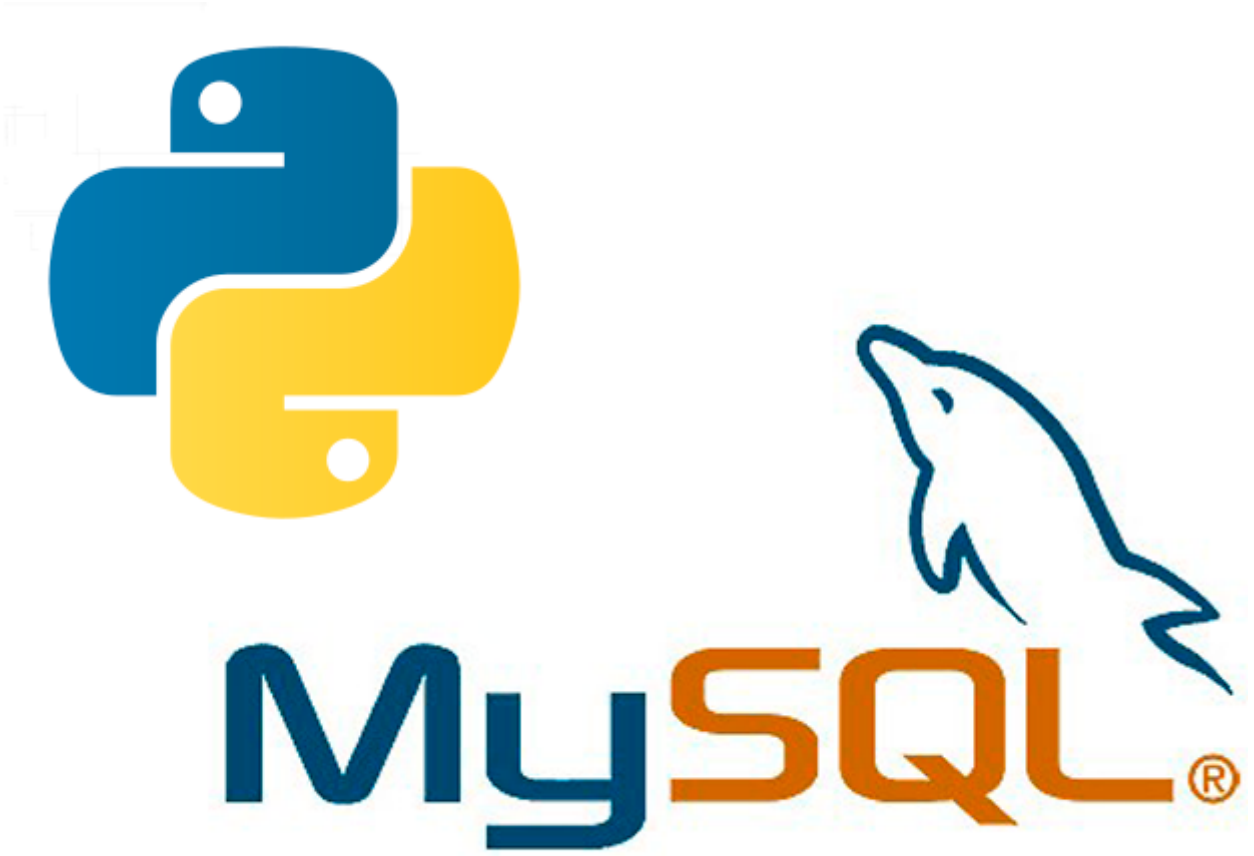
# Bases de Datos con MySQLdb



- 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.

- Fácil de usar

- Tiempo reducido de puesta en marcha

- Disponible bajo licencia GPL

- MySQL es multiplataforma

- En linux usaremos 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.

- La versión de MySQL que utilizaremos sera la 14.14 , versión estable.

# Características de MySQL

- Está desarrollado en C/C++.
- Multiplataforma..
- 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.
- 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.
- 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.

# Instalación MySQL 1/3

**Windows**

- En Windows el modo más simple de instalar MySQL es ir a su página oficial y descargar el instalador automático

  **http://dev.mysql.com/downloads/installer/.**

- Una vez descargado se ha de ejecutar y este irá dirigiendo los pasos del usuario. La instalación no plantea más dificultad que recordar la clave que se use como root pero, si hubiese algún problema, hay una extensa página de ayuda en la instalacion [http://dev.mysql.com/doc/refman/5.0/es/windows-installation.html] en la página oficial de MySQL

# Instalación MySQL 2/3

**Mac-Os**

- Para instalar en sistemas Mac-Os, se pueden seguir los pasos descritos en esta página web:

**http://www.elwebmaster.com/articulos/instalando-mysql-en-mac-os-x**

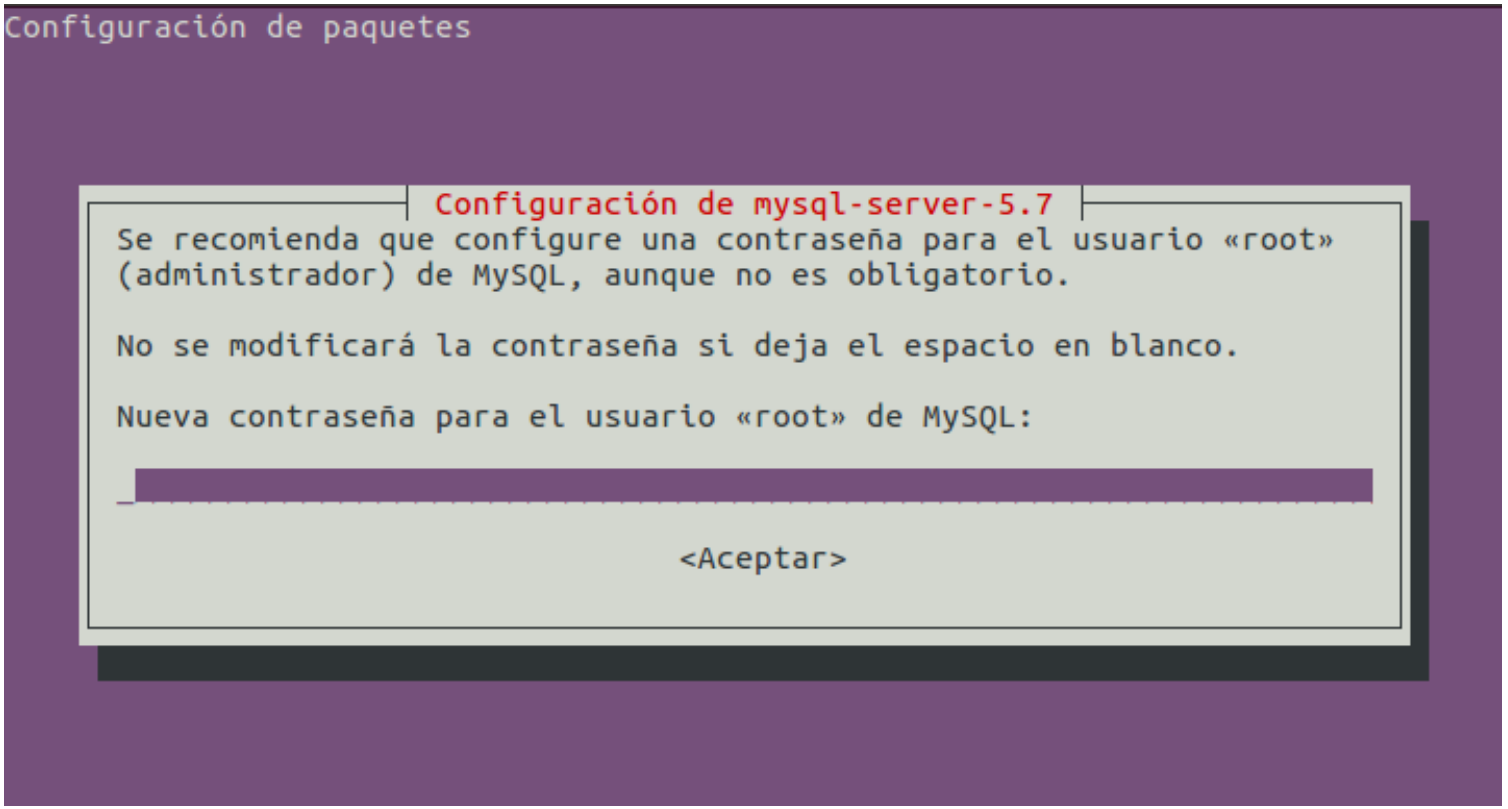
# Instalación MySQL 3/3

**Linux**

- Desde Linux es muy fácil instalar MySQL usando un gestor de paquetes.

- Por ejemplo, en Debian, Ubuntu y derivadas se puede instalar a traves de la línea de comandos usando el siguiendo comando:

**sudo apt-get install mysql-server mysql-client**




Una vez que finalice la instalación, ejecuta el siguiente comando a fin de securizar el servidor MySQL (esta configuración, es válida también, para servidores de producción):

**sudo mysql_secure_installation**

```
Enter current password for root (enter for none):
```

```
Enter current password for root (enter for none):
```

```
Remove anonymous users? [Y/n] Y
 ... Success!
```

```
Disallow root login remotely? [Y/n] Y
 ... Success!
```

```
Remove test database and access to it? (Press y|Y for Yes, any other key for No) : Yes
 - Dropping test database...
Success.

 - Removing privileges on test database...
Success.
```

```
Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : Yes
Success.

All done!
```

# Iniciar, reiniciar y detener el servidor MySQL

Las opciones disponibles son:

**stop** → detiene el servidor

**start** → inicia el servidor

**restart** → reinicia el servidor

- Para iniciar, reiniciar o detener el servidor, se debe ejecutar el siguiente comando seguido de la opción deseada:

  **sudo /etc/init.d/mysql opcion_deseada**

# Conectarse y desconectarse al servidor

Para conectar se debe ejecutar el siguiente comando:

**mysql -u root -p**

- A continuación, hay que ingresar la contraseña de root de MySQL (no es la del root del SO. Es la creada durante la instalación).
- -u → usuario y -p → password .
- Aparecerá un shell interactivo para MySQL:

# Comandos para administrar MySQL desde el shell interactivo

- **show databases;** → Muestra todas las bases de datos creadas en el servidor
- **use nombre_de_la_base_de_datos;** → Indicar la base de datos elegida para trabajar
- **create database nombre_de_la_db;** → Crea una nueva base de datos
- **quit** Salir del shell interactivo

# Importante

- Todas las sentencias introducidas en la shell de mysql deben terminar con “ ; ”.

**show databases;**

# TIPOS DE DATOS

| Tipo de dato | Denominación | Especificaciones | Ejemplo |
|---|---|---|---|
| Entero | **INT(N)** | N = cantidad de dígitos | INT(5) |
| Número decimal | **DECIMAL(N, D)** | N = cantidad de dígitos totales<br>D = cantidad de decimales | DECIMAL(10, 2) |
| Booleano | **BOOL** | | BOOL |
| Fecha | **DATE** | | DATE |
| Fecha y hora | **DATETIME** | | DATETIME |
| Fecha y hora automática | **TIMESTAMP** | | TIMESTAMP |
| Hora | **TIME** | | TIME |
| Año | **YEAR(D)** | D = cantidad de dígitos (2 o 4) | YEAR(4) |
| Cadena de longitud fija | **CHAR(N)** | N = longitud de la cadena – entre 0 y 255 | CHAR(2) |
| Cadena de longitud variable | **VARCHAR(N)** | N = longitud máxima de la cadena – entre 0 y 65532 | VARCHAR(100) |
| Bloque de texto de gran longitud variable | **BLOB** | | BLOB |

# Sintaxis básica de las sentencias SQL

Una sentencia SQL (“query”), es una instrucción escrita en lenguaje SQL.

- 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.

# FORMAS DE HACER CONSULTAS

- select user(), connection_id(), version();

- select user(),
  -> connection_id(),
  -> version();

- select now(); select user();

# 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.

**mysql -u root -h localhost -p < demo.sql**

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

**mysql> source demo.sql**

# demo.sql

```
create database demo;

use demo;

create table productos (nombre VARCHAR (20));

describe productos;

insert into productos (nombre) values ('Impresora');

insert into productos (nombre) values ('Portatil');

insert into productos (nombre) values ('Pantalla');

select * from productos
```

# Registro de operaciones

- Si se desea llevar un registro de todas las operaciones de una sesión, se puede utilizar la expresión siguiente; todos los comandos y sus resultados en ejemplo.txt:

  **mysql> tee ejemplo.txt**

- Para cancelar la captura, basta con teclear lo siguiente:

  **mysql> notee**

# CREAR BASE DATOS

- **CREATE DATABASE “NombreBaseDatos”**

- **CREATE DATABASE Tienda;**

  **CREATE DATABASE** → Crear base datos

  **Tienda** → Nombre Base Datos

# BORRAR BASE DATOS

- **DROP DATABASE “nombre base datos”;**

- **DROP DATABASE Tienda;**

# USAR BASE DE DATOS

- **USE “Nombre base datos”;**

- USE Tienda;

# VER BASES DE DATOS CREADAS

- **SHOW DATABASES;**

# SABER QUE BASE DE DATOS ESTAMOS USANDO

- **SELECT DATABASE ();**

# USUARIO Y ACCESO

- **GRANT ALL ON Tienda.* TO 'oso'@'localhost' IDENTIFIED BY 'panda';**

- Lo que hacemos es crear un usuario "oso" que tenga la contraseña "panda" y que tenga todos los permisos en la base de datos Tienda.

# Crear tablas en una base de datos

**CREATE TABLE;**

**CREATE TABLE nombre_tabla(nombre_campo TIPO_DE_DATO, nombre_de_otro_campo TIPO_DE_DATO);**

CREATE TABLE articulos(articulo VARCHAR(100), descripcion BLOB, precio DECIMAL(6, 2), stock BOOL);

- **CREATE TABLE articulos**

  Crear una nueva tabla llamada “articulos”
- **articulo VARCHAR(100),**

  Crear un campo llamado articulo, de tipo cadena de texto de longitud variable, con una longitud máxima de 100 caracteres
- **descripcion BLOB,**

  Crear un campo llamado descripción, de tipo bloque de texto de gran longitud
- **precio DECIMAL(6, 2),**

  Crear un campo precio de tipo numérico de longitud máxima de 6 digítos de los cuales, solo 2 pueden ser decimales
- **stock BOOL**

  Crear un campo llamado “stock” del tipo booleano

# Insertar datos en una tabla

**INSERT INTO nombre_de_la_tabla(campo1, campo2, campo10..) VALUES(dato1, dato2, dato10...);**

**INSERT INTO articulos(articulo, precio, stock) VALUES(“pelota”, 19.95, TRUE);**

- **INSERT INTO articulos (articulo, precio, stock)**

  Insertar un nuevo registro en los campos producto, precio y stock de la tabla productos

- **VALUES(“pelota”, 19,95, TRUE);**

  Con los valores “pelota”, 19,95 y verdadero, respectivamente en cada uno de los campos indicados

# VER CONTENIDO TABLA

mysql> describe articulos;

# Seleccionar elementos de una tabla

- **SELECT campo1, campo2, campo10 from tabla;**

- SELECT  articulo precio from articulos;

- **SELECT articulo, precio**

  Seleccionar los campos articulo y precio

- **FROM articulos;**

  De la tabla articulos

# Modificar registros

- **UPDATE  tabla SET campo1 = valor, campo2 = valor, campo10 = valor;**

- UPDATE articulos SET precio = 0;

- **UPDATE articulos**

  Actualizar la tabla productos

- **SET precio = 0;**

  y el campo precio lo cambiamos a 0

# Eliminar registros

- **DELETE FROM tabla;**
- DELETE FROM articulos;

- **DELETE FROM articulos;**

  Eliminar todos los registros de la tabla articulos

# Consultas avanzadas

**La cláusula WHERE**

- Las sentencias en SQL, se componen de cláusulas. Y WHERE es una de ellas. La cláusula WHERE nos permite filtrar registros en una sentencia SQL.
- Esta cláusula, funciona de forma similar a la comparación de expresiones en Python, utilizando los siguientes operadores de comparación:

```
>         mayor que                  <         menor que
=         igual que                  <>        distinto que
>=        mayor o igual que          <=        menor o igual que

BETWEEN n1 AND n2              entre n1 y n2
IS NULL|TRUE|FALSE             es nulo | es verdadero | es falso
IN(valor1, valor2, va...)      contiene
```

```
AND (y)          NOT (negación)          OR (o)
```

- Seleccionar productos donde precio sea menor que 10:

  **SELECT articulo, precio FROM articulo WHERE precio <= 10;**

- Aumentar el 10% del precio de los artículos, que esten entre 1 y 20:

  **UPDATE articulos SET precio = (precio * 1.10) WHERE precio BETWEEN 1 AND 20;**

- **Seleccionar articulo donde stock no sea falso**

  SELECT articulo FROM articulos WHERE stock IS NOT FALSE;

- **Eliminar productos cuyos precios sean 10, 20 y/o 30 y además, stock sea falso o articulo sea nulo:**

  DELETE articulo FROM articulos WHERE precio IN (10,20,30) AND (stock IS FALSE OR articulo IS NULL);

- **Modificar stock a verdadero donde precio sea menor que 50 y producto no sea nulo:**

  UPDATE articulo SET stock = TRUE WHERE precio <50 and stock IS NOT NULL;

# Ordenando consultas: la cláusula ORDER BY

- Es posible además, ordenar los resultados de una consulta, en forma ascendente (ASC) o descendente (DESC):

  SELECT articulo, precio FROM articulos **ORDER BY** precio DESC;

También es posible, ordenar los resultados de la consulta, por más de un campo:

- SELECT articulo, precio FROM articulos ORDER BY precio DESC, articulo ASC;

# Bases de datos con Python

- En Python, el acceso a bases de datos se encuentra definido a modo de estándar en las especificaciones de DB-API,:

  http://www.python.org/dev/peps/pep-0249/.

- Esto, significa que independientemente de la base de datos que utilicemos, los **métodos** y **procesos** de **conexión**, **lectura** y **escritura** de datos, desde Python, siempre serán los mismos, más allá del conector.

# Libreria MYSQLdb

sudo apt-get install python-mysqldb

# Instalar MySQLlib

MySQLlib es una librería que gestiona el acceso a MySQL desde Python. Es la que se ocupa de que Python y MySQL “se entiendan”, y debe estar instalada para poder usar MySQL.

- Método básico (Linux y Windows) :El procedimiento para instalar es el mismo que para cualquier otro módulo:

  http://sourceforge.net/projects/mysql-python/

- El archivo que se obtiene en su página de sourceforge es un archivo comprimido (con extensión *.tar.gz). Se debe descomprimir en cualquier directorio (es sólo temporal, luego se podrá borrar) y después, en ese mismo directorio, ejecutar la siguiente orden:

  **python setup.py install**

- Necesitarás permisos de superusuario

  **sudo python setup.py install**

**apt-get install python-mysqldb**

- Binarios precompilados (Windows)

  Algunas versiones de Windows dan errores durante la instalación, a causa de un problema en el registro. Para ellas existen una serie de binarios precompilados (con extensión *.exe) que sólo hay que descargar y ejecutar para que se instalen.
- Se pueden descargar de la siguiente dirección: **http://www.codegood.com/**

- Como segunda opción, también puede ser interesante conocer un repositorio no oficial de librerías de Python para Windows:

  **http://www.lfd.uci.edu/~gohlke/pythonlibs/**

- En esta página sólo tienes que buscar el módulo que te interesa, descargar la versión adecuada para tu arquitectura, y ejecutar el instalador.

# Conectando a MySQLdb

- Para empezar, como con cualquier otro módulo, hay que importarlo:

  **import MySQLdb**

- Lo siguiente es establecer una conexión

  **Conexion = MySQLdb.connect(host='localhost', user='oso',passwd='panda', db='Tienda')**

Dado que una base de datos puede ser servida desde un ordenador remoto a través de una red, la opción "**host**" nos da la oportunidad de indicar aquí la dirección de esa máquina. Normalmente (y este va a ser nuestro caso), se conectará con un servidor de bases de datos ubicada en el mismo ordenador, lo que indicamos con "**localhost**".

- En "**user**" indicamos nombre de usuario con el que conectaremos a la base de datos (y cuyos permisos serán los que tenga nuestra conexión) y, en "**passwd**" indicamos la contraseña de este usuario.

  Por último, dado que un servidor de bases de datos puede servir varias de ellas, tenemos que indicar el nombre de la base a la que conectamos, lo que hacemos en "**db**".

- La función "**connect**" nos devuelve un objeto "**Connection**" que recogeremos en la variable "**Conexion**" para poder referenciarla luego.

Algunas configuraciones de bases de datos, principalmente para optimizar recursos, usan lo que se llama "transacciones" (particularmente, las bases de datos del tipo InnoDB permiten transacciones).

Significa que pueden ir acumulando las órdenes que se les dan y ejecutarlas de una sola vez como un paquete.

- Lo bueno es que se pueden hacer cosas como cancelar peticiones o grupos de ellas en función de otras entradas posteriores y otras cosas parecidas.
- Lo malo de esto es que puede darse el caso de que creamos haber insertado datos y estos no estén aún allí.

  Afortunadamente, tenemos una orden para obligar a la base de datos a ejecutar las órdenes que le hemos mandado.

- Se trata del método **commit** de la clase **Connection** que, si la base de datos admite transacciones, hace ese envío de los datos, de este modo:

  **Conexion.commit()**

- En caso de que la base de datos no admita transacciones (con lo que las órdenes se ejecutan sobre la marcha), este método no hace nada, lo cual hace que podamos usarlo "preventivamente" aún sin saber qué tipo de base de datos estamos usando, y si usa transacciones.

# Ejecutar sentencias SQL

Una sentencia puede retornar un montón de campos de muchos registros. Para poder manipular estos, usamos una estructura intermedia llamada "Cursor". El cursor recoge los datos devueltos por la base de datos y los almacena de modo que puedan ser recogidos por nuestro programa.

- Usamos el método "cursor" del objeto "Connection" de este modo:

**micursor = Conexion.cursor()**

- Con esto creamos un objeto Cursor que, esta vez sí, ya podemos usar para enviar órdenes a la base de datos y recoger sus resultados.Por ejemplo, es hora de introducir algún dato, usando el método execute:

- **micursor.execute("INSERT INTO articulos (articulo, precio, descripcion, precio, stock) VALUES (teclado, \"Teclado para teclear\",10, TRUE;")**

- En realidad, es mucho más práctico guardar la sentencia SQL en una variable y usarla luego en el cursor, de un modo similar a este:

  **query= "INSERT INTO articulos (articulo, precio, descripcion, precio, stock) VALUES (pantalla, \"Pantalla para pantallear\",120, TRUE;");"**

  **micursor.execute(query)**

- Del mismo modo, podemos usarlo para leer registros. Por ejemplo:

  **query= "SELECT articulo FROM articulos WHERE precio=120;"**

  **micursor.execute(query)**

- Esto recogerá el campo articulo de los registros que coincidan con la clausula WHERE (en nuestro caso, uno solo). Pero ¿Cómo accedemos a ellos?

- Para ello tenemos fetchone, que sitúa un puntero en la lista de registros y nos retorna el primero de ellos, pasando dicho puntero al siguiente, y así sucesivamente mientras queden registros.

  **registro= micursor.fetchone()**

- De este modo, cada vez que llamemos a fetchone nos retornara un registro de la tabla. En caso de que no queden registros, fetchone retorna un valor falso, con lo que es trivial usarlo dentro de una estructura while o similar para ir tomando registros mientras existan.

- En caso de que sea necesario, el número de registros obtenidos (o insertados, si se trataba de un INSERT, o borrados, si era un DELETE...) puede obtenerse con la propiedad rowcount de la clase cursor, de este modo:

  **numero_de_registros= micursor.rowcount**

# EJEMPLO 1

# EJEMPLO 1_PARTE 2

# Fetchall

- Cuando se van a recibir unos pocos registros fetchone cumple su labor perfectamente pero, si se van a recibir muchos registros, es mejor cargar estos de una sola vez en una estructura más manejable y manipularlos por nosotros mismos.

  **fetchall** se comporta como el anterior, pero, en lugar de retornar una tupla conteniendo un registro, lo que retorna es un array de ellas con todos los registros retornados.

- Naturalmente, esto puede ser una locura en bases de datos grandes, en las que es más que normal recibir cientos de miles de registros en una petición.

- Una solución intermedia a recogerlos uno a uno (fetchone) o recogerlos todos (fetchall) es recogerlos por grupos. Para ello tenemos fetchmany.

**fetchmany** admite un parámetro numérico que indica cuántos registros recogerá, de este modo:

**registros= micursor.fetchmany(10)**

- En este ejemplo, el método retorna un array de tuplas (igual que hacía **fetchall**), pero con un **límite máximo de 10** (que es el número que le hemos indicado). Además, al igual que vimos que hacía fetchone, mantiene un **puntero interno** que indica por qué registro va, de modo que, en la siguiente llamada, seguirá por el registro siguiente al último que hubiese retornado antes.

# EJEMPLO 2

# DictCursor

- Las tres formas de acceder a los datos que hemos visto nos retornan los propios datos, pero no nos dan información de a qué campo se corresponde cada uno de ellos. En principio esto no es necesario porque, después de todo, se los estamos pidiendo nosotros en el **SELECT**. Pero sí que habrá muchas circunstancias en las que nos interese conservar esa información de a qué campo se corresponde cada uno de los datos recibidos.

- Afortunadamente tenemos **DictCursor**, una herramienta que nos permitirá precisamente eso:

  **micursor = Conexion.cursor(MySQLdb.cursors.DictCursor)**

- **DictCursor** es, en realidad, una **subclase** de cursor y, por tanto, se usa y se comporta como vimos que hacía Cursor, con la diferencia de que, en lugar de retornar una tupla de datos retorna un diccionario.

# EJEMPLO 3

# SQLITE3 Y PYTHON

- **http://www.pythondiario.com/2013/12/python-y-sqlite3-como-base-de-datos.html?m=1**