

# Re: Transforming ascii file (pseduo database) into proper database

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2008-01/msg02496.html>

---

- *From:* Tim Chase <[python.list@xxxxxxxxxxxxxxxxxxxxx](mailto:python.list@xxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 21 Jan 2008 15:11:28 -0600
- 

I need to take a series of ascii files and transform the data contained therein so that it can be inserted into an existing database.

[snip]

I need to transform the data from the files before inserting into the database. Now, this would all be relatively simple if not for the following fact: The ascii files are each around 800MB,

[snip]

My questions are:

1. Has anyone done anything like this before, and if so, do you have any advice?

Yes, I regularly do ETL on files from cellular providers to transform hundreds of megs worth (some approach a gig) of data into our internal system.

2. In the abstract, can anyone think of a way of amassing all the related data for a specific identifier from all the individual files without pulling all of the files into memory and without having to repeatedly open, search, and close the files over and over again?

if the file is sorted by something you can use, you can iterate over it and just deal with one grouping at a time. In my case, iterating over gobs of call-detail, the file happens to be sorted by the phone-number on the account. So I iterate over the file maintaining a list of calls for the given phonenumber, and when the phonenumber changes, I deal with the previous cache of data, then re-initialize with the new phone's data.

Other ideas:

- 1) create a temp DB (such as sqlite), skim through the file inserting all your data into a table in this DB, then use DB functionality on it

## Re: Transforming ascii file (pseduo database) into proper database

2) in a light-weight way, assuming there's lots of data per row, and that you have multiple rows associated with a given ID (in my case, such as a phonenumber), you can create a dictionary of an ID to a list of file-offsets in which that ID is used. You can then skim through the file once gathering all the offsets with calls to tell() and then when you want to process an item, you can seek to that particular offset and read in the line. Not greatly efficient, but hackable.

But mostly, it helps if you have a sorted field that's useful to you :)

-tkc

.