# Caso de estudio

Marc Gibert Ginesta

P06/M2109/02154

www.uoc.edu

# Índice

**Introducción** ..... 5

**Objetivos** ..... 6

**1. Presentación del caso de estudio** ..... 7

**2. El modelo relacional y el álgebra relacional** ..... 8
2.1. Determinar las relaciones ..... 8
2.2. Definición de claves ..... 9
2.3. Reglas de integridad ..... 11
2.4. Álgebra relacional ..... 12

**3. El lenguaje SQL** ..... 13
3.1. Sentencias de definición ..... 13
3.2. Sentencias de manipulación ..... 15

**4. Introducción al diseño de bases de datos** ..... 17
4.1. Diseño conceptual: el modelo ER ..... 17
4.2. Diseño lógico: la transformación del modelo ER al modelo relacional ..... 19

**5. Bases de datos en MySQL** ..... 22

**6. Bases de datos en PostgreSQL** ..... 24

**7. Desarrollo de aplicaciones en conexión con bases de datos** ..... 25

**Resumen** ..... 30

# Introducción

Este módulo forma parte del curso "Bases de datos" del itinerario "Administrador web y comercio electrónico" dentro del Máster Internacional de Software Libre de la Universitat Oberta de Catalunya.

El módulo está estructurado en apartados que corresponden al resto de los módulos de la asignatura, de modo que el estudiante puede ir siguiendo este caso de estudio a medida que va progresando en el curso.

Aunque algunos de los módulos ya disponen de ejercicios de autoevaluación, el caso de estudio presenta una visión completa de un proyecto de bases de datos y proporciona una visión práctica de cada uno de ellos.

# Objetivos

Los objetivos que deberíais alcanzar al finalizar el trabajo con la presente unidad son los siguientes:

- Comprender desde un punto de vista práctico los conceptos explicados en las unidades didácticas teóricas.

- Disponer de un modelo de referencia para emprender proyectos de bases de datos.

- Adquirir el criterio suficiente para identificar las actividades clave y tomar decisiones en un proyecto que implique el uso de bases de datos.

# 1. Presentación del caso de estudio

Acabamos de entrar a trabajar en una pequeña empresa –SuOrdenadorAMedida, S.L.– dedicada a la venta de ordenadores a particulares y otras empresas. Cuando nos hicieron la entrevista de trabajo, comentamos nuestra pasión por el software libre y en concreto hicimos hincapié en nuestro conocimiento de los motores de bases de datos libres y las ventajas que podían aportar respecto de los gestores propietarios. No sabemos si eso fue lo que convenció a la persona de recursos humanos o no, pero, en todo caso, y visto el resultado de la reunión que hemos tenido en nuestro primer día de trabajo, vamos a tener que aplicar nuestros conocimientos a fondo.

Nos han contado que, hasta ahora, la gestión de la empresa se llevaba a cabo con programas propietarios de gestión y contabilidad, pero que debido a problemas con la empresa que desarrollaba estos programas, se está considerando la migración de la gestión administrativa y de operaciones a entornos abiertos. Para acabar de decidirse, nos proponen que empecemos por renovar el sistema de gestión de peticiones e incidencias por parte de los clientes, de modo que esté basada en software libre.

Actualmente, las peticiones e incidencias se reciben telefónicamente, por correo electrónico o en persona en alguno de los locales que tiene la empresa. La persona que atiende al teléfono o lee los correos electrónicos de plantea una serie de preguntas al cliente y escribe en una plantilla de documento las respuestas. A continuación, se imprime el documento y se deja en una bandeja que recogen los técnicos cada mañana.

A medida que los técnicos van avanzando en la solución de la incidencia (o han llamado al cliente para pedir más datos), van apuntando las acciones y el estado del problema en la hoja que recogieron, hasta que la incidencia queda resuelta. En ese momento, la dejan en una bandeja que recoge cada mañana el personal de administración, que se pone en contacto con el cliente y factura el importe correspondiente a las horas de trabajo y componentes sustituidos.

Es obvio que este sistema presenta numerosas deficiencias, y que el rendimiento tanto de los técnicos, como del personal administrativo y de atención al cliente podría aumentar enormemente si muchos de estos procesos fueran automáticos, centralizados y, a poder ser, conectados con el resto del sistema de información de la empresa.

Éste es, a grandes rasgos, el problema que se nos plantea, y que utilizaremos como caso de estudio para aplicar los conocimientos adquiridos durante el desarrollo del curso.

# 2. El modelo relacional y el álgebra relacional

Visto el proyecto planteado, decidimos hacer las cosas bien hechas para, de paso, impresionar a nuestro jefe con nuestros conocimientos en bases de datos. El primer paso será presentarle un documento que describa el modelo relacional que vamos a utilizar, en el que incluiremos algunas consultas de muestra para que pueda comprobar qué será capaz de hacer con nuestro proyecto cuando esté acabado.

## 2.1. Determinar las relaciones

En primer lugar determinaremos las relaciones, sus atributos y los dominios de cada uno de ellos:

```
PETICION(referencia, cliente, resumen, estado, fecharecepcion, fechainicio, fechafin,
tiempoempleado)
NOTA_PETICION(peticion, nota, fecha, empleado)
MATERIAL_PETICION(nombrematerial, peticion, cantidad, precio)
CLIENTE(nombre, nif, telefono, email)
EMPLEADO(nombre, nif)
```

En la relación `PETICION`, hemos decidido que convendría tener una referencia interna de la petición, que nos ayudará al hablar de ella con el cliente (si tuviese varias abiertas) y evitará confusiones al trabajar. El resto de atributos son bastante explícitos.

Como una petición puede evolucionar con el tiempo, a medida que se piden más datos al cliente, la incidencia va evolucionando, etc., hemos creado las relaciones `NOTA_PETICION` y `MATERIAL_PETICION` para reflejarlo.

También hemos tenido que definir las relaciones `CLIENTE` y `EMPLEADO` para poder relacionarlas con las peticiones y las notas que se vayan generando durante su resolución.

A continuación vamos a definir los dominios de los atributos:

```
PETICION:
dominio(referencia)=números
dominio(cliente)=NIF
```

```
dominio(resumen)=texto
dominio(estado)=estados
dominio(fecharecepcion)=fechayhora
dominio(fechainicio)=fechayhora
dominio(fechafin)=fechayhora
dominio(tiempoempleado)=horasyminutos

NOTA_PETICION:
dominio(peticion)=números
dominio(nota)=texto
dominio(fecha)=fechayhora
dominio(empleado)=NIF

MATERIAL_PETICION:
dominio(nombrematerial)=nombreMaterial
dominio(peticion)=números
dominio(precio)=precio
dominio(cantidad)=números

CLIENTE:
dominio(nombre)=nombreCliente
dominio(nif)=NIF
dominio(telefono)=teléfonos
dominio(email)=emails

EMPLEADO:
dominio(nombre)=nombreEmpleado
dominio(nif)=NIF
```

Al definir los dominios de cada atributo, ya nos hemos avanzado en la toma de algunas decisiones: al decidir, por ejemplo, que el dominio del atributo `empleado` en la relación `NOTA_PETICION` es `NIF`, estamos implícitamente determinando que la clave primaria de la relación `EMPLEADO` será del dominio NIF y que usaremos un atributo de este dominio para referirnos a él.

Este proceso descrito indicando directamente su resultado, normalmente es fruto de una revisión de las entidades a medida que se van definiendo y analizando las necesidades de éstas.

**Nota**

La regla de integridad del modelo correspondiente a la clave primaria obligará a que no existan dos notas sobre la misma petición hechas en la misma fecha y hora por parte del mismo empleado, lo cual es perfectamente lícito y coherente.

### 2.2. Definición de claves

Aunque algunas claves ya se intuyen a partir de los atributos de las relaciones, vamos a determinarlas para completar el caso.

```
PETICION:
Claves candidatas: {referencia}
Clave primaria: {referencia}

NOTA_PETICION:
Claves candidatas: {peticion,fecha,empleado}
Clave primaria: {peticion, fecha,empleado}

MATERIAL_PETICION:
Claves candidatas: {nombrematerial,peticion}
Clave primaria: {nombrematerial,peticion}

CLIENTE:
Claves candidatas: {nif}
Clave primaria: {nif}

EMPLEADO:
Claves candidatas: {nif}
Clave primaria: {nif}
```

**En todos los casos sólo tenemos una clave candidata y, por lo tanto, no caben dudas a la hora de escoger la clave primaria. Esto no tiene por que ser así: en la relación EMPLEADO, podríamos haber incluido más atributos (número de la seguridad social, un número de empleado interno, etc.) que serían claves candidatas susceptibles de ser clave primaria.**

Ahora podemos reescribir las relaciones:

```
PETICION(referencia, cliente, resumen, estado, fecharecepcion, fechainicio, fechafin,
tiempoempleado)
NOTA_PETICION(peticion, nota, fecha, empleado)
MATERIAL_PETICION(nombrematerial, peticion, cantidad, precio)
CLIENTE(nombre, nif, telefono, email)
EMPLEADO(nombre, nif)
```

Las claves foráneas ya se intuyen a partir de las relaciones, aunque vamos a comentarlas para completar el caso:

**NOTA_PETICION:**

Tiene de clave foránea el atributo {peticion}, que establece la relación (y pertenece al mismo dominio) con el atributo {referencia} de la relación PETICION.

También tiene la clave foránea {empleado}, que establece la relación con EMPLEADO a partir de su clave primaria {nif}.

**MATERIAL_PETICION:**

Tiene de clave foránea el atributo {peticion}, que establece la relación (y pertenece al mismo dominio) con el atributo {referencia} de la relación PETICION.

### 2.3. Reglas de integridad

En este punto, no es necesario preocuparse por las reglas de integridad del modelo que tratan sobre la clave primaria, ya que nos vendrán impuestas en el momento de crear las tablas en el SGBD.

Es conveniente, no obstante, fijar las decisiones sobre la integridad referencial; en concreto, qué vamos a hacer en caso de restricción. Así pues, para cada relación que tiene una clave primaria referenciada desde otra, deberemos decidir qué política cabe aplicar en caso de modificación o borrado:

**PETICION**

- Modificación del atributo {referencia} referenciado desde NOTA_PETICION y MATERIAL_PETICION: aquí podemos optar por la restricción o por la actualización en cascada (más cómoda, aunque no todos los SGBD la implementan, como veremos más adelante).

- Borrado del atributo {referencia}. Aquí optaremos por una política de restricción. Si la petición tiene notas asociadas o materiales, significa que ha habido alguna actividad y, por lo tanto, no deberíamos poder borrarla. Si se desea anularla, ya estableceremos un estado de la misma que lo indique.

**CLIENTE**

- Modificación del atributo {nif} referenciado desde PETICION. Es probable que si un cliente cambia de NIF (por un cambio del tipo de sociedad, etc.) deseemos mantener sus peticiones. Aquí la política debe ser de actualización en cascada.

- Borrado del atributo {nif}. Es posible que si queremos borrar un cliente, es porque hemos terminado toda relación con él y, por lo tanto, es coherente utilizar aquí la política de anulación.

**EMPLEADO**

- Modificación del atributo {nif} referenciado desde NOTA_PETICION. No es probable que un empleado cambie su NIF, salvo caso de error. Aun así, en caso de que se produzca, es preferible la actualización en cascada.

- Borrado del atributo {nif}. Aunque eliminemos un empleado si termina su relación con la empresa, no deberíamos eliminar sus notas. La mejor opción es la restricción.

## 2.4. Álgebra relacional

Para probar el modelo, una buena opción es intentar realizar algunas consultas sobre él y ver si obtenemos los resultados deseados. En nuestro caso, vamos a realizar las siguientes consultas:

- Obtención de una petición junto con los datos del cliente:

```
R:= PETICION [cliente=nif] CLIENTE
```

- Obtención de una petición con todas sus notas:

```
NP(peticionnota, nota, fechanota, empleado):=NOTA_PETICION(peticion, nota, fecha, empleado)
R:=PETICION[peticion=peticionnota]NOTA_PETICION
```

- Obtención de los datos de todos los empleados que han participado en la petición 5:

```
NP:=NOTA_PETICION[peticion=5]
RA:=EMPLEADO[nif=empleado]NP
R:=RA[nombre,nif]
```

**Ejercicio**

Os sugerimos que intentéis más operaciones sobre el modelo para familiarizaros con él.

# 3. El lenguaje SQL

Una vez terminado el modelo relacional, decidimos completar la documentación que veníamos realizando con las sentencias SQL correspondientes. Así, veremos en qué se concretará el modelo relacional.

Como aún no sabemos en qué sistema gestor de base de datos vamos a implantar la solución, decidimos simplemente anotar las sentencias según el estándar SQL92, y, posteriormente, ya examinaremos las particularidades del sistema gestor escogido para adaptarlas.

## 3.1. Sentencias de definición

- Creación de la base de datos

```
CREATE SCHEMA GESTION_PETICIONES;
```

**Atención**

En algunos casos es conveniente la definición de dominios para facilitar el trabajo posterior de mantenimiento de la coherencia de la base de datos. No es aconsejable definir dominios para cada dominio relacional, pero sí en los casos en que una columna puede tomar una serie de valores determinados.

- Definición de dominios

```
CREATE DOMAIN dom_estados AS CHAR (20)
   CONSTRAINT estados_validos
   CHECK (VALUE IN ('Nueva','Se necesitan más datos','Aceptada','Confirmada',
'Resuelta',
'Cerrada'))
   DEFAULT 'Nueva';
```

- Creación de las tablas

```
CREATE TABLE PETICION ( referencia INTEGER NOT NULL,
   cliente INTEGER NOT NULL,
   resumen CHARACTER VARYING (2048),
   estado dom_estados NOT NULL,
   fecharecepcion TIMESTAMP NOT NULL,
   fechainicio TIMESTAMP, fechafin TIMESTAMP,
   tiempoempleado TIME, PRIMARY KEY (referencia),
   FOREIGN KEY cliente REFERENCES CLIENTE(nif)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   CHECK (fecharecepcion < fechainicio),
   CHECK (fechainicio < fechafin) );
```

**Atención**

Aquí deberemos tener en cuenta las reglas de integridad, ya que habrá que explicitar la política escogida como restricción.

```
CREATE TABLE NOTA_PETICION ( peticion INTEGER NOT NULL,
   nota CHARACTER VARYING (64000),
   fecha TIMESTAMP NOT NULL,
   empleado CHARACTER (9),
   FOREIGN KEY (peticion) REFERENCES PETICION(referencia)
      ON DELETE NO ACTION
      ON UPDATE CASCADE,
   FOREIGN KEY (empleado) REFERENCES EMPLEADO(nif)
      ON DELETE NO ACTION
      ON UPDATE CASCADE );

CREATE TABLE MATERIAL_PETICION ( nombrematerial CHARACTER
VARYING (100) NOT NULL,
   peticion INTEGER NOT NULL,
   precio DECIMAL(8,2),
   cantidad INTEGER,
   FOREIGN KEY (peticion) REFERENCES PETICION(referencia)
      ON DELETE NO ACTION
      ON UPDATE CASCADE );

CREATE TABLE CLIENTE ( nombre CHARACTER VARYING (100) NOT
NULL,
   nif CHARACTER (9) NOT NULL,
   telefono CHARACTER (15),
   email CHARACTER (50),
   PRIMARY KEY (nif) );

CREATE TABLE EMPLEADO (nombre CHARACTER VARYING (100) NOT
NULL,
   nif CHARACTER (9) NOT NULL,
   PRIMARY KEY (nif) );
```

- Creación de vistas

**Función de vistas**

Las vistas agilizarán las consultas que prevemos que van a ser mas frecuentes.

– Peticiones pendientes:

```
CREATE VIEW peticiones_pendientes (referencia, nombre_cliente, resumen, estado, duracion,
fecharecepcion) AS (
SELECT P.referencia, C.nombre, P.resumen, P.estado, (P.fechainicio     P.fecharecepcion),
P.fecharecepcion
FROM PETICION P JOIN CLIENTE C ON P.cliente = C.nif
WHERE estado NOT IN ('Resuelta','Cerrada') ORDER BY fecharecepcion )
```

– Tiempo y precio de los materiales empleados para las peticiones terminadas en el mes en curso:

```
CREATE VIEW peticiones_terminadas (referencia, nombre_cliente, resumen,
tiempo_empleado, importe_materiales) AS (
     SELECT P.referencia, C.nombre, P.resumen, P.tiempoempleado, SUM(M.precio)
     FROM PETICION P, CLIENTE C, MATERIAL_PETICION M
     WHERE P.cliente=C.nif AND M.peticion=P.referencia AND estado='Resuelta'
     GROUP BY P.referencia)
```

### 3.2. Sentencias de manipulación

A continuación, decidimos indicar algunas sentencias de manipulación corrientes para completar la documentación. De esta manera, cuando empecemos el desarrollo, tendremos mucho más claras estas operaciones sobre la base de datos:

- Nuevo cliente:

```
INSERT INTO CLIENTE VALUES ('Juan Pérez','42389338A','912223354','juanperez@gmail.com');
```

- Nueva petición:

```
INSERT INTO PETICION VALUES (5,'42389338A', 'No le arranca el ordenador',
'Nueva',CURRENT_TIMESTAMP,NULL,NULL,NULL);
```

- Cambio de estado de la petición, añadimos una nota y un material:

```
UPDATE PETICION SET estado='Aceptada' WHERE referencia=5;
INSERT INTO NOTA_PETICION VALUES (5,'Parece un problema del disco duro. Vamos examinarlo
más a fondo.',CURRENT_TIMESTAMP,'35485411G');

INSERT INTO MATERIAL_PETICION VALUES ('Disco duro 20Gb',5,250.00,1);
```

- Materiales solicitados en la petición 5:

```
SELECT nombrematerial, cantidad, precio FROM MATERIAL_PETICION WHERE peticion=5
```

- Número de peticiones abiertas del cliente '42389338A':

```
SELECT COUNT(*) FROM PETICION WHERE cliente='42389338A' AND estado NOT IN ('Resuelta',
'Cerrada');
```

La creación de vistas del apartado anterior nos ha mostrado también algunas consultas complejas que repetimos a continuación:

- Peticiones abiertas:

```
SELECT P.referencia, C.nombre, P.resumen, P.estado, (P.fechainicio   P.fecharecepcion),
P.fecharecepcion FROM PETICION P JOIN CLIENTE C ON P.cliente = C.nif WHERE estado NOT IN
('Resuelta','Cerrada') ORDER BY fecharecepcion;
```

- Tiempo y precio de los materiales empleados para las peticiones terminadas en el mes en curso:

```
SELECT P.referencia, C.nombre, P.resumen, P.tiempoempleado, SUM(M.precio) FROM PETICION
P, CLIENTE C, MATERIAL_PETICION M WHERE P.cliente=C.nif AND M.peticion=P.referencia AND
estado='Resuelta' GROUP BY P.referencia;
```

Finalmente, vamos a practicar con las consultas que realizamos en álgebra relacional en el apartado anterior:

- Obtención de una petición junto con los datos del cliente:

```
R:= PETICION [cliente=nif] CLIENTE
SELECT * FROM PETICION JOIN CLIENTE ON PETICION.cliente=CLIENTE.nif;
```

- Obtención de una petición con todas sus notas:

```
NP(peticionnota,nota,fechanota,empleado):=NOTA_PETICION (peticion,nota,fecha,empleado)
R:=PETICION[referencia=peticionnota]NP
SELECT PETICION.*, peticion AS peticionnota, nota, fecha as fechanota, empleado FROM
PETICION JOIN NOTA_PETICION ON referencia=peticionnota;
```

- Obtención de los datos de todos los empleados que han participado en la petición 5:

```
NP:=NOTA_PETICION[peticion=5]
RA:=EMPLEADO[nif=empleado]NP
R:=RA[nombre,nif]
SELECT E.nombre, E.nif FROM EMPLEADO E, NOTA_PETICION N WHERE E.nif=NOTA_PETICION.empleado
AND NOTA_PETICION.peticion=5;
```

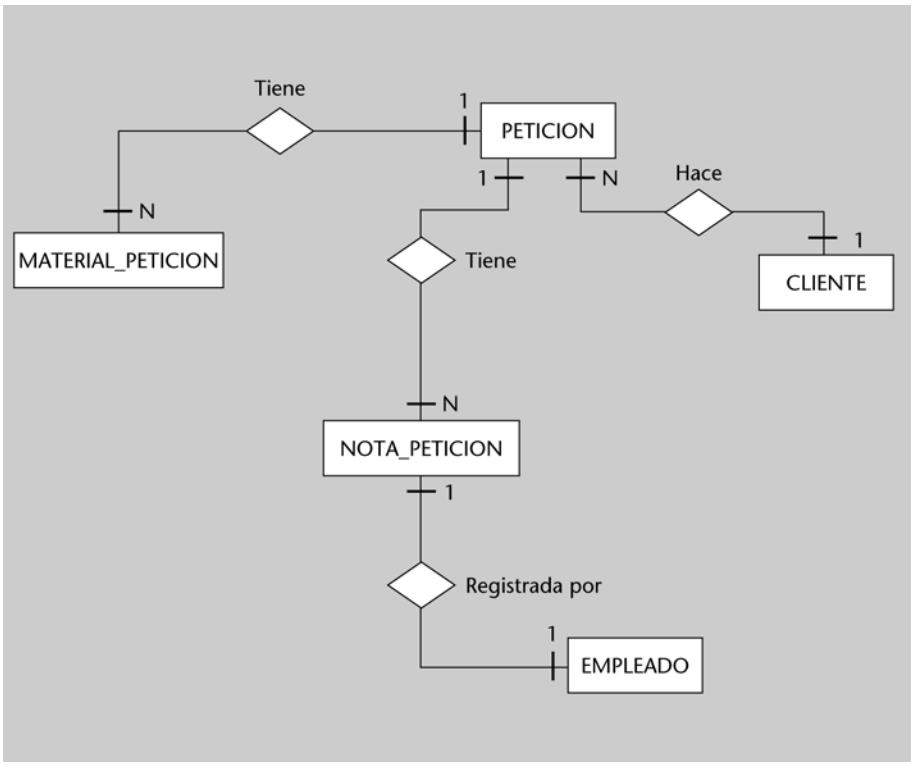
# 4. Introducción al diseño de bases de datos

Aunque las sentencias SQL de creación de tablas son bastante claras para un usuario técnico, de cara a la reunión previa a la toma de decisión sobre el SGBD concreto en el que vamos a implantar la solución, necesitaremos algo más.

Teniendo en cuenta que entre los asistentes a la reunión no hay más técnicos especializados en bases de datos que nosotros, hemos pensado que disponer un modelo entidad-relación del sistema nos ayudará a comunicar mejor la estructura que estamos planteando y, de paso, a demostrar (o, si es necesario, corregir) que el modelo relacional que planteamos al inicio es el correcto.

## 4.1. Diseño conceptual: el modelo ER

Vamos a plantear en primer lugar el modelo obtenido y, después, comentaremos los aspectos más interesantes:



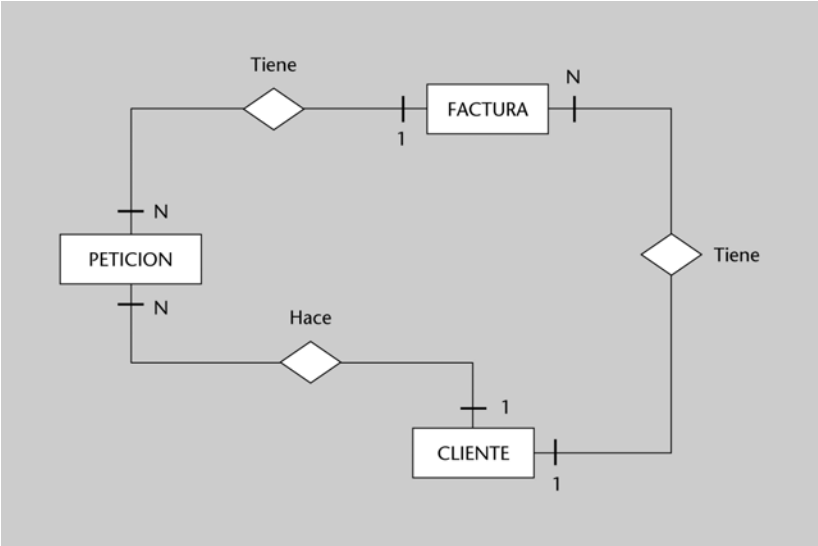

El modelo, expresado de este modo, es mucho más comprensible por parte de personal no técnico o no especializado en tecnologías de bases de datos.

A partir de la expresión gráfica del modelo, identificamos limitaciones o puntos de mejora, que anotamos a continuación:

- Probablemente, debemos incluir información de facturación a clientes por las peticiones realizadas.
- Probablemente, habrá peticiones que puedan agruparse en una entidad superior (un proyecto o trabajo), o bien peticiones relacionadas entre ellas (peticiones que deban resolverse antes que otras).

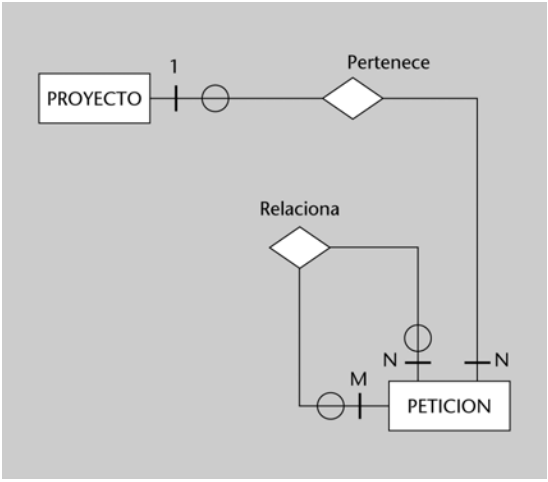
Una vez identificadas estas limitaciones, vamos a ampliar el modelo para corregirlas e impresionar a nuestros superiores de cara a la reunión.

- Información de facturación a clientes:




Lo único que hemos introducido ha sido la entidad `FACTURA`, que se relaciona con *N* peticiones y con un único cliente. Un `CLIENTE` puede tener varias facturas asociadas, pero una petición sólo puede pertenecer a una única factura.

- Grupos de peticiones y relaciones entre ellas.



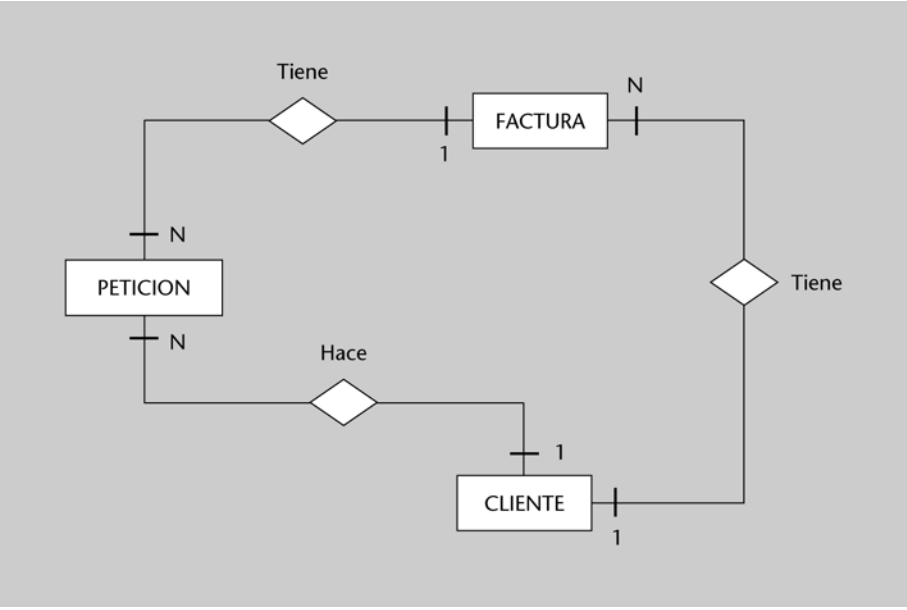
La pertenencia a un proyecto será opcional, y así lo indicamos en el diagrama. Por lo que respecta a las relaciones de peticiones entre ellas, se trata de una interrelación recursiva. Si queremos contemplar casos como los siguientes, debemos expresar la relación como M:N recursiva y opcional. En la interrelación `RELACIONA`, debemos contemplar algún atributo que indique de qué tipo de relación se trata en cada caso:

- Una petición depende de una o más peticiones.
- Una petición bloquea a una o más peticiones.
- Una petición es la duplicada de una o más peticiones.
- Una petición está relacionada con una o más peticiones.

### 4.2. Diseño lógico: la transformación del modelo ER al modelo relacional

En el apartado anterior sugerimos unas ampliaciones sobre el modelo ER que proporcionaban más prestaciones al proyecto. A continuación, vamos a realizar la transformación al modelo relacional de estas ampliaciones:

- Información de facturación a clientes.



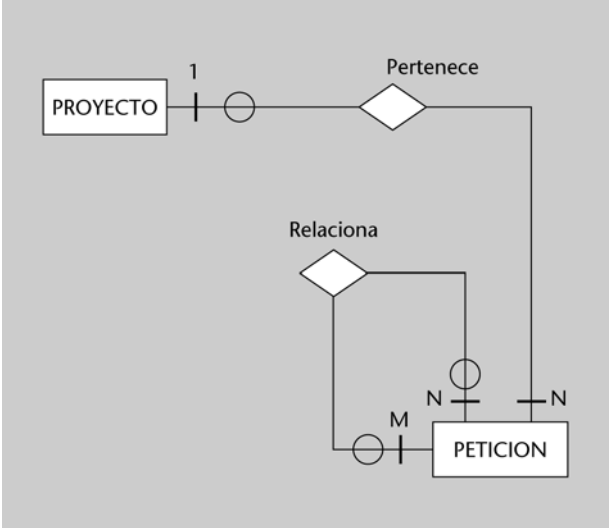

Según las transformaciones vistas en el módulo "El lenguaje SQL", la entidad `FACTURA` se transforma en la relación `FACTURA`, con los siguientes atributos:

```
FACTURA(numfactura, fecha, cliente)
```

Donde `cliente` es una clave foránea que corresponde a la interrelación `TIENE` entre `CLIENTE` y `FACTURA`. Un cliente puede tener *N* facturas, pero una factura pertenece sólo a un único cliente.

La interrelación entre FACTURA y PETICION del tipo 1:N se transforma también en una nueva clave foránea, que aparece siempre en el lado *N* de la interrelación; o sea, en la relación PETICION. Si existen peticiones que no deban facturarse (porque se han cerrado sin resolverse, o eran duplicadas de otras, etc.), su clave foránea tomaría el valor NULO.

- Grupos de peticiones y relaciones entre ellas.




Por una parte, la entidad proyecto debe transformarse en la relación PROYECTO, con atributos como los siguientes:

```
PROYECTO(codigo, nombre, fechainicio, fechafin)
```

La relación 1:N entre PROYECTO y PETICIÓN se transformará en la inserción de una nueva clave foránea en la relación PETICION, que podrá tener valor NULO si la petición no pertenece a ningún proyecto; es decir, si se trata de una petición aislada. La relación PETICION quedaría así:

```
PETICION(referencia, cliente, resumen, estado, fecharecepcion, fechainicio, fechafin,
tiempoempleado, factura, proyecto)
```

Por lo que respecta a las relaciones entre peticiones, se trata de una interrelación recursiva N:M, y por lo tanto se transformará en una nueva relación, PETICION_RELACION:

```
PETICION_RELACION(referencia_peticion1, referencia_peticion2, tiporelacion)
```

En este caso, y según el valor que pueda tomar el atributo {tiporelacion}, tendrá importancia o no qué referencia de petición aparece en cada atributo de la relación.

**Ejemplo**

Si la relación debe indicar simplemente que dos peticiones están relacionadas, entonces no importa qué referencia sea, la 1 o la 2.

En cambio, si el atributo {tiporelacion} indica un bloqueo o una dependencia entre relaciones (porque una debe resolverse antes que otra, por ejemplo), entonces sí tiene sentido qué referencia de petición se almacena en el atributo 1 y cuál en el 2. En todo caso, esta tarea corresponderá a la solución que se adapte y trabaje con la base de datos en último término, no al propio modelo.

# 5. Bases de datos en MySQL

Una vez hemos terminado el proceso de diseño de nuestra solución, en cuanto a su sistema de información, es hora de implantarlo sobre un sistema gestor de bases de datos.

Ya que disponemos de dos alternativas (MySQL y PostgreSQL), y no nos corresponde tomar la decisión final (sólo hacer la recomendación), vamos a elaborar una lista con los aspectos clave en la toma de decisiones y a puntuar, o comentar, cada SGBD según los ítems siguientes:

- Modelo de licencia, precio.
- Soporte por parte del fabricante.
- Conexión desde PHP.
- Prestaciones en creación de las estructuras (tablas, índices, etc.).
- Prestaciones en tipos de datos.
- Prestaciones en consultas simples.
- Prestaciones en consultas complejas.
- Prestaciones en manipulación de datos.
- Facilidad en la administración de usuarios.
- Facilidad en la gestión de copias de seguridad.

**Nota**

Mostraremos esta lista en forma de tabla, y al final elaboraremos unas conclusiones. Aunque no realicemos una ponderación de cada aspecto de la lista anterior, la simple comparación nos servirá para llegar a una conclusión rápida.

| Concepto | Valoración | Comentarios |
|---|---|---|
| Modelo de licencia, precio | 2 | Aunque no nos planteamos vender nuestra solución, ni comercializarla con una licencia propietaria, la licencia dual de MySQL siempre será un aspecto que tendremos que tener en cuenta si alguien se interesa por nuestra aplicación. |
| Soporte por parte del fabricante | 3 | Tenemos tanto la opción de contratar soporte en varias modalidades, como la de optar por consultar a la amplísima gama de usuarios del producto. En todo caso, en ambas situaciones obtendremos un excelente soporte. |
| Conexión desde PHP | 3 | PHP siempre ha incluido soporte para este SGBD bien con funciones especiales dedicadas que aprovechan al máximo sus características, o bien con librerías PEAR como DB que nos abstraen del SGBD y que soportan MySQL a la perfección. Aunque recientemente ha habido algún problema con la licencia y parecía que PHP no incluiría soporte para MySQL en sus últimas versiones, MySQL ha hecho una excepción con PHP (que sin duda ha contribuido mucho a la popularización de MySQL) por el bien de la comunidad y de sus usuarios. |
| Prestaciones en creación de las estructuras (tablas, índices, etc.) | 2 | MySQL es francamente fácil de manejar en este aspecto, y aunque no ofrece todas las prestaciones contempladas en el estándar, es "satisfactorio" para la aplicación que estamos planeando. |
| Prestaciones en tipos de datos | 2 | Los tipos de datos soportados por MySQL así como los operadores incluidos en el SGBD son más que suficientes para nuestra aplicación. |
| Prestaciones en consultas simples | 3 | Ésa es precisamente la característica que hace que MySQL sea uno de los SGBD mejor posicionados. |

**Clave de valoración**

- 1: no satisfactorio
- 2: satisfactorio
- 3: muy satisfactorio

| Concepto | Valoración | Comentarios |
|---|---|---|
| Prestaciones en consultas complejas | 2 | Hasta hace poco, MySQL no soportaba subconsultas y esto implicaba un mayor esfuerzo por parte de los programadores. Ahora ya las soporta y a un nivel igual al de sus competidores. |
| Prestaciones en manipulación de datos | 3 | MySQL incluye multitud de opciones no estándares para cargar datos externos, insertar o actualizar sobre la base de consultas complejas y la utilización de operadores como condiciones para la manipulación. |
| Facilidad en la administración de usuarios | 3 | Soporta muy bien el estándar en cuanto a la creación de usuarios y la gestión de sus privilegios con GRANT y REVOKE. Además, todos estos datos están accesibles en tablas de sistema, lo que hace muy sencilla la verificación de permisos. |
| Facilidad en la gestión de copias de seguridad | 3 | Disponemos tanto de herramientas de volcado, como la posibilidad de copia binaria de la base de datos. Además, dada su popularidad varios fabricantes de soluciones de copias de seguridad proporcionan conectores para realizar *backups* de la base de datos en caliente. |
| Conclusión | 2,6 | Estamos ante un "más que satisfactorio" SGBD para la solución que nos planteamos. No hay ninguna carencia insalvable. |

# 6. Bases de datos en PostgreSQL

| Concepto | Valoración | Comentarios |
|---|---|---|
| Modelo de licencia, precio | 3 | La licencia BSD no nos limita en ningún aspecto. Simplemente tendremos que incluir la nota sobre la misma en nuestro software, tanto si lo queremos comercializar como si no. |
| Soporte por parte del fabricante | 2 | PostgreSQL no ofrece soporte directamente, aunque sí que proporciona los mecanismos para que la comunidad lo ofrezca (listas de correo, IRC, enlaces, etc.). También tiene una lista (corta) de empresas que ofrecen soporte profesional de PostgreSQL. En España sólo hay una, y no es una empresa de desarrollo de software. |
| Conexión desde PHP | 3 | PHP siempre ha incluido soporte para este SGBD, bien con funciones especiales dedicadas que aprovechan al máximo sus características o bien con librerías PEAR como DB, que nos abstraen del SGBD y que soportan PostgreSQL a la perfección. |
| Prestaciones en creación de las estructuras (tablas, índices, etc.) | 3 | PostgreSQL es muy potente en este aspecto, ofrece prácticamente todas las prestaciones contempladas en el estándar, y tiene un fantástico sistema de extensión. |
| Prestaciones en tipos de datos | 3 | Los tipos de datos soportados por PostgreSQL, así como los operadores incluidos en el SGBD son más que suficientes para nuestra aplicación. Además, su sistema de extensiones y definición de tipos y dominios incluye casi todo lo que podamos necesitar. |
| Prestaciones en consultas simples | 3 | Por supuesto, PostgreSQL no decepciona en este punto. |
| Prestaciones en consultas complejas | 3 | PostgreSQL ha soportado subconsultas, vistas y todo lo que podamos necesitar en nuestra aplicación desde hace varios años. Su implementación de éstas es ya muy estable. |
| Prestaciones en manipulación de datos | 3 | PostgreSQL incluye multitud de opciones no estándares para cargar datos externos, insertar o actualizar sobre la base de consultas complejas y la utilización de operadores como condiciones para la manipulación. |
| Facilidad en la administración de usuarios | 2 | Soporta bastante bien el estándar en cuanto a la creación de usuarios y la gestión de sus privilegios con GRANT y REVOKE. Su sistema múltiple de autenticación lo hace demasiado complejo en este aspecto. |
| Facilidad en la gestión de copias de seguridad | 3 | Disponemos tanto de herramientas de volcado, como de la posibilidad de copia binaria de la base de datos. |
| Conclusión | 2,8 | Estamos ante el SGBD casi ideal. Sólo le falta facilitar la gestión de usuarios y mejorar su soporte. |

**Clave de valoración**

- 1: no satisfactorio
- 2: satisfactorio
- 3: muy satisfactorio

# 7. Desarrollo de aplicaciones en conexión con bases de datos

En la reunión mantenida con la dirección se examinaron muy cuidadosamente los análisis de los SGBD seleccionados. Al ser la diferencia de valoración tan leve, no fue fácil tomar una decisión, pero al final se decidió la implementación de la solución sobre el SGBD PostgreSQL.

Se decidió, también, hacer la implementación en PHP, abstrayéndonos del SGBD con el que trabajáramos. Así, en caso de que la mayor dificultad en la administración de PostgreSQL nos hiciera rectificar la decisión en el futuro, el tiempo de puesta en marcha del cambio sería mínimo.

Antes de iniciar la implantación, vamos a realizar unas pruebas conceptuales de la propia implementación que, después, pasaremos a un equipo de desarrollo interno para que haga el resto. En concreto, tomaremos algunas de las consultas vistas en el capítulo 3 y programaremos los *scripts* PHP de las páginas correspondientes, documentándolas al máximo para facilitar el trabajo al equipo de desarrollo.

En primer lugar, crearemos un fichero .php con la conexión a la base de datos, para incluirlo en todos los PHP que lo vayan a necesitar, y evitar, así, tener que repetir código cada vez. Esta acción también ayudará a mantener centralizados los datos de la conexión y, en caso de que debiéramos cambiar el usuario o la contraseña o cualquier otro dato de la conexión, sólo tendríamos que actualizar dicho fichero.

**datosconexion.php**

```
<?php
// Incluimos la librería una vez instalada mediante PEAR
require_once 'DB.php';

// Creamos la conexión a la base de datos, en este caso PostgreSQL
$db =& DB::connect('pgsql://usuario:password@servidor/basededatos');

// Comprobamos error en la conexión
if (DB::isError($db)) {
   die($db->getMessage());
}
?>
```

**a. Nuevo cliente. Página de resultado de la inserción de un nuevo cliente**

```
<?php
// Incluimos el fichero con los datos de la conexión.
include_one 'datosconexion.php';

// Utilizamos el método quoteSmart() para evitar que determinados caracteres
// (intencionados o no) puedan romper la sintaxis de la sentencia SQL.
// El método insertará automáticamente comillas alrededor de las cadenas
// de texto, o tratará los valores NULL correctamente según el SGBD, etc.
$db->query("INSERT INTO CLIENTE VALUES ("
   . $db->quoteSmart($_REQUEST['nombre']) . ","
   . $db->quoteSmart($_REQUEST['dni']). ","
   . $db->quoteSmart($_REQUEST['telefono']) . ","
   . $db->quoteSmart($_REQUEST['email']) . ")");

if (DB::isError($db)) {
   echo "<h2>Error al insertar el cliente</h2>";
   die($db->getMessage());
}

$db->disconnect();
```

**c. Cambio de estado de la petición:**

```
<?php
include_once 'datosconexion.php';

// Vamos a trabajar con todas las operaciones de esta página en forma de
// transacción, ya que, si se produce un error al insertar una nota o un
// material, la petición puede quedar en un estado erróneo.
$db->autoCommit(false);

// Le decimos a PHP que almacene en un buffer la salida, para poder así
// rectificar en caso de producirse un error.
ob_start();

// Suponemos que los estados nos llegan directamente con los valores
// soportados por el dominio, por ejemplo, a partir de los valores
// fijos de un desplegable.

// Suponemos que la fecha de inicio y fecha de fin nos llegan en formato
// español dd/mm/yyyy y los convertimos a YYYY-mm-dd según ISO 8601.

// Suponemos que el tiempo empleado nos llega en dos campos, horas y minutos, de
// forma que concatenándolos e insertando un ":" en medio, obtenemos una
// hora en formato ISO 8601.
```

```
if (isset($_REQUEST['fechainicio']) && !empty($_REQUEST['fechainicio'])) {
   $fechainicio_array=split("/",$_REQUEST['fechainicio']);
   $fechainciioDB=date("Y-M-d", mktime(0, 0, 0, $fechainicio[1], $fechainicio[0],
$fechainicio[2]));
} else {
   $fechainicioDB=NULL;
}

if (isset($_REQUEST['fechafin']) && !empty($_REQUEST['fechafin'])) {
   $fechafin_array=split("/",$_REQUEST['fechafin']);
   $fechafinDB=date("Y-M-d", mktime(0, 0, 0, $fechafin[1], $fechafin[0], $fechafin[2]));
} else {
   $fechafinDB=NULL;
}

$db->query("UPDATE PETICION SET "
   . "cliente=" . $db->quoteSmart($_REQUEST['cliente']) . ","
   . "resumen=" . $db->quoteSmart($_REQUEST['resumen']) . ","
   . "estado=" . $db->quoteSmart($_REQUEST['estado']) . ","
   . "fechainicio=" . $db->quoteSmart($fechainicioDB) . ","
   . "fechafin" . $db->quoteSmart($fechafinDB) . ","
   . "tiempoempleado=". $db->quoteSmart($_REQUEST['hora'] . ":" . $_REQUEST['minutos']));

if (DB::isError($db)) {
   echo "<h2>Error al insertar el cliente</h2>";
   ob_flush();
   die($db->getMessage());
} else {
   echo "<h2>Petición actualizada correctamente</h2>";
}

// Comprobamos si han añadido alguna nota
if (isset($_REQUEST['texto_nota']) && !empty($_REQUEST['texto_nota'])) {
   // Tenemos el identificador de petición en $_REQUEST['referencia']
   $db->query("INSERT INTO NOTA_PETICION VALUES ("
   . $db->quoteSmart($_REQUEST['referencia']) . ","
   . $db->quoteSmart($_REQUEST['texto_nota']) . ","
   . $db->quoteSmart(date("Y-M-d",mktime())) . ","
   . $db->quoteSmart($_REQUEST['nifEmpleado']) . ")");
   if (DB::isError($db)) {
      ob_clean();
      echo "<h2>Error al insertar la nota. Datos de la petición no actualizados</h2>";
      ob_flush();
      $db->rollback();
      die($db->getMessage());
   } else {
      echo "<h3>Nota actualizada correctamente</h2>";
   }
}
```

```
// Comprobamos si han añadido algún material+
if (isset($_REQUEST['nombrematerial']) && !empty($_REQUEST['nombrematerial'])) {
   // Tenemos el identificador de la petición en $_REQUEST['referencia']
   $db->query("INSERT INTO MATERIAL_PETICION VALUES "
   . $db->quoteSmart($_REQUEST['nombrematerial']) . ","
   . $db->quoteSmart($_REQUEST['referencia']) . ","
   . $db->quoteSmart($_REQUEST['precio']) . ","
   . $db->quoteSmart($_REQUEST['cantidad']) . ")");
   if (DB::isError($db)) {
      ob_clean();
      echo "<h2>Error al insertar el material. Datos de la petición no actualizados</h2>";
      ob_flush();
      $db->rollback();
      die($db->getMessage());
   } else {
      echo "<h3>Material actualizado correctamente</h2>";
   }
}

$db->commit();
$ob_flush();

$db->disconnect();
?>
```

**f. y g. Peticiones abiertas de un cliente y resumen de los materiales usados en cada una**

```
<?php
include_once 'datosconexion.php';

// Buscamos las peticiones abiertas de un cliente
$res=$db->query("SELECT P.referencia, P.resumen, P.estado, P.fecharecepcion FROM
PETICION P JOIN CLIENTE C ON P.cliente=C.nif WHERE ESTADO NOT IN ('Resuelta',
'Cerrada') ORDER BY fecharecepcion");

if (DB::isError($db)) {
   die($db->getMessage());
}

// Antes de empezar la iteración por las peticiones, vamos a preparar la consulta
// correspondiente a los materiales empleados en cada una.
$queryMaterial=$db->prepare("SELECT SUM(M.precio) as precioMateriales,
COUNT(M.nombrematerial) numMateriales FROM MATERIAL_PETICION M WHERE M.peticion=? ");

if (DB::isError($db)) {
   die($db->getMessage());
}
```

```
echo "<table>";
echo "<tr><th>Referencia</th><th>Resumen</th><th>Estado</th><th>Duración</th><th>
Fecharecepción</th><th>Precio materiales</th><th>Núm. materiales</th></tr>";
while($res->fetchInto($row,DB_FETCHMODE_ASSOC)) {
   echo "<tr>";
   echo "<td>" . $row['referencia'] . "</td>";
   echo "<td>" . $row['resumen'] . "</td>";
   echo "<td>" . $row['estado'] . "</td>";
   echo "<td>" . $row['fecharecepcion'] . "</td>";
   $resMaterial=$db->execute($queryMaterial,$row['referencia']);
   if (DB::isError($db)) {
      die($db->getMessage());
   } else {
      $res->fetchInto($rowMaterial,DB_FETCHMODE_ASSOC);
      echo "<td>" . $rowMaterial['precioMateriales'] . "</td>";
      echo "<td>" . $rowMaterial['numMateriales'] . "</td>";
   }
   echo "</tr>";
}

echo "</table>";
?>
```

Mediante estos tres ejemplos, disponemos ya de una base tanto de código, como de estilo y mecanismos de comprobación de error, para desarrollar el resto de la aplicación, sin tener en cuenta su diseño.

Hemos intentado escoger consultas y operaciones representativas del funcionamiento de la aplicación y, a la vez, que se correspondieran con las vistas en apartados anteriores. Además, hemos introducido algunas funciones PHP que suelen utilizarse en combinación con el trabajo en bases de datos para tipos concretos, y para el tratamiento de errores, para evitar que el usuario reciba información confusa en la página de resultados.

## Resumen

En esta unidad hemos visto las actividades más destacadas de las fases iniciales de un proyecto de desarrollo con conexión a bases de datos.

Más que resolver el propio caso, se trataba de identificar los aspectos clave de los casos reales y enlazarlos con el contenido del resto de unidades del curso.

Habréis podido identificar qué actividades son más relevantes para el resultado final del proyecto, en cuáles conviene invertir más tiempo y cuáles no son tan críticas para los objetivos de éste u otro caso similar.

Si habéis seguido la planificación sugerida y repasado cada unidad a la que que se hacía referencia, habréis podido comprender la aplicación práctica del material y se habrán alcanzado los objetivos que nos proponíamos al redactar esta unidad didáctica.

También es posible leer este caso de estudio como un capítulo final del curso, donde se desarrolla un ejemplo completo. Se ha intentado redactarlo con este doble cometido.