

# Bases de datos en MySQL

Luis Alberto Casillas Santillán  
Marc Gibert Ginestà  
Óscar Pérez Mora



# Índice

<b>Introducción</b>	5
<b>Objetivos</b>	6
<b>1. Características de MySQL</b>	7
1.1. Prestaciones	7
1.2. Limitaciones	8
<b>2. Acceso a un servidor MySQL</b>	9
2.1. Conectándose con el servidor	9
2.1.1. Servidores y clientes	9
2.1.2. Conectarse y desconectarse	10
2.2. Introducción de sentencias	10
2.2.1. Sentencias	11
2.2.2. Comandos en múltiples líneas	11
2.2.3. Cadenas de caracteres	12
2.2.4. Expresiones y variables	13
2.2.5. Expresiones	14
2.3. Proceso por lotes	14
2.4. Usar bases de datos	17
<b>3. Creación y manipulación de tablas</b>	20
3.1. Crear tablas	20
3.2. Tipos de datos	23
3.2.1. Tipos de datos numéricos	23
3.2.2. Cadenas de caracteres	24
3.2.3. Fechas y horas	25
3.3. Modificar tablas	25
3.3.1. Agregar y eliminar columnas	25
3.3.2. Modificar columnas	26
3.4. Otras opciones	27
3.4.1. Copiar tablas	27
3.4.2. Tablas temporales	27
<b>4. Consultas</b>	28
4.1. La base de datos demo	28
4.2. Consultar información	29
4.2.1. Funciones auxiliares	30
4.2.2. La sentencia EXPLAIN	31
4.3. Manipulación de filas	33
<b>5. Administración de MySQL</b>	35
5.1. Instalación de MySQL	35

5.2. Usuarios y privilegios .....	38
5.2.1. La sentencia GRANT .....	39
5.2.2. Especificación de lugares origen de la conexión .....	40
5.2.3. Especificación de bases de datos y tablas .....	41
5.2.4. Especificación de columnas .....	42
5.2.5. Tipos de privilegios .....	42
5.2.6. Opciones de encriptación .....	44
5.2.7. Límites de uso .....	44
5.2.8. Eliminar privilegios .....	45
5.2.9. Eliminar usuarios .....	45
5.2.10. La base de datos de privilegios: mysql .....	45
5.3. Copias de seguridad .....	48
5.3.1. mysqlhotcopy .....	50
5.3.2. mysqldump .....	50
5.3.3. Restaurar a partir de respaldos .....	51
5.4. Reparación de tablas .....	52
5.4.1. myisamchk .....	54
5.5. Análisis y optimización .....	55
5.5.1. Indexación .....	55
5.5.2. Equilibrio .....	57
5.5.3. La cache de consultas de MySQL .....	58
5.6. Replicación .....	59
5.6.1. Preparación previa .....	60
5.6.2. Configuración del servidor maestro .....	60
5.6.3. Configuración del servidor esclavo .....	61
5.7. Importación y exportación de datos .....	62
5.7.1. mysqlimport .....	63
5.7.2. mysqldump .....	63
<b>6. Clientes gráficos .....</b>	<b>65</b>
6.1. mysqlcc .....	65
6.2. mysql-query-browser .....	66
6.3. mysql-administrator .....	67
<b>Resumen .....</b>	<b>70</b>
<b>Bibliografía .....</b>	<b>71</b>

## Introducción

MySQL es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

MySQL está disponible para múltiples plataformas, la seleccionada para los ejemplos de este libro es GNU/Linux. Sin embargo, las diferencias con cualquier otra plataforma son prácticamente nulas, ya que la herramienta utilizada en este caso es el cliente *mysql-client*, que permite interactuar con un servidor MySQL (local o remoto) en modo texto. De este modo es posible realizar todos los ejercicios sobre un servidor instalado localmente o, a través de Internet, sobre un servidor remoto.

Para la realización de todas las actividades, es imprescindible que dispongamos de los datos de acceso del usuario administrador de la base de datos. Aunque en algunos de ellos los privilegios necesarios serán menores, para los capítulos que tratan la administración del SGBD será imprescindible disponer de las credenciales de administrador.

### Nota

Las sentencias o comandos escritos por el usuario estarán en *fuentes monoespaciada*, y las palabras que tienen un significado especial en MySQL estarán en **negrita**. Es importante hacer notar que estas últimas no siempre son palabras reservadas, sino comandos o sentencias de *mysql-client*.

La versión de MySQL que se ha utilizado durante la redacción de este material, y en los ejemplos, es la 4.1, la última versión estable en ese momento, aunque no habrá ningún problema en ejecutarlos en versiones anteriores, hasta la 3.23.

### Nota

Podremos utilizar la licencia GPL de MySQL siempre que el programa que lo use también lo sea, en caso contrario se debe adquirir la "licencia comercial", entre 250 y 500 €, en el momento de escribir este material.

## Objetivos

Adquirir las habilidades y conocimientos de MySQL necesarios para utilizar y administrar este SGBD (sistema gestor de bases de datos).

## 1. Características de MySQL

En este apartado enumeraremos las prestaciones que caracterizan a este SGBD, así como las deficiencias de diseño, limitaciones o partes del estándar aún no implementadas.

### 1.1. Prestaciones

MySQL es un SGBD que ha ganado popularidad por una serie de atractivas características:

- Está desarrollado en C/C++.
- Se distribuyen ejecutables para cerca de diecinueve plataformas diferentes.
- La API se encuentra disponible en C, C++, Eiffel, Java, Perl, PHP, Python, Ruby y TCL.
- Está optimizado para equipos de múltiples procesadores.
- Es muy destacable su velocidad de respuesta.
- Se puede utilizar como cliente-servidor o incrustado en aplicaciones.
- Cuenta con un rico conjunto de tipos de datos.
- Soporta múltiples métodos de almacenamiento de las tablas, con prestaciones y rendimiento diferentes para poder optimizar el SGBD a cada caso concreto.
- Su administración se basa en usuarios y privilegios.
- Se tiene constancia de casos en los que maneja cincuenta millones de registros, sesenta mil tablas y cinco millones de columnas.
- Sus opciones de conectividad abarcan TCP/IP, *sockets* UNIX y *sockets* NT, además de soportar completamente ODBC.
- Los mensajes de error pueden estar en español y hacer ordenaciones correctas con palabras acentuadas o con la letra 'ñ'.
- Es altamente confiable en cuanto a estabilidad se refiere.

Para todos aquellos que son adeptos a la filosofía de UNIX y del lenguaje C/C++, el uso de MySQL les será muy familiar, ya que su diseño y sus interfaces son acordes a esa filosofía: “crear herramientas que hagan una sola cosa y que la hagan bien”. MySQL tiene como principal objetivo ser una base de datos fiable y eficiente. Ninguna característica es implementada en MySQL si antes no se tiene la certeza que funcionará con la mejor velocidad de respuesta y, por supuesto, sin causar problemas de estabilidad.

La influencia de C/C++ y UNIX se puede observar de igual manera en su sintaxis. Por ejemplo, la utilización de expresiones regulares, la diferenciación de funciones por los paréntesis, los valores lógicos como 0 y 1, la utilización del tabulador para completar sentencias, por mencionar algunos.

## 1.2. Limitaciones

Al comprender sus principios de diseño, se puede explicar mejor las razones de algunas de sus carencias. Por ejemplo, el soporte de transacciones o la integridad referencial (la gestión de claves foráneas) en MySQL está condicionado a un esquema de almacenamiento de tabla concreto, de forma que si el usuario no va a usar transacciones, puede usar el esquema de almacenamiento “tradicional” (MyISAM) y obtendrá mayor rendimiento, mientras que si su aplicación requiere transacciones, deberá usar el esquema que lo permite (InnoDB), sin ninguna otra restricción o implicación.

### Nota

El esquema de tabla que hay que usar se decide para cada una en el momento de su creación, aunque puede cambiarse posteriormente. Actualmente, MySQL soporta varios esquemas y permite la incorporación de esquemas definidos por el usuario.

Otras limitaciones son las siguientes:

- No soporta procedimientos almacenados (se incluirán en la próxima versión 5.0).
- No incluye disparadores (se incluirán en la próxima versión 5.0).
- No incluye vistas (se incluirán en la próxima versión 5.0).
- No incluye características de objetos como tipos de datos estructurados definidos por el usuario, herencia etc.



## 2. Acceso a un servidor MySQL

En este apartado veremos las distintas formas de acceso a un servidor MySQL existente que nos proporciona el propio SGBD. El acceso desde lenguajes de programación o herramientas en modo gráfico se tratará en otros apartados.

### 2.1. Conectándose con el servidor

Para conectarse con el servidor deberemos asegurarnos de que éste está funcionando y de que admite conexiones, sean éstas locales (el SGBD se está ejecutando en la misma máquina que intenta la conexión) o remotas.

Adicionalmente, deberemos disponer de las credenciales necesarias para la conexión. Distintos tipos de credenciales nos permitirán distintos niveles de acceso. Para simplificar, supondremos que disponemos de las credenciales (usuario y contraseña) del administrador de la base de datos (normalmente, usuario *root* y su contraseña). En el apartado que concierne a la administración de MySQL, se comenta detalladamente los aspectos relacionados con el sistema de usuarios, contraseñas y privilegios del SGBD.

#### 2.1.1. Servidores y clientes

El servidor MySQL es el servicio *mysqld*, que puede recibir solicitudes de clientes locales o remotos a través TCP/IP, *sockets* o *pipes* en forma de ficheros locales a la máquina en que se está ejecutando. En la distribución se incluye un cliente llamado *mysql-client*, al que en adelante nos referiremos simplemente como *mysql* (así es como se llama el programa ejecutable). Si se invoca sin parámetros, *mysql* realiza una conexión al servidor local utilizando el nombre del usuario UNIX que lo ha invocado, y supone que este usuario no requiere contraseña. La conexión a un servidor remoto y un nombre de usuario específicos requiere de al menos dos argumentos:

- *-h* para especificar el nombre del servidor.
- *-u* para el nombre del usuario.

Para que el programa cliente pregunte la contraseña de conexión al usuario, deberemos proporcionar adicionalmente el parámetro *-p*.

#### Nota

El servidor MySQL es *mysqld*. A él se pueden conectar múltiples clientes. *mysql* es el cliente en modo texto que proporciona el propio SGBD.

```
$ mysql -h servidor.misitio.org -u <usuario> -p
```

### 2.1.2. Conectarse y desconectarse

Si se tiene algún problema para realizar la conexión, es necesario consultar con el administrador del sistema, que nos proporcionará un nombre de usuario, contraseña y el nombre del servidor, según sea necesario, y nos informará de las restricciones que tiene nuestra cuenta.

La administración y seguridad de MySQL está diseñada sobre un esquema de usuarios y privilegios. Los usuarios deben ser creados por el administrador con sus respectivos privilegios y restricciones. Es el administrador quien decide si los nombres de los usuarios de MySQL se corresponden o no a los del sistema operativo.

#### Nota

Los usuarios del sistema operativo y los de MySQL no son los mismos, aunque el administrador de MySQL (con fines prácticos) pueda utilizar los mismos nombres para las cuentas de los usuarios MySQL.

Apariencia de *mysql* al ingresar en el modo interactivo:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.49-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Con el comando *help* obtenemos una serie de opciones (veremos las más utilizadas).

Para salir del cliente podemos escribir '*\q*' o '*quit*':

```
mysql> quit;
```

Tanto para el comando *quit* como para el comando *help*, el punto y coma al final es opcional.

### 2.2. Introducción de sentencias

El cliente de MySQL en modo interactivo nos permite tanto la introducción de sentencias SQL para trabajar con la base de datos (crear tablas, hacer consultas y ver sus resultados, etc.) como la ejecución de comandos propios del SGBD para obtener información sobre las tablas, índices, etc. o ejecutar operaciones de administración.

#### Sentencias

Las sentencias en *mysql* pueden abarcar múltiples líneas y terminan con punto y coma.

### 2.2.1. Sentencias

A continuación presentamos una ejecución de la sentencia *select* con cuatro columnas de datos:

```
mysql> select user(), connection_id(), version(), database();
+-----+-----+-----+-----+
| user() | CONNECTION_ID() | version() | database() |
+-----+-----+-----+-----+
| yo@localhost | 4 | 3.23.49-log | |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql>
```

En esta consulta se solicita, a través de funciones incorporadas en el SGBD, el nombre del usuario actual de MySQL, el número de conexión al servidor, la versión del servidor y la base de datos en uso. Las funciones se reconocen por los paréntesis al final. *mysql* entrega sus resultados en tablas, en la que el primer renglón son los encabezados de las columnas. Es importante no dejar espacio entre el nombre de una función y los paréntesis, de otro modo, *mysql* marcará un mensaje de error.

La última línea entregada por *mysql* informa sobre el número de filas encontrado como resultado de la consulta y el tiempo estimado que llevó su realización. Esta medida de tiempo no se debe considerar muy precisa para medir el rendimiento del servidor, se trata simplemente de un valor aproximado que puede verse alterado por múltiples factores.

Observamos que la columna con el nombre de la base de datos actual esta vacía. Esto es natural, ya que no hemos creado aún ninguna base de datos ni le hemos indicado al gestor sobre cuál queremos trabajar.

### 2.2.2. Comandos en múltiples líneas

Los comandos pueden expandirse en varias líneas por comodidad, sobre todo al escribir largas sentencias SQL. El cliente no enviará la sentencia SQL al servidor hasta encontrar el punto y coma, de este modo, el comando anterior puede escribirse así:

```
mysql> select user(),
-> connection_id(),
-> version(),
-> database();
+-----+-----+-----+-----+
| user() | CONNECTION_ID() | version() | database() |
+-----+-----+-----+-----+
| yo@localhost | 4 | 3.23.49-log | |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql>
```

Obsérvese el indicador de *mysql* que se transforma en `->`, signo que significa que el comando aún no está completo. También pueden escribirse varios comandos en una sola línea, cada uno debe llevar su respectivo punto y coma:

```
mysql> select now(); select user();
+-----+
| CONNECTION_ID() |
+-----+
| 4               |
+-----+
1 row in set (0.00 sec)
+-----+
| user()          |
+-----+
| yo@localhost    |
+-----+
1 row in set (0.01 sec)
mysql>
```

Se ejecutarán en el orden que están escritos. Los comandos se pueden cancelar con la combinación `\c`, con lo que el cliente nos volverá a mostrar el indicador para que escribamos de nuevo la sentencia.

```
mysql> select now(),
-> uso
-> ver \c
mysql>
```

Indicadores de *mysql*

Indicador	Significado
mysql>	Espera una nueva sentencia
->	La sentencia aún no se ha terminado con ;
">	Una cadena en comillas dobles no se ha cerrado
'>	Una cadena en comillas simples no se ha cerrado

### 2.2.3. Cadenas de caracteres

Las cadenas de caracteres pueden delimitarse mediante comillas dobles o simples. Evidentemente, deben cerrarse con el mismo delimitador con el que se han abierto.

```
mysql> select "Hola mundo",'Felicidades';
```

y pueden escribirse en diversas líneas:

```
mysql> select "Éste es un texto
        "> en dos renglones";
```

Al principio, es común olvidar el punto y coma al introducir un comando y, también, olvidar cerrar las comillas. Si éste es el caso, hay que recordar que *mysql* no interpreta lo que está entre comillas, de tal modo que para utilizar el comando de cancelación '\c' es preciso antes cerrar las comillas abiertas:

```
mysql> select "Éste es un texto
        "> \c
        "> " \c
mysql>
```

#### 2.2.4. Expresiones y variables

MySQL dispone de variables de sesión, visibles únicamente durante la conexión actual. Éstas pueden almacenar valores de tipos enteros, flotantes o cadenas, pero no tablas. Se definen como en el siguiente ejemplo:

```
mysql> select @x := 1;
```

La variable local @x tiene ahora el valor 1 y puede utilizarse en expresiones:

```
mysql> select @x, sqrt(@x), sin(@x), @x + 10, @x > 10;
+-----+-----+-----+-----+-----+
| @x    | sqrt(@x) | sin(@x) | @x + 10 | @x > 10 |
+-----+-----+-----+-----+-----+
| 1     | 1.000000 | 0.841471 | 11      | 0       |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

Las variables locales permiten almacenar datos entre consultas y, en la práctica, es recomendable utilizarlas exclusivamente con este fin, por ejemplo:

```
mysql> select @hora_ingreso := now();
mysql> select now() - @ingreso;
+-----+
| now() - @ingreso |
+-----+
| 20040124138051   |
+-----+
1 row in set (0.00 sec)
```

### 2.2.5. Expresiones

Hay que tener cuidado con el uso de las variables locales por los motivos siguientes:

- Se evalúan en el servidor al ser enviadas por el cliente.
- Se realizan conversiones de tipo implícitas.


```
mysql> do @ingreso := now();
```

#### Nota

El comando **do** evalúa expresiones sin mostrar los resultados en pantalla. Se puede evaluar cualquier expresión que admite el comando **select**.

Las variables no requieren declaración y, por omisión, contienen el valor NULL que significa “ausencia de valor”, observad en la siguiente consulta los resultados de utilizar valores nulos:

```
mysql> select @y,
-> sqrt( @y ),
-> @y + 10,
-> @y < 1 ;
+-----+-----+-----+-----+
| @y    | sqrt( @y ) | @y + 10 | @y < 1 |
+-----+-----+-----+-----+
| NULL  | NULL       | NULL    | NULL    |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

La razón de este comportamiento es que no es posible realizar ninguna operación cuando se desconoce algún valor. La entrada de valores NULL siempre significará salida de valores NULL. 

#### Nota

En una expresión donde cualquiera de sus elementos sea NULL, automáticamente entregará como resultado el valor NULL.

### 2.3. Proceso por lotes

MySQL puede procesar por lotes las sentencias contenidas en un archivo de texto. Cada sentencia deberá terminar en ';' igual que si la escribiéramos en el cliente. La sintaxis es la siguiente:

```
$ mysql -u juan -h servidor.misitio.org -p < demo.sql
```

En este caso, se realizará una conexión con el servidor, nos pedirá la contraseña del usuario 'juan' y, si ésta es correcta, ejecutará los comandos incluidos en el archivo *demo.sql*, uno a uno y por el mismo orden. Imprimirá los resultados (o errores) en la salida estándar (o de error) y terminará. De este modo evitaremos la molestia de procesarlos uno por uno de forma interactiva.

Otra forma de procesar un archivo es mediante el comando `source` desde el indicador interactivo de MySQL:

```
mysql> source demo.sql
```

El archivo *demo.sql* crea una nueva base de datos.

El usuario debe tener permisos para crear bases de datos si quiere que sea procesado el archivo *demo.sql*. Si el administrador crea la base de datos por nosotros, será necesario editarlo, comentando la línea donde se crea la base de datos con el símbolo '#' al inicio:

```
# create database demo;
```

Es necesario procesar el contenido del fichero *demo.sql* tal como los transcribimos aquí, con el fin de poder realizar los ejemplos del resto del apartado. Si se observa su contenido, posiblemente muchas cosas se expliquen por sí mismas, de cualquier manera, serán explicadas en este apartado. También pueden ejecutarse sus órdenes en el cliente directamente.

#### Contenido del fichero demo.sql

```
#drop database demo;
create database demo;
use demo;
---
--- Estructura de la tabla productos
---

create table productos (
  parte      varchar(20),
  tipo       varchar(20) ,
  especificación varchar(20) ,
  psugerido  float(6,2),
  clave int(3) zerofill not null auto_increment,
  primary key (clave)
);
insert into productos (parte,tipo,especificación,psugerido) values
  ('Procesador','2 GHz','32 bits',null),
  ('Procesador','2.4 GHz','32 bits',35),
  ('Procesador','1.7 GHz','64 bits',205),
  ('Procesador','3 GHz','64 bits',560),
  ('RAM','128MB','333 MHz',10),
  ('RAM','256MB','400 MHz',35),
  ('Disco Duro','80 GB','7200 rpm',60),
```

```
    ('Disco Duro','120 GB','7200 rpm',78),
    ('Disco Duro','200 GB','7200 rpm',110),
    ('Disco Duro','40 GB','4200 rpm',null),
    ('Monitor','1024x876','75 Hz',80),
    ('Monitor','1024x876','60 Hz',67)
;

--
-- Estructura de la tabla 'proveedor'
--
create table proveedores (
    empresa    varchar(20) not null,
    pago       set('crédito','efectivo'),
    primary key (empresa)
);
--
-- Valores de la tabla 'proveedor'
--
insert into proveedores (empresa,pago) values
    ('Tecno-k','crédito'),
    ('Patito','efectivo'),
    ('Nacional','crédito,efectivo')
;
create table ganancia(
    venta      enum('Por mayor','Por menor'),
    factor      decimal(2,2)
);
insert into ganancia values
    ('Por mayor',1.05),
    ('Por menor',1.12)
;
create table precios (
    empresa    varchar(20) not null,
    clave      int(3) zerofill not null,
    precio      float(6,2),

    foreign key (empresa) references proveedores,
    foreign key (clave) references productos
);

insert into precios values
    ('Nacional',001,30.82),
    ('Nacional',002,32.73),
    ('Nacional',003,202.25),
    ('Nacional',005,9.76),
    ('Nacional',006,31.52),
    ('Nacional',007,58.41),
    ('Nacional',010,64.38),
    ('Patito',001,30.40),
    ('Patito',002,33.63),
    ('Patito',003,195.59),
    ('Patito',005,9.78),
    ('Patito',006,32.44),
```



```

('Patito',007,59.99),
('Patito',010,62.02),
('Tecno-k',003,198.34),
('Tecno-k',005,9.27),
('Tecno-k',006,34.85),
('Tecno-k',007,59.95),
('Tecno-k',010,61.22),
('Tecno-k',012,62.29)
;

```

Si se desea llevar un registro de todas las operaciones de una sesión, se puede utilizar la expresión siguiente; de este modo se guardarán todos los comandos y sus resultados en *archivo\_registro.txt*:

```
mysql> tee archivo_registro.txt
```

Para cancelar la captura, basta con teclear lo siguiente:

```
mysql> notee
```

## 2.4. Usar bases de datos

La siguiente consulta informa sobre la base de datos actualmente en uso.

```

mysql> select database();
+-----+
| database() |
+-----+
|           |
+-----+
1 row in set (0.13 sec)

```

El campo está vacío porque no estamos haciendo uso de ninguna base de datos. Para ver las bases de datos existentes en el sistema, se debe efectuar la siguiente consulta:

```

mysql> show databases;
+-----+
| Database |
+-----+
| demo     |
| mysql    |
| test     |
+-----+
3 rows in set (0.01 sec)

```

MySQL nos muestra el listado de las bases de datos definidas en el servidor. Debe aparecer la base de datos *demo* que creamos con el archivo *demo.sql*. Para poder trabajar con ella, tenemos que abrirla:

```
mysql> use demo;
```

#### use

El comando *use base\_de\_datos* permite abrir una base de datos para su uso.

#### Nota

Es posible realizar consultas en una base de datos sin utilizar el comando **use**, en ese caso, todos los nombres de las tablas deben llevar el nombre de la base de datos a que pertenecen de la forma: *demo.productos*.

Otra posibilidad consiste en proporcionar el nombre de la base de datos al iniciar una sesión interactiva con *mysql*:

```
$ mysql demo -u juan -p
```

La consulta de las tablas que contiene la base de datos *demo* se realiza con la sentencia **show** de la siguiente manera:

```
mysql> show tables;
+-----+
| Tables_in_demo |
+-----+
| partes          |
| proveedores     |
+-----+
2 rows in set (0.00 sec)
```

#### Nota

El comando *show* es útil para mostrar información sobre las bases de datos, tablas, variables y otra información sobre el SGBD. Podemos utilizar *help show* en el intérprete de comandos para obtener todas las variantes de esta sentencia.

Asimismo, podemos consultar las columnas de cada una de las tablas:

```
mysql> describe productos;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| parte          | varchar(20)   | YES  |     | NULL    |                |
| tipo           | varchar(20)   | YES  |     | NULL    |                |
| especificación | varchar(20)   | YES  |     | NULL    |                |
| Field          | float(6,2)    | YES  |     | NULL    |                |
| Field          | int(3) unsigned zerofill | YES  | PRI | NULL    | auto_increment |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Para crear una nueva base de datos usaremos la sentencia `create database`:

```
mysql> create database prueba;
```

Para eliminar una base de datos, usaremos la sentencia `drop database`:

```
mysql> drop database prueba;
```

MySQL es sensible al uso de mayúsculas y minúsculas, tanto en la definición de bases de datos, como de tablas o columnas.


### 3. Creación y manipulación de tablas

#### 3.1. Crear tablas

Una vez realizada la conexión con el servidor MySQL y después de abrir una base de datos, podemos crear tablas en ella de la siguiente manera:

```
mysql> create table personas (  
-> nombre char(30),  
-> dirección char(40),  
-> teléfono char(15)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

En este caso, la sentencia `create table` construye una nueva tabla en la base de datos en uso. La tabla contiene tres columnas, *nombre*, *dirección* y *teléfono*, todas de tipo carácter y de longitudes 30, 40 y 15 respectivamente. Si se intenta guardar en ellas valores que sobrepasen esos límites, serán truncados para poderlos almacenar. Por ese motivo, es importante reservar espacio suficiente para cada columna. Si se prevé que muchos registros ocuparán sólo una fracción del espacio reservado, se puede utilizar el tipo `varchar`, similar a `char`, con la diferencia de que el valor ocupará un espacio menor al especificado si la cadena es más corta que el máximo indicado, ahorrando así espacio de almacenamiento.

Los nombres de las columnas admiten caracteres acentuados. 

Las tablas pueden eliminarse con **drop table**:

```
mysql> drop table personas;  
Query OK, 0 rows affected (0.01 sec)
```

Alternativamente, se puede utilizar la sintaxis siguiente:

```
mysql> drop table if exists personas;
```

## Atributos de columna

Atributo	Significado
<b>null</b>	Se permiten valores nulos, atributo por omisión si no se especifica lo contrario.
<b>not null</b>	No se permiten valores nulos.
<b>default valor</b>	Valor por omisión que se asigna a la columna.
<b>auto_increment</b>	El valor se asigna automáticamente incrementando en uno el máximo valor registrado hasta ahora. Se aplica sólo a las columnas marcadas como clave primaria.
<b>primary key</b>	Señala al campo como clave primaria, implícitamente también lo declara como <b>not null</b> .


Veámoslo con un ejemplo:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> dirección varchar(50) null,
-> edo_civil char(13) default 'Soltero',
-> num_registro int primary key auto_increment,
-> );
Query OK, 0 rows affected (0.01 sec)
```

En este caso la tabla contiene cuatro columnas, de las cuales *nombre* y *edo\_civil* permiten valores nulos, en *edo\_civil* está implícito al no declarar lo contrario. La columna *num\_registro* no acepta valores nulos porque está definida como clave primaria.

**Nota**

La definición de columnas tiene el siguiente formato:  
**nombre\_columna tipo atributos.**

Aunque la creación de una clave primaria puede declararse como atributo de columna, es conveniente definirla como restricción de tabla, como se verá enseguida. 

También es posible indicar restricciones sobre la tabla y no sobre columnas específicas:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> nacimiento date not null,
-> pareja varchar(40),
-> proveedor int not null,
->
-> primary key (nombre,nacimiento),
-> unique (pareja),
-> foreign key (proveedor) references proveedores
-> );
Query OK, 0 rows affected (0.01 sec)
```

## Restricciones de tabla

Restricción	Significado
<b>primary key</b>	Define la o las columnas que servirán como clave primaria. Las columnas que forman parte de la clave primaria deben de ser <b>not null</b> .
<b>unique</b>	Define las columnas en las que no pueden duplicarse valores. Serán las claves candidatas del modelo relacional.
<b>foreign key</b> ( <i>columna</i> ) <b>references</b> <i>tabla</i> ( <i>columna2</i> )	Define que los valores de <i>columna</i> se permitirán sólo si existen en <i>tabla(columna2)</i> . Es decir, <i>columna</i> hace referencia a los registros de <i>tabla</i> , esto asegura que no se realicen referencias a registros que no existen.

Se definen tres restricciones sobre la tabla después de la definición de cuatro columnas:

- La primera restricción se refiere a la clave primaria, compuesta por las columnas *nombre* y *nacimiento*: no puede haber dos personas que se llamen igual y que hayan nacido en la misma fecha. La clave primaria permite identificar de manera unívoca cada registro de la tabla.
- La segunda restricción define que la pareja de una persona debe ser única: dos personas no pueden tener la misma pareja. Todo intento de insertar un nuevo registro donde el nombre de la pareja ya exista, será rechazado. Cuando se restringe una columna con **unique**, los valores **null** reciben un trato especial, pues se permiten múltiples valores nulos.
- La tercera restricción afecta a la columna *proveedor*, sólo puede tomar valores que existan en la clave primaria de la tabla *proveedores*.

Las restricciones de tabla pueden definirse con un identificador útil para hacer referencias posteriores a la restricción:

```
mysql> create table personas (
-> nombre varchar(40) not null,
-> nacimiento date not null,
-> pareja varchar(40),
-> proveedor int not null,
->
-> constraint clave primary key (nombre,nacimiento),
-> constraint monogamo unique (pareja),
-> constraint trabaja_en foreign key (proveedor) references
proveedores
-> );
```

**key / index**

La definición de índices puede hacerse también en el momento de creación de la tabla, mediante la palabra clave **key** (o **index**), a la que deberemos proporcionar el nombre que vamos a asignar a esta clave y las columnas que la forman, entre paréntesis. Existen modificadores opcionales sobre el índice que nos permiten especificar si se trata de un índice único o múltiple (según puedan existir o no varios valores iguales del índice en la tabla).

En versiones recientes de MySQL existen otros tipos de índices (espaciales, de texto completo, etc.) para tipos de datos concretos y que ofrecen prestaciones adicionales.

**Claves foráneas**

Las restricciones de tabla **foreign key** no tienen efecto alguno en MySQL 4.0 y anteriores, ya que esta característica no está implementada. Se admite en la sintaxis por compatibilidad, ya que será implementada en una versión posterior. En la versión 4.1, está soportada si se utiliza el tipo de tabla InnoDB.

### 3.2. Tipos de datos

MySQL cuenta con un rico conjunto de tipos de datos para las columnas, que es necesario conocer para elegir mejor cómo definir las tablas. Los tipos de datos se pueden clasificar en tres grupos:

- Numéricos.
- Cadenas de caracteres
- Fechas y horas

El valor **null** es un caso especial de dato, ya que al significar *ausencia de valor* se aplica a todos los tipos de columna. Los siguientes símbolos se utilizan en la definición y descripción de los tipos de datos en MySQL:

- **M** - El ancho de la columna en número de caracteres.
- **D** - Número de decimales que hay que mostrar.
- **L** - Longitud o tamaño real de una cadena.
- **[ ]** - Lo que se escriba entre ellos es opcional.

#### 3.2.1. Tipos de datos numéricos

Los tipos de datos numéricos comprenden dos categorías, los enteros y los números con punto flotante.

##### Números enteros


La principal diferencia entre cada uno de los tipos de enteros es su tamaño, que va desde 1 byte de almacenamiento hasta los 8 bytes. Las columnas de tipo entero pueden recibir dos atributos adicionales, que deben especificarse inmediatamente después del nombre del tipo:

- **unsigned**. Indica que el entero no podrá almacenar valores negativos. Es responsabilidad del usuario verificar, en este caso, que los resultados de las restas no sean negativos, porque MySQL los convierte en positivos.
- **zerofill**. Indica que la columna, al ser mostrada, rellenará con ceros a la izquierda los espacios vacíos. Esto de acuerdo al valor especificado por **M** en la declaración del tipo. Una columna con el atributo **zerofill** es al mismo tiempo **unsigned** aunque no se especifique.

##### Ejemplo

```
create table números (  
  x int(4) zerofill not null,  
  y int(5) unsigned  
);
```

El comando anterior crea una tabla con dos columnas. Ambas ocuparán un espacio de 4 bytes, pero al mostrarse, la columna *x* ocupará un espacio de 4 dígitos y la columna *y*, de 5.

Tanto **zerofill** como **unsigned** deben escribirse siempre antes que cualquier otro atributo de columna. 

Tipos enteros

Tipo	Espacio de almacenamiento	Significado
<b>tinyint</b> [(M)]	1 byte	Entero muy pequeño
<b>smallint</b> [(M)]	2 bytes	Entero pequeño
<b>mediumint</b> [(M)]	3 bytes	Entero mediano
<b>int</b> [(M)]	4 bytes	Entero
<b>bigint</b> [(M)]	8 bytes	Entero grande

### Números con punto flotante

MySQL cuenta con los tipos **float** y **double**, de 4 y 8 bytes de almacenamiento. Además incluye el tipo decimal, que se almacena como una cadena de caracteres y no en formato binario.

Números de punto flotante

Tipo	Espacio de almacenamiento	Significado
float	4 bytes	Simple precisión
double	8 bytes	Doble precisión
decimal	M + 2 bytes	Cadena de caracteres representando un número flotante

### 3.2.2. Cadenas de caracteres

Cadenas de caracteres

Tipo	Equivalente	Tamaño máximo	Espacio de almacenamiento
<b>char</b> [(M)]		M bytes	M bytes
<b>varchar</b> [(M)]		M bytes	L+1 bytes
<b>tinytext</b>	tinyblob	2 <sup>8</sup> –1 bytes	L+1 bytes
text	blob	2 <sup>16</sup> –1 bytes	L+2 bytes
mediumtext	mediumblob	2 <sup>24</sup> –1 bytes	L+3 bytes
longtext	longblob	2 <sup>32</sup> –1 bytes	L+4 bytes
<b>enum</b> ('v1','v2',...)		65535 valores	1 o 2 bytes
<b>set</b> ('v1','v2',...)		64 valores	1 a 8 bytes

Si observamos la tabla, vemos que el único tipo de dato que siempre utiliza el tamaño especificado por M es el tipo **char**. Por este motivo, se ofrece el tipo **varchar** que ocupa sólo el espacio requerido por el valor de la columna.

#### Ejemplo

```
create table persona(  
  comentario char(250),  
  recado varchar(250)  
);
```



La columna *comentario* ocupará 250 bytes de espacio de almacenamiento, sin importar el valor almacenado. Por el contrario, la columna *recado* ocupará sólo el espacio necesario según el valor asignado; por ejemplo, la cadena “Instalar MySQL” tiene 14 bytes de longitud, y el campo *recado* ocuparía 15 bytes para almacenarla.

Los tipos `text` y `blob` son equivalentes, pero `text` respeta las mayúsculas, minúsculas y caracteres acentuados en la ordenación.

### Ejemplo del uso de los tipos enumerados o `enum`

```
create table persona (
  edo_civil enum('soltero','casado','viudo','divorciado')
);
```

La columna *edo\_civil* de la tabla en la sentencia anterior, solo podrá almacenar los valores ‘soltero’, ‘casado’, ‘viudo’, ‘divorciado’, que son especificados por el tipo `enum`. La columna ocupará el espacio de un byte, ya que los valores `enum` son representados internamente por números.

### 3.2.3. Fechas y horas

Fechas y horas

Tipo	Espacio de almacenamiento	Rango
<code>date</code>	3 bytes	‘1000-01-01’ al ‘9999-12-31’
<code>time</code>	3 bytes	‘-838:59:59’ a ‘838:59:59’
<code>datetime</code>	8 bytes	‘1000-01-01 00:00:00’ a ‘9999-12-31 23:59:59’
<code>timestamp[(M)]</code>	4 bytes	19700101000000 al año 2037
<code>year[(M)]</code>	1 byte	1901 a 2155

## 3.3. Modificar tablas

### 3.3.1. Agregar y eliminar columnas

Alterar la estructura de una tabla es una tarea más frecuente de lo que uno puede imaginar en un principio. La sentencia ***alter table*** permite una amplia gama de formas de modificar una tabla. La siguiente sentencia nos recuerda un poco a la estructura de la sentencia ***create table***, en donde modificamos la tabla *personal* creada en la sección anterior.

```
mysql> alter table personal add (
-> mascota char(30) default 'perro',
-> pasatiempo char(20) not null
-> );
```

#### Nota

Siempre es posible consultar la estructura de una tabla con el comando ***describe tabla***.

Después de ejecutar la sentencia anterior, aparecen dos nuevas columnas en la tabla. Si queremos agregar una sola columna, podemos usar la sintaxis siguiente:

```
mysql> alter table personal add capital int not null  
-> after nom;
```

Este formato de **alter table** permite, además, insertar las columnas antes (**before**) o después (**after**) de una columna en cuestión.

Las columnas no deseadas pueden eliminarse con la opción **drop**.

```
mysql> alter table personal drop pasatiempo;
```

### 3.3.2. Modificar columnas

La modificación de una columna con la opción **modify** es parecida a volver a definirla.

```
mysql> alter table personal modify  
-> mascota char (14) default 'gato';
```

Después de la sentencia anterior, los atributos y tipo de la columna han cambiado por los especificados. Lo que no se puede cambiar con esta sintaxis es el nombre de la columna. Para ello, se debe utilizar la opción **change**:

```
mysql> alter table personal change nom  
-> nombre char (20);
```

La columna que se llamaba *nom* cambia a *nombre*.

Con el mismo comando **alter table** podemos incluso realizar la ordenación física de una tabla bajo una columna específica:

```
mysql> alter table personal order by nom;  
Query OK, 0 rows affected (0.06 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

#### Nota

En general, una tabla no puede durar mucho tiempo con un **order** respecto a una columna, ya que las inserciones no se realizarán respetando el orden establecido. Solamente en tablas que no van a ser actualizadas es útil aplicar este comando.

Finalmente, podemos cambiar de nombre la tabla:

```
mysql> alter table personal rename gente;
```

#### rename table

El comando **rename table** *viejo\_nombre* to *nuevo\_nombre* es una forma alternativa de cambiar el nombre a una tabla.

### 3.4. Otras opciones

#### 3.4.1. Copiar tablas

Aunque no existe un comando explícito para copiar tablas de una base de datos a otra, es posible utilizar el comando **rename table** para este propósito; basta con especificar la base de datos a la que pertenece una tabla:

```
mysql> rename table base_uno.tabla to base_dos.tabla;
```

También es posible crear una tabla nueva con el contenido de otra ya existente (copiando los datos):

```
mysql> create table nueva_tabla select * from otra_tabla;
```

La siguiente sentencia es equivalente, pero no copia los datos de la tabla origen:

```
mysql> create table nueva_tabla like otra_tabla;
```

#### 3.4.2. Tablas temporales

MySQL permite la creación de tablas temporales, visibles exclusivamente en la sesión abierta, y guardar datos entre consultas. La creación de una tabla temporal sólo requiere la utilización de la palabra **temporary** en cualquier formato del comando **create table**. La utilidad de las tablas temporales se limita a consultas complejas que deben generar resultados intermedios que debemos consultar (hacer 'join' con ellas) varias veces o en consultas separadas. Internamente, MySQL genera también tablas temporales para resolver determinadas consultas:

```
mysql> create temporary table nueva_tabla ...
```

## 4. Consultas

Como ya hemos explicado, las consultas sobre la base de datos se ejecutan mediante sentencias `SELECT` introducidas en el propio programa cliente y los resultados se presentan en forma de tabla.

### 4.1. La base de datos *demo*

En esta sección utilizaremos la base de datos *demo* que hemos creado con el comando `source demo.sql`. Así que, antes de estudiar las consultas en MySQL, revisaremos brevemente la estructura de esta base de datos, que consta de las siguientes tablas:

!   
Podeis ver la creación de la base de datos *demo* en el apartado "Proceso por lotes" de esta misma unidad didáctica.

```
mysql> show tables;
+-----+
| Tables_in_demo |
+-----+
| ganancia       |
| precios        |
| productos      |
| proveedores    |
+-----+
```

Las cuatro tablas representan, de manera ficticia, la base de datos de un distribuidor de equipos de procesamiento. Están diseñadas para servir de ejemplo a los casos presentados en este capítulo, por lo que no necesariamente serán útiles en la vida real.

En nuestro ejemplo imaginario representamos la siguiente situación.

- Nuestro vendedor tiene una relación de proveedores que venden sus productos a crédito, en efectivo o ambos. Las compras a crédito pagan intereses, pero son útiles porque no siempre es posible pagar en efectivo. Se utiliza una columna de tipo conjunto para *pago*, que puede tomar los valores '*crédito*', '*efectivo*' o ambos:

```
create table proveedores (
  empresa varchar(20) not null,
  pago set('crédito','efectivo'),
  primary key (empresa)
);
```

Los productos que se distribuyen son partes de equipo de cómputo. Para la mayoría de los productos en el mercado, los fabricantes sugieren un precio de venta al público que, aunque no es obligatorio, los consumidores no están dispuestos a pagar más. Las claves de los productos son asignadas para control

interno con un número consecutivo. Con estas especificaciones, la tabla *productos* se define de la manera siguiente:

```
create table productos (  
  parte varchar(20),  
  tipo varchar(20) ,  
  especificación varchar (20) ,  
  psugerido float(6,2),  
  clave int(3) zerofill not null auto_increment,  
  primary key (clave)  
);
```


- La empresa define una política para las ganancias mínimas que se deben obtener en ventas: el 5% al por mayor y el 12% al por menor. Estos valores se almacenan en la tabla *ganancias*, donde se decidió incluir una columna de nombre *factor*, con el número por el que se multiplica el precio de compra para obtener el precio de venta. Los tipos de venta '*Por mayor*' y '*Por menor*' se definen con un tipo de datos **enum**:

```
create table ganancia(  
  venta enum('Por mayor','Por menor'),  
  factor decimal (2,2)  
);
```

- La lista de precios se define a partir de la empresa proveedor y el producto, asignándole un precio. Por ese motivo, las columnas *empresa* y *clave* se definen como **foreign key**.

```
create table precios (  
  empresa varchar(20) not null,  
  clave int(3) zerofill not null,  
  precio float(6,2),  
  foreign key (empresa) references proveedores,  
  foreign key (clave) references productos  
);
```

## 4.2. Consultar información

MySQL ofrece un conjunto muy amplio de funciones auxiliares (tanto estándares como propias) que nos pueden ayudar mucho en determinados momentos, dejando parte del trabajo de manipular los resultados al propio gestor. Debido al rápido ritmo en el desarrollo de este SGBD, es muy conveniente consultar siempre la documentación de nuestra versión para conocer sus posibilidades concretas. 

En el módulo 3 de este curso ya estudiamos en detalle el Lenguaje SQL, por lo que no vamos a extendernos aquí en su uso y posibilidades. Únicamente mostraremos los aspectos destacables, facilidades o limitaciones que ofrece MySQL respecto a él.

#### 4.2.1. Funciones auxiliares

Las funciones auxiliares que podemos utilizar en nuestras consultas (tanto en la proyección de las columnas como en condiciones en su selección) se pueden clasificar según el tipo de datos con el que trabajan.

##### Ejemplo

```
mysql> select concat(parte,' ',tipo) as producto,
-> psugerido as 'precio sugerido',
-> psugerido + 10 as precio_con_envio
-> from productos;
```

producto	precio sugerido	precio_con_envio
Procesador 2 GHz	NULL	NULL
Procesador 2.4 GHz	35.00	45.00
Procesador 1.7 GHz	205.00	215.00
Procesador 3 GHz	560.00	570.00
RAM 128MB	10.00	20.00
RAM 256MB	35.00	45.00
Disco Duro 80 GB	60.00	70.00
Disco Duro 120 GB	78.00	88.00
Disco Duro 200 GB	110.00	120.00
Disco Duro 40 GB	NULL	NULL
Monitor 1024x876	80.00	90.00
Monitor 1024x876	67.00	77.00

```
12 rows in set (0.00 sec)
```

Algunos ejemplos de las funciones más usadas:

##### Operadores lógicos

Comparación. Aparte de los estándares =, !=, <, >, IS NULL, IS NOT NULL, BETWEEN, IN, destacan COALESCE, INTERVAL, LEAST, GREATEST para trabajar con listas de valores.

##### Control del flujo de ejecución

- CASE .. WHEN .. THEN .. ELSE .. END: Similar a la estructura que crearíamos mediante cualquier lenguaje de programación:

```
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
```

CASE WHEN 1>0 THEN 'true' ELSE 'false' END
true

```
1 row in set (0.00 sec)
```

- IF(expr1,expr2,expr3): Típica estructura condicional, si la expr1 es cierta, devuelve la expr2, en caso contrario, la expr3:

```
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
+-----+
| IF(STRCMP('test','test1'),'no','yes') |
+-----+
| no                                     |
+-----+
1 row in set (0.00 sec)
```

### Funciones para trabajar con cadenas de caracteres (sólo algunos ejemplos)

- CONCAT, INSTR (encontrar en una cadena), SUBSTRING, LCASE/RCASE, LENGTH, REPLACE, TRIM, entre otras, son funciones similares a las que podemos encontrar en lenguajes de programación para manipular cadenas de caracteres.
- QUOTE: delimita una cadena de texto correctamente para evitar problemas al usarla en sentencias SQL. La cadena resultante estará delimitada por comillas simples. Las comillas, el valor ASCII NUL y otros potencialmente conflictivos serán devueltos precedidos del carácter '\'.
- ENCODE/DECODE, CRYPT, COMPRESS/UNCOMPRESS, MD5, etc. son funciones que nos pueden ayudar mucho en el almacenamiento de datos sensibles como contraseñas, etc.

### Funciones numéricas

- Los operadores aritméticos clásicos para realizar todo tipo de operaciones, suma, resta, división, producto, división entera, etc.
- Funciones matemáticas de todo tipo, trigonométricas, logarítmicas, etc.

### Funciones para trabajar con fechas y horas

- Obtención de fechas en cualquier formato: DATE\_FORMAT, DATE, NOW, CURRDATE, etc.
- Manipulación y cálculos con fechas: ADDDATE, ADDTIME, CONVERT\_TZ, DATE\_DIFF, etc.

#### 4.2.2. La sentencia EXPLAIN

MySQL nos ofrece también facilidades a la hora de evaluar las sentencias SQL, gracias a la sentencia EXPLAIN.

Presentamos primero la ejecución de una sentencia SQL más o menos compleja:

```
mysql> select productos.clave, concat(parte,' ',tipo,' ', especificación) as producto, proveedores.em-
presa , precio , pago from productos natural join precios natural join proveedores;
+-----+-----+-----+-----+-----+
| clave | producto | empresa | precio | pago |
+-----+-----+-----+-----+
| 003 | Procesador 1.7 GHz 64 bits | Tecno-k | 198.34 | crédito |
| 005 | RAM 128MB 333 MHz | Tecno-k | 9.27 | crédito |
| 006 | RAM 256MB 400 MHz | Tecno-k | 34.85 | crédito |
| 007 | Disco Duro 80 GB 7200 rpm | Tecno-k | 59.95 | crédito |
| 010 | Disco Duro 40 GB 4200 rpm | Tecno-k | 61.22 | crédito |
| 012 | Monitor 1024x876 60 Hz | Tecno-k | 62.29 | crédito |
| 001 | Procesador 2 GHz 32 bits | Patito | 30.40 | efectivo |
| 002 | Procesador 2.4 GHz 32 bits | Patito | 33.63 | efectivo |
| 003 | Procesador 1.7 GHz 64 bits | Patito | 195.59 | efectivo |
| 005 | RAM 128MB 333 MHz | Patito | 9.78 | efectivo |
| 006 | RAM 256MB 400 MHz | Patito | 32.44 | efectivo |
| 007 | Disco Duro 80 GB 7200 rpm | Patito | 59.99 | efectivo |
| 010 | Disco Duro 40 GB 4200 rpm | Patito | 62.02 | efectivo |
| 001 | Procesador 2 GHz 32 bits | Nacional | 30.82 | crédito,efectivo |
| 002 | Procesador 2.4 GHz 32 bits | Nacional | 32.73 | crédito,efectivo |
| 003 | Procesador 1.7 GHz 64 bits | Nacional | 202.25 | crédito,efectivo |
| 005 | RAM 128MB 333 MHz | Nacional | 9.76 | crédito,efectivo |
| 006 | RAM 256MB 400 MHz | Nacional | 31.52 | crédito,efectivo |
| 007 | Disco Duro 80 GB 7200 rpm | Nacional | 58.41 | crédito,efectivo |
| 010 | Disco Duro 40 GB 4200 rpm | Nacional | 64.38 | crédito,efectivo |
+-----+-----+-----+-----+-----+
20 rows in set (0.00 sec)
```

Ahora utilizamos la sentencia EXPLAIN para que MySQL nos explique cómo ha realizado esta consulta:

```
mysql> explain select productos.clave, concat(parte,' ',tipo,' ', especificación)
as producto, proveedores.empresa , precio , pago from productos natural join
precios natural join proveedores;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| precios | ALL | NULL | NULL | NULL | NULL | 20 | |
| productos | eq_ref | PRIMARY | PRIMARY | 4 | precios.clave | 1 | |
| proveedores | ALL | PRIMARY | NULL | NULL | NULL | 3 | where used |
+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

En cada fila del resultado, nos explica cómo ha utilizado los índices de cada tabla involucrada en la consulta. La columna 'type' nos indica el tipo de "join" que ha podido hacer. En nuestro caso, 'eq\_ref', 'ref' o 'ref\_or\_null' indica que se ha consultado una fila de esta tabla para cada combinación de filas de las otras. Es una buena señal, se están utilizando los índices, tal como indican el resto de columnas (en concreto el atributo 'clave' que es su clave primaria).

Vemos que en las otras dos tablas, el tipo de 'join' es ALL, esto indica que el gestor ha tenido que leer toda la tabla para comprobar las condiciones que le hemos exigido en la consulta. En el caso de la tabla proveedores, habría podido utilizar la clave primaria ('possible\_keys'), pero no lo ha hecho.



Vamos a intentar mejorar esta consulta. Vemos que en la tabla precios no se ha definido ningún índice, lo que facilitaría la labor al SGBD:

```
mysql> alter table precios add index empresa_idx (empresa);
Query OK, 20 rows affected (0.00 sec)
Records: 20 Duplicates: 0 Warnings: 0

mysql> alter table precios add index clave_idx (clave);
Query OK, 20 rows affected (0.00 sec)
Records: 20 Duplicates: 0 Warnings: 0

mysql> explain select productos.clave, concat(parte,' ',tipo,' ', especificación) as producto,
proveedores.empresa , precio , pago from productos natural join precios natural join proveedores;
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| proveedores | ALL | PRIMARY | NULL | NULL | NULL | 3 | |
| precios | ref | empresa_idx,clave_idx | empresa_idx | 20 | productos.emp | 7 | |
| productos | eq_ref | PRIMARY | PRIMARY | 4 | precios.clave | 1 | |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Las cosas han cambiado sustancialmente. El gestor ha pasado de leer 24 filas de datos, a leer 11. También ha cambiado el orden de lectura de las tablas, haciendo primero una lectura total de la tabla proveedores (que es inevitable ya que no hemos puesto ninguna condición en el SELECT) y, después, ha aprovechado los índices definidos en 'precios' y en 'productos'.

Veremos más sobre los índices en el subapartado 5.5 "Análisis y optimización" de esta unidad didáctica.

### 4.3. Manipulación de filas

Para la manipulación de filas disponemos de las sentencias SQL INSERT, UPDATE y DELETE, su uso y sintaxis ya se ha visto en el módulo 3 de este curso. En algunos casos, MySQL nos proporciona extensiones o modificadores que nos pueden ayudar mucho en determinadas situaciones.

- INSERT [DELAYED]. Cuando la sentencia INSERT puede tardar mucho en devolver el resultado (tablas muy grandes o con muchos índices que deben recalcularse al insertar una nueva fila) puede ser interesante añadir la palabra clave DELAYED para que MySQL nos devuelva el control y realice la inserción en segundo plano.
- INSERT [[LOW\_PRIORITY] | [HIGH\_PRIORITY]]. En tablas muy ocupadas, donde muchos clientes realizan consultas constantemente, una inserción lenta puede bloquear al resto de clientes durante un tiempo. Mediante estos modificadores podemos variar este comportamiento.
- INSERT [IGNORE]. Este modificador convierte los errores de inserción en avisos. Por ejemplo, si intentamos insertar una fila que duplica una clave primaria existente, el SGBD nos devolverá un aviso (y no insertará la nueva fila), pero nuestro programa cliente podrá continuar con su cometido si el resultado de la inserción no era importante para su correcta ejecución.

- UPDATE [LOW\_PRIORITY] [IGNORE]. Se comportan de igual modo que en la sentencia INSERT.
- DELETE [QUICK]. Borra el/los registros sin actualizar los índices.
- TRUNCATE. Es una forma muy rápida de borrar todos los registros de una tabla, si no necesitamos saber el número de registros que ha borrado. DELETE FROM <tabla> realiza el mismo cometido, pero devuelve el número de registros borrados.
- LAST\_INSERT\_ID(). Devuelve el último identificador asignado a una columna de tipo AUTO\_INCREMENT después de una sentencia INSERT.