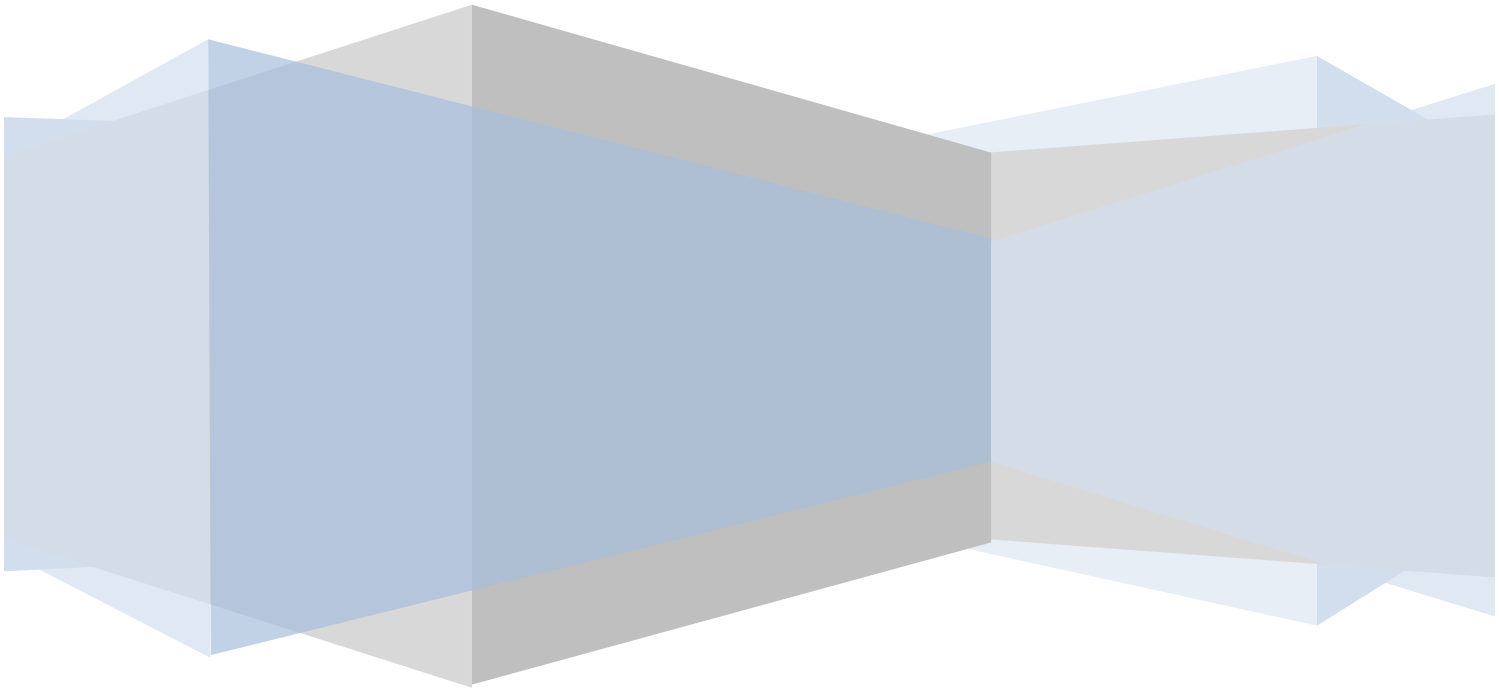
Universidad de Zaragoza

Práctica 3:

Diseño de Bases de Datos Federadas

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
P1 - Obtener Esquema de Banquete	Diego	3 h
P1 - Diseño conceptual de Banquete: E/R	Diego	1,5 h
P1 - Diseño lógico relacional de Banquete	Diego	1
P1 – Implementación	Adrián	4 h
P2 - Plantear problema de diseño de BD 1	Cristian	1 h
P2 - Diseño conceptual: E/R 1	Cristian	0,5 h
P2 - Diseño lógico relacional 1	Cristian	1 h
P2 – Implementación 1	Cristian	1,5 h
P2 - Plantear problema de diseño de BD 2	Sergio	1 h
P2 –Diseño conceptual : E/R 2	Sergio	1 h
P2 – Implementación 2	Sergio	2 h
P2 - Integración	Sergio	2 h

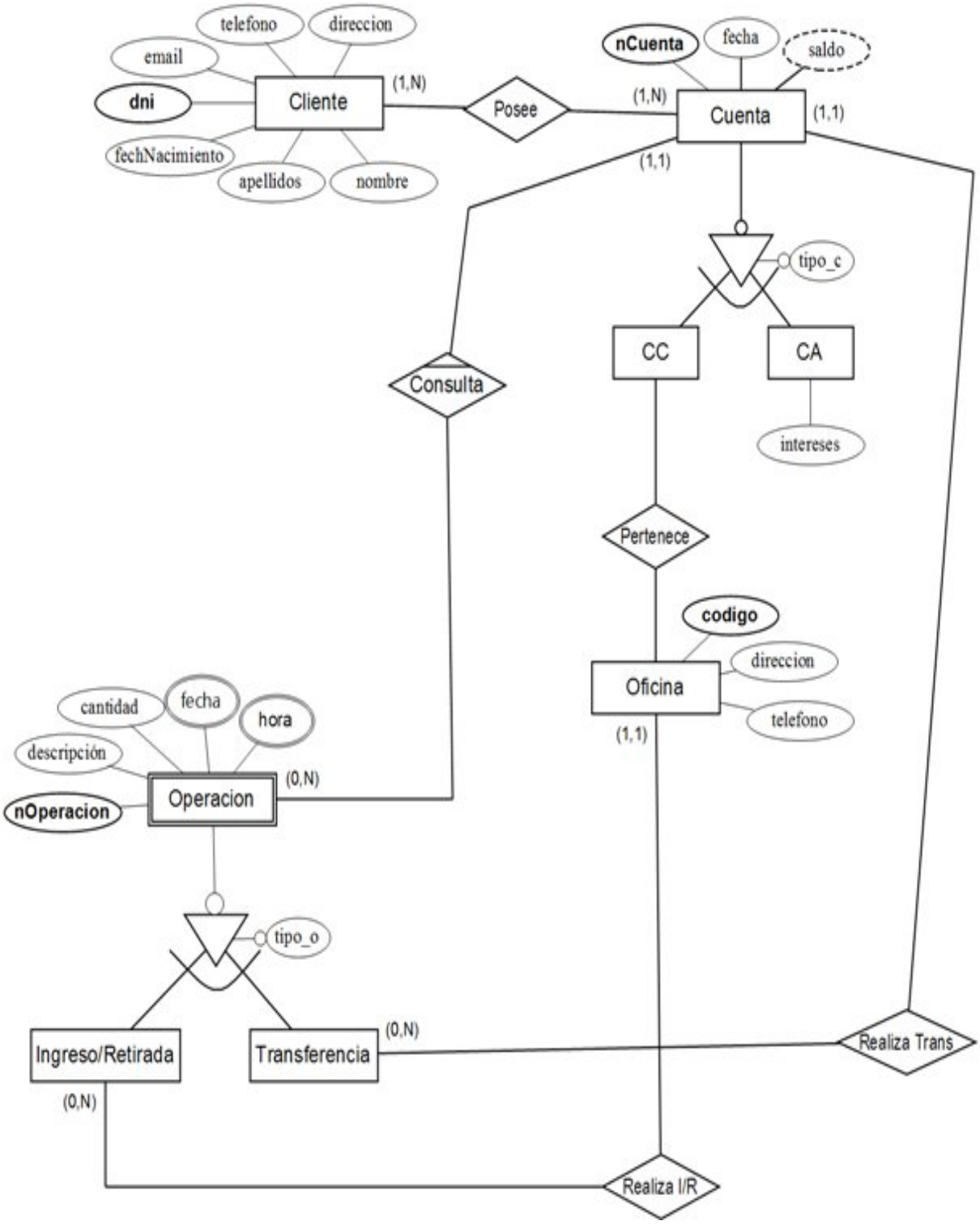
Objetivos:

Los objetivos de la siguiente practica son el realizar una integración de bases de datos para definir una base de datos federada. En mayor detalle:

- La conexión a una base de datos Oracle proporcionada y analizarla para tratar de obtener su esquema conceptual.
- Integrar el esquema de dicha base de datos con el desarrollado previamente en la practica 1 ofreciendo un esquema global.
- Definirse un nuevo problema para realizar una integración de dos bases de datos en PostgreSQL.

Parte 1:

Esquema conceptual y lógico de la base de datos relacional de la practica anterior



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);

)

Obtención del esquema de Banquete

Para obtener el esquema de la base de datos Banquete, la cual se encontraba en un SGBD de Oracle, se han utilizado principalmente 3 consultas.
Por un lado se han obtenido el nombre de las tablas con la siguiente consulta.

```
SQL> select table_name from user_tables@schema2bd2;

TABLE_NAME
------------------------------
CODENTIDADES
CODPOSTAL
CUENTA
CUENTAAHORRO
CUENTACORRIENTE
DIRECCION
OPEFECTIVO
OPERACION
OPTRANSFERENCIA
SUCURSAL
TITULAR

11 rows selected.
```

Como se observa hay 11 tablas.

A continuación se ha obtenido la sintaxis de cada tabla aplicando la consulta que se muestra a continuación, lógicamente hay que cambiar el atributo 'table_name' por nombre de la tabla que queremos obtener la información.

```
SQL> select column_name "Campo", data_type "Tipo de datos",
        data_length "Tamaño"
        from all_tab_columns@schema2bd2
        where table_name='TITULAR';  2    3    4

Campo
------------------------------
Tipo de datos
--------------------------------------------------------------------------------
    Tamaqo
----------
DNI
VARCHAR2
         9
```

De esta forma se han obtenido los atributos de cada entidad y su sintaxis, como se observa se obtiene el campo (nombre del atributo), tipo de datos y su tamaño.

Finalmente obtenemos las restricciones de cada tabla con la siguiente consulta.

```
SQL> SELECT cols.column_name, cons.constraint_type,     cons.search_condition
  2     FROM all_constraints@schema2bd2 cons,   all_cons_columns@schema2bd2 cols
  3     WHERE cols.table_name = 'TITULAR'
  4     AND cons.constraint_name = cols.constraint_name
  5     AND cons.owner = cols.owner
        ORDER BY cols.table_name, cols.position;
  6
COLUMN_NAME
--------------------------------------------------------------------------------
C
-
SEARCH_CONDITION
--------------------------------------------------------------------------------
DNI
P


DIRECCION
R


COLUMN_NAME
--------------------------------------------------------------------------------
C
-
SEARCH_CONDITION
--------------------------------------------------------------------------------

DNI
C
"DNI" IS NOT NULL
```

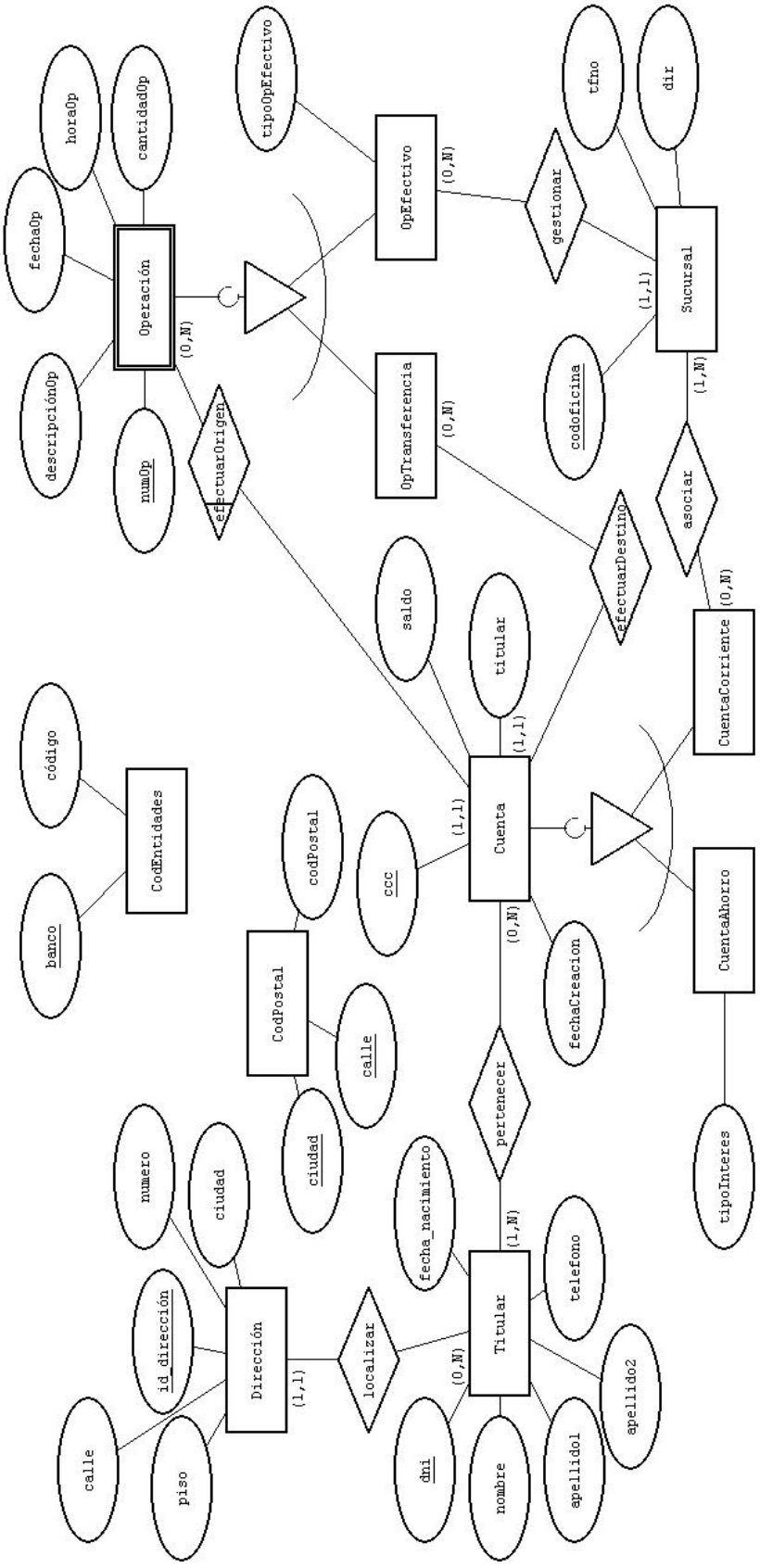
Como se observa se obtiene el atributo al que hace referencia la restricción, el tipo de restricción y la condición si la tiene. Los tipos de restricciones se relacionan de la siguiente forma: P: primary key, R: foreign key, C: constraint y U: unique. Una cosa que nos ha llamado la atención es que en la tabla 'cuenta' aparecen atributos sobre los cuales luego no se puede consultar consultar, como 'iban', 'tipo'…

Además se han consultado las vistas que tiene el esquema.

```
SQL>  select view_name from user_views@schema2bd2;

VIEW_NAME
------------------------------
INFOTRANSFERENCIAS

SQL> select * from infotransferencias@schema2bd2;

cuenta ordenante     cuenta receptora     fecha     hora  transferido beneficia
-------------------- -------------------- --------- ----- ----------- ---------
8763111113100000002  2345456798000123 45  02-OCT-07 13:15         300
8763222207000000012  8763333381000000013  03-APR-09 20:35        3000 16762311B
8763222207000000012  8763222202000000007  05-JUN-09 19:25         500 11565611B

SQL>
```

Como se observa hay definida una vista que engloba la información de las transferencias, ya que tal y como está el esquema, cuenta ordenante se encuentra en la tabla de 'operación' y cuenta receptora en 'opTransferencia', además que no se sabría el beneficiario, el cual se encuentra en la tabla 'cuenta'. De forma que permite obtener las transferencias de una forma más clara.

Esquema Conceptual de Banquete



Esquema Relacional Banquete

Dirección *(*

Id_direccion = tpNumero, clave primaria;
calle = tpTexto, NO NULO;
numero = tpNumero, NO NULO;
piso = tpTexto, NO NULO;
ciudad = tpTexto, NO NULO;

)

CodPostal (

calle = tpTexto, clave primaria;
ciudad = tpTexto, clave primaria;
codPostal = tpNumero, NO NULO;

)

Titular *(*

dni = tpDni, clave primaria;
nombre = tpTexto, NO NULO;
apellido1 = tpTexto, NO NULO;
apellido2 = tpTexto;
direccion= tpTexto, NO NULO;
telefono = tpTelefono, NO NULO;
fechNacimiento = tpFecha, NO NULO;

)

Relación 'localizar', como cada titular sólo tiene una dirección se exporta a la tabla titular.

Sucursal *(*

codOficina = tpNumero, clave primaria;
dir = tpTexto, NO NULO;
telefono = tpNumero, NO NULO;

)

CodEntidades (

banco = tpNumCuenta, clave primaria;
codigo = tpCodigo, NO NULO;

)

Cuenta *(*

ccc = tpNumCuenta, clave primaria;
fechaCreacion = tpFecha, NO NULO;
saldo = tpSaldo, NO NULO, POR DEFECTO 0;
titular = tpDni, clave foránea de titular, NO NULO;

)

Restricciones de **Cuenta**:

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

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

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

9) No puede haber una cuenta que se a la vez de ahorro y corriente.

CuentaAhorro (

ccc = tpNumCuenta, clave primaria, clave foránea de cuenta;
tipoInteres = tpNumero, NO NULO;

)

CuentaCorriente (

ccc = tpNumCuenta, clave primaria, clave foránea de cuenta;
sucursal_codOficina = tpNumero, clave foránea de sucursal;

)

Relación ‘asociar’: como una cuenta de corriente sólo puede estar asociada a una sucursal, se exporta el código de la sucursal a cuenta de corriente.

Operacion (

ccc = tpNumCuenta, clave ajena de Cuenta;
numOperacion = tpNumero;
fechaOp = tpFecha, NO NULO;
horaOp = tpHora, NO NULO;
descripcionOp = tpTexto;
cantidadOp = tpSaldo, NO NULO;

tipo_o = tpTipoOperacion, NO NULO;

)

opTransferencia (

numOp = tpNumero, clave primaria, clave ajena de Operacion;

ccc = tpNumCuenta, clave primaria, clave ajena de Cuenta;

cuentaDestino = tpNumCuenta;

)

opEfectivo (

numOp = tpNumero, clave primaria, clave ajena de Operación;

tipoOpEfectivo = tpTipoOp, NO NULO;

sucursal_codOficina = tpNumero, clave foránea de sucursal;

ccc = tpNumCuenta, clave primaria, clave foránea de cuenta;

)

Parte 2:

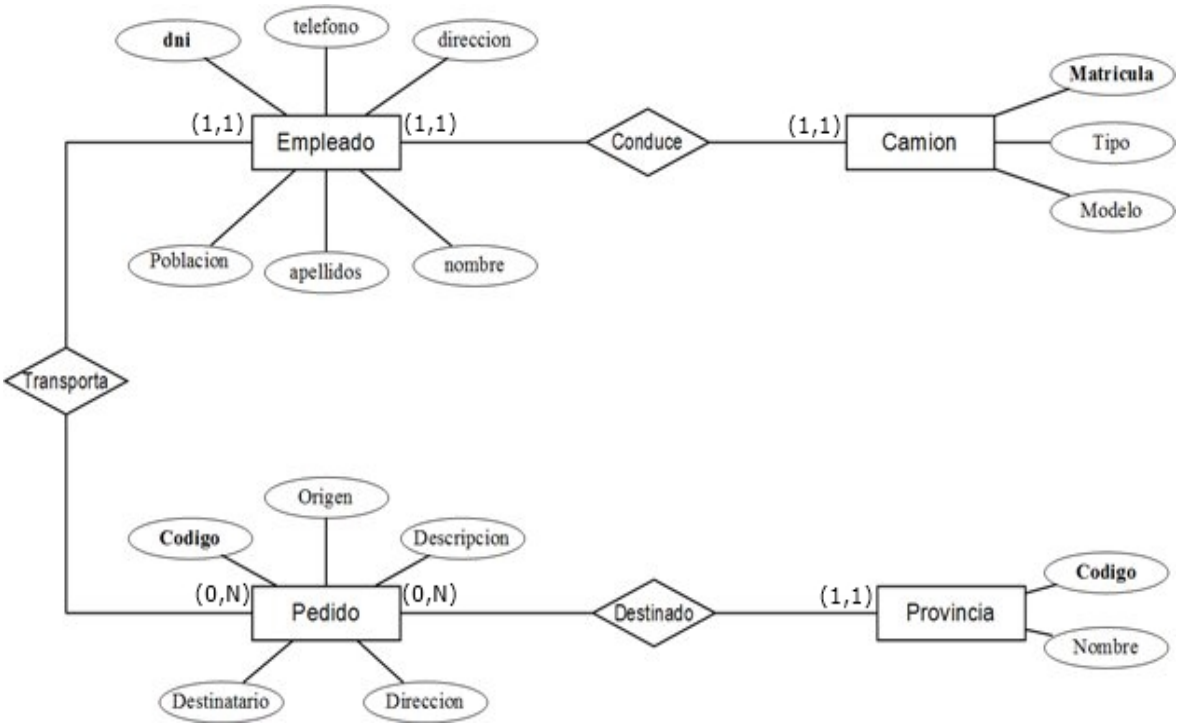
Planteamiento de un problema de diseño de bases de datos:

Tras la fusión de una de las más importantes cadenas de transporte de mercancías de España con otra empresa del sector, ambas con sus propias bases de datos para almacenar su información, desean realizar una integración de ambas bases de datos.

Para ello, nuestro objetivo es diseñar e implementar un esquema global que permita ver ambas bases de datos de manera uniforme, ya que por el momento se ha tomado la decisión de que ambas bases de datos mantendrán su autonomía. Dichas bases de datos están implementadas en PostgreSQL siguiendo el esquema relacional, lo que en teoría debería facilitarnos la tarea.

Como es lógico, para realizar la integración de ambas compañías, se usaran copias de las bases de datos que contendrán un subconjunto de datos de prueba. Posteriormente y tras un tiempo para garantizar que todo funciona como debe, se realizara la carga del conjunto de datos completo.

Esquema conceptual:



Esquema relacional:

Empleado

{

dni = tpDni, clave primaria;

nombre = tpTexto, NO NULO;

apellidos = tpTexto, NO NULO;

direccion= tpTexto, NO NULO;

telefono = tpTelefono, NO NULO;

fechNacimiento = tpFecha, NO NULO;

}

Camion

{

matricula = tpMatricula, clave primaria;

tipo = tpTexto, NO NULO;

modelo = tpTexto, NO NULO;

}

Pedido

{

codigo = tpCodigo, clave primaria;

origen = tpTexto, NO NULO;

descripcion = tpTexto, NO NULO;

destinatario = tpTexto, NO NULO;

direccion = tpTexto, NO NULO;

}

Provincia

{

codigoProvincia = tpCodigoProvincia, clave primaria;

nombre = tpTexto, NO NULO;

}

Transporta

{

dni = tpDni, clave ajena Empleado;

codigo = tpCodigo, clave ajena Pedido;

clave primaria (dni, codigo);

}

Conduce
{
dni = tpDni, clave ajena Empleado;
matricula = tpMatricula, clave ajena Camion;

clave primaria (dni, matricula);
}

Transporta
{
codigo = tpCodigo, clave ajena Pedido;
codigoProvincia = tpCodigoProvincia, clave ajena Provincia;

clave primaria (codigo, codigoProvincia);
}

Dominios:

tpDni = natural(8)
tpTelefono = natural(9)
tpMatricula = cadena(7)
tpCodigo = natural(8)
tpCodigoProvincia = natural(5)
tpTexto = cadena(250)

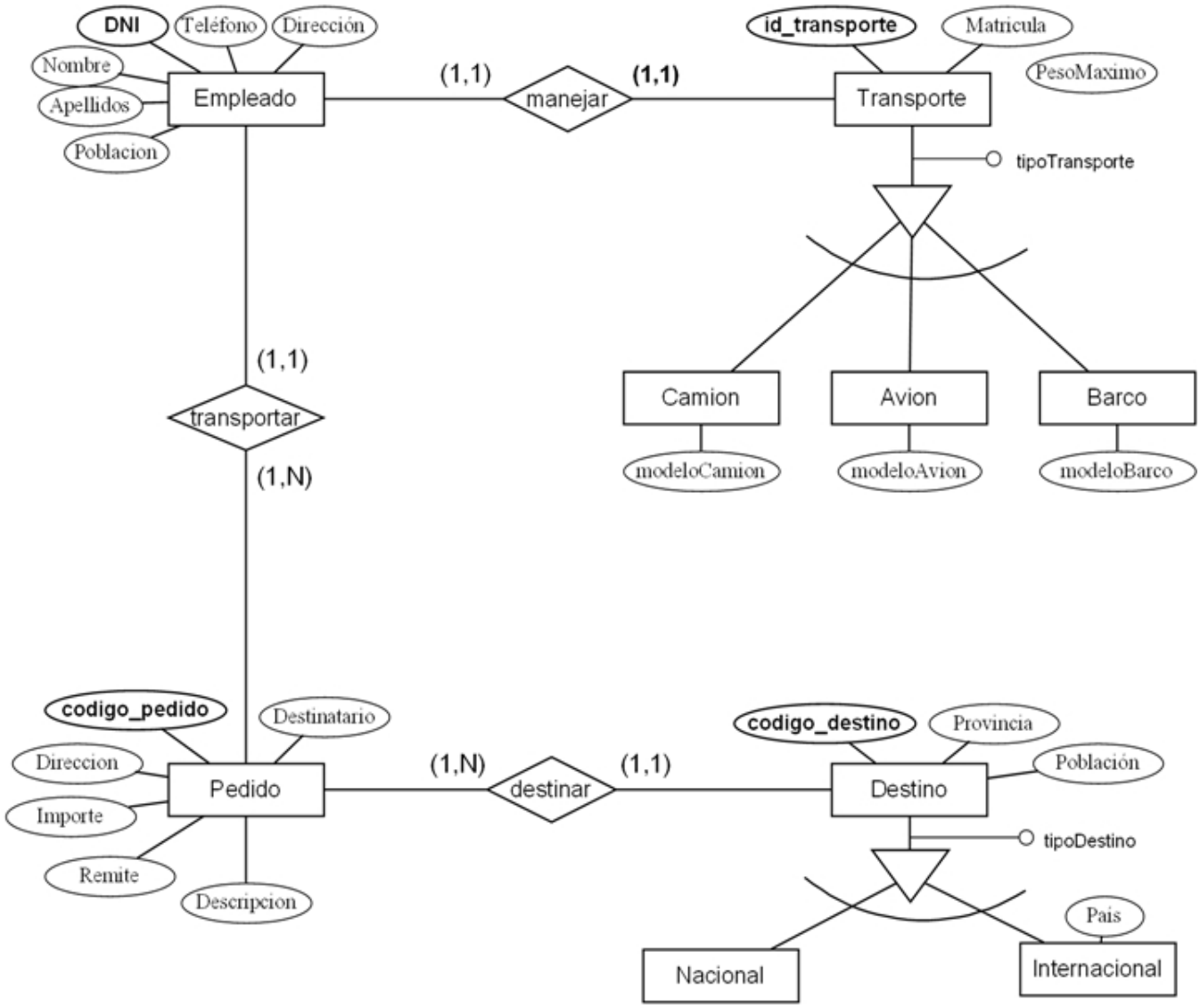
tpFecha = tpDia/tpMes/tpAño
tpDia = 1..31
tpMes = 1..12
tpAño = natural(4)

Implementación:

```
CREATE TABLE empleado(
        dni decimal(8),
        nombre varchar(30) NOT NULL,
        apellidos varchar(60) NOT NULL,
        direccion varchar(80) NOT NULL,
        telefono decimal(9) NOT NULL,
        fechaNacimiento date NOT NULl,
        CONSTRAINT pk_empleado PRIMARY KEY(dni)
);

CREATE TABLE camion(
        matricula varchar(7),
        tipo varchar(80) NOT NULL,
        modelo varchar(9) NOT NULL,
        CONSTRAINT pk_camion PRIMARY KEY(matricula)
);

CREATE TABLE pedido(
        codigo decimal(8),
        origen varchar(20) NOT NULL,
        descripcion varchar(50) NOT NULL,
        destinatario varchar(30) NOT NULL,
        direccion varchar(50) DEFAULT NULL,
        CONSTRAINT pk_pedido PRIMARY KEY(codigo)
);

CREATE TABLE provincia(
        codigoProvincia decimal(8),
        nombre varchar(50),
        CONSTRAINT pk_codigoProvincia PRIMARY KEY(codigoProvincia)
);

CREATE TABLE conduce(
        dni decimal(8),
        matricula varchar(7),
        CONSTRAINT pk_conducir PRIMARY KEY(dni, matricula),
        CONSTRAINT fk_empleado FOREIGN KEY (dni) REFERENCES empleado(dni),
        CONSTRAINT fk_camion FOREIGN KEY (matricula) REFERENCES camion(matricula)
);

CREATE TABLE transporta(
        dni decimal(8),
```

```
        codigo decimal(8),
        CONSTRAINT pk_transporta PRIMARY KEY(dni, codigo),
        CONSTRAINT fk_empleado FOREIGN KEY (dni) REFERENCES empleado(dni),
        CONSTRAINT fk_pedido FOREIGN KEY (codigo) REFERENCES pedido(codigo)
);

CREATE TABLE destinado(
        codigo decimal(8),
        codigoProvincia decimal(4),
        CONSTRAINT pk_destinado PRIMARY KEY(codigo, codigoProvincia),
        CONSTRAINT fk_pedido FOREIGN KEY (codigo) REFERENCES pedido(codigo),
        CONSTRAINT fk_provincia FOREIGN KEY (codigoProvincia) REFERENCES
provincia(codigoProvincia)

);
```

Esquema conceptual 2ª empresa:



Empleados especializados en transportes concretos, de ahí la correspondecia entre Empleado y Transporte 1:1

El número de transportes debe ser como mínimo igual al numero de transportistas.

Esquema relacional 2ª empresa:

Empleado
{
dni = tpDni, clave primaria;
nombre = tpTexto, NO NULO;

apellidos = tpTexto, NO NULO;

direccion= tpTexto, NO NULO;

telefono = tpTelefono, NO NULO;

poblacion = tpTexto, NO NULO;

}

Transporte
{
id_transporte= tpCodigo,clave primaria;
matricula = tpTexto, NO NULO;
pesoMaximo = tpPeso, NO NULO;
tipoTransporte = tpTipo, NO NULO;
modeloCamion = tpTexto, POR DEFECTO NULO;
modeloAvion = tpTexto, POR DEFECTO NULO;
modeloBarco = tpTexto POR DEFECTO NULO;
}

Pedido
{
código_pedido = tpCodigo, clave primaria;
remite = tpTexto, NO NULO;
descripcion = tpTexto, NO NULO;
importe = real,NO NULO;
destinatario = tpTexto, NO NULO;
direccion = tpTexto, NO NULO;
}

Destino
{
código_destino = tpCodigo, clave primaria;
provincia = tpTexto, NO NULO;
población = tpTexto, NO NULO;
tipoDestino = tpTipo NO NULO;
país = tpTexto, POR DEFECTO NULO;
}

Transportar
{
dni = tpDni, clave ajena Empleado;
código_pedido = tpCodigo, clave ajena Pedido;

clave primaria (dni, codigo);
}

Manejar
{
dni = tpDni, clave ajena Empleado;
id_transporte = tpID, clave ajena Camion;

clave primaria (dni, id_transporte);
}

Destinar {
código_pedido = tpCodigo, clave ajena Pedido;
código_destino = tpCodigo, clave ajena Destino;

clave primaria(código_pedido, código_destino);
}

Dominios:

tpDni = natural(8)
tpTelefono = natural(9)
tpMatricula = cadena(7)
tpCodigo = natural(8)
tpCodigoProvincia = natural(5)
tpTexto = cadena(250)
tpID = cadena(40)

tpFecha = tpDia/tpMes/tpAño
tpDia = 1..31
tpMes = 1..12
tpAño = natural(4)
tpPeso = real;

Implementación 2ª empresa:

Codigo del script en PostgreSQL en el que se diseña la estructura de la base de datos de la segunda empresa dedicada al transporte de paquetería.

```
CREATE TABLE empleado(
        dni decimal(8),
        nombre varchar(30) NOT NULL,
        apellidos varchar(60) NOT NULL,
        direccion varchar(80) NOT NULL,
        telefono decimal(9) NOT NULL,
        poblacion varchar(40) NOT NULL,
        CONSTRAINT pk_empleado PRIMARY KEY(dni)
);

CREATE TABLE transporte(
        idTransporte varchar(7),
        matricula varchar(7) NOT NULL,
        pesoMaximo decimal(3) NOT NULL,
        tipoTransporte decimal(1) NOT NULL,
        modeloCamion varchar(20) DEFAULT NULL,
        modeloAvion varchar(20) DEFAULT NULL,
        modeloBarco varchar(20) DEFAULT NULL,
        CONSTRAINT pk_transporte PRIMARY KEY(idTransporte)
);

CREATE TABLE pedido(
        codigoPedido decimal(8),
        remite varchar(20) NOT NULL,
```

```
        descripcion varchar(50) NOT NULL,
        importe decimal NOT NULL,
        destinatario varchar(20) NOT NULL,
        direccion varchar(50) NOT NULL,
        CONSTRAINT pk_pedido PRIMARY KEY(codigoPedido)
);

CREATE TABLE destino(
        codigoDestino decimal(8),
        provincia varchar(80) NOT NULL,
        poblacion varchar(80) NOT NULL,
        tipoDestino decimal(1) NOT NULL,
        pais varchar(80) DEFAULT NULL,
        CONSTRAINT pk_codigoDestino PRIMARY KEY(codigoDestino)
);

CREATE TABLE manejar(
        dni decimal(8),
        idTransporte varchar(7),
        CONSTRAINT pk_manejar PRIMARY KEY(dni, idTransporte),
        CONSTRAINT fk_empleado FOREIGN KEY (dni) REFERENCES empleado(dni),
        CONSTRAINT fk_transporte FOREIGN KEY (idTransporte) REFERENCES
transporte(idTransporte)
);

CREATE TABLE transportar(
        dni decimal(8),
        codigoPedido decimal(8),
```

```
	CONSTRAINT pk_transportar PRIMARY KEY(dni, codigoPedido),
	CONSTRAINT fk_empleado FOREIGN KEY (dni) REFERENCES empleado(dni),
	CONSTRAINT fk_pedido FOREIGN KEY (codigoPedido) REFERENCES pedido(codigoPedido)
);

CREATE TABLE destinar(
	codigoPedido decimal(8),
	codigoDestino decimal(4),
	CONSTRAINT pk_destinado PRIMARY KEY(codigoPedido, codigoDestino),
	CONSTRAINT fk_pedido FOREIGN KEY (codigoPedido) REFERENCES pedido(codigoPedido),
	CONSTRAINT fk_provincia FOREIGN KEY (codigoDestino) REFERENCES
destino(codigoDestino)
);
```

Integración:

La integración de las dos bases de datos se ha realizado usando la extensión DBLINK de PostgreSQL para la consulta de tablas en bases de datos remotas y poder así proponer una visión global de las dos para que su visualización y manipulación resulte mas sencilla y atractiva.

1. **Como configurar dblink para poder usarlo en PostgreSQL y la máquina virtual Turnkey**

   Es necesario instalar el siguiente paquete para proveernos de la funcionalidad dblink proporcionada por el gestor. Esto lo hacemos así :

   ```
   $> sudo apt-get install postgresql-contrib
   ```

   Luego iniciamos sesión como usuario postgres desde la consola :

   ```
   # su postgres
   ```

   Le indicamos al postegreSQL que con una base de datos x vamos a usar el dblink

```
#psql basede_datos_x <
/usr/share/postgresql/9.1/extension/dblink--1.0.sql
```

Luego abrimos la consola de postgreSQL en una base de datos concreta:

```
# psql bd1
```

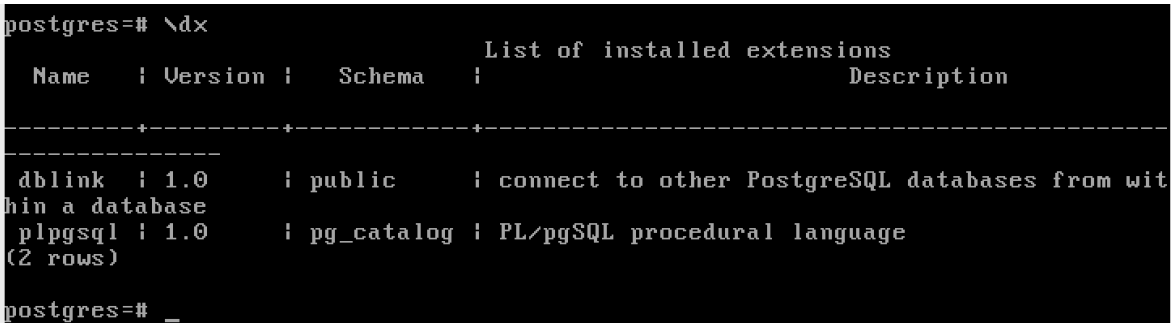
Creamos la extensión

```
postgres=# CREATE EXTENSION dblink;
```

Y si queremos comprobar que todo ha ido con éxito tecleamos :

```
postgres=# \dx
```

Y debe aparecer esto :



2. **Esquema global resultante para la base de datos federada.**

El esquema global resultante sería el compuesto por las siguientes vistas :

- Trabajadores : integra las dos entidades empleado de las dos bases de datos en una vista que muestra toda la información relacionada.
- FlotaCamiones : integra la flota de camiones transportistas de ambas compañías en una vista sola
- Aviones : una vista para ver exclusivamente los aviones y facilitar consultas.
- Barcos : una vista para ver exclusivamente los barcos y facilitar consultas.
- Pedidos : una vista que engloba los pedidos de ambas compañías.
- destinosInternacionales : una vista que congrega los destinos a puntos internacionales.
- destinosNacionales : una vista para los destinos nacionales a los que servían las dos compañías.

3. **Script en SQL**

El script con el código para PostgreSQL sería el siguiente :

```
CREATE VIEW trabajadores AS(

    SELECT * FROM empleado UNION SELECT * FROM dblink('dbname=transporte1', 'SELECT dni,nombre,apellidos,direccion,telefono,poblacion from empleado') AS (dni decimal(8), nombre varchar(50), apellidos varchar(100), direccion varchar(250),telefono numeric(9), poblacion varchar(40));

    UNION SELECT * FROM dblink('dbname=transporte1',

    'SELECT dni,nombre,apellidos,direccion,telefono,poblacion from empleado')

    AS tmp_trabajador(dni decimal(8), nombre varchar(50), apellidos varchar(100), direccion varchar(250),

    telefono numeric(9),fechaNacimiento date)

);

-- Camiones tipoTransporte = 3

CREATE VIEW flotaCamiones AS(

    SELECT matricula,modeloCamion FROM transporte WHERE tipoTransporte = 3

    UNION SELECT * FROM dblink('dbname=transporte1',

    'SELECT matricula,modelo from camion')

    AS  tmp_camion(matricula varchar(7) , modelo varchar(80))
```

```
);

-- Aviones tipoTransporte = 1

CREATE VIEW aviones AS (

        SELECT idTransporte,matricula,pesoMaximo,modeloAvion FROM transporte WHERE tipoTransporte = 1

);

-- Barcos tipoTransporte = 2

CREATE VIEW barcos AS (

        SELECT idTransporte,matricula,pesoMaximo,modeloBarco FROM transporte WHERE tipoTransporte = 2

);



CREATE VIEW pedidos AS(

        SELECT codigoPedido,remite,descripcion,destinatario,direccion FROM pedido

        UNION SELECT * FROM dblink('dbname=transporte1',

        'SELECT codigo,origen,descripcion,destinatario,direccion from pedido')

        AS tmp_pedido(codigoPedido decimal(8), remite varchar(20), descripcion varchar(50), destinatario varchar(20), direccion varchar(50))

);

--tipo desdino = 1 nacional

CREATE VIEW destinosNacionales AS(

        SELECT codigoDestino,provincia FROM destino WHERE tipoDestino = 1

        UNION SELECT * FROM dblink ('dbname=transporte1',

                'SELECT codigoProvincia,nombre FROM provincia ')

        AS tmp_destinoInternacional(codigosProvincias decimal(8), provincia varchar(50))
```

```
);


-- tipo destino = 1 internacional


CREATE VIEW destinosInternacionales AS(

        SELECT pais,provincia,poblacion from destino WHERE tipoDestino = 0

);
```