

Don't Resort to Re-Sort

Elizabeth Axelrod, Abt Associates Inc., Cambridge, MA

ABSTRACT

You have a huge dataset, happily sorted by one or more keys. But you need to modify your key fields on a relatively small number of records, which will result in an un-sorted file. This paper presents a quick way to re-sort your file ... without re-sorting your file!

THE PROBLEM

You have a huge dataset – lots of records and lots of variables – and it is happily sorted by one or more keys. But you discover that some of these keys must be modified. If you modify or update these keys, your file will no longer be sorted. And you *really* don't want to sort this whole file – again!

The real data I was using contained eligibility information on millions of Medicare beneficiaries, and the IDs which uniquely identify an individual are 11 characters long. There are circumstances in which a Medicare beneficiary's ID can change over time, and this requires updating some IDs, using a technique referred to as 'cross-referencing'. But wait – this is not a lesson in Medicare data. So for this paper, I'll use a mere six records with 4-digit IDs to illustrate the technique.

Obs	ID	FNAME	LNAME	AGE
1	0624	CARL	CARLSON	24
2	1729	JEFF	JETHR	10
3	2095	MATT	MATTHEWS	19
4	2350	LIZA	LUNE	26
5	2603	SUSAN	SUSSMAN	45
6	4209	JOHN	JOHNSON	55

Fig. 1

My sample file (Fig. 1) is sorted by the ID field. But I learn that the IDs for Jeff and Susan need to be updated, as shown in Figure 2.

Obs	ID	NEWID	FNAME	LNAME	AGE
1	0624		CARL	CARLSON	24
2	1729	9271	JEFF	JETHRO	10
3	2095		MATT	MATTHEWS	19
4	2350		LIZA	LUNE	26
5	2603	2306	SUSAN	SUSSMAN	45
6	4209		JOHN	JOHNSON	55

Fig. 2

Updating the IDs for these two records will result in a file which is no longer sorted by ID. I could just re-sort the entire file – but remember that this is really a huge file – millions of records and hundreds of variables – so I'd like to avoid sorting the whole thing.

THE SOLUTION - STEP 1

So instead, I'll begin by reading my incoming file, modifying the necessary IDs, and outputting two intermediate files: a *big* file – which contains all the records whose IDs do not change, and a *little* file containing just those few records whose IDs *do* change.

There are several ways to achieve this, and the best way, of course, will depend on your specific situation. For example, your file might already contain the new values for ID (as shown in Fig. 2). If this is the case, your code to create the big and little files might look like this:

```
data big      /* No change to IDs */
  little     /* Modified IDs      */
;
set in.whatever;
if NEWID > ' ' then do;
  ID = NEWID; /* change ID*/
  output little;
end;
else          /* No change to ID*/
```

```

        output big;
    drop NEWID;
run;

```

Alternatively, you might have a transaction or crosswalk file that contains both the original and the new values for ID, in which case your code might look like this:

```

data big          /* No change to IDs */
  little          /* Modified IDs */
;
merge in1.whatever (in=inmain)
      in2.xwalk    (in=inxwalk
                    keep=ID NEWID);
by ID;
if inmain;
if inxwalk then do;
    ID = NEWID;
    output little;
end;
else
    output big;
drop NEWID;
run;

```

If your big file is *really* big, then you might need to write this intermediate file out to a permanent file (you can always delete it later). Whatever method you use to update your IDs and create your big and little files, you now have two files, and you're ready for the next step.

STEP 2

Sort the little file by ID. Yeah, okay, I lied. There will be some sorting. But just on your little file (which contains only the records with modified IDs). You do not need to re-sort the *big* file, because that file is already in sort order.

```

proc sort
  data=little out=sorted;
  by ID;
run;

```

Remember that the ID var now holds the *modified* value of the original ID.

STEP 3

For the last step, interleave the re-sorted little file and the already-sorted big file into a final output dataset. The `SET` statement with a `BY` var will do this nicely:

```

data out.whatever;
  set BIG
      SORTED;
  by ID;
run;

```

The resulting file will now be in sort order by the updated ID, as shown in Figure 3.

Obs	ID	FNAME	LNAME	AGE
1	0624	CARL	CARLSON	24
2	2095	MATT	MATTHEWS	19
3	2306	SUSAN	SUSSMAN	45
4	2350	LIZA	LUNE	26
5	4209	JOHN	JOHNSON	55
6	9271	JEFF	JETHRO	10

Fig. 3

The code shown for this last step is just the bare bones of code you'd want to use. Tampering with IDs is pretty heady stuff, and you should be familiar with your data. If your original file contained exactly one record per ID, you're probably hoping that your resulting file does, as well. You might want to add a bit of code to verify that you did not introduce any duplicate IDs. The following hunk of code added into the data step will help you to find any offending duplicates:

```
data out.whatever;
  set BIG
    SORTED;
  by ID;

  /* check for duplicates */
  if not(first.ID & last.ID)
  then do;
    /* boo-boo! */
    /* do something! */
  end;
run;
```

CONCLUSION

It is possible to avoid re-sorting a large file when the unsorted records can be identified and separated out. By sorting just this smaller set of records, and interleaving them back with the remaining pre-sorted records, the user can create a newly-sorted file.

CONTACT INFORMATION

Elizabeth Axelrod
Abt Associates Inc.
55 Wheeler Street
Cambridge, MA 02138
(617) 349-2458
elizabeth_axelrod@abtassoc.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.