysql-client

Con esto obtendremos la versión más reciente de MySQL. Una vez se han descargado los paquetes necesarios, el proceso se inicia automáticamente y pide introducir una contraseña para el administrador de la base de datos.

Para conectar con el servidor ya instalado simplemente hay que ejecutar:

mysql –u nombre_usuario –p

Para permitir las conexiones remotas es necesario editar el fichero de configuración */etc/MySQL/.my.cnf* y comentar con una almohadilla la siguiente línea:

bind_address 127.0.0.1

Y a continuación reiniciar el servicio para aplicar la configuración:

service mysql stop
service mysql start

No se permite que el usuario *root* conecte de manera remota, por lo que deberemos crearnos un nuevo usuario con los privilegios oportunos:

CREATE USER 'user' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON *.* TO 'user'@'%';

Finalmente, para crear una base de datos y comenzar a trabajar con ella tendremos que ejecutar:

CREATE DATABASE nombre_bd;

- **Oracle:**

Para la instalación de Oracle en la maquina virtual primero debemos descargar el siguiente fichero :

oracle-xe_10.2.0.1-1.1_i386.deb

A continuación enviamos el fichero de nuestro SO anfitrión a la maquina virtual Turnkey ,por ejemplo con **FileZilla**. Una vez enviado nos situamos desde la maquina virtual en la ruta correspondiente y lanzamos:

$ apt-get install bc

$ apt-get install libaio1

$ dpkg –install oracle-xe_10.2.0.1-1.1_i386.deb

A continuación se ejecuta el siguiente comando para configurar Oracle:

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

Se siguen los pasos de instalación y configuración facilitados en la ayuda para la instalación de Oracle de Moodle:

https://moodle2.unizar.es/add/mod/page/view.php?id=560959

Una vez completada la configuración se debe ejecutar el siguiente comando :

$ vi /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin/nls_lang.sh

reemplazando :

```
if [[ -n "$LC_ALL" ]]; then
        locale=$LC_ALL
elif [[ -n "$LANG" ]]; then
        locale=$LANG
else
        locale=
fi
```

por:

if [-n "$LC_ALL"]; then

locale=$LC_ALL

elif [-n "$LANG"]; then

locale=$LANG

else

locale=

fi

(Se cambian los dos corchetes por uno solo)

Después editamos el siguiente fichero :

$ vi /etc/profile

Y añadimos al final del mismo la siguiente línea:

source .usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin/oracle_env.sh

Con esto se consigue que se encuentre la ruta adecuada para conectar con el SGBD.

El ultimo paso será editar el siguiente fichero :

$ vi .bashrc

Sustituyendo :

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Por:

PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Con esto ya tendremos listo el gestor para poder arrancarlo.

Ejecutamos desde nuestro directorio de usuario :

$ ./etc/profile

A continuación debemos habilitar el servicio de Oracle para que arranque automaticamente :

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

y arrancarlo :

$ /etc/init.d/oracle-xe start

Esto ya permite acceder al gestor mediante uno de los siguientes comandos:

$ sqlplus sys/ORA-PASS1 as sysdba

o:

$ sqlplus sys as sysdba

Oracle XE no permite crear nuevas bases de datos y la opción que hemos escogido es crear un usuario con los privilegios necesarios.

/* Se crea usuario para acceso a SGBD */
CREATE USER “USER” IDENTIFIED BY PASSWORD
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;

/* Se le da privilegios de superusuario al usuario previo*/
GRANT ALL PRIVILEGES TO “USER”;

Una vez creado el usuario se pueden llevar a cabo las tareas pertinentes para realizar la practica usando dicho gestor.

- **PostgreSQL:**

Para obtener **PostgreSQL** en una de sus últimas versiones es necesario ejecutar el comando :

$ apt-get install postgresql

Por ahora **apt-get oferta la versión 9.1**, aunque en la web se encuentra disponible la 9.4.

Durante la instalación , en el sistema se crea un usuario con el nombre “postgres”. Es necesario cambiar la contraseña de dicho usuario. Para ello se ejecuta el comando:

$ passwd postgres

Al mismo tiempo, dentro del SGBD existe un usuario con el mismo nombre(postgres). Es necesario cambiar la contraseña de dicho usuario y escribir la misma que para el usuario anterior. Para ello hay que entrar en la consola de administración mediante el comando :

$ su –l postgres

(El uso del “su” es obligatorio, aunque se jecute desde una cuenta “root”. Por otra parte, el uso del “-l” a continuación es para ver el log y evitar errores)

Dentro del nuevo entorno, es necesario conectarse al servidor y modificar la contraseña desde ahí. Para llevarlo a cabo la secuencia es la siguiente :

postgres@maquina:/dir$ > psql postgres

postgres=# ALTER ROLE postgres PASSWORD ‘contraseña’;

PostgreSQL confirma la modificación con la salida “ALTER ROLE”. Finalmente para salir:

postgres=# \q

postgres@maquina:/~$> exit

Además de lo anterior, también es necesario configurar el acceso al gestor. Para ello hay que editar los ficheros de configuración ya existentes en el directorio de instalación (generalmente **“/etc/postgresql/9.1/main/”**).

Por una parte hay que editar el fichero de configuración del servidor denominado **“postgresql.conf”.** Descomentar la línea que contiene la clave **“listen addresses”** y dependiendo del tipo de acceso requerido modificar el valor a **‘localhost’** para acceso local o a **‘*’** para acceso remoto. También existe la posibilidad de especificar varias direcciones ip separadas mediante espacios. Además, en este fichero se puede modificar el puerto en el que el sistema permanece a la espera de nuevas conexiones(por defecto 5432).

Por otra parte, hay que editar el fichero de configuración de los clientes denominado “pg_hba.conf”. Por defecto permite las conexiones locales a todos los clientes, pero si se quiere acceder remotamente es necesario añadir al final la línea:

host all all all md5

(tipo BD usuario dirección método)

Pudiendo especificar en el campo BD las bases de datos autorizadas para el usuario dado. En el campo dirección se puede especificar la IP permitida(pudiendo usar mascaras de red para abarcar rangos de direcciones). El campo método especifica el método de cifrado para el envío de la contraseña a través de la red.

La opción **“all”** en los campos BD usuario y dirección permite eliminar cualquier tipo de restricción de acceso.

Para hacer efectiva la nueva configuración hay que reiniciar el gestor y para ello basta con ejecutar:

$ service postgresql restart

A partir de este momento ya se puede acceder al gestor con el usuario **"postgres"** y la contraseña especificada para dicho usuario.

Por motivos de seguridad, lo ideal es crear nuevos usuarios y asignarles únicamente los privilegios que necesiten . Para realizar dicha operación, es necesario volver a entrar en la consola de administración del gestor.

$ su –l postgres

postgres@maquina:/~$> createuser –S –R –l usuario

Acto seguido, pregunta si el usuario debe tener permisos para crear bases de datos (**Introducir "y" para permitir, "n" para denegar**).

La opción –S establece que el usuario no tenga permisos de superusuario. La opción –R establece que el usuario no pueda crear nuevos usuarios. La opción –l establece que el usuario pueda iniciar sesión.

Si se desean eliminar usuarios existe un comando similar:

postgres@maquina:/~$> dropuser usuario

Par modificar la contraseña del nuevo usuario hay que conectarse al servidor postgreSQL y realizar el cambio de forma muy similar a como se hizo para el usuario postgres. Esta vez es necesario especificar que la contraseña se almacene encriptada.

Las instrucciones son las siguientes:

postgres@maquina:/~$> psql postgres

postgres=# ALTER USER usuario WITH ENCRYPTED PASSWORD 'contraseña';

El servidor confirma la modificación con la salida "ALTER ROLE".

En postgreSQL, crear una nueva base de datos implica duplicar una plantilla predefinida en el gestor, de modo que la nueva base de datos es una copia de otra ya existente. Por defecto se duplica la plantilla #1, pero es posible que contenga modificaciones a posteriori. Para crear una base de datos totalmente limpia, existe la opción de duplicar la plantilla #0.

Por otra parte, y teniendo en cuenta que el castellano contiene caracteres especiales, lo ideal es usar una codificación "UTF-8" capaz de soportar dichos caracteres.

Teniendo en cuenta lo anterior podemos realizar las siguientes operaciones:

postgres@maquina:/~$> createdb –Ttemplate0 –O usuario –EUTF-8 nombreBD

Donde –O especifica el usuario propietario de la nueva base de datos.

Por último, es necesario otorgar privilegios al usuario. Para ello hay que volver a conectar con el servidor y ejecutar la siguiente instrucción:

postgres@maquina:/~$> psql postgres

postgres=# GRANT ALL ON DATABASE nombreBD TO usuario;

Donde **"ALL"** puede ser el conjunto de operaciones SQL a permitir, separadas por comas.

Nuevamente el servidor confirma la modificación con la salida **"GRANT".**

(Para salir \q).

- **DB2:**

Comentarios acerca de las licencias:

- **MySQL**

Este servidor de bases de datos se distribuye bajo los temimos de la licencia publica general (GNU), pero cuenta con una licencia comercial disponible para aquellos que quieran distribuir aplicaciones no GPL que requieran MySQL.

Es por esto por lo que, si no vamos a realizar una aplicación para un "uso interno", sino que vamos a distribuirla deberemos adquirir una licencia comercial por cada aplicación y maquina servidora corriendo MySQL. Estas licencias tienen un valor unitario de $200.

- **Oracle**

Oracle nos ofrece una única versión gratuita (Oracle XE). Para el resto de sus productos necesitaremos adquirir una licencia:

  - **Oracle Database Standard Edition One, SE1 (15.194€):** Producto orientado a PYME y que tiene la limitación de que se puede instalar en servidores que tengan como máxima capacidad de cremiento hasta 2 procesadores.
  - **Oracle Database Standard Edition, SE (5.036€):** Producto también orientado a PYME, pero esta vez se puede instalar en servidores que tengan como máxima capacidad de crecimiento hasta 4 procesadores. Viene con la opción RAC incluida, que nos permite crear sitios con escalabilidad y alta disponibilidad.
  - **Oracle Database Enterprise Edition, EE (42.240€):** Orientado a corporaciones medianas y grandes, esta vez sin limitaciones de procesadores. En esta opción se tiene acceso a todas las opciones de Oracle (Algunas con pago adicional).

- **PostgreSQL**

Se trata de proyecto de código abierto, distribuido bajo la "licencia PostgreSQL", el desarrollo de este sistema no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada.

Gracias a esto está permitido el uso, copia, modificación y la distribución, ya sea para uso personal o incluso comercial.

- **DB2**

Sistema que es gestionado por IBM para el que es necesario licencia para su uso, en 2006 IBM anuncio una estrategia de precio por procesador basada en unidades de valor de procesador. Los límites de procesador para productos de base de datos DB2 son los siguientes:

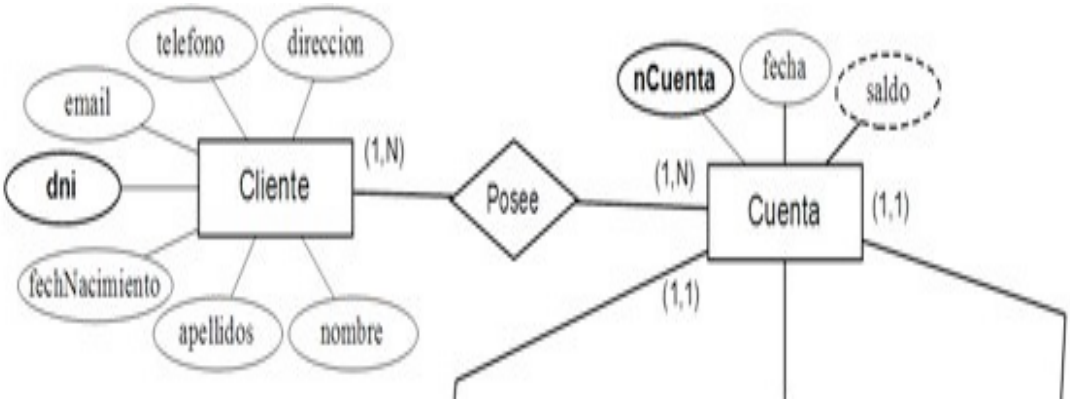
  - **DB2 Express Edition:** se pueden tener hasta 4 procesadores, individuales o de doble núcleo; hasta un máximo de 200 PVUs.
  - **DB2 Workgroup Server Edition:** el limite depende de la arquitectura. Para una arquitectura Intel o AMD se pueden tener dieciséis procesadores: de un solo núcleo, de doble núcleo o de cuatro núcleos. Para otras arquitecturas está limitado a 16 procesadores de un solo núcleo o bien 16 núcleos en total.

El precio de DB2 arranca a partir de los 7.200€.

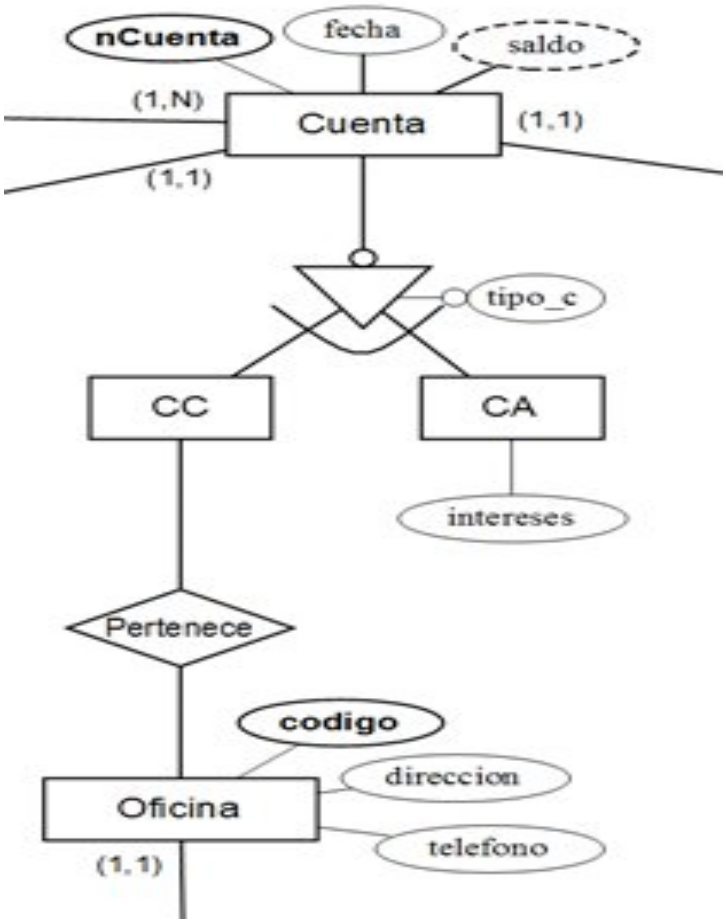
Diseño de la base de datos relacional:

Modelo E/R:

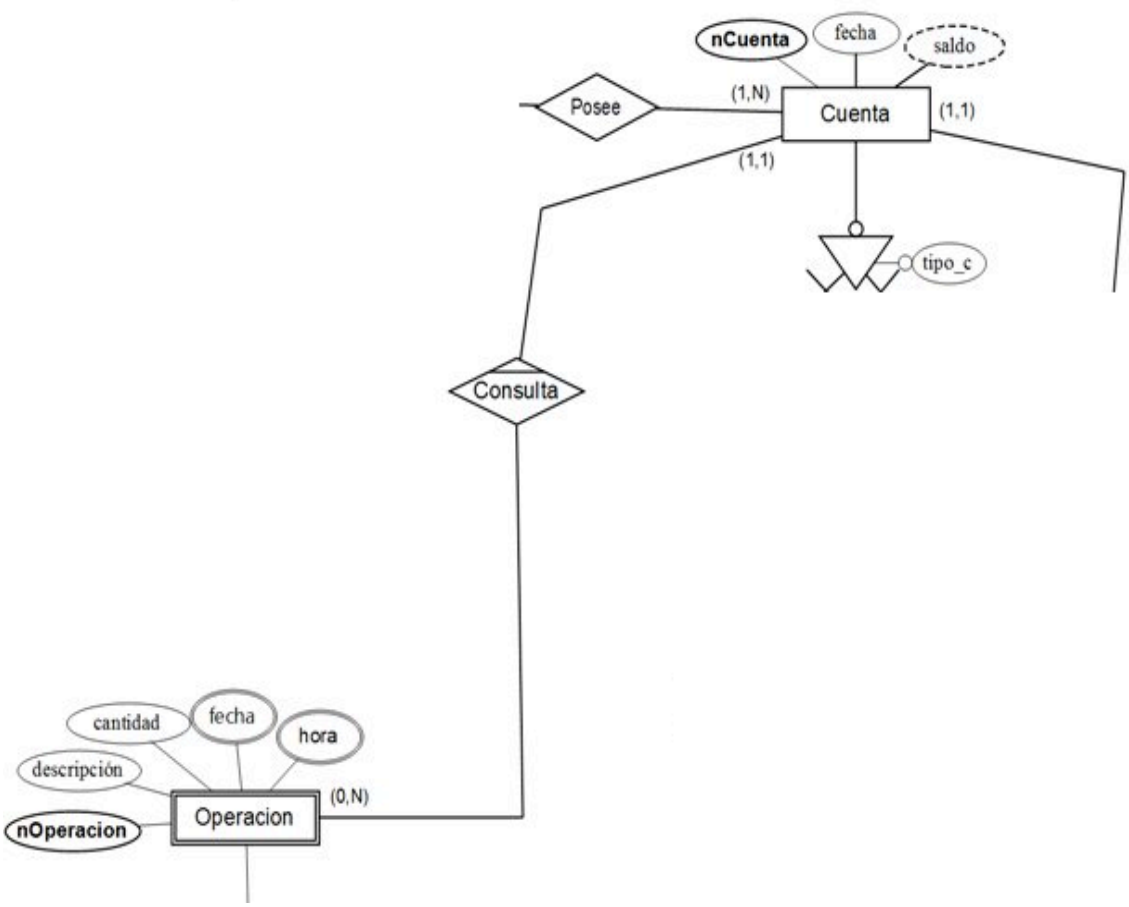
- **Relación entre cliente y cuenta:**



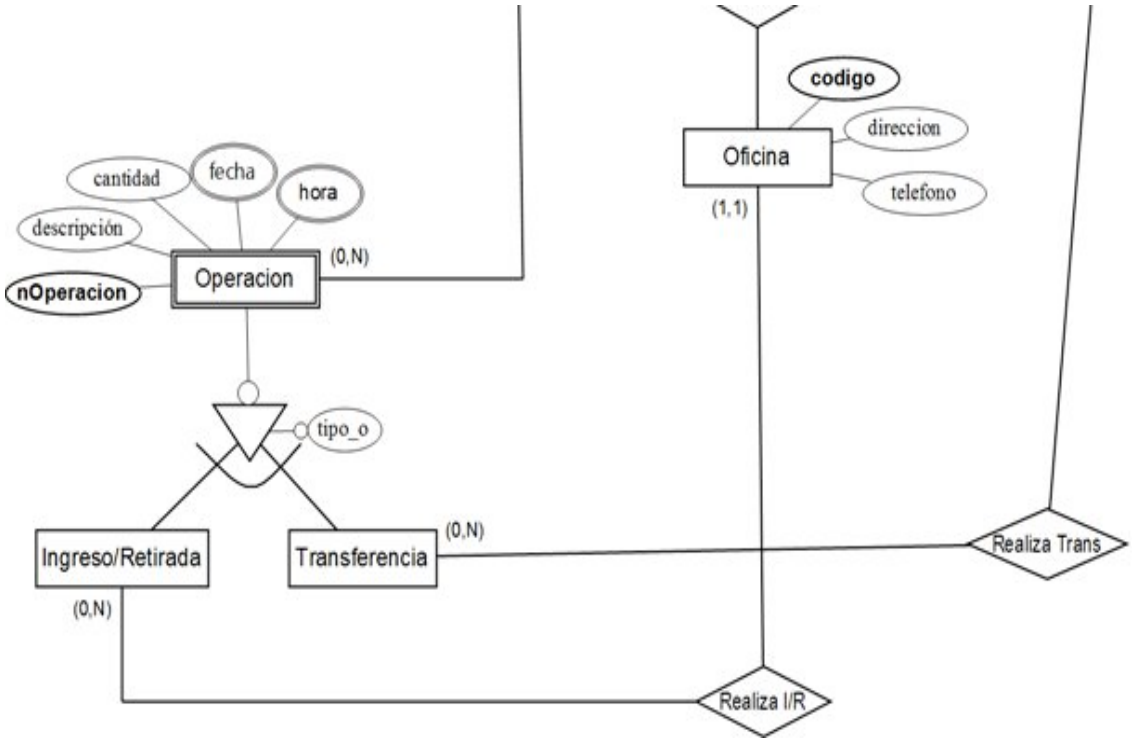
- **Relación entre cuenta y oficina:**



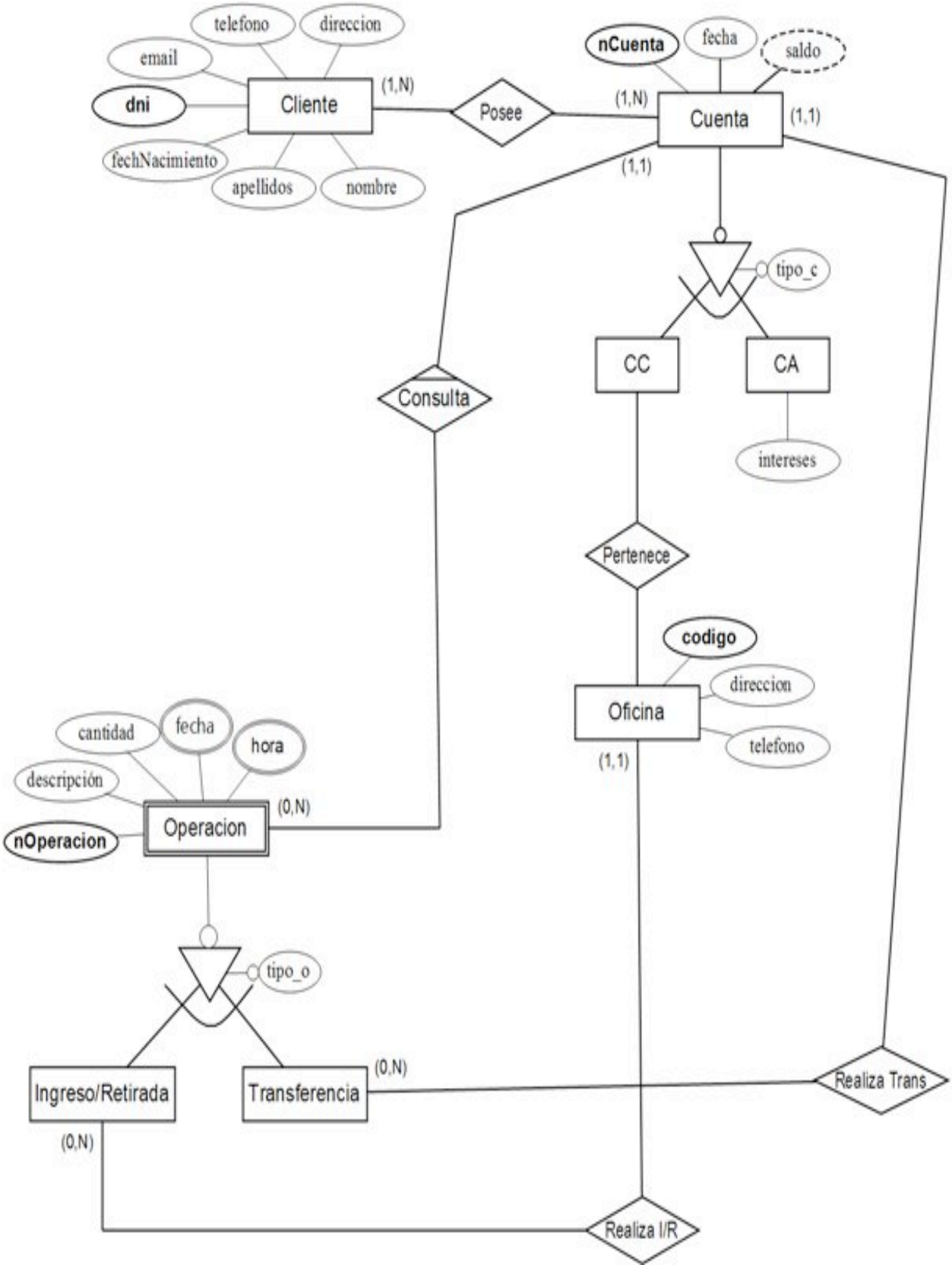
- **Relación entre cuenta y operación:**



- **Relación entre operación y oficina:**



- **Completo**



Modelo relacional:

Cliente

(

dni = tpDni, clave primaria;
nombre = tpTexto, NO NULO;
apellidos = tpTexto, NO NULO;
direccion= tpTexto, NO NULO;
email = tpTexto, NO NULO;
telefono = tpTelefono, NO NULO;
fechNacimiento = tpFecha, NO NULO;

)

Oficina

(

codigo = tpCodOficina, clave primaria;
direccion = tpDireccion, NO NULO;
telefono = tpTelefono, NO NULO;

)

Cuenta

(

nCuenta = tpNumCuenta, clave primaria;
fecha = tpFecha, NO NULO;
saldo = tpSaldo, NO NULO, POR DEFECTO 0;
tipo_c = tpTipo, NO NULO;
interes = tpInteres, POR DEFECTO NULO;
codigo = tpCodOficina, clave ajena de Oficina;

)

Restricciones de **Cuenta**:

1) Cuando cuenta nueva, saldo = 0.
   saldo = ingresos – retiradas + transferencias

2) El saldo de las cuentas de ahorro se revisa todas las noches para ver si hay que aplicarle el aumento proporcionado por los intereses.

3) Ingresos serán cuando cantidad sea positiva y retiradas cuando cantidad sea negativa y en ambos casos operación sea 0. Cuando operación sea 1, será transferencia

4) Fecha será la fecha de creación de dicha cuenta.

5) Cuenta será cuenta corriente cuando tipo_c = 0. Cuando tipo_c = 1 será cuenta de ahorro y interés será no nulo.

Operacion

(

nCuenta_origen = tpNumCuenta, clave ajena de Cuenta;
nOperacion = tpNumOperacion;
fecha = tpFecha, NO NULO;
hora = tpHora, NO NULO;
descripcion = tpTexto;
cantidad = tpSaldo, NO NULO;
tipo_o = tpTipoOperacion, NO NULO;
nCuenta_dst = tpNumCuenta, clave ajena de Cuenta, POR DEFECTO NULO;
codigo = tpCodOficina, clave ajena de Oficina, POR DEFECTO NULO;

clave primaria (nCuenta_orgn, nOperacion);

)

Restricciones de **Operacion**:

1) Ingresos serán cuando cantidad sea positiva y retiradas cuando cantidad sea negativa y en ambos casos operación sea 0. Cuando operación sea 1, será transferencia.

2) Si es una transferencia nCuenta_dst será no nulo. Si es un ingreso o retirada, código será no nulo.

Poseer

(

dni = tpDNI, clave ajena de Cliente;
nCuenta = tpNumCuenta, clave ajena de Cuenta;

clave primaria (dni, nCuenta);

)

Implementación con el modelo relacional:

Código en SQL para la creación de la BD relacional:

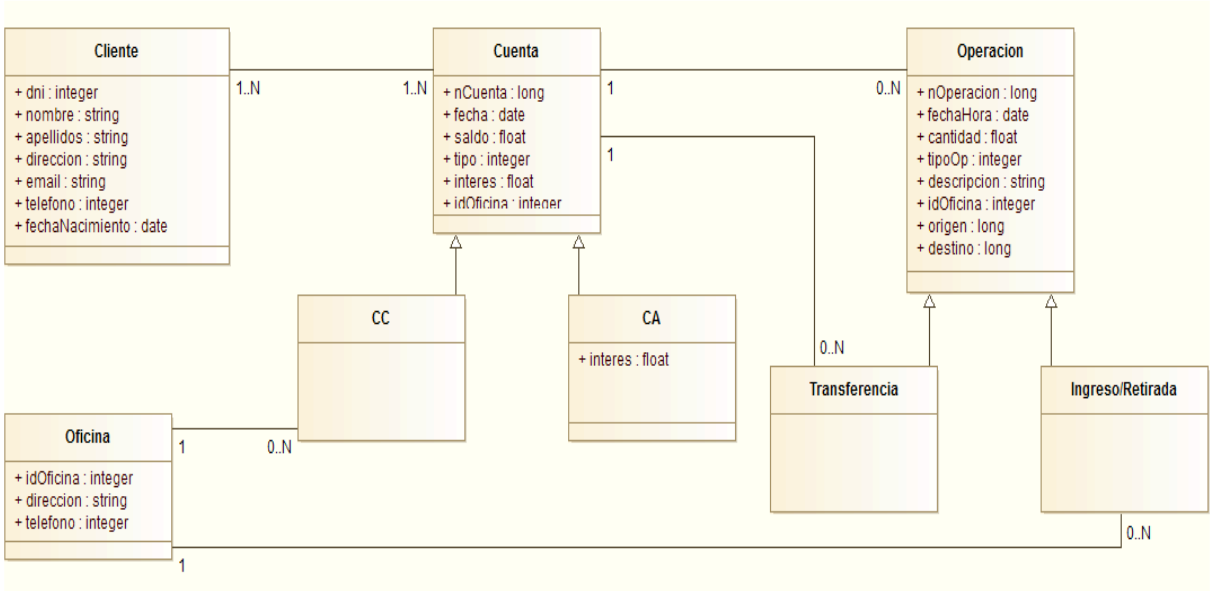
```
CREATE TABLE clientes(
        dni number(8),
        nombre varchar(30) NOT NULL,
        apellidos varchar(60) NOT NULL,
        direccion varchar(80) NOT NULL,
        email varchar(60) DEFAULT NULL,
        telefono number(9) NOT NULL,
        fechaNacimiento date NOT NULL,
        CONSTRAINT pk_cliente PRIMARY KEY(dni)
);

 CREATE TABLE oficinas(
        idOficina number(4),
        direccion varchar(80) NOT NULL,
        telefono number(9) NOT NULL,
        CONSTRAINT pk_oficina PRIMARY KEY(idOficina)
);

CREATE TABLE cuentas(
        nCuenta number(20),
        fecha date NOT NULL,
        saldo float DEFAULT 0,
        tipo number(1), -- 0 = cuenta corriente, 1 = cuenta ahorro
        interes float DEFAULT NULL,
        idOficina number(4) DEFAULT NULL,
        CONSTRAINT pk_cuenta PRIMARY KEY(nCuenta),
        CONSTRAINT fk_oficinaDeCuenta FOREIGN KEY(idOficina) REFERENCES
oficinas(idOficina)
);

CREATE TABLE operaciones(
        nOperacion number(20),
        fechaHora timestamp NOT NULL,
        cantidad float NOT NULL,
```

```
        tipoOp number(1) NOT NULL, -- 0 = tranferencia, 1 = ingreso, 2 = retirada,
        descripcion varchar(200),
        idOficina number(4),
        origen number(20),
        destino number(20) DEFAULT NULL,
        CONSTRAINT fk_origen FOREIGN KEY(origen) REFERENCES  cuentas(nCuenta),
        CONSTRAINT fk_destino FOREIGN KEY(destino) REFERENCES cuentas(nCuenta),
        CONSTRAINT fk_oficina FOREIGN KEY(idOficina) REFERENCES
oficinas(idOficina),
        CONSTRAINT pk_operacion PRIMARY KEY(nOperacion)
);

CREATE TABLE poseer(
        dni number(8),
        nCuenta number(20),
        CONSTRAINT pk_poseer PRIMARY KEY(dni, nCuenta),
        CONSTRAINT fk_cliente FOREIGN KEY (dni) REFERENCES clientes(dni),
        CONSTRAINT fk_cuenta FOREIGN KEY (nCuenta) REFERENCES cuentas(nCuenta)
);

/* Vista con los datos de cuentas corrientes y la oficina a la que estan ligadas */
CREATE VIEW cuentas_corrientes AS
SELECT nCuenta, fecha,saldo,idOficina FROM cuentas WHERE tipo = 0;

CREATE VIEW cuentas_ahorro AS
SELECT nCuenta,fecha,saldo,interes FROM cuentas WHERE tipo = 1;

CREATE VIEW tranferencia AS
SELECT nOperacion,fechaHora,cantidad,descripcion,origen,destino FROM operaciones
WHERE tipoOp = 0;

CREATE VIEW retirada AS
SELECT origen,nOperacion,fechaHora,cantidad,descripcion,idOficina FROM
operaciones WHERE tipoOp = 1;

CREATE VIEW ingreso AS
SELECT origen,nOperacion,fechaHora,cantidad,descripcion,idOficina FROM
operaciones WHERE tipoOp = 2;


CREATE TRIGGER actualizar_saldo
```

```
AFTER INSERT ON operaciones
FOR EACH ROW
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;

BEGIN
        /* Si es una transferencia(op==0) */
        IF: NEW.tipoOp = 0 THEN
        /* Actualiza saldo de las cuentas origen y destino al llevar a cabo la
transferencia*/
        UPDATE cuentas SET saldo = saldo - :NEW.cantidad WHERE nCuenta =
:NEW.origen;
        UPDATE cuentas SET saldo = saldo + :NEW.cantidad WHERE nCuenta =
:NEW.destino;

        /* Si se trata de un ingreso*/
        ELSIF :NEW.tipoOp = 1 THEN
        UPDATE cuentas SET saldo = saldo + :NEW.cantidad WHERE nCuenta =
:NEW.origen ;
        END IF;

        /* Si se trata de una retirada */
        ELSIF :NEW.tipoOp = 2 THEN
        UPDATE cuenta SET saldo = saldo - :NEW.cantidad WHERE nCuenta =
:NEW.origen;
        END IF;
        COMMIT;
END;
```

Implementación con el modelo objeto/relacional:

Esquema conceptual:



Modelo Objeto/relacional:

Cliente:

CREATE TYPE *client_udt* ***AS****(*
dni ***NUMBER****(8),*
nombre ***VARCHAR****(30),*
apellidos ***VARCHAR****(60),*
dirección ***VARCHAR****(80),*
email ***VARCHAR****(80),*
teléfono ***number****(9),*
fechNacimiento ***DATE****)*
NOT FINAL,
REF IS SYSTEM GENERATED*;*

CREATE TABLE *cliente* ***OF*** *client_udt(*
CONSTRAINT *pk_cliente* ***PRIMARY KEY****(dni),*
REF IS *oid* ***SYSTEM GENERATED****);*

Oficina:

```
CREATE TYPE oficina_udt AS(
        codigo NUMBER(4),
        dirección VARCHAR(80),
        teléfono NUMBER(9))
NOT FINAL,
REF IS SYSTEM GENERATED;

CREATE TABLE oficina OF oficina_udt(
        CONSTRAINT pk_oficina PRIMARY KEY(codigo),
        REF IS oid SYSTEM GENERATED);
```

Cuentas:

```
CREATE TYPE cuenta_udt AS(
        nCuenta NUMBER(20),
        fecha DATE,
        saldo REAL,
        tipo NUMBER(1)
NOT FINAL,
REF IS SYSTEM GENERATED;

CREATE TABLE cuenta OF cuenta_udt(
        CONSTRAINT pk_cuenta PRIMARY KEY(nCuenta),
        REF IS oid SYSTEM GENERATED);

CREATE TYPE cCorriente_udt UNDER cuenta_udt AS (
        codigo REF(oficina_udt) SCOPE oficina)
NOT FINAL;

CREATE TABLE cCorriente OF cCorriente_udt UNDER cuenta;

CREATE TYPE cAhorro_udt UNDER cuenta_udt AS (
        interés REAL)
NOT FINAL;

CREATE TABLE cAhorro OF cAhorro_udt UNDER cuenta;
```

Operaciones:

```
CREATE TYPE operacion_udt AS(
        nCuenta_origen REF(cuenta_udt) SCOPE cuenta,
        nOperacion NATURAL(11),
        fecha DATE,
        hora VARCHAR(8),
        descripción VARCHAR(250),
        cantidad REAL,
        tipo NATURAL(1))
NOT FINAL,
REF IS SYSTEM GENERATED;

CREATE TABLE operación OF operación_udt(
        CONSTRAINT pk_operacion PRIMARY KEY (nCuenta_origen,
nOperacion),
        REF IS oid SYSTEM GENERATED);

CREATE TYPE transferencia_ud UNDER operación_udt AS (
        nCuenta_dst REF(cuenta_udt) SCOPE cuenta)
NOT FINAL;

CREATE TABLE transferencia OF transferencia_udt UNDER operación;

CREATE TYPE movFisico_ud UNDER operación_udt AS (
        codigo REF(oficina_udt) SCOPE oficina)
NOT FINAL;

CREATE TABLE movFisico OF movFisico_udt UNDER operación;
```

Poseer:

```
CREATE TYPE poseer_udt AS(
        cliente     REF(cliente_udt) SCOPE cliente,
        nCuenta     REF(cuenta_udt) SCOPE cuenta)
NOT FINAL,
REF IS SYSTEM GENERATED;

CREATE TABLE poseer OF poseer_udt(
        CONSTRAINT pk_poseer PRIMARY KEY(cliente, nCuenta),
        REF IS oid SYSTEM GENERATED);
```

Código SQL:

```
DROP TABLE poseer;
DROP TRIGGER operacion_autoseq;
DROP SEQUENCE operacion_seq;
DROP TABLE operacion;
DROP TABLE cuenta;
DROP TABLE oficina;
DROP TABLE cliente;
```

- **Oracle:**

```
CREATE OR REPLACE TYPE cliente_udt AS OBJECT(
        dni number(8),
         nombre varchar(30),
         apellidos varchar(60),
         direccion varchar(80),
        email varchar(80),
         telefono number(9),
        fechNacimiento date);

CREATE TABLE cliente OF cliente_udt(
        CONSTRAINT pk_cliente PRIMARY KEY(dni),
        CONSTRAINT nombre_check CHECK(nombre IS NOT NULL),
        CONSTRAINT apellidos_check CHECK(apellidos IS NOT NULL),
        CONSTRAINT direccion_check CHECK(direccion IS NOT NULL),
        CONSTRAINT telefono_check CHECK(telefono IS NOT NULL),
        CONSTRAINT fechNacimiento_check CHECK(fechNacimiento IS NOT NULL));


CREATE OR REPLACE TYPE oficina_udt AS OBJECT(
        codigo number(4),
        direccion varchar(80),
        telefono number(9));

CREATE TABLE oficina OF oficina_udt(
        CONSTRAINT pk_oficina PRIMARY KEY(codigo),
        CONSTRAINT direccionOf_check CHECK(direccion IS NOT NULL),
        CONSTRAINT telefonoOf_check CHECK(telefono IS NOT NULL));
```

```
CREATE OR REPLACE TYPE cuenta_udt AS OBJECT(
        nCuenta number(20),
        fecha date,
        saldo real,
        tipo number(1),
        interes number(3,3),
        codigo number(4));

CREATE TABLE cuenta OF cuenta_udt(
        CONSTRAINT pk_cuenta PRIMARY KEY(nCuenta),
        saldo DEFAULT 0,
        fecha DEFAULT SYSDATE,
        CONSTRAINT fk_cc_oficina FOREIGN KEY (codigo) REFERENCES oficina(codigo),
        CONSTRAINT fechCuenta_check CHECK(fecha IS NOT NULL),
        CONSTRAINT interes_check CHECK(interes IS NULL OR interes>0 OR interes
<=100),
        CONSTRAINT tipoCuenta_check CHECK(tipo IS NOT NULL AND ((tipo=0 AND
codigo IS NOT NULL AND interes IS NULL) OR (tipo=1 AND interes IS NOT NULL AND
codigo IS NULL))));


CREATE OR REPLACE TYPE operacion_udt AS OBJECT(
        nCuenta_origen number(20),
        num_operacion number(6),
        fecha date,
        descripcion varchar(250),
        cantidad real,
        tipo number(1),
        nCuenta_dst number(20),
        codigo number(4));

CREATE TABLE operacion OF operacion_udt(
        CONSTRAINT pk_operacion PRIMARY KEY(nCuenta_origen, num_operacion),
        CONSTRAINT fk_nCuenta_origen FOREIGN KEY(nCuenta_origen) REFERENCES
cuenta(nCuenta),
        CONSTRAINT fk_codigo_op FOREIGN KEY(codigo) REFERENCES oficina(codigo),
        CONSTRAINT fk_nCuenta_dst FOREIGN KEY(nCuenta_dst) REFERENCES
cuenta(nCuenta),
        fecha DEFAULT SYSDATE,
        CONSTRAINT fecha_op_check CHECK(fecha IS NOT NULL),
        CONSTRAINT cantidad_check CHECK(cantidad IS NOT NULL AND cantidad>0),
```

CONSTRAINT *tipo_op_check* ***CHECK****(tipo* ***IS NOT NULL AND*** *((tipo=0* ***AND*** *codigo* ***IS NOT NULL AND*** *nCuenta_dst* ***IS NULL****)* ***OR*** *(tipo=1* ***AND*** *nCuenta_dst* ***IS NOT NULL AND*** *codigo* ***IS NULL****))));*

CREATE SEQUENCE *operacion_seq* ***START WITH*** *1* ***INCREMENT BY*** *1 NOMAXVALUE;*

CREATE TRIGGER *operacion_autoseq*
BEFORE INSERT ON *operacion*
FOR EACH ROW
BEGIN
SELECT *operacion_seq.nextval* ***INTO*** *:****new****.num_operacion* ***from*** *dual;*
END*;*

CREATE OR REPLACE TYPE *poseer_udt* ***AS OBJECT****(*
cliente ***number****(8),*
nCuenta ***number****(20));*

CREATE TABLE *poseer* ***OF*** *poseer_udt(*
CONSTRAINT *pk_poseer* ***PRIMARY KEY****(cliente,nCuenta),*
CONSTRAINT *fk_cliente_poseer* ***FOREIGN KEY****(cliente)* ***REFERENCES*** *cliente(dni),*
CONSTRAINT *fk_cuenta_poseer* ***FOREIGN KEY****(nCuenta)* ***REFERENCES*** *cuenta(nCuenta));*

- **PostgreSQL:**

CREATE TYPE *cliente_udt* ***AS****(*
dni ***decimal****(8),*
nombre ***varchar****(30),*
apellidos ***varchar****(60),*
direccion ***varchar****(60),*
email ***varchar****(80),*
telefono ***decimal****(9),*
fechNacimiento ***date****);*

CREATE TABLE *cliente* ***OF*** *cliente_udt(*
CONSTRAINT *pk_cliente* ***PRIMARY KEY****(dni),*
CONSTRAINT *nombre_check* ***CHECK****(nombre* ***IS NOT NULL****),*
CONSTRAINT *apellidos_check* ***CHECK****(apellidos* ***IS NOT NULL****),*
CONSTRAINT *direccion_check* ***CHECK****(direccion* ***IS NOT NULL****),*
CONSTRAINT *telefono_check* ***CHECK****(telefono* ***IS NOT NULL****),*

```
        CONSTRAINT fechNacimiento_check CHECK(fechNacimiento IS NOT
        NULL));


CREATE TYPE oficina_udt AS(
        codigo decimal(4),
        direccion varchar(80),
        telefono decimal(9));

CREATE TABLE oficina OF oficina_udt(
        CONSTRAINT pk_oficina PRIMARY KEY(codigo),
        CONSTRAINT direccionOf_check CHECK(direccion IS NOT NULL),
        CONSTRAINT telefonoOf_check CHECK(telefono IS NOT NULL));


CREATE TYPE cuenta_udt AS(
         nCuenta decimal(20),
         fecha date,
        saldo real,
        tipo decimal(1),
        interes decimal(3,3),
        codigo decimal(4));

CREATE TABLE cuenta OF cuenta_udt(
        CONSTRAINT pk_cuenta PRIMARY KEY(nCuenta),
        CONSTRAINT fk_cc_oficina FOREIGN KEY (codigo) REFERENCES oficina(codigo),
        CONSTRAINT fechCuenta_check CHECK(fecha IS NOT NULL),
        CONSTRAINT interes_check CHECK(interes IS NULL OR interes>0 AND interes
<=100),
        CONSTRAINT tipoCuenta_check CHECK(tipo IS NOT NULL AND ((tipo=0 AND
codigo IS NOT NULL AND interes IS NULL) OR (tipo=1 AND interes IS NOT NULL AND
codigo IS NULL))));

ALTER TABLE cuenta
ALTER COLUMN fecha SET DEFAULT CURRENT_DATE,
ALTER COLUMN saldo SET DEFAULT 0;


CREATE TYPE operacion_udt AS(
        nCuenta_origen decimal(20),
        num_operacion decimal(6),
```

```
        fecha date,
        descripcion varchar(250),
        cantidad real,
        tipo decimal(1),
        nCuenta_dst decimal(20),
        codigo decimal(4));

CREATE TABLE operacion OF operacion_udt(
        CONSTRAINT pk_operacion PRIMARY KEY(nCuenta_origen, num_operacion),
        CONSTRAINT fk_nCuenta_origen FOREIGN KEY(nCuenta_origen) REFERENCES
cuenta(nCuenta),
        CONSTRAINT fk_codigo_op FOREIGN KEY(codigo) REFERENCES oficina(codigo),
        CONSTRAINT fk_nCuenta_dst FOREIGN KEY(nCuenta_dst) REFERENCES
cuenta(nCuenta),
        CONSTRAINT fecha_op_check CHECK(fecha IS NOT NULL),
        CONSTRAINT cantidad_check CHECK(cantidad IS NOT NULL AND cantidad>0),
        CONSTRAINT tipo_op_check CHECK(tipo IS NOT NULL AND ((tipo=0 AND codigo
IS NOT NULL AND nCuenta_dst IS NULL) OR (tipo=1 AND nCuenta_dst IS NOT NULL
AND codigo IS NULL))));

CREATE SEQUENCE operacion_seq;

ALTER TABLE operacion
ALTER COLUMN fecha SET DEFAULT CURRENT_DATE,
ALTER COLUMN num_operacion SET DEFAULT NEXTVAL('operacion_seq');


CREATE TYPE poseer_udt AS(
        cliente decimal(8),
        nCuenta decimal(20));

CREATE TABLE poseer OF poseer_udt(
        CONSTRAINT pk_poseer PRIMARY KEY(cliente,nCuenta),
        CONSTRAINT fk_cliente_poseer FOREIGN KEY(cliente) REFERENCES cliente(dni),
        CONSTRAINT fk_cuenta_poseer FOREIGN KEY(nCuenta) REFERENCES
cuenta(nCuenta));
```

- **db2**

Para insertar operaciones en db2 se debe utilizar para la columna "num_operacion" el valor "NEXT VALUE FOR numOperacion_seq"

Para generar valores automáticamente a los OID de los diferentes tipos, se ha creado una secuencia y un trigger para cada tipo.

```
CREATE SCHEMA schema1;
SET SCHEMA schema1;

CREATE TYPE cliente_udt AS(
        dni numeric(8),
        nombre varchar(30),
        apellidos varchar(60),
        direccion varchar(80),
        email varchar(80),
        telefono numeric(9),
         fechNacimiento date)
REF USING INT MODE DB2SQL;

CREATE TABLE cliente of schema1.cliente_udt(
        REF IS oid USER GENERATED,
        dni WITH OPTIONS NOT NULL,
        nombre WITH OPTIONS NOT NULL,
        apellidos WITH OPTIONS NOT NULL,
        direccion WITH OPTIONS NOT NULL,
        telefono WITH OPTIONS NOT NULL,
        fechNacimiento WITH OPTIONS NOT NULL,
        CONSTRAINT pk_cliente PRIMARY KEY(dni));

CREATE SEQUENCE clienteOid AS REF(schema1.cliente_udt;

CREATE TRIGGER Gen_cliente_oid
        NO CASCADE
        BEFORE INSERT ON cliente
        REFERENCING NEW AS new
        FOR EACH ROW
        MODE DB2SQL
        SET new.oid = NEXT VALUE FOR clienteOid;
```

```
CREATE TYPE oficina_udt AS(
        codigo numeric(4),
        direccion varchar(80),
        telefono numeric(9))
REF USING INT MODE DB2SQL;

CREATE TABLE schema1.oficina OF schema1.oficina_udt(
        REF IS oid USER GENERATED,
        codigo WITH OPTIONS NOT NULL,
        direccion WITH OPTIONS NOT NULL,
        telefono WITH OPTIONS NOT NULL,
        CONSTRAINT pk_oficina PRIMARY KEY(codigo));

CREATE SEQUENCE oficinaOid AS REF(schema1.oficina_udt);

CREATE TRIGGER Gen_oficina_oid
        NO CASCADE
        BEFORE INSERT ON oficina
        REFERENCING NEW AS new
        FOR EACH ROW
        MODE DB2SQL
        SET new.oid = NEXT VALUE FOR oficinaOid;


CREATE TYPE cuenta_udt AS(
        nCuenta numeric(20),
        fecha date,
        saldo real,
        tipo numeric(1),
        interes numeric(3,3),
        codigo numeric(4))
REF USING INT MODE DB2SQL;

CREATE TABLE schema1.cuenta OF schema1.cuenta_udt(
        REF IS oid USER GENERATED,
        nCuenta WITH OPTIONS NOT NULL,
        fecha WITH OPTIONS NOT NULL DEFAULT CURRENT_DATE,
        saldo WITH OPTIONS DEFAULT 0,
        CONSTRAINT pk_cuenta PRIMARY KEY(nCuenta),
        CONSTRAINT fk_cc_oficina FOREIGN KEY (codigo) REFERENCES oficina(codigo),
```

```
    CONSTRAINT interes_check CHECK(interes IS NULL OR interes>0 OR interes
<=100),
    CONSTRAINT tipoCuenta_check CHECK(tipo IS NOT NULL AND ((tipo=0 AND
codigo IS NOT NULL AND interes IS NULL) OR (tipo=1 AND interes IS NOT NULL AND
codigo IS NULL))));

CREATE SEQUENCE cuentaOid AS REF(schema1.cuenta_udt);

CREATE TRIGGER Gen_cuenta_oid
    NO CASCADE
    BEFORE INSERT ON cuenta
    REFERENCING NEW AS new
    FOR EACH ROW
    MODE DB2SQL
    SET new.oid = NEXT VALUE FOR cuentaOid;


CREATE TYPE operacion_udt AS(
    nCuenta_origen numeric(20),
    num_operacion numeric(6),
    fecha date,
    descripcion varchar(250),
    cantidad real,
    tipo numeric(1),
    nCuenta_dst numeric(20),
    codigo numeric(4))
REF USING INT MODE DB2SQL;

CREATE SEQUENCE numOperacion_seq AS INT
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE
    NO CYCLE
    NO CACHE
    ORDER;

CREATE TABLE schema1.operacion OF schema1.operacion_udt(
    REF IS oid USER GENERATED,
    nCuenta_origen WITH OPTIONS NOT NULL,
    num_operacion WITH OPTIONS NOT NULL,
```

```
        fecha WITH OPTIONS NOT NULL DEFAULT CURRENT_DATE,
        cantidad WITH OPTIONS NOT NULL,
        CONSTRAINT pk_operacion PRIMARY KEY(nCuenta_origen, num_operacion),
        CONSTRAINT fk_nCuenta_origen FOREIGN KEY(nCuenta_origen) REFERENCES
schema1.cuenta(nCuenta),
        CONSTRAINT fk_codigo_op FOREIGN KEY(codigo) REFERENCES
schema1.oficina(codigo),
        CONSTRAINT fk_nCuenta_dst FOREIGN KEY(nCuenta_dst) REFERENCES
schema1.cuenta(nCuenta),
        CONSTRAINT cantidad_check CHECK(cantidad>0),
        CONSTRAINT tipo_op_check CHECK(tipo IS NOT NULL AND ((tipo=0 AND codigo
IS NOT NULL AND nCuenta_dst IS NULL) OR (tipo=1 AND nCuenta_dst IS NOT NULL
AND codigo IS NULL))));

CREATE TRIGGER operacion_autoseq
        NO CASCADE
        BEFORE INSERT ON operacion
        REFERENCING NEW AS new_operacion
        FOR EACH ROW MODE DB2SQL
        SET new_operacion.num_operacion = NEXT VALUE FOR numOperacion_seq;

CREATE SEQUENCE operacionOid AS REF(schema1.operacion_udt);

CREATE TRIGGER Gen_operacion_oid
        NO CASCADE
        BEFORE INSERT ON operacion
        REFERENCING NEW AS new
        FOR EACH ROW
        MODE DB2SQL
        SET new.oid = NEXT VALUE FOR operacionOid;


CREATE TYPE poseer_udt AS(
        cliente numeric(8),
        nCuenta numeric(20))
REF USING INT MODE DB2SQL;

CREATE TABLE poseer OF schema1.poseer_udt(
        REF IS oid USER GENERATED,
        cliente WITH OPTIONS NOT NULL,
        nCuenta WITH OPTIONS NOT NULL,
```

```
        CONSTRAINT pk_poseer PRIMARY KEY(cliente,nCuenta),
        CONSTRAINT fk_cliente_poseer FOREIGN KEY(cliente) REFERENCES cliente(dni),
        CONSTRAINT fk_cuenta_poseer FOREIGN KEY(nCuenta) REFERENCES
cuenta(nCuenta));

CREATE SEQUENCE poseerOid AS REF(schema1.poseer_udt);

CREATE TRIGGER Gen_poseer_oid
        NO CASCADE
        BEFORE INSERT ON poseer
        REFERENCING NEW AS new
        FOR EACH ROW
        MODE DB2SQL
        SET new.oid = NEXT VALUE FOR poseerOid;
```

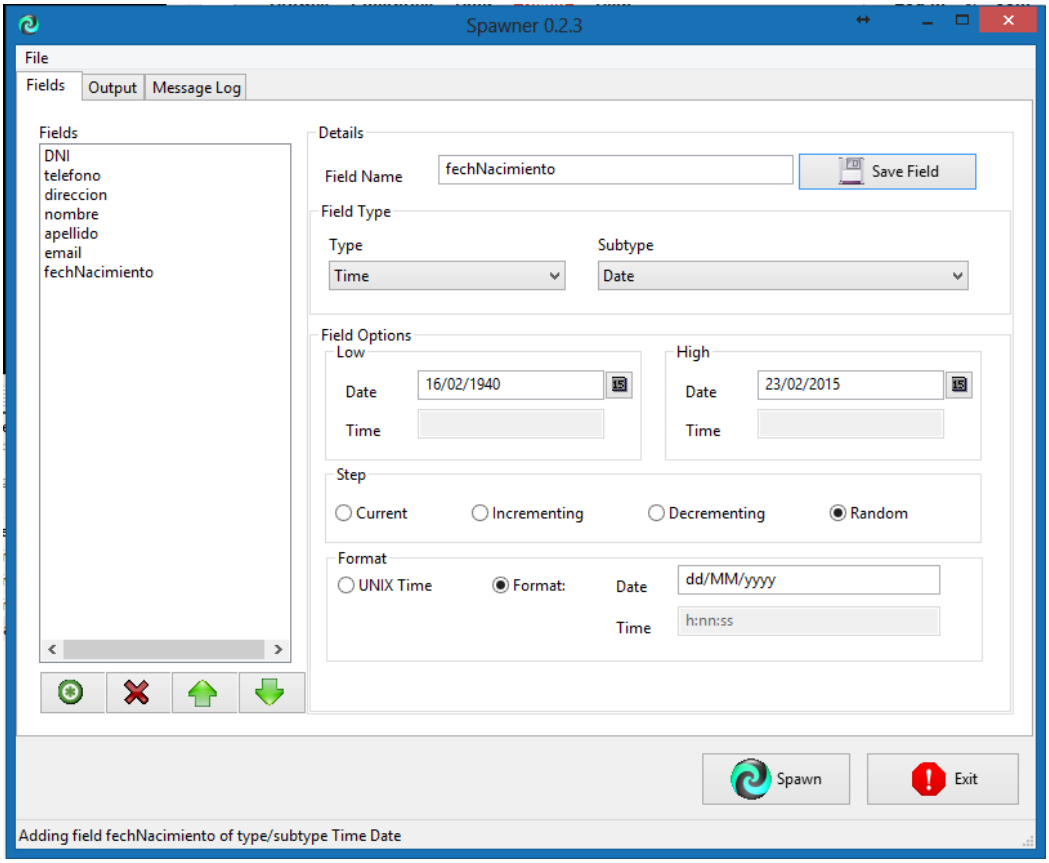
Generación de datos y pruebas:

1. Generación de datos

- **Generatedata (www.generatedata.com):**
  Herramienta de generación automática de datos que posee una interfaz web. Valido para Oracle y mySQL. Nos ofrece 5 formatos de salida: html, xls, cvs y sql.
  Problemas: Imposibilidad de añadir tipos de datos ya que son predeterminados y el máximo número de filas es de 100.

- **Datagenerator (www.sourceforge.net/projects/datagenerator):**
  Herramienta que permite exportar directamente los datos generados a una base de datos o bien en un fichero sql o csv.
  Problemas: Escasa documentación. Es necesario definir un archivo de configuración XML para cada tabla.

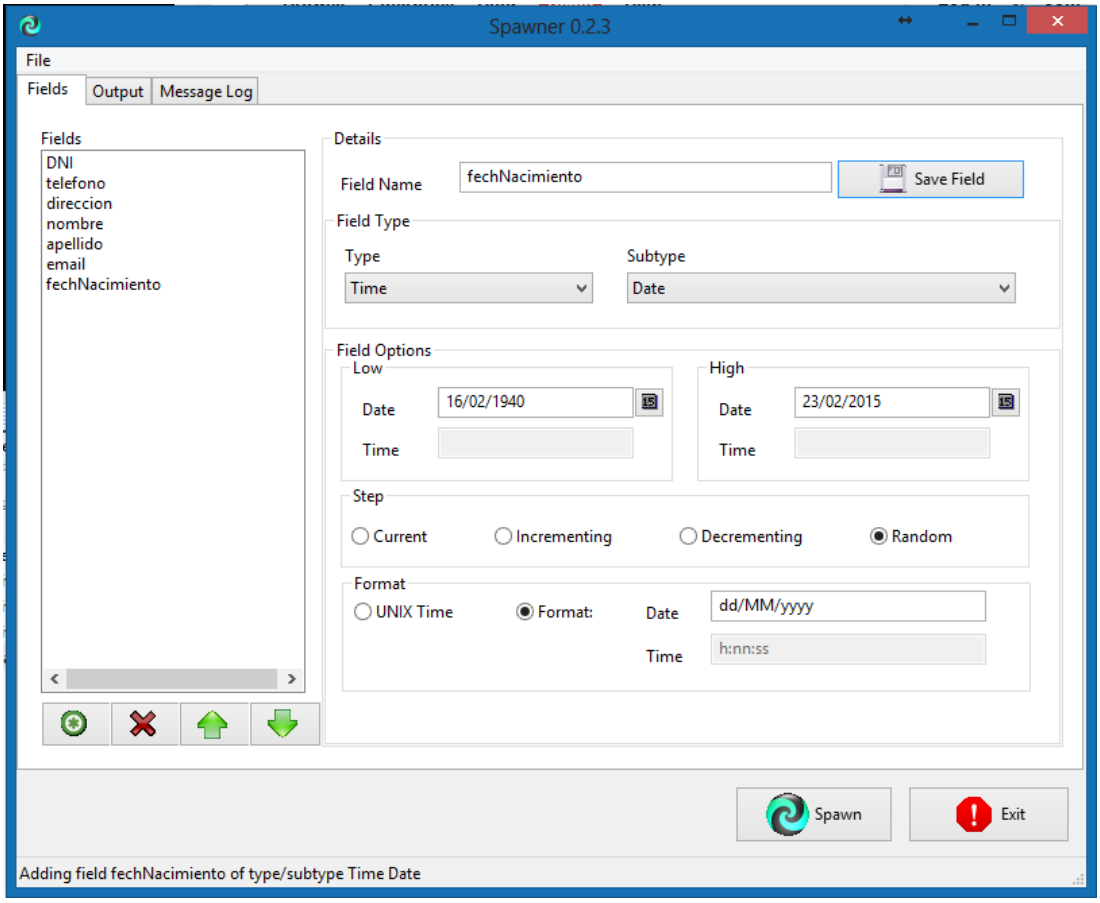
- **Spawner (www.spawner.sourceforge.net):**
  Herramienta con una interfaz sencilla, que permite la posibilidad de exportar los datos generados a una base de datos MySQL o a un fichero de texto.
  Problemas: Datos ya predeterminados y por lo tanto no se pueden añadir nuevos.



- **Datanamic (www.datanamic.com/datagenerator/index.html):** Herramienta de pago pero con 14 días de prueba. Multiplataforma (Oracle, MySQL, Postgresql, ...). Permite definir patrones para así ajustarlos a los tipos de datos y establecer relaciones con otras tablas. Permite la inserción directa de los datos generados a la base de datos.

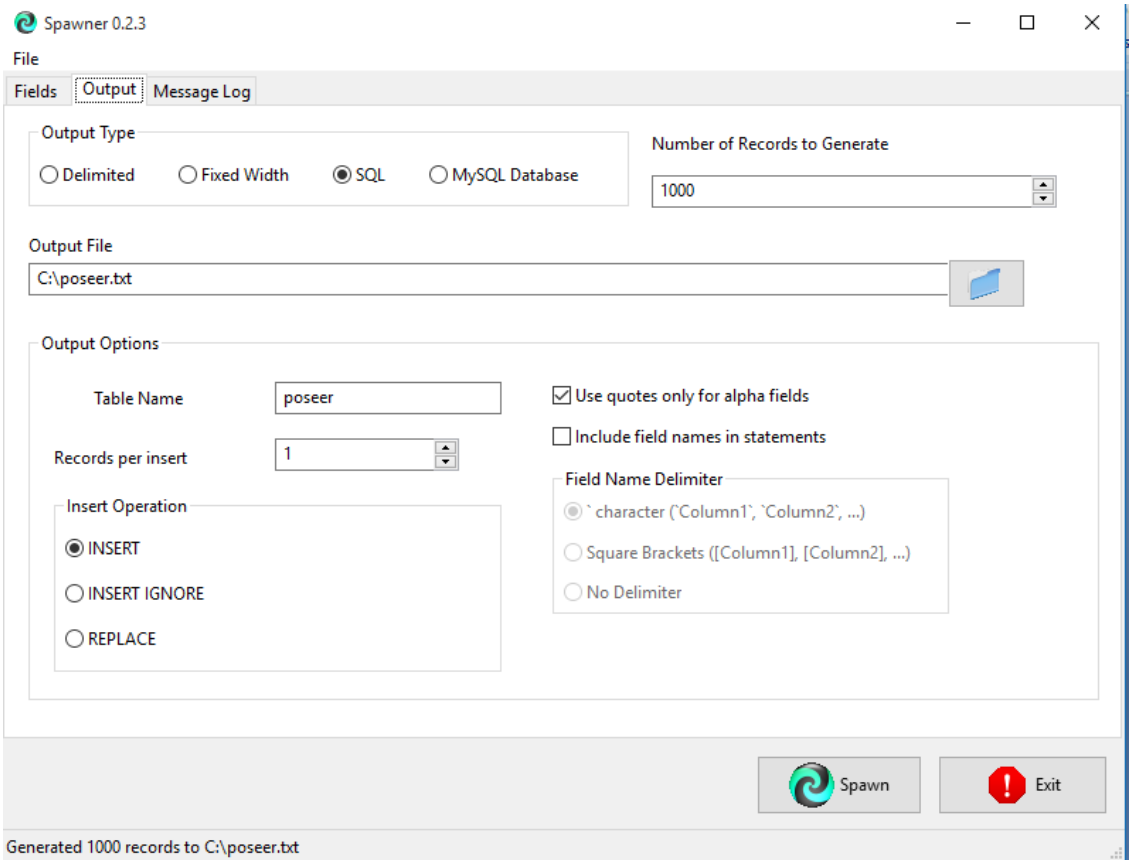
Conclusión:

Optaremos por utilizar **Spawner**, ya que se trata de una herramienta gratuita y es la que mejor se adapta además de tener una buena facilidad de uso.



Para el uso de esta herramienta simplemente deberemos ir añadiendo los diferentes campos de la tabla que queremos crear, para ello deberemos seleccionar el tipo de dato y el programa nos proporcionara diferentes formatos (enteros, reales, direcciones, ...), una vez seleccionado el que queremos podremos ajustar los valores, ya sea definiendo rangos, números de palabras, ...

Una vez tenemos la tabla procederemos a generar los datos, en nuestro caso los generaremos en SQL:



Pondremos que lo queremos en formato SQL, el fichero donde se generaran las sentencias, el número de sentencias y el nombre de la tabla en las que las ejecutaremos. Con esto obtendremos un fichero de texto con todas las sentencias necesarias para crear un script para insertar todos los datos en nuestra base de datos.

2. Pruebas

Una vez generados los distintos ficheros de prueba de inserción de datos (se ha generado un fichero **".sql"** por tabla para ello con las sentencias correspondientes) procedemos a realizar una serie de consultas para simular algunas de las posibles funciones que podría tener la base de datos implementada :

1) Se desea saber el numero de clientes que poseen una cuenta corriente en una sucursal 5044.

   Solución trivial(gracias a vistas previas) :

   SELECT count(*) FROM cuentas_corrientes C ,poseer P
   WHERE C.nCuenta = P.nCuenta AND C.idOficina = 5044;

   Alternativa :
   SELECT count(*) FROM poseer P, cuentas C where P.nCuenta = C.nCuenta AND C.tipo = 0 AND C.idOficina = 5044;

   Dando como salida:

```
COUNT(*)
---------
3
```

2) Se desea consultar el nombre, e-mail y teléfono de los clientes anteriores :

SELECT C.nombre,C.email,C.telefono FROM clientes C,poseer P, cuentas C where P.nCuenta = C.nCuenta AND C.idOficina = 5044 AND C.dni = P.dni;

Alternativa:
SELECT C.nombre,C.email,C.telefono FROM clientes C,poseer P, cuentas C where P.nCuenta = C.nCuenta AND C.tipo = 0 AND C.idOficina = 5044 AND C.dni = P.dni;

Dando como salida **:**

```
NOMBRE
--------------------------------------------------
----
EMAIL
--------------------------------------------------
----
  TELEFONO
----------
Amery
Nero@iaculis.net
 635947311

Bell
Azalia@viverra.com
 914067170
```

3) Se desea consultar las retiradas realizadas en un periodo de fechas concreto dado un DNI de un cliente:

CREATE VIEW RETIRADASDNI AS

SELECT C.dni FROM clientes C, ingresos I WHERE C.dni =

Primero se crearía la vista que engloba todos los datos de los ingresos del dni correspondiente.

A continuación se consulta delimitando el rango de fechas con la clausula "BETWEEN".

SELECT * from INGRESOSFECHA WHERE fechaHora Between '01/01/1910' AND '01/01/1951';

Obteniendo como salida:

```
NOPERACION
--------------------------------------------------
----
```

```
FECHAHORA
-------------------------------------------------------
----
CANTIDAD
TIPOOP
DESCRIPCION
-------------------------------------------------------
----

834763044
03/05/1919 17:22:30
834.05
1
abril 19

951157899
13/05/1949 11:12:00
60.04
1
concepto de luz y agua

436830637
23/05/1950 09:31:46
10.1
abril 50
```

Uno de los problemas encontrados es que el generador ha generado datos de tipo DATE y TIMESTAMP con el formato DD/MM/YYYY cuando por defecto se trata de formato YYYY-MM-DD por lo que se debe reestructurar el formato del tipo de dato antes de realizar cualquier inserción o consulta.

Con estas dos sentencias cambiamos primero el formato del tipo date y a continuación el del tipo timestamp

alter session set nls_date_format='DD/MM/YYYY HH24:MI';

alter session set nls_timestamp_format='DD/MM/YYYY ;

También el generador nos ha provocado conflicto con datos de tipo FLOAT ya que en lugar de usar "." Para la parte decimal usaba ",". Mediante prueba y error hemos sido capaces de conseguir solventar estos problemas. Se ha medio solventado a mano.

Implementación con db4o:

- **ODL:**

```
class cliente (key dni) {
        attribute int dni;
        attribute string nombre;
        attribute string apellidos;
        attribute string direccion;
        attribute string email;
        attribute int telefono;
        attribute date fechaNacimiento;

        constraint<notnull> on nombre;
        constraint<notnull> on apellidos;
        constraint<notnull> on direccion;
        constraint<notnull> on telefono;
        constraint<notnull> on fechaNacimiento;
}

class oficina (key idOficina) {
        attribute int idOficina;
        attribute string direccion;
        attribute int telefono;

        constraint<notnull> on direccion;
        constraint<notnull> on telefono;
}

class cuenta (key nCuenta) {
        attribute long nCuenta;
        attribute date fecha;
        attribute float saldo;
        attribute int tipo;
        attribute float interes;
        attribute int idOficina;

        constraint<notnull> on fecha;
        constraint<notnull> on saldo;
```

```
        constraint<notnull> on tipo;
}

class CC extends cuenta {
        relationship int idOficina;
}

class CA extends cuenta {
        attribute float interes;
}

class operacion (key nOperacion) {
        relationship cuenta nCuenta_origen;
        attribute long nOperacion;
        attribute date fechaHora;
        attribute float cantidad;
        attribute int tipoOp;
        attribute string descripcion;
        attribute int idOficina;
        attribute float origen;
        attribute float destino;

        constraint<notnull> on fechaHora;
        constraint<notnull> on cantidad;
        constraint<notnull> on tipoOp;
}

class ingreso/retirada extends operacion {
        relationship int idOficina;
}

class transferencia extends operacion {
        relationship cuenta nCuenta_destino;
}
```

- **Instalación:**

Para instalar Db4o es necesario descargar el fichero db4o-8.0.276.16149-java. Una vez descargado, descomprimir el paquete db4o-*-java.zip, acceder a la carpeta "/ome" y descomprimir el archivo "ObjectManagerEnterprise.zip". Se trata de una herramienta disponible para Eclipse que permite la navegación y la gestión de bases de datos db4o a través de una interfaz gráfica. Se puede ver cómo se almacenan los objetos y examinar la estructura. También posee un generador de consultas además de la posibilidad de modificar o eliminar objetos de la base de datos.

Para instalar el plugin es necesario abrir Eclipse y seleccionar Help -> Install new Software...-> Available Software y elegir Add Site -> Local y seleccionar la carpeta descomprimida del OME.
Para visualizar la interfaz seleccionar Window -> Open Perspective -> Other y seleccionar OME.

Para poder funcionar correctamente es necesario copiar la librería "db4o-all" a la carpeta /lib del proyecto y añadirla al proyecto de Eclipse (en propiedades -> Java Build Path -> Add external JARs y seleccionando la ubicación de la librería).

- **Código JAVA:**

Para la implementación en db4o hemos creado una clase por cada objeto existente, definiéndonos en cada uno de ellos métodos "get" y "set" para poder trabajar con sus datos, por ejemplo:

Para utilizarlo lo primero que debemos hacer es crearnos una nueva base de datos:

```
ObjectContainer db = Db4oEmbedded.openFile(
                Db4oEmbedded.newConfiguration(), DB4OFILENAME);
```

Donde *DB4OFILENAME* Es el nombre de nuestra base de datos. Una vez la tengamos, podremos hacer las consultas sobre esa base de datos, por ejemplo:

```
// Consulta que muestra todos los clientes del banco
ObjectSet result = db.queryByExample(Cliente.class);
listResult(result);

// Consulta que muestra todos los clientes que se llamen "Peter"
Cliente cli = new Cliente(0, "Peter", null, null, null, 0, null);
result = db.queryByExample(cli);
listResult(result);
```

```
// Inserción de una nueva cuenta en un cliente
public static void insertarCuentas(ObjectContainer db) {
        // RESTRICCION :
        // NO SE PUEDEN AÑADIR CUENTAS CON MISMO NUM_CUENTA
        System.out.println("Insertando Cuentas...");
        try {
                Scanner scan = new Scanner(new File(FICH_CUENTAS));
                String linea;
                String[] campo;
                while (scan.hasNextLine()) {
                        linea = scan.nextLine();
                        campo = linea.split(",");
                        Cuenta c = buscarCuenta(db, Integer.parseInt(campo[0]));

                        if (c == null) { // comprueba que no existe la cuenta
                                if (campo[3].equals("0")) { // CC
                                        Cuenta cuenta = new
                                                CCorriente(Integer.parseInt(campo[0]),
                                        campo[1], Double.parseDouble(campo[2]),
                                        Integer.parseInt(campo[5]));
                                        db.store(cuenta);
                                        } else { // CA
                                                Cuenta cuenta = new

        CAhorro(Integer.parseInt(campo[0]),
                                                campo[1], Double.parseDouble(campo[2]),
                                                Float.parseFloat(campo[4]));
                                                db.store(cuenta);
                                        }
                                } else {
                                        System.out.println("Error: la cuenta ya existe.");
                                }
                        }
                        scan.close();
                } catch (Exception e) {
                        e.printStackTrace();
                        e.getMessage();
                }
        }
```

Como se puede ver, para el uso de Db4o, si se desean hacer consultas complejas se requieren ciertos conocimientos de programación.

Comparación entre los SGBD:

Tras el trabajo realizado con los diferentes gestores, y aunque sería necesario probar los sistemas con usuarios reales para observarlos en un campo real y poder obtener mas información, hemos observado que MySQL es el mas rápido y fácil de instalar y configurar, ya que es un sistema prácticamente listo para funcionar. Todo lo contrario ocurre con Oracle y PostgreSQL, que se deben tener más aspectos en cuenta a la hora de su instalación y su configuración.

A la hora de crear las bases de datos y las tablas que las conforman hemos podido apreciar pequeñas diferencias entre los tipos de datos ofrecidos por cada gestor, ya que cada uno ofrece una forma de nombrarlos. Esto también ocurre con los disparadores, que también tienen una sintaxis ligeramente distinta, por ejemplo en el caso de PostgreSQL, en el que es necesario definirnos una función ajena.

En cuanto a las licencias tenemos, en el caso de Oracle que nos ofrece una versión gratuita básica, pero para desplegar su potencial deberemos adquirir una versión de pago. Lo mismo nos ocurre con DB2, que es de pago. Sin embargo MySQL y postgreSQL son totalmente gratuitos (en uso interno).