SESUG Paper 281-2018

How To Dupe A Dedup

Paul M. Dorfman, Independent Consultant

ABSTRACT

Many ETL transformation phases begin with cleansing the extracted data of duplicate data. For example, transactions with the same key but different dates may be deemed duplicate, and the ETL needs to select the latest transaction. Usually, this is done by sorting the extract by the key and date and then choosing the most recent record. However, this technique is quite resource-costly, particularly when the non-key variables are numerous and/or long and the result needs to be resorted back into the original order. In this paper, we show how the same goal can be achieved via a principally different algorithm based on modifying the extract file by marking the duplicate records for deletion and thus obviating the need to sort the entire extract even once. For large extract files and relatively sparse duplicate records, this approach may result in cutting the processing time by an order of magnitude or more.

INTRODUCTION

The paper presents an algorithm of eliminating duplicate records from a data collection based on data recency with no need to sort and/or rewrite the entire input. When the input file is very large but the number of duplicates to be eliminated, as it often happens, is relatively small, the method offers an enormous performance and time-saving advantages compared to the usual scheme of sorting the entire file, eradicating the duplicates, and, if need be, restoring it to the original order. The method is based on a very simple idea of identifying the duplicates using only the file keys and the "recency" variable and, based on the findings, using the MODIFY statement to mark the corresponding records in the original file for deletion. This way, when it is then read in the ensuing processing, the marked records are skipped and so filtered out automatically. When used for cleansing long and wide input files, the method can literally shave hours of processing required for needless sorting and re-sorting.

DATA MODEL AND COMMON APPROACH

The sample file created in the DATA step below describes the results of two fictitious "tests":

- Each test is identified by variable *Test_ID*.
- It has been done on a number of test participants identified by variable *Part_ID*.
- The results, ranked from 1 to 100, are stored in variable *Results*.
- The dates on which the tests were taken, are identified by variable *Date*.
- Since some participants have retaken the tests, the file contains a few duplicate values of *Part_ID* for each *Test_ID*.
- The rest of the variables in the file are *satellite* variables, merely providing additional information about the participants.

Note that the file is already *intrinsically ordered* by (*Test_ID,Result*), which is intentionally reflected in the value of SORTEDBY= data set option below.

```
data Tests (sortedBy = Test_ID Result) ;
  input Test_ID:$1. Part_ID Date:yymmdd10.
        Result Age Height Weight Smoker:$1. Alcohol Lifestyle:$12. ;
  cards ;
A 2 2018-03-13 98 45 162 57 N 102 Sedentary
A 5 2018-03-17 97 47 174 72 N 111 Semi-active
A 4 2018-03-16 93 42 180 70 N  76 Semi-active
A 9 2018-03-22 82 47 186 78 N  41 Sedentary
```

```
A 8 2018-03-20 75 42 167 69 Y  54 Sedentary
A 1 2018-03-12 70 35 194 79 N 116 Semi-active
A 3 2018-03-15 66 58 171 64 N  27 Couch-potato
A 9 2018-03-23 64 47 186 78 N  41 Sedentary
A 9 2018-03-21 57 47 186 78 N  41 Sedentary
A 6 2018-03-18 55 38 174 68 Y  19 Fitness nut
A 7 2018-03-19 54 38 193 81 Y  16 Semi-active
A 2 2018-03-14 51 45 162 57 N 102 Sedentary
B 5 2018-03-16 99 47 174 72 N 111 Semi-active
B 6 2018-03-18 98 38 174 68 Y  19 Fitness nut
B 3 2018-03-14 96 58 171 64 N  27 Couch-potato
B 1 2018-03-12 86 35 194 79 N 116 Semi-active
B 5 2018-03-18 73 47 174 72 N 111 Semi-active
B 4 2018-03-15 60 42 180 70 N  76 Semi-active
B 2 2018-03-13 57 45 162 57 N 102 Sedentary
B 7 2018-03-19 53 38 193 81 Y  16 Semi-active
;
run ;
```

Printing the file results in the following output:

Obs	Test_ID	Part_ID	Date	Result	Age	Height	Weight	Smoker	Alcohol	Lifestyle
1	**A**	**2**	**2018-03-13**	**98**	**45**	**162**	**57**	**N**	**102**	**Sedentary**
2	A	5	2018-03-17	97	47	174	72	N	111	Semi-active
3	A	4	2018-03-16	93	42	180	70	N	76	Semi-active
4	**A**	**9**	**2018-03-22**	**82**	**47**	**186**	**78**	**N**	**41**	**Sedentary**
5	A	8	2018-03-20	75	42	167	69	Y	54	Sedentary
6	A	1	2018-03-12	70	35	194	79	N	116	Semi-active
7	A	3	2018-03-15	66	58	171	64	N	27	Couch-potato
8	**A**	**9**	**2018-03-23**	**64**	**47**	**186**	**78**	**N**	**41**	**Sedentary**
9	**A**	**9**	**2018-03-21**	**57**	**47**	**186**	**78**	**N**	**41**	**Sedentary**
10	A	6	2018-03-18	55	38	174	68	Y	19	Fitness
11	A	7	2018-03-19	54	38	193	81	Y	16	Semi-active
12	**A**	**2**	**2018-03-14**	**51**	**45**	**162**	**57**	**N**	**102**	**Sedentary**
13	**B**	**5**	**2018-03-16**	**99**	**47**	**174**	**72**	**N**	**111**	**Semi-active**
14	B	6	2018-03-18	98	38	174	68	Y	19	Fitness
15	B	3	2018-03-14	96	58	171	64	N	27	Couch-potato
16	B	1	2018-03-12	86	35	194	79	N	116	Semi-active
17	**B**	**5**	**2018-03-18**	**73**	**47**	**174**	**72**	**N**	**111**	**Semi-active**
18	B	4	2018-03-15	60	42	180	70	N	76	Semi-active
19	B	2	2018-03-13	57	45	162	57	N	102	Sedentary
20	B	7	2018-03-19	53	38	193	81	Y	16	Semi-active

Note that the duplicate records we are going to deal with are shown above in boldface. Also note that within each (*Test_ID,Part_ID*) pair, the "recency" variable *Date* indicates which duplicates are more or less recent (or, conversely, more or less dated).

THE TASK

The intent of using this file in the downstream processing is to make use of its existing order and, for each value of *Test_ID*, extract 3 records with the highest values of *Result*.

One wrinkle on this peachy picture is that some tests have been retaken, and the business is interested only in the results of the *most recent* tests. Thus, the duplicate records related to the *dated* tests must be ignored. It means that the observations #1, #4, #9 for *Test_ID*="A" and #13 for *Test_ID*="B" will have to

go, and the rankings of *Result* will change accordingly. For example, the ranking of *Part_ID*=2 in *Test_ID*="A" will be relegated from the top to the bottom, and so forth.

In other words, before we can proceed with using the file *Tests* in the downstream data processing, we need to eliminate the duplicate records for the key (*Test_ID,Part_ID*), whose test *Date* values are not the most recent.

Thus, *if the duplicates were already eliminated*, to attain our downstream processing goal, we could code:

```
data downstream ;
  do _N_ = 1 by 1 until (last.Test_ID) ;
    set Tests ;
    by Test_ID ;
    if _N_ le 3 then output ;
  end ;
run ;
```

However, since the duplicates are not eliminated yet, we have to deal with them first.

STANDARD APPROACH

Note that in this case we need, not only to keep the records with the most recent *Date* within each (*Test_ID,Part_ID*) group, but also preserve the original record sequence. The *standard* approach of dealing with this kind of issue usually goes along lines of code similar to the following:

```
data seq / view=seq ;
  set Tests ;
  Seq = _N_ ;
run ;
proc sort data=seq out=seq_sorted ;
  by Test_ID Part_ID Date ;
run ;
data Tests_nodup ;
  set seq_sorted ;
  by  Test_ID Part_ID ;
  if last.Part_ID ;
run ;
proc sort data=Tests_nodup out=Tests_orig_seq ;
  by Seq ;
run ;
data downstream (drop=Seq) ;
  do _N_ = 1 by 1 until (last.Test_ID) ;
    set Tests_orig_seq ;
    by Test_ID ;
    if _N_ le 3 then output ;
  end ;
run ;
```

SAVING DUPLICATES

Oftentimes, it is required that the rejected duplicates be saved in a separate file in order to be examined a posteriori. This functionality can be added by adding the name of the file to the DATA statement of the third step above and rewriting it as follows:

```
data Tests_nodup Tests_dupes ;
  set seq_sorted ;
  by  Test_ID Part_ID ;
  if last.Part_ID then output Tests_nodup ;
  else output Tests_dupes ;
run ;
```

Note that by using this standard method of killing duplicates, we have to:

- Sort the entire file - including the satellite variables, which can be quite numerous - and store the sorted result somewhere. It means that we need to have 3 times the size of the input file in the SORTWORK space and to write another full copy to the WORK library. The most burdensome part or this operation from the standpoint of machine resources is the need to drag the satellite variables through the sorting process.
- Read the result and write another WORK library file, with the duplicates eliminated.
- Re-sort the result into the original order.

None of the above means much when dealing with a miniscule file like the sample file *Tests* above. However, for a real-world file with, say, 50 million observations and 50 satellite variables, it is a whole lot of work for the computer to do just for the sake of eliminating a small percentage of duplicates. The question is: Is all that labor really necessary; and is there a better way? Fortunately, the answer is "yes".

DUPLICATE-MARKING APPROACH

The concept of duplicate marking is surprisingly simple:

- Discard the burden of all the satellite variables and sort the input file by (*Test_ID,Part_ID,Date*) as above, *but only keeping the keys and Date and the record identification number RID.* Alternatively, it can be done without sorting by using the SAS hash object - if there is enough memory to keep all the distinct values of (*Test_ID,Part_ID*).
- Locate the duplicate records using the variable *last.Part_ID*. Write out *only the RID* values corresponding to the duplicate records to be eliminated. If the hash object is used, just make it write a file with the respective values of *RID*.
- Use the MODIFY statement with the POINT=*RID* option to mark the corresponding duplicate records in the original file for deletion.
- Execute the needed downstream processing by reading the original file *directly*. The records marked for deletion will be automatically eliminated as the file is being read.

Translating the plan into the SAS language (in this case, using the SORT procedure coupled with the DATA step rather than the hash object):

```
data add_RID / view=add_RID ;
  set Tests (keep=Test_ID Part_ID Date) ;
  RID = _N_ ;
run ;
proc sort data=add_RID out=key_RID ;
  by Test_ID Part_ID Date ;
run ;
data dup_RID (keep=RID) ;
  set key_RID ;
  by Test_ID Part_ID ;
  if not last.Part_ID ;
run ;
proc sort data=dup_RID ;
  by RID ;
run ;
data Tests ;
  set dup_RID ;
  modify Tests point=RID ;
  remove ;
run ;
```

At this point, the observations in the file *Tests* identified as duplicates have been marked for deletion by the MODIFY statement. If you now open the file *Tests* in the SAS viewer, it does not display them (note that the observations 1, 4, 9, 13 are absent), showing only the 16 remaining records:

	Test_ID	Part_ID	Date	Result	Age	Height	Weight	Smoker	Alcohol	Lifestyle
2	A	5	2018-03-17	97	47	174	72	N	111	Semi-active
3	A	4	2018-03-16	93	42	180	70	N	76	Semi-active
5	A	8	2018-03-20	75	42	167	69	Y	54	Sedentary
6	A	1	2018-03-12	70	35	194	79	N	116	Semi-active
7	A	3	2018-03-15	66	58	171	64	N	27	Couch-potato
8	A	9	2018-03-23	64	47	186	78	N	41	Sedentary
10	A	6	2018-03-18	55	38	174	68	Y	19	Fitness
11	A	7	2018-03-19	54	38	193	81	Y	16	Semi-active
12	A	2	2018-03-14	51	45	162	57	N	102	Sedentary
14	B	6	2018-03-18	98	38	174	68	Y	19	Fitness
15	B	3	2018-03-14	96	58	171	64	N	27	Couch-potato
16	B	1	2018-03-12	86	35	194	79	N	116	Semi-active
17	B	5	2018-03-18	73	47	174	72	N	111	Semi-active
18	B	4	2018-03-15	60	42	180	70	N	76	Semi-active
19	B	2	2018-03-13	57	45	162	57	N	102	Sedentary
20	B	7	2018-03-19	53	38	193	81	Y	16	Semi-active

This is because when the viewer reads the file, it ignores the observations marked for deletion. And this is exactly what will happen if you now read the file in any other way. Thus, we can proceed with the required downstream processing by reading the modified original file *Tests* directly:

```
data downstream ;
  do _N_ = 1 by 1 until (last.Test_ID) ;
    set Tests ;
    by Test_ID ;
    if _N_ le 3 then output ;
  end ;
run ;
```

Now, when the file *Tests* is being read, all the duplicate observations marked for deletion are automatically ignored, so we arrive at the output required by the downstream processing:

Test_ID	Part_ID	Date	Result	Age	Height	Weight	Smoker	Alcohol	Lifestyle
A	5	2018-03-17	97	47	174	72	N	111	Semi-active
A	4	2018-03-16	93	42	180	70	N	76	Semi-active
A	8	2018-03-20	75	42	167	69	Y	54	Sedentary
B	6	2018-03-18	98	38	174	68	Y	19	Fitness
B	3	2018-03-14	96	58	171	64	N	27	Couch-potato
B	1	2018-03-12	86	35	194	79	N	116	Semi-active

Note that the SORT step sorting the file *dup_RID* is optional. However, it may be quite beneficial, since the statements with the POINT=*RID* option work best when the observations are accessed by the *RID* values in ascending order because it reduces the need to re-read different out-of-order pages. On the other hand, sorting the typically small file *dup_RID* containing but a single numeric variable does not exact any noticeable toll on the overall performance.

SAVING DUPLICATES

If the rejected duplicate records need to be saved in a separate file, it is easy since the file *Dup_RID* already has their *RID* observation numbers. Naturally, if it has to be done, it must be done *before* the duplicates in *Tests* are marked for deletion, since after that they will be unavailable. Hence, to save the duplicates, we only need to insert this DATA step *before* the step with the MODIFY statement:

```
data Tests_dupes ;
  set dup_RID ;
  set Tests point=RID ;
run ;
```

(The variable RID will be automatically dropped because it is named in the POINT= option.) Saving the duplicates has the added benefit of essentially preserving all the records from the originally unduplicated file *Tests*, since if need be, *Tests_nodup* and *Tests_dupes* can be read in concatenation.

REAL-WORLD DIFFERENCE

The author was asked by one of his clients to streamline a production ETL process used to pre-aggregate massive laboratory data for online reporting. The problem was that the process ran so slowly that it could into be fit into the overnight production time slot, typically taking about 15 hours to execute.

PROBLEM SETTING

The examination of the ETL process revealed the following:

- It extracted data from five different data warehouse systems to be combined thereafter.
- The extract from each stream was already ordered by SQL during extraction as needed for the downstream processing.
- Each stream typically contained a few processing keys up to ~40 bytes in combined length and ~50 satellite variables with the total length of ~400-500 bytes.
- The number of records in the extracts varied from about 50 to 80 million rows.
- Before any downstream processing could be done, each extract had to be unduplicated in the manner similar to described earlier in the paper, i.e. only the most recent test results needed to be kept, while the rest had to be filtered out.
- The number of duplicates in each stream typically were between up to ~10 per cent relative to the total number of records.

As it turned out, the bottleneck in the process lied in the unduplication stages coded according to the standard "sort - kill duplicates - sort back" approach:

- It resulted in about 1.5-2 hours of processing needed to unduplicate every data stream and added up to 8-10 hours of extra running time in total.
- The situation was additionally aggravated by shortages of the sort work utility space, as well as by overcrowding the WORK library by the files holding interim results and the final unduplicated files.

SOLUTION AND RESULT

To address the problem, the unduplication stages were re-coded according to the MODIFY approach described above. As a result, the time needed to execute them was reduced to 3 to 5 minutes apiece, and the entire ETL process was now taking about ~5-7 hours in total to run. The disk space issues, that had been previously plaguing the process, also disappeared.

To recap, the reasons for such seemingly incredible game change were rather simple:

- Instead of sorting the entire extract file with the record ~500 bytes wide, only the key variables, the "recency" variable, and the *RID* variable needed to be sorted.
- The need to temporarily store the interim files equal in size to that of the extract was eliminated.
- It was no longer necessary to store the results of unduplication in separate files. Rather, the downstream processing could now read the extract files (already ordered as needed) *directly* and automatically filter out the duplicate records as marked for deletion.

AUTOMATION

Of course, the alternative approach to unduplication described above can be encoded as an encapsulated routine, such as a macro, whose parameters, for example, may include:

- The name of the data set name to be unduplicated.
- The keys by which to unduplicate.
- The variable, such as *Date* above, indicating the recency of the data.
- Whether to keep the least or most recent duplicate.
- Whether or not save the duplicate records and, if needed, the data set name for their destination.
- Other options one may need or think of.

Needless to say, the author wrote a macro along these lines for the client. While it cannot be shared here due to proprietary reasons, it should not take a competent SAS programmer more than an hour or so to encapsulate the concept once it is understood.

CONCLUSION

If there is a need to unduplicate a long and wide data set based on criteria of data recency, it is not necessary to sort the entire data collection for the sake of eliminating a few duplicate records. Rather, the records to which they belong can be inexpensively identified and then marked for deletion by using the MODIFY statement. Doing so may save gobs of processing time and disk space.

ACKNOWLEDGMENTS

The author would like to thank Gregg Snell and Mark Warner from a global clinical laboratory company for the opportunity to work on a project where the idea presented here materialized as a result of the need, on part of the author, to help resolve real-world data processing efficiency problems.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Paul M. Dorfman
Independent Consultant
(904) 260-6509
sashole@gmail.com