

An Overview of SQLite

Albert Danial
lilax meeting
October 9, 2004

Overview of SQLite in Three Parts

- Introduction to SQL
- SQLite
 - Features
 - Comparison to traditional SQL engines
 - Installation
- Demos
 - Unix command line & shell scripting
 - C interface
 - Perl interface
 - SQL with a real database

Why SQL?

- SQL came into being to solve problems in business applications.
- Wade through large collections of data and extract useful information.
- SQL is a complicated solution
- If you can get away with it, a flat file + tools like grep & wc go a long way.
- If you need to write code (Perl, C, Python) to extract useful information, SQL may help.
- SQLite minimizes the start-up headaches involved with an SQL solution.

SQL = Structured Query Language

- A computer language for creating and extracting data from a “database” in a relational manner
- SQL is interpreted by SQL engines or programs.
Examples: Oracle, Sybase, MySQL, Postgres, SQLite
- Initial concept by E.F. Codd, IBM, 1970
- ANSI standard since 1986
- Something you can make a living from doing exclusively (DBA = Database Administrator = SQL jockey)
- My opinion: An incomplete computer language (still need Perl, C, Python, tcl, etc. to drive SQL)
- While incomplete as a language, SQL can be incredibly complex; take years to master.

SQL Basic Concepts

- Information is stored in tables.
- You define the number of columns in a table and the kind of data that each column can hold.
- SQL terminology: a column is called a field.
- A database has multiple related tables.
- SQL commands to
 - Insert rows into tables
 - Extract rows from tables
 - Create/delete entire tables
 - Manage user access & permissions

SQL Basic Concepts, continued

- You have to decide how many tables to create, and which fields to put in the tables.
- A given collection of tables and their fields is known as a database 'schema'.
- Much thought needs to go into the design of the database schema. Goals:
 - A piece of information only appears once.
 - Queries (data extraction) are easy.
 - Queries are fast.

SQL Schema Example

- You want to make a database of your family's recipes.
- Be able to perform searches that answer questions like:
 - Which dessert dishes require four eggs?
 - Is there an Italian appetizer that does not need salt?
 - What should I write on my shopping list to make French bread, a cheesecake, and chocolate chip cookies?

The Cookbook Schema

- A recipe has these attributes:
 - Has a name.
 - Belongs to one or more categories (Thai, dessert, sea food, etc.).
 - Has one or more ingredients.
 - Has preparation instructions, yield, preparation duration, nutritional information, source, rating, et cetera.
- Can store all the information in a flat file, but will need to write a program to extract useful data from the flat file.
- SQL can definitely help.

A naïve Schema

Table RECIPE

name, cat1, cat2, cat3, yield, ingr1, ingr2, ingr3, ingr4, .., ingr20, prep

A single table with 26 fields, each of which is a string.

Identical to a comma separated value data loaded into a spreadsheet.

Limitations:

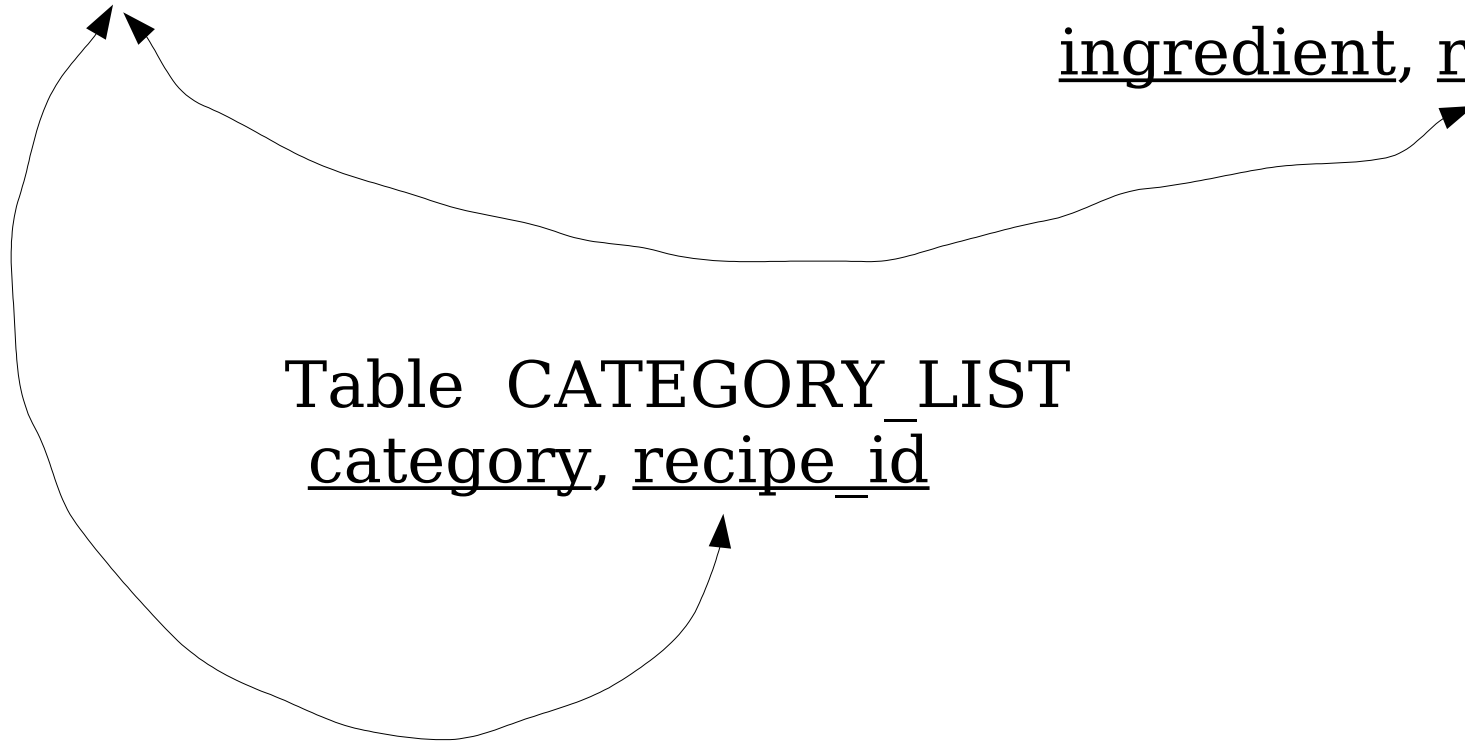
- hardcoded maximum number of categories (3 here)
- hardcoded maximum number of ingredients (20 here)

A Better Schema

Table RECIPE
id, name, yield, prep

Table INGREDIENT_LIST
ingredient, recipe_id

Table CATEGORY_LIST
category, recipe_id



Sample Data

Table RECIPE

id, name, yield, prep

```
27, "burgers" , 4, "Heat grill..."  
28, "apple pie", 6, "Peel apples..."
```

Table CATEGORY_LIST

category, recipe_id

```
"American", 27  
"barbeque", 27  
"picnic", 27  
"dessert", 28
```

Table INGREDIENT_LIST

ingredient, recipe_id

```
"Ground beef 1 lb", 27  
"burger buns 4" , 27  
"ketchup" , 27  
"mustard" , 27  
"apples 3 lb" , 28  
"flour 1 cup" , 28  
"sugar 1 cup" , 28
```

The “Better” Schema also Deficient

We are violating data duplication in two places:

The same category can appear many times in the `category_list` table.

The same ingredients can appear many times in the `ingredient_list` table

Joins

- The fields `recipe.id`,
`ingredient_list.recipe_id`
`category_list.recipe_id`
are keys which link the tables together.
- An SQL query that establishes a relationship between tables by equating their keys is called a join.
- Example:

```
select name,ingredient from recipe, ingredient_list
where recipe.id = ingredient_list.recipe_id;
```

Introducing SQLite

- A command line tool which implements an SQL engine + a C library you can link your code to.
- The engine is easy to use.
- The engine is fast.
- Implements most of ANSI92 standard.
- Stand-alone (or embedded), not client/server.
- Entire database stored in a single file.
- Public domain license.

Comparison to MySQL/Postgres/Oracle/...

• Client/Server:

- Need a database daemon.
- Considerable effort to install, set up.
- Security issue w/open ports.
- Database client can be on remote computer.
- Have to “dump” database to a file to relocate it, back it up.
- SQL app needs daemon running somewhere (?)
- Easily handle simultaneous users.
- Performance at best equals SQLite; mostly slower.

• SQLite

- No daemon.
- Easy to install; trivial to create database.
- No open ports.
- Database is one file.
- Easy to write stand-alone SQL app.
- Can force entire database to reside in memory (no db file!)
- Cannot run in client/server mode (?)
- Works best for single user.
- Small code base; great for embedded processors.
- “Manifest typing” -- can insert any datatype in any field but will store in native form where it can.

When to use it?

- Data storage, information extraction too cumbersome for a flat file.
- Want full power of SQL.
- SQL performance is important.
- Type checking not important.
- Only need to support one (writing) user at a time.
- Don't need to run the database app on a remote computer.
- For me: SQLite fulfills >95% of my database needs.

SQLite & Agile Programming

Principles behind the Agile Manifesto

<http://www.agilemanifesto.org/principles.html>

"Simplicity—the art of maximizing the amount of work not done—is essential."

Let's start: Installation

Use v3.0.7 (latest as of Oct. 9, 2004) or newer when available.

```
# useful: apt-get install libreadline4-dev
cd /tmp
wget http://www.sqlite.org/sqlite-3.0.7.tar.gz
tar xzfv sqlite-3.0.7.tar.gz
mkdir build
cd build
../sqlite/configure --prefix=/usr/local/sqlite-3.0.7
make
make test # fails on my Debian system, no big deal
make install
ln -s /usr/local/sqlite-3.0.7/bin/sqlite3 /usr/local/bin/sqlite
```

Perl Programmer? Install DBD::SQLite

```
cd /tmp
wget http://search.cpan.org/CPAN/authors/id/M/MS/MSERGEANT/DBD-SQLite-1.06.tar.gz
tar zxfv DBD-SQLite-1.06.tar.gz
cd DBD-SQLite-1.06
perl Makefile.PL      # requires DBI
make
make test
make install
```

Create 1st SQLite Database

```
sqlite -version          # should show 3.0.7
```

```
echo "create table t(a,b);" | sqlite simple.db  
echo "insert into t values ('October', 9);" | sqlite simple.db  
echo "insert into t values ('November', 13);" | sqlite simple.db
```

```
sqlite simple.db "insert into t values ('December', 11);"
```

```
sqlite simple.db '.tables'  
sqlite simple.db '.schema'  
sqlite simple.db '.dump'
```

```
sqlite simple.db 'select * from t where b > 10;'
```

```
sqlite simple.db "insert into t values (3.14159265, 'pi');" # ??
```

~/.sqliterc

Get a nicer view of the output with these settings in your ~/.sqliterc:

```
.header on  
.mode column
```

Add a lot of data to 1st Database

```
perl -e 'foreach (1..1000) { printf "insert into t values\n(\"string_%04d\", %4d);\n", $_, $_; }' | sqlite simple.db
```

I thought SQLite was supposed to be fast??!

Performance Tip 1: Transactions

Grouping inserts together in blocks of thousands of operations at a time increases insert performance by 100x

```
rm simple.db
echo "create table t(a,b);" > inserts.sql
echo "begin transaction;" >> inserts.sql
perl -e 'foreach (1..1000) { printf "insert into t values(\"string_%04d\", %
4d);\n", $_, $_; }' >> inserts.sql
echo "commit;" >> inserts.sql
```

```
time cat inserts.sql | sqlite simple.db
```

Let's Create a Cookbook Database

- Many on-line resources for recipes.
- Need a format that is easy to parse.
- <http://home.earthlink.net/~darkstar105/>
- “Menu-Master” format is a flat file, inconsistent, sloppy, but usable.

Script to download MM recipes

```
#!/bin/sh
```

```
site=http://home.earthlink.net/~darkstar105/
```

```
for R in alcohol app bagels bean bev bev2 breads breads2 breakf \
    bredmakr brownies cajun cakes candies casserol cheese \
    chescake chili chinese choco chutrel condment cookies \
    crockpot des diabetic dips drinks fish fruits game german \
    gravy grbeef greek hot indian info italian jellies kidsrec \
    lambveal lasagne lowcal marinad meats mexican mixes \
    muffins newapp newbev newbred newcake newcand newcooky \
    newdes newegch newmain newmeats newpies newpoult newpudd \
    newrel newsalad newsauc newseaf newside newsoups newvegs \
    noodles nuts pancakes pasta2 pickled pies pizza potato \
    pouldish poultry puddings rice roasts rolls salads sandwich \
    sauces sausage savpie seafood shelfish sidedish snacks \
    souffles soups soups2 spices squid steak stews stirfry \
    stock stufdres thai tofu torte veg vegetar
```

```
do
```

```
    wget $site/$R.zip
```

```
    unzip $R.zip          # extracts to .TXT file
```

```
done
```

```
# strip control-M's from end of lines:
```

```
perl -pi -e 's/\cM$//' *.TXT
```

Perl Program to convert *.TXT to .db

```
txt_to_db -r *.TXT # creates file rec.storable  
txt_to_db -w rec.db # reads data from rec.storable, creates rec.db
```

Fun with rec.db

```
sqlite rec.db '.tables'  
sqlite rec.db '.schema'
```

```
sqlite rec.db 'select count (*) from recipe;'
```

```
# which recipes have my favorite ingredient?
```

```
sqlite rec.db # enter interactive mode
```

```
sqlite> select title from recipe, ingr_list  
        where ingr_list.recipe_id = recipe.id and  
        ingr_list.food = "beer" ;
```

I thought SQLite was supposed to be fast ??!

Performance Tip 2: Indices

```
sqlite> create index recipe_idx on recipe ( title );  
create index cat_idx1 on cat_list ( cat );  
create index cat_idx2 on cat_list ( recipe_id );  
create index ingr_idx1 on ingr_list ( food );  
create index ingr_idx2 on ingr_list ( recipe_id );
```

Database size increases; joins with these fields much faster; try the query again.

Performance Tip 3: Subqueries

Create complicated queries by building them up one piece at a time.

Recipe ID's for ingredient of 'beer':

```
select recipe_id from ingredient_list where food = "beer";
```

Title of recipe for a given recipe ID's:

```
select title from recipe where id in (10404, 10805);
```

Combine the two:

```
select title from recipe where id in (  
    select recipe_id from ingredient_list where food = "beer");
```

rec.db ingredients, categories are poorly defined

Use % as regular expression “match anything”
metacharacter along with LIKE operator:

```
sqlite> select count(*) from ingr_list where food='beer';  
count(*)  
-----  
50
```

```
sqlite> select count(*) from ingr_list where food like '%beer%';  
count(*)  
-----  
117
```

I also like chocolate

hmmmmmmmmmm....

```
sqlite> select title from recipe where id in (  
    select recipe_id from ingr_list where  
        food like "%beer%" and recipe_id in (  
            select recipe_id from ingr_list where  
                food like "%choco%"));
```

Could also write using joins instead of subqueries.

Closing Remarks

- My opinion: client/server database engines are overkill for many applications.
- SQLite is fast, simple, powerful.
- Growing adoption:
 - PHP v5
 - OpenOffice ? Register and vote for it!
 - My open source package (announce when?)