In memory and on-disk databases for SQLite

Ben Klemens

16 January 2013

This is mostly a housekeeping entry on Apophenia, but I'll start with a useful trick for speeding up dealings with SQLite. Next time, some statistical epistemology.

SQLite[1] is exactly as much as you need to produce and store data tables and deal with them using SQL. It stores the database in a single file on disk or in memory. Naturally, the in-memory database will be faster, though as soon as your program exits, your database is wiped.

Here's how I use SQLite, where possible: I open a database in memory, and use that as the primary database. To get to the data on my on-disk database, I use SQL's `attach` command. So I'm opening the base database using the C-side databse opening command, and then attaching an auxiliary database on the SQL side. This is why Apophenia has only one global database handle, and has never seriously needed more.

There may be scale issues, but they're easily manageable, because we generally have a reasonable handle on how big any given table is going to be, and we can create scratch tables guaranteed to be much less than a gigabyte in memory, and potentially larger tables with a name like `diskdb.bigtable`, where `diskdb` is the attached on-disk database.

If you want more exposition, I detail this sort of thing in both *Modeling with Data* and *21st Century C* (one of the few overlaps of the two books).

Some functions for merging If you need a (not-too-large) table from the disk, you can copy it to the database without much effort; if you have a table in memory that needs to be saved for posterity, you can copy it to the disk. E.g.

```
create table diskdb.results as
select * from results
```

I pictured the workflow wherein the user opens an in-memory database, copies up the entire on-disk database, and then gets to work, as being much more useful than it turned out to be. So I wrote a few functions to make it happen, and put them into Apophenia in 2005 (whoa—eight years ago). They just sat around, without serious testing or use, to the best of my knowledge.

So I trimmed them from Apophenia. If you still find them to be useful, here they are. The documentation is in Doxygen format. Apophenia has its own means of implementing C-standard variable argument lists, so I removed that part, leaving fixed-argument functions.

[1] `http://sqlite.org/`

```
/** Merge a single table from a database on the hard drive with the database cur

\param db_file The name of a file on disk.
\param tabname The name of the table in that database to be merged in.
\param inout  Do we copy data in to the currently-open main db [\c 'i'] or out t

If the table exists in the new database but not in the currently open one, then
is simply copied over. If there is a table with the same name in the currently o
database, then the data from the new table is inserted into the main database's
with the same name. [The function just calls <tt>insert into main.tab select * f
merge_me.tab</tt>.]
*/
void apop_db_merge_table(char *db_file, char *tabname, char inout){
    Apop_assert_n(tabname, "I need a non-NULL tabname");
    char maine[] = "main";
    char merge_me[] = "merge_me";
    char *from = inout == 'i' ? merge_me : maine;
    char *to = inout == 'i' ? maine : merge_me ;
if (db_file !=NULL)
apop_query("attach database \"%s\" as merge_me;", db_file);
int d = apop_query_to_float("select count(*) from %s.sqlite_master where name ==
if (!d){ //just import table
Apop_notify(2, "adding in %s", tabname);
apop_query("create table %s.%s as select * from %s.%s;", to, tabname, from, tabn
}
else { //merge tables.
Apop_notify(2, "merging in %s", tabname);
apop_query("insert into %s.%s select * from %s.%s;", to, tabname, from, tabname)
}
if (db_file !=NULL)
apop_query("detach database merge_me;");
}

/** Merge a database on the hard drive with the database currently open.

\param db_file The name of a file on disk.
\param inout  Do we copy data in to the currently-open main db [\c 'i'] or out t

If a table exists in the new database but not in the currently open one, then it
simply copied over. If there are  tables with the same name in both databases, t
the data from the new table is inserted into the main database's table with the
name. [The function just calls <tt>insert into main.tab select * from merge_me.t

\li This is SQLite-only; I'm not sure if it really makes much sense for mySQL.
*/
void apop_db_merge(char *db_file, char inout){
```

```
    apop_data *tab_list;
apop_query("attach database \"%s\" as filedb;", db_file);
tab_list= apop_query_to_text("select name from %s.sqlite_master where type==\"ta
              , inout == 'i' ? "filedb" : "main" );
    if(!tab_list) return; //No tables to merge.
for(int i=0; i< tab_list->textsize[0]; i++)
apop_db_merge_table(db_file, (tab_list->text)[i][0], inout);
apop_query("detach database filedb;");
apop_data_free(tab_list);
}
```

For bonus points, here is a command-line program making use of the above.

```
/** \file cmd_apop_merge_dbs.c A little command-line utility to merge
 two databases. Try <tt>apop_merge_dbs -h</tt> for help.*/

/* Copyright (c) 2005--2007, 2009 by Ben Klemens.  Licensed under the modified G

#include <apop.h>
#include <unistd.h>

int main(int argc, char **argv){
    int  merge_ct = 0;
    char c, msg[1000], **merges = NULL;
sprintf(msg, "%s [-v] [-t table_name] [-t next_tabname] main_db.db db_to_merge_i
     "   -t\ttable to merge. If none, do all in the source db. Use as many as yo
     "   -v\tverbose\n", argv[0]);
if(argc<3){
printf("%s", msg);
return 1;
}
while ((c = getopt (argc, argv, "vht:")) != -1){
switch (c){
  case 'v':
apop_opts.verbose ++;
break;
  case 'h':
printf("%s", msg);
return 1;
          case 't':
            merges = realloc(merges, sizeof(char*)*merge_ct++);
            merges[merge_ct-1] = optarg;
}
}
apop_db_open(argv[optind]);
    if (merge_ct)
```

```
        for (int i=0; i< merge_ct; i++)
            apop_db_merge_table(argv[optind +1], merges[i]);
    else
        apop_db_merge(argv[optind +1]);
apop_db_close('n');
}
```