Tip 77: Attach databases for greater speed

Ben Klemens

4 March 2012

level: intermediate database user
purpose: cross data sets, don't touch the disk

Apophenia only allows you to have one database handle at a time.

But, you protest, *I have information in several databases that need linking up*. This is not a problem.

Databases are a little like POSIX filesystems. When you plug in a new hard drive, you use the `mount` command to attach the filesystem to some point in the main system.

SQL uses dots instead of slashes, but the idea is basically the same: if you have a database table named `dtab` and it has a column named `c`, then you can always refer to it with the path `dtab.c`. If it is on disk in `diskdb` then you can `attach database 'diskdb' as ddb` and refer to that column with the path `ddb.dtab.c`.

So attached databases behave just like the base database, except that you need to give a longer path. The original database gets a name of `main`, by the way, which might help to resolve occasional ambiguity.

What makes this interesting for SQLite is that you can have an in-memory database by opening one named `":memory:"`. Apophenia will let you send a `NULL` name of `apop_db_open` with the same effect.

Reading/writing memory really is faster than reading/writing a file on disk, so if your database work is running a bit too slow, moving to an in-memory database should be high on your list of fixes.

The procedure is to load the data table(s) into memory, do work, and write the output back to the database:

```
    //open in-memory database; attach disk db
apop_db_open(NULL);
apop_query("attach database 'ondisk.db' as diskdb");

    //read the data table to the in-memory database
apop_query("create table mdata as select * from diskdb.datatab");

[do math here on mdata, all in memory]

    //write final results to disk
apop_query("create table diskdb.output_table as select ...");
```

Lately I've been using this form by default, because the extra line to attach the database is not all that onerous, and I'm more aware of exactly when I'm modifying the on-disk database.

You'll have to think hard about how you handle the many-gigabyte datasets that don't easily fit into your machine's memory, but you knew that already. Maybe write a `for` loop that pulls chunks of the table via `select ... limit ... offset ...`?