Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Bases de datos empotradas (I) **(28315 lectures)**
Per **Antoni Aloy López**, ***aaloy*** (http://trespams.com)
Creado el 30/08/2002 20:42 modificado el 30/08/2002 20:42

Entendemos por **bases de datos empotradas** aquellas que no inician un servicio en nuestra máquina independiente de la aplicación, pudiendose enlazar directamente a nuestro código fuente o bien utilizarse en forma de librería.
También podemos definirlas a partir de lo que no son: servidores de bases de datos.Así no son bases de datos empotradas Postgresql, mySQL, Firebird, Oracle,...

Bases de datos empotradas (I)

Entendemos por **bases de datos empotradas** aquellas que no inician un servicio en nuestra máquina independiente de la aplicación, pudiendose enlazar directamente a nuestro código fuente o bien utilizarse en forma de librería.
También podemos definirlas a partir de lo que no son: servidores de bases de datos.Así no son bases de datos empotradas Postgresql, mySQL, Firebird, Oracle,...

Normalmente las bases de datos empotradas comparten una serie de características comúnes: su **pequeño tamaño**, el "small footprint" en términos de tamaño de código añadido a nuestra aplicación y recursos que consumen, y que en general no están pensadas para el acceso multi usuario.

En algunos casos las bases de datos empotradas representan el primer paso hacia un servidor de bases de datos tradicional (mySQL utiliza una de estas bases de datos), en otros casos simplemente su pequeño tamaño las hace ideales como sistema de soporte de información en sistemas monousuario que deba utilizar los recursos de la forma más eficiente posible, o bien como soporte a nuestros scripts de gestión de sistemas.

Buceando un poco por la red podemos encontrar varios ejemplos de este tipo de base de datos:

- SQLite(1)
- Gadfly(2)
- Berkeley DB(3)
- Metakit(4)

La elección de estos ejemplos no es aleatoria: representan algunos de los ejemplos más populares y comparten la característica de que **existe un API para Python** y/o los lenguajes script más populares, lo que nos permitirá probarlas y utilizarlas con un mínimo de esfuerzo. Las dos primeras son bases de datos que intentan en mayor o menor medida ser compatibles con SQL 92, las dos últimas representan báse de datos empotradas clásicas, con muchos años de historia a sus espaldas y que se han utilizado y se utilizan en algunas de las aplicaciones unix más comunes.

En este artículo intentaré daros a conocer la primera, dejando las otras y una comparativa de rendimiento para posteriores artículos.

SQLite(1)

SQLite es una librería escrita en C que implementa un motor de base de datos SQL empotrable. Sus desarrolladores destacan entre sus principales caracteristicas su completo soporte de tablas e índices en un único archivo por base de datos, soporte **transaccional**, su rapidez y su escaso **tañaño** (unas 25 mil líneas de código C) y su completa **portabilidad**.

Su instalación es muy sencilla, requiere únicamente los pasos habituales de tar, configure, make y make install. Los desarrolladores recomiendan estos pasos:

```
$ tar xzf sqlite.tar.gz
$ mkdir bld
$ cd bld
$ ../sqlite/configure
$ make
$ make test
```

Existen ports de la API para distintos lenguajes, incluso un proyecto de ODBC para SQLite[5] La API para Python la podemos encontrar en http://pysqlite.sourceforge.net/[6]

En web de SQLite[1] aparece un enlace hacia una comparativa de velocidad [7]de esta base de datos. Aquí podemos ver que sale muy bien parada trabajando en modo asíncrono en relación a mySQL o Postgresql, pero no olvidemos que se trata de motores de bases de datos para propósitos distintos. SQLite no permite múltipes usuarios accediendo en modo escritura a la base de datos, ya que el mecanismo de bloqueo que utiliza es muy "basto": **bloquea toda la base de datos**. Así esta librería está especialmente indicada cuando queramos una **gran rapidez en las consultas** y nos baste que sólo un único usuario pueda realizar modificaciones. Por ejemplo es ideal como base de datos para guardar configuraciones, logs de scripts, o sencillamente como base de datos monousuario.

Como característica adicional que debe tenerse también en cuenta es la falta de tipos de datos definidos en una tabla SQLite:por defecto todos los datos se almacenan en como cadenas de carácteres acabadas en nulo (típico de algo hecho en C) a excepción de las columnas definidas como INTEGER PRIMARY KEY que se almacenan como enteros y hacen las veces de campo autoincremental. Podemos suplir esta carencia mediante la directiva *pysqlite_pragma expected_types* si utilizamos la librería pysqlite[6]

SQLite cuenta con una utilidad llamada **sqlite** que nos permitirá ejecutar comandos SQL contra una base de datos SQLite. Desde aquí podemos crear nuestra base de datos, realizar consultas, insertar datos, etc. Además es muy sencillo utilizar este programa en un shell script, ya que sqlite permite ejecutar un comando SQL directamente desde la línea de comando, así basta ejecutar sqlite junto con el nombre de la base de datos a la que queramos acceder y la sentencia SQL que queramos ejecutar.

Además de su soporte transaccional, su rapidez, su facilidad de instalación, una de las características que más me ha sorprendido de esta base de datos es la manera tan sencilla como podemos extender su funcionalidad. Permite definir funciones que operarán sobre los datos devueltos como si se tratasen de sentencias sql, toda la potencia del C o del Python al servicio de una base de datos!

Por ejemplo, supongamos que queremos crear una base de datos con los ordenadores de nuestra red, que por cierto se supone que son mucho más de los que yo doy como ejemplo. Podemos crear la base de datos y la tabla directamente con **sqlite** pero es mucho más práctico crear la estructura y la los datos a introducir en un fichero de texto. Así, he creado un fichero llamado db.sql con el siguiente contenido

```
CREATE TABLE MAQUINAS (
        ID      INTEGER UNIKE PRIMARY KEY,
        NOMBRE  VARCHAR(30),
        IP      VARCHAR(15),
        USUARIO VARCHAR(30)
        );
COMMIT;

INSERT INTO MAQUINAS (NOMBRE, IP, USUARIO)
        VALUES ('DELL','192.168.1.10','TONI');
INSERT INTO MAQUINAS (NOMBRE, IP, USUARIO)
        VALUES ('HP','192.168.0.1', 'PEP');
INSERT INTO MAQUINAS (NOMBRE, IP, USUARIO)
        VALUES ('ROUTER','192.168.0.100','ROUTER');
COMMIT;
```

Y he ejecutado simplemente

```
$ sqlite test.db <db.sql
```

Veamos como podemos generar un sencillo archivo html con los datos de nuestra red en forma de tabla utilzando sqlite y awk (por cuestiones de claridad he troceado las líneas)

```
$ sqlite test.db "select * from maquinas" | awk
        'BEGIN {FS="|";
        print "&lt table&gt&lt b&gt &lt tr&gt &lt td&gt ORDENADOR&lt /td&gt &lt td&gt
        &lt CENTER &gt IP&lt/CENTER &gt &lt/td&gt &lt/tr&gt &lt/b&gt ";
        {print "&lt td&gt",$2,"&lt/td&gt ","&lt td&gt ",$3,"&lt /td&gt &lt/tr&gt ";
        END{print "&lt/table&gt "}' &gt  prova.html
```

Para nuestros datos produce la salida

ORDENADOR	IP
DELL	192.168.1.10
HP	192.168.0.1
ROUTER	192.168.0.100

Aquí el truco está en decirle a awk que utilice el mismo separador de campos que sqlite, en nuestro caso el carácter |

Veamos ahora cómo podemos acceder a esta base de datos en Python y emplear alguna de las características propias de sqlite.Primero **lo más básico**, la conexión a la base de datos, inserción de algún registro y selección

```
#!/usr/bin/python
import sqlite

conn = sqlite.connect(db='test.db', mode=077)
cursor = conn.cursor()


# Con esto ya tenemos la conexión preparada para escritura
# Ahora insertaremos una nueva máquina

SQL = """insert into maquinas (nombre, ip, usuario)
         values ('VECTRA','192.168.1.27','JJC')"""
cursor.execute(SQL)
conn.commit()
#Ojo! Conviene no olvidar el commit

# Veamos cómo queda la tabla
# La selección devuelve un diccionario, por lo que podemos acceder
# a los resultados utilizando el nombre del campo

SQL = "select NOMBRE, IP, USUARIO from maquinas order by nombre"
cursor.execute(SQL)
row = cursor.fetchone()
plantilla = "%30s %15s %30s \n"
print plantilla % ('MAQUINA','IP','USUARIO')

while row != None:
        print plantilla % (row['NOMBRE'],
                           row['IP'],
                           row['USUARIO'])
        # I ahora a por el siguiente registro
        row = cursor.fetchone()
# Cerramos la conexión
conn.close()
```

I ahora vayamos a por algo más sofisticado, vamos a crear una función que utilizaremos para saber si esta máquina está en el rango 192.160.1.x o en el rango 192.168.0.x

```
#!/usr/bin/python
import sqlite
from string import split

conn = sqlite.connect(db='test.db', mode=077)
```

```
# Creamos la función que nos dará el rango deseado
def get_rango(ip):
        try:
                return split(ip,'.')[2]
        except:
                return "npi"

# Ahora signamos la función a nuestra conexión

conn.create_function("rango",1,get_rango)
cursor = conn.cursor()

cursor.execute("
        select NOMBRE, IP, USUARIO, rango(IP) RED from maquinas")
row = cursor.fetchone()
plantilla = "%20s %15s %20s %5s \n"
print plantilla % ('MAQUINA','IP','USUARIO','RANGO')

while row != None:
        print plantilla % (row['NOMBRE'],
                           row['IP'],
                           row['USUARIO'],
                           row['RED'])
        # I ahora a por el siguiente registro
        row = cursor.fetchone()
# Cerramos la conexión
conn.close()
```

Que al ejecutarse y con nuestro datos, y alguno más, nos da la salida

```
MAQUINA               IP      USUARIO RANGO
   DELL    192.168.1.10         TONI     1
     HP     192.168.0.1          PEP     0
 ROUTER   192.168.0.100       ROUTER     0
    AMD    192.168.1.25    ESPARTACO     1
   AMD2    192.168.1.26     ESPARTAC     1
```

Como podéis comprobar sqllite junto con la librería de python nos da una gran facilidad de manipulación de nuestros datos. En nuestros ejemplos sólo hemos trabajado sobre una tabla, pero como toda base de datos sql que se precie sql lite permite crear varias tablas y establecer relaciones entre ellas utilizando toda la potencia del un lenguaje sql que intenta hacercarse al máximo al estandard al tiempo que se intenta mantener la rapidez y simplicidad de la base de datos.

Lista de enlaces de este artículo:

1. http://www.hwaci.com/sw/sqlite/
2. http://gadfly.sourceforge.net/
3. http://www.sleepycat.com/
4. http://www.equi4.com/metakit/
5. http://www.ch-werner.de/sqliteodbc/
6. http://pysqlite.sourceforge.net/
7. http://www.hwaci.com/sw/sqlite/speed.html

E-mail del autor: aaloy _ARROBA_ bulma.net
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1472