Pruebas de Sqlite en un sistema Linux
guia para iniciarse

1.Generalidades

Sqlite es una base de datos muy similar a la conocida Access del mundo Windows pero a diferencia de esta posee una serie de ventajas que la hacen interesante de aplicar. Para comenzar es multiplataforma y cumple con los estándares (en su mayoría) SQL92 por lo que su sintaxis y forma de uso casi no posee curva de aprendizaje a los conocedores de SQL y sus MySql, porque además como este ultimo (implementación de mysql en php) sqlite también esta contemplada en el tratamiento dinámico de php profusamente.

Sqlite se puede usar en modo ventana de comandos (Shell) o embebido en aplicaciones de codigo (casi cualquier lenguaje de programación directamente o indirectamente) por ejemplo c, c++, bash etc. En aplicaciones se puede utilizar con OpenOffice mediante la aplicación de los drivers ODBC adecuados. Sqlite es un motor que trabaja embebido con la aplicación que lo use (en el servidor, en programas escritos por nosotros, etc) y otra de sus ventajas es la portabilidad.

Sqlite es ideal para trabajar con volúmenes medianos o pequeños de información, de manera ágil y eficiente. Aunque sus diseñadores aducen que es posible manejar bases de datos de 2 terabytes sin mayores inconvenientes.
Cuando un Mysql nos queda grande la solución ideal de código abierto, libre y gratuito es Sqlite. Este se descarga de su sitio en internet http://www.sqlite.org.

Recordemos que para instalar sqlite es necesario estar corriendo el servidor local de la máquina (Localhost) tanto en linux como en windows, pues hay sqlite para este sistema operativo también, y cuya instalación es mas sencilla porque es un ejecutable que se descarga de la misma pagina que el anterior.

Una de las ventajas de Sqlite es que permite trabajar en dos modalidades: carga en memoria o en disco, pudiendo pasar la base de uno a otro estado con un par de comandos. La modalidad de cargar en memoria brinda rapidez adicional

2. Instalacion en el shell de linux

La operación es sencilla, una vez ubicado el archivo en el sitio de descargas (en nuestro caso es sqlite-3.3.8.tar.gz) lo descomprimimos y nos queda la carpeta “sqlite3.3.8” Allí dentro usando los privilegios de root lo creamos e instalamos según esta secuencia que reproducimos a continuación:

por ejemplo:

```
tar xzf sqlite.tar.gz    ;# descomprimimos en una carpeta
mkdir bld                ;# creamos un directorio dentro de
“sqlite” para trabajar
cd bld                   ;# cargamos el directorio creado
../sqlite/configure      ;# corremos el script para configurar
```

```
                              Sqlite a nuestro sistema
make                        ;#  creamos la instalacion
make install                ;#  la instalamos en el sistema.
```

Esta instalación colocara el motor de la base de datos dentro de los comandos del bash (ventana o shell) que nos permitirá llamarla invocando (en mi caso)

```
[edwin@localhost ~]$ sqlite3
SQLite version 3.3.8
Enter ".help" for instructions
sqlite>
```

Demostrando que la base corre y esta pronta a recibir nuestras instrucciones.

3. Manejando algunas generalidades

Una de la primeras cosas que suelen enloquecer a quienes venimos de MySql es el uso de los comandos con el punto delante= ".help" Esto que es una trivialidad es razoné para infinidad de errores al inicio, púes al no incluir este punto suelen saltar errores y lo primero que pensamos es "¿en donde esta la falla?"

Asi que para los comandos principales no olvidemos el punto delante. Para las sentencias de uso de la base la sintaxis es similar a la de sql y Mysql sin mayores complicaciones, incluyendo el ";" al final de cada sentencia. Si no lo hacemos el Sqlite nos dira -al igual que mysql- que "....>" falta cerrarla.

4. Ejemplo de entrada

Para entrar y curiosear es como vimos anteriormente. Para acceder al menú de ayudas debes colocar ".help" y para salir al prompt del sistema ".exit":

```
[edwin@localhost pruebas_sqlite]$ sqlite3
SQLite version 3.3.8
Enter ".help" for instructions
sqlite> .help
.databases             List names and files of attached databases
.dump ?TABLE? ...      Dump the database in an SQL text format
.echo ON|OFF           Turn command echo on or off
.exit                  Exit this program
.explain ON|OFF        Turn output mode suitable for EXPLAIN on or off.
.header(s) ON|OFF      Turn display of headers on or off
.help                  Show this message
.import FILE TABLE     Import data from FILE into TABLE
.indices TABLE         Show names of all indices on TABLE
.load FILE ?ENTRY?     Load an extension library
.mode MODE ?TABLE?     Set output mode where MODE is one of:
                         csv      Comma-separated values
                         column   Left-aligned columns.  (See .width)
                         html     HTML <table> code
                         insert   SQL insert statements for TABLE
                         line     One value per line
                         list     Values delimited by .separator string
                         tabs     Tab-separated values
                         tcl      TCL list elements
```

```
.nullvalue STRING      Print STRING in place of NULL values
.output FILENAME       Send output to FILENAME
.output stdout         Send output to the screen
.prompt MAIN CONTINUE  Replace the standard prompts
.quit                  Exit this program
.read FILENAME         Execute SQL in FILENAME
.schema ?TABLE?        Show the CREATE statements
.separator STRING      Change separator used by output mode and .import
.show                  Show the current values for various settings
.tables ?PATTERN?      List names of tables matching a LIKE pattern
.timeout MS            Try opening locked tables for MS milliseconds
.width NUM NUM ...     Set column widths for "column" mode
sqlite>
```

aquí hemos llamado a la ayuda y nos ha dado el menú explicativo. Veamos ahora como listar las bases de datos existentes, tablas y columnas. Un detalle importante es que debemos estar en el directorio de trabajo que contenga las bases cuando abramos el shell (ventana de comandos) en nuestro caso el clasico Bash invocando la base de datos como "sqlite3 base.db":

```
sqlite> .database
seq  name             file
---  ---------------  ----------------------------------------
0    main             /home/edwin/pruebas_sqlite/base.db
sqlite>
```

Ahota las tablas dentro de esta base de datos "base.db" y las columnas:

```
sqlite> .tables
Datos
sqlite> .schema Datos
CREATE TABLE Datos (id int(9) primary key, apellido chars(30), nombre chars(30),
dni inst (12), observaciones chars(255));
sqlite>
```

5. Creamos la primara base de datos

Para hacerlo simplemente escribimos en la ventana de comandos el nombre de la base que vamos a crear, si existe esta se abrira para que trabajemos en ella. Por ejemplo escribimos "sqlite3 base.db" que no tenemos aun. Se creara la base y luego procederemos a cargar las tablas y las columnas mediante este comando *"CREATE TABLE [nombre de la tabla] (columna1, columna2, columna etc);"* Como en este caso que hemos creado la tabla "Datos":

```
sqlite> CREATE TABLE Datos (
   ...> id int(3) not null,
   ...> Apellido char(30) not null,
   ...> Nombre char(30) not null,
   ...> Documentos char(30) not null,
   ...> Observaciones chat(60) not null,
   ...> primary key (id));
```

Lo cual una vez ingresado no nos da error, lo que significa que salio bien, cosa que confirmamos con el comando *.schema [tabla]* :

```
sqlite> .schema Datos
CREATE TABLE Datos (
id int(3) not null,
Apellido char(30) not null,
Nombre char(30) not null,
Documentos char(30) not null,
Observaciones chat(60) not null,
primary key (id));
```

Podriamos haber creado la tabla "Datos" de varias maneras, sin indicar el tipo de datos (int= integer o numeros enteros) caracteristicas ("not null" que significa que deben ser completadas) o la clave primaria que es quien crea el autoincremento cuando se cargan nuevas filas de datos. Pero es costumbre adquirida hacerlo de esta manera, por mas que posteriormente se puedan modificar.

Para eliminar una tabla que este mal o nos agrade se usa la sentencia *"drop table",* en este ejemplo teniamos dos tablas Datos y Otra:

```
sqlite> drop table Otra;
sqlite> .tables
Datos
```

La salida indica que solo queda Datos. Otra fue eliminada.

6. Cargamos las primeras filas de datos

Para hacerlo usamos la conocida sentencia *"insert into [Tabla] (campos) values ('valores')"* como vemos en este modelo:

```
sqlite> insert into Datos (id,Apellido, Nombre, Documentos, Observaciones)
values ('1','Aguiar','Edwin','Si, tiene','Ninguna que destacar');
sqlite> select * from Datos;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
```

Cargamos la primera fila y comprobamos con *"select * from"* en la tabla "Datos" la salida. Como vemos da bien y seguimos cargando algunos registros mas, Como es notorio la sintaxis de sql y mysql facilita las cosas para los que ingresen al sqlite. Y para lso novatos que prueban por primera vez es sencillo de comprender ya que este lenguaje es muy racional y claro. Despues de un rato de cargar datos tenemos que la base ya esta un poco mas amplia:

```
sqlite> select * from Datos;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|hja de siete meses
```

Esta salida es un poco horrenda, para los que venimos de otras bases y estamos acostumbrados a una presentación mejor, en sqlite esta es la que provee por defecto pero se puede mejorar con los comandos ".*mode MODE [TABLE]*" de esta manera:

```
sqlite> .mode tabs Datos
```

```
sqlite> select * from Datos;
1       Aguiar  Edwin   Si, tiene       Ninguna que destacar
2       Esparza Silvina Tambien posee   Esposa
3       Aguiar Esparza  Sabina  recien obtenido hja de siete meses
```

Con lo cual ha mejorado un poco. Existen varios modos de configurar esta salida, descriptos en ".help" El ancho de columnas también es predeterminado por "*.width NUM NUM ...*"

Una cosa que puede ser muy útil es la posibilidad de hacer DUMP de las tablas (crear un archivo sql) que luego sea importable:

```
BEGIN TRANSACTION;
CREATE TABLE Datos (
id int(3) not null,
Apellido char(30) not null,
Nombre char(30) not null,
Documentos char(30) not null,
Observaciones chat(60) not null,
primary key (id));
INSERT INTO "Datos" VALUES(1, 'Aguiar', 'Edwin', 'Si, tiene', 'Ninguna que
destacar');
INSERT INTO "Datos" VALUES(2, 'Esparza', 'Silvina', 'Tambien posee', 'Esposa');
INSERT INTO "Datos" VALUES(3, 'Aguiar Esparza', 'Sabina', 'recien obtenido',
'Hija, bebe de 7 meses');
COMMIT;
```

7.Modificacion de Datos

Uno de los problemas a la hora de manejar bases de datos es que suele ser necesario cambiarlos. Para eso usamos la sentencia *"UPDATE [tabla] SET columna="dato nuevo" WHERE columna="dato viejo";* Por ejemplo de esta manera:

```
sqlite> update Datos set Nombre="Marcelo" where Nombre="Edwin";
sqlite> select * from Datos;
1       Aguiar  Marcelo Si, tiene       Ninguna que destacar
2       Esparza Silvina Tambien posee   Esposa
3       Aguiar Esparza  Sabina  recien obtenido hja de siete meses
```

Fijense que en el registro 3 hay una falta de ortografía y además no nos agrado el cambio de nombre, lo deberemos hacer en dos pasos, para resumir solo presentaremos el segundo:

```
sqlite> update Datos set Observaciones="Hija, bebe de 7 meses" where
Observaciones="hja de siete meses";
sqlite> select *from Datos;
1       Aguiar  Edwin   Si, tiene       Ninguna que destacar
2       Esparza Silvina Tambien posee   Esposa
3       Aguiar Esparza  Sabina  recien obtenido Hija, bebe de 7 meses
```

También se suele usar la sentencia Replace en estos casos.

8. Eliminar registros

La eliminación de registros se efectúa con la sentencia Delete, la cual al igual que todas hay que usar con algunas precauciones porque no hay vuelta atrás. Una vez eliminado el registro solo se

puede volver a cargar manualmente. Para el ejemplo cargamos un dato que luego eliminaremos:

```
sqlite> insert into Datos (id, Apellido, Nombre, Documentos,Observaciones)values
('4','Perez','Juan','no tiene','ninguna');
sqlite> select * from Datos;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
4|Perez|Juan|no tiene|ninguna
```

Y ahora procederemos a eliminar este registro:

```
sqlite> delete from Datos where id="4";
sqlite> select * from Datos;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
```

El registro 4 (id=4) fue eliminado, aquí usamos el numero de id pero se puede utilizar cualquier indicador como Nombre, Apellido , Documentos u Observaciones.

9.Buscando los datos

Para los usuarios de sql o mysql esta sintaxis es muy conocida, pero aquí trataremos las generalidades para quienes aun no la manejan. La expresión universal de búsqueda es *"select"* y admite muchas opciones que hacen de su uso una sentencia muy poderosa.

La mas simple es "select * from [tabla];" en donde el asterisco es un comodin que indica todas las columnas de esa tabla, pero puede usarse indicando una especifica: "select Apellido from Datos;" y solo listara el total de datos de la columna Apellidos como vemos en este ejemplo:

```
sqlite> select Apellido from Datos;
Aguiar
Esparza
Aguiar Esparza
sqlite>
```

Una forma de búsqueda es ordenar los resultados por algún criterio ("*order by"*), en este ejemplo por el alfabético de la columna Nombre:

```
sqlite> select * from Datos order by Nombre;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
2|Esparza|Silvina|Tambien posee|Esposa
```

Un criterio similar se ejecuta cuando utilizamos "group by" que agrupa resultados en razón de la expresión que usemos para esta sentencia. Un ejemplo: *SELECT * FROM (SELECT * FROM [tabla] GROUP BY [columna]*

Para búsqueda de valores (máximo, mínimo, promedios o contar) la sentencia es la siguiente:

SELECT [exp](tabla o comodin) FROM [Tabla]; por ejemplo

```
sqlite> select max(id) from Datos;
3
sqlite> select COUNT(*) from Datos;
3
```

Como ambos casos solo hay 3 registros numericos (id) el resultado es similar, pero en uno cuenta y en otro halla el maximo, veamos otras expresiones:

```
sqlite> select sum(id) from Datos;
6
sqlite> select min(id) from Datos;
1
```

En *sum(id)* suma la totalidad de los valores id y en *min(id)* el valor mínimo de id. Hay otras formas de selección de resultados e inclusive subquerys (consultas dentro de consultas). Y por supuesto una forma de calcular el "average" o promedio de la suma de una columna:

```
sqlite> select avg(id) from Datos;
2.0
```

Si tenemos un total sumatorio de 6 dividido los 3 registros el resultado obvio es 2. finalmente hay combinaciones:

```
sqlite> select sum(id)/avg(id) from Datos;
3.0
sqlite> select sum(id)+avg(id)/min(id) from Datos;
8.0
```

10.Mas consultas

Establecer el numero de respuesta acotados por LIMIT:

```
sqlite> select * from Datos limit 2;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
2|Esparza|Silvina|Tambien posee|Esposa
sqlite> select * from Datos limit 40;
1|Aguiar|Edwin|Si, tiene|Ninguna que destacar
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
```

el "limit" es mejor utilizado cuando hay una condición where, por ejemplo "select [columna] from [tabla] where [columa] = [condición] LIMIT x; donde x es la cantidad de resultados que esperamos tener en pantalla.

Se pueden ordenar las formas de presentación en pantalla por un criterio de orden descendente o ascendente usando "select [columna o comodín] from [tabla] ORDER BY [columna] DESC;" o ASC para organizar la salida según la columna que deseemos.

```
sqlite> select Nombre, Apellido from Datos order by Apellido desc;
```

```
Silvina|Esparza
Sabina|Aguiar Esparza
Edwin|Aguiar
sqlite> select Nombre, Apellido from Datos order by Apellido asc;
Edwin|Aguiar
Sabina|Aguiar Esparza
Silvina|Esparza
```

consultas por aproximaciones de cadenas de texto: like

Sin duda una interesante posibilidad es la de obtener resultados por aproximación. Supongamos que lo único que tenemos es un apellido o parte de el (lo mismo se aplica a cualquier cadena de textos)

```
sqlite> select Nombre, Apellido from datos where Apellido like 'A%';
Edwin|Aguiar
Sabina|Aguiar Esparza
```

El resultado es la búsqueda de cualquier cadena de texto que inicie con 'A' y siga con otras letras dentro de la columna Apellido. Probemos con otra variante:

```
sqlite> select Nombre, Apellido from Datos where Apellido like '%za%';
Silvina|Esparza
Sabina|Aguiar Esparza
```

aquí buscamos una cadena de texto dentro de Apellido que tenga 'za' en el medio o en cualquier lugar. El comodín '%' indica la posibilidad de mas letras anteriores o posteriores según se coloque. Like se aplica a cualquier columna incluyendo numéricas e interacciona con las demás según la compleja y creciente construcción de sentencias que nos animemos a crear.

Si nos interesa hallar un rango de cadenas de texto dentro de una columna en particular podemos usar IN de la siguiente forma:

```
sqlite>      select      *      from      Datos      where      Nombre      in
('Silvina','Carlos','Sabina','Eduardo');
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
```

Todo lo que cumpla la condicioné IN dentro de los paréntesis es colocado a la salida de pantalla, lo que no es exonerado y los condicionantes que no existen (Carlos y Eduardo) tampoco. También podemos hacer a la inversa mediante operadores como != (distinto de):

```
sqlite> select * from Datos where Nombre !='Edwin';
2|Esparza|Silvina|Tambien posee|Esposa
3|Aguiar Esparza|Sabina|recien obtenido|Hija, bebe de 7 meses
```

De igual manera funcionan “=” y con números > < <= >= etc.

11.¿Y si deseamos consultar mas de una tabla al mismo tiempo?

No seria inusual que necesitaremos cotejar datos entre dos tablas. La sintaxis es simple:

SELECT [columna1],[columna2],[columna_etc(*)] FROM [Tabla1], [TABLA2] WHERE [condicion] [columna tabla1]=[columna tabla1]

(*) De la misma tabla 1 por supuesto y de la tabla 2 que deseamos aparezcan en la salida de pantalla. ¿Parece complicado? Veamos un ejemplo concreto. Tenemos una tabla llamada Consultas y otra Consultor; cada una de ellas posee columnas que se denominan id, nombre y numero_consultor, respuesta. La ecuación seria:

```
select id, nombre FROM Consulta, Consultor where id=numero_consultor;
```

No agradezcan, hagan la prueba.
Hasta aquí llegamos con este primer acercamiento de SQLITE, la sintaxis es la de SQL por lo que salvo algunas escasas diferencias (como el punto antes de los comandos específicos de este motor de base de datos) pudiendo consultar indistintamente un manual de sql o sqlite para el caso. Este tutorial no se agota en lo expuesto, aun queda mucho mas por detallar, pero lo haremos en otra oportunidad.

Cualquier sugerencia, critica o comentario a nuestro mail "elobservadordelsur@gmail.com"
Edwin Aguiar
Diciembre de 2006